**MANUAL DE USUARIO**

**Autenticador Drive**

**# Vincular colab con google drive**

1. **from google.colab import drive**

* **Importando el módulo drive:** Esta línea trae al entorno de Colab un módulo específico llamado drive. Este módulo es como una herramienta que nos permite interactuar con Google Drive directamente desde nuestro notebook de Colab.

1. **drive.mount('/content/drive/')**

* **Montando Google Drive:** Esta línea hace que nuestro Google Drive se "conecte" a una carpeta específica dentro de Colab, que es /content/drive/. Al montar el drive, estás haciendo que los archivos de tu Google Drive están disponibles para que los uses en Colab como si estuvieran guardados en tu computadora.

**En resumen:**

Estas dos líneas te permiten conectar tu Google Drive a Colab para que puedas trabajar con tus archivos de una manera más fácil y eficiente.

**Instalar librerías a usar**

**# @title instalar librerias**

2. **!pip install sweetviz**

* !: Este signo de exclamación al principio de la línea indica a la consola (como Google Colab o Jupyter Notebook) que la siguiente línea es un comando del sistema operativo, y no una línea de código Python.
* pip: Pip es un gestor de paquetes de Python. Se utiliza para instalar, actualizar y eliminar paquetes (bibliotecas) de Python.
* install: Esta palabra clave le indica a pip que queremos instalar un nuevo paquete.
* sweetviz: Este es el nombre del paquete que queremos instalar. Sweetviz es una biblioteca de Python especializada en análisis exploratorio de datos. Genera informes interactivos que proporcionan una visión rápida y completa de un conjunto de datos.

**3. !pip install category\_encoders**

* !: Al igual que en la línea anterior, indica que es un comando del sistema operativo.
* pip install: La misma función que en la línea anterior: instalar un paquete.
* category\_encoders: Este es el nombre del paquete a instalar. Category Encoders es una biblioteca que proporciona técnicas para codificar variables categóricas. Las variables categóricas son aquellas que representan categorías o grupos (por ejemplo, género, color, país). Estas técnicas son necesarias porque muchos algoritmos de machine learning requieren que los datos sean numéricos.

En pocas palabras, estas dos líneas hacen lo siguiente:

* Preparan tu entorno de Python: Aseguran que tengas las herramientas necesarias para trabajar con tus datos.
* Facilitan el análisis de datos: Sweetviz te ayudará a entender rápidamente tus datos, mientras que Category Encoders te permitirá preparar tus datos para modelos de machine learning.

**# Importando librerías y módulos a usar:**

**1. import os**

* Importa el módulo os: Este módulo proporciona funciones para interactuar con el sistema operativo, como trabajar con directorios, archivos y rutas.

**2. import re**

* Importa el módulo re: Este módulo permite trabajar con expresiones regulares, que son patrones de búsqueda para encontrar y manipular texto.

**3. import math**

* Importa el módulo math: Este módulo proporciona funciones matemáticas comunes, como cálculos trigonométricos, logarítmicos, exponenciales, etc.

**4. import numpy as np**

* Importa el módulo numpy y lo asigna al alias np: Este módulo es fundamental para trabajar con arrays multidimensionales (matrices) y operaciones numéricas eficientes.

**5. import pandas as pd**

* Importa el módulo pandas y lo asigna al alias pd: Este módulo proporciona estructuras de datos de alto rendimiento y herramientas de análisis de datos, como DataFrames.

**6. import seaborn as sns**

* Importa el módulo seaborn: Este módulo es una biblioteca de visualización de datos basada en Matplotlib, que proporciona una interfaz de alto nivel para crear gráficos estadísticos atractivos.

**7. import matplotlib.patches as mpatches**

* Importa el submódulo patches del módulo matplotlib.pyplot: Este submódulo proporciona herramientas para crear formas geométricas como rectángulos, círculos, elipses, etc., que se pueden usar para personalizar gráficos.

**8. import matplotlib.pyplot as plt**

* Importa el submódulo pyplot del módulo matplotlib y lo asigna al alias plt: Este submódulo proporciona una interfaz de estilo MATLAB para crear gráficos 2D.

**9. import sweetviz as sw**

* Importa el módulo sweetviz: Este módulo es una herramienta de análisis exploratorio de datos que genera informes interactivos sobre los datos, incluyendo estadísticas descriptivas, visualizaciones y detección de anomalías.

**10. import category\_encoders as ce**

* Importa el módulo category\_encoders: Este módulo proporciona técnicas de codificación de variables categóricas, como One-Hot Encoding, Target Encoding, y otras, que son útiles para preparar datos para modelos de machine learning.

**11. from google.colab import files**

* Módulo específico de Google Colab que permite manejar archivos, como subir y descargar archivos desde el entorno de Colab.

**# Importando herramientas de aprendizaje automático de scikit-learn**

Estas líneas de código importan varias herramientas del módulo sklearn que son fundamentales para el aprendizaje automático. Vamos a desglosar cada importación:

1. **from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV:**
   * **train\_test\_split**: Esta función se utiliza para dividir un conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba. Esto es esencial para entrenar un modelo y luego evaluarlo en datos no vistos.
   * **cross\_val\_score**: Esta función se utiliza para realizar validación cruzada, una técnica que ayuda a evaluar el rendimiento de un modelo en diferentes subconjuntos de datos.
   * **GridSearchCV**: Esta función se utiliza para realizar búsqueda en cuadrícula, una técnica para encontrar los mejores hiperparámetros para un modelo.
2. **from sklearn.preprocessing import OneHotEncoder, StandardScaler:**
   * **OneHotEncoder**: Este codificador se utiliza para transformar variables categóricas en variables numéricas. Por ejemplo, si tienes una variable "color" con valores "rojo", "verde" y "azul", el codificador creará nuevas columnas binarias para cada valor.
   * **StandardScaler**: Este escalador se utiliza para estandarizar las características numéricas de un conjunto de datos. Esto significa que los valores se transforman para tener una media de 0 y una desviación estándar de 1.
3. **from sklearn.impute import SimpleImputer**:
   * **SimpleImputer**: Este imputador se utiliza para manejar valores faltantes en los datos. Puede reemplazar los valores faltantes con la media, la mediana, la moda o un valor constante.
4. **from sklearn.pipeline import Pipeline**:
   * **Pipeline**: Esta clase se utiliza para crear un pipeline de procesamiento de datos y entrenamiento de modelos. Un pipeline combina múltiples pasos de procesamiento, como imputación, codificación, escalado y entrenamiento del modelo, en una sola secuencia.
5. **from sklearn.feature\_selection import SelectKBest**:
   * **SelectKBest**: Esta clase se utiliza para seleccionar las mejores características de un conjunto de datos. Puede seleccionar las características más importantes basadas en diferentes criterios, como la correlación con la variable objetivo.
6. **from sklearn.compose import ColumnTransformer**:
   * **ColumnTransformer**: Esta clase se utiliza para aplicar diferentes transformaciones a diferentes columnas de un conjunto de datos. Por ejemplo, puedes aplicar un codificador a las columnas categóricas y un escalador a las columnas numéricas.
7. **from sklearn.metrics import accuracy\_score, f1\_score, classification\_report, roc\_auc\_score**:
   * **accuracy\_score**: Esta métrica calcula la precisión del modelo, es decir, la proporción de predicciones correctas.
   * **f1\_score**: Esta métrica combina la precisión y la sensibilidad del modelo.
   * **classification\_report**: Esta función genera un informe detallado sobre el rendimiento del modelo, incluyendo precisión, sensibilidad, F1-score, etc.
   * **roc\_auc\_score**: Esta métrica se utiliza para evaluar el rendimiento de modelos de clasificación binaria.
8. **from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report**:
   * Estas importaciones son similares a las anteriores y se utilizan para evaluar el rendimiento del modelo.

**En resumen:**

Estas importaciones proporcionan las herramientas necesarias para preparar los datos, entrenar modelos de aprendizaje automático, evaluar su rendimiento y hacer predicciones.

9. from sklearn.decomposition import PCA

* **Importando PCA:** Esta línea importa la clase PCA del módulo sklearn.decomposition.
* **¿Qué es PCA?** PCA, o Análisis de Componentes Principales, es una técnica de reducción de dimensionalidad. Esto significa que toma un conjunto de datos con muchas características (columnas) y lo transforma en un nuevo conjunto de datos con menos características, pero que capturan la mayor parte de la variabilidad original. Es útil para visualizar datos de alta dimensión y mejorar el rendimiento de algunos modelos de aprendizaje automático.

10. import xgboost as xgb

* **Importando XGBoost:** Esta línea importa la biblioteca XGBoost y le asigna el alias xgb para facilitar su uso.
* **¿Qué es XGBoost?** XGBoost es una librería de optimización de gradiente extremo que se utiliza para construir modelos de machine learning. Es conocido por su alta eficiencia y precisión, y se utiliza tanto para tareas de regresión como de clasificación. XGBoost crea un modelo de aprendizaje de conjunto que combina múltiples árboles de decisión para hacer predicciones.

**En resumen:**

* **PCA** es una herramienta para reducir la dimensionalidad de los datos.
* **XGBoost** es una librería poderosa para construir modelos de machine learning.

**# Balanceo**

**from imblearn.under\_sampling import CondensedNearestNeighbour, EditedNearestNeighbours, InstanceHardnessThreshold, RepeatedEditedNearestNeighbours, OneSidedSelection**

* **Importaciones de Técnicas de Submuestreo (Under-sampling)**:
  + **CondensedNearestNeighbour**: Esta técnica reduce el conjunto de datos eliminando ejemplos de la clase mayoritaria que no son necesarios para mantener la clasificación de la clase minoritaria. Se basa en un algoritmo de vecinos más cercanos.
  + **EditedNearestNeighbours**: Esta técnica edita el conjunto de datos eliminando ejemplos de la clase mayoritaria que son incorrectamente clasificados por sus vecinos más cercanos. Esto ayuda a limpiar el conjunto de datos.
  + **InstanceHardnessThreshold**: Selecciona y elimina ejemplos de la clase mayoritaria que son fáciles de clasificar, manteniendo aquellos que son más difíciles. Esto se hace para equilibrar el conjunto de datos eliminando ejemplos que no aportan información valiosa.
  + **RepeatedEditedNearestNeighbours**: Es una versión mejorada de EditedNearestNeighbours que realiza múltiples pasadas a través del conjunto de datos para asegurar una mejor limpieza y reducción.
  + **OneSidedSelection**: Esta técnica selecciona ejemplos de la clase mayoritaria y elimina ejemplos de la clase minoritaria que están demasiado cerca de la clase mayoritaria, ayudando a balancear las clases.

**from imblearn.under\_sampling import TomekLinks**

* **TomekLinks**: Esta técnica identifica pares de ejemplos de diferentes clases que son vecinos cercanos y elimina aquellos que están cerca. Esto puede ayudar a mejorar la separación entre las clases y a reducir el ruido en el conjunto de datos.

**from collections import Counter**

* **Counter**: Importa la clase Counter de la biblioteca collections, que se utiliza para contar la frecuencia de los elementos en un iterable, como una lista o un array. Esto es útil para contar la cantidad de instancias de cada clase en un conjunto de datos.

**from imblearn.over\_sampling import SMOTE**

* **SMOTE (Synthetic Minority Over-sampling Technique)**: Esta técnica genera ejemplos sintéticos para la clase minoritaria interpolando entre ejemplos existentes. Es útil para aumentar el número de instancias en la clase minoritaria, ayudando a equilibrar el conjunto de datos.

**from imblearn.pipeline import Pipeline**

* **Pipeline**: Importa la clase Pipeline, que permite encadenar múltiples transformaciones y estimadores en un solo objeto. Esto facilita la creación de flujos de trabajo complejos, donde puedes aplicar varios pasos a los datos de manera secuencial.

**from sklearn.cluster import MiniBatchKMeans**

* **MiniBatchKMeans**: Importa el algoritmo de clustering MiniBatchKMeans, que es una versión eficiente de K-Means que utiliza pequeños lotes de datos para mejorar el rendimiento y la velocidad, especialmente útil para conjuntos de datos grandes.

**from imblearn.under\_sampling import ClusterCentroids**

* **ClusterCentroids**: Esta técnica utiliza el clustering (como K-Means) para agrupar ejemplos de la clase mayoritaria y luego utiliza los centroides de esos grupos como nuevos ejemplos representativos. Esto ayuda a reducir el tamaño del conjunto de datos mientras se conserva la información.

**from imblearn.under\_sampling import NearMiss**

* **NearMiss**: Esta técnica selecciona ejemplos de la clase mayoritaria para igualar el número de ejemplos de la clase minoritaria, basándose en la proximidad a los ejemplos de la clase minoritaria. Hay diferentes variantes de NearMiss, que difieren en cómo se seleccionan los ejemplos de la clase mayoritaria.

### Resumen

Estas importaciones te permiten trabajar con diferentes técnicas para manejar problemas de clases desbalanceadas en conjuntos de datos. Tanto el submuestreo como el sobremuestreo son estrategias comunes en la preparación de datos para mejorar el rendimiento de modelos de machine learning en situaciones donde hay un desbalance significativo entre las clases.

Algoritmos

**Importando algoritmos de clasificación**

Estas líneas de código importan varios algoritmos de clasificación de la biblioteca scikit-learn:

from sklearn.neighbors import KNeighborsClassifier

* Importa el clasificador **K-Nearest Neighbors** (Vecinos Más Cercanos). Este clasificador se basa en la distancia entre puntos de datos, clasificando un punto de acuerdo con la mayoría de las clases de sus vecinos cercanos.

from sklearn.svm import SVC

* Importa el clasificador de **Máquinas de Vectores de Soporte** (Support Vector Classifier). Este método intenta encontrar el mejor "margen" o línea divisoria entre distintas clases para clasificar los datos.

from sklearn.tree import DecisionTreeClassifier

* Importa el **clasificador de Árbol de Decisión**, un modelo que crea un árbol para dividir los datos en diferentes grupos según características y valores específicos. Muy útil en problemas de clasificación.

from sklearn.tree import DecisionTreeRegressor

* Importa el **regresor de Árbol de Decisión**, similar al clasificador pero usado en problemas de regresión, donde el objetivo es predecir un valor continuo.

from sklearn.tree import export\_text

* Importa la función **export\_text**, que permite visualizar la estructura de un árbol de decisión en formato de texto. Es útil para interpretar y comprender cómo se están tomando las decisiones en el modelo.

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

* Importa varios modelos de ensamble (combinación de múltiples modelos):
  + **RandomForestClassifier**: Un clasificador basado en muchos árboles de decisión que trabajan juntos para mejorar la precisión y reducir el sobreajuste.
  + **AdaBoostClassifier**: Un clasificador que crea múltiples modelos secuenciales, donde cada uno intenta corregir los errores del anterior.
  + **GradientBoostingClassifier**: Un clasificador que también usa múltiples modelos secuenciales, pero mejora la precisión usando gradientes para ajustar cada modelo en base a los errores acumulados.

from sklearn.ensemble import RandomForestRegressor

* Importa el **regresor de Bosque Aleatorio**, similar al clasificador pero enfocado en problemas de regresión. Usa múltiples árboles de decisión para mejorar la precisión y reducir el sobreajuste en regresión.

from sklearn.ensemble import GradientBoostingRegressor

* Importa el **regresor de Gradient Boosting**, una técnica de ensamble para regresión que ajusta múltiples modelos de manera secuencial para minimizar el error.

from sklearn.metrics import make\_scorer

* Importa **make\_scorer**, una función para crear puntuadores personalizados. Esto es útil si deseas evaluar el rendimiento de un modelo usando una métrica que no está incluida en las métricas predefinidas de scikit-learn.

from sklearn.linear\_model import LogisticRegression

* Importa el modelo de **Regresión Logística**, que se usa para problemas de clasificación y estima la probabilidad de una clase usando una función logística.

from sklearn.linear\_model import LinearRegression

* Importa el modelo de **Regresión Lineal**, que se usa en problemas de regresión para modelar la relación lineal entre variables dependientes e independientes.

from sklearn.preprocessing import PolynomialFeatures

* Importa **PolynomialFeatures**, una herramienta para crear nuevas características a partir de las existentes, elevándolas a diferentes potencias. Esto se usa comúnmente para crear modelos de regresión polinómica que ajusten relaciones no lineales entre las variables.

Estos algoritmos se utilizan para construir modelos de clasificación, que pueden predecir la clase de una instancia dada. La elección del algoritmo adecuado depende de las características del conjunto de datos y del problema específico.

**Cargamos las funciones relacionadas**

def analizar\_columnas(df):

* + Define una función llamada analizar\_columnas, que toma un parámetro df, que es un **DataFrame** de pandas. La función analizará cada columna del DataFrame y devolverá un resumen estadístico.

"""

Analiza las columnas de un DataFrame y devuelve información estadística.

Parámetros:

df (DataFrame): El DataFrame a analizar.

Retorno:

Un DataFrame con la información estadística de cada columna.

Ejemplo:

df\_analizado = analizar\_columnas(df)

print(df\_analizado)

"""

* + Esta es una **docstring** o comentario de documentación. Explica qué hace la función, sus parámetros y lo que retorna. También da un ejemplo de uso.

info = []

* + Crea una lista vacía llamada info para almacenar los resultados del análisis de cada columna del DataFrame.

for columna in df.columns:

* + Inicia un bucle for que recorrerá todas las columnas del DataFrame (df.columns).

unicos = df[columna].nunique()

* + Calcula el número de valores únicos en la columna actual usando nunique() y lo guarda en la variable unicos.

nulos = df[columna].isnull().sum()

* + Cuenta cuántos valores nulos (NaN) hay en la columna actual usando isnull().sum() y guarda el resultado en nulos.

total = len(df)

* + Calcula el número total de filas del DataFrame (len(df)) y lo guarda en total.

porcentaje\_nulos = (nulos / total) \* 100

* + Calcula el **porcentaje de valores nulos** en la columna dividiendo el número de nulos (nulos) entre el total de registros (total) y multiplicando por 100. Guarda este porcentaje en porcentaje\_nulos.

info.append({

'Columna': columna,

'Cantidad de registros': total,

'Únicos': unicos,

'Cantidad de nulos': nulos,

'Porcentaje de nulos': porcentaje\_nulos,

'Registros relacionados con nulos': total - nulos

})

* + Agrega un diccionario a la lista info con las estadísticas de la columna actual. Este diccionario contiene:
    - Nombre de la columna.
    - Número total de registros.
    - Cantidad de valores únicos.
    - Cantidad de valores nulos.
    - Porcentaje de valores nulos.
    - Cantidad de registros que no son nulos.

return pd.DataFrame(info)

- Al finalizar el bucle, convierte la lista `info` en un \*\*DataFrame de pandas\*\* y lo devuelve. Este DataFrame contiene el resumen estadístico de cada columna.

**Ruta de datos**

1.path="/content/drive/MyDrive/NASA COURSE/Proyecto predicción cuencas hídricas/1. PROYECTO FINAL INTELIGENCIA ARTIFICIAL/1. BASES DE DATOS SELECCIONADAS/"

Esta línea de código en Python define una variable llamada path y le asigna una cadena de texto que representa una ruta a un directorio en Google Drive.

**Desglose de la ruta:**

* **/content/drive/MyDrive/**: Esta parte inicial indica la raíz de tu Google Drive, donde se almacenan todos tus archivos y carpetas.
* **NASA COURSE/Proyecto predicción cuencas hídricas/**: Esta parte especifica la ruta a una carpeta dentro de tu Google Drive llamada "NASA COURSE" y luego a otra subcarpeta llamada "Proyecto predicción cuencas hídricas".
* **1. PROYECTO FINAL INTELIGENCIA ARTIFICIAL/**: Esta parte indica otra subcarpeta dentro de la carpeta anterior, llamada "1. PROYECTO FINAL INTELIGENCIA ARTIFICIAL".
* **1. BASES DE DATOS SELECCIONADAS/**: Finalmente, esta última parte señala la carpeta donde se encuentran las bases de datos seleccionadas para el proyecto.

**En resumen:**

La variable path apunta a la ubicación exacta de una carpeta en tu Google Drive donde tienes almacenadas las bases de datos que serán utilizadas en un proyecto de predicción de cuencas hídricas utilizando inteligencia artificial.

**Base de datos**

**Explicación de las líneas de código:**

Estas líneas de código en Python, utilizando la biblioteca Pandas, se encargan de leer diferentes archivos de datos climáticos y almacenarlos en DataFrames de Pandas.

1. **precipitacion = pd.read\_csv(path+"Precipitacion Total diaria Manizales 2.csv", encoding='latin1', delimiter=";")**:
   * Lee un archivo CSV llamado "Precipitacion Total diaria Manizales 2.csv" desde la ruta especificada en la variable path.
   * Especifica la codificación latin1 para manejar caracteres especiales.
   * Utiliza el carácter ; como delimitador de campos.
   * Almacena los datos en un DataFrame llamado precipitacion.
2. **temp = pd.read\_csv(path+"Temperatura Maxima Diaria Manizales 2.csv", encoding='latin1', delimiter=";")**:
   * Lee un archivo CSV llamado "Temperatura Maxima Diaria Manizales 2.csv" desde la ruta especificada.
   * Utiliza la codificación latin1 y el delimitador ;.
   * Almacena los datos en un DataFrame llamado temp.
3. **hum = pd.read\_csv(path+"Humedad Relativa Maxima diaria Manizales 2.csv", encoding='latin1', delimiter=";")**:
   * Lee un archivo CSV llamado "Humedad Relativa Maxima diaria Manizales 2.csv" desde la ruta especificada.
   * Utiliza la codificación latin1 y el delimitador ;.
   * Almacena los datos en un DataFrame llamado hum.
4. **vel = pd.read\_csv(path+"Velocidad del Viento Media diaria Manizales 2.csv", encoding='latin1', delimiter=";")**:
   * Lee un archivo CSV llamado "Velocidad del Viento Media diaria Manizales 2.csv" desde la ruta especificada.
   * Utiliza la codificación latin1 y el delimitador ;.
   * Almacena los datos en un DataFrame llamado vel.
5. **nivel = pd.read\_csv(path+"Nivel Medio Diario Manizales 2.csv", encoding='latin1', delimiter=";")**:
   * Lee un archivo CSV llamado "Nivel Medio Diario Manizales 2.csv" desde la ruta especificada.
   * Utiliza la codificación latin1 y el delimitador ;.
   * Almacena los datos en un DataFrame llamado nivel.

**En resumen:**

Estas líneas de código cargan diferentes conjuntos de datos climáticos y de eventos relacionados con el nivel del río, almacenándolos en DataFrames de Pandas para su posterior análisis y procesamiento.

**Limpieza de datos**

**Explicación de las líneas de código**

Estas líneas de código en Python, utilizando la biblioteca Pandas, se encargan de **renombrar** una columna específica en cada uno de los DataFrames que previamente habíamos cargado.

**¿Por qué renombrar columnas?**

* **Claridad:** Al renombrar las columnas con nombres más descriptivos, facilitamos la comprensión del DataFrame y evitamos confusiones.
* **Consistencia:** Asegurar que los nombres de las columnas sean coherentes y estén en un formato estándar es importante para realizar análisis y visualizaciones.
* **Evitar conflictos:** Si diferentes DataFrames tienen columnas con el mismo nombre, renombrarlas evita conflictos al combinar o unir los DataFrames.

**Desglose de cada línea:**

1. **nivel = nivel.rename(columns={'Valor': 'Nivel'})**:
   * Toma el DataFrame nivel y crea una copia con las columnas renombradas.
   * La columna original llamada Valor es renombrada a Nivel.
   * El DataFrame modificado se asigna nuevamente a la variable nivel.
2. **precipitacion = precipitacion.rename(columns={'Valor': 'Precipitacion'})**:
   * Realiza lo mismo que la línea anterior, pero para el DataFrame precipitacion.
   * Renombra la columna Valor a Precipitacion.
3. **temp = temp.rename(columns={'Valor': 'Temperatura'})**:
   * Renombra la columna Valor a Temperatura en el DataFrame temp.
4. **hum = hum.rename(columns={'Valor': 'Humedad'})**:
   * Renombra la columna Valor a Humedad en el DataFrame hum.
5. **vel = vel.rename(columns={'Valor': 'Velocidad'})**:
   * Renombra la columna Valor a Velocidad en el DataFrame vel.

**En resumen:**

Estas líneas de código están estandarizando los nombres de las columnas de los diferentes DataFrames, reemplazando el nombre genérico "Valor" por nombres más descriptivos que reflejan el tipo de dato que contienen (Nivel, Precipitacion, Temperatura, Humedad, Velocidad). Esto hace que el código sea más legible y facilita el trabajo posterior con estos DataFrames.

#Muestra los datos únicos dentro de una variable

1.nivel.Latitud.unique()

**nivel.Latitud.unique()**

Esta línea de código en Python, utilizando la biblioteca Pandas, realiza una operación específica sobre un DataFrame de datos llamado nivel. Vamos a desglosarla paso a paso:

**1. nivel:**

* **DataFrame:** Representa una estructura de datos tabular, similar a una hoja de cálculo, que contiene los datos relacionados con el nivel de algún elemento (posiblemente el nivel de un río, según el contexto).

**2. nivel.Latitud:**

* **Acceso a una columna:** Latitud es el nombre de una columna dentro del DataFrame nivel. Al escribir nivel.Latitud, estamos accediendo específicamente a los datos contenidos en esa columna. Esta columna probablemente contiene los valores de latitud geográfica para cada registro en el DataFrame.

**3. .unique():**

* **Método para encontrar valores únicos:** El método unique() es una función de Pandas que se aplica a una serie (una columna de un DataFrame) y devuelve un array con los valores únicos presentes en esa serie. En otras palabras, esta función elimina los valores duplicados y te muestra solo una vez cada valor distinto que aparece en la columna.

**En resumen:**

La línea de código nivel.Latitud.unique() está haciendo lo siguiente:

* **Tomando el DataFrame nivel:** Seleccionando el conjunto de datos completo.
* **Accediendo a la columna Latitud:** Enfocándose en los valores de latitud geográfica.
* **Encontrando los valores únicos:** Identificando todas las latitudes distintas que aparecen en los datos.

### # Después de analizar cuáles columnas no nos interesan, las eliminamos

nivel.drop(columns=['Latitud', 'Longitud', 'Altitud', 'Categoria', 'Entidad', 'AreaOperativa', 'Departamento', 'Municipio',

'FechaInstalacion', 'FechaSuspension', 'IdParametro', 'Etiqueta', 'DescripcionSerie', 'Frecuencia',

'Grado', 'Calificador', 'NivelAprobacion'], inplace=True)

1. **nivel.drop(...)**:
   * Este es el método drop de un DataFrame de pandas. Se utiliza para eliminar filas o columnas del DataFrame. En este caso, se está utilizando para eliminar columnas.
2. **columns=[...]**:
   * Se especifica el parámetro columns para indicar que se desean eliminar columnas en lugar de filas. Se pasa una lista de nombres de columnas que se desean eliminar del DataFrame nivel.
3. **['Latitud', 'Longitud', 'Altitud', 'Categoria', 'Entidad', 'AreaOperativa', 'Departamento', 'Municipio', 'FechaInstalacion', 'FechaSuspension', 'IdParametro', 'Etiqueta', 'DescripcionSerie', 'Frecuencia', 'Grado', 'Calificador', 'NivelAprobacion']**:
   * Esta es la lista de nombres de las columnas que se desean eliminar del DataFrame. Las columnas incluyen información geográfica (como Latitud, Longitud, Altitud) y detalles administrativos o de metadatos (como Categoria, Entidad, AreaOperativa, etc.). Se puede inferir que estas columnas no son relevantes para el análisis que estás realizando.
4. **inplace=True**:
   * Este parámetro indica que la operación debe realizarse en el mismo DataFrame nivel en lugar de crear una copia del mismo. Al establecer inplace=True, se modifica el DataFrame original directamente y no se devuelve un nuevo DataFrame. Si fuera inplace=False (el valor por defecto), tendrías que asignar el resultado a una nueva variable para mantener los cambios.

### Resumen

Este código elimina un conjunto específico de columnas del DataFrame nivel que no son necesarias para el análisis. Esto es un paso común en la preparación de datos, donde se eliminan características que no contribuyen al modelo o al análisis que se está llevando a cabo. Al limpiar el conjunto de datos de esta manera, puedes mejorar la eficiencia del procesamiento y el rendimiento del modelo.

**# Estandarizamos los formatos de fecha en todas las bases de datos**

Estas líneas de código están convirtiendo las columnas de fechas en diferentes DataFrames para que todas tengan el mismo formato y sean fáciles de manipular.

1. **nivel['Fecha'] = pd.to\_datetime(nivel['Fecha'], format='%Y-%m-%d %H:%M')**:
   * Aquí se está tomando la columna Fecha del DataFrame nivel y convirtiéndola al tipo de dato datetime usando la función pd.to\_datetime() de la biblioteca pandas.
   * El parámetro format='%Y-%m-%d %H:%M' especifica el formato de la fecha que ya existe en la columna:
     + %Y es el año en 4 dígitos,
     + %m es el mes en 2 dígitos,
     + %d es el día en 2 dígitos,
     + %H:%M representa la hora y minutos.
2. **precipitacion['Fecha'] = pd.to\_datetime(precipitacion['Fecha'], format='%m/%d/%Y')**:
   * En esta línea, se está convirtiendo la columna Fecha del DataFrame precipitacion a un formato de fecha.
   * El formato aquí es '%m/%d/%Y', donde:
     + %m es el mes en 2 dígitos,
     + %d es el día en 2 dígitos,
     + %Y es el año en 4 dígitos.
3. **temp['Fecha'] = pd.to\_datetime(temp['Fecha'], format='%m/%d/%Y')**:
   * Similar al paso anterior, esta línea convierte la columna Fecha del DataFrame temp al formato de fecha '%m/%d/%Y'.
4. **hum['Fecha'] = pd.to\_datetime(hum['Fecha'], format='%m/%d/%Y')**:
   * Esta línea convierte la columna Fecha del DataFrame hum al mismo formato de fecha '%m/%d/%Y'.
5. **vel['Fecha'] = pd.to\_datetime(vel['Fecha'], format='%m/%d/%Y')**:
   * Esta última línea convierte la columna Fecha del DataFrame vel también al formato de fecha '%m/%d/%Y'.

En resumen, estas líneas estandarizan el formato de las fechas en varias bases de datos para que tengan un tipo de dato consistente y faciliten el análisis de datos y la comparación entre ellas.

**# llevamos las variables que nos interesan a la base de datos principal usando como llave la variable fecha**

1. **df1 = pd.merge(nivel, precipitacion[['Fecha', 'Precipitacion']], how='left', on=['Fecha'])**:
   * Esta línea usa la función pd.merge() de pandas para combinar dos DataFrames: nivel y precipitacion.
   * Estamos tomando solo las columnas Fecha y Precipitacion del DataFrame precipitacion.
   * El parámetro how='left' indica que queremos realizar una *combinación izquierda* (left join), lo que significa que mantendremos todas las filas de nivel y agregaremos la columna Precipitacion de precipitacion en función de las coincidencias en la columna Fecha.
   * El parámetro on=['Fecha'] especifica que la combinación se hace usando la columna Fecha como clave.
   * El resultado se guarda en el nuevo DataFrame df1.
2. **df2 = pd.merge(df1, temp[['Fecha', 'Temperatura']], how='left', on=['Fecha'])**:
   * Esta línea combina df1 y el DataFrame temp, tomando únicamente las columnas Fecha y Temperatura de temp.
   * La combinación sigue siendo una *combinación izquierda* (left join) basada en la columna Fecha.
   * El resultado se guarda en el nuevo DataFrame df2, que ahora incluye la información de Temperatura en función de Fecha.
3. **df3 = pd.merge(df2, hum[['Fecha', 'Humedad']], how='left', on=['Fecha'])**:
   * Esta línea combina df2 y el DataFrame hum, utilizando solo las columnas Fecha y Humedad de hum.
   * Nuevamente, es una *combinación izquierda* y se basa en la columna Fecha.
   * El resultado se almacena en df3, que ahora incluye la columna Humedad basada en la fecha.
4. **df4 = pd.merge(df3, vel[['Fecha', 'Velocidad']], how='left', on=['Fecha'])**:
   * Finalmente, esta línea combina df3 y el DataFrame vel, utilizando solo las columnas Fecha y Velocidad de vel.
   * La combinación es también una *combinación izquierda* basada en Fecha.
   * El resultado final se almacena en df4, que ahora contiene las columnas Precipitacion, Temperatura, Humedad y Velocidad, todas alineadas por la variable Fecha.

En resumen, estas líneas combinan las variables seleccionadas de diferentes DataFrames en uno solo, usando la fecha como clave para unir los datos en una sola tabla principal (df4).

1. **df4['day'] = pd.DatetimeIndex(df4['Fecha']).day**:
   * Esta línea crea una nueva columna llamada day en el DataFrame df4.
   * pd.DatetimeIndex(df4['Fecha']).day convierte la columna Fecha en un índice de fecha y extrae el día del mes de cada fecha.
   * El valor del día de cada fecha se almacena en la columna day.
2. **df4['month'] = pd.DatetimeIndex(df4['Fecha']).month**:
   * Esta línea crea otra columna nueva llamada month en el DataFrame df4.
   * pd.DatetimeIndex(df4['Fecha']).month convierte la columna Fecha en un índice de fecha y extrae el mes de cada fecha.
   * El valor del mes de cada fecha se almacena en la columna month.
3. **df4.head(32)**:
   * Este comando muestra las primeras 32 filas del DataFrame df4.
   * head() es una función que permite ver las primeras filas de un DataFrame, y al especificar 32, estamos indicando que queremos ver las primeras 32 filas.

En conjunto, estas líneas extraen los días y meses de cada fecha en Fecha y los almacenan en columnas separadas (day y month), luego muestran las primeras 32 filas del DataFrame resultante.

**Verificar la forma de los datos**

**print(f"Forma de los datos (filas, columnas): {df4.shape}")**

1. **df4.shape**:
   * shape es un atributo de los DataFrames en pandas que devuelve una tupla con dos valores: el número de filas y el número de columnas del DataFrame. Por ejemplo, si df4 tiene 100 filas y 10 columnas, df4.shape devolverá (100, 10).
2. **print(f"...")**:
   * Este es un *f-string*, una forma de crear cadenas de texto que permite insertar variables directamente dentro del texto usando {}.
   * Dentro del f-string, {df4.shape} se sustituirá por el valor de df4.shape, mostrando así el tamaño del DataFrame en filas y columnas.

En conjunto, esta línea imprime en la consola un mensaje con la forma (dimensiones) del DataFrame df4, indicando cuántas filas y columnas tiene. Por ejemplo, el mensaje podría ser: Forma de los datos (filas, columnas): (100, 10).

**Verificar los datos de cada columna**

**# @title verificar los tipos de datos de cada columna**

1. **print("\nTipos de datos por columna:")**:
   * print() es una función que muestra un texto en la consola.
   * El texto "\nTipos de datos por columna:" incluye \n, que representa un salto de línea para que el mensaje se muestre en una nueva línea.
   * Este mensaje es solo informativo, indicando que a continuación se mostrarán los tipos de datos de cada columna.
2. **print(df4.dtypes)**:
   * df4.dtypes es un atributo de los DataFrames en pandas que devuelve el tipo de datos de cada columna (por ejemplo, int64, float64, object, datetime64[ns], etc.).
   * print(df4.dtypes) muestra en la consola los tipos de datos de todas las columnas de df4.

En resumen, estas líneas imprimen un mensaje y luego los tipos de datos de cada columna en el DataFrame df4. Esto es útil para verificar que cada columna tenga el tipo de dato esperado.

**Contar los valores únicos por columna**

**#@title Contar los valores únicos por columna**

1. **print("\nValores únicos por columna:")**:
   * print() es una función que muestra texto en la consola.
   * "\nValores únicos por columna:" incluye \n (un salto de línea) para que el mensaje aparezca en una nueva línea.
   * Este mensaje es informativo, indicando que a continuación se mostrará el conteo de valores únicos por columna.
2. **print(df4.nunique())**:
   * df4.nunique() es un método de pandas que devuelve la cantidad de valores únicos en cada columna de df4.
   * nunique() calcula la cantidad de valores distintos en cada columna.
   * print(df4.nunique()) muestra estos resultados en la consola, indicando cuántos valores únicos hay en cada columna del DataFrame.

En resumen,

estas líneas imprimen un mensaje y luego muestran la cantidad de valores distintos (únicos) en cada columna de df4. Esto es útil para entender la variedad de datos en cada columna y detectar posibles datos repetidos.

**Verifica si hay datos faltantes**

**#@title Verificar si hay datos faltantes**

1. **print("\nDatos faltantes por columna:")**:
   * print() es una función que muestra un texto en la consola.
   * "\nDatos faltantes por columna:" incluye \n (un salto de línea) para que el mensaje aparezca en una nueva línea.
   * Este mensaje es informativo y sirve para indicar que se va a mostrar la cantidad de valores faltantes por columna.
2. **print(df4.isnull().sum())**:
   * df4.isnull() genera un DataFrame booleano donde cada valor es True si el valor correspondiente en df4 es nulo (falta) o False si no lo es.
   * .sum() suma los valores True (nulos) en cada columna, lo que da como resultado la cantidad total de datos faltantes en cada una.
   * print(df4.isnull().sum()) muestra estos conteos en la consola.

En resumen

, estas líneas imprimen un mensaje y luego muestran la cantidad de datos faltantes en cada columna de df4. Esto es útil para identificar qué columnas tienen datos incompletos.

**Verificación de los campos con relación a la base de datos almacenada en formato csv**

**# @title Verificación de los campos con relación a la composición de la base de dato almacenada en formato CSV**

1. **df\_analizado = analizar\_columnas(df4):**
   * analizar\_columnas(df4) llama a una función llamada analizar\_columnas, pasando el DataFrame df4 como argumento.
   * Esta función analizar\_columnas (que debe haber sido definida antes) probablemente realiza algún análisis específico en las columnas de df4, como obtener estadísticas, tipos de datos, valores únicos, o valores faltantes.
   * El resultado de esta función se asigna a una nueva variable llamada df\_analizado, que parece ser también un DataFrame.
2. **df\_analizado**:
   * Esta línea simplemente escribe df\_analizado, lo cual en un entorno interactivo de Python (como Jupyter Notebook) mostrará el contenido de df\_analizado en la salida.

En resumen, estas líneas ejecutan la función analizar\_columnas sobre el DataFrame df4, guardan el resultado en df\_analizado y luego muestran el contenido de df\_analizado.

**Realizamos descriptiva estadistica**

**# @title Realizamos la descriptiva estadística**

1. **df4.describe()**:
   * describe() es un método de pandas que genera un resumen estadístico de las columnas numéricas en un DataFrame.
   * Este resumen incluye varias estadísticas descriptivas, tales como:
     + **count**: el número de valores no nulos en cada columna.
     + **mean**: la media (promedio) de los valores en cada columna.
     + **std**: la desviación estándar, que mide la dispersión de los datos en relación a la media.
     + **min**: el valor mínimo en cada columna.
     + **25%**: el primer cuartil (el valor bajo el cual se encuentra el 25% de los datos).
     + **50%**: la mediana (el valor medio que divide a los datos en dos mitades).
     + **75%**: el tercer cuartil (el valor bajo el cual se encuentra el 75% de los datos).
     + **max**: el valor máximo en cada columna.

El resultado de df4.describe() será una tabla que muestra estas estadísticas para todas las columnas numéricas del DataFrame df4, lo cual es útil para entender la distribución y las características generales de los datos en df4.

**Análisis exploratorio de los datos de EDA**

**# @title Análisis exploratorio de datos (EDA)**

**#Configurar las opciones de visualizacion Sweetviz**

1. **sw.config\_parser.read\_string(""" ... """)**:
   * Esta línea configura las opciones de visualización para el paquete Sweetviz, que es una biblioteca de Python para el análisis exploratorio de datos (EDA).
   * read\_string(""" ... """) lee una cadena de texto que contiene varias configuraciones. Dentro de esta cadena, se establecen opciones como el diseño del HTML, la escala del contenido, y la configuración del ancho y alto de la visualización en un cuaderno de Jupyter.
   * Las opciones incluidas son:
     + **html\_layout**: establece el diseño del archivo HTML como "widescreen".
     + **html\_scale**: define la escala del HTML en 1.0.
     + **notebook\_layout**: también establece el diseño en "widescreen" para notebooks.
     + **notebook\_scale**: define la escala del notebook en 0.9.
     + **notebook\_width**: establece el ancho del notebook en 100%.
     + **notebook\_height**: establece la altura del notebook en 700.
     + **show\_logo**: desactiva la visualización del logo.
2. **nombre = 'Cuencas'**:
   * Aquí se asigna la cadena 'Cuencas' a la variable nombre, que probablemente se usará como título o nombre del conjunto de datos en el análisis.
3. **advert\_report = sw.analyze([df4, nombre])**:
   * Esta línea llama a la función analyze() de Sweetviz, que realiza el análisis exploratorio de datos en el DataFrame df4.
   * Se pasa una lista que contiene el DataFrame y el nombre asociado ('Cuencas').
   * El resultado del análisis se almacena en la variable advert\_report.
4. **advert\_report.show\_html('EDA\_df.html')**:
   * Esta línea genera un reporte en formato HTML del análisis realizado y lo guarda con el nombre 'EDA\_df.html'.
   * show\_html() crea y guarda el archivo HTML en el directorio de trabajo.
5. **almacenar\_archivo = input('Alamcenar archivo Si o No: ').title()**:
   * Esta línea solicita al usuario que ingrese una respuesta sobre si desea almacenar el archivo HTML. La respuesta se almacena en la variable almacenar\_archivo.
   * title() transforma la entrada del usuario para que la primera letra de cada palabra esté en mayúscula (por ejemplo, "Si" o "No").
6. **if almacenar\_archivo == 'Si':**:
   * Esta línea inicia una condición que verifica si el usuario ha respondido "Si".
7. **files.download('/content/EDA\_df.html')**:
   * Si la condición anterior se cumple (el usuario respondió "Si"), esta línea utiliza la función download() del módulo files para descargar el archivo 'EDA\_df.html' en el sistema del usuario.

En resumen, este código configura las opciones de visualización para Sweetviz, realiza un análisis exploratorio de datos sobre el DataFrame df4, genera un informe en HTML, y ofrece al usuario la opción de descargar ese informe.

**# borramos los registros con datos faltantes de precipitación, temperatura y humedad**

**df4.dropna(subset=['Precipitacion', 'Temperatura', 'Humedad'], inplace=True)**

se utiliza para eliminar filas del DataFrame df4 que contengan valores nulos (faltantes) en ciertas columnas. Aquí tienes la explicación en español:

1. **df4.dropna(...)**:
   * dropna() es un método de pandas que se utiliza para eliminar filas (o columnas) que contienen valores nulos.
2. **subset=['Precipitacion', 'Temperatura', 'Humedad']**:
   * El parámetro subset especifica una lista de columnas en las que se debe verificar la existencia de valores nulos.
   * En este caso, se están revisando las columnas Precipitacion, Temperatura y Humedad.
   * Esto significa que solo se eliminarán las filas donde al menos uno de estos tres valores sea nulo.
3. **inplace=True**:
   * Este parámetro indica que la operación se debe realizar directamente en el DataFrame original (df4), en lugar de devolver una copia modificada.
   * Al establecer inplace=True, se actualiza df4 eliminando las filas con valores nulos en las columnas especificadas sin necesidad de crear un nuevo DataFrame.

En resumen, esta línea elimina las filas del DataFrame df4 que tienen valores faltantes en las columnas Precipitacion, Temperatura y Humedad, asegurando que solo queden las filas con datos completos en esas columnas.

**# codificamos las variables day y month**

**#Target encoding**

**#encoder = ce.TargetEncoder(cols=['day', 'month'])**

**#df4[['day', 'month']] = encoder.fit\_transform(df4[['day', 'month']], df4['Nivel'])**

**#Leave one out encoding**

1. **encoder = ce.LeaveOneOutEncoder(cols=['day', 'month'])**:
   * Se crea una instancia de un codificador llamado LeaveOneOutEncoder de la biblioteca category\_encoders (importada como ce). Este codificador se utiliza para transformar las columnas categóricas day y month en variables numéricas.
   * cols=['day', 'month'] indica que se están seleccionando las columnas day y month para aplicar la codificación.
2. **df4[['day', 'month']] = encoder.fit\_transform(df4[['day', 'month']], df4['Nivel'])**:
   * Esta línea aplica el codificador LeaveOneOutEncoder a las columnas day y month del DataFrame df4.
   * fit\_transform() realiza dos operaciones: primero ajusta el codificador a los datos y luego transforma las columnas especificadas.
   * df4[['day', 'month']] indica que los resultados de la transformación se almacenarán nuevamente en las columnas day y month del DataFrame.
   * df4['Nivel'] se utiliza como la variable objetivo para la codificación, lo que significa que el codificador considera cómo los valores de day y month se relacionan con los valores en la columna Nivel durante el proceso de codificación.

En resumen, estas líneas de código utilizan el codificador de "Leave One Out" para transformar las variables categóricas day y month en representaciones numéricas en función de su relación con la variable objetivo Nivel, y luego reemplazan los valores originales de esas columnas en el DataFrame df4 con los nuevos valores codificados.

**# separamos caracteristicas y etiquetas**

1. **X, y = df4[['Precipitacion', 'Temperatura', 'Humedad', 'day', 'month']].values, df4['Nivel'].values**:
   * Esta línea separa las características (o variables independientes) y las etiquetas (o variable dependiente) del DataFrame df4.
   * df4[['Precipitacion', 'Temperatura', 'Humedad', 'day', 'month']] selecciona las columnas que se utilizarán como características, que en este caso son Precipitacion, Temperatura, Humedad, day y month.
   * .values convierte el DataFrame resultante en un array de NumPy, que se asigna a X.
   * df4['Nivel'] selecciona la columna Nivel, que es la variable objetivo (o etiqueta).
   * .values también convierte esta columna en un array de NumPy, que se asigna a y.
   * En resumen, X contiene las características y y contiene las etiquetas.
2. **print('Features:', X[:10], '\nLabels:', y[:10], sep='\n')**:
   * Esta línea imprime en la consola las primeras diez filas de las características y las etiquetas.
   * 'Features:' es una cadena que se imprime antes de mostrar los valores de X.
   * X[:10] selecciona las primeras diez filas del array X.
   * '\nLabels:' es otra cadena que se imprime después de un salto de línea, antes de mostrar las etiquetas.
   * y[:10] selecciona las primeras diez filas del array y.
   * sep='\n' especifica que cada parte de la impresión debe separarse por un salto de línea.

En resumen, estas líneas de código separan las características y etiquetas del DataFrame df4, y luego imprimen las primeras diez filas de cada uno para que el usuario pueda ver un vistazo de los datos que se están utilizando.

**Seleccionar solo las columnas numéricas del dataframe**

**#@title Seleccionar solo las columnas numéricas del DataFrame**

1. **numeric\_cols = df4[['Precipitacion', 'Temperatura', 'Humedad', 'day', 'month']].select\_dtypes(include=['number'])**:
   * Esta línea selecciona solo las columnas numéricas del DataFrame df4.
   * df4[['Precipitacion', 'Temperatura', 'Humedad', 'day', 'month']] selecciona las columnas especificadas.
   * .select\_dtypes(include=['number']) filtra estas columnas, devolviendo solo aquellas que contienen datos numéricos. El resultado se asigna a la variable numeric\_cols.
2. **conf\_matrix = numeric\_cols.corr()**:
   * Esta línea calcula la matriz de correlación para las columnas numéricas seleccionadas.
   * numeric\_cols.corr() devuelve un nuevo DataFrame que muestra la correlación entre cada par de columnas numéricas en numeric\_cols.
   * La matriz de correlación mide la relación lineal entre las variables, con valores que van de -1 a 1, donde 1 indica una correlación positiva perfecta, -1 indica una correlación negativa perfecta, y 0 indica que no hay correlación.
3. **conf\_matrix**:
   * Esta línea simplemente devuelve el DataFrame conf\_matrix, que contiene la matriz de correlación calculada. En un entorno interactivo como Jupyter Notebook, esto mostrará la matriz en la salida.

En resumen, este código selecciona las columnas numéricas del DataFrame df4, calcula la matriz de correlación entre ellas, y luego muestra el resultado, lo que es útil para analizar las relaciones entre las variables numéricas.

**Visualización de la correlación entre variables numéricas**

**# @title Visualización de la correlación entre variables numéricas**

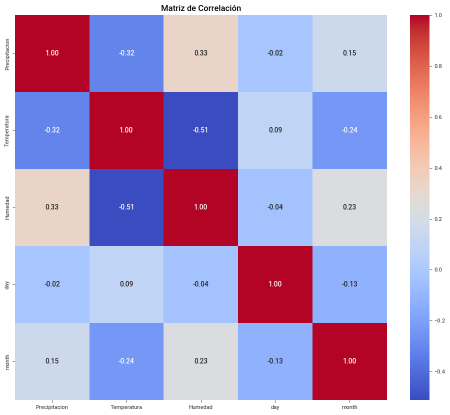
plt.figure(figsize=(12, 10)): Esta línea crea una figura de tamaño 12x10 (ancho por alto) usando Matplotlib. Esto define el área sobre la cual se dibujará el gráfico.

sns.heatmap(conf\_matrix, annot=True, cmap='coolwarm', fmt=".2f"):

* sns.heatmap(...) crea un mapa de calor (heatmap) usando la librería Seaborn.
* conf\_matrix es la matriz de confusión (o cualquier matriz de datos que estás visualizando).
* annot=True muestra los valores numéricos dentro de cada celda de la matriz.
* cmap='coolwarm' define la escala de colores, que en este caso usa una paleta de colores de "frío a cálido".
* fmt=".2f" establece el formato de los números dentro de las celdas, mostrando dos decimales.

plt.title('Matriz de Correlación'): Esta línea agrega un título a la gráfica, en este caso, “Matriz de Correlación”.

plt.show(): Muestra la figura generada en pantalla.



### . ¿Qué es una matriz de correlación?

Una **matriz de correlación** es una tabla que muestra los coeficientes de correlación entre múltiples variables. Cada celda de la matriz contiene el coeficiente de correlación entre dos variables, que puede ir de **-1** a **1**. Este coeficiente indica la relación entre las variables de la siguiente manera:

* **1** significa **correlación positiva perfecta**: cuando una variable aumenta, la otra también aumenta en la misma proporción.
* **0** significa **sin correlación**: no hay una relación lineal directa entre las dos variables.
* **-1** significa **correlación negativa perfecta**: cuando una variable aumenta, la otra disminuye en la misma proporción.

### 2. Interpretación de los colores en el mapa de calor

El mapa de calor utiliza una escala de colores para representar visualmente estos coeficientes:

* Los colores **rojos** (en este caso, de intensidad media a alta) indican una **correlación positiva** (coeficientes positivos).
* Los colores **azules** (de intensidad media a alta) indican una **correlación negativa** (coeficientes negativos).
* Los colores **neutros** (como los tonos blancos o grises) indican **correlaciones cercanas a cero**, lo que sugiere una relación muy débil o nula entre las variables.

### 3. Análisis de las correlaciones entre variables específicas

Veamos en detalle las relaciones que aparecen en la matriz:

#### Precipitación

* La **precipitación** muestra una correlación negativa moderada con la **temperatura** (-0.32). Esto sugiere que, en general, cuando la temperatura baja, la precipitación tiende a aumentar, o viceversa, aunque esta relación no es extremadamente fuerte.
* También hay una correlación positiva moderada con la **humedad** (0.33), lo que indica que cuando aumenta la humedad, la precipitación tiende a ser más alta.
* La correlación con las variables day y month es débil (cercana a 0), lo que indica que la precipitación no tiene una relación significativa con el día o el mes en este conjunto de datos.

#### Temperatura

* La **temperatura** tiene una correlación negativa significativa con la **humedad** (-0.51). Esto implica que en días con temperaturas más altas, la humedad tiende a ser más baja, lo cual es coherente con condiciones climáticas típicas.
* La correlación con las variables day y month es baja, especialmente con day, lo que sugiere que la temperatura no varía de manera sistemática con el día en este conjunto de datos, aunque puede variar ligeramente con el mes (-0.24).

#### Humedad

* La **humedad** tiene una correlación moderada y positiva con la **precipitación** (0.33), lo cual tiene sentido porque días con más humedad pueden facilitar la formación de lluvias.
* Al igual que con las otras variables, la **humedad** no muestra una correlación significativa con day o month, lo que sugiere que estos valores no influyen mucho en los niveles de humedad.

#### day y month

* Estas variables (day y month) tienen una correlación débil con las otras variables climáticas (precipitación, temperatura y humedad), indicando que no están fuertemente relacionadas con las condiciones meteorológicas en este conjunto de datos.
* La correlación entre day y month es ligeramente negativa (-0.13), lo que puede deberse a cómo están distribuidos los datos en tu conjunto, pero no representa una relación de interés climática.

**Visualizar la varianza de las variables numéricas con numeric\_cols.var() en barra y líneas juntas**

**#@title visualizar la varianza de las variables numéricas con numeric\_cols.var() en barras y en linea juntas**

### variance = numeric\_cols.var()

Esta línea calcula la varianza de cada columna numérica en numeric\_cols. La varianza mide la dispersión de los datos; es decir, qué tan lejos están los valores de una variable respecto a su media. Cuanto mayor sea la varianza, mayor es la dispersión de los datos.

### # Crear Gráfico de Barras

Este bloque de código crea un gráfico de barras para visualizar la varianza de cada variable numérica:

* plt.figure(figsize=(10, 6)) establece el tamaño del gráfico.
* plt.bar(variance.index, variance.values) crea las barras donde:
  + variance.index representa los nombres de las variables numéricas en el eje X.
  + variance.values representa los valores de la varianza en el eje Y.
* plt.title('Varianza de las variables numéricas (Barras)') añade un título al gráfico.
* plt.xlabel('Variables') y plt.ylabel('Varianza') etiquetan los ejes.
* plt.xticks(rotation=90) rota los nombres de las variables en el eje X 90 grados para mejorar la legibilidad.
* plt.show() muestra el gráfico en pantalla.

### # Crear Gráfico de Línea

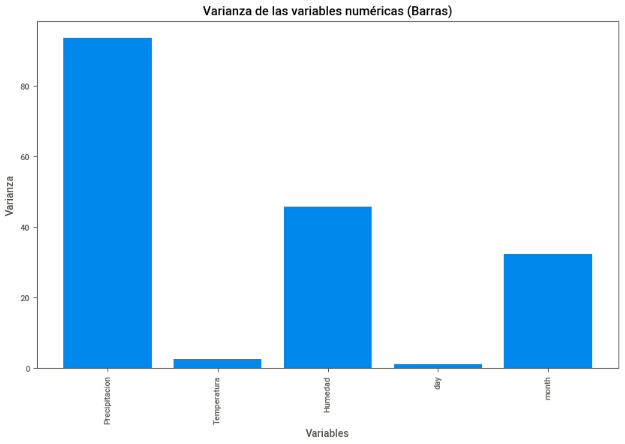
Este bloque de código crea un gráfico de línea para visualizar la varianza:

* plt.plot(variance.index, variance.values) crea la línea donde:
  + variance.index representa los nombres de las variables numéricas en el eje X.
  + variance.values representa los valores de la varianza en el eje Y.
* plt.title('Varianza de las variables numéricas (Línea)') añade un título al gráfico.
* plt.xlabel('Variables') y plt.ylabel('Varianza') etiquetan los ejes.
* plt.xticks(rotation=90) rota los nombres de las variables en el eje X 90 grados.
* plt.show() muestra el gráfico.

### Comparación de ambos gráficos

* **Gráfico de Barras**: Es útil para ver la magnitud de la varianza de cada variable individualmente y comparar visualmente entre ellas.
* **Gráfico de Línea**: Permite ver tendencias de varianza de manera más continua, especialmente si las variables están organizadas en un orden lógico.

Estos dos gráficos juntos ofrecen una visión complementaria de la dispersión en tus datos, ayudándote a identificar variables que tienen una mayor o menor variabilidad.



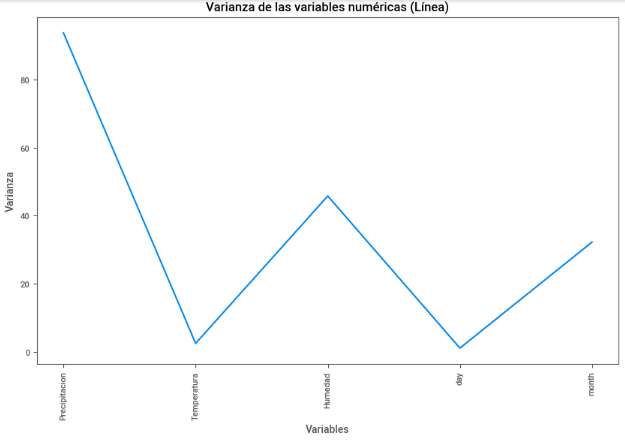
En esta gráfica de barras, se muestra la **varianza** de cada variable numérica de tu conjunto de datos. La varianza refleja cuánto varían los valores de cada variable respecto a su media; cuanto mayor sea la varianza, mayor es la dispersión de los datos.

### Interpretación de cada barra:

1. **Precipitación**: Tiene la mayor varianza, con un valor de alrededor de 80. Esto indica que los valores de la precipitación en tu conjunto de datos están muy dispersos en torno a su media. Es la variable con más variabilidad entre todas.
2. **Humedad**: Tiene una varianza considerable, pero significativamente menor que la de la precipitación. Esto sugiere que los valores de humedad también presentan una dispersión considerable, aunque menor que la precipitación.
3. **Temperatura** y **day**: Estas variables muestran una varianza muy baja, cercana a 0. Esto indica que sus valores son bastante constantes en el conjunto de datos y apenas varían respecto a su media. La baja variabilidad puede indicar que la temperatura tiene valores similares a lo largo del periodo analizado, y lo mismo ocurre con el día, que podría estar en una escala discreta con poco rango.
4. **month**: La varianza de month es menor que la de humedad pero superior a la de temperatura y day. Esto sugiere que los valores de month varían en cierto grado, pero no tanto como la precipitación o la humedad. Esto es esperable si month representa los meses del año, que tienen un rango limitado.

### Conclusión general

La gráfica sugiere que la **precipitación** es la variable con mayor dispersión en tus datos, seguida de la **humedad**. Las otras variables tienen una varianza mucho menor, indicando que sus valores son más consistentes o presentan menor variabilidad. Esta información es útil para entender cómo se comportan las diferentes variables en el conjunto y para tomar decisiones sobre su relevancia en análisis posteriores.



En esta gráfica de línea se muestra la **varianza** de cada variable numérica en tu conjunto de datos, lo que permite visualizar la dispersión de cada variable respecto a su media. La variación en los valores de cada variable se representa en el eje Y, mientras que las variables están dispuestas en el eje X.

### Análisis de la gráfica:

1. **Pico alto en Precipitación**: La **precipitación** muestra la varianza más alta, lo que indica que esta variable tiene la mayor dispersión en los datos, es decir, sus valores fluctúan ampliamente respecto a su media.
2. **Valle en Temperatura**: La **temperatura** tiene una varianza muy baja, cercana a cero, lo cual sugiere que los valores de temperatura no varían mucho. Es decir, la temperatura es bastante constante en el conjunto de datos.
3. **Pico medio en Humedad**: La **humedad** tiene una varianza significativa (menor que la de precipitación, pero considerable). Esto implica que la humedad presenta una dispersión moderada, por lo que hay cierta variación en sus valores.
4. **Valle en day**: Similar a la temperatura, el **día** (day) muestra una varianza muy baja, lo que sugiere que sus valores también son bastante constantes. Esto es esperable si el valor de day está limitado a un rango de días en el mes y no varía mucho en un análisis climático.
5. **Aumento en month**: La **variable month** muestra una varianza mayor que day y temperatura, aunque sigue siendo menor que la de precipitación y humedad. Esto refleja que hay algo de variación en los meses representados, pero dentro de un rango limitado (probablemente de 1 a 12).

### Conclusión general

Esta gráfica de línea permite ver una tendencia clara: **precipitación** y **humedad** son las variables con mayor variabilidad en los datos, mientras que **temperatura** y **day** tienen poca o nula variación. Esto puede ayudarte a identificar qué variables tienen más cambios y podrían ser más influyentes en análisis de predicción o modelado.

**# Dividimos los datos 70%-30% en datos de entrenamiento y datos de pruebas**

Este código divide el conjunto de datos en dos partes: **datos de entrenamiento** (70%) y **datos de prueba** (30%). La división es aleatoria, pero se utiliza una semilla (o estado aleatorio) con random\_state=0 para asegurar que la división sea la misma cada vez que se ejecuta el código.

### Explicación detallada

1. **División de los datos**  
   X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=0)
   * train\_test\_split(X, y, test\_size=0.30, random\_state=0):
     + X y y son las variables de entrada y salida, respectivamente.
     + test\_size=0.30 indica que el 30% de los datos se destinará para pruebas y el 70% para entrenamiento.
     + random\_state=0 fija una semilla para la aleatoriedad, garantizando que los datos se dividan de la misma manera cada vez que se ejecute el código.
   * La función devuelve cuatro subconjuntos:
     + X\_train y y\_train contienen los datos de entrenamiento para las variables de entrada y salida.
     + X\_test y y\_test contienen los datos de prueba para las variables de entrada y salida.
2. **Impresión del tamaño de los subconjuntos**  
   print('Datos de entrenamiento: %d rows\nDatos de prueba: %d rows' % (X\_train.shape[0], X\_test.shape[0]))
   * X\_train.shape[0] obtiene el número de filas en los datos de entrenamiento.
   * X\_test.shape[0] obtiene el número de filas en los datos de prueba.
   * Finalmente, imprime el número de filas de cada subconjunto, lo cual te permite verificar que la división de datos se realizó correctamente (70% para entrenamiento y 30% para pruebas).

# Entrenamiento del modelo

En este código, estás entrenando un modelo de **regresión lineal** con los datos de entrenamiento que dividiste previamente. La regresión lineal es un modelo de aprendizaje supervisado que encuentra la mejor línea recta que minimiza el error entre las predicciones y los datos reales.

### 

model = LinearRegression().fit(X\_train, y\_train)

* + LinearRegression(): Crea una instancia del modelo de regresión lineal de la biblioteca sklearn.
  + .fit(X\_train, y\_train): Ajusta el modelo de regresión lineal usando los datos de entrenamiento (X\_train para las variables independientes y y\_train para la variable dependiente).
  + Al entrenar el modelo, este aprende la relación entre las variables de entrada (X\_train) y la salida (y\_train), calculando los coeficientes de la línea recta que mejor ajusta los datos.

1. **Impresión del modelo entrenado**  
   print(model)
   * Este print muestra una descripción básica del modelo entrenado. No proporciona información detallada sobre los coeficientes o la precisión, pero confirma que el modelo ha sido entrenado con éxito.

### ¿Qué puedes hacer después?

Después de entrenar el modelo, puedes:

* Evaluar su rendimiento en los datos de prueba (X\_test y y\_test) para ver qué tan bien generaliza a nuevos datos.
* Examinar los coeficientes y la intersección (pendiente y ordenada al origen) del modelo para entender la influencia de cada variable en la predicción.

**Evaluación del modelo entrenado**

**# Generamos las predicciones con los datos de prueba**

Este código genera predicciones usando el modelo de regresión lineal entrenado y las compara con los valores reales de las etiquetas de los datos de prueba. Aquí tienes una explicación detallada:

predictions = model.predict(X\_test)

* + model.predict(X\_test): Usa el modelo entrenado para hacer predicciones con los datos de prueba X\_test.
  + El resultado es un arreglo (predictions) que contiene las etiquetas predichas para cada instancia en X\_test.

1. **Ajuste de opciones de impresión**  
   np.set\_printoptions(suppress=True)
   * Esta línea configura las opciones de impresión de NumPy para suprimir la notación científica, de modo que los valores se muestren en formato decimal normal (útil para mejorar la legibilidad de los resultados).
2. **Impresión de predicciones y etiquetas reales**  
   print('Etiquetas predichas: ', np.round(predictions)[:10])

print('Etiquetas actuales : ', y\_test[:10])

* + np.round(predictions)[:10]: Redondea las primeras 10 predicciones a números enteros (para facilitar la comparación con las etiquetas reales).
  + y\_test[:10]: Muestra las primeras 10 etiquetas reales (valores objetivos) en el conjunto de prueba.
  + Imprime las primeras 10 etiquetas predichas y las correspondientes etiquetas reales, lo cual te permite observar qué tan cerca están las predicciones de los valores reales.

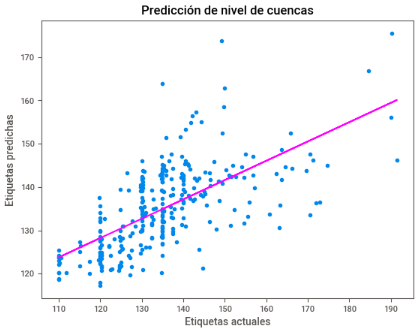
### Propósito del código

Este código es útil para evaluar visualmente si el modelo está realizando buenas predicciones. Si las etiquetas predichas son cercanas a las etiquetas actuales, esto indica que el modelo ha capturado bien la relación entre las variables de entrada y la salida en los datos de prueba. Para una evaluación cuantitativa, se pueden calcular métricas como el error cuadrático medio (MSE) o el coeficiente de determinación (R²).

**# Muestra las gráficas generadas en la misma celda**

El código que has compartido es para visualizar las predicciones de un modelo de regresión frente a los valores reales.

1. **%matplotlib inline**: Esta línea permite que las gráficas de Matplotlib se muestren directamente en la celda de Jupyter Notebook.
2. **plt.scatter(y\_test, predictions)**: Esta función crea un diagrama de dispersión donde el eje X representa las etiquetas reales (y\_test) y el eje Y representa las etiquetas predichas (predictions).
3. **Etiquetas y título**: Las líneas plt.xlabel, plt.ylabel y plt.title añaden etiquetas a los ejes y un título a la gráfica para hacerla más informativa.
4. **Ajuste de línea de regresión**:
   * **np.polyfit(y\_test, predictions, 1)**: Esta función calcula los coeficientes de una línea de regresión lineal (grado 1) que mejor se ajusta a los puntos de datos.
   * **np.poly1d(z)**: Crea una función polinómica a partir de los coeficientes obtenidos con np.polyfit.
   * **plt.plot(y\_test, p(y\_test), color='magenta')**: Esta línea traza la línea de regresión en la misma gráfica.
5. **plt.show()**: Finalmente, se muestra la gráfica.



**Introducción:** Esta gráfica muestra cómo de bien un modelo matemático predice el nivel de cuencas (probablemente en relación al agua o al caudal). Compara los valores actuales, o reales, con los valores que el modelo ha predicho.

**Descripción de los Ejes:**

1. **Eje X** - Etiquetas actuales: Aquí se representan los valores reales del nivel de cuenca, basados en mediciones o datos existentes.
2. **Eje Y** - Etiquetas predichas: Aquí están los valores predichos por el modelo, es decir, los resultados que el modelo cree que deberían ser.

**Puntos en la Gráfica:**

* Cada punto azul en la gráfica representa una predicción que el modelo hizo. Idealmente, cada predicción debería coincidir con el valor real.
* Si el modelo fuera perfecto, todos los puntos azules estarían alineados en una línea diagonal, porque en un modelo ideal los valores predichos y los actuales son iguales.

**Línea de Tendencia (Línea Morada):**

* La línea morada muestra la tendencia general de las predicciones frente a los valores reales. Cuanto más cerca están los puntos de esta línea, mejor está prediciendo el modelo.
* Si la línea tiene una pendiente de 1 y pasa cerca del origen (esquina inferior izquierda), significa que el modelo predice bien en promedio.

**Interpretación de la Gráfica:**

* Observamos que muchos puntos están cerca de la línea, lo que significa que el modelo está haciendo un trabajo decente en predecir el nivel de cuenca. Sin embargo, también hay puntos alejados de la línea, indicando errores de predicción en algunos casos.
* Esta dispersión de puntos indica que, aunque el modelo tiene una relación con los valores reales, no siempre es preciso y puede mejorar.

**Conclusión:**

* En general, esta gráfica nos dice que el modelo es útil para predecir el nivel de cuencas, pero que no es perfecto. Hay ciertos casos en los que el modelo falla, y eso se ve en los puntos alejados de la línea de tendencia.
* Para mejorar el modelo, podríamos considerar agregar más datos, ajustar los parámetros o incluso probar otros métodos de modelado.

**# calculamos las métricas para este modelo**

Para calcular las métricas de este modelo y evaluar su rendimiento, estás utilizando tres métricas comunes:

1. **Error Cuadrático Medio (MSE)**:
   * mse = mean\_squared\_error(y\_test, predictions)
   * Este valor mide el error promedio entre los valores reales (y\_test) y las predicciones (predictions). Un valor más bajo de MSE indica una mayor precisión del modelo.
2. **Raíz del Error Cuadrático Medio (RMSE)**:
   * rmse = np.sqrt(mse)
   * La RMSE es la raíz cuadrada del MSE y mantiene las mismas unidades que los datos originales, lo cual facilita la interpretación. Un valor bajo también sugiere que el modelo es preciso.
3. **Coeficiente de Determinación (R²)**:
   * r2 = r2\_score(y\_test, predictions)
   * Esta métrica indica qué porcentaje de la variabilidad de los datos reales es explicada por el modelo. Un R² cercano a 1 indica que el modelo se ajusta bien a los datos, mientras que un valor cercano a 0 sugiere que el modelo no es efectivo.

Estos cálculos permiten entender cómo de bien está funcionando el modelo en términos de precisión (RMSE), promedio de error (MSE) y ajuste general (R²).

**Regresión Polinómica**

# Entrenamiento del modelo de regresión polinómica

Este código entrena un modelo de regresión polinómica para predecir el nivel de cuencas, o cualquier otra variable en tus datos.

1. **Definir el Grado del Polinomio**:  
     
   poli\_reg = PolynomialFeatures(degree = 2)

Aquí estamos creando un objeto PolynomialFeatures con un grado de 2, lo que significa que estamos transformando las variables originales en variables de segundo grado. Si quisieras un modelo más complejo, podrías aumentar el grado del polinomio (por ejemplo, cambiando degree = 2 a degree = 7, como sugiere el comentario).

1. **Transformar las Características**:  
     
   X\_train\_poli = poli\_reg.fit\_transform(X\_train)

X\_test\_poli = poli\_reg.fit\_transform(X\_test)

En este paso, el conjunto de entrenamiento y el conjunto de prueba se transforman para incluir términos de segundo grado (como x2x^2x2, xyxyxy, etc.). Esto permite al modelo capturar relaciones no lineales entre las variables.

1. **Definir el Modelo de Regresión Lineal**:  
     
   pr = LinearRegression()

Aquí se crea un modelo de regresión lineal, pero este modelo será entrenado en las características transformadas polinómicamente, lo que lo convierte en un modelo de regresión polinómica.

1. **Entrenamiento del Modelo**:  
     
   pr.fit(X\_train\_poli, y\_train)

En esta línea, el modelo se entrena utilizando las características de entrenamiento transformadas (X\_train\_poli) y las etiquetas reales (y\_train). Aquí el modelo aprende los coeficientes que mejor representan la relación entre las variables transformadas y el valor a predecir.

1. **Realizar una Predicción**:  
     
   Y\_pred\_pr = pr.predict(X\_test\_poli)

Una vez que el modelo está entrenado, podemos hacer predicciones sobre el conjunto de prueba (X\_test\_poli), que también ha sido transformado con las características polinómicas.

1. **Resultados del Modelo**:

* **Coeficientes (pendientes)**:  
    
  print('Valor de la pendiente o coeficiente "a":')

print(pr.coef\_)

* + Esto imprime los coeficientes del modelo (los valores "a" de la ecuación polinómica), que muestran la influencia de cada término en la predicción.
* **Intersección (intercepto)**:  
    
  print('Valor de la intersección o coeficiente "b":')

print(pr.intercept\_)

* + Esto imprime el valor de la intersección o intercepto "b" de la ecuación, que representa el valor inicial del modelo cuando todas las variables independientes son 0.
* **Precisión del Modelo**:  
    
  print('Precisión del modelo:')

print(pr.score(X\_train\_poli, y\_train))

* + pr.score() calcula el R² del modelo en los datos de entrenamiento, indicando qué tan bien se ajusta el modelo a esos datos. Un valor cercano a 1 significa un buen ajuste, mientras que un valor bajo sugiere que el modelo podría mejorarse.

### Resumen

Este proceso crea un modelo de regresión polinómica de grado 2, que captura relaciones no lineales entre las variables, mejorando la precisión en comparación con un modelo de regresión lineal simple. Con estos resultados, podrías analizar si aumentar el grado del polinomio ayuda a mejorar el ajuste o si el modelo comienza a sobreajustarse a los datos.

# Calcular el R^2 en el conjunto de prueba

Esta línea de código calcula el coeficiente de determinación R2R^2R2 para el conjunto de prueba, lo que te permitirá evaluar la precisión del modelo en datos que no fueron utilizados durante el entrenamiento:

r2\_test = r2\_score(y\_test, Y\_pred\_pr)

print('R^2 en el conjunto de prueba:', r2\_test)

* **r2\_score(y\_test, Y\_pred\_pr)**: Esta función calcula el valor R2R^2R2, que representa qué tan bien el modelo predice los datos de prueba (y\_test) en comparación con los valores predichos (Y\_pred\_pr).
* **Interpreta el R²**:
  + Un valor de R2R^2R2 cercano a 1 indica que el modelo tiene un buen ajuste y puede explicar gran parte de la variabilidad en los datos de prueba.
  + Un valor más bajo (cercano a 0) sugiere que el modelo no es bueno para predecir en el conjunto de prueba y podría necesitar ajustes.

Este R2R^2R2 en el conjunto de prueba es una métrica importante porque muestra si el modelo generaliza bien o si está sobreajustado a los datos de entrenamiento.

**# Entrenamiento del módelo de Random Forest Regressor usando los datos de entrenamiento**

Este código entrena un modelo de **Random Forest Regressor** para predecir el nivel de cuencas y luego evalúa su rendimiento usando métricas y una gráfica de comparación entre los valores actuales y las predicciones. Aquí tienes una explicación de cada paso:

**# Entrenamiento del Modelo:**  
  
model = RandomForestRegressor().fit(X\_train, y\_train)

print(model, "\n")

Aquí se crea y entrena un modelo de bosque aleatorio (Random Forest Regressor) usando los datos de entrenamiento (X\_train y y\_train). Los bosques aleatorios son potentes en tareas de regresión porque combinan múltiples árboles de decisión para mejorar la precisión y reducir el riesgo de sobreajuste.

**# Predicción**:

predictions = model.predict(X\_test)

Después de entrenar el modelo, se utiliza para hacer predicciones sobre el conjunto de prueba (X\_test).

**# Cálculo de Métricas de Evaluación**:

**# Error Cuadrático Medio (MSE):**  
  
mse = mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

* + El MSE mide el error promedio entre los valores reales y las predicciones. Un MSE bajo indica que el modelo predice con mayor precisión.

**# Raíz del Error Cuadrático Medio (RMSE):**  
rmse = np.sqrt(mse)

print("RMSE:", rmse)

* + La RMSE es la raíz cuadrada del MSE y está en las mismas unidades que los datos, lo que facilita la interpretación de los errores promedio.

**# Coeficiente de Determinación (R²):**  
  
r2 = r2\_score(y\_test, predictions)

print("R2:", r2)

* + El R2R^2R2 indica qué tan bien el modelo explica la variabilidad de los datos de prueba. Un valor cercano a 1 indica un buen ajuste.

**# Gráfica de Predicción vs. Valores Actuales:**  
  
plt.scatter(y\_test, predictions)

plt.xlabel('Etiquetas actuales')

plt.ylabel('Etiquetas predichas')

plt.title('Predicción de nivel de cuencas')

Este gráfico de dispersión compara los valores reales (y\_test) en el eje X con los valores predichos (predictions) en el eje Y. Cada punto representa una predicción. Si el modelo fuera perfecto, todos los puntos caerían en una línea diagonal.

**# Sobreponer la Línea de Regresión:**

z = np.polyfit(y\_test, predictions, 1)

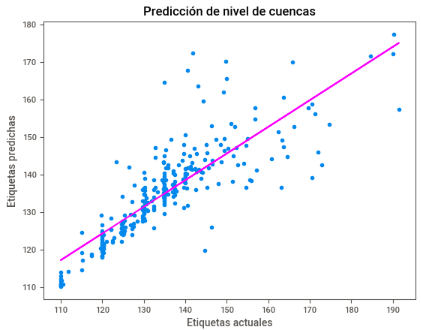
p = np.poly1d(z)

plt.plot(y\_test, p(y\_test), color='magenta')

plt.show()

Aquí se ajusta una línea de regresión a los datos predichos para visualizar mejor la relación entre las etiquetas actuales y las predichas. Esta línea ayuda a ver si el modelo tiende a sobreestimar o subestimar los valores.

Esta evaluación te da una visión completa de la calidad del modelo Random Forest Regressor para predecir el nivel de cuencas.



**Descripción General:** Esta gráfica muestra la comparación entre los valores reales y los valores predichos por un modelo de regresión. En este caso, el modelo ha sido diseñado para predecir el nivel de cuencas, y la gráfica nos ayuda a visualizar qué tan precisas son esas predicciones.

**Explicación de los Ejes:**

1. **Eje X (Etiquetas actuales):** Representa los valores reales o medidos del nivel de cuencas, es decir, los datos que queremos que el modelo prediga.
2. **Eje Y (Etiquetas predichas):** Representa los valores que el modelo ha predicho basándose en los datos de prueba.

**Interpretación de los Puntos en la Gráfica:**

* Cada punto azul en la gráfica representa una comparación entre el valor real y el valor predicho para un dato específico.
* Si el modelo fuera perfecto, todos los puntos estarían alineados en una línea diagonal, donde las predicciones coinciden exactamente con los valores reales.

**Línea de Regresión (Línea Morada):**

* La línea morada representa la tendencia general de las predicciones en relación con los valores reales.
* Cuanto más cerca están los puntos de esta línea, mejor está prediciendo el modelo, ya que indica que las etiquetas predichas son similares a las etiquetas actuales.
* Si la pendiente de esta línea es cercana a 1, eso sugiere que el modelo predice con bastante precisión, pero si la línea se aleja mucho de los puntos, es una señal de que el modelo necesita ajustes.

**Conclusión:**

* La gráfica muestra que, en general, las predicciones están cerca de los valores reales, pero también hay cierta dispersión, especialmente en valores altos. Esto indica que el modelo hace un buen trabajo en promedio, pero puede fallar en algunos casos específicos.
* Para mejorar la precisión, podríamos considerar ajustar el modelo o probar con otros modelos de predicción.

**# Entrenamos el modelo Decision Tree Regressor con los datos de entrenamiento**

Este código entrena un modelo de **Árbol de Decisión para Regresión** (Decision Tree Regressor) y luego visualiza la estructura del árbol entrenado.

1. **Entrenamiento del Modelo**:  
     
   model = DecisionTreeRegressor().fit(X\_train, y\_train)

print(model, "\n")

En esta línea, se crea y entrena un modelo de árbol de decisión para regresión usando los datos de entrenamiento (X\_train y y\_train). Los árboles de decisión funcionan dividiendo el espacio de datos en regiones más pequeñas, y cada nodo del árbol representa una regla de decisión que ayuda a predecir el valor final.

1. **Visualización del Árbol de Decisión**:  
     
   tree = export\_text(model)

print(tree)

Aquí, se utiliza la función export\_text para obtener una representación en texto del árbol de decisión entrenado. Este texto describe la estructura del árbol, mostrando las reglas de decisión que el modelo ha aprendido. Cada nodo del árbol representa una división en los datos basada en un umbral específico de una de las características, y las hojas del árbol representan los valores de predicción finales.

### Ejemplo de la Salida del Árbol

La salida podría verse algo como esto:

|--- feature\_1 <= 10.5

| |--- feature\_3 <= 5.0

| | |--- prediction: 120.0

| |--- feature\_3 > 5.0

| | |--- prediction: 130.0

|--- feature\_1 > 10.5

| |--- feature\_2 <= 7.0

| | |--- prediction: 140.0

Cada línea representa una decisión que el modelo toma para dividir los datos, y en las hojas (predictions) está el valor final que se usa para las predicciones en esas ramas.

### Interpretación

El árbol de decisión muestra cómo el modelo segmenta los datos en función de los valores de las características (features). Esto te da una idea de qué características son las más relevantes y cómo contribuyen a la predicción final. La profundidad del árbol también indica la complejidad del modelo; un árbol más profundo tiene más reglas, lo que puede ayudar en precisión, pero también aumenta el riesgo de sobreajuste.

**# Evaluamos el modelo usando los datos de prueba**

Este bloque de código evalúa el modelo **Decision Tree Regressor** usando el conjunto de datos de prueba y muestra las métricas de rendimiento junto con una gráfica de comparación entre los valores reales y las predicciones.

predictions = model.predict(X\_test)

El modelo entrenado se utiliza para hacer predicciones sobre el conjunto de prueba (X\_test). Esto genera un conjunto de valores predichos (predictions) que luego se compararán con los valores reales (y\_test).

**# Cálculo de Métricas de Evaluación:**

* **Error Cuadrático Medio (MSE)**:  
    
  mse = mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

* + El MSE mide el error promedio entre los valores reales y las predicciones. Un MSE más bajo sugiere que el modelo tiene una precisión alta en promedio.
* **Raíz del Error Cuadrático Medio (RMSE)**:  
    
  rmse = np.sqrt(mse)

print("RMSE:", rmse)

* + La RMSE es la raíz cuadrada del MSE y proporciona una métrica de error en las mismas unidades que los datos, lo cual facilita su interpretación.
* **Coeficiente de Determinación (R²)**:  
    
  r2 = r2\_score(y\_test, predictions)

print("R2:", r2)

* + El R2R^2R2 muestra qué tan bien el modelo explica la variabilidad en los datos de prueba. Un valor de R2R^2R2 cercano a 1 indica un buen ajuste, mientras que un valor bajo sugiere que el modelo podría mejorarse.

**# Gráfica de Predicciones vs. Valores Actuales**:  
  
plt.scatter(y\_test, predictions)

plt.xlabel('Etiquetas actuales')

plt.ylabel('Etiquetas predichas')

plt.title('Predicción de nivel de cuencas')

Esta gráfica de dispersión muestra la relación entre los valores reales (y\_test) en el eje X y los valores predichos (predictions) en el eje Y. Cada punto azul representa una comparación entre un valor real y su correspondiente valor predicho.

**# Sobreponer la Línea de Regresión:**  
  
z = np.polyfit(y\_test, predictions, 1)

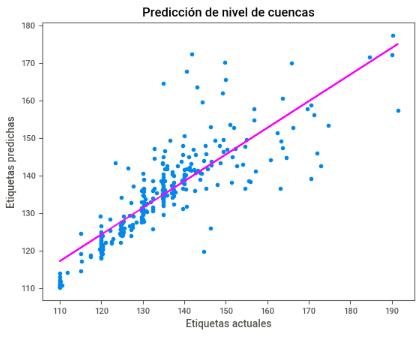
p = np.poly1d(z)

plt.plot(y\_test, p(y\_test), color='magenta')

plt.show()

Esta línea morada representa una regresión lineal entre los valores reales y predichos. Cuanto más cerca están los puntos de esta línea, mejor está funcionando el modelo. Si los puntos se agrupan cerca de la línea diagonal, indica que el modelo es preciso en sus predicciones.

Esta gráfica y las métricas ofrecen una visión clara del rendimiento del modelo de árbol de decisión para predecir el nivel de cuencas.



La gráfica es una dispersión que compara los valores actuales de nivel de cuencas (en el eje X) con los valores predichos (en el eje Y). Algunos puntos clave para el análisis son:

1. **Tendencia positiva**: La línea de tendencia (en color magenta) muestra una relación positiva entre los valores actuales y los predichos. Esto sugiere que, en general, el modelo de predicción tiene una relación lineal directa con los valores reales, lo cual es un buen indicio de precisión.
2. **Distribución de puntos**: Aunque muchos puntos están cercanos a la línea de tendencia, lo que sugiere una buena predicción, también se observan algunos puntos dispersos, especialmente en la parte superior de la gráfica. Esto indica que en ciertos casos, las predicciones no se ajustan exactamente a los valores reales y pueden existir errores de predicción.
3. **Precisión del modelo**: La proximidad de la mayoría de los puntos a la línea de tendencia sugiere que el modelo tiene un desempeño razonablemente bueno. Sin embargo, la dispersión indica que todavía hay margen de mejora, ya que algunas predicciones difieren significativamente de los valores actuales.
4. **Rango de predicción**: Los valores predichos se encuentran en un rango de aproximadamente 110 a 170, similar al rango de los valores actuales. Esto muestra que el modelo está capturando bien el rango de los datos reales.

En resumen, la gráfica sugiere que el modelo tiene un rendimiento aceptable, con una tendencia lineal bien definida. Sin embargo, los puntos dispersos indican que hay variabilidad en la precisión de las predicciones, y mejorar el modelo podría reducir los errores en algunas predicciones.

**# Entrenamiento del modelo Gradient Boosting Regressor**

1. **Entrenamiento del Modelo**:  
     
   model = GradientBoostingRegressor().fit(X\_train, y\_train)

print(model, "\n")

Aquí entrenas el modelo utilizando los datos de entrenamiento X\_train y las etiquetas correspondientes y\_train.

1. **Predicción y Evaluación**:  
     
   predictions = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse = np.sqrt(mse)

print("RMSE:", rmse)

r2 = r2\_score(y\_test, predictions)

print("R2:", r2)

En esta sección, realizas predicciones sobre el conjunto de prueba X\_test y luego calculas tres métricas de evaluación:

* + **MSE (Error Cuadrático Medio)**: Mide la media de los errores al cuadrado, lo que penaliza más los errores grandes.
  + **RMSE (Raíz del Error Cuadrático Medio)**: Es la raíz cuadrada del MSE, proporciona una interpretación en las mismas unidades que las etiquetas originales.
  + **R² (Coeficiente de Determinación)**: Indica qué tan bien se ajusta el modelo a los datos, donde un valor de 1 indica un ajuste perfecto.

1. **Visualización**:  
     
   plt.scatter(y\_test, predictions)

plt.xlabel('Etiquetas actuales')

plt.ylabel('Etiquetas predichas')

plt.title('Predicción de nivel de cuencas')

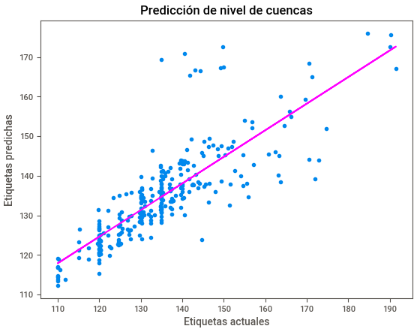
z = np.polyfit(y\_test, predictions, 1)

p = np.poly1d(z)

plt.plot(y\_test, p(y\_test), color='magenta')

plt.show()

Creas un gráfico de dispersión para comparar las etiquetas actuales con las predicciones. La línea de regresión que sobrepones ayuda a visualizar cómo se alinean las predicciones con las etiquetas reales.



### 1. Tendencia Lineal y Correlación

* La línea magenta representa la tendencia de los datos y sugiere una relación lineal positiva entre las etiquetas actuales y las predichas. Esto indica que, en promedio, el modelo predice valores que se alinean con los valores reales.
* La inclinación de la línea sugiere que el modelo es generalmente capaz de captar la relación entre las variables, aunque no es perfecta debido a la dispersión de puntos alrededor de la línea.

### 2. Distribución y Dispersión de los Puntos

* La mayoría de los puntos están cerca de la línea de tendencia, lo que indica que el modelo tiene un buen ajuste en muchos casos. Sin embargo, hay una dispersión notable, especialmente en ciertos rangos de valores.
* En el rango de valores actuales entre 120 y 150, la dispersión es moderada, pero se observan varios puntos que están separados de la línea, lo cual indica que el modelo tiene mayor variabilidad en sus predicciones en este rango.
* En el rango de valores más altos (170-190), aunque el modelo sigue la tendencia, se observa una ligera desviación que sugiere que el modelo podría estar subestimando o sobrestimando ciertos valores.

### 3. Errores de Predicción y Sesgo

* Los puntos que se alejan de la línea magenta representan errores de predicción. La variación en estos puntos sugiere la presencia de error aleatorio, y quizás un pequeño sesgo sistemático en algunos rangos.
* Algunos puntos más altos por encima de la línea pueden indicar sobreestimación del modelo, mientras que los puntos por debajo de la línea pueden indicar subestimación.
* El modelo parece ser menos preciso en el rango medio de valores actuales (aproximadamente entre 130 y 150), donde la dispersión es mayor. Esto podría sugerir que el modelo no captura bien la complejidad de los datos en ese intervalo.

### 4. Rango de Predicciones

* Los valores predichos se extienden aproximadamente entre 110 y 170, mientras que los valores actuales también están en un rango similar, lo que es un buen indicio de que el modelo no realiza predicciones fuera de escala.
* La coherencia del rango entre los valores actuales y predichos indica que el modelo es capaz de capturar el rango de variación de los datos, lo cual es importante para evitar problemas de extrapolación.

### 5. Calidad de las Predicciones

* La alineación de los puntos con la línea sugiere que el modelo es razonablemente bueno para hacer predicciones en la mayoría de los casos, aunque con margen de mejora.
* Los puntos dispersos sugieren que ciertos datos de entrada pueden estar afectando negativamente la precisión del modelo. Este análisis sugiere que se podría trabajar en reducir estos errores, tal vez mediante una selección de características más precisa o ajustes en los hiperparámetros del modelo.

### Conclusión General

* El modelo logra capturar una relación lineal con los datos actuales, lo cual es un buen punto de partida. Sin embargo, la dispersión de los puntos en ciertos rangos indica que el modelo aún no es perfecto y podría mejorarse.
* Podría ser útil aplicar técnicas adicionales, como regularización o aumentar la cantidad de datos de entrenamiento, para reducir la variabilidad y mejorar la precisión del modelo en aquellos rangos donde la predicción es menos precisa.

En resumen, la gráfica muestra un modelo de predicción que en general sigue la tendencia de los datos actuales, pero presenta áreas de dispersión que sugieren errores en las predicciones, especialmente en el rango medio de los valores actuales. Estos insights podrían ayudar en la mejora del modelo.