COMANDOS

# Instalar la extensión de Python

# Para crear un virtual enviroment en VS Code tenemos que abrir la terminal y escribir el siguiente comando:

#   python3 -m venv nombre

# Se creará una carpeta llamada "nombre" con Python 3 como lenguaje.

# O hacer: Ctrl + Shift + P, escribir Python y buscar la opción de Crear Virtual Enviroment y seleccionar un interperetador de Python.

# Comandos:

# Cambiar de directorio:

#   cd "nombre de directorio"

# Ir a la carpeta anterior:

#   cd ..

# Ejecutar streamlit:

#   streamlit run "nombre\_archivo"

#   python -m streamlit run "nombre\_archivo"

# Detener ejecución:

#   Ctrl + C

# Para crear requirements.txt usamos la librería freeze

#   pip install freeze-requirements

#   pip freeze > requirements.txt

# Para actualizar una librería a su última versión:

#   pip install -U "librería"

TEXT WRITE

import streamlit as st

import numpy as np

import pandas as pd

import requests

def main():

    # Title

    st.title("Hello Streamlit App.")

    # Text

    # st.text("Hello World!")

    # st.text("New Line.")

    # name = "Daniel"

    # st.text("Siu.")

    # st.text(f"Hola soy {name}")

    # print("hola soy daniel")

    # # Header

    # st.header("Header1")

    # # Subheader

    # st.subheader("Subheader1")

    # # Markdown

    # st.markdown("# This is markdown.")

    # # Display Colored Text/Boostraps Alert

    # st.success("Siuuuuu.")

    # st.warning("This is not a Siuuuuuu.")

    # st.info("Info about the Siuuuuuu.")

    # st.error("This is a error Siuuuuuu.")

    # st.exception("This is a Siuuuuu exception.")

    # .write()

    # st.write("Normal Text.")

    # st.write("## This is a markdown text")

    # st.write(1 + 2)

    # st.dataframe(dir(st))

    # # Help

    # st.help(range)

    # Display Data

    # df = pd.read\_csv(filepath\_or\_buffer = "sources/AccidentesBicicletas\_2021.csv", sep = ";")

    # Dinamic Data

    # st.dataframe(df)

    # st.write(df)

    # Static Table

    # st.table(df)

    # Adding Color

    # st.dataframe(df.select\_dtypes(include=np.number).style.highlight\_max(axis = 0))

    # Display JSON

    endpoint = "https://api.frankfurter.app/latest"

    response = requests.get(url = endpoint)

    st.json(response.json())

    # # Display Code

    # code = """

    # def func():

    #     return x\*\*2

    # """

    # st.code(body = code, language = "python")

    pass

if \_\_name\_\_ == "\_\_main\_\_":

    main()

IMAGENES

import streamlit as st

from PIL import Image

# Local

# image = Image.open("sources/curiosidades-del-oso-panda-1280x720x80xX.jpg")

# st.image(image            = image,

#          caption          = "pandas2",

#          use\_column\_width = False)

# st.image(image           = image,

#         caption          = "pandas",

#         use\_column\_width = True)

# # URL

# st.image(image = "data:image/jpeg;base64,",

#          caption = "image from url",

#          use\_column\_width = False)

# st.image(image = "data:image/jpeg;base64,",

#          caption = "image from url2",

#          use\_column\_width = True)

BUTTONS SELECT

import streamlit as st

import numpy as np

import pandas as pd

def main():

    # # Buttons

    # name = "Daniel"

    # if st.button(label = "Submit", key = "submit1"):

    #     st.write(f"{name.upper()}")

    # if st.button(label = "Submit", key = "submit2", type = "primary"):

    #     st.write(f"First Name: {name.title()}")

    # # Radio Buttons

    # status = st.radio(label = "What is your status?",

    #                   options = ("None", "Active", "Inactive"),

    #                   index = 0,

    #                   disabled = False,

    #                   horizontal = True,

    #                   )

    # if status == "Active":

    #     st.success("You are Active.")

    # elif status == "Inactive":

    #     st.warning("You are Inactive.")

    # # CheckBox

    # if st.checkbox(label = "Show/Hide"):

    #     st.text("Showing Something")

    # # Expander

    # with st.expander(label = "DataFrame", expanded = False):

    #     st.dataframe(pd.read\_csv("sources/AccidentesBicicletas\_2021.csv", sep = ";"))

    # # SelectBox

    # modulos = ["Python", "Matemáticas", "Data Science", "SQL", "ML", "Big Data", "Streamlit"]

    # choice = st.selectbox(label = "Modulo", options = modulos)

    # st.write(f"Modulo: {choice}")

    # # Multiple Selection

    # librerias = ["numpy", "pandas", "random", "datetime", "sklearn"]

    # libreria = st.multiselect(label = "Librerias",

    #                           options = librerias,

    #                           default = librerias,

    #                           placeholder = "Placeholder",

    #                          )

    # st.write(libreria)

    # # Slider Int, Float, Date

    # age = st.slider(label     = "Age",

    #                 min\_value = 1,

    #                 max\_value = 100,

    #                 value     = 50,

    #                 step      = 2)

    # age = st.slider(label     = "Age",

    #                 min\_value = 1.0,

    #                 max\_value = 100.0,

    #                 value     = 50.0,

    #                 step      = 0.01)

    # # Select Slider

    # colores = ["Amarillo", "Azul", "Rojo", "Morado", "Verde"]

    # color = st.select\_slider(label  = "Choose Color",

    #                          options = colores,

    #                          value = "Morado")

    pass

if \_\_name\_\_ == "\_\_main\_\_":

    main()

USER IMPUT

import streamlit as st

import pandas as pd

# pip instal lxml

import folium

from modules.weather\_func import \*

# pip install streamlit\_folium

from streamlit\_folium import st\_folium

def main():

    st.set\_page\_config(page\_title = "Spain Weather - Streamlit",

                       page\_icon  = ":cloud:",

                       layout     = "centered") # wide

    # Title

    st.title("OpenWeatherAPI & Streamlit :face\_in\_clouds:")

    st.markdown("Spain's current weather by capital.")

    # Weather Data

    df = get\_weather\_data()

    # DataFrame

    with st.expander(label = "Open Weather DataFrame", expanded = False):

        st.dataframe(df)

        st.write(f"DataFrame dimensions: {df.shape[0]}x{df.shape[1]}")

        st.balloons()

    # Mapa

    center = st.selectbox(label   = "Select capital:",

                          options = df["capital"].values,

                          index   = 26)

    map = folium.Map(location   =  df[df["capital"] == "Zamora"][["lat", "lon"]].values,

                     zoom\_start = 5)

    # Data

    data\_columns = ["capital", "temp", "lat", "lon"]

    points = folium.map.FeatureGroup()

    for capital, temp, lat, lng, in df[data\_columns].values:

        display\_info = f"Capital: {capital}\nTemp: {temp}"

        points.add\_child(folium.Marker(location = [lat, lng],

                                       popup    = display\_info))

    map.add\_child(points)

    st\_folium(fig = map, width = 1000)

    df\_center = df[df["capital"] == center]

    st.write(f"Current weather in {center} :umbrella\_with\_rain\_drops: :")

    st.write(df\_center.iloc[0, :-2].to\_dict())

if \_\_name\_\_ == "\_\_main\_\_":

    main()

PLOTS

import streamlit as st

import pandas as pd

# pip instal lxml

import folium

from modules.weather\_func import \*

# pip install streamlit\_folium

from streamlit\_folium import st\_folium

def main():

    st.set\_page\_config(page\_title = "Spain Weather - Streamlit",

                       page\_icon  = ":cloud:",

                       layout     = "centered") # wide

    # Title

    st.title("OpenWeatherAPI & Streamlit :face\_in\_clouds:")

    st.markdown("Spain's current weather by capital.")

    # Weather Data

    df = get\_weather\_data()

    # DataFrame

    with st.expander(label = "Open Weather DataFrame", expanded = False):

        st.dataframe(df)

        st.write(f"DataFrame dimensions: {df.shape[0]}x{df.shape[1]}")

        st.balloons()

    # Mapa

    center = st.selectbox(label   = "Select capital:",

                          options = df["capital"].values,

                          index   = 26)

    map = folium.Map(location   =  df[df["capital"] == "Zamora"][["lat", "lon"]].values,

                     zoom\_start = 5)

    # Data

    data\_columns = ["capital", "temp", "lat", "lon"]

    points = folium.map.FeatureGroup()

    for capital, temp, lat, lng, in df[data\_columns].values:

        display\_info = f"Capital: {capital}\nTemp: {temp}"

        points.add\_child(folium.Marker(location = [lat, lng],

                                       popup    = display\_info))

    map.add\_child(points)

    st\_folium(fig = map, width = 1000)

    df\_center = df[df["capital"] == center]

    st.write(f"Current weather in {center} :umbrella\_with\_rain\_drops: :")

    st.write(df\_center.iloc[0, :-2].to\_dict())

if \_\_name\_\_ == "\_\_main\_\_":

    main()

CONF PAGE

import streamlit as st

from PIL import Image

from modules.page\_config\_dict import \*

st.set\_page\_config(page\_title = "DSB03RT",

                   page\_icon = ":panda\_face:",

                #    page\_icon = Image.open("sources/curiosidades-del-oso-panda-1280x720x80xX.jpg"),

                   layout = "wide",

                   initial\_sidebar\_state = "collapsed",)

# st.set\_page\_config(\*\*PAGE\_CONFIG)

# st.json(PAGE\_CONFIG)

# Emoji: https://streamlit-emoji-shortcodes-streamlit-app-gwckff.streamlit.app/

def main():

    st.title("Streamlit App.")

    st.sidebar.success("Hello")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

FOLIUM

import streamlit as st

import pandas as pd

# pip instal lxml

import folium

from modules.weather\_func import \*

# pip install streamlit\_folium

from streamlit\_folium import st\_folium

def main():

    st.set\_page\_config(page\_title = "Spain Weather - Streamlit",

                       page\_icon  = ":cloud:",

                       layout     = "centered") # wide

    # Title

    st.title("OpenWeatherAPI & Streamlit :face\_in\_clouds:")

    st.markdown("Spain's current weather by capital.")

    # Weather Data

    df = get\_weather\_data()

    # DataFrame

    with st.expander(label = "Open Weather DataFrame", expanded = False):

        st.dataframe(df)

        st.write(f"DataFrame dimensions: {df.shape[0]}x{df.shape[1]}")

        st.balloons()

    # Mapa

    center = st.selectbox(label   = "Select capital:",

                          options = df["capital"].values,

                          index   = 26)

    map = folium.Map(location   =  df[df["capital"] == "Zamora"][["lat", "lon"]].values,

                     zoom\_start = 5)

    # Data

    data\_columns = ["capital", "temp", "lat", "lon"]

    points = folium.map.FeatureGroup()

    for capital, temp, lat, lng, in df[data\_columns].values:

        display\_info = f"Capital: {capital}\nTemp: {temp}"

        points.add\_child(folium.Marker(location = [lat, lng],

                                       popup    = display\_info))

    map.add\_child(points)

    st\_folium(fig = map, width = 1000)

    df\_center = df[df["capital"] == center]

    st.write(f"Current weather in {center} :umbrella\_with\_rain\_drops: :")

    st.write(df\_center.iloc[0, :-2].to\_dict())

if \_\_name\_\_ == "\_\_main\_\_":

    main()

PLOTLY

import streamlit as st

import numpy as np

import pandas as pd

import seaborn as sns

from datetime import datetime

import plotly.express as px

from modules.page\_config\_dict import PAGE\_CONFIG

st.set\_page\_config(\*\*PAGE\_CONFIG)

def main():

    # st.title("Streamlit App. & Plotly.")

    # with st.expander(label = "DataFrame - Accidentes Bicicletas 2021", expanded = False):

    #     df = pd.read\_csv(filepath\_or\_buffer = "sources/AccidentesBicicletas\_2021.csv", sep = ";")

    #     st.dataframe(df)

    # with st.expander(label = "DataFrame - Accidentes Bicicletas 2021 GroupBy Distrito", expanded = False):

    #     df1 = df.groupby(by = "distrito", as\_index = False)["num\_expediente"].count()

    #     st.dataframe(data = df1, use\_container\_width = True)

    # fig\_pie = px.pie(data\_frame = df1,

    #                  names      = "distrito",

    #                  values     = "num\_expediente",

    #                  title      = "Num. Accidentes por Distrito")

    # st.plotly\_chart(figure\_or\_data = fig\_pie, use\_container\_width = True)

    # fig\_bar = px.bar(data\_frame = df1,

    #                  x          = "distrito",

    #                  y          = "num\_expediente",

    #                  title      = "Num. Accidentes por Distrito")

    # st.plotly\_chart(figure\_or\_data = fig\_bar, use\_container\_width = True)

    # Streamlit Plots

    # with st.expander(label = "DataFrame - Taxis", expanded = False):

    #     df2 = sns.load\_dataset("taxis")

    #     df2 = df2.dropna().iloc[:50, :]

    #     df2 = df2.sort\_values("pickup")

    #     st.dataframe(df2)

    # # Bar Chart

    # st.bar\_chart(df2[["total", "fare"]])

    # # Line Chart

    # columns\_list = df2.\_get\_numeric\_data().columns

    # columns\_choices = st.multiselect(label = "Choose Columns", options = columns\_list, default = columns\_list[-1])

    # df2\_filter = df2[columns\_choices]

    # st.line\_chart(df2\_filter)

    # # Area Chart

    # st.area\_chart(df2\_filter)

    pass

if \_\_name\_\_ == "\_\_main\_\_":

    main()

FRANKFURTER

import streamlit as st

import plotly.express as px

import modules.frankfurter\_funcs as ffunc

from modules.currencies import currencies

from datetime import datetime

def main():

    st.title("Frankfurter API App.")

    st.markdown("""Simple script that calls the `Frankfurter API` and shows the EUR's evolution compared

                   to another currency over a selected year.""")

    currency = st.selectbox(label = "Select Currency", options = currencies)

    year = st.slider(label     = "Select Year",

                     min\_value = 2015,

                     max\_value = datetime.now().year,

                     value     = datetime.now().year - 1)

    currency = currency.split(" - ")[0]

    df = ffunc.currency\_evolution(currency = currency, year = year)

    df.rename(mapper = {"currency" : currency}, axis = 1, inplace = True)

    with st.expander(label = "Frankfurter DataFrame", expanded = False):

        st.dataframe(df)

    df["month"] = df["date"].apply(lambda x : x.strftime("%B"))

    # Line Chart

    line\_fig = px.line(data\_frame = df,

                       x          = "date",

                       y          = currency,

                       title      = f"{currency} - EUR Relationship")

    # Lineas (Mean, Median, Max, Min)

    line\_fig.add\_hline(y = df[currency].mean(),

                       line\_dash = "dash",

                       line\_color = "blue",

                       line\_width = 1,

                       annotation\_position = "top left",

                       annotation\_text = f"mean {df[currency].mean().round(2)}")

    line\_fig.add\_hline(y = df[currency].median(),

                       line\_dash = "dash",

                       line\_color = "yellow",

                       line\_width = 1,

                       annotation\_position = "bottom right",

                       annotation\_text = f"median {df[currency].median().round(2)}")

    line\_fig.add\_hline(y = df[currency].max(),

                       line\_dash = "dash",

                       line\_color = "green",

                       line\_width = 1,

                       annotation\_position = "bottom right",

                       annotation\_text = f"max {round(df[currency].max(), 2)}")

    line\_fig.add\_hline(y = df[currency].min(),

                       line\_dash = "dash",

                       line\_color = "red",

                       line\_width = 1,

                       annotation\_position = "bottom right",

                       annotation\_text = f"min {round(df[currency].min(), 2)}")

    hist\_fig = px.histogram(data\_frame = df, x = currency,  nbins = 50, opacity = 0.8)

    box\_fig = px.box(data\_frame = df, x = currency)

    # Plots

    st.plotly\_chart(line\_fig)

    st.plotly\_chart(hist\_fig)

    st.plotly\_chart(box\_fig)

    # Color

    line\_fig2 = px.line(data\_frame = df,

                       x           = "date",

                       y           = currency,

                       color       = "month",

                       title       = f"{currency} - EUR Relationship")

    hist\_fig2 = px.histogram(data\_frame = df, x = currency, color = "month",  nbins = 50, opacity = 0.8)

    box\_fig2 = px.box(data\_frame = df, x = currency, color = "month")

    # Plots 2

    st.plotly\_chart(line\_fig2)

    st.plotly\_chart(hist\_fig2)

    st.plotly\_chart(box\_fig2)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Ejercicio 1

import streamlit as st

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler# Configuración de la app

st.title(":lupa: K-Means Clustering Interactivo")

st.write("Este ejercicio permite explorar cómo el algoritmo K-Means agrupa datos en clusters.")# Generar un conjunto de datos fijo

np.random.seed(42)

X = np.random.rand(100, 2) \* 10 # 100 puntos en un espacio 2D# Parámetro interactivo: número de clusters

n\_clusters = st.slider("Selecciona el número de clusters", min\_value=2, max\_value=10, value=3)# Escalar los datos para mejorar la precisión

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)# Aplicar K-Means

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42, n\_init=10)

kmeans.fit(X\_scaled)

labels = kmeans.labels\_

centroids = kmeans.cluster\_centers\_# Convertir a coordenadas originales

centroids\_original = scaler.inverse\_transform(centroids)# Visualización

st.subheader(":gráfico\_de\_barras: Visualización de Clusters")

fig, ax = plt.subplots()

scatter = ax.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', alpha=0.7)

ax.scatter(centroids\_original[:, 0], centroids\_original[:, 1], c='red', marker='X', s=200, label='Centroides')

ax.set\_xlabel("Característica 1")

ax.set\_ylabel("Característica 2")

ax.set\_title("Clusters generados por K-Means")

ax.legend()

st.pyplot(fig)# Métricas del modelo

st.subheader(":gráfico\_con\_tendencia\_ascendente: Métricas del Modelo")

st.write(f":diamante\_azul\_pequeño: Inercia (suma de distancias al centroide): {kmeans.inertia\_:.2f}")

st.write(f":diamante\_azul\_pequeño: Número de iteraciones realizadas: {kmeans.n\_iter\_}")# Explicación del algoritmo

st.subheader(":libro: Explicación")

st.write("El algoritmo K-Means agrupa los datos en clusters minimizando la distancia entre los puntos y sus centroides.")

st.write(":diamante\_azul\_pequeño: \*\*Inercia\*\*: mide la compacidad de los clusters (menor es mejor, pero sin sobreajustar).")

st.write(":diamante\_azul\_pequeño: \*\*Centroides\*\*: son los puntos centrales de cada cluster.")

st.write(":diamante\_azul\_pequeño: \*\*Número de clusters\*\*: puedes cambiarlo con el slider y ver cómo afectan los agrupamientos.")

#ejercicio 1

%%writefile app\_fase1.py

import streamlit as st

import pandas as pd# Título de la aplicación

st.title(":gráfico\_de\_barras: Análisis de Ventas")# Cargar el archivo CSV

archivo = st.file\_uploader(":carpeta\_abierta: Sube el archivo CSV", type=["csv"])if archivo:

# Leer el archivo CSV

df = pd.read\_csv(archivo) # Mostrar los primeros registros del DataFrame

st.write(":lupa: Vista previa de los datos:")

st.dataframe(df.head()) # Muestra las primeras filas en formato interactivo # Visualización: Gráfico de barras de cantidad vendida por producto

st.write(":gráfico\_con\_tendencia\_ascendente: Cantidad total de productos vendidos:")

ventas\_por\_producto = df.groupby("Producto")["Cantidad"].sum()

st.bar\_chart(ventas\_por\_producto) # Mostrar estadísticas descriptivas

st.write(":gráfico\_de\_barras: Estadísticas descriptivas:")

st.write(df.describe())

#ejercicio 2

%%writefile final.pyimport streamlit as st

import pandas as pd

import matplotlib.pyplot as plt

import io# :estrella2: Configuración de la aplicación

st.set\_page\_config(page\_title=":gráfico\_de\_barras: Análisis de Ventas", layout="wide")

st.title(":gráfico\_de\_barras: Análisis de Ventas")# :carpeta\_abierta: Subida de archivo (CSV o Excel)

archivo = st.file\_uploader(":carpeta\_abierta: Sube un archivo CSV o Excel", type=["csv", "xlsx"])if archivo:

# :chincheta: Detectar formato y leer archivo

if archivo.name.endswith(".csv"):

df = pd.read\_csv(archivo)

elif archivo.name.endswith(".xlsx"):

df = pd.read\_excel(archivo, engine="openpyxl") # Para compatibilidad con Excel moderno # :chincheta: Convertir Fecha a formato datetime

df["Fecha"] = pd.to\_datetime(df["Fecha"]) # :lupa: Mostrar primeros registros

st.write(":lupa: Vista previa de los datos:")

st.dataframe(df.head()) # :gráfico\_con\_tendencia\_ascendente: Visualización: Cantidad total vendida por producto

st.subheader(":paquete: Ventas Totales por Producto")

ventas\_por\_producto = df.groupby("Producto")["Cantidad"].sum().sort\_values(ascending=False)

st.bar\_chart(ventas\_por\_producto) # :gráfico\_de\_barras: Mostrar estadísticas descriptivas

st.subheader(":gráfico\_de\_barras: Estadísticas Descriptivas")

st.write(df.describe()) # :dardo: Filtro por producto

producto\_seleccionado = st.selectbox(":lupa: Selecciona un producto:", df["Producto"].unique())

df\_filtrado = df[df["Producto"] == producto\_seleccionado]

st.write(f":paquete: Datos para el producto: {producto\_seleccionado}")

st.dataframe(df\_filtrado) # :fecha: Gráfico de ventas por fecha

st.subheader(":fecha: Evolución de Ventas en el Tiempo") # Extraer el año y mes de la columna 'Fecha' para crear una nueva columna

df["Mes-Año"] = df["Fecha"].dt.to\_period("M").dt.strftime('%Y-%m') # Convertimos a string "YYYY-MM"

ventas\_por\_mes = df.groupby("Mes-Año")["Total"].sum() # Agrupar por Mes-Año # Mostrar gráfico de líneas con la evolución de las ventas por mes

st.line\_chart(ventas\_por\_mes) # :gráfico\_con\_tendencia\_ascendente: Histograma de Precios Unitarios

st.subheader(":bolsa\_de\_dinero: Distribución de Precios Unitarios")

fig, ax = plt.subplots()

ax.hist(df["Precio Unitario"], bins=10, color="skyblue", edgecolor="black")

st.pyplot(fig) # :calendario: Resumen mensual de ventas

st.subheader(":calendario: Resumen de Ventas por Mes")

df["Mes-Año"] = df["Fecha"].dt.to\_period("M").dt.strftime('%Y-%m') # Crear una nueva columna con el mes y el año como string

ventas\_por\_mes = df.groupby("Mes-Año")["Total"].sum()

st.bar\_chart(ventas\_por\_mes) # :trofeo: Top 5 de productos más vendidos

st.subheader(":trofeo: Top 5 de Productos Más Vendidos")

top\_productos = df.groupby("Producto")["Total"].sum().sort\_values(ascending=False).head(5)

st.table(top\_productos) # :fecha: Filtro de fechas

st.subheader(":fecha: Filtrar Ventas por Rango de Fechas") # Selección de fechas con corrección

min\_fecha = df["Fecha"].min().date()

max\_fecha = df["Fecha"].max().date()

fecha\_inicio, fecha\_fin = st.date\_input(

":calendario: Selecciona un rango de fechas",

(min\_fecha, max\_fecha)

) # Convertir a datetime64[ns] antes de filtrar

fecha\_inicio = pd.to\_datetime(fecha\_inicio)

fecha\_fin = pd.to\_datetime(fecha\_fin) df\_filtrado\_fechas = df[(df["Fecha"] >= fecha\_inicio) & (df["Fecha"] <= fecha\_fin)]

st.write(f":calendario: Datos entre {fecha\_inicio.date()} y {fecha\_fin.date()}")

st.dataframe(df\_filtrado\_fechas) # :bandeja\_de\_entrada: Botón para descargar datos filtrados en CSV

st.subheader(":bandeja\_de\_entrada: Descargar Datos Filtrados")

csv = df\_filtrado\_fechas.to\_csv(index=False).encode("utf-8")

st.download\_button(label=":bandeja\_de\_entrada: Descargar CSV", data=csv, file\_name="datos\_filtrados.csv", mime="text/csv") # :bandeja\_de\_entrada: Botón para descargar datos filtrados en Excel

output = io.BytesIO()

with pd.ExcelWriter(output, engine="xlsxwriter") as writer:

df\_filtrado\_fechas.to\_excel(writer, index=False, sheet\_name="Datos Filtrados")

writer.close()

st.download\_button(

label=":bandeja\_de\_entrada: Descargar Excel",

data=output.getvalue(),

file\_name="datos\_filtrados.xlsx",

mime="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"

CON COLORES:

%%writefile color.pyimport streamlit as st

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from pyngrok import ngrok# :estrella2: Configuración de la aplicación

st.set\_page\_config(page\_title=":barco: Análisis de Titanic", layout="wide")

st.title(":barco: Análisis de Titanic")# :carpeta\_abierta: Subida de archivo CSV

archivo = st.file\_uploader(":carpeta\_abierta: Sube un archivo CSV del Titanic", type=["csv"])if archivo:

df = pd.read\_csv(archivo) # :lupa: Mostrar primeros registros

st.write(":lupa: Vista previa de los datos:")

st.dataframe(df.head()) # :gráfico\_de\_barras: Estadísticas descriptivas

st.subheader(":gráfico\_de\_barras: Estadísticas Descriptivas")

st.write(df.describe()) # :gráfico\_con\_tendencia\_descendente: Correlaciones entre variables

st.subheader(":gráfico\_con\_tendencia\_descendente: Correlaciones entre variables")

df["Sex"] = df["Sex"].map({"male": 0, "female": 1})

df["Embarked"] = df["Embarked"].map({"C": 0, "Q": 1, "S": 2})

df\_numeric = df.select\_dtypes(include=["float64", "int64"])

correlaciones = df\_numeric.corr()

st.write(correlaciones) # Mapa de calor de correlación

st.subheader(":gráfico\_de\_barras: Mapa de Calor de Correlación")

fig, ax = plt.subplots()

sns.heatmap(correlaciones, annot=True, cmap="coolwarm", ax=ax)

st.pyplot(fig) # :gráfico\_con\_tendencia\_descendente: Distribución de edades

st.subheader(":gráfico\_con\_tendencia\_descendente: Distribución de Edades")

fig, ax = plt.subplots()

ax.hist(df["Age"].dropna(), bins=20, color="skyblue", edgecolor="black")

ax.set\_title("Distribución de Edades")

ax.set\_xlabel("Edad")

ax.set\_ylabel("Frecuencia")

st.pyplot(fig) # :gráfico\_de\_barras: Gráfico de barras de supervivencia por clase con colores diferentes

st.subheader(":gráfico\_de\_barras: Supervivencia por Clase")

supervivencia\_por\_clase = df.groupby("Pclass")["Survived"].mean() fig, ax = plt.subplots()

colors = ["blue", "orange", "green"]

sns.barplot(x=supervivencia\_por\_clase.index, y=supervivencia\_por\_clase.values, palette=colors, ax=ax)

ax.set\_title("Supervivencia por Clase")

ax.set\_xlabel("Clase")

ax.set\_ylabel("Tasa de Supervivencia")

st.pyplot(fig) # :calendario\_de\_sobremesa: Contar cantidad de sobrevivientes

st.subheader(":calendario\_de\_sobremesa: Número de Sobrevivientes")

sobrevivientes = df["Survived"].value\_counts()

st.write(f"Sobrevivientes: {sobrevivientes[1]} - No sobrevivientes: {sobrevivientes[0]}") # :gráfico\_de\_barras: Gráfico de barras de supervivencia por sexo con colores diferentes

st.subheader(":gráfico\_de\_barras: Supervivencia por Sexo")

supervivencia\_por\_sexo = df.groupby("Sex")["Survived"].mean() fig, ax = plt.subplots()

colors = ["purple", "pink"]

sns.barplot(x=["Male", "Female"], y=supervivencia\_por\_sexo.values, palette=colors, ax=ax)

ax.set\_title("Supervivencia por Sexo")

ax.set\_xlabel("Sexo")

ax.set\_ylabel("Tasa de Supervivencia")

st.pyplot(fig) # :repetir: Gráfico de dispersión: Edad vs. Tarifa

st.subheader(":repetir: Edad vs. Tarifa")

fig, ax = plt.subplots()

ax.scatter(df["Age"], df["Fare"], alpha=0.5, color="green")

ax.set\_title("Edad vs. Tarifa")

ax.set\_xlabel("Edad")

ax.set\_ylabel("Tarifa")

st.pyplot(fig) # :calendario\_de\_sobremesa: Filtro de Edad

st.subheader(":calendario\_de\_sobremesa: Filtrar por Rango de Edad")

edad\_min = st.slider("Edad mínima", 0, 100, 0)

edad\_max = st.slider("Edad máxima", 0, 100, 100)

df\_filtrado\_edad = df[(df["Age"] >= edad\_min) & (df["Age"] <= edad\_max)]

st.write(f":calendario: Datos entre {edad\_min} y {edad\_max} años")

st.dataframe(df\_filtrado\_edad) # :fecha: Botón para descargar los datos filtrados

st.subheader(":fecha: Descargar Datos Filtrados")

csv = df\_filtrado\_edad.to\_csv(index=False).encode("utf-8")

st.download\_button(label=":fecha: Descargar CSV", data=csv, file\_name="datos\_filtrados.csv", mime="text/csv")

%%writefile bayes.py

import streamlit as st

import pandas as pd

from sklearn.naive\_bayes import MultinomialNB

from sklearn.feature\_extraction.text import CountVectorizer

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

from sklearn.metrics import accuracy\_score# :estrella2: Configuración de la app

st.set\_page\_config(page\_title=":correo\_electrónico: Clasificador de Spam", layout="wide")

st.title(":correo\_electrónico: Clasificador de Spam con Naïve Bayes")# :carpeta\_abierta: Crear el dataset

data = {

'mensaje': [

'Oferta especial en relojes, haz clic aquí',

'Querido usuario, su factura está lista',

'Gana dinero rápido desde casa',

'Descuento en tu próximo viaje',

'Tu recibo de pago ha sido procesado',

'Invitación a evento exclusivo',

'¡Felicidades! Has ganado un iPhone, reclama tu premio ahora',

'¡Última oportunidad! Compra ahora y ahorra un 50%',

'Oferta limitada en productos electrónicos',

'Concurso exclusivo, gana premios increíbles',

'Tu cuenta de correo ha sido hackeada, cambia tu contraseña',

'Actualización de seguridad importante para tu cuenta',

'Promoción para estudiantes, descuento especial',

'Tu suscripción a nuestra revista ha sido renovada',

'¡Reclama tu recompensa! Haz clic aquí para más detalles',

'Gran oferta de último minuto para viajar a París',

'Nuevo trabajo disponible, haz clic para más detalles',

'Notificación de seguridad en tu cuenta bancaria',

'Te hemos inscrito en el sorteo de un coche nuevo',

'Ganaste un premio de $1000, haz clic para reclamarlo',

'Te invitamos a conocer nuestro nuevo producto exclusivo', # Añadimos más mensajes spam

'¡Gran descuento en productos tecnológicos, solo por hoy!',

'Estás a punto de ganar un coche, haz clic para más información',

'Gana un iPhone con solo participar en este sencillo formulario',

'¡Oferta exclusiva solo para ti! Compra ahora y ahorra un 70%',

'¡Última oportunidad! Los stocks están a punto de agotarse',

'Has ganado una tarjeta regalo de $500, reclámala ya',

'Te has ganado un viaje a Cancún, ¡reclama tu premio ahora!', # Mensajes no spam

'¿Puedes enviarme el informe de ventas de este mes?',

'La reunión de la próxima semana se ha cancelado',

'Recuerda que la fecha límite para entregar el proyecto es el lunes',

'Revisé tu presentación y creo que necesitas más detalles en la sección de análisis',

'No olvides que hoy es el último día para presentar la solicitud',

'Nuestro equipo ha preparado el informe que me solicitaste'

],

'spam': [

1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, # Spam

1, 1, 1, 1, 1, 1, 1, # Añadimos más spam

0, 0, 0, 0, 0, 0, # No spam

]

}# Verificar que ambas listas tengan la misma longitud

if len(data['mensaje']) != len(data['spam']):

st.error("Las listas no tienen la misma longitud. Por favor revisa los datos.")

else:

# :globo\_terráqueo\_con\_meridianos: Convertir el dataset a un DataFrame

df = pd.DataFrame(data) # Mostrar el DataFrame con los datos

st.subheader(":separadores\_de\_índice\_de\_tarjetas: Datos del Dataset")

st.write(df) # :flechas\_en\_sentido\_antihorario: Vectorización de los mensajes con CountVectorizer

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(df['mensaje'])

y = df['spam'] # :gráfico\_de\_barras: Dividir el dataset en entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42) # :gráfico\_con\_tendencia\_ascendente: Entrenar el modelo Naïve Bayes

model = MultinomialNB()

model.fit(X\_train, y\_train) # :persona\_tecnóloga: \*\*Hacer predicción en los datos de prueba\*\*

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

accuracy = accuracy\_score(y\_test, y\_pred) # :ordenador\_de\_sobremesa: Mostrar la precisión

st.write(f":gráfico\_de\_barras: \*\*Precisión del modelo en datos de prueba:\*\* {accuracy \* 100:.2f}%") # :persona\_tecnóloga: \*\*Entrada del usuario\*\*

st.subheader(":lupa: Introduce un mensaje para clasificarlo como spam o no spam") # Entrada de texto

mensaje\_usuario = st.text\_area("Escribe un mensaje:") # :altavoz\_sonando: \*\*Hacer predicción para el mensaje introducido\*\*

if st.button(":bola\_de\_cristal: Predecir"):

# Vectorizamos el mensaje del usuario

mensaje\_vectorizado = vectorizer.transform([mensaje\_usuario]) # Realizamos la predicción

prediccion = model.predict(mensaje\_vectorizado)[0] # Mostramos el resultado

if prediccion == 1:

st.write(f":luz\_giratoria: El mensaje \*\*ES SPAM\*\*.")

else:

st.write(f":marca\_de\_verificación\_blanca: El mensaje \*\*NO ES SPAM\*\*.")

21:27

%%writefile tension.pyimport streamlit as st

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_diabetes

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score# :estrella2: Configuración de la app

st.set\_page\_config(page\_title=":gráfico\_de\_barras: Regresión Lineal - Diabetes", layout="wide")

st.title(":gráfico\_de\_barras: Regresión Lineal Múltiple - Predicción de Diabetes")# :carpeta\_abierta: Cargar el dataset de diabetes

diabetes = load\_diabetes()

df = pd.DataFrame(diabetes.data, columns=diabetes.feature\_names)

df["Progresión Diabetes"] = diabetes.target # Variable objetivo