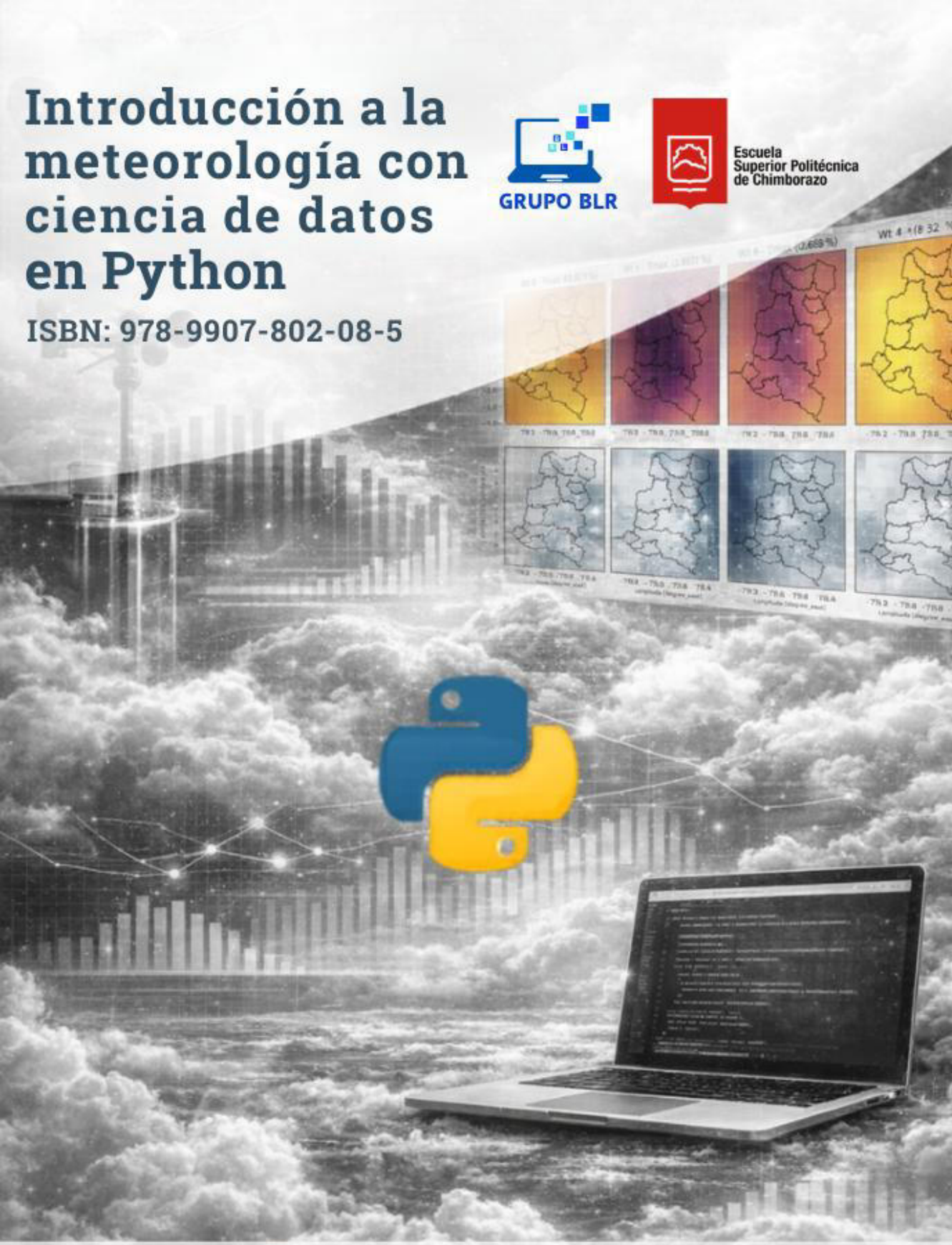
Introducción a la meteorología con ciencia de datos en Python

ISBN: 978-9907-802-08-5

GRUPO BLR

Escuela Superior Politécnica de Chimborazo



Escudero, I., Ramos, C., & Chávez, I.

2026

INTRODUCCIÓN A LA METEOROLOGÍA CON CIENCIA DE DATOS EN PYTHON

AUTORES:

AMALIA ISABEL ESCUDERO VILLA

CRISTINA ESTEFANÍA RAMOS ARAUJO

IVÁN FABRICIO CHÁVEZ VELASCO

Este libro ha sido debidamente examinado y valorado en la modalidad doble par ciego con fin de garantizar la calidad científica.

©Grupo Editorial BLR
Escuela Superior Politécnica de Chimborazo
Riobamba – Ecuador
Correo: publicaciones@grupoblr.com
https://grupoblr.com/libros-investigo
REPOSITORIO

Escudero, A.,Ramos, C.,Chávez, I. (2026) Introducción a la meteorología con ciencia de datos en Python. Grupo Editorial BLR.

ISBN: 978-9907-802-08-5

FILIACIONES DE LOS AUTORES

Amalia Isabel Escudero Villa

Escuela Superior Politécnica de Chimborazo

Correo Electrónico: aescudero@espoch.edu.ec

ORCID: https://orcid.org/0000-0003-4506-932X

Cristina Estefanía Ramos Araujo

Escuela Superior Politécnica de Chimborazo

Correo Electrónico: cristina.ramos@espoch.edu.ec

ORCID: https://orcid.org/0000-0002-8644-5814

Iván Fabricio Chávez Velasco

Docente Investigador

Escuela Superior Politécnica de Chimborazo

Correo Electrónico: fabricio.chavez@espoch.edu.ec

ORCID: https://orcid.org/0000-0003-2650-0145

GRUPO BLR

CONTENIDO

INTRODUCCIÓN

La meteorología es la ciencia que se encarga del estudio del clima y la atmósfera, constituye ejes fundamentales para comprender la dinámica del planeta y anticipar sus efectos sobre la sociedad, los ecosistemas y la economía. En un contexto global donde el cambio climático se ha convertido en una de las principales preocupaciones científicas, estudiar el comportamiento de variables meteorológicas como temperatura ambiente, precipitación, radiación solar, entre otras, es una herramienta para mitigar las consecuencias, a través de la planificación de uso de los recursos naturales.

Con el avance de las tecnologías se han desarrollado instrumentos de medición cada vez más precisos y de monitoreo continuo. Estos registran datos (cada segundo) de variables de interés como temperatura del aire, precipitación, radiación solar, entre otros; generando miles de millones de datos a disposición de los científicos para analizarlos o estudiarlos. Sin embargo, ni la mente más poderosa es capaz de analizar con rapidez grandes cantidades de datos para extraer conocimiento importante. Bajo este sentido, ciencia de datos es un aliado estratégico que ayuda a climatólogos, investigadores y científicos a analizar de manera eficiente información masiva.

El presente libro tiene como propósito integrar el campo de la meteorología y ciencia de datos. Por un lado, se abordan los conceptos esenciales de la atmósfera, las variables climáticas y los sistemas de observación; por otro, se introducen herramientas modernas de análisis de datos, procesamiento y aprendizaje automático utilizando *Python*. La estructura del libro parte

desde conceptualizaciones básicas hasta un enfoque práctico para analizar el clima y la atmósfera; sin explicar a profundidad conceptos complejos computacionales.

Este libro se divide en ocho capítulos, iniciando desde una explicación de las bases de la meteorología y fundamentos epistemológicos, complementando los conceptos de clima y las técnicas modernas de aprendizaje automático. Muestra cómo el Big Data y las técnicas de análisis computacionales modernas abren nuevas posibilidades para estudiar fenómenos atmosféricos. Cada capítulo enfocado en su aplicación como herramienta central para el procesamiento de información climática.

El **capítulo 1: Introducción a la meteorología y ciencia de datos** introduce los conceptos básicos de meteorología y ciencia de datos, explorando una reseña histórica de la evolución en la observación climática desde métodos básicos a utilizar herramientas modernas.

El **Capítulo 2: Variables meteorológicas,** presenta las variables fundamentales que describen el clima: temperatura del aire, presión atmosférica, humedad relativa, precipitación, radiación solar global y difusa, finalmente velocidad y dirección de viento. No solo se describen sus definiciones, sino también los procesos físicos que las generan y cómo se relacionan entre sí. Además, se describe la formación de nubes, qué significa la radiación global y por qué el viento cambia de dirección. Todo esto se conecta con la importancia de medirlas para entender su naturaleza.

El **Capítulo 3. Sistemas de observación, medición y registro en meteorología**, describe cómo se obtienen los datos meteorológicos y el tipo

de herramientas utilizadas para medirlos y registrarlos. Se habla de las escalas de tiempo y espacio, de las unidades de medida y de la precisión de los instrumentos.

El **Capítulo 4. Fuentes y formatos de datos meteorológicos,** describe las principales fuentes de datos disponibles: estaciones meteorológicas locales y redes internacionales como CMIP6, CHIRTS y CHIRPS. Se describe la red ESPOCH-INAMHI. Además, se introduce conceptos de los formatos de datos y estándares comunes para compartir, comparar y analizar la información.

El **Capítulo 5. Introducción a Python,** inicia el camino hacia la programación aplicada a la meteorología. Describe qué es Python, por qué se ha convertido en el lenguaje predominante en ciencia de datos y cómo instalarlo de manera práctica. Se enseña a dar los primeros pasos en el entorno de trabajo Spyder y a entender la lógica básica de la programación. En este punto, el libro invita a perder el miedo a conceptos más avanzados de programación.

En el **Capítulo 6. Python para ciencia de datos**, se introduce las librerías esenciales para trabajar con datos: IPython, NumPy, Pandas y Matplotlib. Cada una se describe con ejemplos sencillos que muestran cómo crear arreglos, manipular bases de datos o graficar resultados. De esta manera el lector aprende como *Python* a realiza el trabajo de forma rápida y eficiente.

En el **Capítulo 7. Introducción al aprendizaje automático (*Machine Learning*),** se describe qué es el aprendizaje automático y cómo se diferencia de la estadística tradicional; algoritmos supervisados y no

supervisados, de manera visual y didáctica. Además, se presentan los algoritmos más utilizados, y cómo usarlos con la librería Scikit-Learn.

Por último, el **Capítulo 8. Análisis de datos climáticos con Python** presenta el formato NetCDF como estándar de almacenamiento de datos climáticos y la librería xarray como la herramienta ideal para manipularlo. Se explica cómo acceder a datos remotos mediante Intake, OPeNDAP y Dask, y cómo procesar bases de datos provenientes de estaciones meteorológicas locales. Finalmente, integra todo lo aprendido en casos prácticos, los cuales forman parte de los resultados del proyecto de investigación "Predecir patrones de comportamiento climático mediante técnicas de aprendizaje automático en la provincia de Chimborazo – IDIPI306" de la Escuela Superior Politécnica de Chimborazo, ejecutado durante el año 2022 hasta el 2025.

CAPÍTULO I

INTRODUCCIÓN A LA METEOROLOGÍA Y CIENCIA DE DATOS

1.1. La meteorología y su importancia

A lo largo de la historia, el estudio del clima y la capacidad de predecir sus cambios han sido fundamentales para el desarrollo de las sociedades. Las comunidades que han logrado comprender y adaptarse a las condiciones climáticas han prosperado, mientras que aquellas que no lo han hecho han enfrentado graves dificultades, e incluso han desaparecido. La meteorología es una de las ciencias más antiguas, ya que nació de la necesidad del ser humano por comprender y adaptarse al comportamiento atmosférico. Esta ciencia estudia los fenómenos atmosféricos, su estructura, composición y procesos para entender cómo se desarrollan los cambios climáticos.

La importancia del clima en la historia de la humanidad propició el surgimiento de la meteorología como una de las ciencias más antiguas, motivada por la necesidad de interpretar el comportamiento de la atmósfera y anticipar sus efectos en la vida cotidiana. Además, existe la ciencia de la climatología, la cual estudia el clima a largo plazo, analizando patrones y variaciones en periodos prolongados. Es importante distinguir los conceptos entre ambas ciencias ya que comparten conceptos, pero el rango temporal de estudio difiere, por lo tanto, la metodología también es distinta.

La meteorología es la ciencia que estudia la estructura, composición, características y procesos importantes que ocurren en una delgada capa circundante de la atmósfera (troposfera, agua y aire), incluyendo el estado

futuro. Su objetivo principal es analizar las condiciones temporales tanto en el presente como en el pasado para predecir eventos climáticos futuros. Básicamente, la meteorología examina los procesos físicos que ocurren en la atmósfera. Esta disciplina se basa en la recopilación de datos de diversas variables meteorológicas, tales como: temperatura del aire, humedad relativa, presión atmosférica, precipitación, entre otras, con fin de comprender y modelar los sistemas. Por otro lado, la climatología es una rama de la meteorología que estudia los estados del tiempo que suceden de manera habitual en un lugar determinado, basada en los datos meteorológicos existentes en la zona de estudio. Una de las características principales es que los patrones climáticos se analizan a largo plazo y a una meso escala.

Según la Unión Internacional de Geodesia y Geofísica (IUGG), la meteorología estudia solo la capa baja de la atmósfera hasta los 20 km de altura. En esta zona se explora las distintas propiedades inherentes al comportamiento de un clima, dando origen a distintos conceptos de importancia. Por un lado, el tiempo atmosférico se define como el estado en que se encuentra la atmósfera en un determinado lugar y momento. Por otro lado, el clima de una zona es el tiempo atmosférico que aparece de manera recurrente a lo largo de un intervalo de tiempo, ya sea anual o mensualmente. La Organización Meteorología Mundial (OMM) en la Conferencia de Varsovia (1935) proporcionó la definición de clima como las condiciones meteorológicas medias para el mes y el año, calculadas sobre un período de 30 años. Es decir, hablar sobre el clima de un lugar corresponde a lo que ocurre normalmente en ese lugar, por ejemplo, inviernos fríos, sequías, etc. Ver figura 1.1.

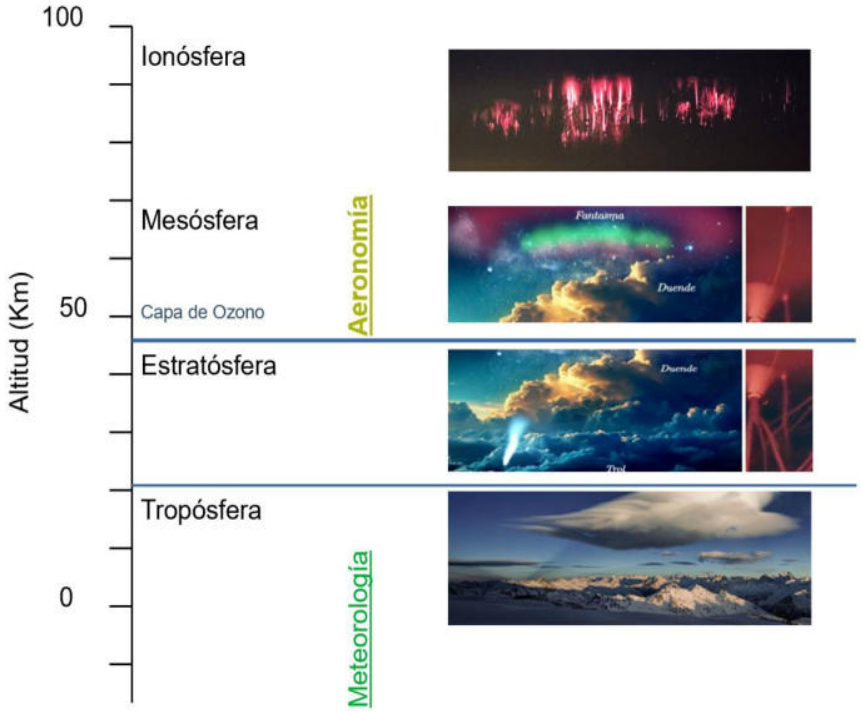


Figura 1.1. Representación de las capas de la atmósfera y las zonas de estudio de la Aeronomía y Meteorología, la cual se centra en la troposfera, donde ocurren la mayoría de los fenómenos atmosféricos y climáticos.

Fuente: elaboración propia.

La meteorología es considerada una ciencia multidisciplinaria porque integra conocimientos y métodos de diversas disciplinas científicas para estudiar los fenómenos atmosféricos y su impacto en el planeta. La atmósfera de la Tierra está compuesta por tres sustancias: gas (atmósfera), líquido (hidrosfera) y sólido (litosfera), cuyos procesos físico-químicos se estudian de manera independiente pero integrados bajo la geofísica.

Además, mantiene un vínculo con la física del Sol, ya que la radiación interviene en varios sistemas dinámicos de la tierra.

Con lo que respecta a la tecnología, la meteorología tiene una estrecha relación con la computación y el análisis de datos. En la meteorología moderna se utiliza herramientas como inteligencia artificial, Big Data, entre otros; para el procesamiento de datos. El propósito principal es identificar modelos estadísticos y realizar pronósticos precisos mediante la resolución de sistemas complejos, a través de computadores de altas prestaciones. De igual manera, la ingeniería es una gran aliada para la toma y registro de datos, puesto que desarrollan sensores meteorológicos, satélites y radares que hacen posible la observación y el análisis del clima en tiempo real.

1.2. Epistemología de la Meteorología

La epistemología es la rama de la filosofía que estudia la naturaleza, el origen y la validez del conocimiento. En el contexto de las ciencias naturales, analiza cómo se generan, justifican y verifican las teorías científicas que explican los fenómenos observables en la naturaleza. La meteorología se fundamenta en principios físicos, matemáticos y computacionales, por lo que está en el grupo de las ciencias empíricas. Sin embargo, debido a la complejidad y el carácter caótico de los sistemas atmosféricos, enfrenta desafíos, como, por ejemplo, la incertidumbre en la predicción, lo que lleva a la utilización de modelos probabilísticos. Esto la convierte en una disciplina que no solo depende de la observación y experimentación, sino también de simulaciones y modelizaciones basadas en principios teóricos.

Desde la antigüedad, el ser humano ha observado los cambios en el clima, asociando los fenómenos atmosféricos con el movimiento de los astros y las estaciones del año. Estas observaciones se basaban en signos visibles, como la apariencia del cielo, el comportamiento de los vientos y la migración de aves. Con el tiempo, estas correlaciones dieron lugar a mitos que intentaban explicar los cambios meteorológicos. En civilizaciones antiguas la elaboración de calendarios y la predicción de eventos naturales surgieron gracias al conocimiento de los ciclos celestes, como los egipcios, los cuales vinculaban la crecida del Nilo con la aparición de Sirio en el cielo, mientras que los babilonios predecían el clima observando el aspecto del firmamento (Valenzuela, 2010).

Uno de los primeros intentos por sistematizar el estudio del clima se encuentra en el libro 'Meteorológica' (del griego *meteoro,* que significa fenómeno celeste y *logos, que en griego* es estudio), escrito por Aristóteles alrededor del 340 a. C., donde combinó observaciones y especulaciones sobre los fenómenos atmosféricos en un intento por dar explicación a la formación de los ríos, las nubes y la neblina, los vientos, el cambio de climas, los truenos y los huracanes (Zen De Figueiredo Neves et al., 2017).

El conocimiento sobre el clima continuó evolucionando durante la época del imperio romano, quienes se dedicaron a recopilar y sistematizar información sobre la naturaleza. Obras como Historia Naturalis de Plinio el Viejo y Tetrabiblos de Ptolomeo se convirtieron en referentes fundamentales para la predicción del tiempo durante la Edad Media. En la misma época, el mundo islámico recopiló y enriqueció el conocimiento meteorológico grecorromano, estableciendo un enfoque basado en

observaciones astronómicas, lo cual acentúo la creencia en que el tiempo podía predecirse exclusivamente a partir del movimiento de los cuerpos celestes. En la edad media, Roger Bacon desafió las ideas tradicionales y defendió la importancia de la observación directa de los fenómenos meteorológicos, lo que llevó a argumentar que la predicción del clima solo es posible con la identificación de reglas exactas.

En el renacimiento existió un cambio sustancial en la meteorología puesto que las ciencias empezaron a desvincularse de la astrología y la especulación filosófica. En este periodo se desarrollaron instrumentos de medición para recopilar datos precisos sobre los fenómenos atmosféricos. Entre los inventos más destacados se encuentran el termómetro de Galileo Galilei, el barómetro de Torricelli, el anemómetro de Robert Hooke y Horace de Saussure construye el higrómetro a cabello para medir la humedad del aire (Fernández-Bobadilla, 1969).

A partir del siglo XVII, los científicos identificaron que la comparación de observaciones meteorológicas realizadas simultáneamente en diferentes regiones del mundo alcanzaría a una mejor comprensión de los patrones atmosféricos. Uno de los primeros intentos de coordinación internacional en la observación del clima ocurrió entre 1649 y 1651 en distintas ciudades de Europa. Posteriormente, el gran duque Fernando II de Toscana promovió la primera red organizada de estaciones meteorológicas en Florencia, Pisa, Bolonia, Milán y Parma, e incluso en lugares más alejados como París, Varsovia e Innsbruck. Durante este periodo, se establecieron procedimientos estandarizados para medir la presión atmosférica, temperatura del aire, humedad, dirección del viento y estado del cielo, y los

registros eran enviados a la academia para ser comparados (Helmis & Nastos, 2012).

A lo largo del siglo XVIII, los estudios sobre la circulación atmosférica global fueron tomando forma. En 1735, George Hadley describió el comportamiento de los vientos alisios, relacionándolos con la rotación terrestre. En el siglo siguiente, el gran avance en la comunicación de datos meteorológicos llegó con la invención del telégrafo en 1843, que abrió la posibilidad de transmitir información climática con rapidez y sin precedentes. Esto facilita la elaboración de mapas del tiempo y mejora significativamente la precisión de los pronósticos meteorológicos. Esta comprensión se completó con el entendimiento de movimientos a gran escala como la fuerza de Coriolis, descrita por Gaspard-Gustave Coriolis en 1835, y la existencia de células de circulación en latitudes intermedias predominantes del oeste descrito por teoría de William Ferrel en 1856 (Spiridonov & Ćurić, 2021).

A comienzos del siglo XX, los avances en el entendimiento de la dinámica atmosférica sentaron las bases para el desarrollo de la predicción moderna del tiempo basada en métodos matemáticos, donde se proponía simplificar las ecuaciones de la dinámica de fluidos, eliminando los términos menos relevantes para facilitar su resolución numérica (Zen De Figueiredo Neves et al., 2017). Bjerknes (1862-1951) fue uno de los fundadores de la meteorología actual, publicó sus estudios sobre el teorema de la circulación, lo que llevo a avances en el entendimiento de la oceanografía dinámica, descubrió la existencia del frente polar, la teoría ondulatoria de los ciclones y la metodología para pronosticar el clima analizando los frentes y las masas

de aire (Fernández-Bobadilla, 1969). No obstante, el volumen de cálculos requerido resultaba abrumador para la época. Bajo este contexto, los avances tecnológicos se presentaban de base a los primeros experimentos de predicción del tiempo mediante cálculo numérico.

En 1960, Edward Lorenz descubrió que el comportamiento caótico de la atmósfera, dando origen a la teoría del caos. Sus hallazgos evidenciaron que pequeñas perturbaciones en las condiciones iniciales son capaces de generar grandes diferencias en la evolución futura del clima, lo que impone un límite natural a la capacidad de predicción atmosférica. Este mismo año aconteció el lanzamiento del primer satélite meteorológico, el TIROS-1, lo que dio paso a una nueva era de observación global del clima y desde entonces los satélites de observación de la Tierra se han convertido en herramientas fundamentales para monitorear fenómenos atmosféricos (Spiridonov & Ćurić, 2021).

En la actualidad, el ser humano hace frente a comportamientos meteorológicos marcados por el cambio climático y el calentamiento global, con la aparición más frecuente de desastres naturales de origen atmosférico, representando una amenaza latente en todas las regiones del planeta (Zen De Figueiredo Neves et al., 2017). La visión contemporánea de la meteorología va más allá de una rama de geofísica o física atmosférica, se ha vuelto una ciencia interdisciplinaria enfocándose al estudio y análisis de los flujos de energía y los procesos planetarios (Campos, 2025). Por esta razón, el sistema climatológico global requiere de la interacción de conocimientos en física, matemática, geología, hidrología, química,

ingeniería, biología, ecología, sensores remotos, meteorología, inteligencia artificial y ciencia de datos.

1.3. La atmósfera

La atmósfera terrestre es una delgada capa de gases que envuelve al planeta y se mantiene unida gracias a la fuerza de gravedad. Como en otros cuerpos celestes, esta capa gaseosa es retenida siempre que la gravedad del planeta sea suficiente y no sea arrastrada por el viento solar (Martin, 2015). Su nombre proviene del griego antiguo atmós (vapor) y sphaîra (esfera), y su función en el planeta Tierra es importante para el mantenimiento de la vida. La altura aproximada de la atmósfera es de hasta 1.000 km y representa la mayor parte de la masa de la Tierra con un valor de $5{,}1 \times 10^{18}$ kg, donde la mayor parte se concentra en las capas más bajas (Landsberg, 1953).

La atmósfera permite el intercambio de energía entre el Sol, la superficie terrestre y entre distintas regiones del planeta, lo que mantiene el equilibrio térmico global y define las condiciones del clima (Martin, 2015).

El origen de la atmósfera en la Tierra es un enigma científico que llama la atención a la humanidad, por lo que a lo largo de la historia surgieron algunas teorías que tratan de dar una explicación científica. Una teoría sugiere que la primera atmósfera terrestre estuvo compuesta por gases ligeros primitivos como helio, hidrógeno, amoníaco y metano, presentes en el medio interestelar, es decir que los gases ligeros que componen el ambiente fueron capturados por la gravedad que se iba generando en la tierra. Otra hipótesis sostiene que luego se generó una segunda atmósfera con origen volcánico y es una de las más aceptadas, donde se plantea que a

medida que se enfrío la tierra se generó una desgasificación volcánica expulsando componentes como nitrógeno (N_2), dióxido de carbono (CO_2), hidrógeno (H_2), monóxido de carbono (CO) y vapor de agua (H_2O) (Landsberg, 1953). Ver figura 1.2.

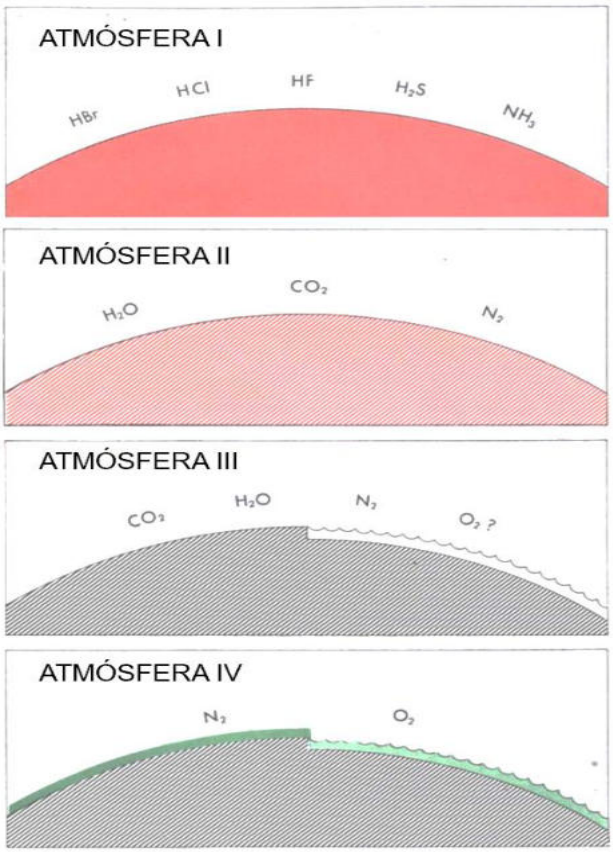


Figura 1.2. Etapas del surgimiento y creación de la atmósfera terrestre.

Fuente: Landsberg, (1953).

Al final de esta segunda atmósfera la tierra era lo suficientemente fría para que el vapor de agua se condensará en océanos, contribuyendo a la

liberación del volumen para que sea ocupado por el oxígeno. Sin embargo, surge una nueva pregunta, ¿de dónde apareció tanto oxígeno? Existen tres hipótesis que podrían ser de gran utilidad a la hora de resolver esta cuestión:

La primera hipótesis plantea que el oxígeno libre fue generado en una etapa temprana y de alta temperatura de la segunda atmósfera mediante la disociación térmica del vapor de agua en hidrógeno y oxígeno. La segunda hipótesis propone que el oxígeno se liberó en una etapa posterior y más fría, a través de la disociación fotoquímica del vapor de agua en las capas superiores de la atmósfera. Por último, la tercera hipótesis sugiere que el oxígeno surgió mediante el proceso de fotosíntesis, en el cual las plantas transforman el dióxido de carbono (CO_2) en oxígeno libre (O_2) (Landsberg, 1953). Esta hipótesis es la más aceptada y convincente, ya que explica la presencia, acumulación y mantenimiento del oxígeno atmosférico, lo que provocó que una gran cantidad de dióxido de carbono terminará transformado en oxígeno o en rocas. Posteriormente, debido al calentamiento del interior del planeta y el proceso de diferenciación planetaria, se liberaron gases más ligeros que ascendieron a las capas externas. Estos gases fueron CO_2, N_2, H_2O y H_2 y se consideran los precursores de la atmósfera actual. Finalmente, la condensación del vapor de agua habría dado origen a los océanos, cerrando así el ciclo de formación del sistema climático terrestre.

De esta manera, se tiene al aire presente en la actualidad, una atmosfera donde el 99 por ciento está constituido por oxígeno y nitrógeno, donde su estructura y composición logran absorber la radiación ultravioleta dañina en la capa de ozono, reducir la amplitud térmica entre el día y la noche y actuar

como escudo frente a meteoritos que se desintegran al ingresar a la atmósfera (Visconti, 2016). Los elementos de los que principalmente se compone la atmósfera son:

Tabla 1.1. Porcentajes de gases permanentes y variables en la atmósfera.

	Gas	Símbolo	% en el aire (por volumen)
PERMANENTES	Nitrógeno	N_2	78.08
	Oxígeno	O_2	20.95
	Argón	Ar	0.93
	Neón	Ne	0.0018
	Helio	He	0.0005
VARIABLES	Agua	H_2O	0–4
	Dióxido de carbono	CO_2	0.0380
	Metano	CH_4	0.00017
	Óxido nitroso	N_2O	0.00003
	Ozono	O_3	0.000004

Fuente: Spiridonov y Ćurić, (2021).

1.4. ¿Qué es ciencia de datos?

Se considera como una ciencia relativamente emergente por la aparición masiva de la información en los últimos años, lo que ha provocado la necesidad de herramientas potentes para su análisis, es decir, el nacimiento de ciencia de datos. De igual manera, al ser una ciencia con poca trayectoria su definición se encuentra en un espectro aún no tan establecido, sin embargo, es posible encontrar algunas definiciones propuestas por expertos, desde su mención por primera vez hasta la actualidad.

1.4.1. Una ciencia en maduración

En 1961 el estadístico Jhon W. Tukey reflexionó sobre su trayectoria en estadística y concluyó que su interés principal no fue únicamente la inferencia estadística (de lo particular a lo general), sino el análisis de datos como un campo más amplio e integrador. Tukey, conocido por el desarrollo de complejos algoritmos y el famoso diagrama de caja y bigotes (Box Plot), resalta que el análisis de datos no se limita a la inferencia muestral, ni a la identificación de patrones ocultos, ni a la asignación de recursos en experimentos; es una disciplina más amplia y diversa. En estas declaraciones se habló por primera vez de la evolución de la estadística aplicada y la estadística matemática como una ciencia aparte; la “ciencia de datos”.

Años después de los aportes de John Tukey, Wu (1997) propuso una taxonomía descriptiva de la ciencia de datos dividida en tres áreas clave. La primera área consiste en la recolección de datos (diseño experimental y encuestas), el segundo se caracteriza por los procesos de modelado, análisis de datos y la tercera es la comprensión de problemas, resolución y toma de decisiones. Wu incluso propuso renombrar este enfoque como "ciencia de datos" o "ciencia estadística”.

En el siglo XXI la ciencia de datos es presentada como un nuevo paradigma de descubrimiento científico, Aunque ha tenido éxitos notables en múltiples áreas, su desarrollo como disciplina científica madura aún está en proceso. Michel Brodie recalca que:

Aunque (ciencia de datos) ha tenido éxitos notables en múltiples áreas, su desarrollo como disciplina científica madura aún está en proceso. Su consolidación tomará al menos una década, al compararse con los siglos de evolución de los paradigmas anteriores: el empírico, el teórico y el paradigma computacional propuesto por Jim Gray conocido como la "Cuarta Paradigma de la Ciencia. […] la mayoría de las aplicaciones actuales de ciencia de datos son específicas de dominio, y pocos métodos han sido generalizados más allá de su contexto original. […] se necesita avanzar en su generalización, validación interdisciplinaria y consolidación académica para alcanzar su madurez como disciplina científica global (Brodie, 2019, p.101).

A pesar de la falta de madurez de ciencia de datos, su desarrollo marcó un hito en la historia de la ciencia y tuvo un impacto inmediato destacando varias aplicaciones y resultados favorables. El mismo autor cita el trabajo de Thomas Piketty, quien utilizó datos observacionales de más de 120 años para identificar, mediante métodos de ciencia de datos, correlaciones entre ingresos laborales y acumulación de riqueza. Este hallazgo fue validado posteriormente con mayor certeza, aunque no fue una conclusión analítica directa, mostrando así sus beneficios.

1.4.2. Definición

La ciencia de datos se define como un nuevo campo interdisciplinario, donde convergen la estadística, las matemáticas, la informática, la computación, la ingeniería y la sociología; para estudiarlos con el fin de transformarlos en decisiones mediante una metodología que transforma los datos en conocimiento, para lo cual emplea métodos científicos, procesos,

algoritmos y sistemas automatizados para analizar, modelar e interpretar datos complejos.

Los datos son información que se caracterizan, describen y ordenan para encontrar patrones, tendencias, relaciones y generar nuevo conocimiento. Esta ciencia, a través del análisis de los datos, busca obtener respuestas óptimas en la toma de decisiones, para lo cual, varios investigadores se han establecido algunas etapas que describen el proceso para generar conocimiento a través de los datos y varían dependiendo del enfoque o campo aplicativo.

En el presente libro se tomó el enfoque presentado en el trabajo realizado por Longbing Cao[1], el cual cuenta con un extenso artículo de revisión bibliográfica del 2017 titulado *"Data science: A comprehensive overview"*, donde escudriñó a profundidad lo que llamaba "el ADN de los datos" y puntualizó que, para comprender este ADN, se requería de ciencia de datos. La importancia de este artículo radica en entender la necesidad del uso de esta nueva ciencia emergente para explorar y entender la información oculta en una cantidad masiva de datos, por esta razón se trató lo explicado por Longbing en los subcapítulos: "Dato-Decisión", "evolución del análisis de datos", "innovación en ciencia de datos: Retos y Oportunidades".

[1] Longbing Cao es un investigador en inteligencia artificial y ciencia de datos, Su interés de investigación es amplio e incluye inteligencia artificial, ciencia de datos, la informática del comportamiento y sus aplicaciones empresariales. Es el editor en jefe fundador del *International Journal of Data Science and Analytics* (JDSA) desde 2016, y también es editor en jefe de *IEEE Intelligent Systems*, la publicación de inteligencia artificial más antigua del IEEE. Fuente: https://profiles.uts.edu.au/Longbing.Cao

1.4.3. Dato - decisión

En el análisis de los datos existe un proceso conocido como “el ciclo de vida del dato”, la analítica de esto consiste en desentrañar la información para responder preguntas, entender los datos y como manifestar los resultados, desde el pasado hasta el futuro, desde un análisis explícito hasta uno implícito (Cao, 2018). Este proceso se conoce como analítica de ciclo completo o "*data-to-insight-to-decision*", y se dividen en cuatro grandes etapas:

Datos del pasado (análisis histórico): Se enfoca en responder “¿qué pasó?” y “¿por qué pasó?”, utilizando modelos y diseño experimental. Aquí se trata de entender hechos conocidos a través de un enfoque reactivo.

Datos del presente (análisis en tiempo real): Busca responder “¿qué está pasando?” y “¿por qué ocurre ahora?”. Este análisis facilita la toma de decisiones inmediatas.

Datos del futuro (análisis predictivo): Intenta prever “¿qué sucederá?” y “¿por qué podría suceder?”, mediante modelos de predicción, agrupamiento y estimación de eventos futuros (Cao, 2018).

1.4.4. Evolución del análisis de datos

La analítica de datos ha evolucionado desde el análisis de datos pequeños y simples, basado en hipótesis, hasta grandes volúmenes y complejos, centrada en el descubrimiento de conocimiento sin hipótesis previas, por ejemplo, detección de patrones con el uso de algoritmos ML no supervisados. Actualmente, la analítica es más reconocida e innovadora

para comprender su evolución y mapa conceptual, (Cao, 2018) propone entenderlo en dos dimensiones:

- Nivel de visibilidad, automatización y capacidad tecnológica: a medida que aumenta la complejidad de los datos, disminuye la visibilidad para los usuarios y se vuelve más difícil automatizar el análisis.

- Grado de complejidad e inteligencia (*X-complexity* y *X-intelligence*): al avanzar hacia formas más avanzadas de analítica, se incrementa la complejidad de los datos, este proceso lleva desde analíticas básicas a capacidades más profundas de interpretación y acción.

De igual manera dentro del espectro de la evolución analítica (ver Figura 1.3.) se distinguen dos eras principales basándose en el enfoque disciplinario. La primera era abarca un enfoque explícito-descriptivo que se centra en tareas como reportes, análisis estadístico, alertas y pronósticos. Esta etapa describe lo que ha ocurrido, con técnicas visibles y estructuradas. Por otro lado, la segunda era comprende un análisis implícito-profundo ya que usa técnicas como modelado predictivo, optimización, analítica prescriptiva y entrega de conocimiento accionable para responder no solo qué pasará, sino qué acciones tomar al respecto.

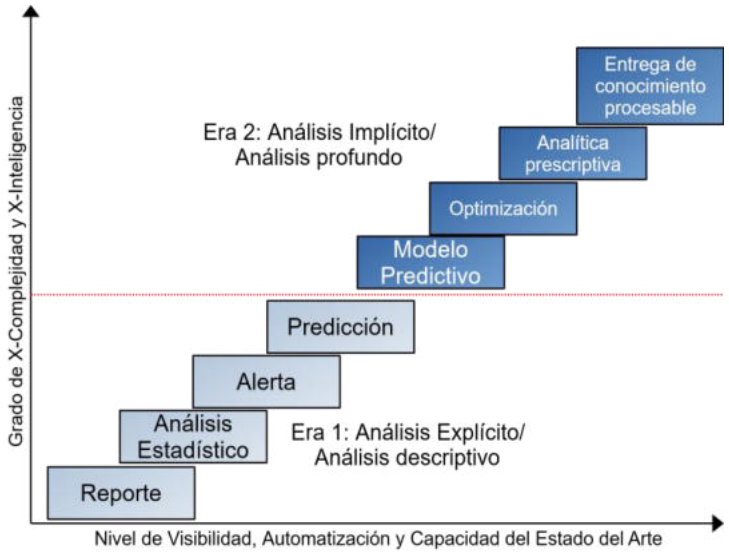


Figura 1.3. Evolución y espectro analítico desde explícito hasta implícito.

Fuente: Adaptado de Cao, (2018).

La **era de la analítica explícita**, centrada en la analítica descriptiva, se caracteriza por enfoques tradicionales en los que se trabaja sobre lo que se conoce y se desea analizar. En este contexto, el objetivo principal es describir e interpretar la distribución, evolución y generación de los datos, partiendo del supuesto de que "sabemos lo que sabemos".

Los problemas que se abordan en esta etapa tienen una naturaleza bien definida hacia aspectos previamente identificados, guiado por hipótesis formuladas con base en el conocimiento experto del dominio, es decir tiene una metodología hipotético-dirigido. En este enfoque los datos son recopilados y analizados con el fin de comprobar y explicar dichas hipótesis, aplicando técnicas estadísticas, matemáticas y computación. Los

resultados que se obtienen en esta fase ayudan a describir qué ha ocurrido o qué podría ocurrir, pero suelen estar limitados al uso de conjuntos pequeños.

La **era de la analítica implícita**, también conocida como analítica profunda, surge como respuesta a las limitaciones evidentes de la analítica explícita tradicional. En los últimos años, la comunidad analítica ha comenzado a reconocer estas limitaciones, lo que ha impulsado un cambio de enfoque hacia técnicas capaces de descubrir significados ocultos y patrones complejos no visibles a simple vista. En este nuevo paradigma, el objetivo ya no es simplemente confirmar lo que se conoce, sino explorar lo que no se sabe, pero esta oculto en los datos que se estudian. Es como estar ciego e interactuar con el entorno, tratando de determinar que objetos son los que conforman el ambiente o con los que se está interactuando, esto a través de la información como texturas, tamaño, etc. Es decir, no se tiene claridad sobre qué se está analizando ni qué se espera descubrir, sin embargo, la respuesta existe y es completamente impulsado por los datos, revelando por sí mismos relaciones y conocimientos ocultos, más allá de las hipótesis planteadas por expertos o el conocimiento del dominio.

1.4.5. Innovación en ciencia de datos: Retos y oportunidades

La innovación en ciencia de datos se revisa desde dos enfoques el uno desde las preocupaciones de las comunidades científicas y profesionales, incluyendo estadística, informática, computación, organismos gubernamentales y centros de investigación. Este enfoque consigue identificar los principales temas y problemas reconocidos por actores clave del ecosistema de ciencia de datos, es decir los tomadores de decisión. El segundo enfoque se basa en analizar la complejidad intrínseca de los

problemas, que es más profundo y exigente ya que busca comprender las dificultades de la manipulación, interpretación y explotación de datos complejos.

Los desafíos que enfrentan las aplicaciones de ciencia de datos son clasificados en diferentes áreas. En el entorno empresarial consiste en identificar, representar y cuantificar las complejidades y formas de inteligencia presentes en los datos, conocidas como X-complejidades y X-inteligencia. Solo con una comprensión profunda de estos factores será posible diseñar metodologías y tecnologías eficaces para incorporarlos a los procesos.

La base matemática y estadística también enfrenta limitaciones, ya que muchas veces no logra capturar adecuadamente complejidades que aparecen en las bases de datos masivas, ni traducirlas en información útil para la toma de decisiones.

1.5. Big Data

El *Big Data* (BD) se refiere al manejo, procesamiento y análisis de grandes volúmenes de datos que son demasiado complejos para ser tratados con herramientas y métodos tradicionales. Los conjuntos de datos en BD se caracterizan por tener volumen (cantidad masiva de datos), variedad (distintos tipos de datos) y velocidad (generación y descarga). En campos como meteorología, climatología, medio ambiente y ciencia de datos, se generan datos masivos a través de distintas fuentes como modelos climáticos, imágenes satelitales y series temporales de distintas variables

meteorológicas medidas por estaciones meteorológicas y otros sensores en tiempo real.

Los servidores locales o computadoras personales suelen tener limitaciones técnicas y tecnológicas para poder extraer datos masivos, en especial el bajo nivel de internet o el almacenamiento. Para resolver este problema existe lo que se conoce como “programación en la nube”, es decir la ejecución de aplicaciones y servicios se realiza en entornos basados en la nube. La nube se refiere a una red de servidores remotos con la capacidad de procesar y administrar datos de manera distribuida, accesible desde cualquier lugar con conexión a internet.

Acceso remoto: Los servicios, datos y aplicaciones se almacenan y gestionan en plataformas remotas y se acceden a través de internet.

Escalabilidad: La nube ofrece la capacidad de escalar los recursos computacionales (como el almacenamiento, la memoria, la CPU, etc.) según las necesidades del usuario.

Flexibilidad y elasticidad: Las aplicaciones en la nube llegan a adaptarse rápidamente a los cambios de demanda.

Costo-eficiencia: el costo que se asume es por los recursos que se utilizan (modelo "*pay-as-you-go*).

Computación distribuida: En la nube, los cálculos y los datos se procesan y almacenan en servidores.

Figura 1.4. Big data en climatología.

Fuente: Obtenido con ChatGpt Open AI, (2025).

La Figura 1.4. indica que en diferentes proyectos se utilizan datos climatológicos espaciales provenientes distintas fuentes como son: los modelos atmosféricos acoplados CMIP6 y datos satelitales con asimilación de datos como CHIRTS. Ambas fuentes manejan grandes volúmenes de datos, por lo tanto, la manera más eficiente de trabajar con estos datos es mediante *Big Data*. Por lo cual, es necesario utilizar herramientas que faciliten el acceso y manejo de este tipo de tecnologías de la información para tener un producto más eficiente.

Los datos meteorológicos son complejos por su tamaño, naturaleza multidimensional y la velocidad a la que se generan. Por ejemplo, los modelos climáticos contienen millones de puntos de datos por cada variable (temperatura del aire, precipitación, etc.) a lo largo del tiempo y el espacio. CMIP6 y CHIRTS contienen valores que abarcan todo el planeta y cubren largos periodos de tiempo (varias décadas o siglos de datos). Estos conjuntos son enormes, multidimensionales (por ejemplo, latitud, longitud,

tiempo, variables climáticas) y su manipulación no es eficientemente con herramientas convencionales. Estos datos están almacenados en la nube y en servidores remotos, y sería ineficiente descargarlos completamente en un equipo local.

Existen herramientas de programación que hacen posible trabajar sobre los metadatos de la información almacenada en la nube, donde el filtro, la selección y el procesamiento de subconjuntos de datos se llega a obtener con rapidez. En el código, por ejemplo, se selecciona la variable de temperatura de la superficie (tas) y los datos diarios, filtrando los resultados según la resolución espacial. Esto es posible gracias a la infraestructura diseñada para manejar *Big Data*.

CAPÍTULO II

VARIABLES METEOROLÓGICAS

2.1. Temperatura del aire

La temperatura del aire es una variable fundamental en el estudio del clima y la meteorología, ya que refleja el contenido térmico de la atmósfera en función de múltiples procesos físicos. En términos físicos, la temperatura representa la energía cinética media de las moléculas de un cuerpo, producto del movimiento aleatorio a nivel microscópico y se manifiesta en los sistemas termodinámicos como el equilibrio térmico cuando dos cuerpos están en contacto.

La temperatura atmosférica se ve influenciada por factores como la radiación solar absorbida por la superficie terrestre, los flujos verticales de calor generados por la convección y la advección horizontal de masas de aire. Por lo que, el valor de temperatura es el resultado de un sistema complejo de intercambios energéticos entre la biosfera, atmósfera y litosfera.

El calor es una forma de energía que se transfiere de un cuerpo a otro debido a una diferencia de temperatura y la temperatura es una propiedad que describe el estado térmico de un sistema. Incluso las unidades de medida son distintas, ya que el calor se mide en julios (J) o calorías (cal), mientras que la temperatura se mide en grados Celsius (°C), Kelvin (K) o Fahrenheit (°F).

La transferencia de energía, según la segunda ley de la termodinámica, fluye desde un cuerpo con mayor temperatura hacia otro con menor temperatura cuando existe un gradiente térmico. La transferencia de calor se produce a través de tres mecanismos fundamentales: conducción, convección y radiación.

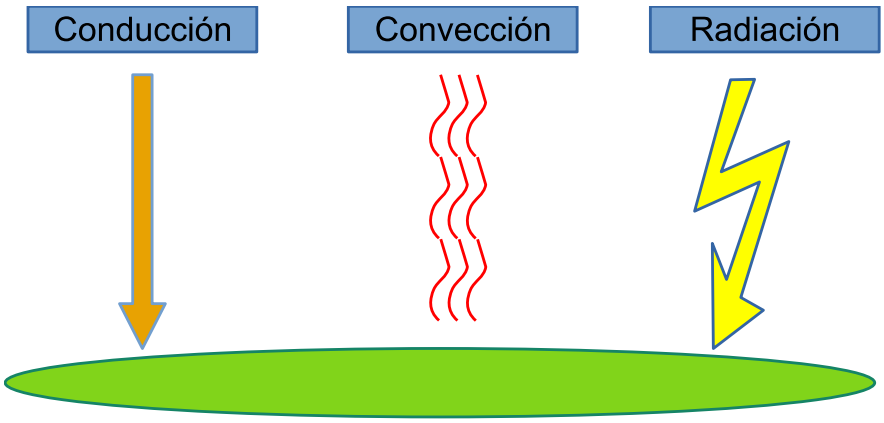


Figura 2.1. Mecanismos de transferencia de calor en la atmósfera.

Fuente: elaboración propia.

Como se observa en la Figura 2.1. la conducción es la transferencia de energía térmica a través de un material debido a una diferencia de temperatura. Su eficacia varía según el medio; por ejemplo, el agua conduce mejor el calor que el aire. La convección implica el movimiento de masas de fluido (líquido o gas) debido a diferencias de densidad causadas por el calentamiento, como en la atmósfera donde el aire caliente asciende y el frío desciende. Finalmente, la radiación es la transferencia de calor mediante ondas electromagnéticas, sin necesidad de un medio material, como la

energía solar que viaja por el vacío del espacio y calienta la Tierra al ser absorbida.

2.1.1. Cambios de temperatura

Los cambios de temperatura que se presentan en la atmósfera son la respuesta de varios sistemas dinámicos complejos que interaccionan cambiando la dinámica atmosférica en un periodo de tiempo. El factor más incidente y macro es la rotación de la Tierra, lo cual influye en las variaciones diarias. De igual manera, existen condiciones meteorológicas locales que afectan en los cambios de magnitud en las variables climáticas y curiosamente se observa cómo los periodos mensuales más cálidos y fríos no coinciden exactamente con los periodos de máxima o mínima radiación solar entrante. Aunque el sol alcanza su punto más alto en el solsticio de verano, el hemisferio norte suele registrar sus temperaturas más elevadas en julio y agosto, y las más bajas en enero y febrero, tras el solsticio de invierno.

En la troposfera, que es la capa donde ocurren los fenómenos meteorológicos, la temperatura disminuye progresivamente con la altitud. Aunque el Sol calienta desde arriba, la superficie terrestre absorbe mejor la radiación y actúa como la principal fuente de calor para el aire circundante. A medida que el aire se eleva, se expande y enfría, un fenómeno que ocurre sin intercambio de calor con el entorno, conocido como enfriamiento adiabático. El gradiente adiabático seco, de -1 °C por cada 100 metros, se aplica al aire no saturado, mientras que el gradiente adiabático húmedo, de entre -0.5 y -0.9 °C por cada 100 metros, se refiere al aire saturado que libera calor latente por condensación (Moller, 1956).

En ciertas condiciones, se llega a dar una inversión térmica, en la que la temperatura aumenta con la altitud. Este fenómeno limita el ascenso del aire y favorece la acumulación de contaminantes, así como la formación de nieblas, especialmente en valles durante el invierno (Spiridonov & Ćurić, 2020).

2.2. Presión atmosférica

La definición de presión atmosférica era desconocida, por sus efectos cotidianos, como la limitación práctica de ascenso de ciertos materiales, se comprendía como evidencia del *horror vacuis*, o sea, el 'terror al vacío' que manifestaba la naturaleza, dado que se pensaba que el aire carecía de peso y se elevaba por propia cuenta.

En 1643, el peso del aire fue descubierto por el físico italiano Evangelista Torricelli (1608-1647), a través de los primeros experimentos que dieron pie a la creación del barómetro. Su experimento más famoso consistió en comparar el comportamiento del mercurio y del agua al ser introducidos en un tubo curvo, conocido hoy como tubo de Torricelli.

Estas experiencias le sirvieron a la polímata francés Blaise Pascal (1623-1662), para medir el peso del aire atmosférico en distintas locaciones geográficas y distintas altitudes, como la cima del volcán Puy-de-Dome al sur de Francia. Pero no fue hasta 1654, gracias a los experimentos con los hemisferios de Magburgo del físico alemán Otto Von Guericke (1602-1686), que la existencia de la presión atmosférica fue demostrada públicamente.

La presión atmosférica o presión barométrica es la fuerza que ejerce el conjunto de gases mezclados que constituyen la atmósfera, sobre la superficie terrestre y los elementos que se encuentren sobre ella. Dicha fuerza se da por unidad de superficie, o sea, es equivalente al peso de la columna de aire que se extiende desde un punto de la superficie de la Tierra, hasta los límites superiores de la atmósfera. En términos más simples, representa el peso de la columna de aire que se encuentra sobre un punto determinado. Cuando la atmósfera está en reposo, esta presión refleja directamente el peso del aire en esa columna.

La presión atmosférica y sus variaciones a lo largo de un período de tiempo constituyen un dato usual en el estudio del clima. Sin embargo, el aire varía de densidad conforme se aleja del suelo y además se ve afectado por la temperatura, por lo que no suele ser fácil calcular la presión en un punto determinado. Su valor disminuye con la altitud, ya que existe menos masa de aire en las capas superiores. Este descenso se explica mediante tres procesos principales: el ascenso del aire, la disminución de la densidad del aire y la reducción de la masa de aire por divergencia en niveles altos. Ver Figura 2.2.

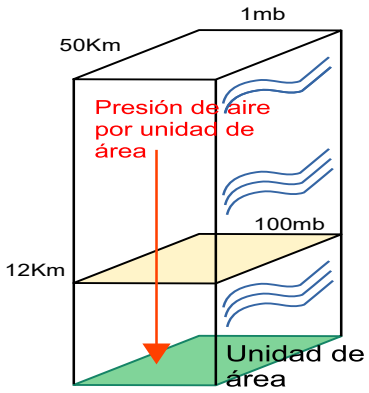


Figura 2.2. Ejemplo del mecanismo de presión atmosférica en una unidad de área.

Fuente: elaboración propia.

La presión estándar al nivel del mar, conocida como atmósfera (atm), se utiliza como valor de referencia y corresponde a 1013,25 hectopascales (hPa) o 760 mmHg equivalente a una fuerza aproximada de 100,000 N/m². Esta presión es resultado del peso del aire debido a la atracción gravitacional del planeta, y depende de factores como la masa y el radio del planeta, así como de la cantidad, composición y distribución vertical de los gases. Además, llega a verse influida por la rotación planetaria y fenómenos locales como la velocidad del viento o las variaciones de temperatura. En meteorología, la presión se expresa comúnmente en milibares (mb) o pascales (Pa), y se distribuye de forma uniforme en todas las direcciones. A su vez, la presión depende directamente de la densidad y la temperatura del aire, y su variación impulsa los movimientos atmosféricos que buscan equilibrar las diferencias de presión.

2.2.1. Distribución de presión: Isobaras

La distribución de la presión barométrica es un elemento fundamental en el análisis y la predicción meteorológica. En superficie, este campo se visualiza mediante cartas isobáricas que trazan líneas llamadas isobaras —líneas que unen puntos con igual presión atmosférica—. En altura, se utilizan cartas de altura geopotencial con isohipsas, que representan la altitud de una superficie de presión constante. Estas representaciones identifican las zonas con diferentes sistemas de presión y sus respectivos gradientes: cuanto más cercanas están las isobaras, mayor es el gradiente de presión y, por lo tanto, más rápida es la velocidad del viento asociada. Gradientes suaves, en cambio, se reflejan en isobaras ampliamente separadas (Escuela Náutica Neptuno 2023). Ver Figura 2.3.

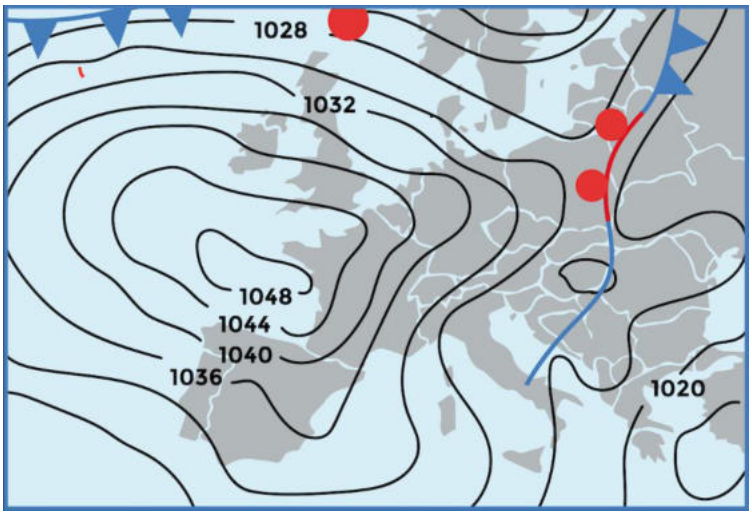


Figura 2.3. Distribución de campos de presión atmosféricas, mostrando líneas isobáricas.

Fuente: Escuela Náutica Neptuno, (2023).

El gradiente de presión describe cómo varía la presión atmosférica. La mayor variación ocurre en sentido vertical, con un descenso aproximado de 1 hPa por cada 10 metros cerca del nivel del mar. En dirección horizontal, las variaciones son menores entre 1 y 2 hPa cada 100 km significativas para la dinámica atmosférica (Hare, 1957). Estos cambios se producen por razones térmicas o dinámicas. Por ejemplo, el ascenso del aire reduce la presión en superficie, mientras que su descenso la incrementa. Las isobaras, al ser trazadas sobre mapas meteorológicos, forman patrones característicos como depresiones o dorsales, sumando datos de estaciones meteorológicas, se realizaría pronósticos a corto y mediano plazo.

2.3. Humedad relativa

La historia del desarrollo del concepto de humedad relativa incluye varios estudios relacionados con la termodinámica del aire y el ciclo hidrológico. En la antigüedad existía una percepción intuitiva de la relación existente entre el clima, aire y la humedad, fue en los siglos XVII y XVIII cuando empezaron a surgir las primeras mediciones cuantitativas gracias a la invención de instrumentos de medición como el higrómetro de cabello. Robert Hooke proporcionó el nombre a este instrumento basándose en términos griegos, de la misma forma, lideró estudios del comportamiento de la humedad relativa en varias ciudades, entre sus observaciones determinó que existe un desfase temporal entre la humedad y la temperatura, observando en otoño un desplazamiento no significativo en el tiempo.

Las herramientas de medición evolucionaron, dando la oportunidad de acuñar un concepto más preciso y técnico de la humedad relativa. En

términos sencillos, es un porcentaje que determina la cantidad de vapor de agua que existe en el ambiente, para obtener este valor porcentual se compara con un valor máximo el cual es determinado por una temperatura específica preestablecida sin que ocurra la condensación del agua. Para calcular este valor se realiza el cociente de la presión de vapor actual y la presión de vapor de saturación a la misma temperatura, multiplicado por 100.

Con los avances de la física del siglo XIX, particularmente en el estudio de los gases y el vapor de agua, se establecieron las bases científicas para definir y calcular la humedad relativa con precisión, incluyendo lecturas de barómetros y termómetros. Ecuaciones físicas utilizan relaciones entre la temperatura, presión y flujo de aire seco. Asimismo, existen modelos dinámicos que simulan el comportamiento de la temperatura y humedad relativa en sistemas específicos, considerando correlaciones no lineales entre variables como temperatura y masa de vapor.

Una de las ecuaciones más comunes es la fórmula de Magnus y sus coeficientes, en donde se explica cómo influye la presión de vapor de saturación en función de la temperatura. La presión de saturación (e_s) es no lineal respecto a la temperatura: se incrementa exponencialmente con la temperatura. Por eso, en días cálidos, el aire contiene más vapor de agua antes de saturarse (Pierrehumbert et al. 2008). La fórmula de Magnus brinda una aproximación para relacionar la temperatura con la e_s, a través de coeficientes los cuales se obtienen con una aproximación empírica, la fórmula se define:

$$e_s(T) = a \cdot exp\left(\frac{b \cdot T}{T+c}\right) \qquad (1)$$

Donde a, b, c son los coeficientes de Magnus y la e_s se obtiene en hPa o mbar. Esta forma de la fórmula de Magnus es válida principalmente en el rango de -45 °C a +60 °C. Utilizando la fórmula (1) se calcula la humedad relativa a partir de la temperatura del aire T y el punto de rocío

T_d(*Pierrehumbert, Brogniez y Roca* 2008):

$$HR = 100 \times \frac{e_s(T_d)}{e_s(T)} \qquad (2)$$

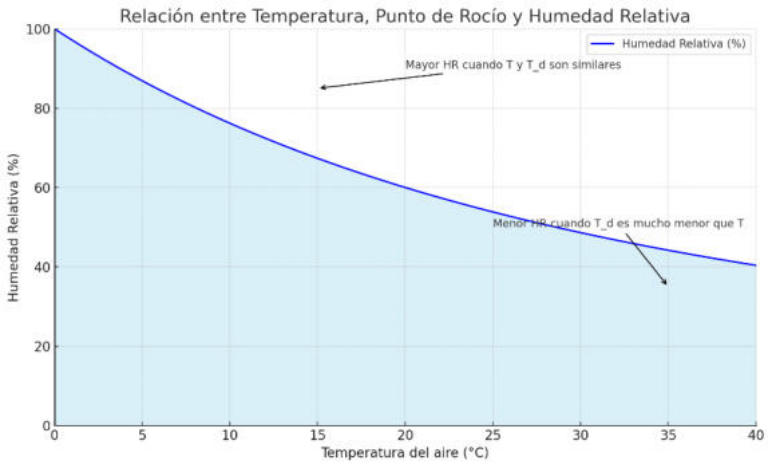


Figura 2.4. Distribución de campos de presión atmosféricas, mostrando líneas isobáricas.

Fuente: Nadal y Muñuzuri, (2006).

El comportamiento de la humedad relativa está influenciado por una serie de factores ambientales, cuya atribución varía significativamente según la región geográfica y las condiciones meteorológicas locales. Se ha revisado cómo la temperatura tiene su influencia sobre la humedad, sin embargo, otras variables climáticas afectan directa o indirectamente, por ejemplo, la presión atmosférica tiene una relación no tan directa como la temperatura, sin embargo, muestra una relación donde presiones más altas comprimen el volumen de aire, facilitando una mayor retención de moléculas de vapor de agua por unidad de volumen. De la misma manera, la velocidad del viento modula los procesos de evaporación y transporte de humedad. En ocasiones vientos suaves acumulan vapor de agua en zonas específicas, favoreciendo una humedad relativa más alta. En cambio, vientos intensos dispersan el vapor de agua y reducen localmente la humedad al incrementar la ventilación.

2.4. Precipitación

La precipitación es uno de los procesos fundamentales del ciclo hidrológico y constituye un elemento esencial en la dinámica atmosférica. Su estudio requiere comprender en profundidad los mecanismos físicos que dan origen a la formación de nubes, ya que éstas son el medio donde se desarrollan los procesos micro físicos que conducen a la liberación de agua en forma líquida o sólida hacia la superficie terrestre. Las nubes, definidas como manifestaciones visibles del proceso de condensación o deposición del vapor de agua en la atmósfera, están compuestas por diminutas gotas de agua y cristales de hielo en suspensión. Estas estructuras adoptan una gran diversidad de formas desde capas estratificadas hasta columnas convectivas

de gran desarrollo vertical y llegan a extenderse desde la superficie terrestre, como en el caso de la niebla, hasta altitudes superiores a los 10–15 km en la parte alta de la troposfera (Bezrukova & Chernokulsky, 2019).

2.4.1. ¿Cómo se forman las nubes?

Antes de que se formen las nubes y la precipitación, deben ocurrir una serie de procesos físicos fundamentales en la atmósfera. El primero consiste en el aumento progresivo de la humedad relativa en un volumen considerable de aire, hasta alcanzar el punto de saturación, es decir, alrededor del 100%. Este fenómeno suele estar asociado a movimientos verticales del aire en condiciones atmosféricas inestables y estratificadas, o bien como consecuencia de procesos dinámicos y térmicos. El segundo proceso implica transformaciones microfísicas del agua, como la nucleación, difusión y recolección de partículas, fenómenos que se estudian en la denominada microfísica de nubes (Bezrukova & Chernokulsky, 2019).

La altura de condensación de nubes (CCL, por sus siglas en inglés) es el nivel al que el aire, al ascender enfriándose adiabáticamente en condiciones secas, alcanza la saturación. En este punto, el vapor de agua comienza a condensarse sobre núcleos de condensación (CCN), partículas microscópicas suspendidas en el aire, y se forman las nubes. Cuando una mezcla de masa de aire alcanza un valor que iguala a la razón de saturación se conoce como “Puntos de rocío”, en ese momento la humedad relativa se aproxima al 100%, iniciando el ascenso húmedo y favoreciendo la formación de nubes.

La formación de nubes depende principalmente de la humedad relativa, ya que ésta indica el grado de saturación del aire y, por ende, la probabilidad de que ocurra condensación. Aproximadamente el 99% del vapor de agua se concentra en la troposfera, lo cual explica por qué casi todos los tipos de nubes, desde cúmulos pequeños hasta cumulonimbos, se desarrollan en esta capa atmosférica. Incluso los sistemas más violentos, como huracanes o tornados, están confinados dentro de la troposfera (Spiridonov & Ćurić, 2020).

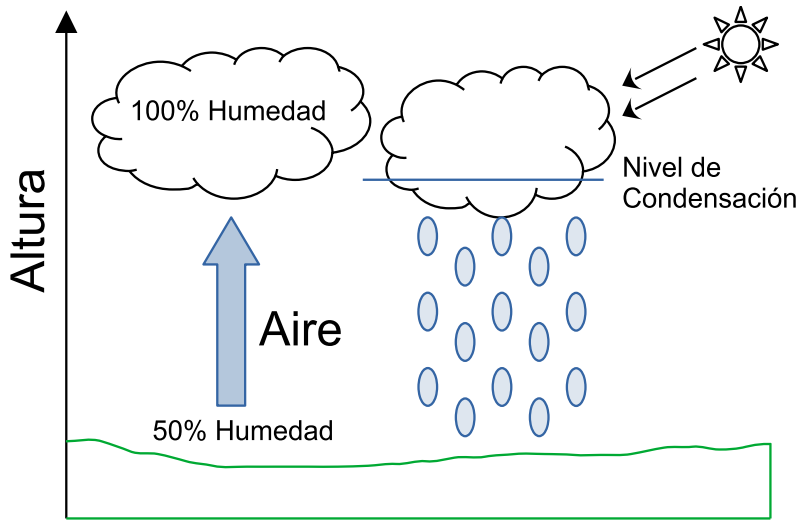


Figura 2.5. Proceso de formación de las nubes.

Fuente: elaboración propia

La Figura 2.5. indica que las nubes se generan cuando el aire se enfría por debajo de su punto de rocío, esto sucede por la radiación nocturna, mezcla con aire más frío o ascenso y expansión en altura. La cantidad máxima de vapor que es capaz de contener una porción de aire en equilibrio con una

superficie de agua pura depende exclusivamente de la temperatura. Cuando se excede esa cantidad, el exceso de vapor está disponible para condensarse en forma de gotas de agua.

El ritmo al que se libera vapor de agua para formar gotas de nube a medida que el aire se enfría se determina con la ecuación de Clausius-Clapeyron, que relaciona la presión de vapor de saturación con la temperatura absoluta (Kelvin). Esta ecuación consigue estimar tanto el contenido de vapor disponible como la evolución del grado de saturación con respecto al agua o al hielo. El aire húmedo requiere menos enfriamiento para alcanzar la saturación, mientras que el aire seco necesita mayor enfriamiento y un ascenso más prolongado para lograrlo (Hare, 1957).

2.4.2. Neblina

La neblina se define como un aerosol visible compuesto por diminutas gotas de agua o cristales de hielo suspendidos en el aire. Estas gotas varían en tamaño: en condiciones de temperaturas bajo cero, sus diámetros oscilan entre 2 y 5 micras, mientras que en temperaturas positivas se encuentran entre 7 y 15 micras, además se considera que hay presencia de neblina cuando la visibilidad atmosférica se reduce por debajo de los 1000 metros (Spiridonov & Ćurić, 2020).

La neblina corresponde a la base de una nube ubicada en la superficie terrestre o muy próxima, y para que se forme es necesario que el aire esté saturado de vapor de agua, lo cual implica una humedad relativa cercana al 100%, su origen está asociado a la condensación de diminutas gotas de agua suspendidas en el aire, mientras que las más densas tienden a desarrollarse

en zonas industriales debido a la alta presencia de partículas contaminantes que actúan como núcleos de condensación, facilitando la formación y crecimiento de las gotas (Bancroft, 1918).

Dentro de la atmósfera, entre las condiciones que favorecen la aparición de neblina se incluyen: alta presión atmosférica (tiempo anticiclónico), atmósfera estable, presencia de inversión térmica, ausencia de viento o circulación débil, condiciones topográficas favorables y existencia de núcleos de condensación.

Existen varios tipos de neblina, clasificados según el mecanismo de formación. Las que están formadas por enfriamiento del aire se dividen en tres categorías: neblina por radiación, neblina advectiva o neblina orográfica. Existe, además, la neblina formada por evaporación la cual aparece cuando el vapor de agua que se eleva sobre una masa de agua cálida, se condensa al entrar en contacto con aire más frío. La neblina sobre zonas continentales se forma principalmente debido al enfriamiento radiactivo ("neblina por radiación"), por lo que alcanza su máximo anual en otoño e invierno, y su máximo diario durante la noche y primeras horas de la mañana (Sachweh & Koepke, 1997). Y finalmente, la neblina frontal, es aquella que se genera cuando el aire cálido se desliza sobre aire frío en una zona delantera, produciendo condensación.

En zonas urbanas, intervienen otros elementos en comparación con las zonas rurales circundantes, el clima urbano se caracteriza por temperaturas del aire más elevadas (fenómeno de "isla de calor"), menor evapotranspiración, baja humedad relativa, reducción del movimiento del aire y acumulación de contaminantes atmosféricos, que intensifican la

formación de lo que se conoce como neblina urbana, a menudo asociada al smog (Sachweh & Koepke, 1997).

Un caso particular es la neblina subfundida, que se forma en noches despejadas cuando la superficie terrestre se enfría rápidamente por pérdida de calor hacia el espacio. Durante este proceso, el vapor de agua se condensa en gotas líquidas que permanecen en estado líquido aun estando por debajo de los 0 °C. Cuando estas microgotas entran en contacto con superficies, se congelan formando depósitos de cristales de hielo en forma de plumas blancas (Spiridonov & Ćurić, 2020).

2.4.3. Precipitación y lluvia

La lluvia es un tipo de precipitación que se produce cuando gotas individuales de agua, formadas en las nubes, caen hacia la superficie terrestre. El agua de fuentes como los océanos, lagos, entre otros, se evapora, luego se condensa formando nubes, luego se precipita y retorna al sistema, a este ciclo se le conoce como ciclo hidrológico.

El proceso físico que explica la formación de la lluvia se conoce como "proceso de Bergeron", propuesto inicialmente por Tor Bergeron en la década de 1930, y posteriormente desarrollado por Walter Findeisen. Este proceso explica desde la condensación del vapor de agua hasta su posterior crecimiento en forma de gotas, las cuales se vuelven lo suficientemente pesadas para precipitar. Se basa en la diferencia de presión de vapor entre el agua líquida y el hielo a temperaturas bajo cero. A una misma temperatura, la presión de vapor sobre una superficie de agua líquida es mayor que la presión de vapor sobre una superficie de hielo. Esto significa

que, en una nube mixta, el vapor de agua tiende a depositarse sobre los cristales de hielo, en lugar de mantenerse alrededor de las gotas de agua líquida. Este desequilibrio provoca dos procedimientos simultáneos, las gotas de agua se evaporan lentamente, mientras los cristales de hielo crecen por deposición directa del vapor de agua sobre su superficie (Pruppacher et al. 1998). Cuando los cristales crecen suficientemente como para caer, en ocasiones llegan a fundirse en gotas, mantenerse como hielo o formar agua nieve. Ver Figura 2.6.

Figura 2.6. Las distintas maneras en las que los cristales llegan en la superficie (izquierda: agua nieve; centro: nieve; derecha: lluvia).

Fuente: Spiridonov y Ćurić, (2020).

No todas las gotas de lluvia alcanzan el suelo; en presencia de capas de aire seco y cálido, una parte de ellas se evaporan durante la caída. Cuando la precipitación se evapora antes de tocar el suelo, el fenómeno se denomina virga, y es común en regiones áridas y cálidas.

Como fenómeno atmosférico, la lluvia se caracteriza por dos aspectos principales los cuales hacen posible su clasificación (ver Figura 2.7.):

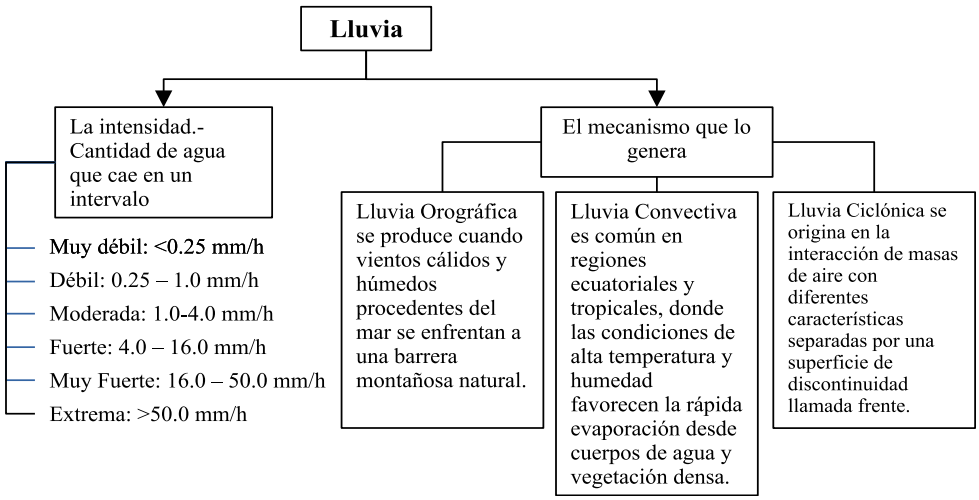


Figura 2.7. Tipos de lluvia según sus características.

Fuente: elaboración propia.

2.5. Radiación solar

La radiación solar constituye el principal motor energético del sistema Tierra-atmósfera. Aproximadamente el 99.9% de la energía que calienta la superficie terrestre proviene del Sol, lo que la convierte en el factor externo más determinante en la regulación del clima, los procesos atmosféricos y el soporte de la vida en el planeta (Visconti, 2016). Esta energía no solo calienta la atmósfera, los océanos y los continentes, sino que también impulsa los mecanismos generales de circulación atmosférica y oceánica, dando origen a la formación de masas de aire y sistemas meteorológicos, que genera vientos y corrientes marinas, y activa la fotosíntesis, proceso clave para el mantenimiento de los ecosistemas.

Desde una perspectiva comparativa, el aporte energético del interior terrestre es prácticamente insignificante: se estima que la cantidad de calor geotérmico es unas 5000 veces menor que la energía solar que incide diariamente sobre el límite superior de la atmósfera, y que incluso la energía térmica total disponible en el sistema terrestre es unas 300 veces menor que la radiación solar recibida cada día (Spiridonov & Ćurić, 2020). La marcada asimetría evidencia que los grandes flujos de energía solar son los responsables de las variaciones térmicas que, a su vez, originan fenómenos dinámicos a escala global como los cambios estacionales, la convección atmosférica y la redistribución energética entre regiones. Por lo tanto, comprender cómo el Sol transfiere energía al planeta y cómo esta energía se distribuye en la superficie terrestre es fundamental para entender los procesos climáticos y meteorológicos.

La energía liberada por el Sol a través de sus procesos atómicos internos se emite principalmente en forma de radiación electromagnética, comúnmente denominada energía solar (Nadal & Muñuzuri, 2006).

Desde un punto de vista físico, la radiación solar es una forma de energía radiante que se propaga en todas las direcciones en el espacio, y que aproximadamente la mitad del espectro solar corresponde a radiación de onda corta visible, mientras que el resto se distribuye mayoritariamente en el infrarrojo cercano y en menor proporción en la ultravioleta (Helmis & Nastos, 2012). Aunque solo una pequeña fracción de la energía emitida por el Sol alcanza la superficie terrestre, esta cantidad resulta suficiente para calentar la atmósfera e impulsar los procesos físicos, químicos y biológicos que mantienen las condiciones necesarias para la vida en el planeta.

Existen algunas leyes que describen y comprenden la radiación solar, donde se relacionan algunas variables físicas entre las que se destacan la temperatura y la longitud de onda. Entre las leyes principales cabe destacar la ley de Wien, ley de Kirchoff y la ley de Stephan-Boltzmann.

2.5.1. Espectros de emisión

El espectro de emisión de la radiación solar es la distribución de la energía electromagnética emitida por el Sol, según sus diferentes longitudes de onda o frecuencias y describe cuánta energía radiante emite el Sol en cada parte del espectro electromagnético (IDEAM, 2025). No obstante, no se termina de entender cómo interactúa la radiación solar con la atmósfera, los océanos y la superficie terrestre.

El espectro solar mostrado en la Figura 2.8. se asemeja al de un cuerpo negro ideal a una temperatura efectiva de aproximadamente 5778 K. Sin embargo, presenta irregularidades debidas a la absorción de ciertos gases en la atmósfera solar, lo que genera las conocidas líneas de Fraunhofer (Mecherikunnel & Richmond, 1980).

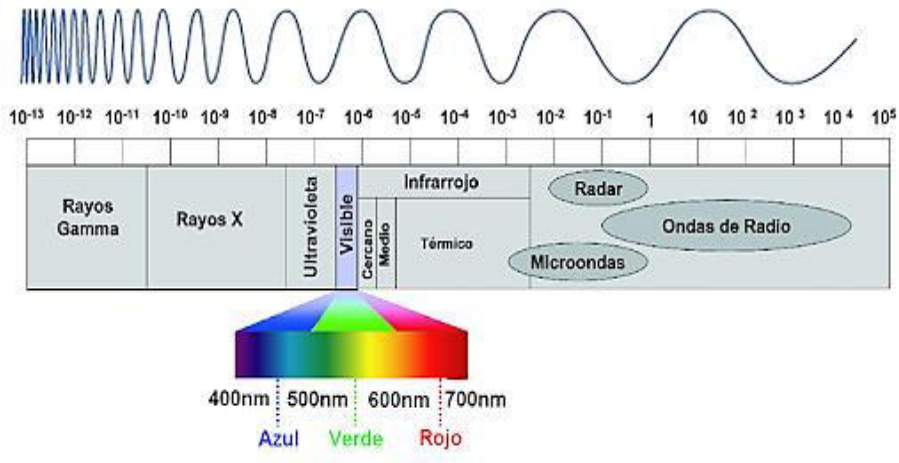


Figura 2.8. Espectro electromagnético y su longitud de onda característico en metros.

Fuente: IDEAM, (2025).

La energía solar asociada a longitudes de onda superiores a 4000 nm y la energía emitida por fuentes terrestres inferiores de 4000nm tienen un bajo porcentaje de influencia. Por esa razón, la radiación se suele dividirse en tres grupos importantes dentro de un rango. El primer grupo conforma los rayos x, rayos gamma y rayos ultravioleta en conjunto transportan un 9% del total de la energía solar, luego existe la radiación visible con el 41% de energía transportada y finalmente los rayos infrarrojos que transportan el 50% del total de la energía (Mecherikunnel & Richmond, 1980).

2.5.2. Constante solar

Aunque el Sol emite una enorme cantidad de energía en todas las direcciones, la Tierra recibe solo una pequeña fracción de esta, y dicha energía se cuantifica en condiciones ideales a partir de un parámetro

conocido como constante, es definida como la cantidad de energía radiante que incide por unidad de tiempo sobre una unidad de superficie perfectamente perpendicular a los rayos solares, situada en el límite superior de la atmósfera terrestre y a una distancia media entre la Tierra y el Sol; su valor aproximado es de 2 cal/cm²·min, equivalente en unidades del SI, alrededor de 1361 W/m² (Rodriguez & León, 2012).

La constante solar experimenta fluctuaciones del orden del ±1.5% debido a variaciones naturales en la actividad solar, como la aparición de manchas solares o perturbaciones en la emisión del Sol. Además, la distancia Tierra-Sol cambia ligeramente a lo largo del año debido a la excentricidad de la órbita terrestre, por lo que durante el año existen variaciones estacionales adicionales de entre 2% y 5% (Rodriguez & León, 2012).

Debido a la inclinación del eje terrestre, la esfericidad del planeta y la interacción con la atmósfera, la radiación que llega realmente a la superficie terrestre es notablemente menor a la constante solar (Nadal & Muñuzuri, 2006). Se estima que aproximadamente un 50% de la energía solar se pierde durante su tránsito por la atmósfera debido a procesos de absorción, dispersión y reflexión (Spiridonov & Ćurić, 2021). Estos fenómenos son fundamentales en el balance energético terrestre y condicionan la distribución de temperatura, el clima, y la dinámica atmosférica global.

Al estudio de cómo la radiación electromagnética se propaga a través de un medio se conoce como “transferencia radiativa” o “transferencia de radiación”, en climatología se utiliza este concepto para entender como la radiación solar se propaga a través de la atmósfera (Williams, 2021). Existen dos tipos de transferencia radiativa solar, y los resultados

correspondientes en reflexión/transmisión se tratan por separado: directo y difuso. Dichos conceptos se hablarán en los capítulos siguientes.

2.5.3. Radiación directa (S)

La radiación directa se asocia con el haz solar incidente directo. La radiación solar directa es aquella fracción del flujo solar que llega a la superficie terrestre sin alteraciones por procesos atmosféricos como la dispersión o la reflexión.

La magnitud de la radiación directa depende de múltiples factores: la altura solar sobre el horizonte (ángulo de elevación solar), la altitud geográfica, la nubosidad, la transparencia atmosférica y la cantidad de vapor de agua presente en el aire, que absorbe parte de la radiación. En condiciones ideales (cielo completamente despejado y atmósfera limpia), la radiación directa alcanza su máximo al mediodía, cuando el Sol está más alto en el cielo. Ver figura 2.9.

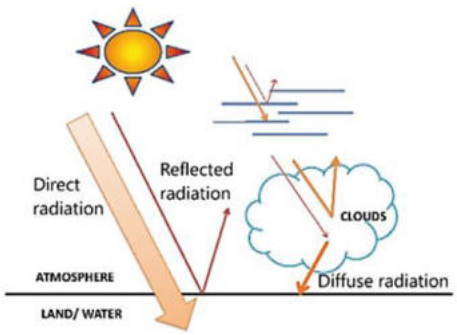


Figura 2.9. Radiación directa, difusa y reflejada.

Fuente: Spiridonov y Ćurić,(2021).

2.5.4. Radiación difusa (D)

La radiación difusa se asocia con la radiación incidente isótropa asumida y se diferencia principalmente en la condición de contorno, además es utilizada en el proceso de transferencia radiactiva multicapa basado en el método de duplicación/suma que se deriva del principio de invariancia propuesto por Chadrasekehar (Zhang et al., 2013).

Este fenómeno se conoce como dispersión atmosférica o reflexividad difusa, y afecta principalmente a las longitudes de onda más cortas del espectro electromagnético, como el azul, violeta y ultravioleta, lo que explica, por ejemplo, por qué el cielo se ve azul en un día claro (dispersión de Rayleigh) (Visconti, 2016).

La radiación difusa está presente en días nublados o parcialmente cubiertos. Su intensidad depende de la posición solar y la altitud del lugar (Spiridonov & Ćurić, 2021).

En días despejados, la radiación difusa también está presente desde el amanecer hasta el atardecer, complementando la radiación directa. Sin embargo, en días muy nubosos o con cielo cubierto, la componente difusa representa la totalidad de la radiación solar disponible en superficie.

2.5.5. Radiación global

La radiación global (G) es la suma de la radiación directa (S) y la radiación difusa (D) que incide sobre una superficie horizontal:

$$G = S + D$$

Este valor representa la radiación total que recibe la superficie terrestre a lo largo del día, y es fundamental para aplicaciones en meteorología, climatología, energía solar fotovoltaica, y agricultura, entre otras. La radiación global varía no solo con la hora del día y las condiciones atmosféricas, sino también con la latitud del lugar y la geometría de la superficie receptora que depende de inclinación y orientación del lugar (Nadal & Muñuzuri, 2006).

2.5.6. Relación con la atmósfera

La atmósfera terrestre es como un gas que rodea completamente al planeta y desempeña un papel fundamental en el equilibrio energético, el clima y la protección de la vida. Aunque se extiende más allá de los 1000 km, el 99% de su masa se concentra en los primeros 30 km sobre la superficie. Su estructura vertical está organizada en capas bien diferenciadas por su composición, temperatura, densidad y dinámica.

La radiación solar incidente sufre diversas modificaciones al atravesar la atmósfera. A unos 150 km de altitud, prácticamente llega el 100% de la radiación original, pero esta se reduce progresivamente en su trayecto hacia la superficie terrestre debido a procesos de absorción, dispersión y reflexión (Rodriguez & León, 2012). En la ionosfera, a unos 88 km, los rayos X y parte de los ultravioleta son absorbidos por átomos de hidrógeno y oxígeno, lo que provoca la ionización del aire. Más abajo, en la estratosfera, el ozono absorbe la mayor parte de la radiación ultravioleta restante. En la troposfera, el vapor de agua y el dióxido de carbono absorben radiación de onda larga en el espectro infrarrojo, provocando un aumento de temperatura en esta capa. Esta absorción varía en función de la humedad atmosférica, pudiendo

representar entre el 5% y el 20% de la energía solar incidente (Spiridonov & Ćurić, 2020), (Visconti, 2016).

El último proceso es la reflexión. Esta se produce cuando las partículas contra las que impacta la radiación son mayores que las longitudes de onda involucradas. Las gotas de agua presentes en las nubes reflejan entre un 30% y un 60% de la radiación, dependiendo de su densidad y extensión. Además, parte de la radiación que alcanza la superficie terrestre también es reflejada hacia el espacio (ver Figura 2.10). Esta fracción reflejada, conocida como albedo, depende del tipo de superficie: la nieve fresca alcanza a reflejar hasta un 90% de la energía incidente, mientras que superficies oscuras como los bosques reflejan apenas entre un 3% y un 10% (Spiridonov & Ćurić, 2021).

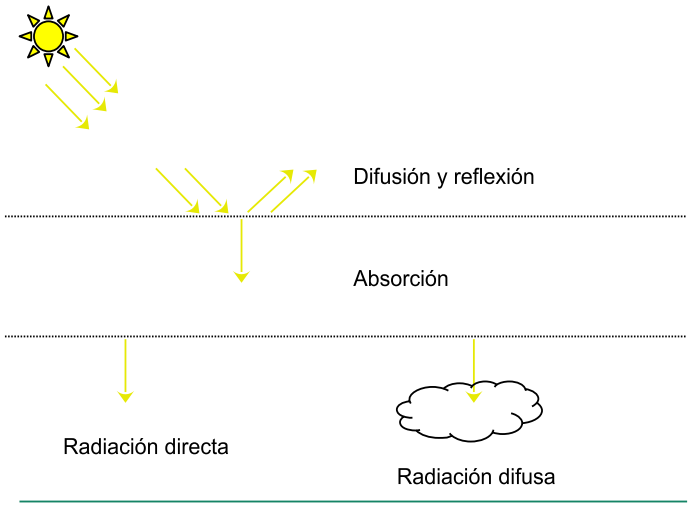


Figura 2.10. Procesos de la radiación solar en contacto con la atmósfera de la Tierra.

Fuente: elaboración propia.

El balance de radiación es el resultado neto entre la energía que ingresa al sistema Tierra-atmósfera y la que es reflejada o reemitida al espacio. En condiciones de cielo despejado, se estima que alrededor del 80% de la radiación solar alcanza la superficie terrestre, mientras que el 20% se pierde por absorción y dispersión. El balance térmico anual que se mantiene entre la radiación entrante del Sol y la radiación saliente emitida por la Tierra se conoce como el balance energético terrestre. Este balance hace que sea posible conservar una temperatura media global relativamente constante, a pesar de las fluctuaciones estacionales que provocan la llegada de ondas de frío o calor en distintas regiones del planeta. Pero hay que considerar que en algunas zonas no se recibe radiación de la misma manera ya que en las zonas comprendidas aproximadamente entre los 36° de latitud norte y sur reciben, en promedio, más energía solar de la que pierden hacia el espacio (Mecherikunnel & Richmond, 1980).

Esta distribución desigual de energía a nivel sinóptico es el motor que genera la circulación de la atmósfera y de los océanos, a través de procesos de compensación de este desequilibrio que se está produciendo, para lo cual transfiere el exceso de calor desde los trópicos hacia los polos mediante vientos globales y corrientes marinas.

2.6. Velocidad y dirección del viento

La circulación del aire en la atmósfera, comúnmente conocida como viento, es consecuencia directa del desigual calentamiento de la superficie terrestre por la radiación solar. Este calentamiento diferencial genera variaciones en la temperatura del aire en las capas inferiores de la atmósfera, lo cual provoca diferencias en la densidad del aire y, por consiguiente, en la presión

atmosférica. En zonas donde el aire se calienta intensamente, este se dilata, se vuelve menos denso y tiende a ascender, generando una disminución de la presión. Por el contrario, el aire frío es más denso y ejerce mayor presión sobre la superficie. Así, el desequilibrio térmico origina áreas de alta y baja presión atmosférica, que se distribuyen dinámicamente sobre la superficie del planeta (Rodriguez & León, 2012).

El viento se define como el desplazamiento del aire desde zonas de alta presión hacia zonas de baja presión, siguiendo la dirección del gradiente de presión. Este desplazamiento no es lineal, sino que está influenciado por múltiples factores, como la rotación terrestre (fuerza de Coriolis), el rozamiento con la superficie, y la configuración de los sistemas de presión (Grogg, 2005). La intensidad del viento está determinada por la magnitud del gradiente de presión: cuando las isobaras en un mapa meteorológico se presentan muy próximas, el gradiente es alto y se generan vientos fuertes; si las isobaras están más separadas, el gradiente es débil y los vientos resultantes son suaves (INDIES & AMERICA, 1892). Ver Figura 2.11.

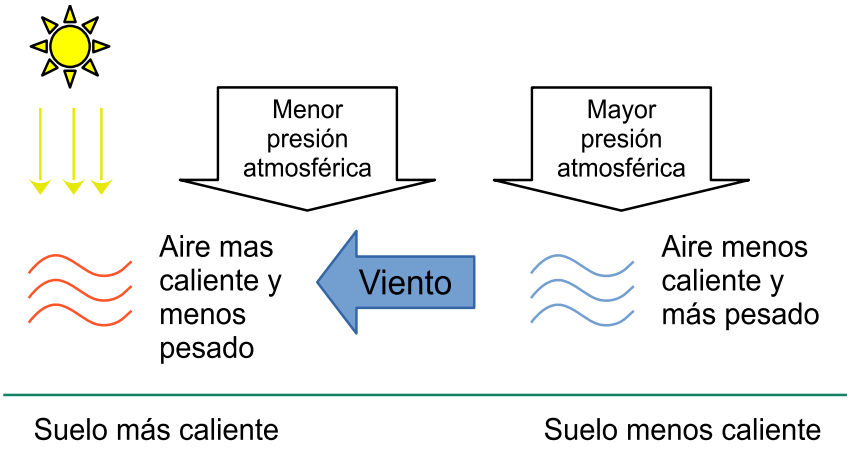


Figura 2.11. Gradiente de presión y movimiento del viento.

Fuente: elaboración propia.

La figura 2.12. muestra que la dirección del viento se expresa según la rosa de los vientos, que divide el horizonte en 8, 16 o hasta 32 sectores, denominando el viento según la dirección desde la cual proviene. Por ejemplo, un viento del norte es aquel que se desplaza desde el norte hacia el sur. Además, la velocidad del viento —generalmente medida en metros por segundo (m/s) o kilómetros por hora (km/h)— tiene una importancia crítica en múltiples aplicaciones, como el diseño de estructuras, la ubicación de instalaciones industriales, la dispersión de contaminantes y la planificación urbana (Rodríguez & León, 2012), (Spiridonov & Ćurić, 2021).

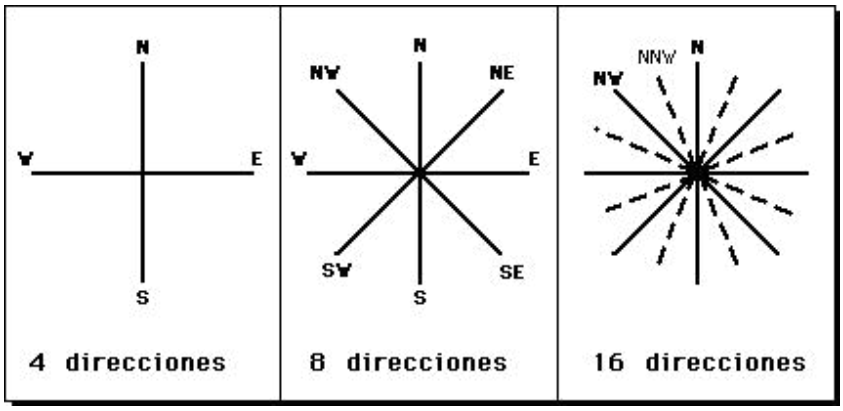


Figura 2.12. Distintos modelos de dirección de viento según la "Rosa de los vientos".

Fuente: Rodríguez y León, (2012).

CAPÍTULO III

SISTEMAS DE OBSERVACIÓN, MEDICIÓN Y REGISTRO

Uno de los primeros programas de observación global fue el *World Weather Watch* en la década de 1960, desde ese año hasta la actualidad ha existido una transformación digital en curso, la observación meteorológica ha evolucionado hacia un enfoque integrado, donde plataformas espaciales y terrestres se complementan para ofrecer una visión del sistema Tierra-atmósfera (Spiridonov & Ćurić, 2021).

Los sistemas de observación atmosférica miden variables meteorológicas que describen el clima como temperatura, humedad relativa, presión atmosférica, radiación solar, precipitación, velocidad y dirección del viento, entre otros, que son indispensables para la elaboración de pronósticos meteorológicos operativos, desarrollar modelos de alertas tempranas ante eventos extremos, estudios hidrológicos, monitoreo agrícola, evaluación ambiental y una amplia gama de aplicaciones socioeconómicas (OMM, 2010).

3.1. Escalas temporales y espaciales

Las observaciones sinópticas deben representar condiciones atmosféricas típicas en un radio de hasta 100 km alrededor de la estación, mientras que aplicaciones de escala local, como la meteorología agrícola o el monitoreo urbano, requieren representatividad en rangos espaciales mucho más reducidos, de hasta 10 km (OMM, 2010). Los requerimientos para la obtención de datos en meteorología son cubiertos mediante mediciones in

situ a través de estaciones o con sistemas de teledetección, incluidos aquellos instalados en satélites.

El Sistema Global de Observación (*Global Observing System*, GOS), diseñado para satisfacer estos requerimientos, está conformado por dos subsistemas, uno basado en la superficie que comprende tipos de estaciones, según su finalidad, como sinópticas de superficie, de altitud (*upper-air*), climatológicas, entre otras. Por otro lado, el subsistema espacial está compuesto por múltiples satélites con misiones de sondeo a bordo y un segmento terrestre encargado del control, comando y recepción de datos (Spiridonov & Ćurić, 2020).

La densidad de la red de estaciones y la frecuencia de medición a escala espacial y temporal proporcionan información como para realizar predicciones meteorológicas a corto plazo que demandan observaciones frecuentes y uniformemente distribuidas, mientras que los estudios climáticos globales se basan en escalas mayores y menor resolución temporal.

Según (Orlanski, 1975), las escalas horizontales se clasifican de la siguiente forma (con una incertidumbre de hasta un factor dos):

Microescala (<100 m): relevante en estudios de evaporación y procesos muy localizados, clave para la meteorología agrícola.

Topoescala o escala local (100 m a 3 km): adecuada para fenómenos como tornados o dispersión de contaminantes.

Mesoescala (3 km a 100 km): útil para analizar tormentas, brisas costeras y circulaciones locales.

Escala sinóptica o de gran escala (100 km a 3 000 km): abarca sistemas como frentes y ciclones.

Escala planetaria (>3 000 km): incluye ondas atmosféricas de gran longitud en la troposfera superior.

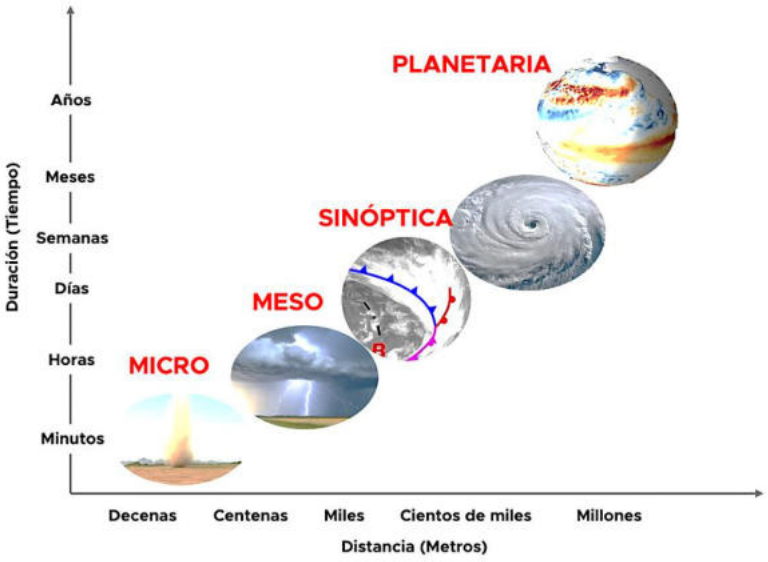


Figura 3.1. Distintas escalas de observación meteorológica según Orlanski 1975.

Fuente: Campos, (2025).

Cada una de estas escalas requiere una configuración distinta de observación, tanto en resolución como en cobertura temporal, para capturar

los procesos que caracterizan a la atmósfera en diferentes niveles de organización espacial los cuales están normalizados por la OMM. El término “estándar” se refiere a los instrumentos, métodos y protocolos usados para minimizar la incertidumbre en las mediciones. La OMM ha adoptado un sistema de estándares nacionales y regionales y que los instrumentos deben cumplir calibración y mantenimiento periódicamente.

Además, requiere personal capacitado y/o certificado por los servicios meteorológicos autorizados para garantizar que las observaciones se realicen conforme a los estándares exigidos. Los observadores deben interpretar adecuadamente las instrucciones para el uso tanto de técnicas manuales como instrumentales, de acuerdo con el sistema de observación implementado en su estación (OMM, 2010).

3.2. Medición de variables meteorológicas

Las mediciones instrumentales generan valores numéricos que representan el estado físico o meteorológico de la atmósfera local. Para las prácticas meteorológicas se emplean instrumentos de medición que tienen su base física.

3.2.1. Temperatura del aire

Esta medición se realiza mediante termómetros, cuya lectura estándar se toma entre 1,25 m y 2 m sobre el nivel del suelo en un entorno libre de interferencias térmicas externas. Según la OMM los requisitos meteorológicos para las mediciones de temperatura en meteorología están relacionados principalmente con el aire cerca de la superficie de la Tierra,

la superficie del terreno, la temperatura del suelo en distintas profundidades, la atmósfera superior y los niveles de superficies del mar y lagos (OMM, 2010).

Existen tres tipos de aparatos de medida de temperatura. El primero es el termómetro de dilatación los cuales se basan en una lectura analógica a través de la expansión líquida del mercurio o de alcohol dentro de un tubo capilar los cuales miden la temperatura ambiente (ordinarios), temperaturas mínimas o máximas (Nadal & Muñuzuri, 2006). Además, existen los de suelo que se curvan para insertarse en la tierra.

Los termógrafos se basan en el principio físico de la dilatación térmica de los metales donde los registros de temperatura son dados por la variación de longitud de varillas metálicas. Son poco precisos y requieren de calibraciones frecuentes.

Los termómetros eléctricos basados en el cambio de las propiedades eléctricas de algunos materiales ante los cambios de temperatura, por ejemplo, el platino, que cambia su resistencia en función de la temperatura a la que está expuesto, este tipo de medidores se les conoce como termómetros de resistencia eléctrica. También existen los termistores, los cuales son semiconductores con alta sensibilidad térmica; su resistencia alcanza a una variación de hasta 200 veces entre −40 °C y 40 °C. Por último, los termopares aprovechan la diferencia de potencial generada en la unión de dos metales distintos cuando hay un gradiente térmico (Rodriguez & León, 2012; Spiridonov & Ćurić, 2020).

Las unidades de medida para la temperatura (T) son en grados Celsius (°C), grados Fahrenheit (°F) o Kelvin (K). Cada escala de medida tiene sus propias características y existen ecuaciones que relacionan estas escalas.

La escala Celsius, también llamada centígrada, es la más común en la mayoría de los países y en aplicaciones científicas, está basada en dos puntos de referencia, el grado 0 °C es el punto de congelación del agua a 1 atm y los 100 °C representa el punto de ebullición del agua a 1 atm. La escala Kelvin es la unidad del Sistema Internacional (SI) para temperatura termodinámica. No utiliza el símbolo de grado y comienza en el cero absoluto (0 K), la temperatura teórica en la que cesa el movimiento molecular. Finalmente, la escala Fahrenheit es utilizada principalmente en Estados Unidos y algunos países del Caribe, sus dos puntos de referencia son los 32 °F el cual se ubica en el punto de congelación del agua y 212°F es el punto de ebullición del agua a 1 atm. Se presenta el método de conversión entre estas unidades de temperatura. Ver tabla 3.1.

Tabla 3.1. Fórmulas para la conversión de unidades de medida de temperatura.

De / A	Fórmula
°C / K	$K=°C+273.15$
K / °C	$°C=K-273.15$
°C / °F	$°F=(°C\times 9/5)+32$
°F / °C	$°C=(°F-32)\times 5/9$

Fuente: elaboración propia.

3.2.2. Presión atmosférica (p)

La presión atmosférica es una variable, definida como la fuerza por unidad de área que ejerce el peso del aire sobre una superficie dada. Su medición aporta a las predicciones meteorológicas, el análisis sinóptico y la comprensión de los sistemas dinámicos. La unidad de medida utilizada para expresarla es el hectopascal (hPa), equivalente a 100 pascales (Pa) (OMM, 2010).

La presión atmosférica se mide mediante barómetros, estos son electrónicos, de mercurio, aneroides, o hipsómetros. Este último instrumento, basado en la relación entre presión y punto de ebullición de un líquido, tiene un uso limitado.

Los barómetros de mercurio son considerados estándar por su estabilidad, sin embargo, debido a la toxicidad del mercurio, complicados de transportar y requieren correcciones manuales de lectura, su uso es limitado. Los barómetros aneroides son dispositivos compactos y portátiles que no contienen líquido, funcionan mediante una cápsula metálica parcialmente vacía que se deforma con los cambios de presión y son adecuados para aplicaciones de campo y marítimas. Derivado del anterior, los Barógrafos, registran de forma continua las mediciones en forma de gráficas (barogramas) sobre papel o soportes electrónicos, utilizados para el monitoreo prolongado tanto en tierra como en el mar (Spiridonov & Ćurić, 2020).

Finalmente, los barómetros electrónicos, que emplean sensores que convierten la presión en señales eléctricas, las cuales se procesan

digitalmente. Son los más comunes en estaciones meteorológicas automáticas. Sus módulos internos son óptimos para obtener una alta compensación térmica y estabilidad a largo plazo (Rodriguez & León, 2012; Spiridonov & Ćurić, 2021).

La OMM establece estándares para asegurar la calidad de las mediciones:

- Calibración periódica respecto a un barómetro patrón.
- Estabilidad térmica del sensor o correcciones apropiadas por temperatura.
- Protección frente a condiciones externas como radiación solar, vibraciones, viento o fluctuaciones eléctricas.
- Facilidad de lectura, con desviación típica menor a un tercio del margen de error admitido.
- Transporte seguro, especialmente en el caso de barómetros de mercurio, para evitar alteraciones en la calibración.

La unidad de medida para la presión atmosférica en el SI utiliza el Pascal, el cual es equivalente a 1 Newton/m^2. Por otro lado, en sistema cegesimal la unidad es el Baria que equivale a 1 dina/cm^2. En el SI un Pascal es igual a 10 barias. Otras unidades que se encuentran en climatología son la atmosfera (1 atm= 1006 barias = 105 Pa), el milibar (1mb = 103 baria = 100Pa) y el milímetro de mercurio (1 mm Hg = 1333.3 Pa).

3.2.3. Humedad relativa (U) en porcentaje (%)

La humedad del aire es una de las variables meteorológicas que influye directamente en la formación de nubes, la precipitación, la sensación térmica y el equilibrio energético de la atmósfera (Pierrehumbert et al. 2008). Su medición y registro es una práctica necesaria para las aplicaciones meteorológicas, agrícolas, aeronáuticas e hidrometeorológicas.

La humedad atmosférica se expresa como una Razón de mezcla (r), la cual es una relación entre la masa de vapor de agua y la masa de aire seco. La Humedad específica (q) que relaciona la masa de vapor de agua y la masa total de aire húmedo. De igual manera, es válido describir como la Presión de vapor (e'), es decir la presión parcial ejercida por el vapor de agua en una mezcla de gases. Además, se mede como la Tensión de vapor saturante (e_v o e_i), que consiste en medir la presión de vapor en equilibrio con una superficie de agua (e_v) o de hielo (e_i) (Rodriguez & León, 2012).

Por último, es posible medir la humedad del aire a través de la variable Humedad relativa (U) que consiste en la relación entre la presión de vapor actual del aire y la presión de vapor saturante a la misma temperatura, expresada como porcentaje (Pierrehumbert et al. 2008). Es la forma más común de representar la humedad en meteorología.

Los métodos más empleados para la medición de la humedad relativa son:

Método gravimétrico: mide la masa de vapor de agua contenida en una muestra de aire.

Método de condensación: determina el punto de rocío mediante la formación de condensación sobre una superficie fría.

Además, forma parte de los parámetros fundamentales en los sistemas de predicción numérica del tiempo y en los modelos de cambio climático.

Los instrumentos utilizados deben ser calibrados, instalados en condiciones representativas, y su operación debe seguir los estándares establecidos. La OMM advierte que fallas en la medición de la humedad relativa pueden ser:

"a) *La modificación de la muestra de aire, por ejemplo, por la presencia de fuentes o sumideros de calor o de vapor de agua; b) la contaminación del sensor, por ejemplo, por el polvo y los rociones de mar; c) los errores de calibración, incluida la corrección de la presión, el coeficiente de temperatura del sensor y la interfaz eléctrica; d) el tratamiento inadecuado en las fases líquida y sólida (del agua); e) el diseño inadecuado del instrumento, por ejemplo, conducción de calor por el tubo del termómetro húmedo; f) las fallas de funcionamiento, por ejemplo, no se puede alcanzar un equilibrio estable; g) los intervalos de muestreo o de establecimiento de la media, o de ambos, son inadecuados*" (OMM, 2010).

La unidad de medida de la humedad relativa es el porcentaje, donde el rango debe ir desde un 5% a 100%, con una exactitud requerida de 1% cuando los valores son altos y de un 5% cuando se tiene valores medios.

3.2.4. Precipitación

La precipitación representa cuantitativamente el aporte de agua desde la atmósfera hacia la superficie terrestre. Se define como el conjunto de

productos líquidos o sólidos provenientes de la condensación del vapor de agua en la atmósfera, que caen desde las nubes o se depositan directamente desde el aire sobre el suelo (Bezrukova & Chernokulsky, 2019). Entre los tipos más comunes de precipitación se encuentran la lluvia, nieve, granizo, rocío, escarcha, cencellada y la precipitación asociada a la niebla.

La unidad estándar para esta medición es el milímetro (mm), equivalente a un litro por metro cuadrado (l/m²), aunque también se expresa en kilogramos por metro cuadrado (kg/m²) para representar la masa de agua caída (IAEA, 2014). Las intensidades de precipitación se registran en milímetros por hora (mm/h). En el caso de la nieve, las mediciones se realizan en centímetros, con una resolución mínima de 0,2 cm, considerando valores menores como trazas (Spiridonov & Ćurić, 2021).

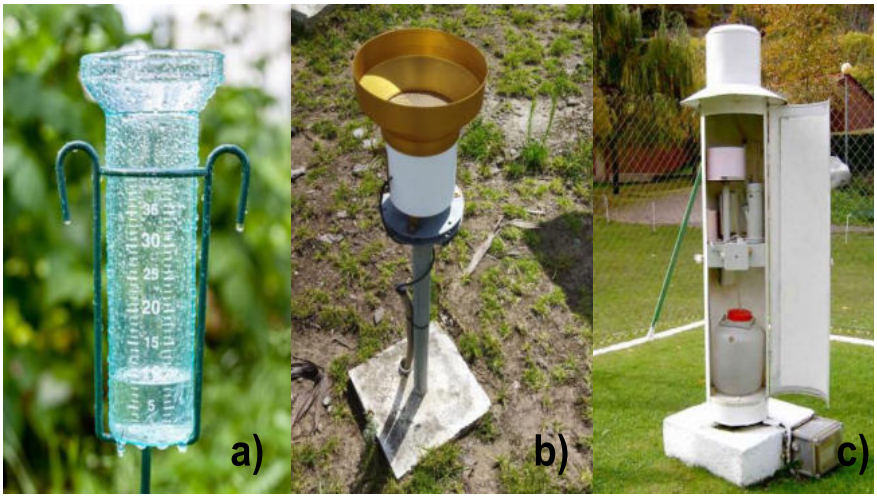


Figura 3.2. Tipos de instrumentos para medir la precipitación: a) Pluviómetro, b) Totalizador, c) Pluviógrafo.

Fuente: elaboración propia.

Las mediciones de precipitación se realizan utilizando pluviómetros cuya precisión dependen de factores como la exposición del instrumento, la influencia del viento, la topografía local y la uniformidad en el diseño de la red de observación (OMM, 2010).

3.2.5. Radiación solar

La radiación solar terrestre representa una de las variables fundamentales en el estudio del balance energético del sistema Tierra-atmósfera. La medición de la radiación facilita la comprensión de los procesos de transferencia de energía que como el análisis de componentes atmosféricos: el ozono, el vapor de agua y los aerosoles (Nadal & Muñuzuri, 2006). Además, sirven como referencia para validar datos satelitales y mejorar algoritmos de estimación de radiación. De la misma forma, permiten construir climatologías de radiación, identificar variaciones de los componentes del balance térmico y establecer relaciones precisas con otras variables meteorológicas.

La medición de la radiación se realiza mediante una red global de estaciones equipadas con instrumentos radiométricos calibrados bajo estándares internacionales, como la Referencia Radiométrica Mundial (RRM), definida a partir de comparaciones entre pirheliómetros absolutos en el Centro Radiométrico Mundial de Davos (OMM, 2010).

La energía solar recibida por unidad de superficie se mide en julios por metro cuadrado por hora ($J/m^2 \cdot h$) o en vatios por metro cuadrado (W/m^2) para flujos instantáneos.

Los Heliógrafos registran la duración del brillo solar mediante la concentración de luz en una tarjeta fotosensible. Los Piranómetros, utilizados para medir la radiación global (directa + difusa) sobre una superficie plana en un rango espectral de 300 a 3000 nm y su configuración es apta para medir únicamente la componente difusa si se protege del sol directo. Los Pirheliómetros miden exclusivamente la radiación solar directa y funcionan apuntando constantemente al Sol mediante un sistema de seguimiento. Los Piranógrafos registran la radiación solar de forma continua a lo largo del tiempo y los Pirgeómetros empleados para medir la radiación de onda larga emitida por la Tierra, ya sea directamente o mediante filtros que bloquean la radiación de onda corta (Spiridonov & Ćurić, 2020).

La OMM sugiere que los instrumentos de medición deben instalarse en lugares despejados, lejos de obstáculos como árboles o edificaciones, y con mantenimiento adecuado frente a condiciones climáticas adversas, especialmente en zonas costeras con alta humedad (OMM, 2010).

3.2.6. Velocidad y dirección del viento

La velocidad y la dirección se utiliza para estimación del potencial eólico, el análisis de la dispersión de contaminantes atmosféricos y la seguridad aeronáutica (Grogg, 2005; Rodriguez & León, 2012). El viento es una magnitud vectorial bidimensional que se describe por dos componentes: su velocidad y su dirección. Ambas características pueden presentar variabilidad alta en el tiempo, especialmente cuando se presenta fluctuaciones turbulentas conocidas como ráfagas o rachas (Spiridonov & Ćurić, 2020).

La velocidad del viento se mide utilizando anemómetros que convierten el movimiento del aire en una señal mecánica o eléctrica. El anemómetro de cazoletas, compuesto por una estructura con tres o cuatro semiesferas montadas en una cruz que rota alrededor de un eje vertical y su velocidad de rotación es proporcional a la velocidad del viento (Spiridonov & Ćurić, 2021). El anemómetro de hélice resulta ser más sensibles a variaciones de baja velocidad. Para registros continuos, se emplean anemógrafos, que almacenan electrónicamente o en papel la evolución del viento a lo largo del tiempo.

La dirección del viento se mide con una veleta, un dispositivo que gira libremente alrededor de un eje vertical por lo cual tiene la capacidad de orientarse en la dirección en la que sopla el viento. Esta dirección se expresa en grados desde el norte verdadero (de 0° a 360°) o mediante claves codificadas que agrupan rangos de 10 grados.

La altura para medir la velocidad de viento es estándar de 10 metros sobre el suelo y en un entorno libre de obstáculos que puedan interferir en la circulación del viento. Sin embargo, debido a la complejidad del relieve es común que existan errores de exposición que deben ser evaluados y corregidos (OMM, 2010). De igual manera, se recomienda informar la velocidad media del viento en intervalos de 10 minutos, y para estudios climatológicos se utilizan promedios horarios, diarios o mensuales (Pannett, 1990).

Figura 3.3. Anemómetro de cazoletas para velocidad de viento y veletas para medir su dirección.

Fuente: Martorell, (2023).

Los datos de velocidad y dirección de viento aportan en el estudio de la energía eólica disponible en una región. Así mismo, en el diseño de estructuras expuestas a cargas de viento. En el área de termodinámica es posible realizar estimaciones de la evaporación y el transporte de calor y masa en la superficie, al igual que estudiar la dinámica atmosférica local, como las brisas de valle o de mar.

La OMM establece que la incertidumbre en la medición de la velocidad y dirección del viento debe ser inferior al 5% (OMM, 2010). Los instrumentos modernos, adecuadamente calibrados y mantenidos, cumplen con estos requisitos. Sin embargo, la calidad de los datos depende en gran medida de la correcta exposición y mantenimiento del equipo.

3.3. Sistemas de referencia espacial

Los paralelos son líneas horizontales y concéntricas respecto al Ecuador, el paralelo de referencia, ubicado a 0 grados de latitud y divide al planeta en hemisferio norte y hemisferio sur. A partir de él se trazan otros paralelos

hacia el norte y sur hasta llegar a los polos, y cada uno mantiene una latitud constante, expresada en grados desde 0° hasta 90° norte o sur (Furones, 2011).

Los meridianos son líneas imaginarias verticales desde el Polo Norte al Polo Sur, el Meridiano de Greenwich, ubicado en 0 grados de longitud, y a partir de él se miden los meridianos hacia el este y el oeste hasta alcanzar los 180°(Fernandez, 2001).

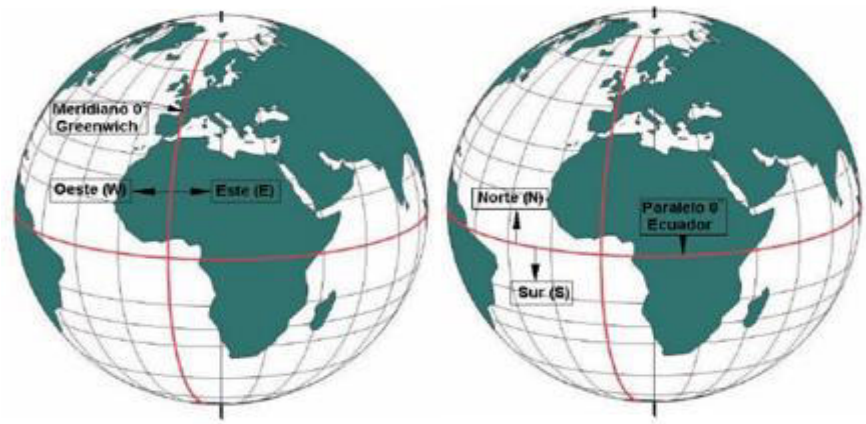


Figura 3.4. Puntos y líneas de referencia en la Tierra para los meridianos y paralelos.

Fuente: Fernandez, (2001).

El sistema de coordenadas geodésicas representa la Tierra como un elipsoide de revolución definido por su semieje mayor y menor, ajustado al eje de rotación terrestre, se calculan en función del ángulo entre el plano ecuatorial y la normal al elipsoide en un punto. Por otro lado, las coordenadas geocéntricas toman como referencia el centro del elipsoide,

que genera una diferencia con las geodésicas en la latitud (Fernández-Coppel, 2010; Furones, 2011). En cambio, el sistema de coordenadas astronómicas considera las observaciones realizadas directamente sobre la superficie terrestre y se basa en el concepto de geoide, una superficie equipotencial del campo gravitatorio terrestre (Furones, 2011).

Los sistemas de referencia geodésicos globales, como el GRS80 y el WGS84, definen la posición y orientación de un elipsoide respecto a la Tierra usando parámetros como dimensión, orientación en el espacio y posición relativa al centro terrestre (Fernández-Coppel, 2010).

La proyección cartográfica UTM, ampliamente adoptada por su utilidad estratégica, fue estandarizada a nivel mundial por el Departamento de Defensa de Estados Unidos. Esta proyección se basa en un cilindro que envuelve al globo terrestre, pero, a diferencia del modelo clásico de Mercator que es tangente al ecuador, la proyección UTM posiciona el cilindro de forma transversal al eje de rotación terrestre (Fernandez, 2001).

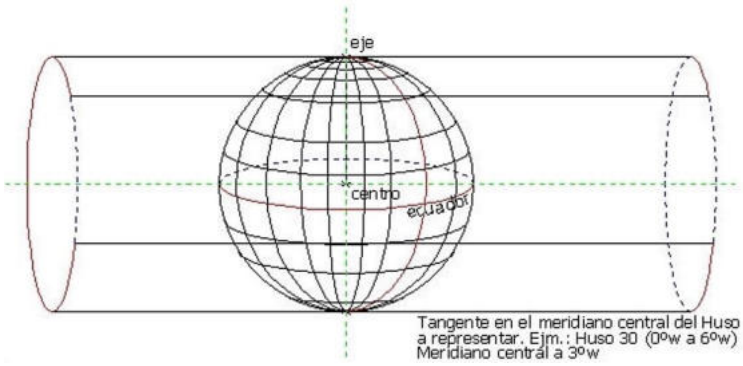


Figura 3.5. Esquema de la proyección UTM.

Fuente: Fernández-Coppel, (2010)

La Figura 3.5. indica como el sistema UTM divide la superficie terrestre en franjas verticales conocidas como husos, cada uno abarcando 6 grados de longitud. Cada huso tiene su propio meridiano central, ubicado a la mitad, es decir, a 3 grados de cada borde del huso. Este meridiano central es la única línea donde la escala es exacta (factor K = 1), y la distorsión aumenta gradualmente hacia los bordes del huso (Fernández-Coppel, 2010).

Las bases de datos adoptan distintas escalas o calendarios para registrar la variable temporal. Uno de los más comunes es el uso de días julianos, que representan una cuenta continua de días desde una fecha base (como el 1 de enero del año 4713 a.C.). Modelos meteorológicos utilizan variantes de este calendario como "calendario proleptico gregoriano", "360_day" (un año con 12 meses de 30 días), o "noleap" (años de 365 días sin años bisiestos), según las necesidades computacionales (Meyer, 1993; Steyn et al., 1981).

El intervalo temporal de agregación es importante considerar: horario, diario, mensual, trimestral o estacional. La elección del período adecuado depende del fenómeno climático que se requiere analizar:

"*Para el estudio de eventos extremos, como olas de calor o precipitaciones intensas, es preferible usar datos diarios u horarios. Para analizar variabilidad climática como El Niño, es habitual usar series mensuales o trimestrales. En climatología comparativa y proyecciones a largo plazo, las medias anuales o de periodos de 30 años son las más indicadas*" (Steyn et al., 1981).

El sistema de referencia temporal más apropiado depende del objetivo del estudio y la fuente de datos, considerando la coherencia con otros productos y la disponibilidad de observaciones de alta frecuencia.

CAPÍTULO IV

FUENTES Y FORMATOS DE DATOS METEOROLÓGICOS

4.1. Estaciones meteorológicas

Las estaciones meteorológicas son instalaciones científicas automáticas o manuales, que tienen un conjunto de instrumentos de medición y registro para diversas variables atmosféricas. Las estaciones son ubicadas, en zonas rurales, agrícolas, áreas urbana o regiones remotas.

Las estaciones meteorológicas automáticas registran mediciones mediante sensores electrónicos que envían la información en tiempo real a servidores centrales o en una memoria ubicada en la misma estación proporcionando acceso en tiempo real en diferentes zonas. El monitoreo se complicaría con estaciones manuales, que implica que un observador este permanente en el sitio, con la tecnología necesaria para transmitir la información (Sensor MKT, 2020).

El Manual de Sistema Mundial de Observación indica que "una estación en la que los instrumentos efectúan y transmiten o registran automáticamente los valores observados de variables atmosféricas, realizando en caso necesario la conversión directamente de la señal eléctrica o convirtiendo la información en una estación transcriptora" (OMM, 2010).

La instalación de estaciones automáticas debe alinearse a un proceso de planificación estratégica, basada en las necesidades específicas del consumidor. Para ello, es fundamental establecer un diálogo con los

interesados, identificar sus requerimientos y diseñar soluciones técnicas adecuadas (Furones, 2011).

Existe diferencia entre los sistemas automáticos y manuales debido a la sensibilidad de los sensores, la frecuencia de muestreo, el tiempo de respuesta y el registro de los datos. El proceso de obtención de datos involucra desde la instalación y mantenimiento del equipo hasta la manera en que los investigadores acceden y usan esa información (Eyring et al. 2016).

No obstante, las estaciones automáticas representan un avance significativo hacia la estandarización de los datos meteorológicos, al eliminar fuentes comunes de error como la subjetividad del observador y los errores de digitación. La automatización no implica ausencia de supervisión; por el contrario, estudiar los cambios en los registros de datos para informar adecuadamente a los usuarios sobre sus alcances, limitaciones y beneficios.

4.1.1. Limitaciones de las estaciones meteorológicos

La limitación de cobertura espacial, cada estación meteorológica mide únicamente las condiciones de un punto, no siempre se logra representar con precisión las condiciones de áreas más amplias o con alta variabilidad topográfica.

La calidad y continuidad de los datos son afectadas por factores técnicos y operativos. Las estaciones automáticas requieren mantenimiento periódico y calibración de sensores para evitar fallos de los instrumentos de medición.

Por otro lado, las estaciones convencionales dependen de la observación humana, que introduce subjetividad o errores de registro.

Además, las condiciones de exposición del sitio de instalación (presencia de obstáculos, cambios en el uso del suelo, interferencias electromagnéticas, etc.) influyen en la representatividad de los datos. De la misma forma, el retraso en la transmisión de datos o la pérdida de conectividad en lugares remotos afectando la disponibilidad en tiempo real de la información meteorológica.

4.1.2. Red de estaciones ESPOCH-INAMHI

La Estación Meteorológica ubicada en la Facultad de Ciencias de la Escuela Superior Politécnica de Chimborazo (ESPOCH) cumple un rol fundamental en el ámbito de la investigación científica, al proporcionar datos meteorológicos continuos desde el 17 de octubre de 2014, manteniendo un monitoreo constante de registro de variables atmosféricas esenciales para el desarrollo de estudios climáticos y proyectos académicos. Ver figura 4.1.

Figura 4.1. Estación meteorológica automática ubicada en Riobamba, Ecuador.

Fuente: GEAA, (2022).

La colaboración entre la ESPOCH y el Instituto Nacional de Meteorología e Hidrología (INAMHI), que se ha extendido por más de cuatro décadas. La estación meteorológica manual del INAMHI, funciona dentro de las instalaciones de la ESPOCH en Riobamba.

En la ejecución de proyectos conjuntos entre la ESPOCH y el Instituto Nacional de Eficiencia Energética y Energías Renovables (INER), se instaló una red de once estaciones meteorológicas automáticas Vaisala distribuidas estratégicamente en la provincia de Chimborazo desde el año 2013 (ver Figura 4.2.). Esta red proporciona información meteorológica precisa, confiable y actualizada para la región centro del país. (GEAA, 2022).

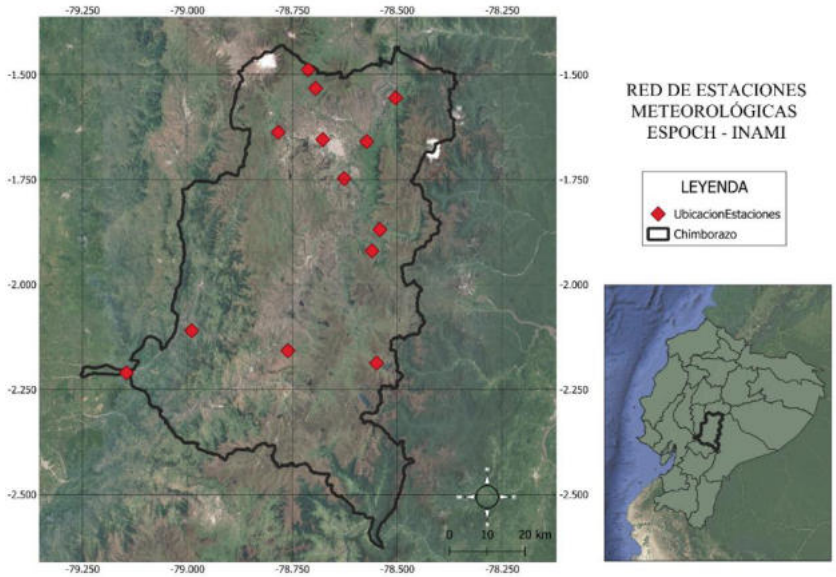


Figura 4.2. Mapa con la ubicación de la red de estaciones meteorológicas ESPOCH-INAMHI.

Fuente: elaboración propia.

Los datos meteorológicos obtenidos a través de esta red están normados por estándares de la Organización Meteorológica Mundial (OMM), que recomienda el uso del formato CSV (*Comma-Separated Values* (Pazmiño Solís, 2021).

4.2. Bases de datos

Las bases de datos representan una herramienta que almacena información a partir de distintas fuentes como: estaciones meteorológicas, sensores remotos, satélites, modelos numéricos y redes de observación internacional.

La información generada concede una comprensión más profunda del comportamiento atmosférico en escalas espaciales.

4.2.1. Datos CMIP-6

Coupled Model Intercomparison Project Phase 6 (CMIP6) es la sexta fase del proyecto de modelos Acoplados Globales, coordinado por el Programa Mundial de Investigaciones Climáticas (WCRP), el cual tiene como objetivo principal mejorar la compresión del cambio climático mediante la comparación de modelos climáticos desarrollado por múltiples centros de investigación a nivel mundial (PCMDI, 2019). El CMIP6 complementa los vacíos detectados en versiones, esta información ha sido utilizada ampliamente por el **IPCC (Panel Intergubernamental sobre Cambio Climático)** para sus evaluaciones científicas (O'Neill et al., 2016).

En CMIP6 se implementa la identificación y corrección de errores sistemáticos presentes en los modelos, la refinación de la estimación de los forzamientos radiactivos tanto en simulaciones de cambio climático y se adiciona la contabilidad más precisa del impacto de agentes forzantes a corto plazo y del cambio en el uso del suelo (Eyring et al., 2016b; O'Neill et al., 2016). De la misma forma, se considera el comportamiento futuro del desarrollo humano mediante la incorporación de escenarios Socioeconómicos Compartidos (*Shared Socioeconomic Pathways*, SSP) los cuales exploran los posibles escenarios futuros del clima tomando en cuenta variables como crecimiento económico, políticas de mitigación, urbanización, y cambio tecnológico (Eyring et al., 2016).

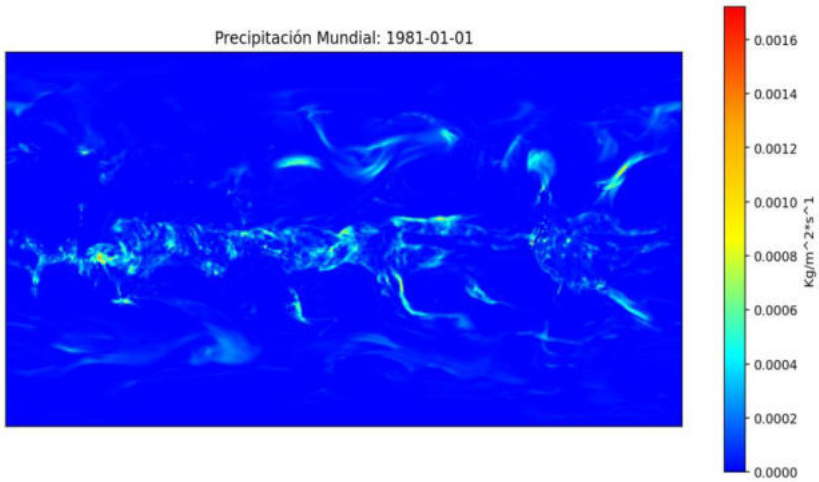


Figura 4.3. Datos históricos a nivel mundial de precipitación CMIP6-CNRM.

Fuente: elaboración propia.

En la gráfica se observa el comportamiento de la precipitación el día 01 de enero de 1981 a escala mundial.

4.2.2. Datos CHIRTS

Los CHIRtS-daily (*Climate Hazards Group InfraRed Temperature with Station data*) son un conjunto de datos de temperatura diaria de alta resolución diseñados para monitorear las condiciones térmicas extremas, y complementar los esfuerzos de monitoreo climático a nivel mundial, especialmente en regiones con escasos datos in situ.

Esta base de datos contiene información desde 1981 hasta 2016, en un rango de 60°S – 70°N, con una resolución espacial de 0.05°. La metodología de CHIRtS-daily combina datos de temperatura máxima (Tmax) y temperatura

mínima (Tmin) utilizando un modelo de desagregación que integra datos mensuales del sistema de reanálisis ERA5 y observaciones de estaciones climáticas, que facilitan el análisis de las tendencias de temperaturas extremas y monitorear eventos climáticos (Verdin et al., 2020).

CHIRTS proporciona estimaciones diarias en áreas donde los datos son distribuidos aleatoriamente, utilizando observaciones térmicas infrarrojas con datos de estaciones. Un análisis detallado de las condiciones climáticas se realiza con información CHIRTS-daily.

Para la generación de la información de CHIRTS-daily, se considera las temperaturas diarias de ERA5, sin embargo, hay una diferencia en las escalas espaciales usados por CHIRTSmax (0.05° x 0.05°) y ERA5 (0.25° x 0.25°) que debe considerarse (Verdin et al., 2020). Para resolver este problema se aplica el método de interpolación bilineal en el Lenguaje Interactivo de Datos (IDL) con el comando CONGRID lo cual ajusta la resolución de los datos de ERA5 para que coincidan con CHIRTSmax, después se obtiene Tmax y Tmin Diarios de CHIRTS tomando en cuenta a T como el total de días registrados hasta la fecha (CHC 2016).

4.2.3. Datos CHIRPS

Los CHIRPS (por sus siglas en inglés *Climate Hazards Group InfraRed Precipitation with Station data*) es una herramienta ambiental diseñada con el fin de monitorear las precipitaciones y condiciones extremas y fue desarrollado para apoyar la FEWS NET, por sus siglas en inglés de *Famine Early Warning Systems Network* de la USAID (*United States Agency for International Development*).

El conjunto de datos de precipitación diaria abarca un periodo desde 1981 hasta la actualidad en un rango de 50°S −50°N. El modelo de simulación CHPclim, considera imágenes satelitales de alta resolución (0.05°) junto con los datos de precipitaciones in situ para la generación de series temporales que se utilizan para monitorear y analizar las tendencias de sequias estacionales (Katsanos et al. 2016).

Una de sus características es la una interpolación inteligente que trabaja con anomalías de una climatología de alta resolución que combina los datos de estaciones y estimaciones de *Cold Cloud Duration* (CCD) para completar la información de datos de precipitación, permitiendo identificar patrones de comportamiento de eventos climáticos y estimar el comportamiento de la precipitación a largo plazo (CHC, 2016).

La metodología de interpolación utilizada para generar la precipitación de alta resolución combina información de estaciones meteorológicas con diversos campos satelitales, se aplica una regresión local para cada celda de una rejilla espacial, incorporando como variables explicativas la latitud, longitud, elevación, pendiente y además alguna variable derivada de datos satelitales. Esta estimación se realiza de forma localizada, ajustándose a las características particulares de cada celda. Luego, se divide el conjunto de datos en 73 recuadros geográficos, representando zonas importantes globales como el sur de Sudamérica, el Caribe o Europa occidental, dentro de cada uno, se ajustan modelos de regresión que varían en complejidad dependiendo de la densidad de estaciones meteorológicas en la zona. Posteriormente, se aplican modelos para estimar los valores mensuales de precipitación utilizando como referencia los datos climáticos de la FAO. La

diferencia entre los valores estimados y las observaciones se interpola espacialmente empleando una técnica de ponderación por distancia inversa. Finalmente, se introducen correcciones con base en los datos mensuales de precipitación correspondientes al periodo 1980-2009, extraídos de la red Global *Historical Climate Network* (GHCN). Con esta información se identifican sesgos regionales que luego se interpolan y se aplican a la estimación preliminar, dando como resultado el comportamiento final de precipitación conocido como CHPclim (Katsanos et al. 2016; CHC 2016).

Datos de Validación: Se utilizaron datos de estaciones meteorológicas obtenidos de agencias nacionales, que no estaban incluidos en los conjuntos de datos FAO o GHCN utilizados para la construcción del CHPclim. Para cada estación, se extrajeron los valores correspondientes del CHPclim, CRU y WorldClim, luego se calcularon estadísticas de validación mensual y finalmente se promediaron a lo largo de los 12 meses. En general, los tres conjuntos de datos climáticos (CHPclim, CRU y WorldClim) capturan bien los promedios generales en Colombia, Etiopía y México. Sin embargo, en Afganistán y el Sahel, tanto CRU como WorldClim muestran sesgos grandes ($\geq \pm 15$ %) posiblemente debido son regiones con escasez de datos (Katsanos et al. 2016; CHC 2016a).

Los datos CHIRPS son de acceso gratuita, están disponibles al público en general sin restricciones ni requisitos acceso, las plataformas que guardan los repositorios de descarga de los datos CHIRPS se describen:

Sitio web del *Climate Hazards Center* de la Universidad de California, Santa Bárbara: https://www.chc.ucsb.edu/data/chirps

Plataforma de *Google Earth Engine*: https://earthengine.google.com/

Repositorio de datos de la NASA:

https://disc.gsfc.nasa.gov/datasets/CHIRPS_V2.0/summary

Los datos CHIRPS están disponibles en varios formatos, entre ellos esta los formatos GeoTIFF, NetCDF y ASCII, además de estar disponible en varias resoluciones espaciales como 0.05°, 0.25° y 0.5°.

CAPÍTULO V

INTRODUCCIÓN A PYTHON

5.1. ¿Qué es Python y por qué usarlo en meteorología?

Python es un lenguaje de programación moderno, de alto nivel y multiparadigma, esto significa que acepta diferentes estilos de programación, como la secuencial, POO (Programación orientada a objetos), funcional, distribuida, paralela, entre otros. Su sintaxis es sencilla e intuitiva por lo que es fácil de enseñar y aprender, por esa razón, se considera a Python como el primer lenguaje de programación para un principiante, para luego adentrarse en el mundo de la programación.

Figura 5.1. Logotipo de Python.

Fuente: Python Software Foundation, (2025).

Python es un lenguaje interpretado, eso quiere decir que no requiere compilación previa: cada instrucción se ejecuta directamente, lo que en la práctica significa un desarrollo más ágil y un proceso de prueba-error más rápido (Valverde et al. 2023). Una de las mayores fortalezas de Python radica en su tipificación dinámica, es decir, las variables no requieren una declaración previa y son capaces de almacenar distintos tipos de datos a lo largo del código, además, cuenta con estructuras de datos integradas muy potentes, como listas, diccionarios y arreglos multidimensionales, dando lugar a una manipulación eficiente y flexible de grandes volúmenes de información (J. W.-B. Lin, 2012).

Al ser un lenguaje con una curva de aprendizaje alta se ha convertido en el aliado para los científicos en el campo de la meteorología y las ciencias de la atmósfera, Python es una herramienta que ha logrado integrar en una sola plataforma todo un ecosistema de bibliotecas científicas especializadas, como NumPy, SciPy, Pandas, Matplotlib, Xarray y MetPy, entre muchas otras, permitiendo el manejo, visualización, análisis estadístico, modelado de grandes conjuntos de datos, incluyendo el procesamiento de datos geoespaciales, de la misma manera, tiene la capacidad de integrarse con otros lenguajes como Fortran o C, facilitando la interoperabilidad con modelos numéricos meteorológicos existentes (J. W. B. Lin, 2012).

El lenguaje Python es de código abierto y no depende de un único proveedor, su evolución ha impulsado tanto a la academia como a la industria. Instituciones como el NCAR (*National Center for Atmospheric Research*) y el LLNL (*Lawrence Livermore National Laboratory*) la han

adoptado como una herramienta central para el análisis de modelos climáticos y simulaciones complejas (J. W. B. Lin, 2012).

En contraste, Python presenta algunas desventajas como: compilación lenta de código, no tener un diseño estático o con punteros de memoria, sus scripts suelen requerir de más memoria que otros lenguajes. En otros contextos, debido al *Global Interpreter Lock* (GIL) no se puede ejecutar código de manera paralela, es decir en distintos hilos desaprovechando el uso de CPUs de manera simultánea[2]. Finalmente, una desventaja es que algunas librerías científicas implementadas en FORTRAN resultan más eficientes que en Python, aunque carecen de buena documentación (J. W.-B. Lin, 2012), no está optimizado originalmente para desarrollar interfaces gráficas de usuario (GUI) complejas.

Se presentan algunas citas de científicos que muestran sus argumentos para impulsar el uso de Python como la herramienta del mañana para el avance en meteorología y las ciencias de la tierra:

Las ciencias de la tierra han contado tradicionalmente con herramientas computacionales poderosas. Python permite escribir código más comprensible y con menos errores lo que facilita el aprovechamiento de recursos computacionales recientes y el acceso a herramientas desarrolladas en otros sectores. Esta combinación de claridad y potencia posiciona a

[2] En la actualidad existen maneras para implementar procesos paralelos utilizando bibliotecas externas, sin embargo, sigue siendo una desventaja ya que agregar estos códigos complica la implementación.

Python como la próxima gran tendencia en la computación para las ciencias de la tierra. (J. W. B. Lin, 2012).

Python cuenta con una comunidad activa que impulsa su uso en diversas áreas científicas, como la meteorología (Perkel, 2015).

La visualización es una parte integral del análisis de datos meteorológicos... La representación gráfica de los datos para convertir en conocimiento la información (Khisanova, 2022).

Python es ahora una plataforma de integración robusta para todo tipo de trabajos de ciencias atmosféricas, desde análisis de datos hasta computación distribuida y desde interfaces gráficas de usuario hasta sistemas de información geográfica (Oktavia Suhyani, 2025).

5.2. Instalación de Anaconda y primeros pasos con Spyder

Anaconda es un software de distribución libre y de código abierto de los lenguajes de programación Python y R, diseñada específicamente para la investigación, ciencia de datos, análisis científico y la ciencia. Esta plataforma incluye una gran variedad de herramientas llamados IDEs (por sus siglas en inglés *Integrated Development Enviroments)*, que son plataformas o aplicaciones que facilitan escribir código, editarlo, depurarlo y ejecutarlo (Rolon-Mérette et al., 2020). Los entornos de trabajo que vienen preinstalados destacan Spyder y Jupyter Notebook por su interfaz fácil de manejar para programar y ejecutar código.

Anaconda incluye el gestor de paquetes `conda`, que instala, actualiza y administra fácilmente bibliotecas y entornos virtuales sin conflictos de dependencias (Anaconca Inc, 2025).

5.2.1. Instalación en Windows

El proceso de instalación de Anaconda es simple, se muestra cómo instalar la última versión de Anaconda en Windows [3] según la documentación proporcionada en su página oficial.

1. Acceder a su página oficial anaconda.org y dirigirse a la opción "*Download Anaconda"* ubicado en el menú de opciones de la parte superior de la ventana.

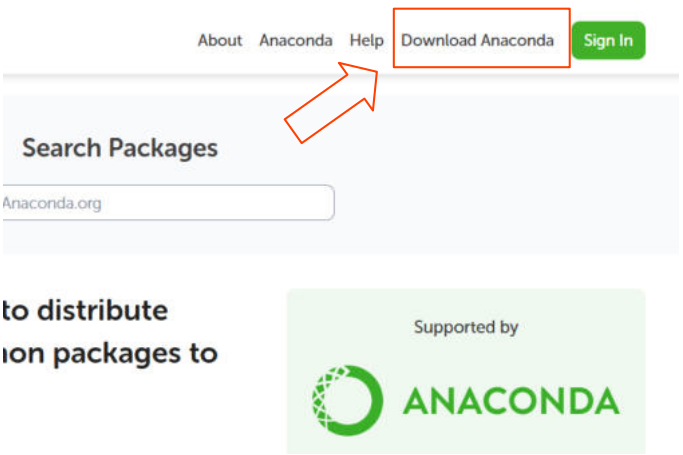


Figura 5.2. Pantalla inicial de la página anaconda.org.

Fuente: elaboración propia.

[3] Hasta la elaboración del presente libro la última versión existente es Anaconda3-2025.12-1

2. Antes de la pantalla de descarga existe la opción de suscribirse a anaconda.org de manera gratuita para recibir noticias de las últimas novedades, herramientas o actualizaciones. No es necesario suscribirse, se puede dar *click* en "*Skip registration"*.

3. En la ventana de descargas seleccionar la distribución que se requiere instalar, el sistema operativo y su arquitectura. En este caso para Windows y la descarga de un archivo exe comenzará automáticamente.

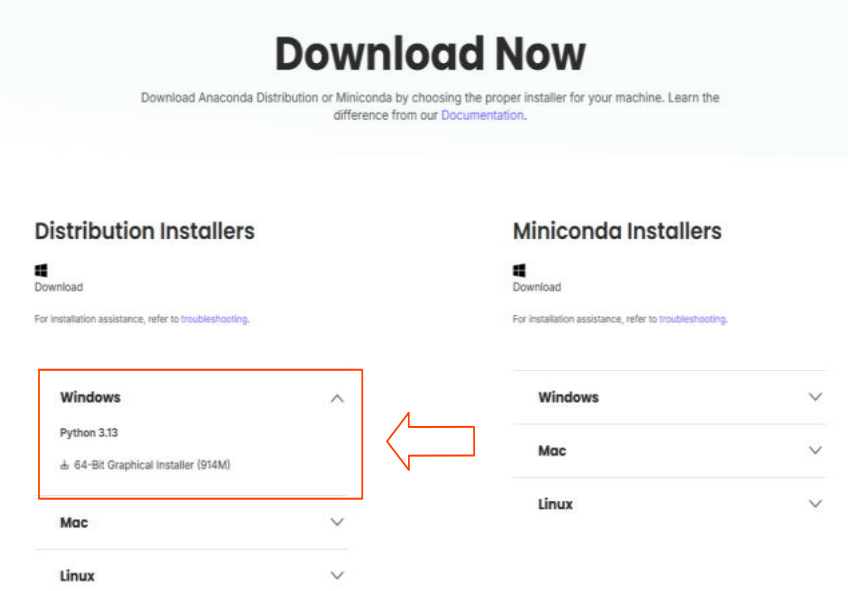


Figura 5.3. Página de descarga de anaconda.

Fuente: elaboración propia.

4. Cuando la descarga haya finalizado, abrir el archivo ejecutable y se mostrará la ventana de instalación de la distribución de Anaconda, en

el cual se debe seleccionar "*next*". Luego, aceptar los términos y condiciones con un *click* en "*I agree*". A continuación, el programa pregunta el tipo de instalación para lo cual es recomendable escoger "Just Me". En la ventana de localización de la instalación se deja tal cual está por defecto y *click* en "*Next*".

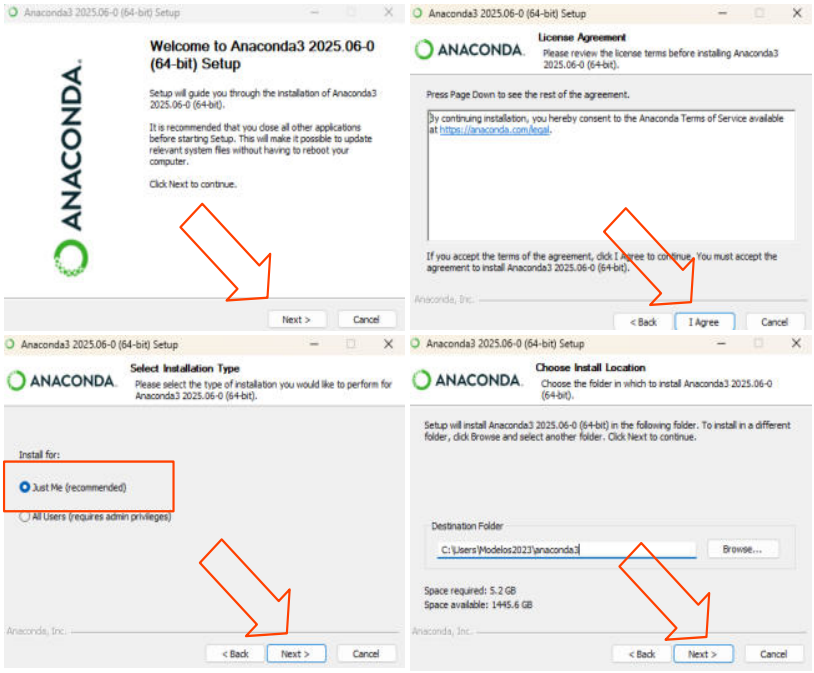


Figura 5.4. Capturas de Pantalla del instalador de Anaconda.

Fuente: elaboración propia.

5. Finalmente, se muestra la página de instalación en donde se asegura que esté seleccionada la primera opción y después dar *click* en "*Install*". La instalación de Anaconda empezará de manera automática.

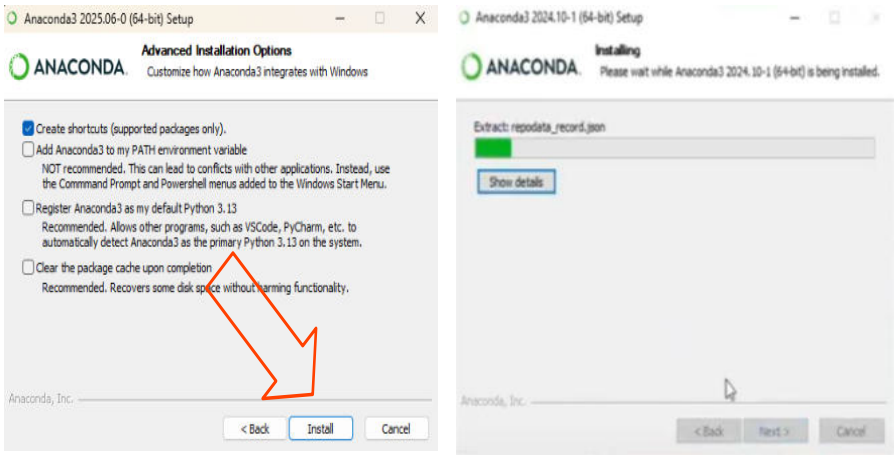


Figura 5.5. Capturas de pantalla de la sección final de instalación.

Fuente: elaboración propia

Finalizada con éxito la instalación, se ejecuta Anaconda desde el escritorio o desde el buscador de Windows. Al ejecutar Anaconda se visualiza el navegador, en el cual se encuentra varias herramientas preinstaladas que ayudan en el análisis de datos y la edición de código, tales como PyCharm, Anaconda AI Navigator, JupyterLab, Jupyter Notebook, Spyder, Oracle, entre otros, de igual manera, se muestra la opción de instalar otras herramientas para el análisis científico con tan solo dar un "*click*", por ejemplo, RStudio.

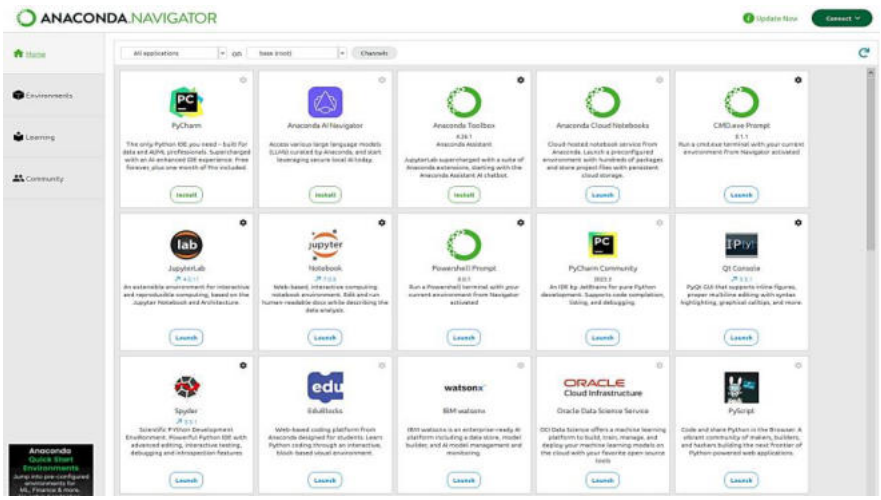


Figura 5.6. Pantalla inicial del navegador de Anaconda.

Fuente: Elaboración propia

5.2.2. Spyder

Spyder es un entorno de desarrollo integrado (IDE), especialmente pensado para el análisis de datos. Este entorno fue desarrollado originalmente por Pierre Raybaut en 2008 como un software de código abierto y desde entonces ha sido gestionado y mejorado por una comunidad activa de desarrolladores, aunque actualmente es gestionado por la organización Spyder-IDE, con el respaldo de instituciones como ***Quansight*** (Spyder-IDE, 2024). Ofrece una interfaz intuitiva y con herramientas que realizan ciencia de datos de manera más eficiente al incluir un editor de scripts (Rolon-Mérette et al., 2020). Los desarrolladores la describen como un IDE de programación interactiva integrada, diseñado para los analistas de datos, científicos e ingenieros para realizar experimentación, retroalimentación

rápida y ciclos de iteración cortos al programar (Spyder-IDE, 2024). Spyder cuenta con utilidades avanzadas para la depuración y revisión del código.

"*Spyder ofrece un entorno de programación que combina la facilidad de uso de Jupyter con características avanzadas presentes en editores como PyCharm o VSCode, todo en una sola plataforma... facilita la transición desde scripts simples hacia módulos y paquetes estructurados y reutilizables. Su instalación es sencilla y rápida, gracias a instaladores que mantienen el programa siempre actualizado*." (Spyder-IDE, 2024).

Figura 5.7. Ventajas del uso del IDE Spyder.

Fuente: Spyder-IDE, (2024).

En la ventana inicial de Spyder se encuentran tres primeros paneles esenciales, en el lado izquierdo está el "Editor", en el lado derecho superior el "*Variable Explore"* (Explorador de variables) y el "*IPython Console"* en la parte inferior. De estos tres, se destaca la consola IPython el cual es un entorno interactivo capaz de escribir y ejecutar códigos de manera inmediata, y visualizar los resultados al instante (VanderPlas, 2016). La forma de ejecución de la consola IPython es similar al ejecutar por celdas

de *Jupyter notebook* porque en general tiene las mismas características como visualización de los valores de las variables, importación con librerías, visualización de gráficas y el historial de comandos.

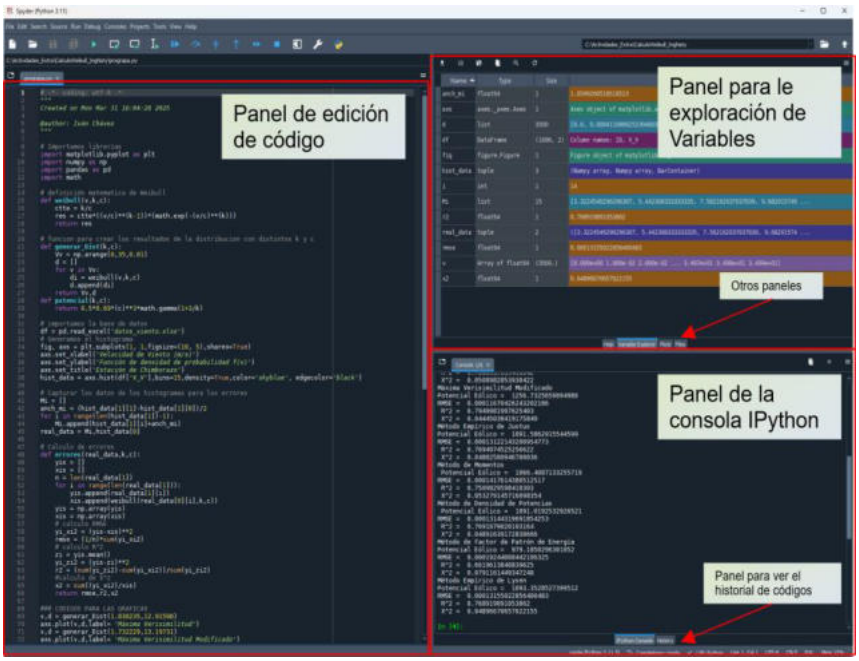


Figura 5.8. Pantalla inicial de Spyder.

Fuente: elaboración propia.

De igual manera, se destaca el panel de “History”, el cual posibilita la revisión de los comandos y archivos script .py que se han ejecutado desde la terminal, para retomar el trabajo que se realizó. El panel “Help”, que facilita la consulta de la sintaxis y funcionalidades de algún código de manera rápida. El panel “Plots” es una sección de la ventana donde el usuario tiene la capacidad de visualizar todas las gráficas que se generan en el código del script o desde la consola, se destaca porque existe la opción

de desacoplarlo de la ventana principal para una mejor visualización además de integrar una barra de herramientas con opciones como guardar, borrar, ampliar imagen, etc. El explorador de archivos llamado “Files” desde el cual se importa al editor un script o corroborar si un archivo generado se exportó y almacenó correctamente. Por último, en la barra de herramientas en la opción View > Panes, se observa que existen varios paneles extras los cuales se integran en el IDE según los requerimientos y preferencias del usuario.

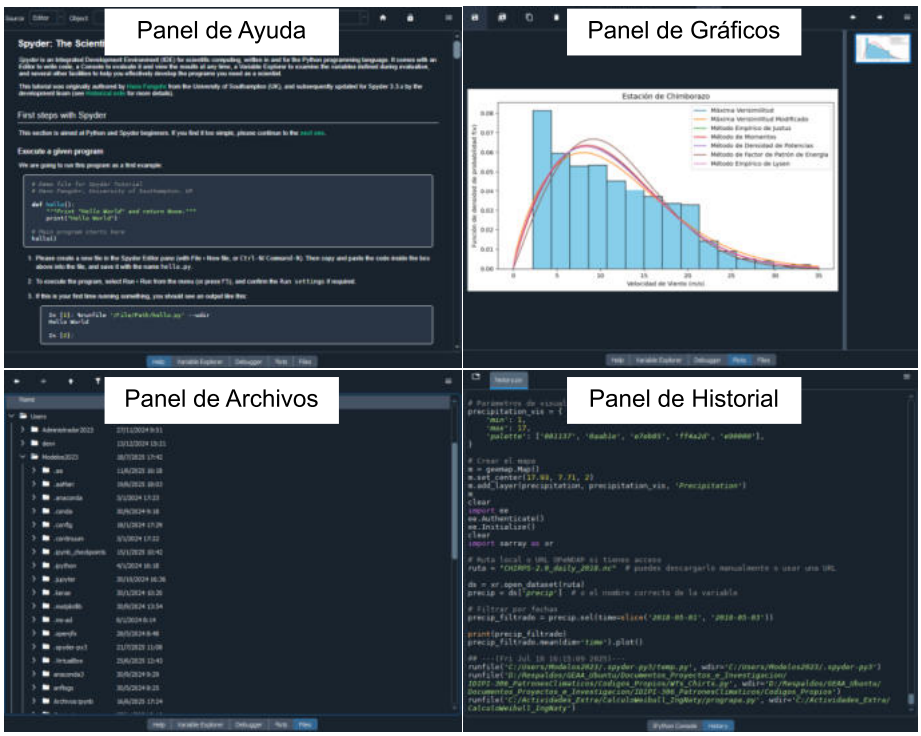


Figura 5.9. Otros paneles de la interfaz principal de Spyder.

Fuente: elaboración propia.

5.2.3. Conda

El gestor de paquetes “Conda”, que revisa de manera automática las dependencias y entornos de desarrollo. Lanzado por la comunidad de Python en 2012, con el objetivo de facilitar la instalación de bibliotecas que requieren compilación de código, se destaca por su multiplataforma y multilenguaje que permite instalar, actualizar y desinstalar paquetes de software escritos no solo en Python, sino también en R, Ruby, C/C++, entre otros (Bertrand & Harrisson, 2019).

Una fortaleza es su capacidad para gestionar entornos de forma limpia y reproducible, es decir, definir diferentes entornos virtuales para instalar versiones específicas de paquetes y bibliotecas según los requerimientos, evitando conflictos y facilitando el trabajo simultáneo en múltiples proyectos dentro de un mismo computador facilitando el trabajo con versiones específicas de software para garantizar la reproducibilidad de los resultados, también se destaca en contextos donde existen múltiples módulos que deben compilarse juntos, como en el caso de EPICS (Experimental Physics and Industrial Control System). De la misma manera, es posible generar binarios portables que son ejecutables en cualquier distribución moderna de Linux, eliminando los problemas comunes que surgen al actualizar un sistema operativo.

Cada paquete conda es un archivo comprimido (.tar.bz2 o .conda) que contiene los archivos a instalar y metadatos organizados en un directorio de información e incluyen desde bibliotecas del sistema hasta archivos de texto y binarios (Bertrand & Harrisson, 2019). Su diseño flexible, junto con el soporte para uso en contenedores como Docker o en servidores privados.

Asimismo, ofrece un repositorio de paquetes con un enfoque centralizado, es decir que existen canales mantenedores que son fiables y albergan decenas de miles de paquetes, garantizando una instalación sencilla. El canal predeterminado, que viene por defecto con Anaconda, incluye miles de bibliotecas de ciencia de datos, aprendizaje automático e inteligencia artificial. Está disponible para Windows, macOS y Linux, y garantiza actualizaciones regulares y compatibilidad multiplataforma. Otro canal muy usado en la comunidad de desarrolladores es *conda-forge,* el cual se mantiene siempre actualizado, disponible en todas las plataformas y cualquier persona puede contribuir. De igual manera, existe *bioconda* que es un canal de la comunidad, pero enfocado exclusivamente a librerías de biomédica (Anaconda Team, 2025).

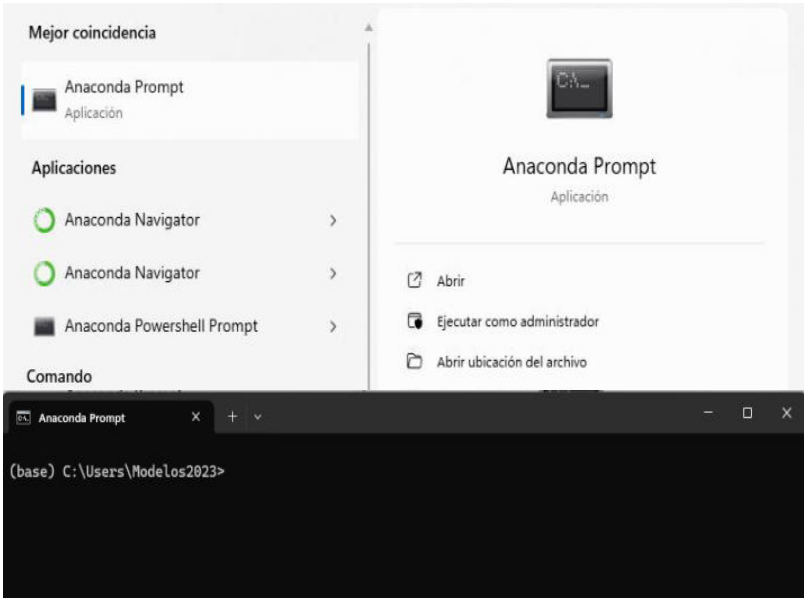


Figura 5.10. Cómo ejecutar el ejecutor de comandos de Anaconda.

Fuente: elaboración propia.

Los canales de repositorios se usan desde el gestor conda mediante permisos de acceso que son implementados en el entorno de trabajo. Para ello, se utilizan comandos desde el "*Anaconda Promp",* el cual es accesible desde el buscador de Windows o Linux, los principales comandos se describen a continuación:

Tabla 5.1. Principales comandos útiles desde el prompt de anaconda para la gestión de entornos virtuales.

Comando	Descripción	Ejemplo	Interpretación
Gestión de entornos			
conda create	Crea un nuevo entorno con el nombre especificado.	*conda create Prueba*	Crea un entorno virtual llamado "Prueba", el cual viene instalado predeterminadamente la versión más actual de Python.
conda activate	Activa el entorno para comenzar a trabajar en él.	*conda activate Prueba*	Activa en la ventana de comandos el entorno virtual llamado "Prueba".
conda deactivate	Desactiva el entorno actual.	*conda deactivate*	Desactiva de la ventana de comandos el entorno en el que se está trabajando y se vuelve a la base.
conda env list	Muestra una lista de los entornos disponibles.	*conda env list*	Aparece una lista de los entornos virtuales que se han creado en el computador desde Anaconda Prompt:
conda remove --all	Elimina completamente un entorno.	*conda remove Prueba --all*	Elimina del computador toda la información del entorno de trabajo llamado Prueba.
conda env export ->	Exporta el entorno actual a un archivo. yml (útil	*conda env export -> Prueba.yml*	Crea en el computador un archivo yml del

	para compartir o reproducir entornos).		entorno virtual de trabajo llamado Prueba.
conda env créate -f	Crea un entorno a partir de un archivo yml.	*conda env créate -f Prueba.yml*	Genera un entorno virtual de trabajo en computador con todas las configuraciones, paquetes y dependencias guardadas en el archivo Prueba.yml
Gestión de paquetes			
conda install	Instala un paquete en el entorno que se está trabajando. Por defecto, se instalará la versión más reciente que no tenga errores de dependencias.	*conda install pandas*	Instala en el entorno de trabajo actual el paquete llamado pandas (el más actual y sin errores de dependencias)
conda update	Actualiza un paquete a su versión más reciente.	*conda update pandas*	Realiza la actualización de la versión más actual del paquete pandas
conda remove	Elimina un paquete del entorno actual.	*conda remove pandas*	Elimina la librería panda del entorno de trabajo
conda list	Genera una lista con todos los paquetes instalados en el entorno.	*conda list*	Se genera en la ventana de comando una lista de librerías, junto con la versión instalada en el entorno de trabajo.
Gestión de canales			
conda config --add channels	Agrega un canal de paquetes	*Conda config --add channels conda-forge*	Se agrega a la anaconda Prompt el canal de conda-forge para poder gestionar los paquetes en este repositorio.

Fuente: elaboración propia

El IDE de Spyder es ejecutable de manera directa desde el *prompt* de Anaconda desde un entorno virtual creado. En primer lugar, se debe activar el entorno en el que se quiere instalar con el comando "*conda activate nombre_del_entorno*", luego se instala el IDE usando "*conda install spyder*". Cada vez que se necesite ejecutar *spyder* desde el *prompt* se logra hacer primero activando el entorno virtual y luego escribiendo *spyder*, pasados unos segundos *spyder* se inicializará.

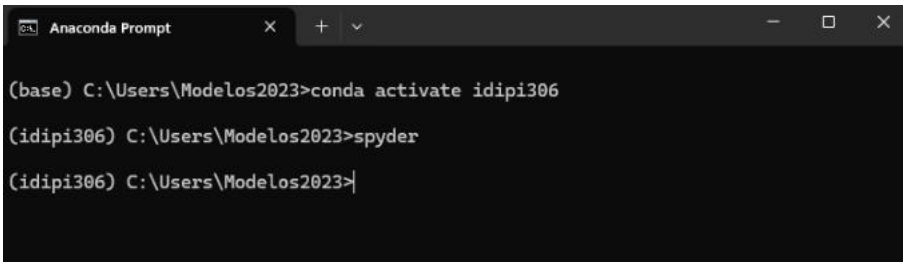


Figura 5.11. Comandos para inicializar Spyder desde la terminal de Anaconda.

Fuente: elaboración propia.

5.3. Sintaxis y control de flujo

En Python, la indentación es como el GPS de tu código: le indica al intérprete exactamente dónde empieza y termina cada bloque de instrucciones, usando espacios o tabulaciones en vez de llaves como hacen otros lenguajes (van Rossum, 2001). Lo recomendado es emplear cuatro espacios por nivel y, sobre todo, no perder la coherencia. Esta consistencia no solo hace que el código luzca más ordenado y fácil de leer, además, es esencial para que funcione (van Rossum, 2001).

Los comentarios, que empiezan con #, son como notas personales al margen. No sirven para repetir lo obvio del código, sino para explicar el "por qué" o el "cómo" detrás de una decisión. Esto ayuda a cualquiera, incluyendo a ti mismo en el futuro a entender mejor el código y sus razones (van Rossum, 2001).

```
if x > 0:

print("Número positivo")  # Esta línea está indentada dentro del bloque "if"
```

Un aspecto importante de la sintaxis de Python es su cualidad de asignar variables de manera dinámica, es decir, no hace falta que se especifique él tipo de dato que se guarda en una variable, porque el programa lo descubre mientras trabaja. Por ejemplo, si asignas *x = 10*, el sistema entiende que x es un número entero; y si después cambias a *x = "hola"*, simplemente ajusta el tipo sin problema. Las variables actúan más como etiquetas que son intercambiables para registrar a distintos objetos según lo que se requiera (Lutz, 2013).

Tabla 5.2. Tipos de Variables en Python.

Tipo de variable	**Ejemplo**	**Memoria aproximada (bytes)**	**Descripción breve**
int	*x = 42*	28	Números enteros de precisión ilimitada.
float	*x = 3.14*	24	Números reales de punto flotante con doble precisión.
str	*x = "hola"*	49 + 1 byte por carácter	Cadenas de texto Unicode.
bool	*x = True*	28	Valores booleanos: True o False.

list	*x = [1, 2]*	64 + 8 bytes por elemento	Colección ordenada y mutable de objetos.
dict	*x = {'a':1}*	240+	Estructura clave-valor, dinámica y mutable.
NoneType	*x = None*	16	Representa la ausencia de valor.

Fuente: elaboración propia

Tabla 5.3. Tipos de operadores en Python.

Tipo de operador	**Operador**	**Descripción**	**Ejemplo**
Aritmético	+	Suma	*3 + 2 → 5*
	-	Resta	*5 - 1 → 4*
	*	Multiplicación	*2 * 4 → 8*
	/	División (resultado flotante)	*7 / 2 → 3.5*
	//	División entera	*7 // 2 → 3*
	%	Módulo (residuo)	*7 % 2 → 1*
	**	Potenciación	*2 ** 3 → 8*
Comparación	==	Igual a	*5 == 5 → True*
	!=	Distinto de	*5 != 3 → True*
	>	Mayor que	*7 > 3 → True*
	<	Menor que	*2 < 4 → True*
	>=	Mayor o igual que	*5 >= 5 → True*
	<=	Menor o igual que	*3 <= 4 → True*
Lógicos	*And*	Devuelve *True* si ambas condiciones son verdaderas	*True and False → False*
	Or	Devuelve *True* si al menos una condición es verdadera	*True or False → True*
	Not	Invierte el valor lógico	*not True → False*
De pertenencia	*In*	Retorna *True* si un elemento está presente	*'a' in 'manzana' → True*

	not in	Retorna *True* si un elemento no está presente	*'x' not in 'casa'* → *True*

Fuente: elaboración propia

Luego de haber entendido los aspectos básicos de Python como operadores y sintaxis, ahora se describe el control de flujo, en programación se determina en qué orden se ejecutan las instrucciones. En Python, se maneja principalmente con las estructuras condicionales, que funcionan como una serie de preguntas que el programa se crea para tomar decisiones. La instrucción principal es el *if*, que indica: "Si esta condición es verdadera, haz esto". Si no se cumple, se pasa al *elif*, que es como decir: "si no se cumple la primera condición, pregunta si cumple esta nueva condición ____, si cumple haz esto___". Y si ninguna condición *if* o *elif* se cumple, entonces pasa a las acciones configuradas en e*lse,* es decir el actúa como plan B. (Lutz, 2013). Por ejemplo:

```
edad = 18

if edad < 18:

print("Es menor de edad")

elif edad == 18:

print("Tiene justo 18 años")

else:

print("Es mayor de edad")
```

Otros controles de flujo, en programación son los bucles, los cuales repiten un bloque de código múltiples veces mientras se cumpla una condición, facilitando la automatización de tareas repetitivas. En Python, los bucles más comunes son el *while* y el *for*. El bucle *while* ejecuta el código mientras una condición lógica sea verdadera; es útil cuando no se conoce cuántas veces se repetirá la acción. Por ejemplo:

```
contador = 0

while contador < 3:

print("Repetición", contador)

contador += 1
```

En este caso, el ciclo imprime el mensaje mientras el contador sea menor que 3, incrementando su valor en cada iteración. Por otro lado, el bucle for recorre una secuencia (como una lista o un rango de números) y ejecuta el bloque de código para cada elemento. Es ideal cuando se conoce el número de repeticiones o se trabaja con colecciones. Por ejemplo:

```
for i in range(3):

print("Iteración", i)
```

Aquí, el bucle for, recorre los números 0, 1 y 2, imprimiendo el texto en cada iteración. Ambos tipos de bucles sirven para escribir programas eficientes y flexibles, permitiendo controlar el flujo de ejecución según las necesidades del programa (Lutz, 2013).

Si el código está en un ciclo repetitivo. Los comandos *break* y *continue* dan control sobre ese bucle, el primer comando es una "salida de emergencia": si una condición se cumple, detiene el bucle inmediatamente y el programa sigue con las líneas de código que están después del bloque de código del bucle. Esto evita trabajo innecesario y optimiza el flujo de tu código. Por ejemplo:

```
for numero in range(10):

if numero == 5:

break

print(numero)
```

En este caso, el bucle imprimirá los números del 0 al 4, y se detendrá cuando número sea igual a 5. Por otro lado, el comando continue omite el resto del código dentro del bucle para una iteración específica y pasa directamente a la siguiente vuelta. Por ejemplo:

```
for numero in range(5):

 if numero == 2:

 continue

 print(numero)
```

Aquí, se imprimirán los números 0, 1, 3 y 4, pero se omitirá el 2, ya que cuando se cumple esa condición, continue salta a la siguiente iteración.

Estos comandos son útiles cuando se desea controlar con precisión el comportamiento de un bucle en tiempo de ejecución (Allen, 2015).

Existe un último tipo de bloque que controla el flujo de trabajo, pero es considerado más como un control de errores, los bloques *try-except.* Cuando se ejecuta una operación que podría fallar (por ejemplo, dividir por cero, abrir un archivo que no existe, convertir un string a número), al usar *try / except* se previene el error que detenga tu programa y se toma alguna acción alternativa. Por ejemplo:

```
try:

# Bloque de código que puede generar un error

resultado = 10 / 0

except:

# Bloque de código que se ejecuta si ocurre un error

print("Ocurrió un error")
```

5.4. Estructuras de datos y funciones

Las estructuras de datos en programación son como herramientas para organizar la información, para que el código sea más claro y rápido. Python dispone de las listas, que funcionan muy bien para guardar colecciones ordenadas y fáciles de modificar; los conjuntos, perfectos cuando se requiere de elementos únicos sin importar el orden; y los diccionarios, ideales para guardar información en formato clave y valor, cada una está

pensada para resolver diferentes tipos de problemas, como insertar, eliminar o buscar datos (Miller et al. 2006).

5.4.1. List (Lista)

Una lista es una estructura de datos ordenada y mutable que almacena elementos heterogéneos (de diferentes tipos). Se crea encerrando los elementos entre corchetes [ ].

Cómo crearla:

```
frutas = ["manzana", "banana", "cereza"]
```

Cómo acceder a un elemento:

```
print(frutas[1])  # Resultado: banana
```

Cómo acceder a un rango (sublista):

```
print(frutas[0:2])  # Resultado: ['manzana', 'banana']
```

Las listas permiten modificar, agregar o eliminar elementos.

5.4.2. Tuple (Tupla)

Una tupla es similar a una lista, pero inmutable: una vez creada, no existe forma posible de modificar sus valores. Se usa para almacenar datos que no deben cambiar.

Cómo crearla:

```
coordenadas = (4, 5)
```

Cómo acceder a un elemento:

```
print(coordenadas[0])  # Resultado: 4
```

Cómo acceder a un rango:

```
print(coordenadas[0:2])  # Resultado: (4, 5)
```

Aunque son similares en acceso a las listas, no reasignan valores.

5.4.3. Dict (Diccionario)

Un diccionario almacena datos en pares clave-valor, lo cual permite buscar elementos por su clave en lugar de por posición.

Cómo crearlo:

```
persona = {"nombre": "Ana", "edad": 30}
```

Cómo acceder a un valor por su clave:

```
print(persona["nombre"])  # Resultado: Ana
```

Cómo agregar o modificar un valor:

```
persona["edad"] = 31
```

Los diccionarios no están ordenados por defecto, pero dan acceso rápido a los datos mediante claves únicas.

Tabla 5.4. Métodos más utilizados de las estructuras de datos en Python.

Estructura	Método	Descripción breve	Ejemplo de uso
list	append()	Agrega un elemento al final de la lista	frutas.append("kiwi")
	remove()	Elimina la primera aparición de un valor específico	frutas.remove("banana")
	pop()	Elimina y retorna el último elemento (o uno por índice)	frutas.pop()
	insert()	Inserta un elemento en una posición específica	frutas.insert(1, "uva")
	sort()	Ordena la lista en su lugar (modifica la original)	frutas.sort()
tuple	count()	Cuenta cuántas veces aparece un valor	coordenadas.count(4)
	index()	Devuelve la posición del primer valor encontrado	coordenadas.index(5)
dict	keys()	Retorna todas las claves del diccionario	persona.keys()
	values()	Retorna todos los valores del diccionario	persona.values()
	items()	Retorna pares clave-valor como tuplas	persona.items()
	get()	Retorna el valor de una clave, sin error si no existe	persona.get("nombre")
	pop()	Elimina una clave y retorna su valor	persona.pop("edad")
	update()	Agrega o modifica múltiples claves y valores	persona.update({"edad": 31})

Fuente: elaboración propia

Una función en Python es como un plano reutilizable para una tarea específica: un bloque de código que define con *def*, recibe un nombre y en ocasiones acepta parámetros. La idea es dividir el programa en piezas más pequeñas y manejables, lo que facilita su comprensión, mantenimiento y depuración. Una vez creada, para usarla se llama a esta función desde cualquier parte del código, cuantas veces se necesite, promoviendo la reutilización del código y la organización lógica. Además, usando *return* las funciones devuelven un valor, ésta es una característica esencial para una programación estructurada y escalable (Miller et al. 2006).

Una función se declara utilizando la palabra clave *def*, seguida del nombre de la función y paréntesis () (que pueden contener parámetros). El cuerpo de la función debe ser inventada. La sintaxis básica de una función:

```
def nombre_funcion(parámetros_opcionales):

# Bloque de código (cuerpo de la función)

 return valor_opcional
```

Ejemplo:

```
def saludar(nombre):

mensaje = f"Hola, {nombre}!"

return mensaje

# Llamada a la función

print(saludar("Carlos"))
```

CAPÍTULO VI

PYTHON PARA CIENCIA DE DATOS

Diversos autores han abordado herramientas y técnicas de ciencia de datos con énfasis en Python y sus bibliotecas. Se destacan herramientas como *NumPy, TensorFlow* y *PyTorch* en conjunto con repositorios como *GitHub* para promover principios de machine learning (Psallidas et al. 2019; Stancin y Jovic, 2019), (Odegua & Ikpotokin, 2020). Las bibliotecas como *NumPy, SciPy, Pandas* y *Matplotlib*, resaltan su aplicación en el análisis estadístico y gráfico (Rogel-Salazar, 2018). En la literatura científica se destaca la herramienta de Python, evidenciando la versatilidad de sus bibliotecas en la visualización, análisis y modelado de datos.

6.1. IPython

IPython (*Interactive Python*) es un entorno creado en 2001 por Fernando Pérez (VanderPlas, 2016) proporcionando un entorno interactivo mejorado que incluye, compatibilidad con la visualización de datos y facilidades para computación distribuida y paralela (Pérez y Granger, 2007). Además, este entorno destaca por su utilidad en tareas interactivas, sus "comandos mágicos" que agilizan procesos comunes, sus funciones para visualizar, comunicar datos de forma clara, la posibilidad de ejecutar códigos por celdas y la facilidad para operar con las variables creadas. Spyder tiene integrado por defecto el entorno de trabajo de *IPython* y está complementado con el editor de código.

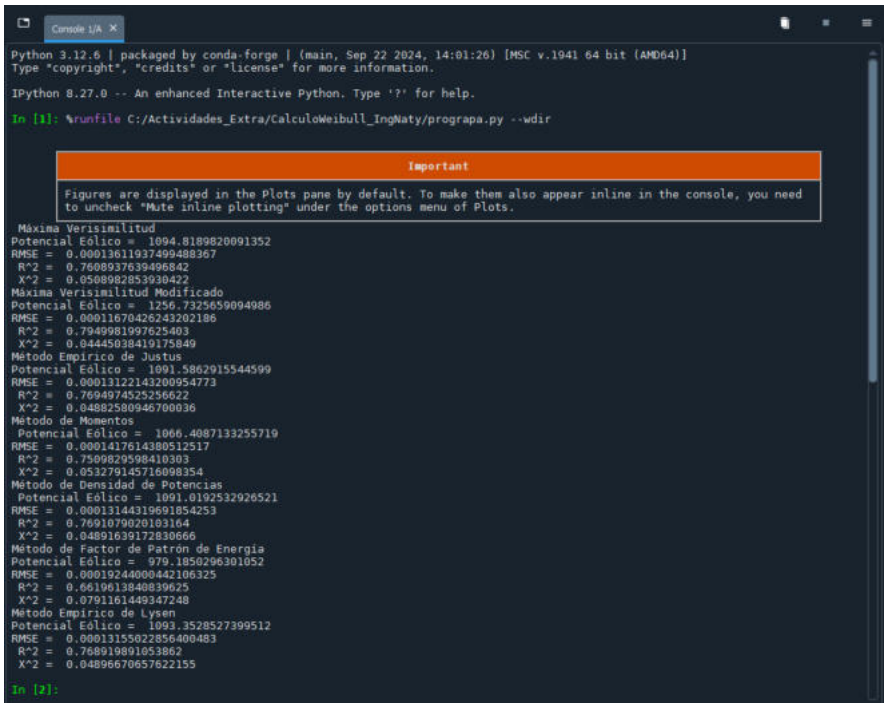


Figura 6.1. Entorno IPython implementado en Spyder.

Fuente: elaboración propia.

A diferencia del intérprete estándar de Python, *IPython* fue diseñado específicamente para potenciar el trabajo interactivo, debido a que su entorno esta enriquecido con una ejecución de pruebas y scripts. Además, ofrece una mejor gestión de errores, por su sistema de depuración mejorado y a comandos mágicos que facilitan la ejecución de tareas comunes como la temporización de código, el acceso a comandos del sistema, o la carga dinámica de scripts.

6.2. Numpy

Es una estructura de datos que sirven para almacenar información. Cumple con una de las tareas esenciales en ciencia de datos, almacenar información de distintas fuentes en una misma estructura eficiente, un *array.*

Figura 6.2. Logotipo de la librería NumPy.

Fuente: elaboración propia.

En un análisis con ciencia de datos el primer paso es el preprocesamiento de datos, es decir, convertir información de imágenes, sonido, texto o mediciones numéricas en representaciones numéricas que puedan ser procesadas por un computador. La información se almacena en un arreglo numérico, por ejemplo, una imagen a color se convierte en una matriz de datos, en donde cada celda tiene un trio de números representando la información RGB de ese punto. Luego, se almacena y manipula estos arreglos de manera eficiente para poder realizar cálculos, análisis, filtrado, selección, etc.

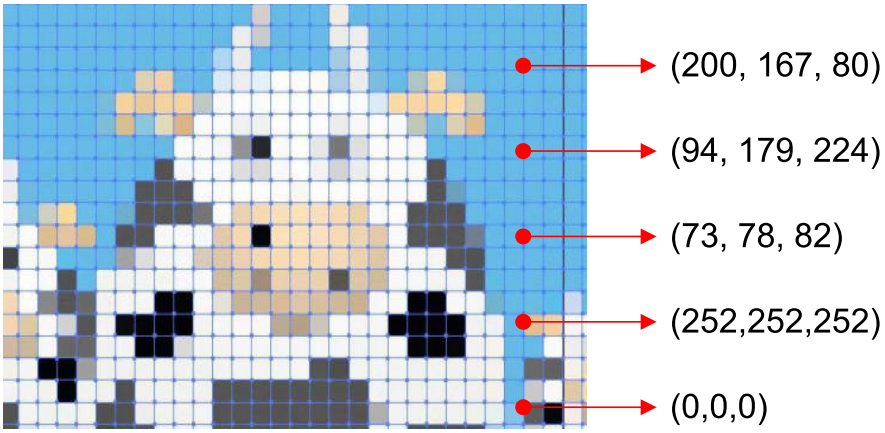


Figura 6.3. Transformación de imagen a valores RGB.

Fuente: elaboración propia.

En Python se trabajar con arreglos multidimensionales de datos numéricos utilizando *NumPy* (abreviatura de *Numerical Python*). Este paquete tiene una ventaja sobre el uso de listas o diccionarios por su flexibilidad con tipos de datos heterogéneos evitando la sobrecarga proporcionando arreglos con elementos homogéneos, permitiendo operaciones vectorizadas mucho más rápidas y eficientes, así como un soporte claro para estructuras multidimensionales (Lin, 2012).

El rendimiento de *NumPy* se aumenta mediante tres técnicas: vectorizar los cálculos, evitar la copia de datos en memoria y minimizar el número de operaciones (Van Der Walt et al. 2011). Además, *NumPy* es una biblioteca de código abierto desarrollada activamente por la comunidad científica y técnica.

Tabla 6.1. Lista de librerías que utilizan los arreglos NumPy como base.

Librería	Descripción y aplicaciones
Dask	Arreglos distribuidos y paralelismo avanzado para análisis, facilitando el rendimiento a gran escala.
CuPy	Biblioteca de arreglos compatible con NumPy para computación acelerada por GPU con Python.
JAX	Transformaciones componibles de programas NumPy: diferenciación, vectorización y compilación *just-in-time* a GPU/TPU.
Xarray	Arreglos multidimensionales etiquetados e indexados para análisis avanzados y visualización.
Sparse	Biblioteca de arreglos dispersos compatible con NumPy que se integra con *Dask* y el álgebra lineal dispersa de *SciPy*.
PyTorch	Es una librería con algoritmos para aprendizaje profundo muy útil porque acelera el prototipado en investigación y la producción.
TensorFlow	Librería integral de python para desarrollar modelos de aprendizaje automático de manera fácil e intuitiva.
Arrow	Plataforma de desarrollo entre lenguajes para datos en memoria en formato columnar y análisis.
Xtensor	Arreglos multidimensionales con difusión *(broadcasting)* y computación perezosa para análisis numérico.
Awkward Array	Manipula datos tipo JSON usando expresiones similares a *NumPy*.
Uarray	Sistema de *backend* en Python que desacopla la API de su implementación; *unumpy* proporciona una API compatible con *NumPy*.
TensorLy	Aprendizaje y álgebra tensorial con soporte para usar de forma fluida *NumPy, PyTorch, TensorFlow o CuPy*.

Fuente: Adaptado de Numpy Team, (2025).

6.2.1. NumPy Array

La librería de *NumPy* se instala de manera rápida y fácil en tu entorno virtual a través del Anaconda *Prompt* con el comando "*conda install numpy*", la manera para llamar a la librería es mediante "*import numpy* as *np*", y se

requiera utilizar un array, sus funciones, objetos y métodos debe ir precedido por *np*.

La manera más simple de crear un arreglo es convirtiendo una estructura de datos tipo lista a través del objeto *np.array*, como se muestra a continuación:

```
1    import numpy as np
2    Lista = [1,2,3,4,5]
3    arr = np.array(Lista)
```

Se verifica que se ha creado un arreglo a través del comando *type* y los valores de este usando *print* en la terminal de *IPython*:

```
1    type(arr)
2    Out[5]: numpy.ndarray
3    print(arr)
4    [1 2 3 4 5]
```

De igual manera, se crea arreglos multidimesionales a través de matrices de datos. Siguiendo el ejemplo de la Figura 6.3, ahora se crear una matriz de tuplas con 3 dígitos de una imagen que tiene una medida de 3x3 pixeles:

```
1    import numpy as np
2    Imagen = [[[1,2,3],[4,5,6],[7,8,9]],
3     [[1,2,3],[4,5,6],[7,8,9]],
4     [[1,2,3],[4,5,6],[7,8,9]]]
5    arr = np.array(Imagen)
```

El *array* de *numpy* permite acceder a los datos almacenados y ejecutar una amplia cantidad de cálculos, que consiste en un puntero de memoria, junto con metadatos para interpretar los datos almacenados: «tipo de dato», «forma» y «pasos» (Harris et al., 2020). A continuación, se presentan algunos de los fundamentos que *NumPy* incorpora en sus arreglos, para explicarlos, se comienza del siguiente *array:*

```
1    lista = [[1,2,3],[4,5,6],[7,8,9]]
2    arr = np.array(lista)

3    arr
4    Out[4]:
5    array([[1, 2, 3],[4, 5, 6],  [7, 8, 9]])
```

a) La estructura de datos del arreglo de *NumPy* incluye campos de metadatos que describen sus características, entre las cuales se tiene *data* que muestra la información guardada, *data type* que muestra el tipo de dato que esta guardado en el arreglo, *shape* incluye las dimensiones que tiene el arreglo y *strides* que brinda la información de la memoria que se está utilizando.

b) Al acceder a un arreglo utilizando cortes (*slices*) y pasos (*steps*), se visualiza los datos en el arreglo por filas, columnas o un rango de filas y columnas.

```
1  arr[1,1]
2  Out[9]: 5
3  arr[:,2]
4  Out[10]: array([3, 6, 9])
5  arr[2,:]
6  Out[11]: array([7, 8, 9])
7  arr[1:,1:]
8  Out[12]:
9  array([[5, 6], [8, 9]])
```

c) Se pueden utilizar mascaras booleanas, es decir extraer los datos que cumplan con un condicional, también es común usar coordenadas escalares u otros arreglos y el resultado será una copia independiente del arreglo original.

```
1  arr[1,1]
2  Out[15]: 5
3  arr[arr>6]
4  Out[16]: array([7, 8, 9])
5  [arr[0,1],arr[1,2]]
6  Out[17]: [2, 6]
```

d) Las operaciones matemáticas sobre los arreglos o entre arreglos se facilita por la vectorización que permite aplicar operaciones de forma eficiente sobre todos los elementos simultáneamente.

```
1  arr + 1
2  Out[18]:
3  array([[ 2,  3,  4],
4  [ 5,  6,  7],
```

```
5       [ 8,  9, 10]])
```

De igual manera, se realiza operaciones entre arreglos.

```
1       arr2 = np.array([[ 2,  3,  4], [ 5,  6,  7],[ 8,  9, 10]])
2       rr + arr2
3       Out[22]: array([[ 3,  5,  7], [ 9, 11, 13], [15, 17, 19]])
```

e) En Numpy se realiza (broadcasting) operaciones matemáticas entre arreglos de distintas dimensiones sin la necesidad de un proceso extra de redimensionarlos manualmente. En el caso que las dimensiones no coincidan para una operación directa, NumPy iguala las dimensiones sin gastar más memoria, lo cual lo vuelve eficiente en términos de recursos.

```
1       arr1 = np.array([ 1,  2,  3])
2       arr2 = np.array([[1],[2]])
3       arr1
4       Out[1]: array([1, 2, 3])
5       arr2
```

```
6   Out[2]:
7   array([[1],
8    [2]])
```

En este ejemplo el arr1 tiene dimensiones 1x3 mientras que el arr2 es de dimensiones 2x1, si se realiza una operación con estos dos arreglos se observar a continuación el resultado luego del *broadcasting* de *numpy*.

```
1   arr1 + arr2
2   Out[26]:
3   array([[2, 3, 4],
4   [3, 4, 5]])
```

```
1   arr1 * arr2
2   Out[27]:
3   array([[1, 2, 3],
4   [2, 4, 6]])
```

6.2.2. Creación, operaciones y "slicing" de arreglos

La manipulación y operaciones con arreglos, bajo este contexto *NumPy* ofrece una variedad de funciones y métodos para realizar numerosas operaciones. La construcción, operación, cálculos y visualización de arreglos permite conocer características específicas de la información para cualquier análisis computacional moderno y para la ciencia en general. Numpy tienen funciones que generan arreglos de las dimensiones como se ejecutan a continuación:

a) Para crear un arreglo relleno de ceros se utiliza *np.zeros* donde la entrada son las dimensiones.

```
1   #Arreglo de una dimensión con 8 ceros
2   np.zeros(8)
3   Out[4]: array([0., 0., 0., 0., 0., 0., 0., 0.])
```

```
1   #Arreglo de una matriz 3x2 con ceros
2   np.zeros((3,2))
3   Out[5]:
4   array([[0., 0.], [0., 0.],[0., 0.]])
```

b) Para crear un arreglo relleno de unos se utiliza np.ones donde la entrada son las dimensiones.

```
1   #Arreglo de una dimensión con 6 unos
2   np.ones(6)
3   Out[6]: array([1., 1., 1., 1., 1., 1.])
```

```
1   #Arreglo de una matriz 4x3 relleno de unos
2   np.ones((4,3))
3   Out[8]:
4   array([[1., 1., 1.],[1., 1., 1.], [1., 1., 1.],[1., 1., 1.]])
```

c) Se crea un arreglo unidimensional con una secuencia de números donde existe un rango establecido y la variación que existe entre cada número. Esto se realiza a través del método np.arange(inicio,final,paso).

```
1   #Arreglo desde 1 hasta 10 con paso de 0.5
2   np.arange(1,10,0.5)
3   Out[9]:
```

```
4       array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. , 6.5, 7.,
5              7.5, 8. , 8.5, 9. , 9.5])
```

d) Se crea un arreglo con una secuencia de números estableciendo el rango y la cantidad de datos que se tiene dentro de los límites, de tal manera que los puntos son equidistantes, esto se realiza a través del comando np.linspace(inicio,final,cantidad_de_datos)

```
1
        #Arreglo con 15 datos dentro del rango de 1 a 10
2
        np.linspace(1,10,15)
3
        Out[10]:
4
        array([ 1.,  1.64285714,  2.28571429,  2.92857143,
5       3.57142857,  4.21428571,  4.85714286,  5.5,  6.14285714,
6       6.78571429,  7.42857143,  8.07142857,  8.71428571,
        9.35714286, 10. ])
7
```

Con lo que respecta, a las operaciones con los arreglos principalmente se realiza las siguientes:

a) Suma *array1 + array2*

b) Resta *array1 - array2*

c) Multiplicación *array1 * array2*

d) División *array1 / array2*

e) Potencia *array1 ** a,* donde a es un numero

f) División entera *array1 // a,* donde a es un numero

g) Módulo *array1 % a,* donde a es un numero

Además, esta biblioteca tiene preestablecido funciones matemáticas universales, métodos de reducción, operaciones de algebra lineal, transformación y funciones estadísticas, de las cuales se describen las más importantes a continuación:

```
1  arr = np.array([ 1.2,  2.3,  3.4 , 5.6])
```

Tabla 6.2. Descripción de las funciones universales implementadas en NumPy.

Función	Implementación	Ejemplo
Seno	np.sin()	1 np.sin(arr) 2 Out[16]: array([0.93203909, 0.74570521, -0.2555411 , -0.63126664])
Coseno	np.cos()	np.cos(arr) 1 Out[17]: array([0.36235775, - 2 0.66627602, -0.96679819, 0.77556588])
Tangente	np.tan()	1 np.tan(arr) 2 Out[18]: array([2.57215162, -1.11921364, 0.2643169 , -0.81394328])
Exponencial	np.exp()	1 np.exp(arr)

		2 Out[19]: array([3.32011692, 9.97418245, 29.96410005, 270.42640743])
Logarítmica	np.log()	1 np.log(arr) 2 Out[20]: array([0.18232156, 0.83290912, 1.22377543, 1.7227666])
Raíz cuadrada	np.sqrt()	1 np.sqrt(arr) 2 Out[21]: array([1.09544512, 1.51657509, 1.84390889, 2.36643191])
Redondeo	np.round()	1 np.round(arr) 2 Out[22]: array([1., 2., 3., 6.])
Suma de elementos	np.sum()	1 np.sum(arr) 2 Out[23]: 12.5
Producto de elementos	np.prod()	1 np.prod(arr) 2 Out[24]: 52.55039999999999
Extraer el mínimo	np.min()	1 np.min(arr) 2 Out[25]: 1.2
Extraer el máximo	np.max()	1 np.max(arr) 2 Out[26]: 5.6
Media	np.mean()	1 np.mean(arr) 2 Out[27]: 3.125
Mediana	np.median()	1 np.median(arr) 2 Out[28]: 2.8499999999999996
Desviación estándar	np.std()	1 np.std(arr) 2 Out[29]: 1.6269219403523942
Varianza	np.var()	1 np.var(arr) 2 Out[30]: 2.6468749999999996
Transpuesta	np.transpose()	1 np.transpose(arr) 2 Out[34]: array([1.2, 2.3, 3.4, 5.6])
Remodelar (reshape)	np.reshape()	1 np.reshape(arr,(2,2)) 2 Out[33]: 3 array([[1.2, 2.3], 4 [3.4, 5.6]])
Dividir en subarreglos	np.split()	1 np.split(arr,2) 2 Out[35]: [array([1.2, 2.3]), array([3.4, 5.6])]

Fuente: elaboración propia.

El concepto de "*array slicing"*, es una técnica que extrae subconjuntos de datos de un arreglo sin necesidad de utilizar bucles ni condicionales de parada. Es una forma de acceder a partes específicas de un array, y se basa en el uso de índices y rangos. A continuación, se muestran ejemplos:

Tabla 6.3. Ejemplo 1 de "array slicing".

1	**arr = np.array([0,1,2,3,4,5,6,7,8,9,10])**
1 **2**	*# Los valores desde índice 4 hasta el final* arr[4:] Out[6]: array([4, 5, 6, 7, 8, 9, 10])
1 **2**	*# Los valores desde el inicio hasta 4* arr[:4] Out[8]: array([0, 1, 2, 3])
1 **2** **3**	*# Los valores desde el índice 2 hasta 7* arr[2:7] Out[10]: array([2, 3, 4, 5, 6])

Fuente: elaboración propia.

En este ejemplo se nota que el primer valor si se toma dentro del corte del arreglo, mientras que el segundo valor representa el final del *slicing,* se considera hasta el número ingresado menos la unidad, por esa razón el array solo tiene los numero hasta el sexto valor.

Tabla 6.4. Ejemplo 2 de "array slicing".

1 **2** **3**	***# Los valores del arreglo con un paso de 2*** **arr[::2]** **Out[13]: array([0, 2, 4, 6, 8, 10])**
1	*# Los valores del arreglo con un paso de 3*

```
2  arr[::3]
3  Out[14]: array([0, 3, 6, 9])
```

Fuente: elaboración propia.

Cuando se trabaja con matrices existen opciones de corte y selección que dependen las filas o columnas. A continuación, se muestran algunas aplicaciones del *slicing* en matrices:

Tabla 6.5. Aplicaciones del slicing en matrices.

```
1  M = np.array([[0,1,2],
2   [3,4,5],
3   [6,7,8]])
```

```
1  # Para acceder a toda la primera fila
2  M[0,:]
3  Out[16]: array([0, 1, 2])
```

```
1  # Para acceder a toda la segunda columna
2  M[:,1]
3  Out[18]: array([1, 4, 7])
```

```
1  #Las primeras dos filas
2  M[:,:2]
3  Out[23]:
4  array([[0, 1],
5  [3, 4],
6  [6, 7]])
7  #Las últimas dos filas
8  M[:,1:]
9  Out[24]:
10 array([[1, 2],
11 [4, 5],
12 [7, 8]])
```

```
1  # Generar una submatriz
2  M[1:3,1:3]
3  Out[28]:
4  array([[4, 5],
5  [7, 8]])
```

Fuente: elaboración propia.

Se ha descrito lo más importante de *NumPy* sobre su creación y manipulación, pero existen muchas funciones, métodos y técnicas de *NumP* (consultar en el siguiente enlace: https://numpy.org/doc/stable/user/whatisnumpy.html).

6.3. Pandas

Pandas, la biblioteca desarrollada en 2008 e impulsada por la necesidad de contar con estructuras de datos expresivas que permitieran manipular y analizar datos etiquetados. Pandas fue diseñada para cerrar la brecha entre los lenguajes de programación de propósito general y las plataformas especializadas como *R, SAS o SQL*, donde el manejo de datos con etiquetas, series temporales o matrices (McKinney, 2011).

Figura 6.4. Logotipo de la librería Pandas.

Fuente: elaboración propia.

Pandas se construyó sobre la base de la biblioteca *NumPy*, aprovechando su eficiencia para operaciones numéricas sobre arreglos, pero incorporando dos nuevas estructuras de datos como las *Series temporales* y los *DataFrames*. Estas estructuras permiten trabajar con datos heterogéneos, datos faltantes, e información etiquetada, como datos financieros, mediciones científicas, series temporales o registros administrativos. A

través de Pandas, tareas como ordenar, agrupar, fusionar, pivotear o filtrar datos se realizan de forma directa y eficiente (VanderPlas, 2016).

Entre sus ventajas destaca la velocidad en el procesamiento de datos en memoria, la flexibilidad para integrarse con otras bibliotecas (como *NumPy, Matplotlib, SciPy y Scikit-learn*), y la productividad que ofrece al usuario para manipular datos complejos con pocas líneas de código. La librería de Pandas ha implementado funcionalidades avanzadas estadísticas y de procesamiento como la alineación automática de datos, el indexado jerárquico y la posibilidad de trabajar de forma eficiente con conjuntos de datos multidimensionales.

Pandas en tu entorno de trabajo se instala desde el *prompt* de Anaconda con el comando *"conda install pandas"*. Luego se importa la librería dentro del código o desde la terminal de *IPython*, en general la sintaxis que se utiliza para esta librería es *"import pandas as pd"*.

6.3.1. Estructuras de datos

Pandas trabaja con dos estructuras principales las *Series* y el *DataFrame.* El primero es un arreglo unidimensional donde cada dato esta etiquetado y se almacena enteros, cadenas, números de puntos flotantes, etc. Estas etiquetas que pertenecen a las filas se le conoce como el *index*. A diferencia de un arreglo de *NumPy*, donde cada valor está asociado únicamente a su posición numérica, en *Series* cada dato es accedido también mediante su valor en el *index*. Por ejemplo, las Series representa las temperaturas diarias de una ciudad, donde el índice serían las fechas y los valores las temperaturas correspondientes. Pandas maneja automáticamente la

alineación y propagación de operaciones con Series, incluso cuando los índices no coinciden perfectamente. Para crear un objeto Series se utiliza el comando *pd.Series(data, index=index)*, donde *data* es creado a través de una estructura de datos unidimensional como una lista o un arreglo, mientras que el *index* está predefinido con el número desde el cero o se especifica las etiquetas que se desee, siempre con cuidado que la dimensión coincida con la de los datos.

```
import pandas as pd

data = [12,34,56,78,90]

index = ['a','b','c','d','e']

serie = pd.Series(data,index=index)

serie

Out[2]:

a   12

b   34

c   56

d   78

e   90

dtype: int64
```

Los objetos *Series* como se basan en estructuras de la librería *array* entonces heredan las mismas funcionalidades con la implementación del uso de las etiquetas del *index.*

```
# Selección

serie['a']

Out[5]: 12

# Mascaras booleanas

serie[serie>50]

Out[6]:

c    56

d    78

e    90

dtype: int64

# Operaciones Vectorizadas

serie - 2*serie

Out[8]:

a   -12

b   -34
```

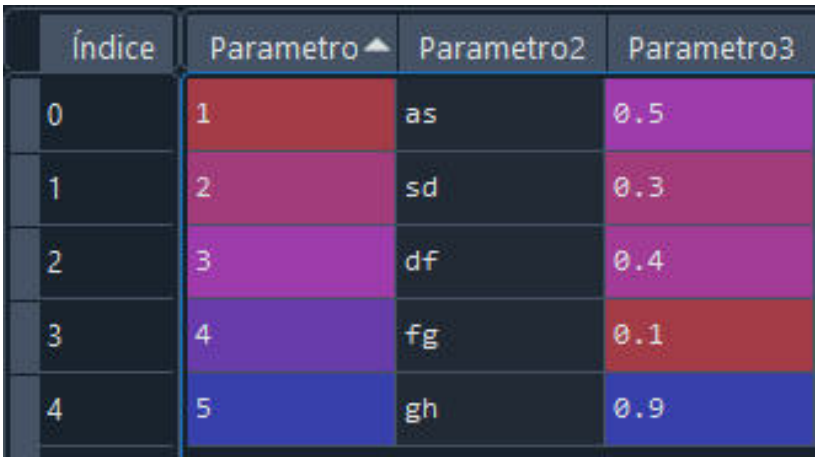
```
c  -56

d  -78

e  -90

dtype: int64

# Slicing

serie[:'d']

Out[9]:

a   12

b   34

c   56

d   78

dtype: int64
```

La segunda estructura importante en Pandas es el *DataFrame,* el cual es una estructura bidimensional que se asemeja a una tabla de base de datos o a una hoja de cálculo, con filas y columnas etiquetadas. En cada columna del *DataFrame* se tiene un objeto tipo *Series* que se identifica con una etiqueta (el atributo de la columna) y cada una almacena distintos tipos de datos como enteros, flotantes, cadenas, booleanos, entre otros, conviviendo en el mismo objeto/variable dentro del programa. Esta capacidad de

almacenamiento heterogéneo permite trabajar con operaciones como filtrado, agrupamiento, agregación y combinación de manera conjunta. Para crear esta estructura se utiliza el comando *pd.DataFrame()*, los parámetros de entrada dependen, de qué tipo de estructura de información se crea la base de datos, se genera el *dataframe* a partir de un diccionario, se muestra a continuación:

```
import pandas as pd

import numpy as np

D = {'Parametro1': [1,2,3,4,5], # usando una lista

'Parametro2': ['as','sd','df','fg','gh'], # a través de un objeto Series

'Parametro3': np.array([0.5,0.3,0.4,0.1,0.9]) # utilizando un arreglo de numpy

}

df = pd.DataFrame(D)
```

En el ejemplo anterior se nota como en el diccionario los valores tienen la misma cantidad de datos, además de ser estructuras como listas, arreglos o series. Las claves del diccionario serán los títulos de las columnas que conforman el *DataFrame.* Con el explorador de variables se observa de mejor manera el resultado:



Índice	Parametro	Parametro2	Parametro3
0	1	as	0.5
1	2	sd	0.3
2	3	df	0.4
3	4	fg	0.1
4	5	gh	0.9

Figura 6.5. Vista de un DataFrame creado a través del explorador de variables de spyder.

Fuente: elaboración propia.

Otra manera de crear un *dataframe* es a partir de una matriz de datos, luego se especifica el nombre de las columnas a través del parámetro *columns* y opcionalmente se configura el nombre los valores del *index.*

```
import pandas as pd

M = [[2,4,'d'],

[1,3,'a'],

[1.5,2,'b']]

df = pd.DataFrame(M,columns=['parametro 1','parametro 2','parametro 3'],index=[1,2,3])

df
```

Out[3]:

parametro 1 parametro 2 parametro 3

1 2.0 4 d

2 1.0 3 a

3 1.5 2 b

Se selecciona una columna a través de la etiqueta. En cambio, si se requiere seleccionar un elemento en específico existen dos métodos principales, el comando *iloc* el cual funciona con los números de las posiciones donde se encuentra el dato, mientras que *loc* utiliza las etiquetas establecidas en las filas y columnas.

```
# Selección de una columna

df['parametro 2']
```

Out[4]:

```
1  4

2  3

3  2

Name: parametro 2, dtype: int64

# Seleción de un dato por su posición
```

```
df.iloc[1,2]

Out[7]: 'a'

# Seleción de un dato por sus etiquetas

df.loc[3,'parametro 1']

Out[10]: 1.5
```

En las Series se realiza *slicing* en el objeto mediante el comando *iloc* y con la misma sintaxis que *NumPy*. Además, se realiza máscaras booleanas para nuevos *sub-dataframes* con el condicional que se requiera.

```
# Slicing de un dataframe

df.iloc[1:,1:]

Out[14]:

parametro 2 parametro 3

2          3          a

3          2          b

# Mascara Booleana

df['parametro 2']>2.5

Out[15]:

1    True
```

```
2     True

3    False

Name: parametro 2, dtype: bool

# Nuevo dataframe a partir de la mascara anterior

df[df['parametro 2']>2.5]

Out[16]:

parametro 1  parametro 2 parametro 3

1         2.0         4         d

2         1.0         3         a
```

6.3.2. Importación y procesamiento

En el mundo de ciencia de datos la información está disponible en distintos formatos, que requieren ser importados al código para su manejo. Por esta razón, se utiliza las herramientas IO (*Input/Output)* que ofrece pandas para obtener información que esta almacenada en diferentes bases de datos.

Pandas contiene herramientas para leer archivos desde múltiples formatos, como CSV, Excel, JSON y bases de datos SQL, permitiendo una integración estructurada con diversas fuentes de datos.

Tabla 6.6. IO Tools de Pandas.

Comando	Descripción	Ejemplo
read_excel(PATH)	Lee datos desde archivos de Excel (*.xls* o *.xlsx*) y los convierte en un *DataFrame*	*df = pd.read_excel("datos.xlsx")*
read_csv(PATH)	Lee datos desde archivos csv (por sus siglas en inglés, *Comma Separated Values)* y los convierte en un *DataFrame*	*df = pd.read_excel("datos.csv")*
read_json(PATH)	Carga datos desde un archivo o *string* en formato JSON (por sus siglas en inglés, *JavaScript Object Notation*) y los convierte en un *DataFrame*.	*df = pd.read_json("datos.json")*
read_sql(PATH)	Ejecuta una consulta SQL sobre una base de datos y carga los resultados como un *DataFrame*.	*conn = sqlite3.connect("mi_base_de_datos.db")* *df = pd.read_sql("SELECT * FROM clientes", conn)*
read_parquet(PATH)	Lee datos desde archivos en formato *Parquet*, un formato binario columnar altamente eficiente usado en grandes volúmenes de datos.	*df = pd.read_parquet("datos.parquet")*
.to_csv()	Guarda un *DataFrame* como un archivo de texto en formato **CSV.**	*df.to_csv("datos.csv")*
.to_excel()	Exporta un *DataFrame* a un archivo *.xlsx* de *Microsoft Excel*.	*df.to_excel("datos.xlsx")*

Fuente: elaboración propia

Los métodos de inspección de los *dataframes* son funciones para revisar de manera general las características de las bases de datos, siendo útil cuando la información es masiva, permitiendo obtener características estadísticas y

computacionales para explorar un esquema general. Estos métodos son *head(), tail(), info()* y *describe*(), a continuación, se muestra cómo trabaja cada uno de estos comandos.

Nota. Para explicar esta sección se utiliza una base de datos que esta predefinida y disponible en la librería sci-kit learning:

```
from sklearn.datasets import load_iris

iris = load_iris()

dataframe = pd.DataFrame(iris.data, columns=iris.feature_names)
```

- *head*() es un método del *DataFrame* que realiza la impresión en la pantalla de *IPython* de las primeras 5 filas que conforman los datos.

```
dataframe.head()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

- *tail*() es un método que realiza la impresión en la consola de las primeras 5 últimas filas que conforman los datos.

dataframe.tail()

Out[9]:

	sepal length (cm)	*sepal width (cm)*	*petal length (cm)*	*petal width (cm)*
145	*6.7*	*3.0*	*5.2*	*2.3*
146	*6.3*	*2.5*	*5.0*	*1.9*
147	*6.5*	*3.0*	*5.2*	*2.0*
148	*6.2*	*3.4*	*5.4*	*2.3*
149	*5.9*	*3.0*	*5.1*	*1.8*

- *info*() es un método que brinda un resumen de los datos que se encuentran almacenados. Proporciona información como el tipo de dato, la cantidad de datos, el número de nulos de cada parámetro y la información de memoria que utiliza la base de datos.

dataframe.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 4 columns):

 # Column Non-Null Count Dtype

--- ------ -------------- -----

0 sepal length (cm) 150 non-null float64

1 sepal width (cm) 150 non-null float64

2 petal length (cm) 150 non-null float64

3 petal width (cm) 150 non-null float64

dtypes: float64(4)

memory usage: 4.8 KB

- *describe*() brinda un resumen estadístico que describe la base y explora el comportamiento de los datos.

dataframe.describe()

Out[11]:

sepal length(cm) sepal width(cm) petal length(cm) petal width(cm)

count 150.0000 150.00000 150.00000 150.00000

mean 5.8433 3.05733 3.75800 1.19933

std 0.8280 0.43586 1.76529 0.76223

min	*4.3000*	*2.00000*	*1.00000*	*0.10000*
25%	*5.1000*	*2.80000*	*1.60000*	*0.30000*
50%	*5.8000*	*3.00000*	*4.35000*	*1.30000*
75%	*6.4000*	*3.30000*	*5.10000*	*1.80000*
max	*7.9000*	*4.40000*	*6.90000*	*2.50000*

Con lo que respecta al procesamiento de los datos almacenados esta librería cuenta con varias funciones y métodos que actúan sobre el *DataFrame*, los cuales se describen:

Tabla 6.7. Método de Pandas para el preprocesamiento de una base de datos.

Comando	Descripción	Implementación
isna() ***isnull()***	Ambos códigos son equivalentes y lo que realizan es devolver un *dataframe* booleano que índica si en esa celda existe un valor *null.*	*dataframe.isna()* *dataframe.isnull()*
notna() ***5notnull()***	Los códigos presentados sirven para detectar las filas donde el valor analizado no tiene un valor *null*.	*dataframe.isna()* *dataframe.isnull()*
isna().sum()	Cuenta cuántos valores nulos hay en cada columna.	*dataframe.isna().sum()*
any()	Verifica si hay al menos un nulo por columna	*dataframe.any()*
all()	Revisa si en alguna columna todos los elementos son nulos.	*dataframe.all()*
duplicated()	Devuelve un vector booleano que indica si una fila es un duplicado de una fila anterior.	*dataframe.duplicated()*

duplicated().sum()	Cuenta cuántas filas duplicadas existen	*dataframe.duplicated().sum()*
drop_duplicates()	Elimina los duplicados del DataFrame	*dataframe.drop_duplicates()*
fillna(valor)	Rellena los valores nulos con el valor que le indiques. No modifica el objeto original a menos que uses inplace=True.	*dataframe.fillna(2)*
fillna(method=__)	El método *fillna* descrito anteriormente tiene programado métodos para determinar el valor con el que debe ser llenado, entre los que existen: *ffill* para llenar con el valor en la posición anterior, *bfill con el valor de la posición siguiente.*	*dataframe.fillna(method='ffill')* *dataframe.fillna(method='bfill')*
astype(tipo)	Convierte el tipo de datos de una o varias columnas a otro tipo compatible	*dataframe["sepal length(cm)"].astype(int)*
columns	Devuelve una lista con los nombres de los parámetros que existe en la base de datos	*dataframe.columns*
index	Devuelven lista con los valores que están ocupando el índice del *DataFrame*	*dataframe.index*
values	Permite obtener un arreglo bidimensional con los datos que se almacenan en el *Dataframe*	*dataframe.values*
reindex	Para cambiar el índice o el nombre de las columnas.	*dataframe.reindex(nuevo_indice)* *dataframe.reindex(column= nuevas_columnas)*
reset_index	Devuelve el índice a valores enteros, es decir restaura columnas originales.	*dataframe.reset_index()*
to_datetime	Convierte cadenas de texto, números u objetos similares a fechas en objetos *datetime* de Pandas, lo cual permite trabajar	*dataframe.to_datetime(df['fecha'])*

	fácilmente con fechas y realizar operaciones temporales.	
diff	Calcula la diferencia entre elementos consecutivos en una serie o columna, para observar de manera rápida las variaciones o el cambio de una variable entre periodos.	*dataframe.diff()*
shift	Permite agregar un paso hacia adelante o hacia atrás según el número de períodos indicado.	

Fuente: elaboración propia.

6.3.3. Operaciones con la base de datos

Las operaciones con Pandas filtran, transforma, agrega y combina información de distintas fuentes de manera flexible y con una sintaxis de programación sencilla

```
import pandas as pd

from sklearn.datasets import load_iris

iris = load_iris()

dataframe = pd.DataFrame(iris.data, columns=iris.feature_names)

df1 = dataframe.iloc[:5,:]

df2 = dataframe.iloc[-5:,:]
```

El df1 creado tiene las siguientes características:

```
df1.head()
```

Out[1]:

	sepal length(cm)	*sepal width(cm)*	*petal length(cm)*	*petal width(cm)*
0	*5.1*	*3.5*	*1.4*	*0.2*
1	*4.9*	*3.0*	*1.4*	*0.2*
2	*4.7*	*3.2*	*1.3*	*0.2*
3	*4.6*	*3.1*	*1.5*	*0.2*
4	*5.0*	*3.6*	*1.4*	*0.2*

df1.describe()

Out[2]:

Mientras que el df2 tiene las siguientes características:

df2.head()

Out[1]:

	sepal length(cm)	*sepal width(cm)*	*petal length(cm)*	*petal width(cm)*
145	*6.7*	*3.0*	*5.2*	*2.3*
146	*6.3*	*2.5*	*5.0*	*1.9*
147	*6.5*	*3.0*	*5.2*	*2.0*
148	*6.2*	*3.4*	*5.4*	*2.3*

149 5.9 3.0 5.1 1.8

df2.describe()

Out[2]:

	sepal length(cm)	*sepal width(cm)*	*petal length(cm)*	*petal width(cm)*
count	*50.0000*	*50.00000*	*50.00000*	*50.00000*
mean	*6.58800*	*2.974000*	*5.55200*	*2.02600*
std	*0.63588*	*0.322497*	*0.55189*	*0.27465*
min	*4.90000*	*2.200000*	*4.50000*	*1.40000*
25%	*6.22500*	*2.800000*	*5.10000*	*1.80000*
50%	*6.50000*	*3.000000*	*5.55000*	*2.00000*
75%	*6.90000*	*3.175000*	*5.87500*	*2.30000*
max	*7.90000*	*3.800000*	*6.90000*	*2.50000*

Utilizando los *dataframes* se realiza operaciones con sus columnas o con todas las filas. Si se requiere una operación con todo el *dataframe* es necesario igualar el índice mediante la función *reset_index(drop=True)*. Una vez que los índices sean equivalentes se realiza operaciones vectorizadas como se muestra a continuación:

 # Se iguala el índice del segundo dataframe

```
df2 = df2.reset_index(drop=True)

# Suma de dataframes

df1 + df2
```

Out[1]:

	sepal length(cm)	sepal width(cm)	petal length(cm)	petal width(cm)
0	11.8	6.5	6.6	2.5
1	11.2	5.5	6.4	2.1
2	11.2	6.2	6.5	2.2
3	10.8	6.5	6.9	2.5
4	10.9	6.6	6.5	2.0

```
#Resta

df1 - df2
```

Out[35]:

	sepal length(cm)	sepal width(cm)	petal length(cm)	petal width(cm)
0	-1.6	0.5	-3.8	-2.1
1	-1.4	0.5	-3.6	-1.7
2	-1.8	0.2	-3.9	-1.8

3	-1.6	-0.3	-3.9	-2.1
4	-0.9	0.6	-3.7	-1.6

```
# Multiplicación

df1 * df2
```

Out[36]:

	sepal length(cm)	sepal width(cm)	petal length(cm)	petal width(cm)
0	34.17	10.50	7.28	0.46
1	30.87	7.50	7.00	0.38
2	30.55	9.60	6.76	0.40
3	28.52	10.54	8.10	0.46
4	29.50	10.80	7.14	0.36

6.4. Visualización de datos con Matplotlib

La visualización de los datos resume información en un gráfico que complementa la interpretación, el análisis y la descripción de los fenómenos representados en los datos. La librería intuitiva para solventar esta necesidad es *Matplotlib* considerada una multiplataforma para la visualización de datos en Python (cabe puntualizar que esta librería no es la única existente para visualización de datos).

Matplotlib está construida sobre arreglos de *NumPy* y *se* integra de manera fluida con el ecosistema *SciPy*. Fue concebida por John Hunter en 2002 como una extensión interactiva de *IPython* para gráficos al estilo MATLAB, y evolucionó rápidamente hasta convertirse en una de las herramientas más utilizadas en ciencia de datos. Desde su versión inicial en 2003, *Matplotlib* ha sido adoptada por instituciones científicas como el Instituto del Telescopio Espacialn (VanderPlas, 2016; Lin, 2012). Su instalación en un entorno virtual de trabajo es muy sencilla a través de *conda*, desde el *prompt* de anaconda se activa el entorno y luego se digita el comando: *conda install matplotlib*.

Matplotlib ofrece una interfaz funcional para crear gráficos básicos de manera rápida que se encuentra en el submódulo pyplot, comúnmente importado con la línea de código *import matplotlib.pyplot as plt*. Además, también tiene una estructura orientada a objetos que otorga a los usuarios un control detallado sobre cada elemento del gráfico como son: el título, los ejes, la grilla, colores, tamaño de fuente, etc.

En esta biblioteca se utiliza arreglos o estructuras de datos como listas, está integrada al ecosistema científico de *SciPy*, funciona en múltiples sistemas operativos y compatible con formatos de salida gráfica. Esta versatilidad promueve una comunidad activa de desarrolladores y usuarios provenientes de diversas disciplinas científicas. En el ámbito de la meteorología, *Matplotlib* y PyNGL (por sus siglas en inglés, *Python Interface to the NCAR Graphics Library*) son utilizadas como una biblioteca de visualización científica, para tareas de visualización 2D y 3D, incluyendo gráficos de

líneas y de contorno, diseñada específicamente para trabajar con datos científicos atmosféricos, oceánicos y del clima.

6.4.1. El Stateful y POO en matplotlib

El *Stateful* es una característica de diseño de programación y se basa en una yuxtaposición de estados, esto quiere decir que sobre la variable que se está trabajando (en este caso la figura o imagen creada por *matplotlib* se actualiza o configurar características mediante programación, este estado permanece guardado como si fuera un *checkpoint*. El *stateful* mantiene su estado a lo largo del tiempo o entre ejecuciones, lo que significa que recuerda interacciones o cambios realizados previamente. *Matplotlib* usa el enfoque *stateful* a través de su módulo *pyplot*, se ejecuta *pyplot* con el código *import matplotlib.pyplot as plt*, en donde aparecen funciones que realizan gráficas fácilmente. Se presenta un ejemplo con el que inicia a trabajar y explicar el *stateful* mostrando el estado de la gráfica en cada comando:

Iteración 1.- Se importa las librerías necesarias (Figura 6.6). Luego, usando *numpy* se genera un arreglo de 100 datos y se calcula el seno de cada valor, creando una gráfica inicial con el comando *plot()*:

```
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)

fig = plt.figure()

plt.plot(x, np.sin(x))
```

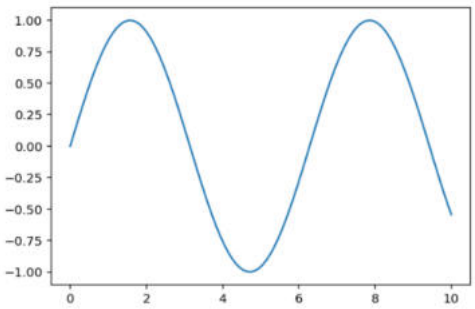


Figura 6.6. Primera iteración.

Fuente: elaboración propia.

Iteración 2.- Partiendo del estado de la iteración 1 (Figura 6.6.) se configura los ejes de la figura con (ver Figura 6.7.):

plt.xlabel('(x)')

plt.ylabel('Seno (x)')

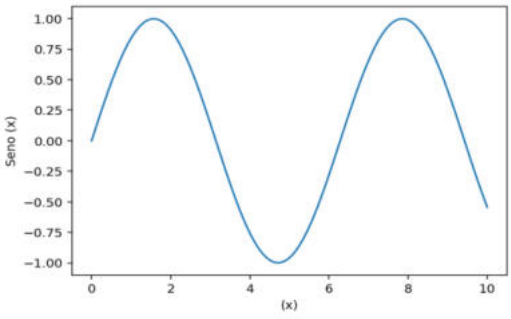


Figura 6.7. Segunda iteración.

Fuente: elaboración propia.

Iteración 3.- Desde el estado anterior (Figura 6.7.) se modifica la imagen para que tenga una grilla y un título con los comandos (ver figura 6.8.):

plt.title('Función Seno')

plt.grid()

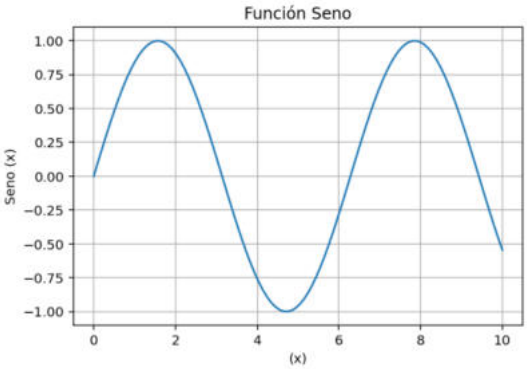


Figura 6.8. Tercera iteración.

Fuente: elaboración propia.

En el ejemplo se observa cómo funciona el diseño de programación basado en estados, ya que se sigue agregando más líneas de programación al código que configura varias propiedades de la imagen como el color, tamaño, tipo de letra, tipo de línea, etc, y estas configuraciones se realizaran sobre la base del estado anterior.

La programación orientada a objetos es un paradigma de programación que organiza el código en clases y objetos, donde cada objeto representa una entidad con estado (atributos) y comportamiento (métodos), este enfoque permite trabajar con *matplotlib* de manera más estructurada y es

especialmente útil cuando se requiere crear gráficos complejos. El *statefull* utiliza la programación orientada a objetos para crear una figura y sobre la misma se realizan actualizaciones de fondo y forma

En el enfoque orientado a objetos, cada gráfico es una instancia de clase, y todas las acciones para ello se realizan llamando a métodos de estos objetos, los cuales cambian su estado permitiendo seguir modificándolo al igual que en el *stateful.* Un objeto *Figure,* es el contenedor principal del gráfico capaz de contener uno o varios subgráficos (*subplots*). Se crea usando plt.figure() y se configura algunos parámetros de los que destacamos:

figsize = a,h : establece el ancho (a) y la altura (h) de la figura; las unidad de medida son pulgadas.

Facecolor = color: configura el color del fondo de la imagen.

Otro objeto *Axes,* representa un sistema de coordenadas dentro de la figura, es decir que es sobre la clase donde se realizaran los gráficos, diagramas, etc, teniendo en cuenta que este objeto solo realizar representaciones en dos dimensiones. Se crea objeto *Axes* vinculado a un espacio de trabajo creado mediante un método del objeto *Figure* preestablecido a través del comando *Figure.subplots()*, el cual tiene como salida un objeto A*xes.*

Nota: Existen distintas formas de crear un objeto *Figure* y *Axes* para obtener un espacio de trabajo donde representar gráficamente datos. El método *add_subplots* tiene varios parámetros iniciales para establecer y configurar la gráfica, de los cuales se destacan los siguientes:

nrows = número de filas, ncols = número de columnas: ambos parámetros solo reciben números enteros y establece la cantidad de imágenes que estarán en la figura, por ejemplo, si se establecen *nrows* igual 3 y *ncols* igual a 2, entonces la figura tendrá una matriz de seis objetos *Axes* distribuidos con el número de filas y columnas determinado (Véase el Ejemplo 3 de la sección 6.3.4).

sharex = booleano, sharey = booleano: Este parámetro establece que los objetos *axes* en la figura compartan la misma escala y numeración de uno de sus ejes. Este parámetro es opcional ya que por defecto el valor de ambos es False.

6.4.2. Tipos de gráficos estadísticos

Los principales tipos de gráficos en dos dimensiones que se realizan con *Matplotlib* se detallan con un conjunto de datos simulado. Para esta sección se trabaja sobre un espacio de una figura con un objeto *Axes*, para lo cual se ejecuta los siguientes códigos:

```
fig = plt.figure()

ax = fig.subplots(1,1)
```

Como resultado se crea la siguiente imagen por defecto:

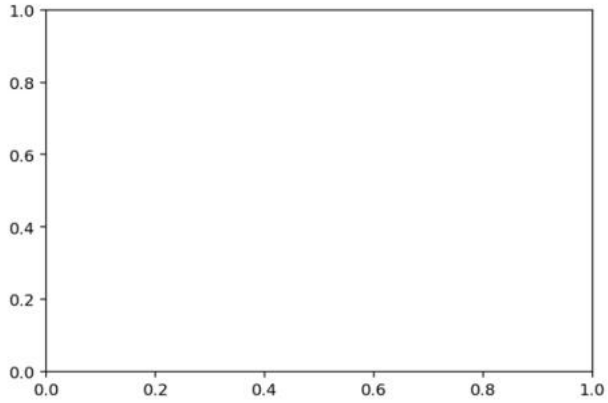


Figura 6.9. Espacio de trabajo sencillo para realizar gráficas.

Fuente: elaboración propia.

Gráfico de líneas: Es una gráfica donde se observa la evolución de los valores entre dos variables relacionadas, dichos valores o puntos están unidos a través de líneas rectas y el comando que hace esto es plot(x,y), donde x e y son listas o arreglos con los valores en los ejes.

```
x = np.linspace(0,10,100)

y = np.sin(x)

ax.plot(x,y)
```

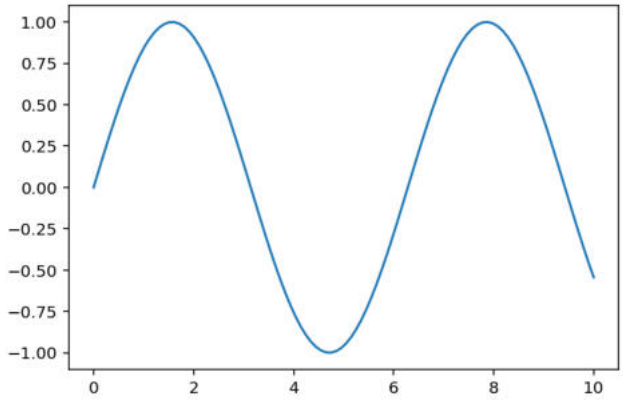


Figura 6.10. Líneas con el comando plot.

Fuente: elaboración propia.

Diagrama de dispersión: sirve para observar las relaciones existentes entre dos variables, pero a través de una nube de puntos. Esto se realiza a través del comando *scatter(x,y)*.

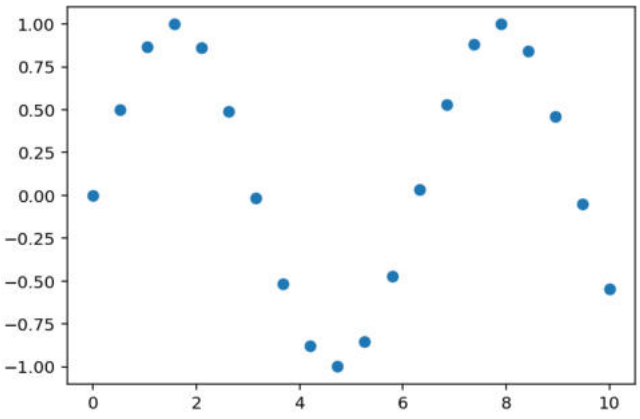


Figura 6.11. Nube de puntos generada con el comando scatter.

Fuente: elaboración propia.

Diagrama de barras: Este tipo de gráficas es muy utilizado en estadística para observar valores de datos entre categorías y entender de mejor manera su comportamiento en términos de conteo por categoría, por ejemplo, se crea dos tipos de diagramas de barras, verticales y horizontales, usando los comandos *bar(categorías,valores)* y *barh(categorías,valores)*, respectivamente.

```
categorias = ['<18 años', '19-40 años', '40-60 años', '>60 años']

valores = [500,462,525,230]

ax.bar(categorias,valores)
```

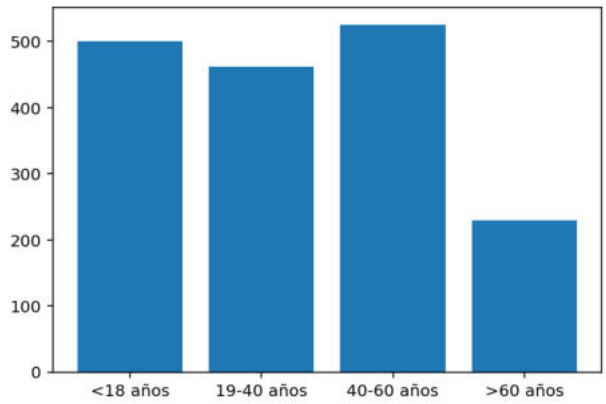


Figura 6.12. Diagrama de barras vertical generado con el comando bar.

Fuente: elaboración propia.

```
categorias = ['<18 años', '19-40 años', '40-60 años', '>60 años']

valores = [500,462,525,230]
```

ax.barh(categorias,valores)

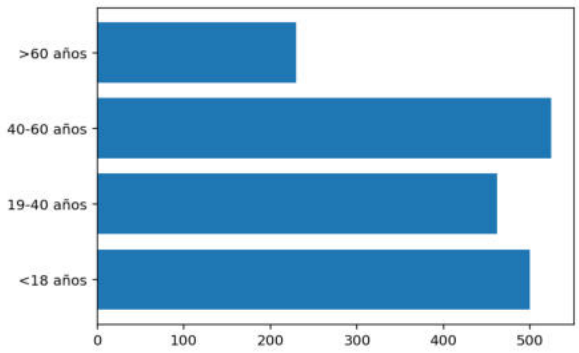


Figura 6.13. Diagrama de barras vertical generado con el comando barh.

Fuente: elaboración propia.

Histograma: Es un tipo de gráfico visualiza la distribución de los datos que se están estudiando. En este gráfico se estudiar una variable en términos de la frecuencia en que se obtiene un valor dentro de rangos determinados. La diferencia primordial con un gráfico de barras es que no existe una separación categórica, sino que cada barra de un histograma representa un intervalo, es decir es continuo a diferencia del gráfico de barras que es discreto. Para generar un histograma usando *matplotlib* se utiliza el comando *hist(valores,n_barras)*, esta función realiza

los cálculos para representar el histograma, donde *n_barras* es el número de intervalos en el que se quiere dividir al rango de datos.

```
import matplotlib.pyplot as plt

import numpy as np
```

```
# 200 valores aleatorios con una distribución normal

rng = np.random.default_rng(19680801)

data = rng.standard_normal(200)

fig = plt.figure()

ax = fig.subplots(1,1)

ax.hist(data,8)
```

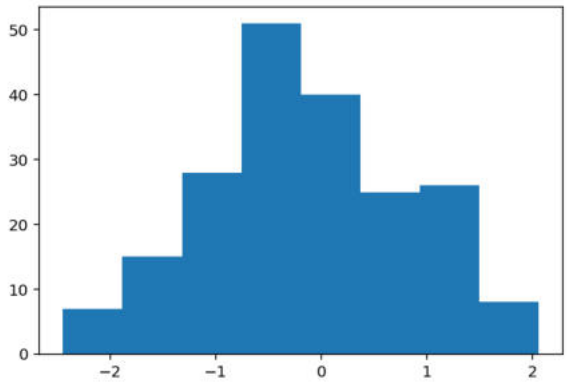


Figura 6.14. Histograma generado con el comando hist.

Fuente: elaboración propia.

Diagrama de Caja y alambres *(Boxplot)*: Esta gráfica es una herramienta para visualizar la distribución de datos, mostrando parámetros estadísticos que describen el comportamiento de los datos como la mediana, los cuartiles, los valores atípicos y el rango de los datos. Un *boxplot* resume la distribución de diferentes grupos de datos, determinando la variabilidad

entre categorías e identificando la presencia de valores atípicos. En la librería se genera un diagrama de caja usando el comando *boxplot(valores)*:

```
import matplotlib.pyplot as plt

import numpy as np

# 200 valores aleatorios con una distribución normal

rng = np.random.default_rng(19680801)

data = rng.standard_normal(200)

fig = plt.figure()

ax = fig.subplots(1,1)

ax.boxplot(data,8)
```

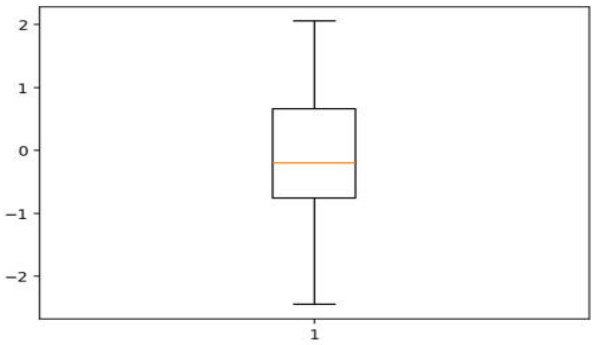


Figura 6.15. Diagrama de caja generado con el comando boxplot.

Fuente: elaboración propia.

En climatología para visualizar datos bidimensionales de información satelital, como la temperatura de una región donde a cada cuadrícula de una matriz se le asigna la temperatura que se midió en ese punto, es decir en una grilla de datos. Esta información se representa en los mapas de calor, que son gráficas 2D. En *matplotlib* implementa este tipo de gráficos a través del método *imshow(M,cmap,interpolation,vmin,vmax)* donde:

M: es un arreglo bidimensional que contiene los valores numéricos en cada punto de la grilla. También se ingresa una matriz con dimensiones (*fils,cols,3*) para graficar imágenes con los datos RGB.

Cmap: este parámetro selecciona el mapa de colores para representar los valores de la matriz, por defecto se crea la gráfica con el mapa de colores "*viridis*", sin embargo, en *matplotlib* existen varios *cmap* que se utilizan (ver detalles en

https://matplotlib.org/stable/users/explain/colors/colormaps.html)

interpolation: este parámetro es opcional y selecciona un método de interpolación para mejorar o disminuir la resolución de la imagen que se está generando. Los tipos de interpolación admitidos *son 'none', 'auto', 'nearest', 'bilinear', 'bicubic', 'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos' y 'blackman'* (Ver Figura 6.15).

vmin y vmax: son valores numéricos que definen el rango de datos que cubre el mapa de colores. Por defecto, el código calcula cual es el valor máximo y mínimo para cubrir todo el rango de valores de los datos proporcionados.

Se establece valores si se requiere comparar entre dos gráficas con un mismo rango. A continuación, se muestra un ejemplo de gráfica bidimensional sencillo:

```
# Se crea una malla aleatoria de valores

Matriz = np.arange(100).reshape((10, 10))

fig = plt.figure()

ax = fig.subplots(1,1)

im = ax.imshow(Matriz)
```

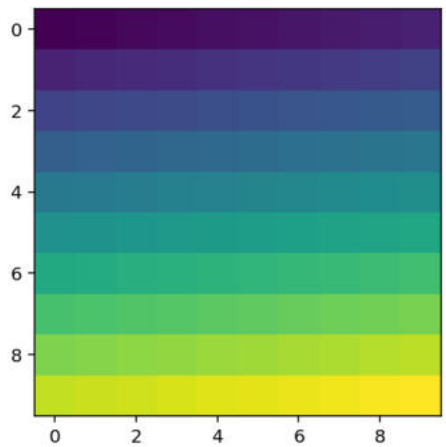


Figura 6.16. Mapa de calor generado con el comando imshow.

Fuente: elaboración propia.

Si se requiere mostrar los valores que representa cada color en la gráfica mediante el comando *fig.colorbar(im1)* donde im1 es el objeto creado con *imshow()*. (Ver figura del ejemplo 2 en la sección 6.4.3).

```
im = ax.imshow(Matriz,cmap='plasma',interpolation='bicubic')
```

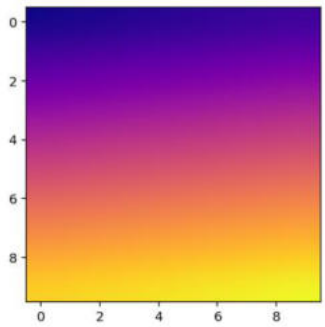


Figura 6.17. Resultado de un gráfico tipo mapa de calor generado con el comando imshow, realizando cambios en sus parámetros iniciales.

Fuente: elaboración propia.

6.4.3. Personalización de gráficos

Las gráficas generadas a partir de los datos deben estar representadas de manera clara, comprensible y estéticamente adecuada. En *Matplotlib*, la personalización de gráficos es una de las cualidades, porque ajusta títulos, etiquetas, colores, estilos de línea, leyendas y otros elementos visuales. Se trabaja con POO mediante los objetos *Figure* y *Axes,* se ejecutan los métodos que personalizan la gráfica y se puede seguir configurando, cambiando el estado, utilizando la característica de diseño *Stateful*.

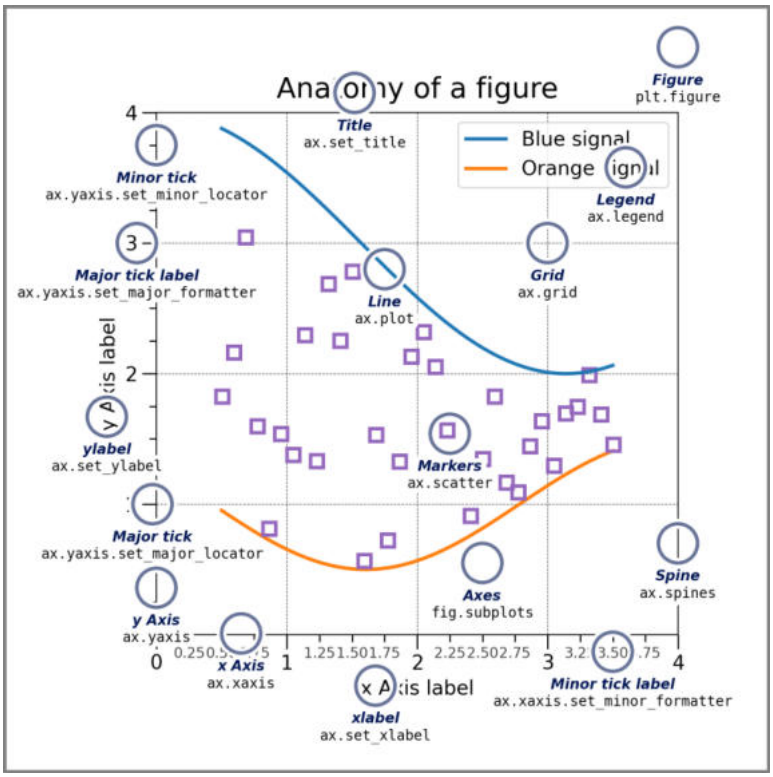


Figura 6.18. Partes de una figura en matplotlib.

Fuente: John et al., (2012)

Tabla 6.8. Métodos más importantes del objeto Figure para personalizar gráficos.

Comando	Descripción	Ejemplo/Implementación
suptitle()	Añade un título para toda la figura, útil cuando tienes múltiples subplots.	*Figure.suptitle('Represetnación de regíme-nes climáticos')*
subplots_adjust(left, right, top,	Controla los márgenes y el	*Figure.subplots_adjust(0.25,0.5,0.5,0.25,2,2)*

bottom, wspace, hspace)	espacio entre subgráficos. Donde *left, right, top, bottom* son los valores entre 0 y 1 que ajustan los bordes, mientras que *wspace y hspace* son los espacios horizontal y vertical entre *subplots* en pulgadas.	
tight_layout()	Ajusta automáticamente los subgráficos y etiquetas para que no se sobrepongan.	*Figure.tight_layout()*
legend()	Coloca una leyenda en la figura para lo cual se detecta de tres maneras el contenido de la leyenda. 1) Detección automática de elementos a través del parámetro *label* cuando se crea una gráfica (Ver ejemplos de la sección 6.3.4). 2)Lista explícita de artistas y sellos	*1) Figure.legend()* *2) Figure.legend([linea1, linea2, linea3], ['Estacion ESPOCH', 'Estación Alao', 'Estación Tixán'])* *3) linea1, = ax1.plot([1, 2, 3], label='Estación Espoch')* *linea2, = ax2.plot([1, 2, 3], label='Estación Alao')* *fig.legend(handles=[linea1, linea2])*

	en la leyenda. 3) Lista explícita de artistas en la leyenda	
text(x,y,cadena)	Coloca un texto en la posición *x,y* muy útil para agregar información extra o anotaciones	*Figure.text(5,3.6,'Punto de recarga hídrica')*
set_facecolor()	Configura el color del fondo de la gráfica	*Figure.set_facecolor('gray')*
set_edgecolor()	Personaliza el filo de todo el contenedor de la gráfica	*Figure.set_edgecolor('blue')*

Fuente: elaboración propia.

Tabla 6.9. Métodos más importantes del objeto Figure para personalizar gráficos.

Comando	**Descripción**	**Ejemplo/Implementación**
ax.set_title('Texto')	Establece el título del *subplot*	*Axes.set_title("Temperatura promedio")*
ax.set_xlabel("Texto")	Modifica el nombre o etiqueta para el eje de las ordenadas del *subplot*	*ax.set_xlabel("Tiempo (s)")*
ax.set_ylabel("Texto")	Modifica el nombre o etiqueta para el eje de las abscisas del *subplot*	*ax.set_ylabel("Temperatura (°C)")*
ax.set_xlim([xmin, xmax])	Define el valor mínimo y máximo de la escala en el eje de las ordenadas	*ax.set_xlim([0, 10])*

ax.set_ylim([ymin, ymax])	Define el valor mínimo y máximo de la escala en el eje de las abscisas	*ax.set_ylim([0, 30])*
ax.set_xscale("log"), ax.set_yscale("log")	Con estos métodos se cambia la escala de cualquiera de los ejes a una logarítmica	*ax.set_yscale("log")* → *cambia el eje de las abscisas a escala logarítmica*
ax.grid(True, linestyle=__, alpha=#)	Activa una cuadrícula de fondo en el *subplot*, se escoge el tipo de línea con el parámetro *linestyle* o el grosor con un número del 0 al 1 en el parámetro *alpha*.	*ax.grid(True, linestyle="--", alpha=0.7)*
ax.axhline(y)	Permite dibujar una línea horizontal en el valor *y*.	*ax.axhline(5)*
ax.axvline(x)	Permite dibujar una línea horizontal en el valor de *x*.	*ax.axvline(1)*
ax.legend(fontsize=#)	Con este comando se activa la leyenda del *subplot*. Opcionalmente se ingresa el tamaño de letra de la leyenda.	*ax.legend(fontsize=10)*
ax.set_facecolor(Color)	Cambia el color de fondo del área de dibujo	*ax.set_facecolor('gray')*
ax.text(x, y, "Texto", fontsize=#, color=Color)	Inserta una cadena de texto en coordenadas las específicas ingresadas en los parámetros x,y. De igual manera, se modifica el tamaño y color de la letra	*ax.text(2.3, 8.5, "Punto de equilibrio", fontsize=12, color='red')*

Fuente: elaboración propia.

6.4.4. Ejemplos

Ejemplo 1: en este primer ejemplo se crea un gráfico de barras y un histograma en un mismo contenedor Figure, para observar las modificaciones que se realizan con *matplotlib*.

```
#valores para graficar

import matplotlib.pyplot as plt

import numpy as np

# Se crea y configura la Figura contenedor global

fig = plt.figure(figsize=(8,5))

axs = fig.subplots(1,2)

fig.suptitle('Ejemplo 1. graficos de barras')

fig.set_facecolor('gray')

fig.set_edgecolor('blue')

# Se perspnaliza cada subplot

# Subplot 1

# Se crean valores y categorias

categorias = ['<5', '6-10', '11-15', '>16']

valores = [500,462,525,230]
```

```
axs[0].barh(categorias,valores,color='orange',label='Temperatura Alao')

axs[0].grid(linestyle='--')

axs[0].set_xlabel('Frecuencia')

axs[0].set_ylabel('Temperatura (°C)')

axs[0].set_title('Gráfico de Barras')

# Subplot 2

# 200 valores aleatorios con una distribución normal

rng = np.random.default_rng(19680801)

data = rng.standard_normal(200)

axs[1].hist(data,10,color='brown',label='Temperatura Espoch')

axs[1].set_title('Histograma')

axs[1].set_xlabel('Anomalias de Temperatura °C')

axs[1].set_ylabel('Frecuencia')

# Se configura la figura para que no se sobrepongan las

# etiquetas

fig.tight_layout()
```

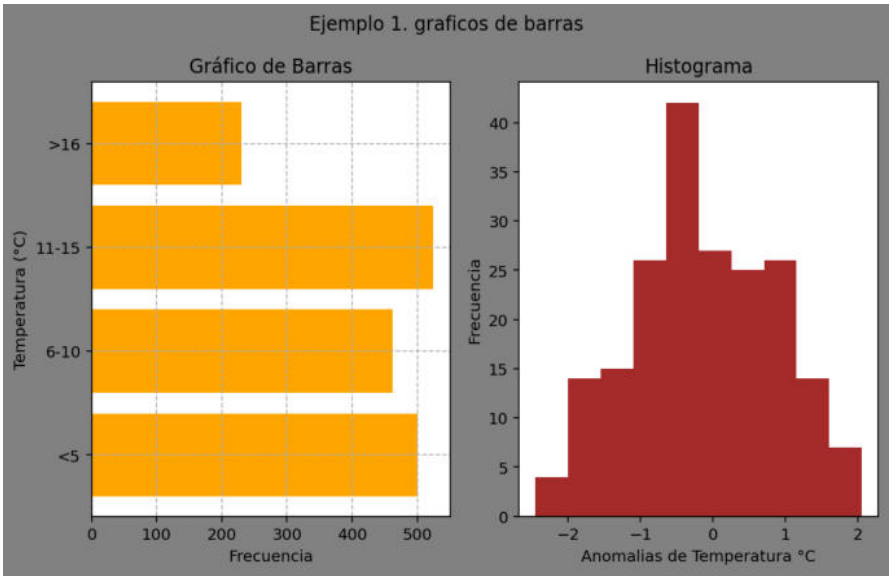


Figura 6.19. Resultado de ejecución del ejemplo 1.

Fuente: elaboración propia.

Ejemplo 2: en el siguiente ejemplo se muestra como personalizar en un gráfico de mapa de calor bidimensional, configurado varios parámetros como los estilos y colores. Se agregan anotaciones para revisar su funcionalidad y se realiza modificaciones en el *Cmap*.

```
# valores para graficar

import matplotlib.pyplot as plt

import numpy as np

# Se crea una malla aleatoria de valores

Matriz = np.arange(100).reshape((10, 10))
```

```
# Se crea y configura la Figura contenedor global

fig = plt.figure(figsize=(8,8))

axs = fig.subplots(2,2)

fig.suptitle('Ejemplo 2. Graficos de Mapas de Calor')

fig.set_facecolor('lightgray')

# Personalizar cada subplot

# Subplot 1

axs[0,0].imshow(Matriz)

axs[0,0].set_xlabel('Longitud')

axs[0,0].set_ylabel('Latitud')

axs[0,0].set_title('Mapa de calor 1')

# Subplot 2

axs[1,0].imshow(Matriz,interpolation='bicubic')

axs[1,0].grid(linestyle='--')

axs[1,0].set_xlabel('Longitud')

axs[1,0].set_ylabel('Latitud')

axs[1,0].set_title('Mapa de calor 2')
```

```
# Subplot 3
axs[0,1].imshow(Matriz,interpolation='nearest',cmap='gray',
                vmin=0,vmax=80)
axs[0,1].grid(linestyle='--')
axs[0,1].set_xlabel('Longitud')
axs[0,1].set_ylabel('Latitud')
axs[0,1].set_title('Mapa de calor 3')
# Subplot 4
im = axs[1,1].imshow(Matriz,interpolation='bilinear',cmap='gray',
                vmin=15,vmax=80)
fig.colorbar(im)
axs[1,1].set_xlabel('Longitud')
axs[1,1].set_ylabel('Latitud')
axs[1,1].set_title('Mapa de calor 4')
# Se configura la figura para que no se sobrepongan las
# etiquetas
fig.tight_layout()
```

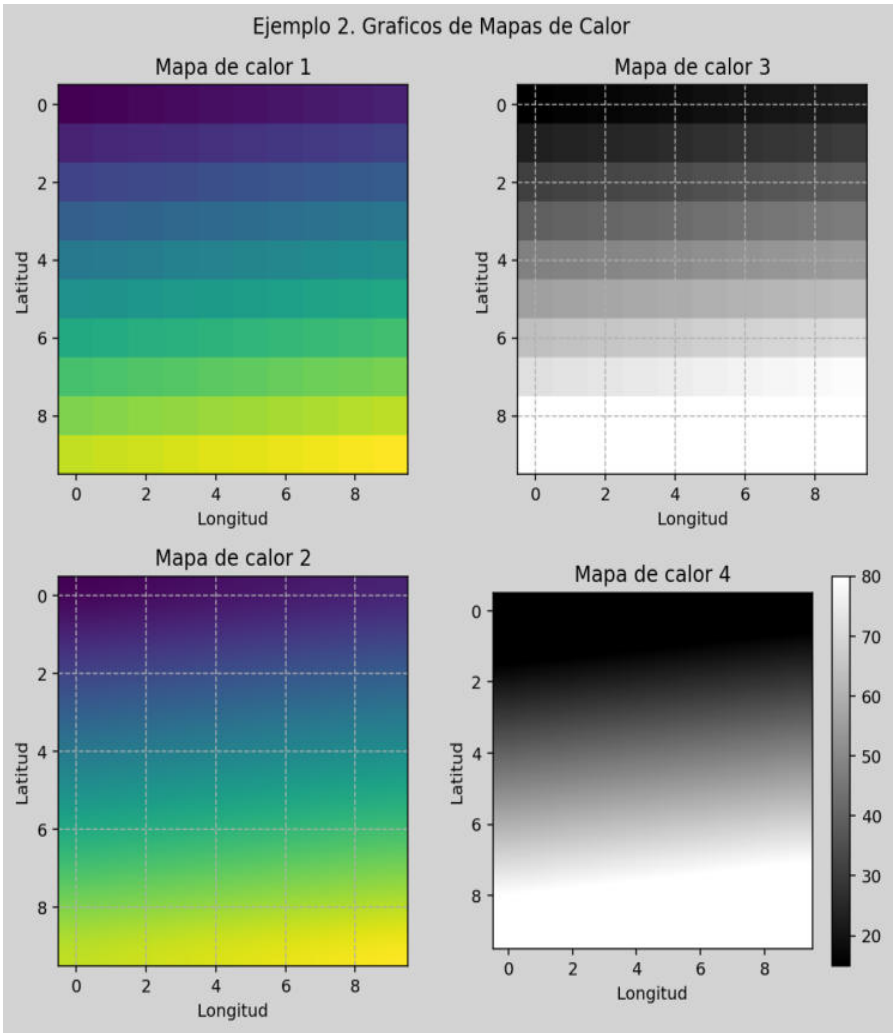


Figura 6.20. Resultado de ejecución del ejemplo 2.

Fuente: elaboración propia.

Ejemplo 3: una figura general que cuenta con 6 subfiguras distribuidas en tres filas y dos columnas. Se personaliza un título global y títulos para cada *subplot*, se agrega una cuadricula en algunos *subplots* con distinto estilo para notar la diferencia, mientras que en otros se cambia el color y tamaño

de fuente de las etiquetas. Se modifica los colores y las escalas de cada *subplot* y de la figura general para observar cómo funciona la personalización de color.

```
import matplotlib.pyplot as plt

import numpy as np

# valores para graficar

x = np.linspace(0,10,100)

y = np.sin(x)

# Se crea y configura la Figura contenedor global

fig = plt.figure(figsize=(12,8))

axs = fig.subplots(3,2)

fig.suptitle('Ejemplo 3. Figura global con 6 subplots')

fig.set_facecolor('lightgray')

# Personalizar cada subplot

# Subplot 1

axs[0,0].plot(x,y)

axs[0,0].grid()

axs[0,0].set_title('Subfigura 1')
```

```
# Subplot 2

axs[1,0].plot(x,y)

axs[1,0].grid(linestyle='--',alpha=0.5)

axs[1,0].set_title('Subfigura 2')

# Subplot 3

axs[2,0].plot(x,y)

axs[2,0].grid(linestyle='--',alpha=0.9)

axs[2,0].axhline(0.5,color='orange')

axs[2,0].set_title('Subfigura 3')

# Subplot 4

axs[0,1].plot(x,y)

axs[0,1].set_title('Subfigura 4')

axs[0,1].set_xlabel('Tiempo (s)')

axs[0,1].set_ylabel('Seno(t)')

# Subplot 5

axs[1,1].plot(x,y)

axs[1,1].set_title('Subfigura 5')
```

```
axs[1,1].grid(linestyle='--',alpha=0.9)

axs[1,1].set_xlabel('Tiempo (s)',color='red')

axs[1,1].set_ylabel('Seno(t)',color='blue')

# Subplot 6

axs[2,1].plot(x,y,label='Línea 6')

axs[2,1].set_title('Subfigura 6',color='red',fontsize=12)

axs[2,1].grid(linestyle='--',alpha=0.9)

axs[2,1].set_xlabel('Tiempo (s)',color='red')

axs[2,1].set_ylabel('Seno(t)',color='blue')

axs[2,1].set_xlim(0,5)

axs[2,1].set_ylim(0,2)

axs[2,1].axvline(2.5,linestyle='--',color='orange')

axs[2,1].legend()

# Configrar la figura para que no se sobrepongan las

# etiquetas

fig.tight_layout()
```

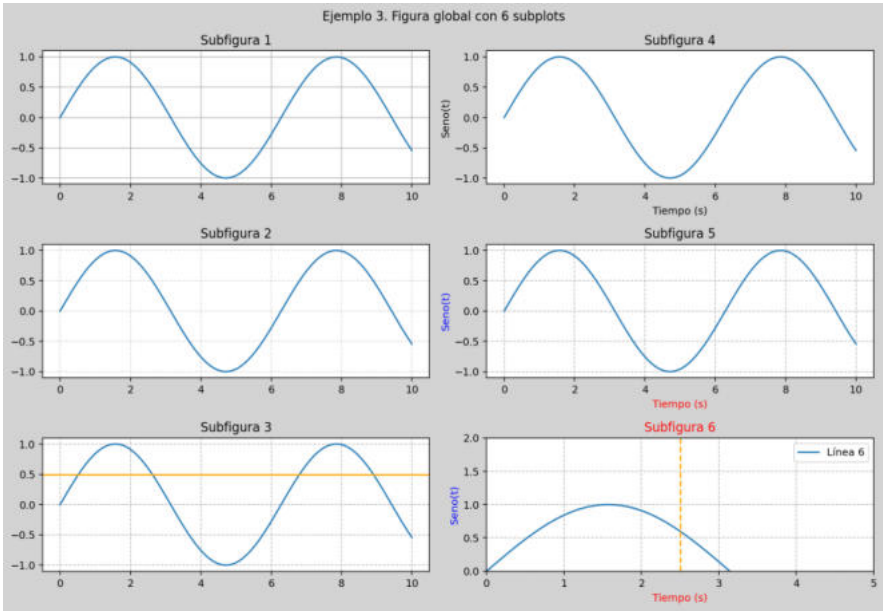


Figura 6.21. Resultado de ejecución del ejemplo 3.

Fuente: elaboración propia.

CAPÍTULO VII

APRENDIZAJE AUTOMÁTICO (MACHINE LEARNING)

Los algoritmos de aprendizaje automático, conocidos como ML (por sus siglas en inglés *Machine Learning*), este campo permite a los computadores aprender de los datos sin necesidad de programarlos de manera explícita. En términos prácticos, un algoritmo de aprendizaje automático mejora su desempeño en una tarea concreta a medida que gana experiencia a partir de la información disponible, a esta acción se le conoce como entrenar al modelo. Este proceso de entrenamiento, por ejemplo, crea un filtro de correo no deseado con la capacidad de diferenciar entre mensajes válidos y spam a partir de ejemplos etiquetados, y mientras más datos etiquetados se tenga más refinado son sus predicciones con el tiempo en función al volumen de los datos de entrenamiento (VanderPlas, 2016).

Las aplicaciones del aprendizaje automático se han utilizado en múltiples áreas, como la meteorología y la climatología, para realizar predicciones de precipitaciones, detección de eventos extremos, estimaciones en ciencias de la tierra, modelar procesos ambientales, proyectar impactos del cambio climático o analizar imágenes satelitales. Su potencial radica en la capacidad de procesar grandes volúmenes de información y extraer características para la toma de decisiones.

Esta técnica moderna simplifica tareas, con un solo algoritmo, se optimiza el proceso y ofrece un mejor rendimiento, lo cual, mediante métodos tradicionales requieren un ajuste manual. También ofrece alternativas eficaces a problemas complejos que son limitados en técnicas

convencionales. Además, los sistemas de aprendizaje automático se adaptan de manera continua a los nuevos datos mediante el entrenamiento del modelo, facilitando la obtención de conocimientos a partir del análisis de grandes volúmenes de información.

7.1. Tipos de algoritmos ML

Los algoritmos de aprendizaje automático se clasifican en: el aprendizaje supervisado y el aprendizaje no supervisado.

El aprendizaje supervisado se centra en modelar la relación entre características de entrada y una variable de salida. Para modelar esta relación entre las variables de entrada y la variable de respuesta se realiza tareas como la clasificación cuando las etiquetas son categorías discretas y la regresión cuando se trata de valores continuos—(Scikit-learn, 2017).

El ejemplo sobre la detección de correos que son Spam se explica el funcionamiento de este tipo de algoritmos a través de la Figura 7.1 en donde se describe a cada correo con una respuesta o etiqueta. La base de datos de entrenamiento para este tipo de algoritmos ML requiere las columnas de parámetros de entrada (al menos dos) y la columna de variable de respuesta (la cual es única).

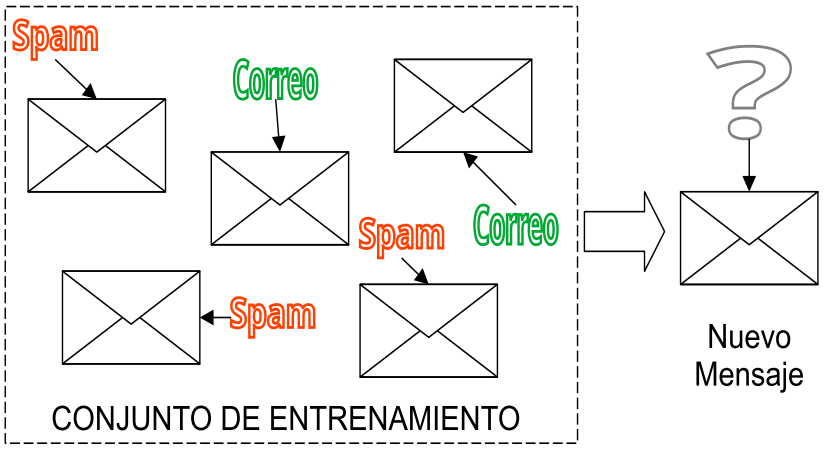


Figura 7.1. Conjunto de datos etiquetados para realizar aprendizaje supervisado (caso, identificación de correos Spam).

Fuente: elaboración propia.

Se detalla algoritmos de aprendizaje supervisado:

- K-Vecinos Más Cercanos
- Regresión Lineal
- Regresión Logística
- Máquinas de Vectores de Soporte (MVS)
- Árboles de Decisión y Bosques Aleatorios
- Redes Neuronales (Nota: existen algunas redes neuronales en las que su arquitectura les permite también usar el enfoque de aprendizaje no supervisado.)

Los algoritmos de aprendizaje automático no supervisado, busca estructuras y patrones internos en los datos sin necesidad de etiquetas. Entre estos algoritmos se destaca los algoritmos de agrupamiento (*clustering*) o de reducción de dimensionalidad, para descubrir grupos con características similares para la representación de la información resumida (Mohri, 2018). En la base de datos de entrenamiento para este tipo de algoritmos no requiere de una variable de respuesta.

Algoritmos de aprendizaje no supervisado más utilizados:

- Agrupamiento (clustering)
  - k-Means
  - Análisis de Clústeres Jerárquico (HCA)
  - Maximización de Expectativas
- Visualización y reducción de dimensionalidad
  - Análisis de Componentes Principales (PCA)
  - PCA Kernel

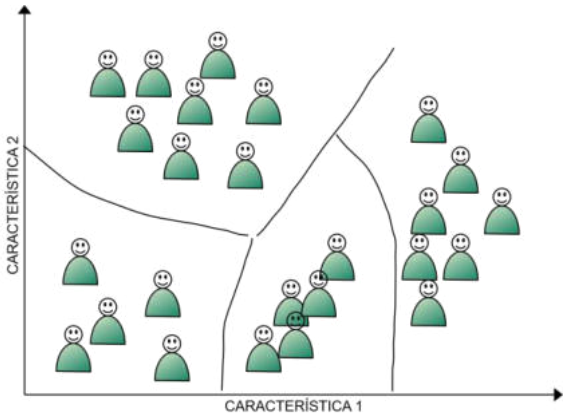


Figura 7.2. Algoritmo de agrupamiento (clustering).

Fuente: elaboración propia.

"Semi-supervised Learning" **(Aprendizaje semi-supervisado):** El sistema de aprendizaje se utiliza un conjunto de datos que mezcla algunos datos etiquetados (como en ML supervisado) y otros no etiquetados. El objetivo es que el modelo aproveche la estructura del conjunto de datos no etiquetados para mejorar su rendimiento, incluso, con la apariencia de pocas etiquetas disponibles.

"Reinforcement Learning" **(Aprendizaje por refuerzo):** En el aprendizaje por refuerzo (llamado agente) interactúa activamente con un entorno, es decir que observa el estado del entorno, toma decisiones (acciones), y recibe recompensas o castigos según la acción. El objetivo del agente es aprender una estrategia óptima (llamada *policy*) para maximizar las recompensas acumuladas. Una característica especial del aprendizaje por refuerzo es que la fase de entrenamiento y la de prueba están entrelazadas, el agente aprende mientras actúa. Sin embargo, no se le

informa de inmediato cuál será el resultado a largo plazo de sus acciones, lo que crea un dilema, ¿Debe explorar nuevas acciones para aprender más? ¿O debe explotar el conocimiento actual para obtener recompensas seguras?

"Transductive Inference" **(Inferencia transductiva):** Muy parecido al aprendizaje semi-supervisado, busca predecir las etiquetas para un conjunto específico de datos no etiquetados, y no generalizar a otros datos. Por ejemplo, se tiene un conjunto fijo de pacientes sin diagnóstico, el modelo necesita predecir para esos pacientes. Esta tarea puede ser sencilla, pero también está sujeta a muchas preguntas abiertas en investigación sobre cuándo y cómo mejora el rendimiento respecto a métodos más generales.

"On-line Learning" **(Aprendizaje en línea):** En este enfoque, el modelo aprende de forma continua, en rondas sucesivas. Se recibe una observación sin etiqueta, se predice y recibe la etiqueta real, provocando una pérdida si existe equivocación. El objetivo es minimizar el error acumulado a lo largo de todas las rondas. En este modelo no asume ninguna distribución específica de los datos, al contrario, se considera que los datos podrían haber sido seleccionados por un adversario que quiere que el modelo falle.

"Active Learning" **(Aprendizaje activo):** En este enfoque, el sistema elige activamente los datos a etiquetar, recibe un conjunto de datos de entrenamiento, el modelo interroga a un experto humano y solicita etiquetas para ciertos puntos específicos. El objetivo es aprender de la información, pero con menos observaciones etiquetadas, que ahorra tiempo y dinero cuando el etiquetado es costoso.

7.2. Librería Scikit learning

Scikit-Learn es una de las bibliotecas más utilizadas para la implementación de algoritmos de "*machine learning"*. Se caracteriza por su API limpia, uniforme y fácil de usar, lo que facilita a usuarios, aplicar modelos de forma rápida y efectiva (Pedregosa et al., 2011), está diseñado para facilitar la representación estructurada. Los datos de entrada se organizan en una matriz de características (X), donde una fila representa cada muestra u observación recolectada y las columnas representan las características de cada muestra, (es importante puntualizar que estos valores deben ser cuantitativos para poder ingresar al entrenamiento de un modelo), por tanto, la matriz tiene la forma *[n_muestras, n_características]*. El otro componente es un vector objetivo (y) para representar la variable que se desea predecir, el cual es unidimensional con longitud igual al número de muestras, y puede contener valores numéricos continuos (en regresión) o etiquetas discretas (en clasificación) (VanderPlas, 2016).

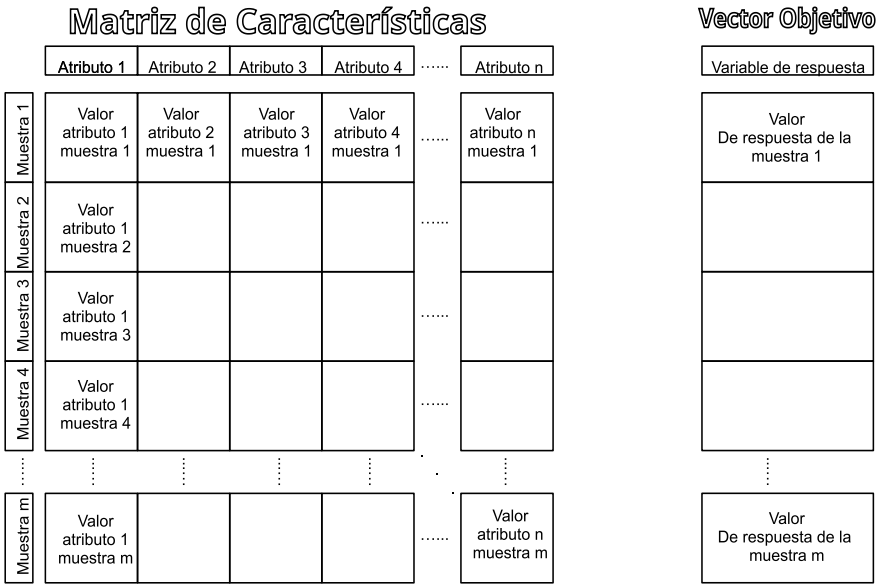


Figura 7.3. Esquema de la matriz de características y el vector de la variable de salida.

Fuente: elaboración propia.

Scikit-Learn se apoya en librerías, como *NumPy*, que proporciona estructuras de datos eficientes y operaciones matemáticas básicas, también utiliza *SciPy* para realizar cálculos de álgebra lineal y estadística avanzada, por último, *Cython* integra código C/C++ con sintaxis cercana a Python para mejorar el rendimiento (Pedregosa et al., 2011). Se instala utilizando el comando *conda install scikit learning* desde el *promt* de anaconda, luego se activa el entorno virtual en el que se desea instalar la librería.

Una ventaja de Scikit-Learn frente a otras bibliotecas es su fácil instalación, portabilidad y licencia permisiva (BSD). Además, su diseño imperativo (orientado a pasos secuenciales), siendo accesible para investigadores y

científicos de datos en áreas como biología, física, economía, entre otras disciplinas fuera de la informática(Pedregosa et al., 2011).

7.3. Preparación de datos

7.3.1. Recolección y preprocesamiento

Los datos se encuentran en las bases de datos en formato de archivos csv o en Excel. Para importar estos datos, se utiliza la librería *pandas*, mediante sus funciones como *read_csv()* y *read_excel()*. En estos archivos las tablas tienen columnas que representan variables y filas que corresponden a las muestras individuales.

Las bases de datos se constituyen por información heterogénea como números, categorías, fechas, descripciones, etc. Para un adecuado desempeño de los modelos de aprendizaje, se recomienda que los datos estén organizados en formato tabular, sin inconsistencias, con nombres claros de columnas y sin redundancias. Además, las columnas con valores categóricos en cadenas se sugieren transformar en valores numéricos, estas transformaciones garantizan que los modelos interpreten de manera adecuada las categorías. En la librería *scikit learning* existen codificadores que convierten los datos categóricos en numéricos:

- *OneHotEncoder* crea variables binarias por categoría
- OrdinalEncoder asigna valores enteros ordenados
- *FeatureHasher* codificación más compacta para datos de alta dimensionalidad.

Un inconveniente que surge en las bases de datos son los vacíos o datos faltantes (Variables como precipitación o radiación solar pueden tener vacíos en los registros), una solución es utilizar herramientas como *SimpleImputer*, que reemplaza valores faltantes con el valor promedio, mediana o moda de las observaciones, o *KNNImputer*, que estima valores a partir de los vecinos más cercanos.

Completada la información, se realiza una estandarización de los datos, las magnitudes de las variables por lo general son distintas, por ejemplo, la temperatura puede estar en grados centígrados con valores que van de 0 a 30, mientras que la radiación puede tener valores de 500 hasta 800 W/m^2. Se considera una escalación a los datos para asegurar que todas las características de entrada tengan el mismo peso al momento de determinar la variable de salida. *Scikit learn* tiene funciones para esta tarea:

- *StandardScaler* que realiza la transformación estadística con características de Distribución Normal Estándar de media cero y varianza unitaria.

- *MinMaxScaler* transforma en una escala entre 0 y 1 considerando el valor máximo y mínimo que tienen las muestras en cada atributo.

- *RobustScaler* este escalador centra los valores restando la mediana a cada dato y escala los datos según el rango de cuartiles. El centrado y el escalado se realizan de forma independiente en cada característica, calculando las estadísticas relevantes de la muestra del conjunto de entrenamiento, la mediana es una medida robusta cuando existe presencia de valores atípicos(Scikit-learn Development Team, 2024).

- *Normalizer* ajusta cada observación a una norma unitaria.

7.3.2. Reducción de dimensionalidad

El Análisis de Componentes Principales (PCA) es una técnica estadística de reducción de dimensionalidad que se utiliza para transformar un conjunto de variables correlacionadas en un nuevo conjunto de variables no correlacionadas, llamadas componentes principales. Elegir el número adecuado de componentes es importante para mantener la información más relevante sin introducir ruido innecesario. La cantidad óptima de componentes principales (*EOFs o PCs*) depende del equilibrio entre retención de información y simplificación del modelo, la manera más sencilla es determinar la varianza acumulada que se requiere retener (se recomienda mayor a 80%):

```
def get_number_eof(X,cumVar):

 ### Se realiza los pca

pca = PCA().fit(X)

var_explained = pca.explained_variance_ratio_ #proporción de la varianza explicada por cada componente principal.

 cum_var = var_explained.cumsum() #calcula la suma acumulativa de la varianza explicada

n_eof = np.where(cum_var > cumVar)[0].min() + 1 # Donde la varianza representa más del 90%
```

```
plt.figure(figsize=(10, 5))

 plt.plot(np.arange(1,n_eof+1),cum_var[0:(n_eof)])

plt.scatter(np.arange(1,n_eof+1),cum_var[0:(n_eof)])

plt.xlabel('Number of EOFs')

plt.ylabel('Cumulative Proportion of Variance Explained')

plt.grid()

plt.title('{} EOF Retained'.format(n_eof))

plt.show()

return n_eof
```

Este código facilita el ingreso de una matriz de características *X* y el valor de la varianza acumulada a través de la variable de entrada *cumVar*. El resultado es una cantidad entera, es decir, el número de funciones ortogonales que retienen la varianza acumulada explicada:

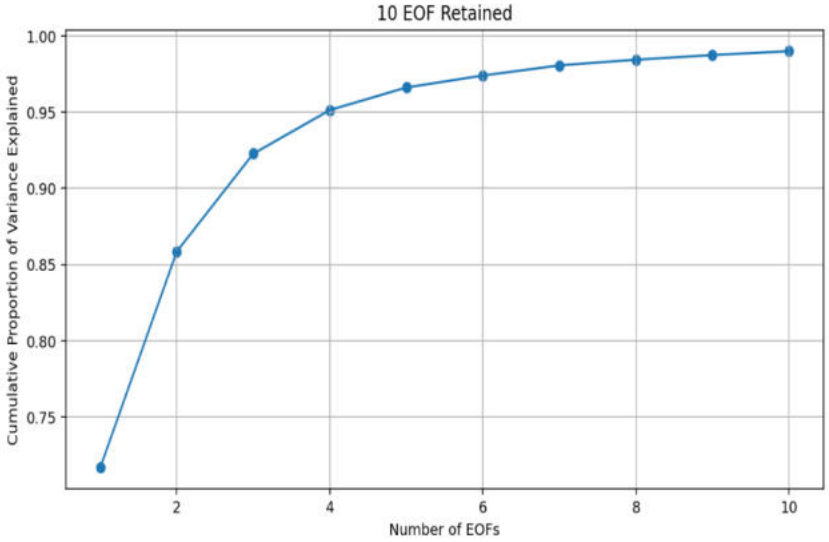


Figura 7.4. Varianza según el número de funciones ortogonales, con límite de 99% de varianza acumulada.

Fuente: elaboración propia

Se determina el número óptimo de componentes que contengan la mayor cantidad de información, la función se utiliza importándola al código mediante *from sklearn.decomposition import PCA.* Para aplicarlo se realiza:

- Se crea el modelo PCA ingresando el numero óptimo *n_eof*

  pca = PCA(n_components=n_eof)

- Aplicar en el análisis PCA en los datos (X)

  model_PCA = pca.fit(X)

- Transformar la base de datos y la salida será el conjunto nuevo con menos columnas

  arr_PCA = model_PCA.transform(X)

7.3.3. Partición del conjunto de datos

Para empezar a crear y entrenar un modelo es una práctica obligatoria dividir el *dataset* en dos subconjuntos: entrenamiento y prueba. Para entender esta división, imagina la realización de un examen, en el cual se quiere evaluar los conocimientos de un alumno en la materia de estadística. Para preparar al alumno se le envía un cuestionario de 10 preguntas, luego el día de la evaluación en el examen se toman dos preguntas exactamente iguales al cuestionario. En este caso la evaluación pierde credibilidad, en primer lugar, no se sabe si el alumno solo memorizo las respuestas y en segundo lugar no se asevera que el alumno realmente podrá resolver otros ejercicios fuera del cuestionario, no se infiere que el alumno realmente aprendió. Algo similar ocurre con los modelos ML, en este caso no tiene sentido entrenar al modelo con un conjunto de datos y evaluar su efectividad con una parte del mismo conjunto. Por esta razón, es necesario que se extraiga un conjunto de entrenamiento y un conjunto de evaluación, en general esta partición suele ser de 80% y 20% respectivamente.

Scikit-learn facilita la tarea de partición mediante la función *train_test_split*, que divide los datos de forma aleatoria, garantizando que los resultados de la evaluación reflejen el comportamiento real del modelo. El comando se importa mediante *from sklearn.model_selection import train_test_split,* la función se describe a continuación:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(

X, y, test_size=0.2, random_state=42, stratify=y)
```

X: las variables independientes (matriz de características).

y: la variable objetivo en el caso de modelos supervisados (vector con la variable de respuesta).

test_size: proporción para la división del conjunto. Ejemplo: 0.2 significa que se extrae el 20% de los datos para que sea el conjunto de evaluación, mientras el 80% restante será para el entrenamiento. También se pasa un entero con el número exacto de muestras.

random_state: es una semilla de datos para que la partición siempre sea igual.

shuffle: por defecto True, mezcla los datos antes de dividirlos.

stratify: En problemas de clasificación es aconsejable usar el argumento *stratify=y* para mantener la proporción de clases en ambos conjuntos.

La salida de la función será dos sub matrices *X_train, X_test* y dos *vectores y_train, y_test,* para el entrenamiento y la evaluación respectivamente.

7.3.4. Ejemplo

En este ejemplo, se trabaja con el *dataset Ames Housing Dataset*, que contiene información sobre más de 70 variables descriptivas sobre

viviendas en Ames, Iowa. Para importar esta base de datos se usa el comando *from sklearn.datasets import fetch_openml,* se asigna la información a una variable y luego se accede a la matriz de características y el vector de la variable de respuesta mediante *.data* y *.target*, respectivamente:

```
from sklearn.datasets import fetch_openml

import pandas as pd

# Cargar dataset Ames Housing

housing = fetch_openml(name="house_prices", as_frame=True)

X = housing.data

y = housing.target
```

Se revisan las características de los atributos en X mediante los comandos de pandas *X.info(),* resultando en*:*

```
<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1460 entries, 0 to 1459

Data columns (total 80 columns):

 #   Column        Non-Null Count  Dtype

---  ------        --------------  -----

0   Id            1460 non-null   int64
```

1 MSSubClass 1460 non-null int64

2 MSZoning 1460 non-null object

3 LotFrontage 1201 non-null float64

4 LotArea 1460 non-null int64

5 Street 1460 non-null object

6 Alley 91 non-null object

7 LotShape 1460 non-null object

8 LandContour 1460 non-null object

9 Utilities 1460 non-null object

10 LotConfig 1460 non-null object

11 LandSlope 1460 non-null object

12 Neighborhood 1460 non-null object

13 Condition1 1460 non-null object

14 Condition2 1460 non-null object

15 BldgType 1460 non-null object

16 HouseStyle 1460 non-null object

17 OverallQual 1460 non-null int64

18 OverallCond 1460 non-null int64

19 YearBuilt 1460 non-null int64

20 YearRemodAdd 1460 non-null int64

21 RoofStyle 1460 non-null object

22 RoofMatl 1460 non-null object

23 Exterior1st 1460 non-null object

24 Exterior2nd 1460 non-null object

25 MasVnrType 1452 non-null object

26 MasVnrArea 1452 non-null float64

27 ExterQual 1460 non-null object

28 ExterCond 1460 non-null object

29 Foundation 1460 non-null object

30 BsmtQual 1423 non-null object

31 BsmtCond 1423 non-null object

32 BsmtExposure 1422 non-null object

33 BsmtFinType1 1423 non-null object

34 BsmtFinSF1 1460 non-null int64

35 BsmtFinType2 1422 non-null object

36 BsmtFinSF2 1460 non-null int64

37 BsmtUnfSF 1460 non-null int64

38 TotalBsmtSF 1460 non-null int64

39 Heating 1460 non-null object

40 HeatingQC 1460 non-null object

41 CentralAir 1460 non-null object

42 Electrical 1459 non-null object

43 1stFlrSF 1460 non-null int64

44 2ndFlrSF 1460 non-null int64

45 LowQualFinSF 1460 non-null int64

46 GrLivArea 1460 non-null int64

47 BsmtFullBath 1460 non-null int64

48 BsmtHalfBath 1460 non-null int64

49 FullBath 1460 non-null int64

50 HalfBath 1460 non-null int64

51 BedroomAbvGr 1460 non-null int64

52 KitchenAbvGr 1460 non-null int64

53 KitchenQual 1460 non-null object

54 TotRmsAbvGrd 1460 non-null int64

55 Functional 1460 non-null object

56 Fireplaces 1460 non-null int64

57 FireplaceQu 770 non-null object

58 GarageType 1379 non-null object

59 GarageYrBlt 1379 non-null float64

60 GarageFinish 1379 non-null object

61 GarageCars 1460 non-null int64

62 GarageArea 1460 non-null int64

63 GarageQual 1379 non-null object

64 GarageCond 1379 non-null object

65 PavedDrive 1460 non-null object

66 WoodDeckSF 1460 non-null int64

67 OpenPorchSF 1460 non-null int64

68 EnclosedPorch 1460 non-null int64

69 3SsnPorch 1460 non-null int64

70 ScreenPorch 1460 non-null int64

71 PoolArea 1460 non-null int64

72 PoolQC 7 non-null object

73 Fence 281 non-null object

74 MiscFeature 54 non-null object

75 MiscVal 1460 non-null int64

76 MoSold 1460 non-null int64

77 YrSold 1460 non-null int64

78 SaleType 1460 non-null object

79 SaleCondition 1460 non-null object

dtypes: float64(3), int64(34), object(43)

memory usage: 912.6+ KB

La base de datos tiene un total de 80 atributos y 1460 observaciones. Se determina que existen variables que tienen datos faltantes, es recomendable eliminar las variables que tienen más de 50% de valores faltantes, utilizando el método *drop*.

L = ['Alley','FireplaceQu','PoolQC','Fence','MiscFeature']

```
X = X.drop(columns=L)
```

Se observa variables categóricas tipo *object* los cuales son cadenas de texto. Sin embargo, los valores categóricos se codifican con números. Para evitar este inconveniente se utilizará la estrategia de la moda a través de '*most frequent*'[4].

```
# Codificación

from sklearn.preprocessing import OrdinalEncoder

enc = OrdinalEncoder()

enc.fit(X)

new_X = enc.transform(X)

# Imputación de datos

from sklearn.impute import SimpleImputer

imputador = SimpleImputer(missing_values=np.nan, strategy='most frequent')

imputador.fit(new_X)

new_X = imputador.transform(new_X)
```

[4] Existe una alternativa más avanzada para poder asignar el tipo de imputación que se realiza en cada columna a través de la creación de un *pipeline*, en este libro no se tratará sobre esta herramienta, sin embargo, se presenta esta nota al pie para que el lector con curiosidad pueda consultar más al respecto en la documentación de la librería *sci-kit learning* en el siguiente enlace: https://scikit-learn.org/stable/modules/compose.html

El nuevo arreglo *Numpy.Array* creado se utiliza en el análisis de componentes principales, se ejecuta la función para buscar el número adecuado de vectores ortogonales con una varianza acumulada mayor a 80%, dando como resultado que se considera 7 componentes principales para el modelo PCA (Ver Figura 7.5). Se aplica el análisis de componentes principales a la matriz de características.

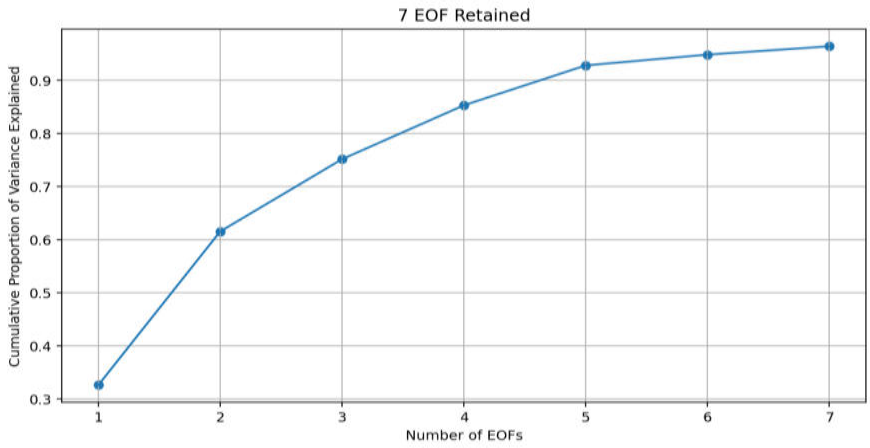


Figura 7.5. Funciones ortogonales usando get_number_eof(new_X,0.95).

Fuente: elaboración propia.

La matriz reducida resultante del PCA se la utiliza para realizar un análisis exploratorio de las características de la información.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=7)

# Aplicar en el análisis PCA en los datos new_X
```

model_PCA = pca.fit(new_X)

new_Xreducido= model_PCA.transform(new_X)

X_train, X_test, y_train, y_test = train_test_split(

new_Xreducido, y, test_size=0.2, random_state=42)

7.4. Selección del modelo

La selección del modelo depende de dos factores principales: las características de la base de datos (tipo de variables, número de observaciones, presencia de ruido, dimensionalidad) y el objetivo del estudio (predicción, clasificación, segmentación, reducción de dimensión).

7.4.1. Aprendizaje supervisado

7.4.1.1. Regresión lineal

La regresión lineal es uno de los modelos más sencillos y utilizados para predecir una variable continua. Su base matemática se centra en ajustar una recta (o hiperplano en dimensiones superiores) que minimice la suma de los errores cuadráticos entre los valores observados y los valores predichos (Mohri et al., 2018):

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

Donde el modelo busca encontrar cuales son los coeficientes a_n para cada característica que minimice el error. En *scikit-learn*, se implementa con el método *LinearRegression* de *sklearn.linear_model*.

```
from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

import numpy as np

# Datos de ejemplo

X = np.array([[1], [2], [3], [4], [5]])

y = np.array([1.2, 2.3, 2.9, 4.2, 5.1])

# Crear y entrenar modelo

model = LinearRegression()

model.fit(X, y)

# Predicciones

y_pred = model.predict(X)

# Gráfico

plt.scatter(X, y, color='blue', label='Datos reales')

plt.plot(X, y_pred, color='red', label='Regresión lineal')

plt.legend()

plt.show()
```

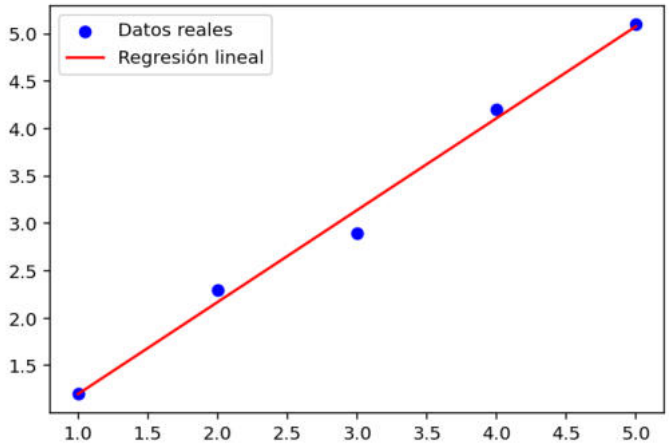


Figura 7.6. Regresión lineal.

Fuente: elaboración propia.

7.4.1.2. Árbol de decisión

Los árboles de decisión dividen recursivamente el espacio de los datos en regiones homogéneas. Matemáticamente, el algoritmo selecciona en cada nodo la variable y el umbral que maximizan la reducción de impureza (medida con entropía o índice Gini en clasificación, y varianza en regresión)(Scikit-learn, 2017). En *scikit-learn*, se implementa con *DecisionTreeClassifier* o *DecisionTreeRegressor*.

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.datasets import load_iris

import matplotlib.pyplot as plt

iris = load_iris()
```

```
X, y = iris.data, iris.target

tree = DecisionTreeClassifier(max_depth=3)

tree.fit(X, y)

plt.figure(figsize=(10,6))

plot_tree(tree, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)

plt.show()
```

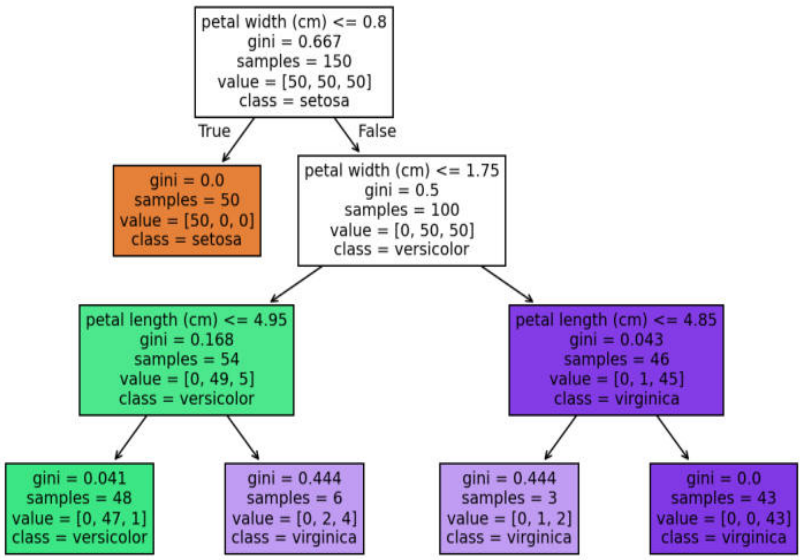


Figura 7.7. Diagrama del árbol de decisión generado.

Fuente: Autores, adaptado de scikit-learn developers (2025).

7.4.1.3. Random Forest

El *Random Forest* es un algoritmo de *ensemble learning* (aprendizaje por conjuntos) que combina múltiples árboles de decisión para producir un modelo más robusto y preciso. Fue propuesto por Leo Breiman y Adele Cutler y se ha convertido en uno de los métodos más populares dentro del aprendizaje supervisado (Mohri et al., 2018).

Se construyen muchos árboles de decisión independientes usando subconjuntos aleatorios de datos y características (*bootstrap aggregation o bagging*). Cada árbol realiza su predicción de manera individual. La predicción final se obtiene combinando los resultados, es decir, se elige la clase más votada por la mayoría de los árboles. Entre los hiper parámetros más importantes están:

- ***n_estimators:*** número de árboles en el bosque. Un valor alto mejora la precisión, pero aumenta el tiempo de cómputo.

- ***max_depth:*** profundidad máxima de cada árbol; controla la complejidad y previene sobreajuste.

- ***min_samples_split:*** número mínimo de muestras necesarias para dividir un nodo interno.

- ***min_samples_leaf:*** tamaño mínimo de cada nodo hoja; ayuda a suavizar el modelo.

- ***max_features:*** número máximo de características consideradas para dividir un nodo, lo que introduce aleatoriedad y mejora la generalización.

7.4.2. ML no supervisado

7.4.2.1. Clustering Kmean

El *clustering* consiste en agrupar observaciones de tal manera que las de un mismo grupo sean más similares entre sí que con las de otros grupos. El algoritmo *K-means* busca particionar los datos en k clústeres minimizando la suma de distancias cuadráticas a los centroides, la cual está definida matemáticamente como:

$$\min \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

En *scikit-learn* se usa este algoritmo importándolo desde la librería a través del comando *from sklearn.cluster import KMeans*. Entre los parámetros más relevantes de este método destacan:

- ***n_clusters:*** número de clústeres.
- ***init:*** método de inicialización de centroides (por defecto "k-means++").
- ***max_iter:*** número máximo de iteraciones.

A continuación, se cita el ejemplo propuesto por la documentación de *scikit-learning*, con el cual explicar porque la elección incorrecta del número de

clústeres puede llevar a una mala clasificación. Además, se crea un conjunto de datos que están distribuidos de distinta manera: Número no óptimo de clústeres, puntos con distribución anisotrópica, varianza desigual y puntos de tamaño desigual

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans

n_samples = 1500

random_state = 170

transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]

X, y = make_blobs(n_samples=n_samples, random_state=random_state)

X_aniso = np.dot(X, transformation)  # Anisotropic blobs

X_varied, y_varied = make_blobs(

n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], random_state=random_state

)  # Unequal variance

X_filtered = np.vstack(
```

```
(X[y == 0][:500], X[y == 1][:100], X[y == 2][:10])
)  # Unevenly sized blobs
y_filtered = [0] * 500 + [1] * 100 + [2] * 10
common_params = {
"n_init": "auto",
"random_state": random_state,
}
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(12, 12))
y_pred = KMeans(n_clusters=2, **common_params).fit_predict(X)
axs[0, 0].scatter(X[:, 0], X[:, 1], c=y_pred)
axs[0, 0].set_title("Número no óptimo de clusters")
y_pred = KMeans(n_clusters=3, **common_params).fit_predict(X_aniso)
axs[0, 1].scatter(X_aniso[:, 0], X_aniso[:, 1], c=y_pred)
axs[0, 1].set_title("Puntos con distribución anisotrópica")
y_pred = KMeans(n_clusters=3,
**common_params).fit_predict(X_varied)
axs[1, 0].scatter(X_varied[:, 0], X_varied[:, 1], c=y_pred)
```

```
axs[1, 0].set_title("Varianza desigual")

y_pred = KMeans(n_clusters=3,
**common_params).fit_predict(X_filtered)

axs[1, 1].scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_pred)

axs[1, 1].set_title("Grupos de tamaño desigual")

plt.suptitle("Unexpected KMeans clusters").set_y(0.95)

plt.show()
```

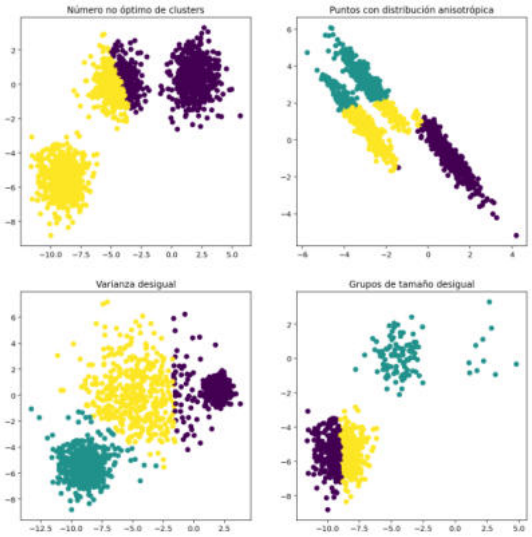


Figura 7.8 Clasificación utilizando KMeans.

Fuente: Autores, adaptado de scikit-learn developers (2025).

7.4.2.2. Minibatch Kmeans

El *MiniBatch K-means* es una variante de *K-means* que utiliza lotes pequeños de datos para actualizar los centroides en cada iteración. Esto reduce el costo computacional y facilita el trabajo con grandes volúmenes de datos, sacrificando ligeramente la precisión.

Parámetros principales son *batch_size*, el cual determina el tamaño del lote y *max_iter* para establecer el número máximo de iteraciones. Se utilizan los mismos datos del ejemplo anterior de *KMeans* adaptando al *nuevo modelo.*

```
from sklearn.cluster import MiniBatchKMeans

common_params = {

    "n_init": "auto",

    "random_state": random_state,

    "batch_size": 50,

}

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(12, 12))

y_pred = MiniBatchKMeans(n_clusters=2, **common_params).fit_predict(X)

axs[0, 0].scatter(X[:, 0], X[:, 1], c=y_pred)

axs[0, 0].set_title("Número no óptimo de clusters")
```

```
y_pred = MiniBatchKMeans(n_clusters=3,
**common_params).fit_predict(X_aniso)

axs[0, 1].scatter(X_aniso[:, 0], X_aniso[:, 1], c=y_pred)

axs[0, 1].set_title("Puntos con distribución anisotrópica")

y_pred = MiniBatchKMeans(n_clusters=3,
**common_params).fit_predict(X_varied)

axs[1, 0].scatter(X_varied[:, 0], X_varied[:, 1], c=y_pred)

axs[1, 0].set_title("Varianza desigual")

y_pred = MiniBatchKMeans(n_clusters=3,
**common_params).fit_predict(X_filtered)

axs[1, 1].scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_pred)

axs[1, 1].set_title("Grupos de tamaño desigual")

plt.show()
```

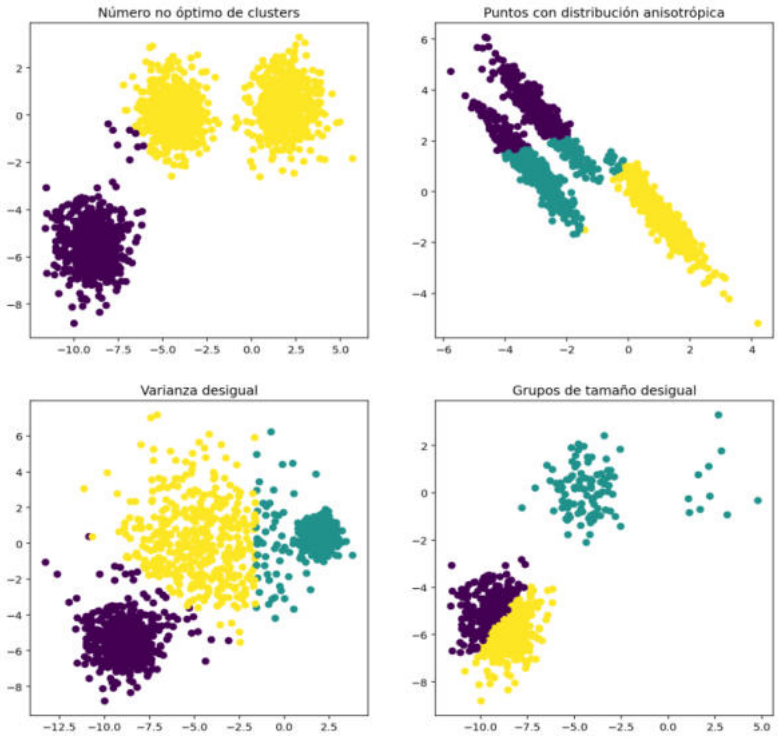


Figura 7.9. Clasificación utilizando MiniBatchKMeans.

Fuente: Autores, adaptado de scikit-learn developers (2025).

7.4.2.3. Otros modelos de clustering

La librería sci-kit learning tiene otros modelos de *clustering* que se pueden crear, entrenar y utilizar en los datos, se muestran a continuación:

Tabla 7.1. Descripción general de otros métodos de agrupamiento.

Nombre del método	**Parámetros**	**Escalabilidad**	**Caso de uso**	**Geometría (métrica usada)**
Propagación de	amortiguación (*damping*),	No escalable	Muchos clústeres, tamaños	Distancia en grafos (ej.

afinidad (*Affinity propagation*)	preferencia de muestra	con *n_samples*	desiguales, geometría no plana, inductivo	grafo de vecinos más cercanos)
Mean-Shift	ancho de banda (*bandwidth*)	No escalable con *n_samples*	Muchos clústeres, tamaños desiguales, geometría no plana, inductivo	Distancias entre puntos
Clustering espectral	número de clusters	*n_samples* medianos, *n_clusters* pequeños	Pocos clústeres, tamaños similares, geometría no plana, transductivo	Distancia en grafos (ej. grafo de vecinos más cercanos)
***Clustering* jerárquico de Ward**	número de clusters o umbral de distancia	*n_samples* y *n_clusters* grandes	Muchos clústeres, posible uso de restricciones de conectividad, transductivo	Distancias entre puntos
Clustering aglomerativo	número de clusters o umbral de distancia, tipo de enlace (*linkage*), distancia	*n_samples* y *n_clusters* grandes	Muchos clústeres, posible uso de restricciones de conectividad, distancias no euclidianas, transductivo	Cualquier distancia por pares
DBSCAN	tamaño del vecindario	Muy grande *n_samples*, *n_clusters* medianos	Geometría no plana, tamaños desiguales de clústeres, eliminación de valores atípicos, transductivo	Distancias entre puntos más cercanos
HDBSCAN	membresía mínima del clúster, vecinos mínimos por punto	*n_samples* grandes, *n_clusters* medianos	Geometría no plana, tamaños desiguales, eliminación de valores atípicos, transductivo, jerárquico, densidad variable de clústeres	Distancias entre puntos más cercanos

OPTICS	membresía mínima del clúster	Muy grande *n_samples*, *n_clusters* grandes	Geometría no plana, tamaños desiguales, densidad variable de clústeres, eliminación de valores atípicos, transductivo	Distancias entre puntos
Mezclas Gaussianas (*Gaussian mixtures*)	muchos	No escalable	Geometría plana, buena para estimación de densidad, inductivo	Distancias de Mahalanobis a los centros
BIRCH	factor de ramificación, umbral, *clusterer* global opcional	*n_clusters y n_samples* grandes	Conjuntos de datos muy grandes, eliminación de valores atípicos, reducción de datos, inductivo	Distancia euclidiana entre puntos
Bisecting K-Means	número de clústeres	Muy grande *n_samples*, *n_clusters* medianos	Propósito general, tamaños de cluster similares, geometría plana, sin clusters vacíos, inductivo, jerárquico	Distancias entre puntos

Fuente: Autores, adaptado de scikit-learn developers, (2025)

7.4.2.4.Seleccionando el número óptimo de clústeres

Existen varios métodos para determinar el número óptimo de *clústeres* en análisis de agrupamiento. En primer lugar, existe el método del Coeficiente de Silhouette que evalúa cuán bien está separado un punto de los puntos de otros *clústeres*. Se calcula con la fórmula

$$Silhouette\ Score\ =\ (b\ -\ a)\ /\ max(a, b)$$

Donde, a es la distancia promedio a los puntos del mismo *clúster* y b es la distancia promedio al *clúster* más cercano (al que no pertenece). Los valores de este parámetro van de $-1 < a < 1$, donde representa que está mal clasificado o bien clasificado, respectivamente. Una ventaja es, que no necesita conocer la forma de los *clústeres*, sin embargo, se reduce la confiabilidad si se integra muchos *clústeres* de diferente densidad.

Otro parámetro calculable es el Índice de Calinski-Harabasz (CH) o también conocido como Índice de Varianza. Este valor evalúa la dispersión *intracluster* vs. *intercluster* y se calcula mediante la siguiente expresión matemática:

$$CH = (Tr(Bk) \,/\, (k - 1)) \,/\, (Tr(Wk) \,/\, (n - k))$$

Donde, *Tr(Bk)* representa la dispersión entre *clústeres* y *Tr(Wk)* es la dispersión dentro del clúster para n muestras y con k número de *clústeres*. Este índice se interpreta revisando que mientras más alto el valor, mejor es la separación entre los *clústeres*.

De igual manera, existe el Índice de Davies-Bouldin (DB) el cual mide la similitud promedio entre cada *clúster* y los datos de la muestra más similar a él. Por esta razón se dice que mientras menor es el valor significa que el promedio de las similitudes es menor, por tanto, la clasificación es mejor. Este índice combina los conceptos de la dispersión dentro del *clúster* y la distancia entre *clústeres*.

Finalmente, el Método del Codo (Inercia), el cual es un proceso más visual y depende de la percepción de quien está creando el modelo. Utiliza la

inercia *intracluster*, es decir, la suma de distancias cuadradas de los puntos a sus centroides. Luego, estos valores se grafican para diferentes valores de k, se trata de escoger el punto donde la reducción de inercia se vuelve menos pronunciada (el "codo") indica el número óptimo de *clústeres*.

Se presenta un algoritmo creado para este libro el cual permite ingresar la matriz de características X y el valor máximo de *clústeres* que se quieren evaluar. Al final el código genera una gráfica donde el eje de x tiene el número de *clústeres* y el eje y de los subgráficos contienen los valores calculados de cada parámetro explicado en esta sección.

```
def clustering_metrics(X, max_k):
"""
Función que recibe la matriz bidimensional comprimida luego de los PCA para calcular distintos índices que permiten observar cual es el mejor número de  clusters para los datos
<- Array bidimensional
-> dict de resultados
    """
inertia = []
silhouette = []
calinski_harabasz = []
```

```
davies_bouldin = []

k_values = range(2, max_k + 1)

for k in k_values:

kmeans = KMeans(n_clusters=k, random_state=42).fit(X)

labels = kmeans.labels_

inertia.append(kmeans.inertia_)

silhouette.append(silhouette_score(X, labels))

calinski_harabasz.append(calinski_harabasz_score(X, labels))

davies_bouldin.append(davies_bouldin_score(X, labels))

 return {

 "k_values": k_values,

   "inertia": inertia,

   "silhouette": silhouette,

   "calinski_harabasz": calinski_harabasz,

   "davies_bouldin": davies_bouldin

   }
```

- Código para visualizar los resultados

```
def graficar_MetricasCluster(D):
    """
Recibe el diccionario generado con la función clustering_metrics para mostrar  de manera amigable al usuario los resultados
    """
 fig, axs = plt.subplots(2, 2,figsize=(10, 10))
 fig.tight_layout()
    axs[0,0].plot(D['k_values'],D['inertia'])
    axs[0,0].set_title('Inercia')
    axs[0,1].plot(D['k_values'],D['silhouette'])
    axs[0,1].set_title('silhouette')
    axs[1,0].plot(D['k_values'],D['calinski_harabasz'])
    axs[1,0].set_title('calinski_harabasz')
    axs[1,1].plot(D['k_values'],D['davies_bouldin'])
    axs[1,1].set_title('davies_bouldin')
    return
```

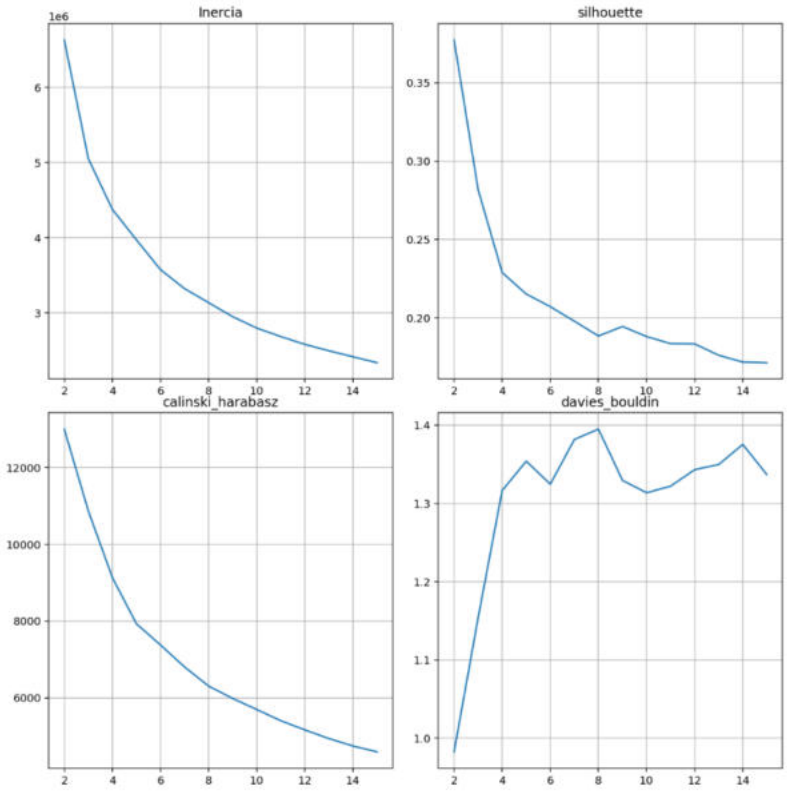


Figura 7.10. Índices para determinar el número óptimo de clústeres.

Fuente: elaboración propia.

7.5. Predicción y evaluación del modelo

En el ámbito de la meteorología y climatología, los modelos de ML se utilizan para predecir temperaturas extremas, probabilidades de lluvia, trayectorias de tormentas, o incluso para detectar patrones de cambio climático. Estos ejemplos muestran la importancia de contar con modelos precisos y confiables en la predicción de eventos naturales que logran gran impacto en la sociedad.

En scikit-learn, la predicción se realiza con el método *predict*, utilizando los datos de validación, se recibe de entrada un conjunto de datos nuevos y se devuelve las etiquetas o valores predichos.

El modelo ML configurado y entrenado con el conjunto de datos de entrenamiento (*X_train,y_train),* se realiza predicciones a través de ese modelo con los conjuntos (*X_valid,y_valid).*

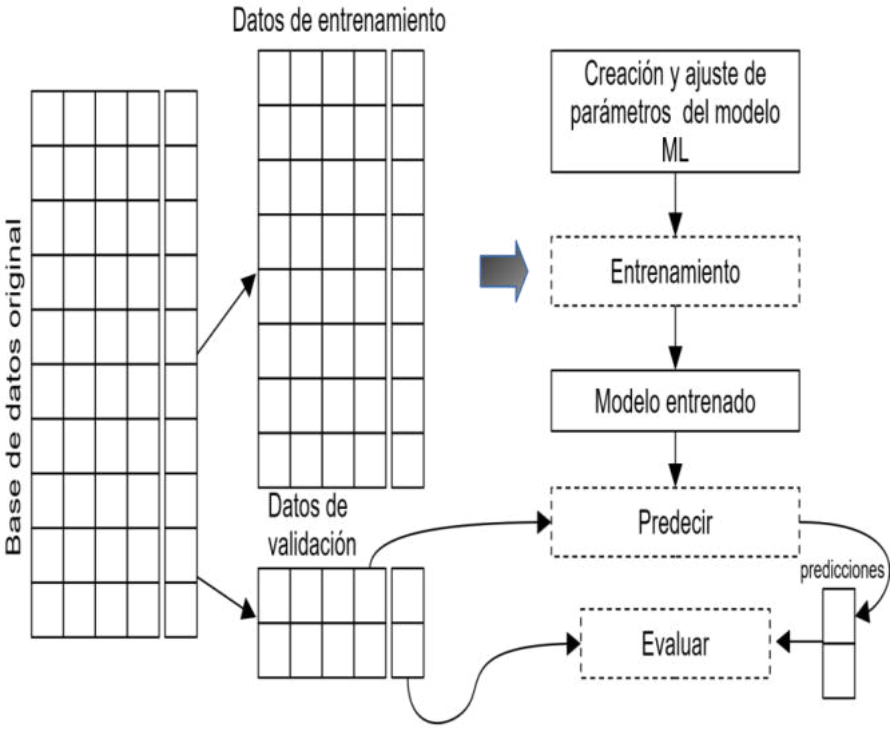


Figura 7.11. Proceso de entrenamiento y validación de un modelo.

Fuente: elaboración propia.

Los resultados obtenidos se evalúan mediante las siguientes de medición para determinar el ajuste de los datos estimados a los reales.

Tabla 7.2. Métricas de medición de regresión (para modelos supervisados).

Comando	Descripción
r2_score	Proporción de la varianza explicada por el modelo.
mean_squared_error	Error cuadrático medio, penaliza errores grandes.
mean_absolute_error	Error absoluto medio, mide la desviación promedio.
median_absolute_error	Error absoluto mediano, más robusto ante valores atípicos.
mean_pinball_loss	Función de pérdida para evaluar predicciones de cuantiles.

Fuente: elaboración propia.

Tabla 7.3. Métricas de medición de clasificación (para modelos no supervisados).

Comando	Descripción
accuracy_score	Proporción de instancias clasificadas correctamente.
precision_recall_fscore_support	Devuelve precisión, exhaustividad (recall) y F1-score.
roc_auc_score	Área bajo la curva ROC, mide la capacidad de discriminación del modelo.
log_loss	Pérdida logística, evalúa la incertidumbre.
balanced_accuracy_score	Cálculo la precisión ajustada para bases de datos desbalanceados.
confusion_matrix	Es una matriz que muestra los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos del modelo que estamos evaluando.
classification_report	Da acceso a un resumen completo de métricas por clase.

Fuente: elaboración propia.

CAPÍTULO VIII

ANÁLISIS DE DATOS CLIMÁTICOS CON PYTHON

En este capítulo se revisa un formato de almacenamiento específicamente desarrollado para información climática y atmosférica, los datos NetCDF. Se muestra las herramientas que facilitan el manejo de estos archivos, desde el procesamiento hasta el análisis visual de la información enfocados en la geoestadística.

8.1. Formato de almacenamiento NetCDF

NetCDF fue creado en la década de 1980 por Unidata, un programa de la University Corporation for Atmospheric Research (UCAR) en Estados Unidos, nació como una extensión y mejora del formato CDF de la NASA, que ofrecía una abstracción de datos multidimensionales, sin embargo, careciendo de ciertas características necesarias: independencia de plataforma, implementación en C, soporte para Unix y manejo eficiente de conjuntos masivos de datos. NetCDF solventa estas limitaciones ofreciendo una representación compacta, portable y transparente en red para datos científicos (Rew & Davis, 1990).

Su objetivo principal es proporcionar de forma eficiente, portable y auto-descriptiva el almacenamiento para compartir información compleja como series temporales, datos bidimensionales atmosféricos y resultados de modelos numéricos multidimensionales. NetCDF tiene ventaja con respecto a otro tipo de archivos como csv, porque su diseño es competente para trabajar con estructuras de datos de múltiples dimensiones, representando

fenómenos geofísicos que dependen de variables como el tiempo, la latitud, la longitud o la altura.

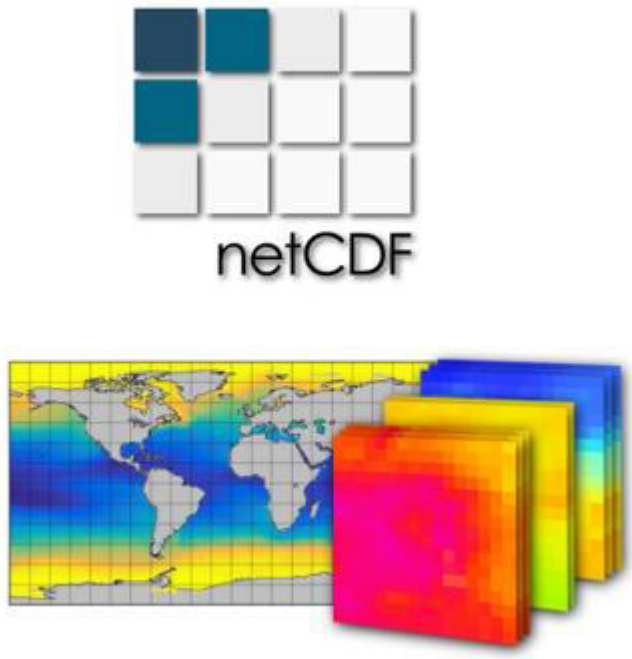


Figura 8.1. Archivos netCDF.

Fuente: Gis&Beer developers, (2018).

Los arreglos multidimensionales representan desde valores simples (escalares) hasta estructuras complejas de temperatura o humedad. Su característica esencial, un formato auto-descriptivo, incluye información adicional (metadatos) que explica su contenido, unidades y dimensiones, facilitando la interpretación y el intercambio de la información sin necesidad de depender de documentación externa (Rew et al., 1993). Actualmente, NetCDF es la base de muchos repositorios y proyectos

científicos, por ejemplo, se utiliza para almacenar y distribuir salidas de modelos climáticos globales generados en CMIP6 (Rew & Davis, 1990).

Un archivo NetCDF se organiza en tres componentes fundamentales que garantizan su carácter auto-descriptivo (Rew et al., 1993):

Dimensiones: representan magnitudes físicas como tiempo, latitud, longitud o nivel atmosférico, facilitando el hecho de agregar registros sin un tamaño predefinido (por ejemplo, nuevas observaciones de tiempo).

Variables: la variable es un arreglo de valores del mismo tipo (entero, flotante, carácter, etc.), descrito por un conjunto de dimensiones. Por ejemplo, la variable de temperatura ambiente, se declara como una matriz de dos dimensiones (latitud, longitud), mientras que para la humedad relativa se incluye una tercera dimensión para niveles de la atmósfera.

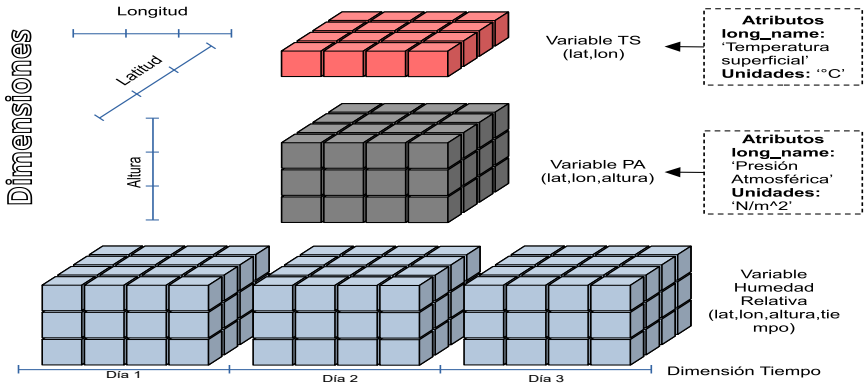


Figura 8.2. Estructura de los archivos netCDF.

Fuente: Gis&Beer developers, (2018).

Atributos: son pares clave-valor que almacenan información adicional, como global (título, historial o comentarios) o específicos de una variable (unidades, nombre largo, rango válido o valores para datos faltantes).

8.2. Librería xarray

X*array*, una librería de código abierto desarrollada inicialmente por Stephan Hoyer y mantenida por una comunidad activa de programadores(Xarray core developers, 2025). Inspirada en pandas, xarray extiende su filosofía a datos de n dimensiones, adoptando el modelo NetCDF y el Common Data Model propuesto por Unidata(Hoyer & Hamman, 2017).

Figura 8.3. Logotipo de la librería xarray.

Fuente: Xarray core developers (2025).

La característica autodescriptiva de las estructuras *xarray* reduce los errores, facilita la manipulación, se integra con bibliotecas como NumPy, Pandas, SciPy, Matplotlib y Dask, favoreciendo el análisis interactivo como procesamiento en paralelo de grandes volúmenes de datos (Hoyer & Hamman, 2017).

Las estructuras xarray son compatibles con la mayoría de las bases de datos distribuidos y con acceso remoto en la nube, incluso en climatología se suele

trabajar desde los catálogos de información. Otra gran ventaja es la capacidad de realizar operaciones vectorizadas con los valores de las variables.

Para instalar esta librería en un entorno de trabajo local se ejecuta desde el *promt* de Anaconda con el comando *conda install xarray,* de manera automática se descarga sus dependencias y actualización de otras librerías que se requieran. Cabe mencionar que para poder utilizar datos netcdf con xarray es necesario instalar otra librería con *conda install netcdf4*, que es una interfaz para la biblioteca C netCDF y concede la lectura y escritura de archivos con el formato netCDF4 y netCDF3.

Cuando se importa un archivo netCDF por primera vez es posible llamarlo o imprimirlo desde la consola de *IPython*, en donde se observan las distintas partes que conforman un archivo netCDF (Ver Figura 8.4).

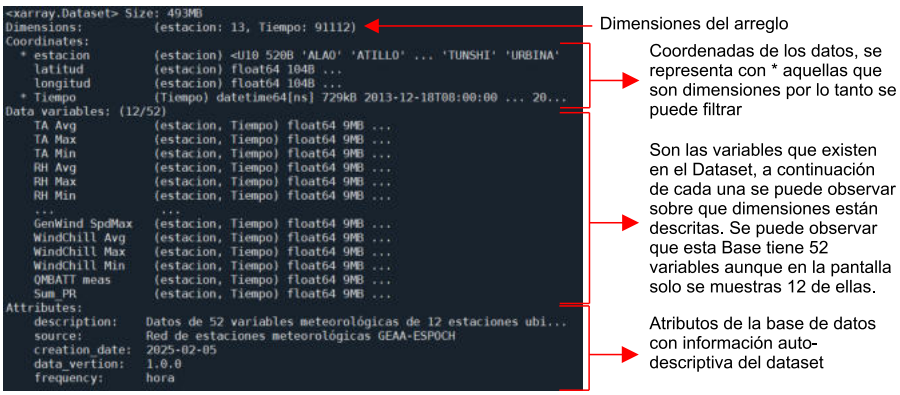


Figura 8.4. Datos netCDF en Python.

Fuente: elaboración propia

El *DataArray* es un arreglo multidimensional con dimensiones etiquetadas, coordenadas y metadatos, pero tiene la particularidad que solo posee la información de una variable. Es equivalente a una Series de *pandas*, pero extendida a múltiples dimensiones. Su principal componente son los valores guardados como un *numpy.ndarray,* tiene las dimensiones en las que están descritos los datos, las coordenadas y los atributos. Por esta razón, si se requiere crear un data array es necesario ingresar estos tres parámetros iniciales, como se muestra en el siguiente código:

```
import xarray as xr

import numpy as np

data = np.random.rand(4, 3)

da = xr.DataArray(data, dims=["time", "location"],

        coords={"time": ["2020-01", "2020-02", "2020-03", "2020-
04"],

            "location": ["Quito", "Ambato", "Riobamba"]},

        name="temperatura",

        attrs={"units": "°C"})
```

El resultado es el siguiente DataArray:

```
<xarray.DataArray 'temperatura' (time: 4, location: 3)> Size: 96B

array([[0.03813886, 0.21192121, 0.45577017],
```

```
       [0.47447336, 0.33511124, 0.63068058],
       [0.68640559, 0.60057318, 0.36868537],
       [0.03099789, 0.5381675 , 0.06282352]])
Coordinates:
  * time      (time) <U7 112B '2020-01' '2020-02' '2020-03' '2020-04'
  * location  (location) <U8 96B 'Quito' 'Ambato' 'Riobamba'
Attributes:
    units:    °C
```

El DataSet, es una colección de múltiples DataArray que comparten dimensiones, actuando como el equivalente multidimensional de un DataFrame, es útil para manejar diferentes variables meteorológicas o climáticas dentro de un mismo conjunto de datos. En el siguiente ejemplo se observa cómo la variable "temperatura está en función del tiempo y una locación, mientras que la variable presión esta descrita solo sobre la dimensión temporal:

```
ds = xr.Dataset(
    data_vars={
        "temperatura": (["time", "location"], np.random.rand(4, 3)),
        "presion": (["time"], np.random.rand(4) * 1000),
```

```
    },
    coords={
        "time": ["2020-01", "2020-02", "2020-03", "2020-04"],
        "location": ["Quito", "Ambato", "Riobamba"],
    },
    attrs={"descripcion": "Datos climáticos de ejemplo"}
)
```

El resultado es el siguiente:

```
<xarray.Dataset> Size: 336B
Dimensions:      (time: 4, location: 3)
Coordinates:
  * time         (time) <U7 112B '2020-01' '2020-02' '2020-03' '2020-04'
  * location     (location) <U8 96B 'Quito' 'Ambato' 'Riobamba'
Data variables:
    temperatura  (time, location) float64 96B 0.7567 0.1569 ... 0.4596 0.5544
    presion (time) float64 32B 116.0 926.7 488.6 825.6
Attributes: descripcion:  Datos climáticos de ejemplo
```

El arreglo *xarray* ejecutado con la información, se consulta las funciones que se describen:

Tabla 8.1. Métodos principales de DataArray y DataSet.

Comando	**Descripción**	**Ejemplo**
.dims	Muestra las dimensiones de un objeto DataArray o Dataset.	*da.dims devuelve ('time', 'lat', 'lon').*
.coords	Devuleve una lista de las coordenadas asociadas a las dimensiones (ej. tiempo, latitud, longitud).	*ds.coords* *[Out]: Coordinates: time, lat, lon.*
.attrs	Accede a los atributos o metadatos del objeto.	*da.attrs* *[Out]: {'units': 'K', 'long_name': 'Temperatura'}.*
.values	Extrae los valores como un arreglo de NumPy.	*da.values*
.shape	Muestra la forma (número de elementos en cada dimensión).	*da.shape* *[Out]: (365, 50, 50)*
.sizes	Devuelve un	*ds.sizes*

	diccionario con las dimensiones y su tamaño.	*[Out]: {'time': 365, 'lat': 50, 'lon': 50}.*
.sel()	Selección basada en etiquetas de coordenadas.	*ds.sel(time='2000-01-01', lat=0, lon=80)*
.isel()	Selección por índices enteros en lugar de etiquetas.	*da.isel(time=0, lat=10, lon=5)*
.mean()	Calcula la media a lo largo de una dimensión.	*da.mean(dim='time')*
.sum()	Suma los valores de una dimensión.	*da.sum(dim='lat')*
.max()	Encuentra el valor máximo.	*da.max(dim='lon')*
.min()	Encuentra el valor mínimo.	*da.min(dim='time')*
.groupby()	Agrupa datos según una coordenada.	*ds.groupby('time.month').mean( )*
.assign()	Agrega nuevas variables o modificar existentes.	*ds.assign(temp_C=ds.temp - 273.15)*
.rename()	Cambia nombres de variables o	*ds.rename({'temp':'temperatura'})*

	dimensiones.	
.transpose()	Reorganiza el orden de las dimensiones.	*da.transpose('lat', 'lon', 'time')*
.to_netcdf("archivo.nc")	Exporta un Dataset a un archivo NetCDF.	*ds.to_netcdf("clima.nc")*
xr.open_dataset("archivo.nc")	Abre un archivo NetCDF como Dataset.	*ds = xr.open_dataset("clima.nc")*

Fuente: elaboración propia.

De igual manera, la librería *xarray* mediante el uso de matplotlib proporciona funciones para representar los datos gráficamente, tales como:

Tabla 8.2. Métodos de DataArray y DataSet para graficar información.

Método	**Descripción**	**Ejemplo**
.plot()	Grafica un objeto DataArray de una o dos dimensiones. Selecciona automáticamente el tipo de gráfico adecuado (línea para 1D, mapa de calor para 2D).	da.plot()
.plot.line()	Crea gráficos de líneas para datos unidimensionales o para un punto específico en un conjunto multidimensional.	da.sel(lat=0, lon=80).plot.line()
.plot.imshow()	Muestra datos bidimensionales como una imagen (similar a plt.imshow).	da.isel(time=0).plot.imshow()
.plot.contour()	Grafica curvas de nivel de datos 2D.	da.isel(time=0).plot.contour()

.plot.contourf()	Grafica contornos sombreados (rellenos).	da.isel(time=0).plot.contourf(cmap="coolwarm")
.plot.hist()	Crea histogramas a partir de los datos.	da.plot.hist(bins=20)
.plot.scatter()	Grafica diagramas de dispersión (scatter plots) entre dos variables.	ds.plot.scatter(x="lon", y="lat", hue="temp")
.plot.pcolormesh()	Representa un arreglo 2D en forma de malla coloreada.	da.isel(time=0).plot.pcolormesh(cmap="viridis")
.plot.hist2d()	Genera histogramas bidimensionales (solo en datasets con dos variables).	ds.plot.hist2d(x="temp", y="precip")

Fuente: elaboración propia.

8.3. Conexión remota para la obtención de datos

El formato NetCDF se ha adoptado ampliamente en sistemas distribuidos y catálogos globales de datos climáticos mediante *Big Data*, donde se referencia a la gran cantidad de información que se genera en climatología. La utilización de la nube para almacenar y acceder a la información de forma remota, favoreciendo el procesamiento, sin necesidad de computadores de altas prestaciones.

Xarray da acceso a los datos remotos a través de los catálogos, es una herramienta que organiza y describe colecciones de datos de manera sistemática, proporcionando información sobre su contenido, formato, estructura y origen.

8.3.1. Pangeo y la librería Intake

Pangeo es una plataforma de código abierto diseñada para habilitar el análisis y visualización de grandes volúmenes de datos, especialmente

aquellos relacionados con climatología, medio ambiente y ciencias geofísicas. Pangeo concede a los investigadores la oportunidad de aprovechar la infraestructura en la nube para procesar y analizar grandes volúmenes de datos. Para acceder a los catálogos de *Pangeo*, se utiliza la biblioteca *intake,* diseñada para facilitar el acceso, exploración y carga de datos desde múltiples fuentes y formatos. Intake proporciona una interfaz sencilla para acceder a los catálogos distribuidos en la nube. Para instalar esta librería, se ejecuta en el *prompt* de anaconda el comando *conda install -c conda-forge intake.*

Se ha desarrollado un algoritmo para utilizar *Intake*, cargar el catálogo de datos en la nube de CMIP6 a través de Pangeo. El catálogo contiene información sobre conjuntos de datos climáticos que están disponibles para su análisis. Para ejecutar esta operación se utiliza el comando *intake.open_esm_datastore().* Una vez que el catálogo está cargado, *Intake* facilita la búsqueda y filtrado de datos según criterios específicos a través del método *search*().

```
def get_Cmip(var,experiment,resolución,ensamble):
    """
    Funcion que extrae los datos de cmip en modo xarray desde elcatalogo a través
    de intake
    """
```

```
    # Cargar el catálogo CMIP6 desde Pangeo
    col_url = "https://storage.googleapis.com/cmip6/pangeo-cmip6.json"
    col = intake.open_esm_datastore(col_url)
    # Filtrar los datos para la variable 'tas' (temperatura del aire cerca de la superficie)
    # y para datos diarios ('day')
    cat = col.search(
        experiment_id = [experiment],
        #source_id=['MPI-ESM1-2-HR'], # Modelo del Planck institute
        table_id = 'day',  # datos diarios
        variable_id = var,
        member_id = ensamble  # Seleccionando un conjunto de miembros de ensamble
    )
    # Verificar los resultados de búsqueda
    print(cat.df[['source_id', 'experiment_id', 'table_id', 'variable_id', 'grid_label', 'dcpp_init_year']])
```

```
    # Convertir el catálogo filtrado en un diccionario de conjuntos de datos
    dsets = cat.to_dataset_dict()
    # Filtrar conjuntos de datos que cumplen con la resolución espacial deseada
    filtered_dsets = {}
    desired_lat_res = resolucion  # Resolución espacial deseada en grados
    desired_lon_res = resolucion  # Resolución espacial deseada en grados
    for key, ds in dsets.items():
        lat_res = abs(ds.lat[1] - ds.lat[0]).values
        lon_res = abs(ds.lon[1] - ds.lon[0]).values
            if lat_res <= desired_lat_res and lon_res <= desired_lon_res:
          filtered_dsets[key] = ds
    # Suponga que selecciona el primer conjunto de datos en la lista
    try:
        key = list(filtered_dsets.keys())[0]
        #dsCmip = filtered_dsets[key]
        #print(filtered_dsets)
```

```
        return filtered_dsets
    except:
        print(filtered_dsets)
        return
```

La función recibe como parámetro de entrada *var* el cual es el identificador de la variable que se quiere buscar, como, por ejemplo:

- **tas:** Temperatura del aire en superficie (K)
- **tasmax, tasmin:** Temperatura máxima diaria /temperatura mínima diaria
- **pr:** Precipitación (kg m-2 s-1)
- **psl:** Presión atmosférica a nivel del mar (Pa)
- **uas, vas:** Componentes zonal y meridional del viento (m/s)
- **hurs:** Humedad relativa (%)
- **rsds, rsus:** Radiación solar descendente/ascendente en superficie (W/m2)

En la entrada *experiment* se ingresa '*historical',* se eligen los datos del pasado o para estimaciones futuras se ingresa el tipo de experimento:

- **ssp126:** Representa un escenario de bajas emisiones, con políticas climáticas fuertes y esfuerzos por mitigar el cambio climático.

- **ssp245:** Escenario de emisiones intermedias, con un enfoque equilibrado entre crecimiento económico y mitigación.

- **ssp370:** Escenario de altas emisiones, con políticas moderadas y algunos esfuerzos de mitigación.

- **ssp585:** Escenario de emisiones muy altas, sin esfuerzos significativos para reducir las emisiones.

De igual manera, se ingresa el miembro de *ensamble*, que representa las condiciones iniciales del experimento. Los miembros *ensamble* tienen su propia notación:

- **r (realization index):** Diferentes realizaciones con el mismo modelo y forzamiento, pero con condiciones iniciales distintas. Ejemplo: r1 y r2 usan el mismo modelo, pero comienzan con perturbaciones mínimas distintas en el estado inicial de la atmósfera.

- **i (initialization method index):** Diferentes métodos de inicialización del modelo. Ejemplo: i1, i2. Representan que tipo de métodos matemáticos usan para empezar la simulación como punto de partido, como reanálisis, esquemas estocásticos, etc.

- **p (physics index):** Cambios en la parametrización física del modelo, como convección o nubes. Los valores que se eligen son p1 y p2.

- **f (forcing index):** Los forzamientos externos representan condiciones como gases de efecto invernadero, aerosoles, uso de suelo, etc. Entre las versiones de los forzamientos externos aplicados tenemos f1 y f2.

Si se detecta un error en un forzamiento o se actualiza una base de datos, llega a generarse un nuevo conjunto con diferente f.

Finalmente, el último parámetro de la función es *resolución*, es un numero flotante que representa el valor de la resolución del píxel en función del radio, por ejemplo, un valor de 0.5 representa 25 Km. Este valor filtra y obtiene la información que tengan una resolución igual o menor al valor de referencia. La función retorna un diccionario con los *datasets* que cumplen todas las condiciones previamente seleccionadas.

A continuación, se mostrará un ejemplo donde se buscan *datasets* con datos estimados de precipitación en el escenario ssp245 y que tenga una resolución menor a 75Km. Esta búsqueda se la realiza en el *ensamble 'r1i1p1f2'*.

```
dic = get_Cmip('pr','ssp245',0.75,'r1i1p1f2')
```

El resultado en pantalla muestra un resumen de los *datasets* que están dentro del *ensamble,* el experimento y con información de la variable ingresada.

```
source_id experiment_id table_id variable_id grid_label  dcpp_init_year
0  CNRM-CM6-1       ssp245     day        pr        gr          NaN
1  CNRM-ESM2-1      ssp245     day        pr        gr          NaN
2  UKESM1-0-LL      ssp245     day        pr        gn          NaN
3  MIROC-ES2L       ssp245     day        pr        gn          NaN
4  EC-Earth3        ssp245     day        pr        gr          NaN
```

5 NorESM2-LM ssp245 day pr gn NaN

--> The keys in the returned dictionary of datasets are constructed as follows:

'activity_id.institution_id.source_id.experiment_id.table_id.grid_label'

| | 100.00% [6/6 00:26<00:00]

En el ejemplo el diccionario solo tiene un *dataset*, si se requiere extraerlo en un ds para imprimir sus características en el panel, es posible mediante:

```
ds = dic[list(dic.keys())[0]]

print(ds)

<xarray.Dataset> Size: 6GB

Dimensions:         (lat: 256, bnds: 2, lon: 512, member_id: 1,

                     dcpp_init_year: 1, time: 11323)

Coordinates:

  * lat             (lat) float64 2kB -89.46 -88.77 -88.07 ... 88.07 88.77 89.46

    lat_bnds        (lat, bnds) float64 4kB dask.array<chunksize=(256, 2),
meta=np.ndarray>

  * lon             (lon) float64 4kB 0.0 0.7031 1.406 ... 357.9 358.6 359.3
```

lon_bnds (lon, bnds) float64 8kB dask.array<chunksize=(512, 2), meta=np.ndarray>

** time (time) datetime64[ns] 91kB 2020-01-01T12:00:00 ... 2050-1...*

time_bnds (time, bnds) datetime64[ns] 181kB dask.array<chunksize=(11323, 2), meta=np.ndarray>

** member_id (member_id) object 8B 'r1i1p1f2'*

** dcpp_init_year (dcpp_init_year) float64 8B nan*

Dimensions without coordinates: bnds

Data variables:

pr (member_id, dcpp_init_year, time, lat, lon) float32 6GB dask.array<chunksize=(1, 1, 132, 256, 512), meta=np.ndarray>

Attributes: (12/65)

Conventions: CF-1.7 CMIP-6.2

activity_id: ScenarioMIP

branch_method: standard

branch_time_in_child: 60265.0

branch_time_in_parent: 60265.0

cmor_version: *3.5.0*

...

intake_esm_attrs:variable_id: *pr*

intake_esm_attrs:grid_label: *gr*

intake_esm_attrs:zstore: *gs://cmip6/CMIP6/ScenarioMIP/EC-Earth...*

intake_esm_attrs:version: *20201015*

intake_esm_attrs:_data_format_: *zarr*

intake_esm_dataset_key: *ScenarioMIP.EC-Earth-Consortium.EC-Ea...*

8.3.2. OPeNDAP para acceso remoto

OPeNDAP (por sus siglas en inglés, *Open-source Project for a Network Data Access Protocol*) es un protocolo que da acceso a los usuarios a datos multidimensionales de forma remota sin descargarlos a una máquina física (Fulker, 2016). El crecimiento de datos climáticos y atmosféricos, que alcanzan escalas de petabytes, el formato NetCDF ha sido adaptado para entornos de Big Data y plataformas de computación en la nube, evolucionando a una integración adecuada con este tipo de protocolos OPeNDAP. Esto facilita que los datos almacenados en servidores OPeNDAP puedan ser accedidos a través de la librería *xarray*.

Además, los principales proveedores de nube (AWS, Google Cloud, Microsoft Azure), integran grandes repositorios de datos climáticos en NetCDF, accesibles mediante APIs o catálogos abiertos. Esto facilita el análisis colaborativo, el uso de Jupyter Notebooks en entornos distribuidos, y la ejecución de modelos de predicción climática en infraestructura de alta disponibilidad(Fulker, 2016).

Se utiliza la librería xarray para manejar los datos satelitales CHIRTS, en el siguiente proceso:

1. Cargar los datos en formato xarray.Dataset desde el enlace OPeNDAP que se obtiene en la página oficial de columbia

(https://iridl.ldeo.columbia.edu/SOURCES/), siguiendo la ruta NOAA (National Oceanic and Atmospheric Administration) → NCEP (National Centers for Environmental Prediction) → CPC (Climate Prediction Center) → temperature→ daily→Daily Maximum Temperatura. En esta pantalla se elige la variable, luego en la sección “Data Files” seleccionando la opción OPeNDAP la página se traslada a una pantalla donde está el URL para acceder a los datos en la nube (ver Figura 8.5, 8.6 y 8.7).

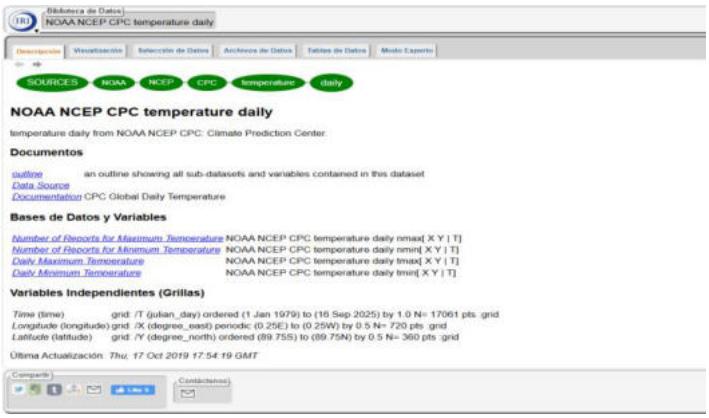


Figura 8.5. Página web para la descarga de datos CHIRTS.

Fuente: elaboración propia.

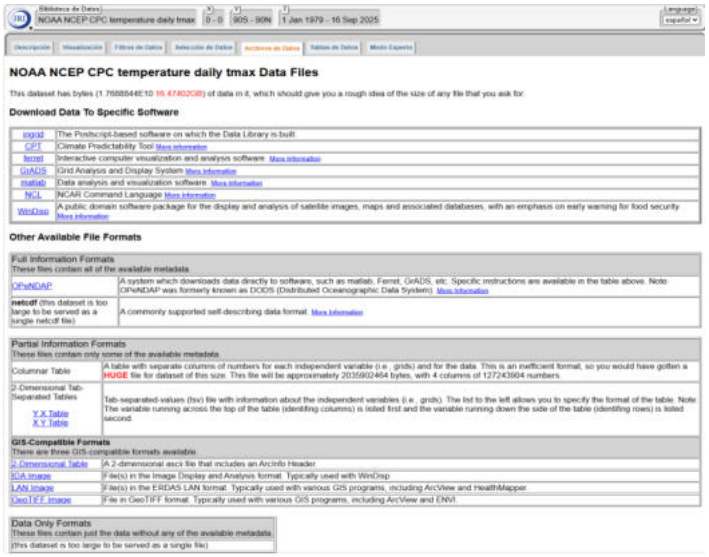


Figura 8.6. Selección del acceso remoto a los datos CHIRPS a través de OPeNDAP.

Fuente: elaboración propia.

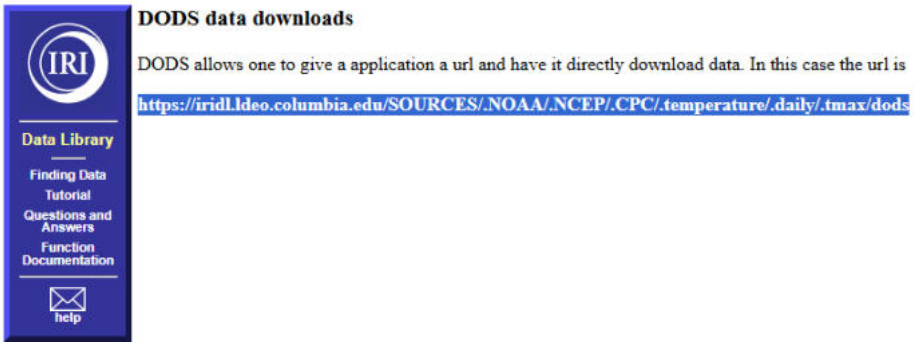


Figura 8.7. Página web para copiar la url que permite el acceso remoto a los datos.

Fuente: elaboración propia

2. El url se copia y se utiliza en el método de xarray llamado xr.open_dataset() de la siguiente manera:

```
dsChirts = xr.open_dataset('http://iridl.ldeo.columbia.edu/SOURCES/.UCSB/.CHIRTS/.v1.0/.daily/.global/.0p05/.tmax/dods')
```

El proceso de carga tardar unos minutos, pero el resultado es un objeto tipo *Dataset*, en el cual habilita el acceso a las diferentes variables, la aplicación de filtros, el cálculo de estadísticas y la manipulación de datos.

```
[Out]:

<xarray.Dataset> Size: 930GB

Dimensions:  (X: 7200, T: 12419, Y: 2600)

Coordinates:
```

** X (X) float32 29kB -180.0 -179.9 -179.9 -179.8 ... 179.9 179.9 180.0*

** T (T) float32 50kB 2.445e+06 2.445e+06 ... 2.458e+06 2.458e+06*

** Y (Y) float32 10kB -59.97 -59.92 -59.88 -59.82 ... 69.88 69.92 69.97*

Data variables:

tmax (T, Y, X) float32 930GB ...

Attributes:

Conventions: IRIDL

Si el usuario requiera acceder a todos los datos CHIRTS se necesitaría de 930 GB de espacio, que ocupa los datos diarios y a nivel mundial. Sin embargo, para el presente estudio se aplica filtros para seleccionar una región de estudio en un rango temporal.

8.3.3. Dask

La biblioteca *Dask,* está diseñada para facilitar el procesamiento paralelo y distribuido de grandes volúmenes de datos, en cálculos complejos que exceden la capacidad de la memoria de una sola máquina o requieren paralelización para aumentar la velocidad de procesamiento (Harris et al., 2020). Entender cómo funciona *Dask* requiere de la introducción del concepto de los *chunk* o "bloques". Un bloque es un subconjunto de cálculos e información que conforman un proceso global, el cual se quiere ejecutar sobre una base de datos masiva, de esta manera un algoritmo se ejecuta de

manera distribuida, ideal para infraestructuras como *clústeres* computacionales o entorno en la nube.

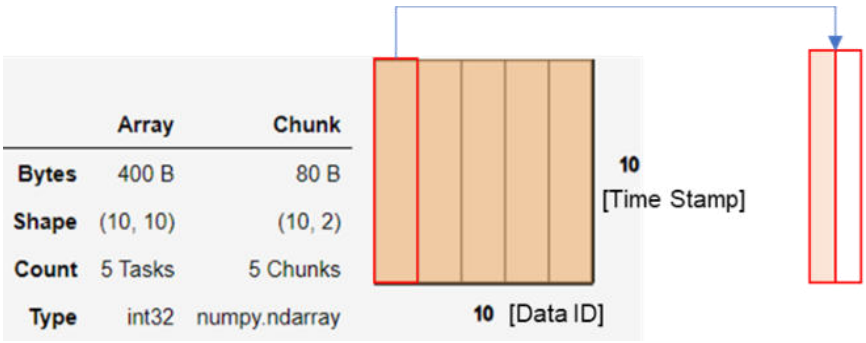


Figura 8.8. Aplicación de Dask para distribuir la ejecución de un algoritmo.

Fuente: Dask discourse group, (2021).

La instalación Dask, se realiza con el comando *conda install dask*. El principal objeto de trabajo de esta librería es el *dask.array,* que determina las dimensiones que caracterizan la partición de datos, es decir, los *chunks.*

8.4. Pre-Procesamiento de datos

La red de estaciones meteorológicas ESPOCH-INAMHI, transmiten información de variables meteorológicas, formando base de datos por segundo, por minuto o por hora, se resume las variables que se miden en cada una de estas estaciones:

Tabla 8.3. Abreviaturas e información de los datos de la red de estaciones meteorológicas ESPOCH-INAMHI.

Nº	ABREVIATURAS		SIGNIFICADOS
1	Stat_TA_1h	Avg	Temperatura Ambiental Promedio (cada hora)
		Max	Temperatura Ambiental Máxima (cada hora)
		Min	Temperatura Ambiental Mínima (cada hora)
2	Stat_RH_1h	Avg	Humedad Relativa Promedio (cada hora)
		Max	Humedad Relativa Máxima (cada hora)
		Min	Humedad Relativa Mínima (cada hora)
3	Stat_PA_1h	Avg	Presión del Aire o Atmosférica Promedio (cada hora)
		Max	Presión del Aire o Atmosférica Máxima (cada hora)
		Min	Presión del Aire o Atmosférica Mínima (cada hora)
4	Stat_SR_Dif_1h	Avg	Radiación Solar Difusa Promedio (cada hora)
		Max	Radiación Solar Difusa Máxima (cada hora)
		Min	Radiación Solar Difusa Mínima (cada hora)
5	Sum_SR_Dif_1h	Sum	Sumatoria de la hora de Radiación Solar Difusa.
6	Stat_SR_Glob_1h	Avg	Radiación Solar Global Promedio (cada hora)
		Max	Radiación Solar Global Máxima (cada hora)
		Min	Radiación Solar Global Mínimo (cada hora)
7	Sum_SR_Glob_1h	Sum	Sumatoria de la hora de Radiación Solar Global.

8	Stat_TS_1h_TG1	Avg	Temperatura de Suelo Promedio a nivel 1 (cada hora)
		Max	Temperatura de Suelo Máxima a nivel 1 (cada hora)
		Min	Temperatura de Suelo Mínima a nivel 1 (cada hora)
9	Stat_TS_1h_TG2	Avg	Temperatura de Suelo Promedio a nivel 2 (cada hora)
		Max	Temperatura de Suelo Máxima a nivel 2 (cada hora)
		Min	Temperatura de Suelo Mínima a nivel 2 (cada hora)
10	Stat_TS_1h_TG3	Avg	Temperatura de Suelo Promedio a nivel 3 (cada hora)
		Max	Temperatura de Suelo Máxima a nivel 3 (cada hora)
		Min	Temperatura de Suelo Mínima a nivel 3 (cada hora)
11	Stat_TS_1h_TG4	Avg	Temperatura de Suelo Promedio a nivel 4 (cada hora)
		Max	Temperatura de Suelo Máxima a nivel 4 (cada hora)
		Min	Temperatura de Suelo Mínima a nivel 4 (cada hora)
12	Stat_TS_1h_TG5	Avg	Temperatura de Suelo Promedio a nivel 5 (cada hora)
		Max	Temperatura de Suelo Máxima a nivel 5 (cada hora)
		Min	Temperatura de Suelo Mínima a nivel 5 (cada hora)
13	Stat_TS_1h_TG6	Avg	Temperatura de Suelo Promedio a nivel 6 (cada hora)

		Max	Temperatura de Suelo Máxima a nivel 6 (cada hora)
		Min	Temperatura de Suelo Mínima a nivel 6 (cada hora)
14	Stat_TS_1h_TG7	Avg	Temperatura de Suelo Promedio a nivel 7 (cada hora)
		Max	Temperatura de Suelo Máxima a nivel 7 (cada hora)
		Min	Temperatura de Suelo Mínima a nivel 7 (cada hora)
15	Sum_PR_1h	Sum	Precipitación de lluvia (Suma) cada hora
16	QMBATT_Meas	meas	Batería
17	GenWind_1h_SDI12	SpdMin	Velocidad de Viento (Mínima) cada hora
		WRun	Recorrido del viento
		DirAvg	Dirección del Viento (Promedio) cada hora
		DirMax	Dirección del Viento (Máxima) cada hora
		GustDir	Dirección de la racha de viento
		GustH	Hora de la racha de viento
		GustM	Minuto de la racha de viento
		SpdAvg	Velocidad de viento (Promedio)
		SpdMax	Velocidad de viento (Máxima)
18	Stat_WindChill_1h_SDI12	Avg	Sensación térmica (Promedio) cada hora
		Max	Sensación térmica (Máxima) cada hora
		Min	Sensación térmica (Mínima) cada hora

Fuente: elaboración propia.

Los datos se almacenan en archivos csv, se genera un archivo por estación y por año, ocupando un espacio adicional en la memoria del (Ver Figura 8.9). Los archivos csv no están digitalizados de manera compacta para que cualquier usuario pueda entender y analizar la información (Ver Figura 8.10).

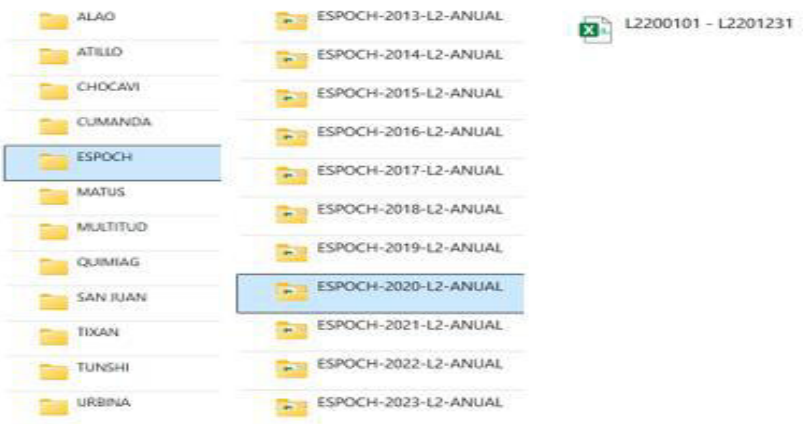


Figura 8.9. Almacenamiento de datos crudos de la red de estaciones.

Fuente: elaboración propia.

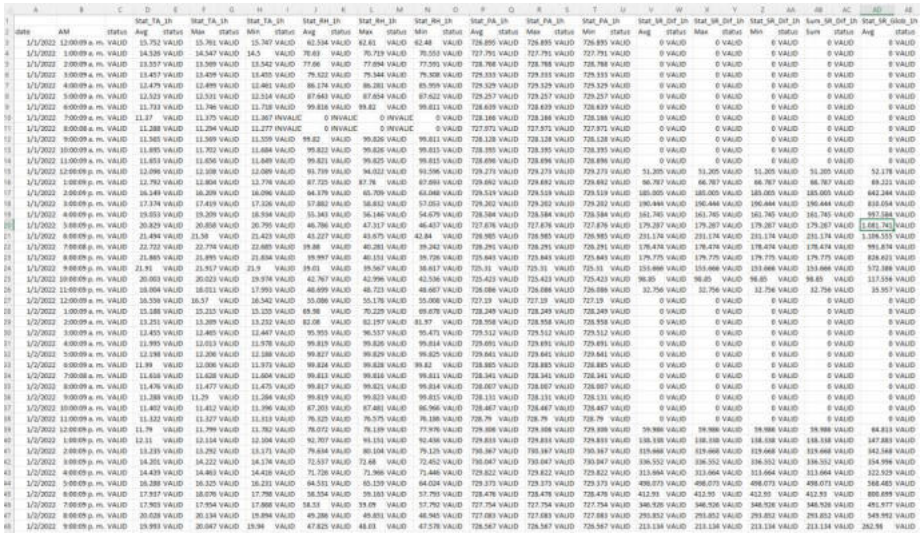


Figura 8.10. Datos crudos de la estación de la ESPOCH del año 2022.

Fuente: elaboración propia.

El proceso de importación, procesamiento, calidad y conversión de datos tiene limitaciones como acceso óptimo a los datos y optimización de memoria, para resolverlo se plantea la creación y configuración de una arquitectura de almacenamiento basado en NetCDF. Se detalla el proceso de transformación de csv a NetCDF:

8.4.1. Eliminación de fallas en los archivos csv

1. Se observa diferencias en la nomenclatura de las horas, donde en algunos csv las horas se encuentran con a.m. y en otros archivos con AM, se estandarizar para que se pueda interpretar la hora, en este caso se selecciona las siglas AM y PM. Otro problema común en varios archivos es que los valores decimales están representados con comas, esto representa un error porque los archivos csv utilizan las comas

para dividir en columnas la información (Figura 8.11). Se crea la función *`repair_File(link)`*, diseñada para reparar y estandarizar archivos de datos delimitados por punto y coma, para lo cual recibe la ruta del archivo que se quiere reparar. El código abre el archivo y separa cada línea en columnas; luego, localiza la columna correspondiente a la hora y corrige las marcas horarias mal escritas en minúsculas. Finalmente, convierte cada línea en una cadena delimitada por comas y sobrescribe el archivo original con esta nueva versión corregida. De este modo, la función no solo normaliza los formatos de tiempo y numéricos, sino que además convierte el archivo a un formato más consistente y compatible. El código descrito es:

```
def repair_File(link):
    with open(link, 'r',encoding='latin-1') as file: Lcads = file.readlines()
    data = [cad.split(';') for cad in Lcads]
    for i in range(len(data)):
        try:
            old_Cad = data[i][1]
            pmam = data[i][1][-5:]
            if pmam[0] == 'p':
                new_Cad = old_Cad.replace(pmam, 'PM')
            else:
```

```
            new_Cad = old_Cad.replace(pmam, 'AM')
        data[i][1] = new_Cad
    except:
        pass
    data[i] = [item.replace(',', '.') for item in data[i]]
    data[i] = ','.join(data[i])
with open(link, 'w') as file:
    for linea in data:
        file.write(linea)
return
```

date	AM	status	Avg	status	Max
1/1/2022	12:00:09 a. m.	VALID	15.752	VALID	15.761
1/1/2022	1:00:09 a. m.	VALID	14.526	VALID	14.547
1/1/2022	2:00:09 a. m.	VALID	13.557	VALID	13.569
1/1/2022	3:00:09 a. m.	VALID	13.457	VALID	13.459
1/1/2022	4:00:09 a. m.	VALID	12.479	VALID	12.499
1/1/2022	5:00:09 a. m.	VALID	12.523	VALID	12.531
1/1/2022	6:00:09 a. m.	VALID	11.733	VALID	11.746
1/1/2022	7:00:09 a. m.	VALID	11.37	VALID	11.375
1/1/2022	8:00:08 a. m.	VALID	11.288	VALID	11.294
1/1/2022	9:00:09 a. m.	VALID	11.565	VALID	11.569
1/1/2022	10:00:09 a. m.	VALID	11.695	VALID	11.702
1/1/2022	11:00:09 a. m.	VALID	11.653	VALID	11.656
1/1/2022	12:00:09 p. m.	VALID	12.096	VALID	12.108
1/1/2022	1:00:09 p. m.	VALID	12.792	VALID	12.804
1/1/2022	2:00:09 p. m.	VALID	16.149	VALID	16.209

date	time	status	Avg		status	Max	
1/24/14	3:00:08 p. m.	VALID	15	109	VALID	15	768
1/24/14	4:00:07 p. m.	VALID	15	193	VALID	15	653
1/24/14	5:00:07 p. m.	VALID	15	353	VALID	15	846
1/24/14	6:00:07 p. m.	VALID	15	558	VALID	15	957
1/24/14	7:00:07 p. m.	VALID	15	78	VALID	16	34
1/24/14	8:00:08 p. m.	VALID	15	403	VALID	15	736
1/24/14	9:00:08 p. m.	VALID	15	247	VALID	15	439
1/24/14	10:00:08 p. m.	VALID	14	987	VALID	15	202
1/24/14	11:00:08 p. m.	VALID	14	881	VALID	15	233
1/25/14	12:00:08 a. m.	VALID	14	773	VALID	14	898
1/25/14	1:00:08 a. m.	VALID	14	587	VALID	14	692
1/25/14	2:00:08 a. m.	VALID	14	408	VALID	14	551
1/25/14	3:00:07 a. m.	VALID	14	3	VALID	14	431
1/25/14	4:00:07 a. m.	VALID	14	304	VALID	14	37
1/25/14	5:00:07 a. m.	VALID	14	304	VALID	14	378

Figura 8.11. Errores existentes en la nomenclatura de las horas y los valores decimales.

Fuente: elaboración propia.

2. Si algunos archivos están dañados, no tienen información suficiente de las variables meteorológicas, probablemente por falta de mantenimiento de las estaciones. Se crea la función elim_BasesSinInfo() que recorre de manera jerárquica las carpetas organizadas por estaciones y años dentro del directorio principal, en cada archivo encontrado, se realiza una lectura mediante pandas.read_csv. Se aplica dos criterios de limpieza, el primero, se

elimina los archivos que presentan menos de 90 columnas, asumiendo que no cumplen con la estructura mínima requerida y luego se elimina los que poseen menos de 10 filas de registros porque esto indica insuficiencia de datos para un análisis. Se imprime en la pantalla el número de columnas o filas y la ruta de cada archivo eliminado, lo cual da seguimiento y revisar las correcciones.

```
def elim_BasesSinInfo():
    ests = os.listdir('ANUAL')
    for est in ests:
        Anios = os.listdir('ANUAL/'+est)
        for anio in Anios:
            files = os.listdir('ANUAL/'+est+'/'+anio)
            for file in files:
                df1 = pd.read_csv('ANUAL/'+est+'/'+anio+'/'+file, low_memory=False, on_bad_lines='skip')
                # Eliminar si tiene pocas columnas
                if len(df1.columns)<90:
                    os.remove('ANUAL/'+est+'/'+anio+'/'+file)
                    print('No de columnas: ',len(df1.columns))
```

```
            print("Se eliminó el archivo de: ",'ANUAL/'+est+'/'+anio+'/'+file)

        files = os.listdir('ANUAL/'+est+'/'+anio)

        for file in files:

            # Eliminar si tiene pocas filas

            if len(df1[df1.columns])<10:

                os.remove('ANUAL/'+est+'/'+anio+'/'+file)

                print('No de filas: ',len(df1[df1.columns]))

                print("Se eliminó el archivo de: ",'ANUAL/'+est+'/'+anio+'/'+file)

    return
```

Nota. *Luego del resultado de este segundo paso se trabaja son con objetos DataFrame de la librería pandas.*

3. Los nombres de las variables están separados en dos filas distintas lo cual resulta problemático al importar los datos para el análisis y desarrollo de programación. Por esta razón, se crearon dos funciones, unir_filas(df) y ejec_UnirFilas(), que en conjunto tienen como propósito la unión de las dos primeras filas. La primera función, unir_filas, toma un DataFrame, guarda los valores de la primera fila y los combina con los nombres originales de las columnas, creando

así etiquetas más completas y descriptivas, elimina la fila inicial redundante y devuelve el DataFrame actualizado.

La segunda función, ejec_UnirFilas, automatiza el proceso de la primera función recorriendo todas las estaciones y cada año dentro del directorio "ANUAL". A este proceso se incluye un paso adicional de corrección: si el archivo procesado tiene exactamente 106 columnas, cambia automáticamente la etiqueta "Unnamed: 1 AM" por "Unnamed: 1 time" para uniformar la nomenclatura. En caso de no poder realizar la unión, se notifica al usuario especificando el archivo problemático y se lleva un conteo de los fallos.

```
def unir_filas(df):
    fila_1 = list(df.iloc[0]) #guardar la primera fila
    df = df.drop([0], axis = 0) #eliminar la primera fila
    df = df.reset_index(drop=True)
    ini_Labs = list(df.columns) #guardar las etiquetas
    ## Crear las nuevas etiquetas
    new_Labs = []
    for i in range(len(ini_Labs)):
        try:
            new_Labs.append(ini_Labs[i] +" "+ fila_1[i])
```

```
        except:
            new_Labs.append(ini_Labs[i])
    df.columns = new_Labs
    return df
def ejec_UnirFilas():
    ests = os.listdir('ANUAL')
    cont = 0
    for est in ests:
        Anios = os.listdir('ANUAL/'+est)
        for anio in Anios:
            files = os.listdir('ANUAL/'+est+'/'+anio)
            for file in files:
                try:
                    df = pd.read_csv('ANUAL/'+est+'/'+anio+'/'+file,
low_memory=False, on_bad_lines='skip')
                    df1 = unir_filas(df)
                    df1.to_csv('ANUAL/'+est+'/'+anio+'/'+file,index=False)
```

```
            print('Se unieron las filas de: '+anio)
        except:
            print('En el archivo:')
            print('ANUAL/'+est+'/'+anio+'/'+file)
            print('No se pudo unir las dos primeras filas')
            cont += 1
    if len(df1.columns) == 106:
        df1 = df1.rename(columns={'Unnamed: 1 AM': 'Unnamed: 1 time'})
        df1.to_csv('ANUAL/'+est+'/'+anio+'/'+file,index=False)
        print("Se cambio la etiqueta time de: ",'ANUAL/'+est+'/'+anio+'/'+file)
    return cont
```

4. Algunas variables están con distintas etiquetas en los archivos csv esto impide el acoplamiento en conjunto de todos los archivos en una sola base de datos. Además, existen columnas que no tienen ningún dato, se eliminan de los datos actuales, estas columnas son: Unnamed 104 y Unnamed 106 (Ver Figura 8.12). Con estas consideraciones se realiza una estandarización de los nombres de las etiquetas en estos archivos utilizando las siguientes etiquetas:

```
In [23]: for i in range(len(df1)):
    ...:     if df3.columns[i] != df4.columns[i]:
    ...:         print(df3.columns[i],'->',df4.columns[i])
GenWind_1h SpdMin -> Sum_PR_1h Sum
GenWind_1h.1 WRun -> QMBATT_Meas meas
GenWind_1h.2 DirAvg -> GenWind_1h_SDI12 SpdMin
GenWind_1h.3 DirMax -> GenWind_1h_SDI12.1 WRun
GenWind_1h.4 GustDir -> GenWind_1h_SDI12.2 DirAvg
GenWind_1h.5 GustH -> GenWind_1h_SDI12.3 DirMax
GenWind_1h.6 GustM -> GenWind_1h_SDI12.4 GustDir
GenWind_1h.7 SpdAvg -> GenWind_1h_SDI12.5 GustH
GenWind_1h.8 SpdMax -> GenWind_1h_SDI12.6 GustM
Stat_WindChill_1h Avg -> GenWind_1h_SDI12.7 SpdAvg
Stat_WindChill_1h.1 Max -> GenWind_1h_SDI12.8 SpdMax
Stat_WindChill_1h.2 Min -> Stat_WindChill_1h_SDI12 Avg
QMBATT_Meas meas -> Stat_WindChill_1h_SDI12.1 Max
Unnamed: 104 -> Unnamed: 104 status
```

15	Sum_PR_1h	Sum	Precipitación de lluvia (Suma) cada hora	39
16	QMBATT_Meas	meas	Batería	40
17	GenWind_1h_SDI12	SpdMin	Velocidad de Viento (Mínima) cada hora	41
		WRun	Recorrido del viento	42
		DirAvg	Dirección del Viento (Promedio) cada hora	43
		DirMax	Dirección del Viento (Máxima) cada hora	44
		GustDir	Dirección de la racha de viento	45
		GustH	Hora de la racha de viento	46
		GustM	Minuto de la racha de viento	47
		SpdAvg	Velocidad de viento (Promedio)	48
		SpdMax	Velocidad de viento (Máxima)	49
18	Stat_WindChill_1h_SDI12	Avg	Sensación térmica (Promedio) cada hora	50
		Max	Sensación térmica (Máxima) cada hora	51
		Min	Sensación térmica (Mínima) cada hora	52

Figura 8.12. Comparación de los nombres de variables que muestran ser distintas.

Fuente: elaboración propia.

Se ha desarrollado la función *elim_Cols(df)*, se encarga de eliminar columnas dañadas o innecesarias, tales como aquellas con nombres genéricos (*"Unnamed: 104", "Unnamed: 106", "Unnamed: 126", etc.)* o

variables poco relevantes como *"planta3 meas"* o *"Charger External_DC"*. Además, revisa sistemáticamente los nombres de las columnas y elimina aquellas asociadas con mediciones redundantes (como las que contienen *"Dev", "WS", "WD" o "Lluvia_Meas PR"*), asegurando que no queden duplicidades o datos inconsistentes.

```
def elim_Cols(df):
    # eliminacion de columnas dañadas
    if 'Unnamed: 104' in list(df.columns):
        df = df.drop(columns=['Unnamed: 104'])
    if 'Unnamed: 106' in list(df.columns):
        df = df.drop(columns=['Unnamed: 106'])
    if 'planta3 meas' in list(df.columns):
        df = df.drop(columns=['planta3 meas'])
    if 'Charger External_DC' in list(df.columns):
        df = df.drop(columns=['Charger External_DC'])
    if 'Unnamed: 126' in list(df.columns):
        df = df.drop(columns=['Unnamed: 126'])
    if 'Unnamed: 142' in list(df.columns):
```

```
        df = df.drop(columns=['Unnamed: 142'])
    else :
        pass
    # Se hace un recorrido por las columnas para eliminar las que tienen Dev
    col_Elim = [] #crer una lista donde se guardan las columnas a eliminar
    for i in range(len(df.columns)):
        bool1 = 'Dev' in list(df.columns)[i]
        bool2 = 'WS' in list(df.columns)[i]
        bool3 = 'WD' in list(df.columns)[i]
        bool4 = 'Lluvia_Meas PR' in list(df.columns)[i]
        if bool1 or bool2 or bool3 or bool4 :
          col_Elim.append(list(df.columns)[i-1])
          col_Elim.append(list(df.columns)[i])
    df = df.drop(columns=col_Elim)
    if ('Unnamed: 94 status' in list(df.columns)) and len(list(df.columns))<100:
        df = df.drop(columns=['Unnamed: 94 status'])
```

```
    return df
```

Luego, se ha creado la función *set_Names()* que estandariza las etiquetas de las columnas renombrándolas de forma clara y consistente, por ejemplo, cambiando *"Unnamed: 1 AM"* a *"Unnamed: time"* o normalizando los nombres relacionados con variables como *"GenWind"* y *"WindChill"*, que representan la velocidad del viento y el recorrido del viento respectivamente.

```
def set_Names(df):
    """
 Funcion que realiza el cambio de nombres de las etiquetas para que todos los archivs    csv esten normalizados
    """
old_Names = list(df.columns)
aux_Names = [elem.split(' ') for elem in old_Names]
if 'Unnamed: 1 AM' in old_Names:
df = df.rename(columns={'Unnamed: 1 AM': 'Unnamed: time'})
for i in range(len(old_Names)):
if 'GenWind' in aux_Names[i][0]:
```

```
            df = df.rename(columns={old_Names[i]: 'GenWind '+aux_Names[i][1]})
        if 'WindChill' in aux_Names[i][0]:
            df = df.rename(columns={old_Names[i]: 'WindChill '+aux_Names[i][1]})
    return df
```

Finalmente, la función *change_ColNames()* automatiza este proceso recorriendo todas las carpetas organizadas por estación y año dentro del directorio principal, como se observa en la Figura 8.13, aplicando en cada archivo la limpieza de columnas y la normalización de nombres. Al concluir, guarda nuevamente los archivos ya corregidos, notificando qué conjuntos de datos han sido procesados, garantizando que los archivos queden depurados, con etiquetas uniformes y libres de información dañada.

```
def change_ColNames():
## Recorrer todos los archivos csv de las estaciones
    ests = os.listdir('ANUAL')
    for est in ests:
        try:
            Anios = os.listdir('ANUAL/'+est)
            for anio in Anios:
```

```
            files = os.listdir('ANUAL/'+est+'/'+anio)
            for file in files:
                df = pd.read_csv('ANUAL/'+est+'/'+anio+'/'+file,
low_memory=False, on_bad_lines='skip')
                df = elim_Cols(df)
                df = set_Names(df)
                df.to_csv('ANUAL/'+est+'/'+anio+'/'+file,index=False)
                print('El archivo de :'+anio+' fue configurado sus etiquetas')
        except:
            pass
    return
```

5. Al explorar las carpetas de la Figura 8.13 existen algunos años donde los datos están divididos en varios archivos, esto se debe a que no se realizó una compilación correcta o no se realizó un solo archivo, realizarlo manualmente llega a ser tedioso debido a que existen fechas y horas repetidas.

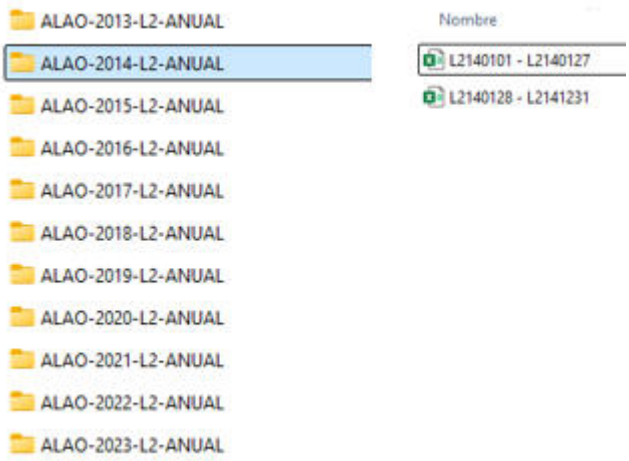


Figura 8.13. Ejemplo de bases de datos dividida en dos o más archivos.

Fuente: elaboración propia.

La implementación de los *DataFrame* realiza un recorrido por las carpetas de las estaciones y por año de forma automática, como se muestra el siguiente proceso:

```
def unir_Csv():
    ## Recorrer todos los archivos csv de las estaciones
    ests = os.listdir('ANUAL')
    for est in ests:
        Anios = os.listdir('ANUAL/'+est)
        for anio in Anios:
```

```
        files = os.listdir('ANUAL/'+est+'/'+anio)
        # detectar si hay dos archivos
        if len(files) > 1:
            dfs = []
            for file in files:
                dfs.append(pd.read_csv('ANUAL/'+est+'/'+anio+'/'+file,
low_memory=False, on_bad_lines='skip'))
                os.remove('ANUAL/'+est+'/'+anio+'/'+file)
            df = pd.concat(dfs, axis=0, ignore_index=True)
            df.to_csv('ANUAL/'+est+'/'+anio+'/NEW'+file,index=False)
            print('En la estación:'+est+' '+anio+' se concatenó sus
archivos')
    return
```

El proceso comienza recorriendo todas las carpetas del directorio y dentro de cada subcarpeta, la función verifica si existen varios archivos CSV correspondientes al mismo período; de ser así, procede a leerlos uno por uno y almacenarlos temporalmente en una lista. Una vez cargados, los archivos originales se eliminan para evitar duplicidad y se concatenan en un único *DataFrame* utilizando la función *pd.concat()*, manteniendo el orden de los registros. Finalmente, el nuevo archivo consolidado se guarda con el

nombre “NEW” seguido del nombre del último archivo procesado en la misma carpeta. Este procedimiento garantiza que cada combinación de estación y año conserve un único archivo completo y uniforme.

8.4.2. Agrupamiento y normalización de la base de datos

1. Las bases de datos de las estaciones tienen para cada variable una columna previa con valores *“VALID” e “INVALID”,* que referencia un valor coherente, o un valor que no se considera como válido ya que existió un error de medición debido al instrumentos de medición, una desconfiguración del equipo, una falla climática, entre otros aspectos que pudieron afectar la correcta toma de ese dato. En la Figura 8.14 se observa cómo la información en datos inválidos, a pesar de tener valor numérico debe ser reemplazado como dato “nulo”. Estos valores de validación se encuentran en una columna previa a la variable con el nombre de *“status”.*

Stat_TS_1h_TG7		Sum_PR_1h		QMBATT_Meas		GenWind_1h		GenWind_1h		GenWind_1h	
Min	status	Sum	status	meas	status	SpdMin	status	WRun	status	DirAvg	status
19.266	VALID	0	VALID	12.78	INVALID	0	INVALID	0	INVALID	360	INVALID
19.27	VALID	0	VALID	12.747	INVALID	0	INVALID	0	INVALID	360	INVALID
19.265	VALID	0	VALID	12.729	INVALID	0	INVALID	0	INVALID	360	INVALID
19.272	VALID	0.1	VALID	12.734	INVALID	0	INVALID	0	INVALID	360	INVALID
19.268	VALID	0	VALID	12.706	INVALID	0	INVALID	0	INVALID	360	INVALID
19.269	VALID	0	VALID	12.713	INVALID	0	INVALID	0	INVALID	360	INVALID
19.267	VALID	0	VALID	12.697	INVALID	0	INVALID	0	INVALID	360	INVALID
19.257	VALID	0	VALID	12.647	INVALID	0	INVALID	0	INVALID	360	INVALID
19.266	VALID	0	VALID	12.631	INVALID	0	INVALID	0	INVALID	360	INVALID
19.253	VALID	0	VALID	12.625	INVALID	0	INVALID	0	INVALID	360	INVALID
19.26	VALID	0	VALID	12.599	INVALID	0	INVALID	0	INVALID	360	INVALID
19.263	VALID	0	VALID	12.535	INVALID	0	INVALID	0	INVALID	360	INVALID
19.251	VALID	0	VALID	12.52	INVALID	0	INVALID	0	INVALID	360	INVALID
19.262	VALID	0	VALID	12.583	INVALID	0	INVALID	0	INVALID	360	INVALID
19.258	VALID	0	VALID	13.607	INVALID	0	INVALID	0	INVALID	360	INVALID

Figura 8.14. Base de datos.

Fuente: elaboración propia.

Primero, identifica dentro del *DataFrame* las columnas que contienen la palabra *"status"*, luego recorre fila por fila y, cuando detecta el valor "INVALID", sustituye el valor correspondiente de la variable asociada por un nulo, en este caso se utiliza el objeto de valor nulo de la librería de *numpy* (*np.nan*), garantizando así que no se consideren registros defectuosos en los análisis posteriores. Finalmente, elimina todas las columnas de *"status"*, puesto que se vuelve inútil conservarlas y la función devuelve un *DataFrame*.

```
def replace_Nan(df):
    # Se extraen las columnas que tienen 'status' y las que son de variables
    Cols = df.columns[2:]
    valids_Cols = []
    vars_Cols = []
      for i in range(len(Cols)):
      if 'status' in Cols[i]:
          valids_Cols.append(Cols[i])
      else:
          vars_Cols.append(Cols[i])
    #print(len(valids_Cols),len(vars_Cols))
```

```
    # Se recorren las columnas y variables para rellenar con nulos los invalidos
    fils,cols = df.shape
    for i in range(fils):
        for j in range(len(vars_Cols)):
            if df.loc[i,valids_Cols[j]] == 'INVALID':
                df.loc[i,vars_Cols[j]] == np.nan
    # Se eliminan las columnas de 'status'
    df = df.drop(columns=valids_Cols)
    return df
```

2. Otra información de las bases de datos que se normalizan es la fecha y hora, debido que aún se encuentran en distintas columnas (ver Figura 8.15). En programación no existe un tipo de dato por separado para fecha y otro para hora, por el contrario, se considera como un solo objeto dentro de la memoria del computador.

		Stat_TA_1h	Stat_TA_1h	Stat_TA_1h	Stat_RH_1h
date	time	Avg	Max	Min	Avg
1/1/2024	12:00:13 a. m.	14.461	14.877	13.953	82.343
1/1/2024	1:00:10 a. m.	13.769	14.102	13.622	86.057
1/1/2024	2:00:10 a. m.	13.741	14.053	13.557	86.986
1/1/2024	3:00:11 a. m.	13.472	13.68	13.365	88.332
1/1/2024	4:00:11 a. m.	13.251	13.654	13.097	90.261
1/1/2024	5:00:11 a. m.	12.919	13.555	12.573	89.982
1/1/2024	6:00:11 a. m.	12.605	12.855	12.443	90.074
1/1/2024	7:00:11 a. m.	12.333	12.787	11.59	91.942
1/1/2024	8:00:11 a. m.	11.634	11.9	11.522	96.953

Figura 8.15. Base de datos luego de la inserción de nulos.

Fuente: elaboración propia.

La función *change_Time(df)* tiene como objetivo unificar la información de fecha y hora en una sola columna de tipo *datetime*. Primero se elimina las filas que contienen valores nulos en la columna de fechas. Luego se recorre cada fila del *DataFrame* para separar y procesar las cadenas de texto que representan la fecha (en formato con "/" o "-") y la hora (incluyendo la indicación AM o PM). A partir de estas cadenas, se convierte los valores en enteros y se aplica las reglas necesarias para transformar el formato de 12 horas al de 24 horas, corrigiendo casos especiales como las 12 AM o las 12 PM. Una vez construida la fecha y la hora en un solo valor *datetime*, la función reemplaza el contenido original de la columna de fechas, elimina la columna de horas redundante y finalmente renombra la columna resultante como "Tiempo".

```
def change_Time(df):
    """
```

```
    Funcion que cambia y configura los valores de fecha y tiempo para ser
    un solo datetiem en la primera columna del DataFrame
    """
    # Se eliminan filas con valores nulos
    df = df.dropna(subset=[df.columns[0]], ignore_index=True)
    # Se recorren todas las filas del DataFrame y se junta la fecha y hora en datetime
    cols = df.columns
    for i in range(len(df)):
        #print(i)
        # Las columnas Fecha y Hora para transformarlo en datetime en una columna
        resp = True
        try :
            mes,dia,anio = df[cols[0]][i].split('/')
        except:
            try:
                mes,dia,anio = df[cols[0]][i].split('-')
```

```
            except:
                print('El dato N° '+str(i)+' no se pudo procesar')
                resp = False
        if resp :
            [h,m,s],AP = df[cols[1]][i].split(' ')[0].split(':'),df[cols[1]][i].split(' ')[1]
            m += 'I'
            del(m)
            if AP == 'AM' and int(h)==12 and int(s)>0:
                h = 0
            elif AP == 'PM' and int(h)==12:
                h = 12

            elif AP == 'PM' and int(h)!=12:
                h = int(h)+12
            else:
                pass
```

```
        if int(anio)< 1000:
            df.loc[i,cols[0]] =
datetime(int(anio)+2000,int(mes),int(dia),int(h))
        else:
            df.loc[i,cols[0]] = datetime(int(anio),int(mes),int(dia),int(h))
    df = df.drop(columns=[cols[1]])
    df = df.rename(columns={'Unnamed: 0 date': 'Tiempo'})
    return df
```

3. Crear una base consolidada de la red de estaciones de la ESPOCH requiere ejecución concatenada de los datos csv para cada estación y que todas las estaciones tengan el mismo número de variables, para ello, se ha creado la función unir_CsvEstacion() que consolida en un solo archivo CSV toda la información disponible de una estación específica y los resultados se guardan, en este caso, en la carpeta denominada "CHIMBORAZO", la función localiza y lee todos los archivos CSV, almacenando sus contenidos en una lista de DataFrames. Una vez recopilados, se concatenan con pd.concat() en un único DataFrame que integra la información completa de todos los años. Finalmente, este archivo unificado se guarda en con el nombre de la estación con extensión csv.

```
def unir_CsvEstacion():
```

```
    ## Recorrer todos los archivos csv de las estaciones
    ests = os.listdir('ANUAL')
    for est in ['CHIMBORAZO']:
        Anios = os.listdir('ANUAL/'+est)
        dfs = []
        for anio in Anios:
            files = os.listdir('ANUAL/'+est+'/'+anio)
            for file in files:
                dfs.append(pd.read_csv('ANUAL/'+est+'/'+anio+'/'+file,
low_memory=False, on_bad_lines='skip'))
        df = pd.concat(dfs, axis=0, ignore_index=True)
        df.to_csv('ANUAL/'+est+'.csv',index=False)
        print('Se creo la compilación de la estacion '+ est)
    return
```

Se comprueba que todas las estaciones tengan el mismo número de columnas, para realizar la concatenación consolidada a través de netCDFs. Sin embargo, se observa que algunas estaciones tienen mediciones de variables extra, debido a que fueron configuradas con más instrumentos de

medición. Toda esta revisión y estandarización del número de variables se realiza con el siguiente código:

```
def config_csvEstacion():
    """
    Realiza el recorrido por cada archivo csv compilado por estación y elimina
    columnas innecesarias. Tambien añade las necesarias
    """
    ests = os.listdir('ANUAL')
    for est in ests:
        resp = False
        try:
            df = pd.read_csv('ANUAL/'+est, low_memory=False, on_bad_lines='skip')
            resp = True
        except:
            pass
        if resp:
```

```
        if 'Unnamed: 100' in list(df.columns):
            df = df.drop(columns=['Unnamed: 100'])
        if 'GenWind DirMin' in list(df.columns):
            df = df.drop(columns=['GenWind DirMin'])
        L = ['Sum_PR_1h Sum','QMBATT_Meas meas']
        for elem in L:
            if elem not in df:
                df[elem] = np.full(len(df),np.nan)
        print('Se configuró el archivo ',est)
        df.to_csv('ANUAL/'+est,index=False)
    return
```

Nuevamente, se comprueba que todas las estaciones tengan el mismo número de columnas y se verifica el mismo valor para todas.

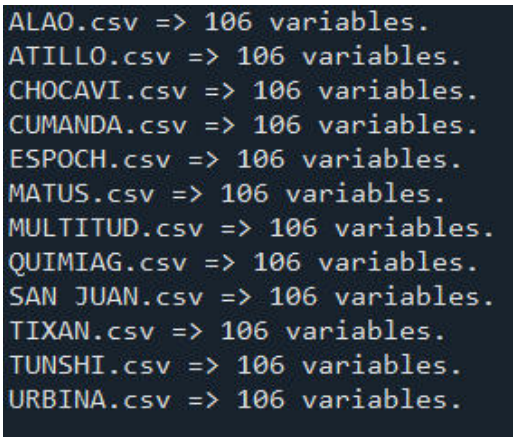


Figura 8.16. Comprobación del número de variables en las bases de datos consolidadas.

Fuente: elaboración propia.

4. Se identifica valores numéricos que no se expresan como números flotantes, por ejemplo, errores como “0.45.5” o valores que son caracteres o cadenas de texto, fallas de digitalización o tipeo, que son poco comunes. De la misma manera, se identifica mediciones tomadas más de una vez en la misma fecha y hora, debido a replicación de la medición, un fallo eléctrico o alteración del Timer. La función fix_Database(df) es diseñada para resolver estos problemas, se encarga de depurar y normalizar los datos climáticos asegurando que todas las variables puedan convertirse en valores numéricos de tipo flotante. Para ello, convierte cada columna a este formato y reemplaza con valores nulos (np.nan) aquellos registros que no puedan transformarse. Posteriormente, agrupa los datos utilizando la columna “Tiempo” y calcula un promedio de las observaciones,

con el fin de evitar la duplicidad de mediciones registradas en el mismo instante temporal.

```
def fix_Database(df):

    # Convertir los datos a flotantes

    # Detectar valores no numéricos en columnas relevantes

    for col in list(df.columns[1:]):

        df[col] = pd.to_numeric(df[col], errors="coerce")  # Convertir y asignar NaN si no es convertible

    # Se agrupa los valores obsevando el datetime en la columna 'Tiempo'

    df_promedio = df.groupby('Tiempo', as_index=False).mean()

    return df_promedio
```

5. Se crea la función delete_GlobNames(df) que realiza una limpieza de los nombres de las columnas dentro del DataFrame, elimina prefijos, sufijos y caracteres innecesarios (como “Stat_”, “1h_”, “_Meas”, entre otros) que suelen estar presentes en las etiquetas generadas por instrumentos o sistemas de adquisición de datos. De esta manera, se obtiene una nomenclatura más clara, homogénea y estandarizada, que facilita la lectura de los datos.

```
def delete_GlobNames(df):
```

```
    elims = ['Stat_','1h_','_1h','.1','.2','.3','.4','.5','.6','.7','.8','_SDI12','Sum','_Meas']

    names = list(df.columns)

    for cad in elims:

        for i in range(len(names)):

            if cad in names[i]:

                names[i] = names[i].replace(cad,'')

    df.columns = names

    return df
```

6. Con la variable datetime, adaptada en las bases de datos, se estandariza en un solo rango temporal. La función config_RangeMin(df) establece un rango temporal continuo desde una fecha mínima predefinida (en este caso, diciembre de 2013) hasta la primera fecha registrada en la base de datos. Para cubrir las horas faltantes, genera una secuencia de valores de tiempo y completa las demás columnas con valores nulos, de modo que se crea una estructura uniforme.

```
def config_RangeMin(df):

    df["Tiempo"] = pd.to_datetime(df["Tiempo"], errors="coerce")

    cols = list(df.columns[1:])
```

```
min_Glob = datetime(2013,12,18,1)

min_Loc = df['Tiempo'][0]

L = []

while min_Glob<min_Loc:

    L.append(min_Glob)

    min_Glob = min_Glob + timedelta(hours=1)

arr = [L]

for i in cols:

    arr.append(np.full(len(L),np.nan))

arr = np.array(arr).T

df_arr = pd.DataFrame(arr,columns=df.columns)

new_df = pd.concat([df_arr,df],axis=0,ignore_index=True)

return new_df
```

El resultado final es una base de datos por estación meteorológica estandarizada y normalizada, que se importan a cualquier software de análisis estadístico, como se muestra en la Figura 8.17.

Tiempo	TA Avg	TA Max	TA Min	RH Avg	RH Max	RH Min	PA Avg	PA Max	PA Min	SR_Dif Avg	SR_Dif Max	SR_Dif Min	Sum_SR_Dif
15/9/2023 16:00	4.778	6.182	3.427	99.801	99.846	96.14	636.376	636.467	636.231	325.229	645.884	78.644	252.377.406
15/9/2023 17:00	5.351	6.266	4.984	98.439	100.016	4.0	636.195	637.494	635.871	483.851	878.056	291.915	1.733.638.875
15/9/2023 18:00	4.762	5.212	4.315	99.832	99.856	99.819	635.708	636.088	635.307	471.678	1.160.587	280.879	1698039.75
15/9/2023 19:00	3.999	4.594	3.495	99.824	99.854	99.808	635.218	635.5	634.864	287.334	476.67	171.101	1.034.404.125
15/9/2023 20:00	3.74	4.108	3.467	99.82	99.847	99.806	634.726	635.02	634.321	244.421	532.527	54.737	879.914.938
15/9/2023 21:00	3.621	4.007	3.434	99.817	99.844	99.805	634.312	634.449	634.078	155.608	648.466	56.388	560.188.438
15/9/2023 22:00	3.509	3.764	3.403	99.772	99.838	98.77	634.229	634.445	634.069	100.103	173.566	53.762	360.370.344
15/9/2023 23:00	3.311	3.698	2.677	94.671	99.793	87.915	634.534	634.664	634.368	71.574	151.716	19.692	257.667.766
16/9/2023 0:00	2.28	2.822	1.965	98.946	99.792	94.083	634.543	634.662	634.391	4.34	41.406	0.0	15.623.083
16/9/2023 1:00	1.974	2.082	1.868	97.172	99.782	92.686	634.786	635.009	634.532	0.0	0.0	0.0	0.0
16/9/2023 2:00	1.996	2.107	1.907	92.808	94.904	89.959	635.353	635.718	634.964	0.0	0.0	0.0	0.0
16/9/2023 3:00	1.883	2.057	1.697	87.233	90.162	85.46	635.899	636.044	635.622	0.0	0.0	0.0	0.0
16/9/2023 4:00	1.819	2.08	1.565	88.804	92.398	86.143	635.936	636.052	635.803	0.0	0.0	0.0	0.0
16/9/2023 5:00	1.456	1.673	1.258	95.955	99.771	92.418	635.758	635.897	635.55	0.0	0.0	0.0	0.0
16/9/2023 6:00	1.693	1.836	1.526	99.482	99.782	98.062	635.525	635.667	635.326	0.0	0.0	0.0	0.0
16/9/2023 7:00	1.639	1.862	1.519	99.764	99.787	99.752	635.065	635.419	634.676	0.0	0.0	0.0	0.0
16/9/2023 8:00	1.475	1.739	1.256	96.596	99.786	87.96	634.503	634.729	634.251	0.0	0.0	0.0	0.0
16/9/2023 9:00	1.213	1.662	1.008	96.373	99.774	86.467	634.329	634.467	634.101	0.0	0.0	0.0	0.0
16/9/2023 10:00	0.987	1.124	0.823	98.274	99.783	93.516	634.367	634.545	634.216	0.0	0.0	0.0	0.0
16/9/2023 11:00	0.923	1.177	0.816	97.693	99.777	90.102	634.459	634.555	634.318	0.014	0.771	0.0	48.724
16/9/2023 12:00	1.398	2.23	0.824	89.024	96.794	77.036	634.79	635.086	634.38	68.006	190.586	0.761	244.822.656
16/9/2023 13:00	2.809	3.459	1.987	78.214	87.064	70.693	635.175	635.433	634.872	250.209	326.01	167.379	900750.75
16/9/2023 14:00	4.109	5.209	3.374	70.935	77.112	67.001	635.671	636.107	635.324	361.804	426.22	315.696	1.302.493.875
16/9/2023 15:00	5.533	6.053	4.875	66.159	69.073	63.14	636.179	636.358	635.961	473.165	577.198	401.147	1.703.394.375
16/9/2023 16:00	6.307	7.08	5.614	65.271	70.273	58.047	636.273	636.427	636.109	662.891	865.867	527.596	2386406.75
16/9/2023 17:00	7.223	8.05	6.623	61.682	65.796	55.806	635.981	636.245	635.724	914.526	1.201.145	546.077	3292294.75
16/9/2023 18:00	7.489	8.71	6.613	62.849	66.365	58.503	635.638	635.893	635.41	1.009.481	1.490.218	295.948	3634132.75
16/9/2023 19:00	8.28	9.117	7.637	61.028	65.887	57.474	635.462	635.639	635.272	1.147.599	1.491.884	256.177	4131356.25
16/9/2023 20:00	7.793	8.922	6.413	60.355	68.31	52.172	635.076	635.373	634.805	709.94	1.390.807	130.56	2555785.75
16/9/2023 21:00	6.359	8.009	5.297	67.504	76.7	55.317	634.803	635.018	634.616	292.567	1.107.902	101.528	1.053.239.875
16/9/2023 22:00	4.712	5.378	4.283	87.02	94.947	76.763	634.846	634.986	634.595	107.102	518.053	35.14	385.568.812
16/9/2023 23:00	4.21	4.681	3.797	89.552	94.759	84.545	634.956	635.118	634.828	48.963	120.124	17.603	176.267.344

Figura 8.17. Base de datos final de la estación ESPOCH.

Fuente: elaboración propia.

8.4.3. Creando una base de datos consolidada netCDF

La transformación de múltiples archivos estandarizados de cada estación en formato csv a una única base de información en formato netCDF, con el objetivo de gestionar la información de forma eficiente, estructurada y escalable. Se ha creado un archivo .txt con la descripción de las variables, como los valores de latitud y longitud, unidad de medida, abreviación, promedios, máximos y mínimos. A este archivo se lo ha llamado *'metadata.txt'* y se describe:

ALAO_LAT=-1.87

ALAO_LON=-78.54

ATILLO_LAT=-2.19

ATILLO_LON=-78.55

CHOCAVI_LAT=-1.53

CHOCAVI_LON=-78.69

CUMANDA_LAT=-2.21

CUMANDA_LON=-79.15

ESPOCH_LAT=-1.65

ESPOCH_LON=-78.68

MATUS_LAT=-1.56

MATUS_LON=-78.50

MULTITUD_LAT=-2.11

MULTITUD_LON=-78.99

QUIMIAG_LAT=-1.66

QUIMIAG_LON=-78.57

SAN JUAN_LAT=-1.64

SAN JUAN_LON=-78.78

TIXAN_LAT=-2.16

TIXAN_LON=-78.76

TUNSHI_LAT=-1.75

TUNSHI_LON=-78.63

URBINA_LAT=-1.49

URBINA_LON=-78.71

GUARGUALLA_LAT=-1.92

GUARGUALLA_LON=-78.56

TA=Temperatura Ambiente

TA_UNIT=Grados Celsius

RH=Humedad Relativa

RH_UNIT=%

PA=Presión Atmosférica

PA_UNIT=Pascal

SR=Radiación solar

DIF=difusa

GLOB=global

SR_UNIT=W/m2

TS=Temperatura de suelo

TS_UNIT=Grados Celsius

PR=Precipitación cada hora

PR_UNIT=mm

QMBATT=Batería

QMBATT_UNIT=V

WINDCHILL=Sensación térmica

WINDCHILL_UNIT=Celsius

GENWIND=Viento

SPDMIN=Velocidad mínima

SDPMIN_UNIT=m/s

WRUN=Recorrido

WRUN_UNIT=metros

DIRAVG=Dirección promedio

DIRAVG_UNIT=Grados

DIRMAX=Dirección máxima

DIRMAX_UNIT=Grados

GUSTDIR=Dirección de la racha

GUSTDIR_UNIT=Grados

GUSTH=Hora de la racha

GUSTH_UNIT=Hora formato 24H

GUSTM=Minuto de la racha

GUSTM_UNIT=Minuto en la hora formato MM

SPDAVG=Velocidad promedio

SPDAVG_UNIT=mm/s

SPDMAX=Velocidad máxima

AVG=Promedio

MAX=Máximo

MIN=Mínimo

SUM=Sumatoria

MEAS=

TG1=Nivel 1 -> 10cm sobre el suelo

TG2=Nivel 2 -> 0cm sobre el suelo

TG3=Nivel 3 -> 10cm bajo el suelo

TG4=Nivel 4 -> 20cm bajo el suelo

TG5=Nivel 5 -> 30cm bajo el suelo

TG6=Nivel 6 -> 40cm bajo el suelo

TG7=Nivel 7 -> 50cm bajo el suelo

La creación de las bases de datos por estación se implementa en la función *crear_netCDF_Base()*. Esta función importa los metadatos de cada estación, recorre todas las carpetas de estaciones dentro del directorio ANUAL, leyendo los archivos CSV de cada una e importándolos como *DataFrames*. Posteriormente, cada *DataFrame* se convierte en un objeto *xarray.Dataset*, indexado por la variable "Tiempo", se recopilan los nombres de las estaciones y sus coordenadas geográficas guardados en el archivo metadata.txt. La función realiza la concatenación de todos los *Dataset* en un único objeto multidimensional mediante *xr.concat()*, usando la dimensión "estacion". A este objeto combinado se asigna como coordenadas los nombres de las estaciones con sus respectivas latitudes y longitudes, creando una base de datos unificada y consolidada en formato netCDF.

```
def crear_netCDF_Base():

    def import_Metadatos():

        with open('metadata.txt','r') as f: cad_L = f.readlines()
```

```
        new_L = [elem[:-1].split('=') for elem in cad_L]
        return dict(new_L)
    # Se importan los metadatos
    meta = import_Metadatos()
    #Recorrer todos los archivos de estaciones e importar la informacion
    ests = os.listdir('ANUAL')
    # crear listas para extraer los csv y convertirlos en xarray
    # tambien extraer los nombres de las estaciones
    est_Names = []
    dfs = []
    lats = []
    lons = []
    for est in ests:
        try:
            df = pd.read_csv('ANUAL/'+est, low_memory=False,
on_bad_lines='skip',parse_dates=['Tiempo'])
            arr_df = xr.Dataset.from_dataframe(df.set_index(["Tiempo"]))
```

```
            dfs.append(arr_df)
            est_Names.append(est[:-4])
            lats.append(float(meta[est[:-4]+'_LAT']))
            lons.append(float(meta[est[:-4]+'_LON']))
            print('Se importó la información de la estacion '+ est)
        except:
            pass
    #realizar la combinación en un solo netCDF
    combined_ds = xr.concat(dfs, dim="estacion")
    combined_ds = combined_ds.assign_coords(
        estacion =("estacion", est_Names),
        latitud =("estacion", lats),
        longitud =("estacion", lons))
    return combined_ds
```

Al dataset resultante se configura los metadatos informativos de las variables, se detalla la función:

```
def config_Metadata(arr):
```

```
# Se realiza el proceso para agregar los metadatos

meta = import_Metadatos()

# Agregar una descripción al dataset

arr.attrs["description"] = "Datos de 52 variables meteorológicas de 12 estaciones ubicadas en la provincia de Chimborazo desde 2013."

arr.attrs["source"] = "Red de estaciones meteorológicas GEAA-ESPOCH"

arr.attrs["creation_date"] = str(datetime.now().date())

arr.attrs["data_vertion"] = "1.0.0"

arr.attrs["frequency"] = "hora"

# Agregar descripción "long name" a cada variable

names = list(arr.data_vars.keys())

new_names = [elem.upper() for elem in names]

L_names = [elem.split(' ') for elem in new_names]

# AGREGAR EL ATRIBUTO 'Long name' y 'units' recorrer todas las variables y verificar

# si existe '_' en el nombre de la varialbe luego buscar esa cadena en los

# metadatos para agregar la cadena correspondiente
```

```
for i in range(len(L_names)):
    cad =''
    uni =''
    # añadir a la cadena el nombre de la variable
    if '_' in L_names[i][0]:
        L_vars = L_names[i][0].split('_')
        for var in L_vars:
            cad = cad +' '+ meta[var]
            try:
                uni= uni+meta[var+'_UNIT']
            except:
                pass
    else:
        var = L_names[i][0]
        cad = cad + meta[var]
        try:
            uni= uni+meta[var+'_UNIT']
```

```
            except:
                pass
            #añadir a la cadena la característica de la variable
            try:
                car = L_names[i][1]
                cad = cad + ' ' + meta[car]
            except:
                pass
            arr[names[i]].attrs['long_name'] = cad
            arr[names[i]].attrs['units'] = uni
        return arr
```

A continuación, se muestra el archivo netCDF resultante:

```
<xarray.Dataset> Size: 493MB
Dimensions:         (estacion: 13, Tiempo: 91112)
Coordinates:
  * estacion        (estacion) <U10 520B 'ALAO' 'ATILLO' ... 'TUNSHI'
'URBINA'
```

latitud (estacion) float64 104B ...

longitud (estacion) float64 104B ...

** Tiempo (Tiempo) datetime64[ns] 729kB 2013-12-18T08:00:00 ... 20...*

Datas variables: (12/52)

TA Avg (estacion, Tiempo) float64 9MB ...

TA Max (estacion, Tiempo) float64 9MB ...

TA Min (estacion, Tiempo) float64 9MB ...

RH Avg (estacion, Tiempo) float64 9MB ...

RH Max (estacion, Tiempo) float64 9MB ...

RH Min (estacion, Tiempo) float64 9MB ...

...

GenWind SpdMax (estacion, Tiempo) float64 9MB ...

WindChill Avg (estacion, Tiempo) float64 9MB ...

WindChill Max (estacion, Tiempo) float64 9MB ...

WindChill Min (estacion, Tiempo) float64 9MB ...

QMBATT meas (estacion, Tiempo) float64 9MB ...

Sum_PR (estacion, Tiempo) float64 9MB ...

Attributes:

description: Datos de 52 variables meteorológicas de 12 estaciones ubi...

source: Red de estaciones meteorológicas GEAA-ESPOCH

creation_date: 2025-02-05

data_vertion: 1.0.0

frequency: hora

Se verifica que los datos informativos se han agregado correctamente, además al seleccionar una variable se observa sus propios metadatos que ayudan al usuario a guiarse:

ds['TA Avg']

Out[6]:

<xarray.DataArray 'TA Avg' (estacion: 13, Tiempo: 91112)> Size: 9MB

[1184456 values with dtype=float64]

Coordinates:

** estacion (estacion) <U10 520B 'ALAO' 'ATILLO' ... 'TUNSHI' 'URBINA'*

latitud (estacion) float64 104B ...

longitud (estacion) float64 104B ...

** Tiempo (Tiempo) datetime64[ns] 729kB 2013-12-18T08:00:00 ... 2024-05-1...*

Attributes:

long_name: Temperatura Ambiente Promedio

units: Grados Celsius

Finalmente se guarda el *dataset* en un archivo netDF en el computador mediante el comando *ds.to_netcdf('Nombre_del_Archivo.nc')*. El resultado es un archivo portable, con un valor reducido de memoria, adecuado para realizar consultas y análisis estadístico de la información incorporada.

8.4.4. Extrayendo resultados usando la nueva base de datos

Se realizar filtrados y sus respectivas representaciones en gráficas de series temporal, por ejemplo, los datos de las estaciones de la variable humedad relativa promedio desde el 3 de julio de 2015 hasta el 15 de marzo del 2019:

da = ds['RH Avg'].sel(Tiempo=slice(datetime(2015,7,3),datetime(2019,3,15)))

Out:

<xarray.DataArray 'RH Avg' (estacion: 13, Tiempo: 32425)> Size: 3MB

[421525 values with dtype=float64]

Coordinates:

** estacion (estacion) <U10 520B 'ALAO' 'ATILLO' ... 'TUNSHI' 'URBINA'*

latitud (estacion) float64 104B ...

longitud (estacion) float64 104B ...

** Tiempo (Tiempo) datetime64[ns] 259kB 2015-07-03 ... 2019-03-15*

Attributes:

long_name: Humedad Relativa Promedio

units: %

Se representa gráficamente el comportamiento de la Humedad Relativa en función del tiempo y de la estación meteorológica, usando. *plot()*.

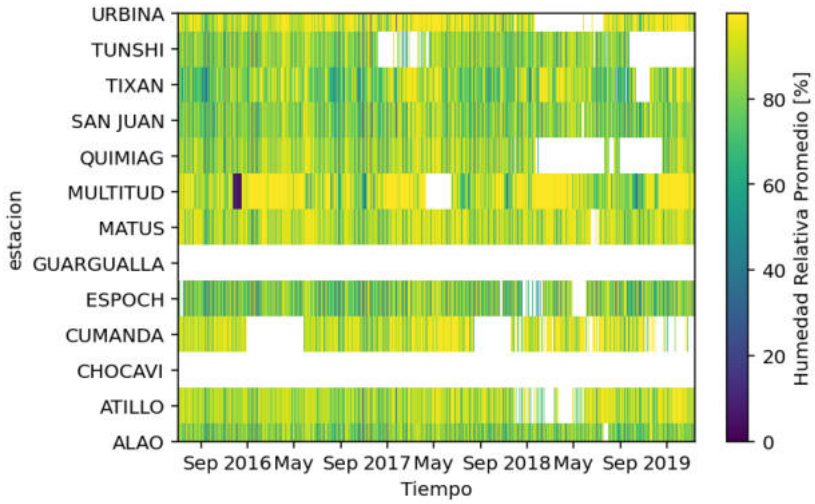


Figura 8.18. Mapa de calor de los valores de humedad relativa promedio por hora de las estaciones desde el 3 de Julio de 2015 hasta el 15 de marzo del 2019.

Fuente: elaboración propia.

La Figura 8.18 corresponde a un mapa de calor de la humedad relativa promedio (%) en varias estaciones meteorológicas de la provincia de Chimborazo entre septiembre de 2016 y finales de 2019. El eje vertical representa las diferentes estaciones, mientras que el eje horizontal muestra la escala temporal. Los colores van desde tonalidades púrpuras (baja humedad relativa, cercanas al 0%) hasta tonos amarillos intensos (alta humedad relativa, cercanas al 100%).

En la figura se identifica patrones espaciales y temporales de la humedad relativa, se observa los valores faltantes (representados por franjas blancas), de la misma forma, anomalías en los registros (como valores sospechosos

de ser atípicos en tonos púrpuras oscuros). Se observa en la mayoría de las estaciones, que la humedad relativa mantiene valores altos, generalmente por encima del 60%, que es consistente con las condiciones climáticas propias de zonas andinas caracterizadas por una marcada influencia de la nubosidad y precipitaciones frecuentes.

Se detalla el comportamiento climático por cada zona, considerando con su ubicación regional:

Patrón general de alta humedad: estaciones como URBINA, QUIMIAG, ATILLO y ALAO presentan registros persistentemente elevados, lo que refleja microclimas húmedos asociados a altitudes superiores a 2000 msnm, influencia orográfica y mayor condensación de humedad atmosférica.

Variabilidad estacional: se distinguen periodos de ligeras disminuciones en la humedad relativa, más notorios en TIXÁN, TUNSHI y SAN JUAN, coincidiendo con meses secos en la región interandina (junio–septiembre). Esta variabilidad se traduce en oscilaciones en la tonalidad del color hacia verdes y azules.

Vacíos de información y heterogeneidad: estaciones como GUARGUALLA y CUMANDÁ presentan largos periodos de datos faltantes. En el caso de la primera estación, fue instalada en año 2023 no tiene información registrada en este periodo.

Datos extremos: en MULTITUD se observa un valor extremo de baja humedad, posiblemente asociado a un error de registro o un fallo puntual del sensor.

Considerando las características geográficas se destaca que las estaciones meteorológicas que se encuentran a mayor altitud presentan un comportamiento estable de la humedad relativa, mientras que en zonas de menor altitud como CUMANDÁ se observa una mayor variabilidad y periodos más secos, reflejando la influencia de gradientes altitudinales en el régimen de humedad relativa.

Se grafica la serie temporal de la variable humedad relativa, se selecciona una sola estación*:*

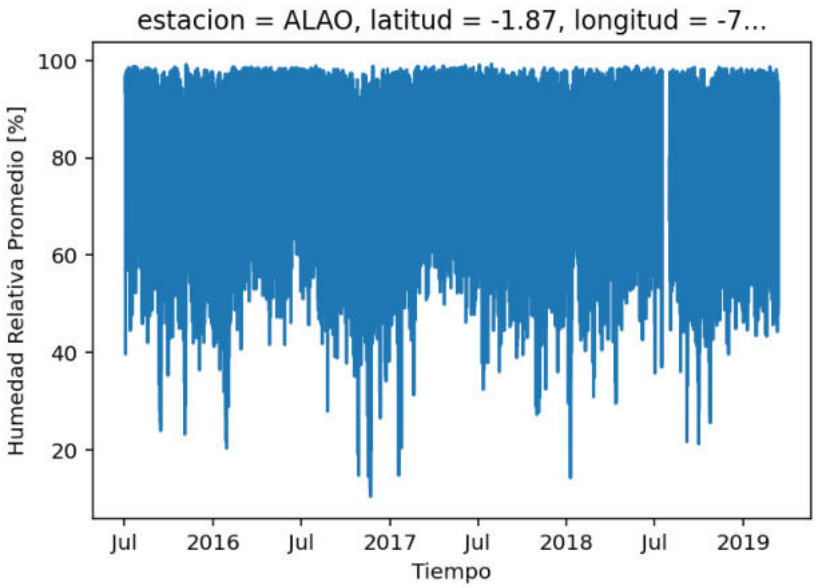


Figura 8.19. Serie temporal de humedad relativa promedio por hora de la estación ALAO desde el 3 de julio de 2015 hasta el 15 de marzo del 2019.

Fuente: elaboración propia.

En la Figura 8.19 se observa el comportamiento de la variable humedad relativa en la estación de Alao, los valores de humedad relativa están concentrados por encima de la media, se observa unos picos periódicos de mediciones de humedad por debajo de la media, que se interpretan como los periodos de días secos, que es típico del comportamiento estacional de la región Andina.

Se considera los datos de la temperatura ambiente de la estación meteorológica de la ESPOCH, para analizar los rangos superiores e inferiores, que aún representan un error, pero aún se mantienen, probablemente debido a que el instrumento siguió funcionando correctamente por lo que no se determinó como un *"INVALID"* y el *datalogger* lo registra como ceros, valores extremadamente bajos o altos. Se filtra la información y se organiza por hora y se grafica.

```
# filtrar los datos

ds_Espoch = ds['TA Avg'].sel(estacion='ESPOCH')

# Convertir a Dataframe

da = ds['TA Avg'].sel(estacion='ESPOCH')

df = da.to_dataframe().reset_index()

df = df[['Tiempo','TA Avg']]

# Eliminar datos nulos

df_limpio = df.dropna(subset=['TA Avg'])
```

```
# Mantener solo la hora del día

df_limpio["Tiempo"] = df_limpio["Tiempo"].dt.strftime("%H:00")

fig, axs = plt.subplots(1, 1,figsize=(16, 8),sharex=True)

axs.scatter(df_limpio['Tiempo'],df_limpio['TA Avg'])
```

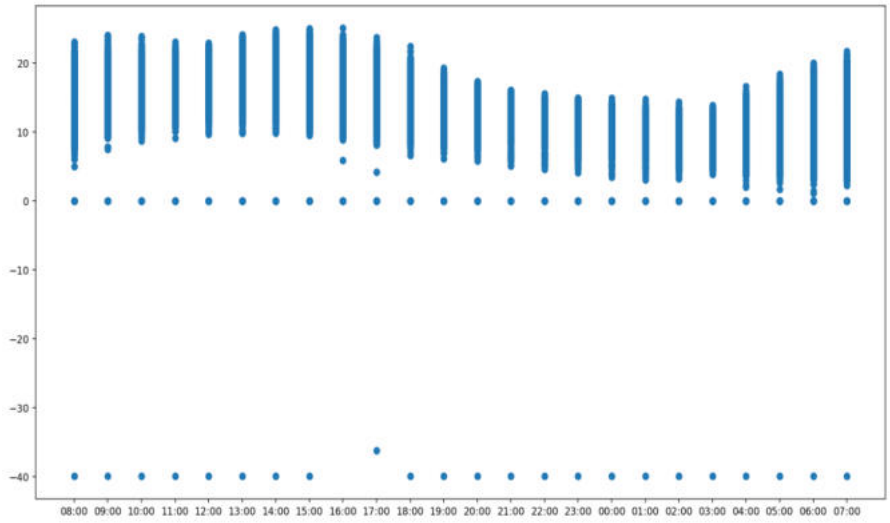


Figura 8.20. Nube de puntos de la temperatura promedio en la estación ESPOCH reorganizado por hora del día.

Fuente: elaboración propia.

Se observa en la Figura 8.20 que existen valores excesivamente bajos como para ser considerados apropiados en la zona de estudio. Se corrigen las observaciones de temperatura incoherentes mediante:

```
ds_Espoch = ds['TA Avg'].sel(estacion='ESPOCH')
```

```
da = ds['TA Avg'].sel(estacion='ESPOCH')

mask = da <= 0

da_limpio = da.where(~mask,np.nan)

df_limpio = df_limpio.reset_index()

L = []

for i in range(len(df_limpio)):

    if df_limpio.loc[i,'TA Avg']<=0:

        L.append(i)

#eliminar las filas

df_filtrado = df_limpio.drop(L, axis = 0).reset_index(drop=True)

fig, axs = plt.subplots(1, 1,figsize=(16, 8),sharex=True)

axs.scatter(df_filtrado['Tiempo'],df_filtrado['TA Avg'])
```

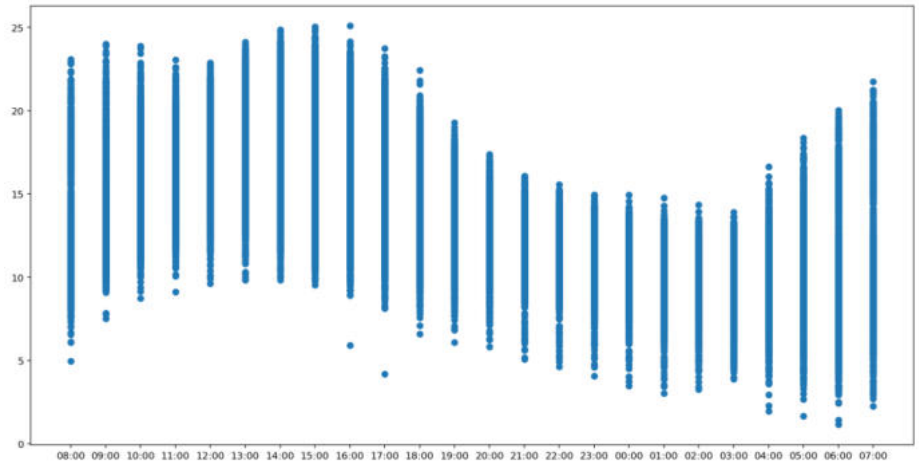


Figura 8.21. Nube de puntos de la temperatura promedio en la estación ESPOCH reorganizado por hora del día luego de filtrar por una temperatura mínima.

Fuente: elaboración propia.

La Figura 8.21 muestra el comportamiento coherente de la temperatura ambiente mediante los datos filtrados. Es relevante mencionar que existen dos picos de temperaturas altas entre las 08h00 y 10h00 de la mañana, mientras que el segundo se encuentra entre las 14h00 y 16h00 de la tarde. Mientras que las bajas temperaturas se encuentran por lo general en la madrugada destacando el punto más bajo entre las 04h00 y 06h00 de la madrugada. La temperatura promedio más alta registrada es de 25 °C y la más baja de aproximadamente 2°C.

8.5. Detección de patrones climáticos

Para el proyecto IDIPI-306 se propuso la implementación de algoritmos de detección de Weather Types, los cuales identifican patrones atmosféricos

regionales. Estos procedimientos se proyectan al análisis de datos bidimensionales, por lo cual se utilizan los CHIRTS o CHIRPS, se explica paso a paso cómo obtener regímenes de clima en una región a partir de esta información. Se proporciona un estudio robusto de la variabilidad y predicción climática a escala regional en la provincia de Chimborazo con los registros históricos. La metodología propuesta se apoya en los enfoques previamente documentados en la literatura:

- Muñoz, Á. G., Goddard, L., Robertson, A. W., Kushnir, Y., & Baethgen, W. (2015). Cross–time scale interactions and rainfall extreme events in southeastern South America for the austral summer. Part I: Potential predictors. *Journal of Climate*, *28*(19), 7894-7913.

- Muñoz, Á. G., Goddard, L., Robertson, A. W., Kushnir, Y., & Baethgen, W. (2015). Cross–time scale interactions and rainfall extreme events in southeastern South America for the austral summer. Part I: Potential predictors. *Journal of Climate*, *28*(19), 7894-7913.

- Michelangeli, P. A., Vautard, R., & Legras, B. (1995). Weather regimes: Recurrence and quasi stationarity. *Journal of Atmospheric Sciences*, *52*(8), 1237-1256.

8.5.1. Obtención de datos y preprocesamiento

Se utiliza las bases de datos climáticas globales a través del protocolo *OPeNDAP* mediante la función *get_Chirts()* se extrae los valores de temperatura máxima diaria provenientes del producto CHIRTS, el cual tiene un conjunto global de observaciones a 0.05° de resolución espacial. Para

ello, se define la dirección URL específica del servidor *IRI/LDEO* y se abre el archivo mediante la librería xarray, se manipula la información directamente en la nube como un objeto tipo Dataset. De forma análoga, la función *get_Chirps()* accede al producto CHIRPS, que corresponde a precipitaciones diarias a nivel global con la misma resolución espacial.

```
def get_Chirts():

    # El url del opendap es distinto para chirts y para chirps

    # URL OPeNDAP, si está disponible para tmax

    opendap_url =
'http://iridl.ldeo.columbia.edu/SOURCES/.UCSB/.CHIRTS/.v1.0/.daily/.gl
obal/.0p05/.tmax/dods'

    # Abrir el dataset de forma remota

    dsChirts = xr.open_dataset(opendap_url)

    return dsChirts

def get_Chirps():

    opendap_url =
'http://iridl.ldeo.columbia.edu/SOURCES/.UCSB/.CHIRPS/.v2p0/.daily-
improved/.global/.0p05/prcp/dods'

    # Abrir el dataset de forma remota

    dsChirps = xr.open_dataset(opendap_url)
```

```
return dsChirps
```

Las funciones proporcionan como salida un objeto *Dataset* que constituyen la información satelital de temperatura y precipitación. Se establece la región de dónde se extrae la información, estableciendo puntos en coordenadas decimales. Al revisar el *dataset* generado, se observa que la coordenada T (tiempo) tiene valores de dimensiones 10^6, esto se debe a que el tiempo de estas bases de datos están en días julianos, esto se trata de una cuenta continua de días y fracciones de día desde un punto de inicio fijo.

```
<xarray.Dataset> Size: 930GB

Dimensions:  (X: 7200, T: 12419, Y: 2600)

Coordinates:

  * X        (X) float32 29kB -180.0 -179.9 -179.9 -179.8 ... 179.9 179.9 180.0

  * T        (T) float32 50kB 2.445e+06 2.445e+06 ... 2.458e+06 2.458e+06

  * Y        (Y) float32 10kB -59.97 -59.92 -59.88 -59.82 ... 69.88 69.92 69.97

Data variables:

    tmax     (T, Y, X) float32 930GB ...

Attributes:

Conventions:  IRIDL
```

El punto de día de inicio en los días julianos es el 1 de enero del año 4713 a. C. del calendario juliano proléptico[5], desde ahí cada día transcurrido tiene su conteo. Se ha creado una función que convierte este sistema de tiempo en días gregorianos, y depués en datos de tipo *datetime*. El código que realiza la transformación es:

```
def convert_JulToGreg(self,ds):

    L = ds['T'].data

    fechas = []

    for i in range(len(L)):

        ini_Jul = int(L[i] - 1721426  )

        date = datetime(1,1,1,12,0,0,0) + timedelta(days=ini_Jul)

        fechas.append(date)

    ds_nuevo = ds.assign_coords(T=np.array(fechas))

    return ds_nuevo
```

En el presente caso de estudio, se extraen los datos de una región que limite a la Sierra centro del Ecuador, en este caso los puntos son P1 = -2.85,-79.61 y P2 = -0.81,-77.71 (véase la Figura 8.22).

[5] Este tipo de calendario es utilizado en modelos CMIP 6 y hace referencia a la extensión retroactiva del calendario juliano hacia fechas anteriores desde que fue instaurado.

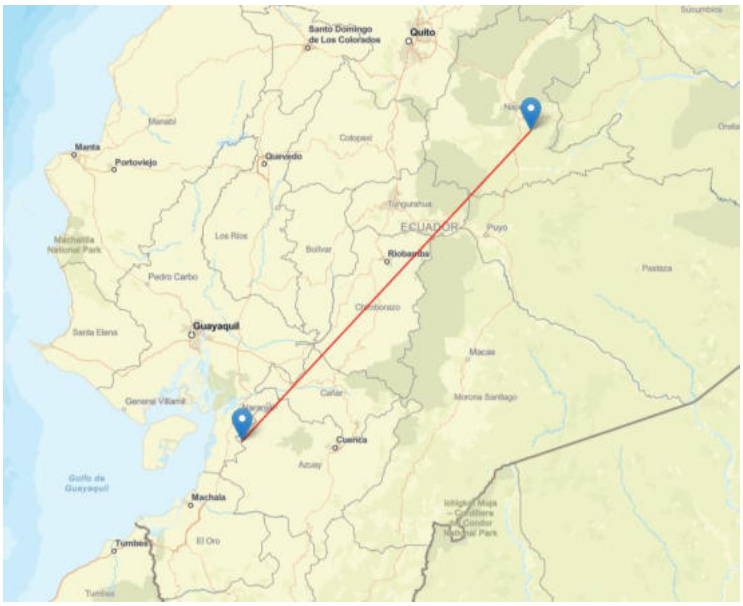


Figura 8.22. Puntos de extracción de datos para los CHIRTS y CHIRPS.

Fuente: elaboración propia.

Luego el filtrado de una región a través del método *sel,* considerando establecer cuáles son los valores mínimo y máximo en las coordenadas de latitud y longitud:

```
dsRegion_Chirts =
dsChirts.sel(Y=slice(P1[0],P2[0]),X=slice(P1[1],P2[1]))

dsRegion_Chirps=
dsChirps.sel(Y=slice(P2[0],P1[0]),X=slice(P1[1],P2[1]))
```

Los dataset finales son descargados y almacenados en el computador a través de to_netcdf('Nombre_del_Archivo.nc'):

dsRegion_Chirts

Out:

<xarray.Dataset> Size: 77MB

Dimensions: (X: 38, Y: 41, T: 12419)

Coordinates:

** X (X) float32 152B -79.58 -79.53 -79.48 ... -77.83 -77.78 -77.73*

** Y (Y) float32 164B -2.825 -2.775 -2.725 ... -0.925 -0.875 -0.825*

** T (T) datetime64[ns] 99kB 1983-01-01T12:00:00 ... 2016-12-31T12:00:00*

Data variables:

tmax (T, Y, X) float32 77MB ...

Attributes:

Conventions: IRIDL

dsRegion_Chirps

Out:

<xarray.Dataset> Size: 65kB

Dimensions: (X: 38, T: 16314, Y: 0)

Coordinates:

* * X (X) float32 152B -79.58 -79.53 -79.48 ... -77.83 -77.78 -77.73

* * T (T) float32 65kB 2.445e+06 2.445e+06 ... 2.461e+06 2.461e+06

* * Y (Y) float32 0B

Data variables:

prcp (T, Y, X) float32 0B ...

Attributes:

Conventions: IRIDL

Una limitación espacial más exacto de la región en estudio, se consigue a través de un *shape,* por ejemplo, en el estudio del proyecto se ha usado el archivo .shp de la provincia de Chimborazo y se aplica sobre los datos que se tiene y con el método *plot()* revisar la información.

Nota. Una buena práctica antes de usar estos métodos es establecer el sistema de coordenadas al dataset que se quiere recortar y al archivo shape importado.

```
shp = gpd.read_file("Chimborazo.shp")

ds = region_Chirts.rio.write_crs("EPSG:4326")

cut_Clip = shp.to_crs("EPSG:4326")

var = list(ds.data_vars.keys())[0]
```

```
da = ds[var]

# aplicar el recorte

da_clip = da.rio.clip(cut_Clip.geometry, cut_Clip.crs, drop=True)

da_clip[0].plot()
```

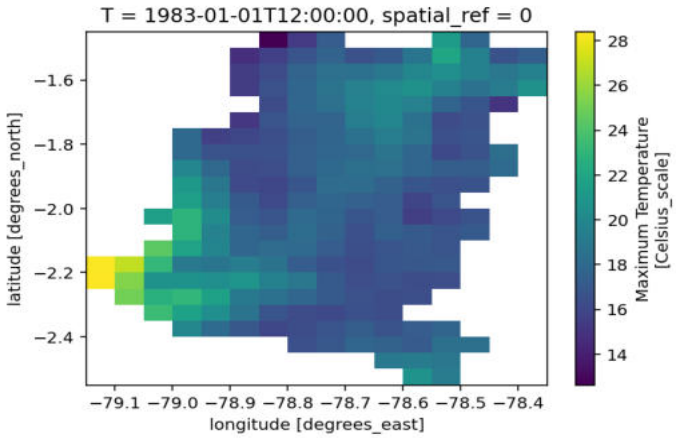


Figura 8.23. Visualización de los datos CHIRTS luego del recorte de la provincia de Chimborazo usando un archivo shape.

Fuente: elaboración propia.

Como muestra la Figura 8.23. los CHIRTS son generados mediante asimilación de datos in situ, produciendo pixeles contaminados o sin datos, debido a que en esas zonas no hay información de las estaciones meteorológicas disponibles. Se revisa los valores máximos y mínimos de la información obtenida, se aplican los métodos *max()* y *min()* respectivamente:

region_Chirts.max()

Out:

<xarray.Dataset> Size: 4B

Dimensions: ()

Data variables:

tmax float32 4B 35.38

region_Chirts.min()

Out:

<xarray.Dataset> Size: 4B

Dimensions: ()

Data variables:

tmax float32 4B 6.445

El umbral de temperatura depende del propósito específico del análisis (por ejemplo, detectar olas de calor, días cálidos extremos, cultivos sensibles, salud pública, etc.). Sin embargo, para la provincia de Chimborazo, Ecuador, con su variabilidad altitudinal (desde ~1.200 msnm hasta más de 6.000 msnm), se establecen rangos de referencia generales:

Tabla 8.4. Resumen de las características físicas y climáticas de la provincia de Chimborazo.

Zona	Altitud (aprox.)	Temperatura media (°C)	Posible umbral extremo
Valle bajo (Cumandá)	< 2.000 msnm	20–25 °C	>30 °C (calor extremo)
Zona media (Riobamba, Colta)	2.000–3.000 msnm	12–18 °C	>25 °C (calor extremo)
Alta montaña (Alausí, Guamote alto)	> 3.000 msnm	6–12 °C	>20 °C (inusual calor)

Fuente: elaboración propia.

Se delimita un umbral superior provisional de 30°, según las referencias generales de la región. Para realizar estos ajustes se ejecutan los siguientes comandos:

```
da_redondeado = da_filtrado.round(decimals=4)
```

```
da_filtrado = da.where(da <= 30)
```

8.5.2. Preparación de la información

Una vez obtenido la información de la región, se realiza un análisis para determinar cuáles son los patrones regionales de temperatura existentes. Para esto, será necesario acudir al concepto de recurrencia sustentado en el trabajo de (Michelangeli et al. 1995), quienes demostraron que los sistemas atmosféricos tienden a presentar recurrencia y cuasi-estacionariedad, es decir, que ciertos estados de la circulación se repiten con frecuencia en el tiempo y mantienen relativa persistencia antes de su transición hacia otros

regímenes. La recurrencia se convierte, por tanto, en el fundamento estadístico que justifica el uso de técnicas de clasificación no supervisada para establecer patrones robustos de circulación y, en consecuencia, de condiciones climáticas asociadas.

En la base de datos de temperaturas máximas se aplica clustering sobre series de datos atmosféricos multivariados, permitiendo descomponer la variabilidad diaria en un número reducido de tipos característicos de clima. Cada uno de estos tipos resume las configuraciones dominantes que, al repetirse a lo largo de una serie temporal extensa, adquieren relevancia como patrones representativos de la región. Este enfoque metodológico validado y aplicado en diferentes regiones, como lo muestran (Muñoz et al., 2015) en su estudio sobre el sureste de Sudamérica (SESA), los autores emplearon un análisis k-means para identificar un conjunto robusto de regímenes de circulación diaria, los cuales fueron vinculados con la ocurrencia de eventos extremos.

Con la información obtenida se calcula las anomalías diarias, que se define como una diferencia entre un valor observado (por ejemplo, la temperatura o precipitación en un día específico) y un valor de referencia o "normal" (como el promedio climático histórico para ese mismo día o mes). Por ejemplo, si el 5 de agosto de 2005 la temperatura fue de 22°C, y el promedio histórico del 5 de agosto entre 1983-2010 fue 20°C, entonces la anomalía es de +2°C. El estudio climatológico a través de las anomalías ofrece varias ventajas como: la eliminación de la estacionalidad, la identificación de eventos extremos y compatibles con modelos climáticos.

Para construir una base de datos de anomalías se calcula para cada punto regional (es decir cada píxel) y en cada día del año el promedio global. Para la implementación y automatización, se crea un algoritmo que recibe un *DataSet* con los valores de las variables, se realiza con los CHIRTS y CHIRPS de la región Sierra Centro de Ecuador, desde 1983 hasta 2016. A partir de estos datos se construye la climatología diaria (valor medio para cada día del año) en cada píxel, luego para cada día del año (1 de enero, 2 de enero, ..., 31 de diciembre), se calcula el promedio de todos los años disponibles para ese día. La función que automatiza este proceso se llama *calcular_Anomalias(ds)* y devuelve un *dataset* con las anomalías espaciales diarias.

```
def calcular_Anomalias(ds):

    # Eliminar el 29 de febrero de todos los años

    mask = ~((ds["T"].dt.month == 2) & (ds["T"].dt.day == 29))

    ds = ds.sel(T=ds["T"][mask])

    # Climatología diaria

    ds_Diario = ds.groupby("T.dayofyear").mean("T")

    # Expandir la climatología al mismo tamaño del original

    anomalies = ds.groupby("T.dayofyear") - ds_Diario

    return anomalies
```

Para realizar la identificación de regímenes de clima mediante *clustering*, se requiere un *reshape* de la base de datos. Los valores obtenidos en la malla de las anomalías diarias se reordenan en una sola fila, esto quiere decir que las columnas son los valores de una latitud y longitud. Se obtiene una matriz de información bidimensional donde cada fila representa un día de información para las variables de temperatura y precipitación, para concatenarlos (Ver Figura 8.24). Se crea un algoritmo que recibe una lista de *datasets*, una fecha inicial y final con el formato "YYYY-MM-DD". El código realiza un recorte en función de ese intervalo, asegurando que solo se conserven los datos del período de interés y determina la región espacial común entre todos los archivos (mínimos y máximos de latitudes y longitudes) para garantizar que el área seleccionada tenga dimensiones consistentes. Una vez identificada esta intersección espacial, cada dataset es filtrado y convertido en un arreglo NumPy; que es tridimensional (días × filas × columnas) y luego se transforma a una forma bidimensional (días × puntos espaciales). Finalmente, todos los arreglos resultantes se concatenan, se reordenan mediante transposición y se integran en una única matriz de características.

```
def get_Arrays(L,Fini,Ffin):

DTini = datetime.strptime(Fini,"%Y-%m-%d")

  DTfin = datetime.strptime(Ffin,"%Y-%m-%d") + timedelta(days=1)

  for i in range(len(L)):

  L[i] = L[i].sel(T=slice(DTini,DTfin))
```

```
# extraer el cuadro región donde se realizará la unión de arreglos
# para que tengan las mismas dimensiones
lons_Min,lons_Max = [],[]
lats_Min,lats_Max = [],[]
for elem in L:
    lons_Min.append(min(elem.X))
    lons_Max.append(max(elem.X))
    lats_Min.append(min(elem.Y))
    lats_Max.append(max(elem.Y))
# recorrer la lista L para crear un solo arreglo bidimensional
arr_tot = []
for elem in L:
# Filtrar la region
    elem_Region = elem.where(
    (elem.Y >= max(lats_Min)-0.01) &
    (elem.Y <= min(lats_Max)+0.01) &
    (elem.X >= max(lons_Min)-0.01) &
```

```
    (elem.X <= min(lons_Max)+0.01),drop=True)
    # crear el arreglo
    var = list(elem.data_vars.keys())[0]
    arr_Elem = elem_Region[var].compute().data
    # relizar un reshape del arreglo 3-D a 2-D
    dias,fils,cols = arr_Elem.shape
    X = arr_Elem.reshape(dias,fils*cols)
    X = repair(X)
    # agregar al arr_tot
    arr_tot.append(X)
arr_Tot = np.array(arr_tot)
arr_Reshaped = arr_Tot.transpose(1, 0, 2)
return arr_Reshaped.reshape(dias, -1)
```

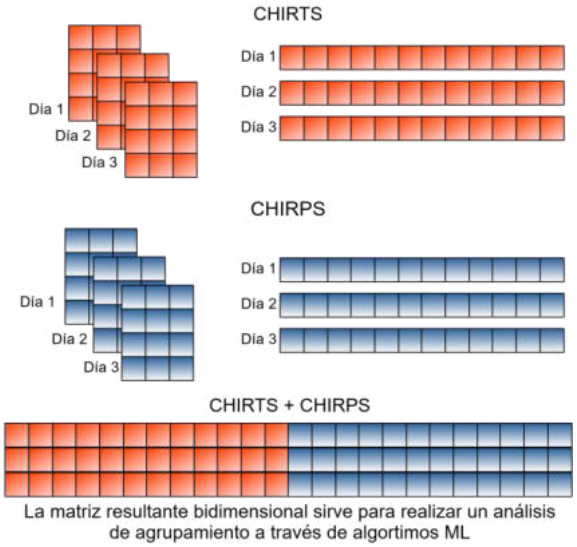


Figura 8.24. Proceso de transformación de las mallas de datos CHIRTS y CHIRPS a una matriz bidimensional.

Fuente: elaboración propia.

La matriz de datos bidimensional obtenida tiene un excesivo número de columnas, las variables de precipitación y temperatura en la región de Chimborazo ocupan un total de 1000 columnas, que representan las características. Se aplica reducción de dimensionalidad mediante la técnica de componentes principales, con el objetivo de encontrar el número de vectores ortogonales empíricos que contenga la información de varianza adecuada. Ver Figura 8.25.

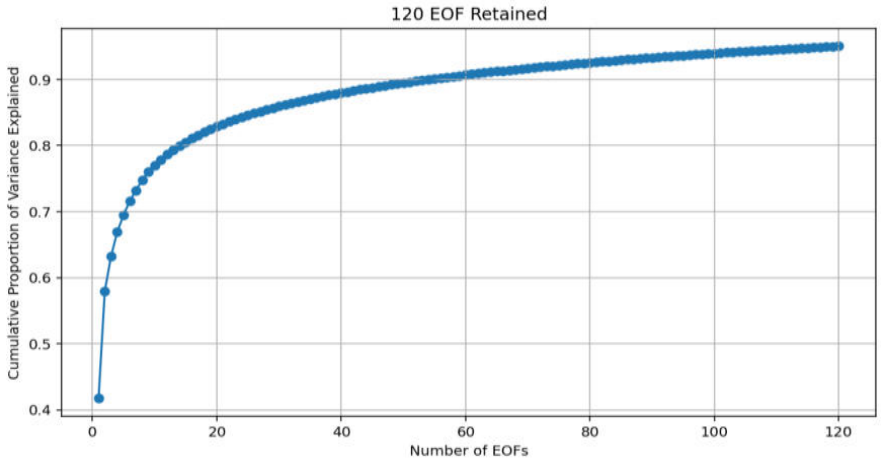


Figura 8.25. Varianza acumulada con distinto número de vectores empíricos ortogonales para el modelo de datos Chirts y Chirps concatenados.

Fuente: elaboración propia.

Se aplica PCA con el siguiente código:

pca = PCA(n_components=n_eof)

model_PCA = pca.fit(X)

arr_PCA = model_PCA.transform(X)

El resultado muestra que 60 componentes principales abarcan el 90% de la varianza acumulada de la información, se aplica el método de k-means para identificar los patrones de comportamiento de la región de Chimborazo.

arr_PCA

Out:

```
array([[ 9.56507187e+01, -1.32058838e+02,  8.04166555e-01, ...,
        -3.39822221e+00,  1.87072051e+00,  6.04926634e+00],
       [ 3.05069458e+02, -1.94072220e+02, -4.46354103e+01, ...,
        -4.85556602e+00, -4.00521421e+00, -3.73208904e+00],
       [ 2.64531097e+02,  1.43952465e+01, -1.07291542e+02, ...,
         3.23216939e+00,  2.37949729e+00, -5.02950907e+00],
       ...,
       [-5.53558807e+01, -1.81634712e+01,  3.49976134e+00, ...,
         1.76511824e+00,  6.29325509e-01,  1.72344756e+00],
       [ 6.10773682e+02,  2.67822968e+02,  1.92466797e+02, ...,
        -4.82652473e+00, -6.96560812e+00,  4.96170139e+00],
       [-6.61593323e+01, -4.49353981e+00,  1.38881226e+01, ...,
        -1.27266693e+00,         -4.42597777e-01,         -1.40933752e+00]],
dtype=float32)
```

arr_PCA.shape

Out: (12419, 120)

model_PCA

Out[13]: PCA(n_components=120)

8.5.3. Weather Types de la provincia de Chimborazo

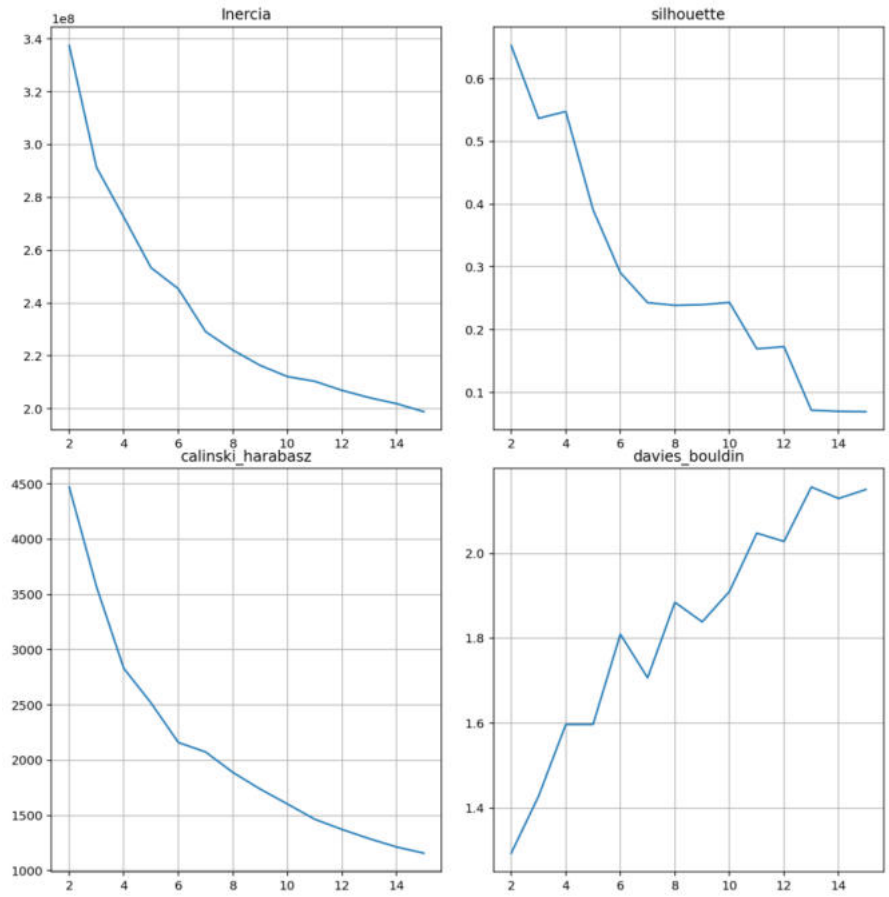


Figura 8.26. Índices para determinar el mejor número de clústeres de los datos de precipitación y temperatura con la matriz reducida a través de PCA.

Fuente: elaboración propia.

Se analiza las métricas de validación del modelo K-Means, en la Figura 8.26. se observa que la inercia presenta un marcado cambio de pendiente entre 3 y 5 clústeres, lo que indica que a partir de ese punto las ganancias en la reducción de la varianza interna son cada vez menores. El índice de Silhouette alcanza su valor máximo en dos clústeres, pero a partir de cuatro o cinco aún conserva valores aceptables antes de decaer de manera significativa. El índice de Calinski–Harabasz muestra su mayor valor en dos clústeres, aunque a partir de cuatro o cinco la disminución es más gradual, lo que sugiere una partición más consistente. El índice Davies–Bouldin registra los mejores valores en el rango de dos a cuatro clústeres, lo que evidencia problemas de sobreajuste en soluciones con un número elevado de grupos. En síntesis, aunque las métricas tienden a favorecer soluciones con dos clústeres, en el contexto del análisis climático este número es insuficiente para capturar la diversidad de regímenes atmosféricos. Por ello, un rango entre cuatro y cinco clústeres se presenta como la elección más adecuada, debido que representa los patrones climáticos regionales sin comprometer la coherencia estadística del modelo. En el caso de este estudio se considera cinco clústeres que representan los patrones de comportamiento climático en la región.

En la metodología *propuesta por Muñoz et al. (2015)*, para la identificación de los patrones climáticos *se aplica algoritmos de clustering*, como *K-Means*. Sin embargo, estos dependen de inicializaciones aleatorias de los centroides, generando resultados distintos en cada ejecución, afectando la estabilidad de los clústeres obtenidos. Por ello, es necesario repetir el proceso múltiples veces y analizar la consistencia de los centroides a través del índice de clasificabilidad propuesto por Michelangeli, Vautard y Legras

(1995). Consiste en cuantificar la partición del modelo calculando una métrica de similitud entre dos conjuntos de centroides usando la correlación de Pearson entre las coordenadas correspondientes a cada centroide.

```
def loop_kmeans(X,n_cluster,n_sim):
    centroids = np.zeros(shape=(n_sim, n_cluster, X.shape[1]))
    w_types = np.zeros(shape=(n_sim, X.shape[0]))
    for i in np.arange(n_sim):
        km = KMeans(n_clusters=n_cluster).fit(X)
        centroids[i, :, :] = km.cluster_centers_
        w_types[i, :] = km.labels_
    return centroids, w_types
```

La función *loop_kmeans* recibe una matriz de datos procesados (X), el número de clústeres a formar (*n_cluster*) y la cantidad de simulaciones a ejecutar (*n_sim*). En cada iteración, se entrena un modelo *K-Means* con los parámetros específicos, se almacenan los centroides de los clústeres generados y las etiquetas asignadas a cada observación. Al finalizar, la función devuelve dos arreglos: uno que contiene los centroides de todos los experimentos y otro que guarda las etiquetas correspondientes a cada ejecución, información utilizada para calcular el índice de clasificabilidad de cada agrupación a través del siguiente código:

```
def _vector_ci(P,Q):
```

```
    k = P.shape[0]

    Aij = np.ones([k, k])

    for i in range(k):

        for j in range(k):

            Aij[i, j] = np.corrcoef(P[i, :], Q[j, :])[0, 1]

    Aprime = Aij.max(axis=0)

    ci = Aprime.min()

    return ci
```

El resultado *(ci)* muestra el índice de clasificabilidad y cuanto más cercano esté el índice a 1, mayor será la consistencia entre simulaciones. Entonces se debe calcular el *ci* de cada modelo de *clustering* generado anteriormente en *centroids* y *w_types,* este proceso se realiza por cada agrupamiento. A continuación, se muestra un algoritmo que automatiza todo este proceso, simplemente ingresando la matriz de los centroides:

```
def get_classifiability_index(centroids):

    nsim = centroids.shape[0] #Obtiene el numero de simulaicones

    #Inicializa una matriz para almacenar los indices de clasificabilidad

    c_pq = np.ones([nsim, nsim])
```

```
    #Calculo de las distancias/indice de clsificabilidad usando la funcion _vector_ci
    for i in range(0, nsim):
        for j in range(0, nsim):
            if i == j:
                c_pq[i, j] = np.nan
            else:
                c_pq[i, j] = _vector_ci(P=centroids[i, :, :], Q=centroids[j, :, :])
    #calculo del promedio ignorando los nulos
    classifiability = np.nanmean(c_pq)
    best_part = np.where(c_pq == np.nanmax(c_pq))[0][0]
    return best_part
```

La salida del código anterior guardada como *best_part* tiene los valores de los centroides con mejor calificación de partición. Estos centroides son utilizados para crear el nuevo modelo de *clustering*:

```
best_fit = KMeans(n_clusters=n_clusters, init=centroids[best_part, :, :], n_init=1, max_iter=1).fit(arr_PCA)
```

Los centroides son representados de nuevo en datos de temperatura y precipitación, para lo cual se utiliza el mismo modelo PCA creado durante el proceso de reducción de dimensiones.

Wts = model_PCA.inverse_transform(arr_Wts_pca)

Al final se tiene la misma matriz bidimensional de mil columnas las cuales deben separarse en dos, es decir el proceso contrario al que se vio en la Figura 8.26, luego separar los datos en la parte que representa temperatura y la de precipitación para poder representarlas usando Python juntos con *matplotlib* y dibujar el *shape* de Chimborazo sobre la imagen mediante el método *shape.boundary.plot(ax).*

Los resultados corroboran que los *tipos de clima* identificados son patrones reales del sistema climático.

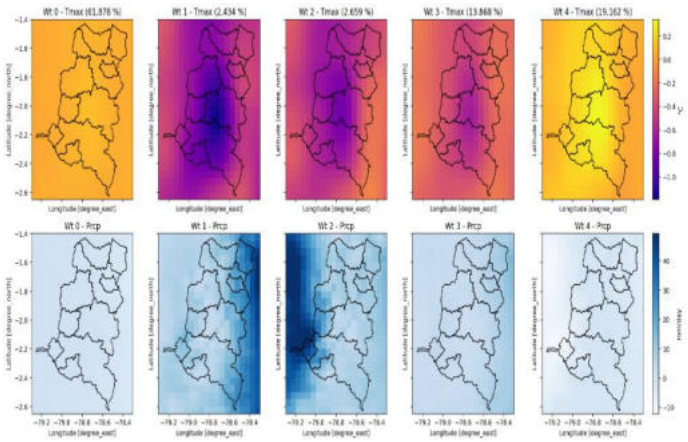


Figura 8.27. Tipos de clima detectados en la provincia de Chimborazo con cinco grupos en un modelo de K-Means.

Fuente: elaboración propia.

La Figura 8.27. muestra los patrones climáticos o también conocidos como *Weather Types (WTs)* en la provincia de Chimborazo, mismos que han sido identificados a partir del análisis de la temperatura máxima (*Tmax*) y la precipitación (*Prcp*). Cada panel representa un tipo de clima distintivo acompañado del porcentaje de ocurrencia en el periodo analizado.

En la primera fila se observa la distribución de anomalías de la temperatura máxima. El WT0 domina claramente con un 61,878 % de ocurrencia, lo que indica que es el régimen climático predominante, por lo que se identifica como la temperatura y comportamiento de precipitación típicas de la zona. En contraste, los regímenes WT1 y WT2 son poco frecuentes (2,434 % y 2,659 % respectivamente) por lo que se consideran como los regímenes que representan valores extremos, se distinguen por presentar un gradiente térmico más marcado de este a oeste, lo cual se asocia a intrusiones puntuales de masas de aire frío o cambios en la circulación atmosférica de corta duración. En estos mismos regímenes se detectan acumulaciones de precipitación más concentradas en la parte centro-este de la provincia en el caso del WT1, y de centro-oeste según el WT2 posiblemente ligado a la convergencia de humedad procedente de la Amazonía. Ambos *weather types* coinciden con su baja frecuencia, pero alta relevancia, ya que estos regímenes, aunque raros, parecen estar vinculados a eventos de lluvia intensa.

El WT3, con un 13,868 %, muestra un patrón intermedio con áreas más frías en la zona central y lluvias en la zona oriental, mientras que el WT4 con 19,162 % representa un régimen relativamente cálido y persistente, posiblemente ligado a fases estacionales cálidas; además se caracteriza por

presencia de precipitaciones bajas, similar a WT0, lo que lo convierte en un régimen cálido y relativamente seco.

Los resultados del tipo de clima se reordenan en una malla bidimensional donde el eje de la ordenada representa los años, mientras que el eje la abscisa el día del año y el resultado será el diagrama de Klee. Esta Figura 8.28 muestra el comportamiento climático según los WTs detectados en todo el periodo de estudio para observar patrones de temporalidad de cada régimen.

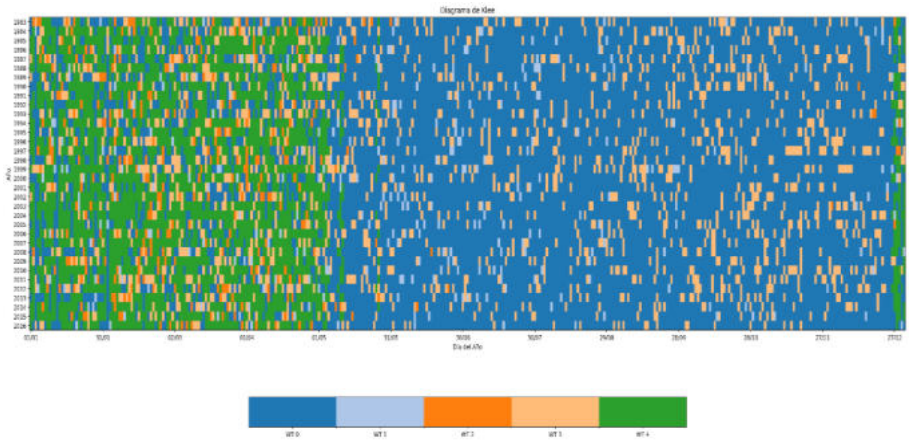


Figura 8.28. Diagrama de Klee del modelo de weather types de la figura 8.27.

Fuente: elaboración propia.

En el diagrama de Klee de la Figura 8.28. se muestra la recurrencia y persistencia temporal de los *Weather Types* (WTs) identificados. Los colores asignados a cada bloque corresponden a los distintos regímenes climáticos (WT0 a WT4), facilitando la visualización de la distribución a lo

largo del tiempo. Se puntualiza sobre la predominancia del WT0 (azul oscuro), considerando este régimen como un “estado base” del sistema climático regional.

El WT1 (celeste claro tiene una recurrencia muy baja sin una distribución uniforme anual, reflejando un régimen poco frecuente, pero climáticamente importantes, posiblemente vinculados a situaciones sinópticas específica. Por otro lado, se observan irrupciones en la primera mitad del año al régimen WT2, que corresponden a patrones asociados a lluvias significativas desde la costa. Mientras que los eventos del WT3 se distribuye de forma dispersa durante el año. Por último, el WT4 (verde), aparece al final del año hasta el mes de mayo con mayor frecuencia en la serie anual, lo que podría asociarse a un comportamiento estacional de la temporada seca.

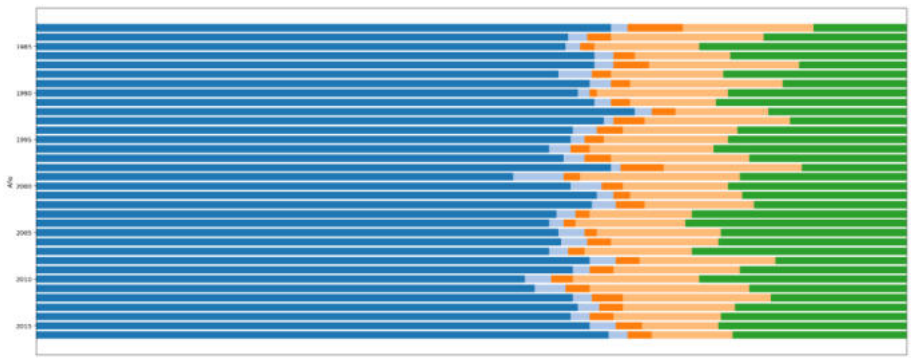


Figura 8.29. Frecuencia en días de cada WT por año.

Fuente: elaboración propia.

Las acumulaciones anuales de cada WT muestran una relación con fenómenos climáticos de importancia como son los eventos de El niño y La

Niña (ver Figura 8.29.). El fenómeno de El Niño es caracterizado por un incremento de lluvias en la región andina, se observa un aumento relativo en la ocurrencia de WT1, WT2 y WT3, que coincide con la intensificación de precipitaciones y la mayor advección de humedad desde la Amazonía o el Pacífico, corroborando que estos regímenes están asociados a episodios de lluvia intensa.

En la literatura científica (Nouvelot & Pourrut, 1984) describe que desde octubre de 1982 a septiembre de 1983 se registraron lluvias muy intensas en la franja costera del Ecuador, las cuales fueron provocadas por el fenómeno de El Niño. Además, puntualizan que su aparición aperiódica produce notables modificaciones en las condiciones climáticas y biológicas de la región, con consecuencias económicas en ocasiones catastróficas. La gráfica 8.22 muestra en ese año un incremento sustancial de los regímenes WT1, WT2 y WT3, los cuales reemplazan al WT4. De la misma manera, en el diagrama se observa un incremento de precipitación en 1997 y 1998, lo cual coincide con el registro del fenómeno del Niño, el cual cobró la vida de al menos 286 personas y unas 30.000 se vieron gravemente afectadas en Ecuador (Vos et al., 1999).

También se observa un aumento de los WT1 al WT3 en el año 1999, pero en esta ocasión el WT0 se reduce considerablemente mientras que el régimen de clima cálido se mantiene normal (WT4) a diferencia de lo que sucedía en el fenómeno del Niño. Este mismo comportamiento ocurre en los años de 1988, 2010 y 2011, al comparar con el valor mínimo del IOS (Índice de la oscilación del Sur) señalando los valores máximos en cada fase

presentado por Pabón Caicedo y Montealegre Bocanegra (2017)se observa que coinciden con los eventos de la Niña:

- De mayo 1988 a mayo de 1989, con duración de 13 meses, IOS igual a 1.8
- De julio 1998 a abril 1999, con duración de 11 meses, IOS igual a 1.8
- De octubre de 1999 a abril del 2000, con duración de 7 meses, IOS igual a 1.7
- De octubre de 2007 hasta abril del 2008, con duración de 7 meses, IOS igual a 2.6
- De agosto de 2008 a abril del 2009, con duración de 9 meses, IOS igual a 1.9

Se destaca que en estos periodos el WT1 aumenta más proporción que el WT2 y revisando la gráfica 8.20 se logra evidenciar que el WT1 tiene como característica bajas temperaturas, que coincide con el aumento de los vientos alisios y el enfriamiento anormal de las aguas del océano Pacífico.

GLOSARIO

CMIP6 (Coupled Model Intercomparison Project Phase 6): Conjunto de datos de temperatura que integra información satelital e in situ para generar series temporales globales.

CSV (Comma-Separated Values): Formato de archivo de texto que almacena datos tabulares separados por comas, ampliamente utilizado para intercambio de datos.

CHIRPS (Climate Hazards Group InfraRed Precipitation with Station Data): Producto climático que combina datos satelitales y estaciones para estimar la precipitación a alta resolución espacial.

CHIRTS (Climate Hazards Group InfraRed Temperature with Stations): Producto climático que combina datos satelitales y estaciones para estimar la precipitación a alta resolución espacial.

DMC (Data Mining Consulting): Formato de archivo de texto que almacena datos tabulares separados por comas, ampliamente utilizado para intercambio de datos.

ENSO (– El Niño–Southern Oscillation): Organización o entorno de trabajo enfocado en análisis de datos y minería de datos aplicada.

FFT (Fast Fourier Transform): Algoritmo eficiente para calcular la transformada de Fourier, utilizado en análisis de señales y series temporales.

GIS (Geographic Information System): Sistema que permite capturar, almacenar, analizar y visualizar datos geoespaciales.

INAMHI (Instituto Nacional de Meteorología e Hidrología): Entidad nacional encargada del monitoreo y estudio de variables meteorológicas e hidrológicas en Ecuador.

IPython (Interactive Python): Entorno interactivo para ejecutar código Python de manera dinámica.

IQR (Interquartile Range): Medida estadística que representa la dispersión de los datos entre el primer y tercer cuartil.

LML (Local Meteoric Line): Relación lineal entre isótopos de hidrógeno y oxígeno en precipitación, utilizada en estudios hidrológicos.

ML (Machine Learning): Conjunto de técnicas y algoritmos que permite a un computador "aprender" de patrones a partir de datos para realizar predicciones o clasificaciones.

NetCDF (Network Common Data Form): Un archivo netCDF es un tipo de formato de almacenamiento de datos científicos multidimensionales, en el cual podemos encontrar metadatos y se puede georefenciar. A pesar de ser multidimensional su interoperabilidad es intuitiva y además los datos son almacenados en tipo binario lo que reduce el consumo de meoria.

OPeNDAP (Open-source Project for a Network Data Access Protocol): Protocolo que permite acceder remotamente a datos científicos sin necesidad de descargarlos completamente.

Pandas (Python Data Analysis Library): Librería de Python utilizada para el manejo, análisis y manipulación de datos estructurados. Esta basado en

arreglos de numpy y su interoperabilidad entre distintos tipos de archivos y de bases de datos lo ha convertido en una herramienta muy potente en la gestión de bases de datos.

PCA (Principal Component Analysis): Es una técnica estadística que reduce la dimensionalidad de los datos a través de la busqueda de correlaciones entre los datos, donde solo mantiene aquellas variables, o como se le llama en este contexto, componentes no correlacionados.

SQL (Structured Query Language): Es un lenguaje de programación utilizad para el manejo, análisis y manipulación de datos estructurados relacionales.

UTM (Universal Transverse Mercator): Es un tipo de coordenadas muy utilizado para gestionar y consultar bases de datos relacionales que estan geo espacialmente localizados.

WMO/ OMM (World Meteorological Organization/ Organización Meteorológica Mundial): Es un organismo internacional que se encarga de establecer estándares sobre la medición, monitoreo y manejo de datos meteorológicos, además que coordina actividades meteorológicas incluido congresos o transferencia de tecnología y conocimientos.

WT (Weather Types): Clasificación de patrones atmosféricos recurrentes que representan configuraciones típicas del clima, por lo que se conocen cómo regímenes climáticos.

BIBLIOGRAFÍAS

Allen, D. (2015). Think Python: How to Think Like a Computer Scientist (Version 2.4. 0). Green Tea Press.

Anaconca Inc. (2025). Working with conda. https://www.anaconda.com/docs/tools/working-with-conda/main

Anaconda Team. (2025). Conda: A Package Manager for Data Science, ML, and AI. https://www.anaconda.com/guides/conda-package-manager-for-data-sciences-ml-and-ai#elementor-toc__heading-anchor-0

Bancroft, W. D. (1918). Some Properties of Fog. The Journal of Physical Chemistry, 22(5), 309–336. https://doi.org/10.1021/j150185a001

Bertrand, B., & Harrisson, A. (2019). Building and packaging epics modules with conda. Proceedings of the 17th International Conference on Accelerators and Large Experimental Physics Control Systems (Icalepcs2019). New York, NY, USA, MOPHA014, 223–227.

Bezrukova, N., & Chernokulsky, A. (2019). Clouds and precipitation. In Russian National Report: Meteorology and Atmospheric (pp. 99–105).

Brodie, M. L. (2019). What Is Data Science? In Applied Data Science (pp. 101–130). Springer International Publishing. https://doi.org/10.1007/978-3-030-11821-1_8

Campos, D. (2025). Escalas en meteorología: desde lo micro hasta lo planetario. Https://Www.Meteored.Cl/Noticias/Ciencia/Escalas-En-Meteorologia-Desde-Lo-Micro-Hasta-Lo-Planetario.Html.

Cao, L. (2018). Data Science. ACM Computing Surveys, 50(3), 1–42. https://doi.org/10.1145/3076253

CHC. (2016a). CHIRPS: Rainfall Estimates from Rain Gauge and Satellite Observations. Climate Hazards Center Santa Barbara. https://www.chc.ucsb.edu/data/chirps/

CHC. (2016b). CHIRTS-daily. https://www.chc.ucsb.edu/data/chirtsdaily

Dask discourse group. (2021). Appropriate way of “np.apply_along_axis” in Dask - Dask Array - Dask Forum. https://dask.discourse.group/t/appropriate-way-of-np-apply-along-axis-in-dask/168

Escuela Nautica Neptuno. (2023, December 17). ¿Qué son Isobaras y para qué sirven? Https://Www.Neptuno.Es/Que-Son-Isobaras-y-Para-Que-Sirven.

Eyring, V., Bony, S., Meehl, G. A., Senior, C. A., Stevens, B., Stouffer, R. J., & Taylor, K. E. (2016a). Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization. Geoscientific Model Development, 9, 1937–1958. https://doi.org/10.5194/gmd-9-1937-2016

Eyring, V., Bony, S., Meehl, G. A., Senior, C. A., Stevens, B., Stouffer, R. J., & Taylor, K. E. (2016b). Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization. Geoscientific Model Development, 9, 1937–1958. https://doi.org/10.5194/gmd-9-1937-2016

Fernandez, I. (2001). Las Coordenadas Geográficas y la Proyección UTM. https://d1wqtxts1xzle7.cloudfront.net/57893811/cartografia-geograficas-utm-datum-libre.pdf

Fernández-Bobadilla, D. J. (1969). La Meteorología a través de la historia. Revista Tiempo y Clima, 2(9).

Fernández-Coppel, I. A. (2010). Las Coordenadas Geográficas. Recuperado de Https://Iotsensores. Com/Libros/Coordenadas. Pdf.

Fulker, D. (2016). Intro and Recent Advances: Remote Data Access via OPeNDAP Web Services. Https://Ntrs.Nasa.Gov/Citations/20160009307.

Furones, A. (2011). Sistema y marco de referencia terrestre, sistemas de coordenadas. Universidad Politécnica de Valencia, Valencia-España.

GEAA. (2022). Monitoreo de Precipitación Hídrica de la Provincia de Chimborazo. Red Meteorológica Monitoreo de Precipitación Hídrica de La

Provincia de Chimborazo. https://ceaa.espoch.edu.ec/redEstaciones/index.php

Gerardo, M. (2019, February 26). ¿QUÉ ES UNA ESTACIÓN METEOROLÓGICA? MeteoEstacion- ESTACIONES METEOROLÓGICAS & METEOROLOGÍA. https://estaciondemeteorologia.com/que-es-una-estacion-meteorologica/

Gis&Beer developers. (2018). Cómo representar y convertir archivos NetCDF - Gis&Beers. https://www.gisandbeers.com/como-representar-convertir-archivos-netcdf/

Grogg, K. (2005). Harvesting the wind: the physics of wind turbines. Physics and Astronomy Comps Papers, 7.

Hare, F. K. (1957). The Dynamic Aspects of Climatology. Geografiska Annaler, 39, 87–104. https://doi.org/10.1080/20014422.1957.11880899

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. Nature 2020 585:7825, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Helmis, C., & Nastos, P. (2012). Advances in Meteorology, Climatology and Atmospheric Physics. Springer Science & Business Media.

Hoyer, S., & Hamman, J. (2017). xarray: ND labeled arrays and datasets in Python. Journal of Open Research Software, 5(1), 10.

IAEA. (2014). IAEA/GNIP precipitation sampling guide.

IDEAM. (2025, May 20). Instituto de Hidrología, Meteorología y Estudios Ambientales. Tiempo y Clima. https://www.ideam.gov.co/web/tiempo-y-clima/radiacion-solar-ultravioleta

INDIES, W., & AMERICA, S. (1892). ISOBARS AND W. The Realm of Nature: An Outline of Physiography, 126.

John, H., Darren, D., Eric, F., Michael, D., & Matplotlib development team. (2012). Quick start guide — Matplotlib 3.10.5 documentation. https://matplotlib.org/stable/users/explain/quick_start.html

Katsanos, D., Retalis, A., & Michaelides, S. (2016). Validation of a high-resolution precipitation database (CHIRPS) over Cyprus for a 30-year period. Atmospheric Research, 169, 459–464. https://doi.org/10.1016/j.atmosres.2015.05.015

Khisanova, V. (2022). Python libraries and their usage in visualizing meteorological data.

Landsberg, H. (1953). "THE ORIGIN OF THE ATMOSPHERE." Https://Www.Jstor.Org/Stable/24944308, 189.

Lin, J. W. B. (2012). Why python is the next wave in earth sciences computing. Bulletin of the American Meteorological Society, 93, 1823–1824. https://doi.org/10.1175/BAMS-D-12-00148.1

Lin, J. W.-B. (2012). A Hands-On Introduction to Using Python in the Atmospheric and Oceanic Sciences. Lulu. com.

Lutz, M. (2013). Learning python: Powerful object-oriented programming. " O'Reilly Media, Inc."

Martin, C. (2015). The invention of atmosphere. Studies in History and Philosophy of Science Part A, 52, 44–54. https://doi.org/10.1016/j.shpsa.2015.05.007

Martorell, S. D. (2023). El Anemómetro en la Agricultura: ¿Qué es y por qué es importante? Https://Prismab.Com/Blog/El-Anemometro-En-La-Agricultura-Que-Es-y-Por-Que-Es-Importante/.

McKinney, W. (2011). pandas: a foundational Python library for data analysis and statistics. Python for High Performance and Scientific Computing, 14(9), 1–9.

Mecherikunnel, A. T., & Richmond, J. (1980). Spectral distribution of solar radiation. Https://Ntrs.Nasa.Gov/Citations/19810016493.

Meyer, P. (1993). Julian and Gregorian Calendars. DOCTOR DOBBS JOURNAL, 18, 152.

Michelangeli, P.-A., Vautard, R., & Legras, B. (1995). Weather regimes: Recurrence and quasi stationarity. Journal of Atmospheric Sciences, 52(8), 1237–1256.

Miller, B. N., Ranum, D. L., & Yasinovskyy, R. (2006). Problem solving with algorithms and data structures using python. Franklin, Beedle & Associates.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). Foundations of machine learning. MIT press.

Moller, F. (1956). The pattern of radiative heating and cooling in the troposphere and lower stratosphere. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 236, 148–156. https://doi.org/10.1098/rspa.1956.0121

Muñoz, Á. G., Goddard, L., Robertson, A. W., Kushnir, Y., & Baethgen, W. (2015). Cross–time scale interactions and rainfall extreme events in southeastern South America for the austral summer. Part I: Potential predictors. Journal of Climate, 28(19), 7894–7913.

Nadal, I. S., & Muñuzuri, V. P. (2006). Fundamentos de meteorología. Servizo de Publicacións e Intercambio Científico da USC. https://books.google.com.ec/books?id=HE3xtRmNg4kC

Nouvelot, J., & Pourrut, P. (1984). El Nino : phénomène océanique et atmosphérique, importance en 1982-1983 et impact sur le littoral équatorien- fdi:21071- Horizon. https://www.documentation.ird.fr/hor/fdi:21071

Numpy Team. (2025, June 7). NumPy. NumPy 2.3.0 Released. https://numpy.org/

Odegua, R., & Ikpotokin, F. (2020). DataSist: A Python-based library for easy data analysis, visualization and modeling. http://arxiv.org/abs/1911.03655

Oktavia Suhyani, N. (2025). Machine learning for Earth sciences: using Python to solve geological problems. International Geology Review, 67, 1635–1636. https://doi.org/10.1080/00206814.2025.2457562

OMM. (2010). Guía de Instrumentos y Métodos de Observación Meteorológicos.

O'Neill, B. C., Tebaldi, C., van Vuuren, D. P., Eyring, V., Friedlingstein, P., Hurtt, G., Knutti, R., Kriegler, E., Lamarque, J.-F., Lowe, J., Meehl, G. A., Moss, R., Riahi, K., & Sanderson, B. M. (2016). The Scenario Model Intercomparison Project (ScenarioMIP) for CMIP6. Geoscientific Model Development, 9, 3461–3482. https://doi.org/10.5194/gmd-9-3461-2016

Orlanski, I. (1975). A rational subdivision of scales for atmospheric processes. Bulletin of the American Meteorological Society, 527–530.

Pabón Caicedo, J. D., & Montealegre Bocanegra, J. E. (2017). Los fenómenos de El Niño y de La Niña, su efecto climático e impactos socioeconómicos.

Pannett, R. A. (1990). Guidance for the education and training of instrument specialists. Secretariat of the World Meteorological Organization.

Pazmiño Solís, C. R. (2021). Desarrollo de un sistema web interactivo para el tratamiento de datos de la Estación Meteorológica de la ESPOCH.

PCMDI. (2019). CMIP6 - Coupled Model Intercomparison Project Phase 6. PCMDI: Earth System Model Evaluation Project Official Webpage. https://pcmdi.llnl.gov/CMIP6/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., & Dubourg, V. (2011). Scikit-learn: machine learning in python, journal of machine learning research. vol 12 (Oct).

Pérez, F., & Granger, B. E. (2007). IPython: A system for interactive scientific computing. Computing in Science and Engineering, 9(3), 21–29. https://doi.org/10.1109/MCSE.2007.53

Perkel, J. M. (2015). Programming: Pick up Python. In Nature (Vol. 518, pp. 125–126). Nature Publishing Group. https://doi.org/10.1038/518125a

Pierrehumbert, R. T., Brogniez, H., & Roca, R. (2008). Chapter 6 On the Relative Humidity of the Atmosphere. In The Global Circulation of the Atmosphere (pp. 143–185). Princeton University Press. https://doi.org/10.1515/9780691236919-008

Pruppacher, H. R., Klett, J. D., & Wang, P. K. (1998). Microphysics of Clouds and Precipitation. Aerosol Science and Technology, 28(4), 381–382. https://doi.org/10.1080/02786829808965531

Psallidas, F., Zhu, Y., Karlas, B., Interlandi, M., Floratou, A., Karanasos, K., Wu, W., Zhang, C., Krishnan, S., & Curino, C. (2019). Data science through the looking glass and what we found there. ArXiv Preprint ArXiv:1912.09536.

Python Software Foundation. (2025). Python-Get Started. https://www.python.org/

Rew, R., & Davis, G. (1990). NetCDF: an interface for scientific data access. IEEE Computer Graphics and Applications, 10(4), 76–82.

Rew, R., Davis, G., Emmerson, S., Davies, H., & Hartnett, E. (1993). NetCDF user's Guide. Unidata Program Center, Boulder, Colorado.

RobustScaler — scikit-learn 1.7.2 documentation. (n.d.). Retrieved September 10, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

Rodriguez, M., & León, C. (2012). Fundamentos de Climatología (2nd ed.). Universidad de La Rioja.

Rogel-Salazar, J. (2018). Data science and analytics with python. Data Science and Analytics with Python, 1–376. https://doi.org/10.1201/9781315151670/DATA-SCIENCE-ANALYTICS-PYTHON-JESUS-ROGEL-SALAZAR/RIGHTS-AND-PERMISSIONS

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., & Church, K. (2020a). Introduction to Anaconda and Python: Installation and setup. The

Quantitative Methods for Psychology, 16(5), S3–S11. https://doi.org/10.20982/tqmp.16.5.s003

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., & Church, K. (2020b). Introduction to Anaconda and Python: Installation and setup. The Quantitative Methods for Psychology, 16(5), S3–S11. https://doi.org/10.20982/tqmp.16.5.s003

Sachweh, M., & Koepke, P. (1997). Fog dynamics in an urbanized area. Theoretical and Applied Climatology, 58, 87–93. https://doi.org/10.1007/BF00867435

scikit-learn developers. (2025). 2.3. Clustering — scikit-learn 1.7.2 documentation. https://scikit-learn.org/stable/modules/clustering.html#k-means

Scikit-learn, L. W. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow. Oâ€TM Reilly Media.

Sensor MKT. (2020, September 15). Estaciones Meteorológicas. ¿Qué son y cómo funcionan? SensorGo Pagina Oficial.

Spiridonov, V., & Ćurić, M. (2020). Fundamentals of Meteorology. Springer International Publishing. https://books.google.com.ec/books?id=kGcHEAAAQBAJ

Spiridonov, V., & Ćurić, M. (2021). Fundamentals of Meteorology. Springer International Publishing. https://doi.org/10.1007/978-3-030-52655-9

Spyder-IDE. (2024). El entorno de desarrollo para Python que los científicos y analistas de datos merecen. https://www.spyder-ide.org/

Stancin, I., & Jovic, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2019 - Proceedings, 977–982. https://doi.org/10.23919/MIPRO.2019.8757088

Steyn, D. G., Oke, T. R., Hay, J. E., & Knox, J. L. (1981). On scales in meteorology and climatology. Climatological Bulletin, 39, 1–8.

Valenzuela, M. del M. (2010). El nacimiento de la astronomía antigua. Estabilizaciones y desestabilizaciones culturales. Gazeta de Antropología, 26(2). https://www.ugr.es/~pwlac/G26_25MariaDelMar_Valenzuela_Vila.html

Valverde González, V. L., García Mora, F. A., & Hernández Dávila, E. S. (2023). Python Aplicado al mantenimiento industrial. Centro de Investigación y Desarrollo Ecuador.

Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. Computing in Science and Engineering, 13(2), 22–30. https://doi.org/10.1109/MCSE.2011.37

Van Rossum, G. (2001). PEP 8 – Style Guide for Python Code | peps.python.org. https://peps.python.org/pep-0008/

VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data. " O'Reilly Media, Inc."

Verdin, A., Funk, C., Peterson, P., Landsfeld, M., Tuholske, C., & Grace, K. (2020). Development and validation of the CHIRTS-daily quasi-global high-resolution daily temperature data set. Scientific Data, 7. https://doi.org/10.1038/s41597-020-00643-7

Visconti, G. (2016). Fundamentals of Physics and Chemistry of the Atmosphere. Springer International Publishing. https://doi.org/10.1007/978-3-319-29449-0

Vos, R., Velasco, M., & de Labastida, E. (1999). Economic and social effects of" El Nino" in Ecuador, 1997-8.

Williams, P. (2021). Chapter 10 - Radiative transfer. In P. D. Williams & M. H. P. Ambaum (Eds.), Thermal Physics of the Atmosphere (Second Edition) (pp. 205–222). Elsevier. https://doi.org/https://doi.org/10.1016/B978-0-12-824498-2.00017-3

Xarray core developers. (2025). Xarray: N-D labeled arrays and datasets in Python. https://xarray.dev/

Zen De Figueiredo Neves, G., Gallardo, N. P., Arthur, F., Vecchia, S., Steinke, V. A., & Aparecido Da Silva, C. (2017). A Short Critical History on the Development of Meteorology and Climatology. Climate 2017, Vol. 5, Page 23, 5(1), 23. https://doi.org/10.3390/cli5010023

Zhang, F., Li, J., Zhang, H., Ma, L., & Zhou, X. (2013). On the relationship between direct and diffuse radiation. Journal of Quantitative Spectroscopy and Radiative Transfer, 115, 60–65. https://doi.org/10.1016/j.jqsrt.2012.09.009

Introducción a la meteorología con ciencia de datos en Python, se publicó en el mes de mayo de 2026.

ISBN: 978-9907-802-08-5

Grupo Editorial BLR
Ecuador
Cel: +593 98 320 4362
https://grupoblr.com/
publicaciones@grupoblr.com

BIOGRAFÍA DE LOS AUTORES

Amalia Isabel Escudero Villa:

Doctora en Estadística Matemática y Aplicada. Máster Universitario en Estadística Matemática y Aplicada. Magister en Matemática Básica. Ingeniera en Estadística Informática. Docente de la Escuela Superior Politécnica de Chimborazo. Investigadora Junior de la ESPOCH, Investigadora en el Grupo de Energías Alternativas y Ambiente GEAA. Autora de varios artículos científicos. Coordinadora de la carrera de Matemática.

Cristina Estefanía Ramos Araujo:

Master of Science in Applied Mathematics. Master Universitario en Análisis y Visualización de Datos Masivos / Visual Analytics and Big Data. Ingeniera en Estadística Informática. Docente de la Escuela Superior Politécnica de Chimborazo. Investigadora en el Grupo de Energías Alternativas y Ambiente GEAA.

Iván Fabricio Chávez Velasco:

Magister en Matemática aplicada con mención en Matemática Computacional. Graduado de Físico. Técnico de investigación de la Escuela Superior Politécnica de Chimborazo. Forma parte del grupo de investigación de Energías Alternativas y Ambiente GEAA.

INTRODUCCIÓN A LA METEOROLOGÍA CON CIENCIA DE DATOS EN PYTHON

Estimado lector. Introducción a la meteorología con ciencia de datos en Python es un libro orientado a investigadores interesados en el manejo de datos masivos para aplicarlo en el área de la climatología mediante herramientas computacionales en Python. A lo largo de sus capítulos se abordan los fundamentos atmosféricos, las principales variables meteorológicas y los sistemas de observación, para luego introducir técnicas avanzadas de manejo y procesamiento de grandes volúmenes de información. Se presenta un enfoque aplicado al uso de estructuras de datos multidimensionales, metodologías de aprendizaje automático (machine learning) y visualización científica, con aplicaciones específicas en la detección de patrones climáticos regionales. Esta obra se concibe como un manual de referencia y soporte bibliográfico para estudiantes, investigadores y profesionales que busquen fortalecer sus capacidades en el análisis computacional de datos meteorológicos y climáticos.

Agradecemos a todos los lectores que se acercan a esta obra con ánimo de aprender, aplicar y transformar.

GRUPO BLR

Grupo Editorial BLR
Ecuador
Cel: +593 98 320 4362
https://grupoblr.com/
publicaciones@grupoblr.com

ISBN: 978-9942-51-520-9