**import pandas as pd**

**import numpy as np**

**import tkinter as tk**

**from tkinter import messagebox**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LinearRegression**

**import joblib**

**from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score**

**df = pd.read\_csv("PREC\_1981\_2010\_Provincias.csv", sep=";")**

**df\_2019 = pd.read\_csv('PREC\_2019\_Provincias.csv', delimiter=';')**

**df\_2020 = pd.read\_csv('PREC\_2020\_Provincias.csv', delimiter=';')**

**df\_2021 = pd.read\_csv('PREC\_2021\_Provincias.csv', delimiter=';')**

**df\_2022 = pd.read\_csv('PREC\_2022\_Provincias.csv', delimiter=';')**

**print(df.head())**

**print('\n\_\_\_\_\_\_\_\_\_\_\_\_INFORMACION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n')**

**df.info()**

**print('\n\_\_\_\_\_\_\_\_\_\_\_\_VALORES FALTANTES\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n')**

**missing\_values = df.isna().sum()**

**print(missing\_values)**

**print('\n\_\_\_\_\_\_\_\_\_\_\_\_DESCRIPCION\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n')**

**df.describe()**

**Importación de Bibliotecas y Carga de Datos**

Para llevar a cabo el proyecto, utilizo varias bibliotecas de Python que proporcionan herramientas poderosas para el análisis de datos y la creación de una interfaz de usuario. Algunas de las bibliotecas como:

* **Pandas**: Esta biblioteca es fundamental para trabajar con datos estructurados. Nos permite cargar y manipular fácilmente conjuntos de datos en formato de tabla, similar a una hoja de cálculo.
* **Numpy**: Numpy es una biblioteca que nos ayuda a realizar cálculos numéricos eficientes. La utilizamos en conjunto con Pandas para operaciones matemáticas.
* **Tkinter**: Tkinter es una biblioteca que nos permite crear una interfaz gráfica de usuario (GUI). La interfaz gráfica hace que nuestro proyecto sea más accesible y fácil de usar.
* **Seaborn y Matplotlib**: Estas bibliotecas se encargan de la visualización de datos. Nos permiten crear gráficos y visualizaciones para comprender mejor nuestros datos.
* **Scikit-Learn (sklearn)**: Scikit-Learn es una biblioteca que incluye herramientas para el aprendizaje automático y la minería de datos. En particular, utilizo el módulo LinearRegression para realizar regresión lineal.
* **Joblib**: Esta biblioteca se utiliza para guardar y cargar modelos de aprendizaje automático. Esto es esencial para preservar los modelos una vez entrenados.

**Carga de Datos**

Luego de importar estas bibliotecas, sigo cargando los datos que serán el núcleo del proyecto. Utilizo archivos CSV que contienen información sobre las precipitaciones en diferentes provincias. Estos archivos se dividen en varios años, lo que nos permitirá analizar tendencias a lo largo del tiempo.

Para tener una vista previa inicial de los datos, imprimo las primeras filas del conjunto de datos principal utilizando **df.head()**. También obtengo información sobre la estructura de los datos con **df.info()** y reviso si hay valores faltantes en cada columna con **missing\_values**. Por último, realizo un análisis descriptivo de los datos mediante **df.describe()**.

Esta fase inicial es esencial para entender la naturaleza de los datos y preparo para las etapas posteriores de procesamiento y análisis.

df\_concatenado = pd.concat([df, df\_2019, df\_2020, df\_2021, df\_2022], ignore\_index=True)

X = df\_concatenado[['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio', 'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre']]

y = df\_concatenado['anual']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

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

joblib.dump(model, 'modelo\_lluvia')

y\_pred = model.predict(X\_test)

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

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

mae = mean\_absolute\_error(y\_test, y\_pred)

print("Error cuadrático medio (MSE):", mse)

print("Coeficiente de determinación (R²):", r2)

print("Coeficiente de error absoluto (MAE):", mae)

**Concatenar los DataFrames**.

Primero, concateno varios DataFrames en uno solo. Los DataFrames contienen datos de diferentes años (2019, 2020, 2021, 2022) que se han combinado en un solo conjunto de datos para un análisis más amplio.

**Definir Características y Objetivo**.

Luego, selecciono las características (X) y el objetivo (y) para el modelo. Las características son los datos mensuales de precipitación (enero, febrero, ..., diciembre), y el objetivo es la precipitación anual (anual).

**Dividir el Conjunto de Datos**.

Divido el conjunto de datos en conjuntos de entrenamiento (X\_train, y\_train) y prueba (X\_test, y\_test) utilizando la función **train\_test\_split**. Esto es fundamental para evaluar el rendimiento del modelo.

**Crear un Modelo de Regresión Lineal**.

Utilizo un modelo de regresión lineal de la biblioteca scikit-learn (**LinearRegression**) y lo entreno con los datos de entrenamiento (X\_train, y\_train).

**Guardar el Modelo**.

El modelo entrenado se guarda en un archivo llamado 'modelo\_lluvia' utilizando la función **joblib.dump**. Esto permitirá cargar y utilizar el modelo en el futuro sin necesidad de volver a entrenarlo.

**Realizar Predicciones**.

Uso el modelo entrenado para hacer predicciones sobre los datos de prueba (X\_test).

**Calcular Métricas de Evaluación**.

Calculo tres métricas importantes para evaluar el rendimiento del modelo:

* + **Error Cuadrático Medio (MSE)**: Mide la calidad de las predicciones, siendo más bajo mejor.
  + **Coeficiente de Determinación (R²)**: Proporciona información sobre cuánta variabilidad en el objetivo se explica por el modelo.
  + **Error Absoluto Medio (MAE)**: Calcula el promedio de las diferencias absolutas entre las predicciones y los valores reales.

Finalmente, imprimo las métricas de evaluación. Estas métricas ayudarán a comprender qué tan bien se desempeña el modelo de regresión lineal en la predicción de la precipitación anual.

datos\_cliente = pd.DataFrame(columns=['provincia'] + ['mes\_' + str(i) for i in range(1, 13)])

def calcular\_meses(lluvia\_mensual):

meses = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre']

meses\_con\_lluvia = [(mes, lluvia) for mes, lluvia in zip(meses, lluvia\_mensual)]

meses\_con\_lluvia.sort(key=lambda x: x[1], reverse=True)

meses\_lluviosos = [mes for mes, lluvia in meses\_con\_lluvia if lluvia > sum(lluvia\_mensual) / len(lluvia\_mensual)]

meses\_menos\_lluviosos = [mes for mes, lluvia in meses\_con\_lluvia if lluvia < sum(lluvia\_mensual) / len(lluvia\_mensual)]

return meses\_lluviosos, meses\_menos\_lluviosos

def calcular\_dato\_anual(lluvia\_mensual, modelo):

dato\_anual = modelo.predict([lluvia\_mensual])[0]

return dato\_anual

para guardar los datos en un archivo CSV

def guardar\_csv(provincia, lluvia\_mensual):

df = pd.DataFrame({

'mes': ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre'],

'precipitacion': lluvia\_mensual

})

df.to\_csv(f'{provincia}.csv', index=False)

def guardar\_csv(provincia, lluvia\_mensual):

df = pd.DataFrame({

'mes': ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre'],

'precipitacion': lluvia\_mensual

})

df.to\_csv(f'{provincia}.csv', index=False)

def mostrar\_resultados\_provincia():

try:

provincia = provincia\_entry.get()

lluvia\_mensual = [float(entry.get()) for entry in entradas\_lluvia]

datos = [provincia] + lluvia\_mensual

datos\_cliente.loc[len(datos\_cliente)] = datos

meses\_lluviosos, meses\_menos\_lluviosos = calcular\_meses(lluvia\_mensual)

dato\_anual = calcular\_dato\_anual(lluvia\_mensual, model)

mes\_maxima = meses\_lluviosos[0]

mes\_minima = meses\_menos\_lluviosos[-1]

precipitacion\_anual = sum(lluvia\_mensual)

guardar\_csv(provincia, lluvia\_mensual)

nota\_agricultura = ""

nota\_ganaderia = ""

if precipitacion\_anual > 1000:

nota\_agricultura = "Con una precipitación anual saludable, la agricultura puede prosperar."

elif 800 <= precipitacion\_anual <= 1000:

nota\_agricultura = "Las condiciones son ideales para el cultivo de cultivos de alto rendimiento."

elif 600 <= precipitacion\_anual < 800:

nota\_agricultura = "La precipitación anual es adecuada para la mayoría de los cultivos comunes."

elif 400 <= precipitacion\_anual < 600:

nota\_agricultura = "La precipitación anual podría requerir prácticas de conservación de agua para la agricultura."

elif precipitacion\_anual < 400:

nota\_agricultura = "La baja precipitación anual podría requerir riego constante y prácticas de conservación de agua."

else:

nota\_agricultura = "No se pudo determinar una nota precisa debido a la precipitación anual."

if precipitacion\_anual > 1000:

nota\_ganaderia = "Las condiciones son óptimas para el pastoreo y la ganadería."

elif 800 <= precipitacion\_anual <= 1000:

nota\_ganaderia = "Las condiciones de pasto son favorables para el ganado."

elif 600 <= precipitacion\_anual < 800:

nota\_ganaderia = "Las condiciones de pasto son adecuadas para la mayoría del ganado."

elif 400 <= precipitacion\_anual < 600:

nota\_ganaderia = "La disponibilidad de pasto puede ser un desafío; considera suplementos de alimentación."

elif precipitacion\_anual < 400:

nota\_ganaderia = "La falta de pasto puede ser un desafío importante; considera prácticas de pastoreo intensivo."

else:

nota\_ganaderia = "No se pudo determinar una nota precisa debido a la precipitación anual."

ventana\_resultados = tk.Toplevel()

ventana\_resultados.title("Resultados Detallados")

resultado\_label = tk.Label(ventana\_resultados, text=f"Provincia: {provincia}", font=("Arial", 14))

resultado\_label.pack()

resultado\_meses\_lluviosos = tk.Label(ventana\_resultados, text=f"Meses más lluviosos: {', '.join(meses\_lluviosos)}", font=("Arial", 12))

resultado\_meses\_lluviosos.pack()

resultado\_meses\_menos\_lluviosos = tk.Label(ventana\_resultados, text=f"Meses menos lluviosos: {', '.join(meses\_menos\_lluviosos)}", font=("Arial", 12))

resultado\_meses\_menos\_lluviosos.pack()

resultado\_dato\_anual = tk.Label(ventana\_resultados, text=f"Dato anual (predicción): {dato\_anual:.2f} mm", font=("Arial", 12))

resultado\_dato\_anual.pack()

resultado\_mes\_maxima = tk.Label(ventana\_resultados, text=f"Mes con máxima lluvia: {mes\_maxima}", font=("Arial", 12))

resultado\_mes\_maxima.pack()

resultado\_mes\_minima = tk.Label(ventana\_resultados, text=f"Mes con mínima lluvia: {mes\_minima}", font=("Arial", 12))

resultado\_mes\_minima.pack()

resultado\_precipitacion\_anual = tk.Label(ventana\_resultados, text=f"Precipitación anual promedio: {precipitacion\_anual:.2f} mm", font=("Arial", 12))

resultado\_precipitacion\_anual.pack()

resultado\_nota\_agricultura = tk.Label(ventana\_resultados, text=f"Nota sobre Agricultura: {nota\_agricultura}", font=("Arial", 12))

resultado\_nota\_agricultura.pack()

resultado\_nota\_ganaderia = tk.Label(ventana\_resultados, text=f"Nota sobre Ganadería: {nota\_ganaderia}", font=("Arial", 12))

resultado\_nota\_ganaderia.pack()

except ValueError:

messagebox.showerror("Error", "Asegúrate de ingresar números válidos en todos los campos.")

ventana = tk.Tk()

ventana.title("Panel de Precipitaciones")

ventana.geometry("1000x600") # Establecer tamaño de la ventana

ventana.configure(bg="lightblue") # Establecer color de fondo

fondo = tk.PhotoImage(file="imagen\_python\_01.png")

fondo = fondo.subsample(5) # Ajustar el tamaño (2 es un factor de escala)

fondo\_label = tk.Label(ventana, image=fondo)

fondo\_label.place(relwidth=1, relheight=1) # Ajustar la imagen al tamaño de la ventana

provincia\_label = tk.Label(ventana, text="Nombre de la Provincia:", font=("Arial", 14),fg="green")

provincia\_label.grid(row=0, column=0, padx=10, pady=10)

provincia\_entry = tk.Entry(ventana, font=("Arial", 12))

provincia\_entry.grid(row=0, column=1, padx=10, pady=10)

entradas\_lluvia = []

for i, mes in enumerate(['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre']):

mes\_label = tk.Label(ventana, text=mes, font=("Arial", 14))

mes\_label.grid(row=i+1, column=0, padx=10, pady=10)

mes\_entry = tk.Entry(ventana, font=("Arial", 12))

mes\_entry.grid(row=i+1, column=1, padx=10, pady=10)

entradas\_lluvia.append(mes\_entry)

calcular\_button = tk.Button(ventana, text="Calcular", font=("Arial", 14), command=mostrar\_resultados\_provincia)

calcular\_button.grid(row=14, column=0, columnspan=2, pady=20)

resultados\_text\_provincia = tk.Label(ventana, text="", font=("Arial", 12), bg="lightgray")

resultados\_text\_provincia.grid(row=15, column=0, columnspan=2, padx=10, pady=10)

ventana.mainloop()

Ahota creo una interfaz gráfica de usuario (GUI) para que los usuarios ingresen datos de precipitación mensual y obtengan información detallada sobre la provincia en cuestión.

Pasos a seguir:

**Creación del DataFrame vacío**: Se crea un DataFrame llamado **datos\_cliente** con columnas para el nombre de la provincia y los datos de precipitación mensual.

**Funciones para el cálculo de datos**:

* + **calcular\_meses**: Esta función determina los meses más lluviosos y menos lluviosos en función de la precipitación mensual.
  + **calcular\_dato\_anual**: Calcula el dato anual (predicción) utilizando un modelo. Se debe reemplazar **‘modelo'** con tu modelo real.
  + **guardar\_csv**: Guarda los datos de precipitación mensual en un archivo CSV con el nombre de la provincia.

**Función mostrar\_resultados\_provincia**: Esta función se activa cuando el usuario hace clic en el botón "Calcular". Realiza lo siguiente:

* + Obtiene el nombre de la provincia y los datos mensuales de precipitación ingresados por el usuario.
  + Calcula los meses más lluviosos y menos lluviosos.
  + Calcula el dato anual utilizando el modelo de regresión lineal (se recomienda reemplazar **'modelo'** con tu modelo real).
  + Determina el mes con la máxima y mínima lluvia.
  + Calcula la precipitación anual promedio.
  + Guarda los datos en un archivo CSV.
  + Genera notas sobre agricultura y ganadería basadas en la precipitación anual.
  + Crea una nueva ventana que muestra todos estos resultados.

**Creación de la ventana principal (GUI)**: Se crea una ventana principal de la GUI utilizando Tkinter. Se establece un fondo de imagen y se agregan etiquetas y campos de entrada para el nombre de la provincia y los datos mensuales de precipitación. Además, se agrega un botón "Calcular" que, cuando se pulsa, activa la función **mostrar\_resultados\_provincia**.

**Etiquetas y campos de entrada para los datos mensuales**: Se generan etiquetas y campos de entrada para que el usuario ingrese los datos mensuales de precipitación para los 12 meses.

**Botón "Calcular"**: Este botón permite al usuario calcular y mostrar los resultados para la provincia ingresada.

**Etiqueta para mostrar los resultados**: Después de calcular los resultados, se utiliza una etiqueta para mostrar los resultados detallados de la provincia.

La interfaz permite a los usuarios ingresar datos, calcular información relevante y presentarla de una manera amigable. Puedes personalizar y adaptar este código para tus necesidades específicas, como reemplazar el modelo de predicción con uno real y ajustar las notas sobre agricultura y ganadería según tus criterios.

import csv

from unidecode import unidecode

data = []

file\_names = ["PREC\_1981\_2010\_Provincias.csv", "PREC\_2019\_Provincias.csv", "PREC\_2020\_Provincias.csv","PREC\_2021\_Provincias.csv","PREC\_2022\_Provincias.csv"] # Reemplaza con los nombres reales de tus archivos

ciudad\_concreta = input('Introduce una ciudad española para conocer sus precipitaciones anuales a lo largo de los años: ')

ciudad\_concreta = unidecode(ciudad\_concreta).upper()

for file\_name in file\_names:

with open(file\_name, 'r', newline='') as file:

csv\_reader = csv.DictReader(file, delimiter=';')

for row in csv\_reader:

if row['region'] == ciudad\_concreta:

anual\_value = float(row['anual'].replace(',', '.')) # Algunos CSV usan ',' en lugar de '.' para números decimales

data.append(anual\_value)

print(f"Valores anuales de {ciudad\_concreta}:")

print(data)

print(f'''

Precipitacion anual 1981\_2010 ---> {data[0]}

Precipitacion anual 2019 ---> {data[1]}

Precipitacion anual 2020 ---> {data[2]}

Precipitacion anual 2021 ---> {data[3]}

Precipitacion anual 2022 ---> {data[4]}

''')

Éste código está diseñado para leer datos de archivos CSV que contienen información sobre precipitaciones anuales por regiones, y luego buscar y mostrar los valores anuales para una ciudad específica ingresada por el usuario. Además, usa la biblioteca **unidecode** para convertir el nombre de la ciudad a mayúsculas y eliminar acentos, lo que es útil para la búsqueda.

Aquí hay un resumen de lo que hace el código:

* + Importa las bibliotecas necesarias: **csv** para trabajar con archivos CSV y **unidecode** para eliminar acentos y convertir el texto a mayúsculas.
  + Crea una lista llamada **data** para almacenar los valores de precipitación anual encontrados en los archivos CSV.
  + Define una lista de nombres de archivos CSV que se deben leer. Hay que asegurarse de que los nombres de los archivos coincidan con los archivos reales en tu sistema.
  + Solicita al usuario que ingrese el nombre de una ciudad española. Luego, aplica **unidecode** y convierte el texto ingresado a mayúsculas para la búsqueda.
  + Utiliza un bucle **for** para recorrer la lista de nombres de archivos CSV y leer cada archivo. Dentro del bucle, se busca la ciudad ingresada por el usuario en la columna 'region' de los archivos CSV. Si se encuentra una coincidencia, se extrae el valor anual de precipitación, se convierte de una cadena (que puede usar ',' como decimal) a un valor decimal y se agrega a la lista **data**.
  + Imprime los valores anuales de precipitación para la ciudad concreta, así como los valores específicos para cada año.

Teniendo en cuenta que el código asume que los archivos CSV tienen una estructura específica con columnas llamadas 'region' y 'anual'. Hay que asegurarse de que los archivos CSV coincidan con esta estructura o ajusta el código según las necesidades.

Principio del formulario