```
# -*- coding: utf-8 -*-
"""PROYECTO.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1-k76XgqBnOLQWW8znEwvwuImTOG0XfTx
"""

!pip install requests

!pip install beautifulsoup4

"""#EXPLICACION DEL CODIGO DE LOS JUGADORES DE LA BUNDESLIGA
Este código realiza varias tareas relacionadas con la gestión de datos sobre valores de mercado de jugadores de fútbol en la Bundesliga. Aquí te explico paso a

##1. Importación de Librerías

```
import pandas as pd

from IPython.display import display, HTML
```

* pandas (pd): Se utiliza para manejar y analizar datos estructurados, como tablas (DataFrames).
* display y HTML: Estas funciones de IPython.display permiten mostrar contenido HTML directamente en notebooks de Jupyter.

##2. Creación del DataFrame
```
data = [
  # (lista de listas con los datos de los jugadores)
  ]
```
* Se define una lista de listas, donde cada sublista representa la información de un jugador. Las columnas incluyen detalles como nombre, posición, nacionalida
* Columnas:
  * "Nombre": Nombre del jugador.
  * "Posición": Posición en el campo (e.g., delantero centro, mediocentro ofensivo).
  * "Nacionalidad": País(es) de origen del jugador.
  * "Edad": Edad actual del jugador.
  * "Equipo": Equipo en el que juega actualmente.
  * "Valor de Mercado en 01/01/2024": Valor estimado en esa fecha.
  * "Valor de Mercado Actual": Valor estimado en el momento presente.
  * "Incremento (%)": Incremento porcentual del valor de mercado.
  * "Incremento (€)": Incremento absoluto en millones de euros.

```df = pd.DataFrame(data, columns=[...])```

* pd.DataFrame(data, columns=...) convierte los datos en una tabla estructurada, donde las columnas tienen nombres específicos.

##3. Generación de Tabla HTML

```html_table = df.to_html(index=False, classes=['table', 'table-striped'])```

* to_html: Convierte el DataFrame en código HTML para que pueda mostrarse como una tabla.
* Parámetros:
  * index=False: Omite la columna de índice.
  * classes=['table', 'table-striped']: Añade clases CSS para mejorar el formato (útil en frameworks como Bootstrap).

##4. Visualización en Jupyter Notebook

```display(HTML(html_table))```

* display y HTML: Muestran la tabla en formato HTML en un entorno interactivo como Jupyter Notebook.

##5. Guardar los Datos en un Archivo CSV
```
df.to_csv("valores_mercado_bundesliga.csv", index=False)
print("Archivo 'valores_mercado_bundesliga.csv' guardado exitosamente.")
```
* to_csv: Exporta el DataFrame a un archivo CSV (valores separados por comas).
  * index=False: No incluye la columna de índice en el archivo.
* El archivo se guarda como "valores_mercado_bundesliga.csv" en el directorio de trabajo actual.

"""

import pandas as pd
from IPython.display import display, HTML

# Primero, crea el DataFrame con tus datos
data = [
    ["Aleksandar Pavlovic", "Pivote", "Alemania/Serbia", "20", "Bayern Múnich", "2.00 mill.", "50.00 mill.", "2400.0", "+48.00 mill."],
["Florian Wirtz", "Mediocentro ofensivo", "Alemania", "21", "Bayer 04 Leverkusen", "100.00 mill.", "130.00 mill.", "30.0", "+30.00 mill."],
["Omar Marmoush", "Delantero centro", "Egipto/Canadá", "25", "Eintracht Fráncfort ", "15.00 mill.", "40.00 mill.", "166.7", "+25.00 mill."],
["Jamal Musiala", "Mediocentro ofensivo", "Alemania/Inglaterra", "21", "Bayern Múnich", "110.00 mill.", "130.00 mill.", "18.2", "+20.00 mill."],
["Benjamin Sesko", "Delantero centro", "Eslovenia", "21", "RB Leipzig", "30.00 mill.", "50.00 mill.", "66.7", "+20.00 mill."],
["Maximilian Beier", "Delantero centro", "Alemania", "22", "Borussia Dortmund", "12.00 mill.", "30.00 mill.", "150.0", "+18.00 mill."],
["Josip Stanisic", "Lateral derecho", "Croacia/Alemania", "24", "Bayern Múnich", "10.00 mill.", "28.00 mill.", "180.0", "+18.00 mill."],
["Enzo Millot", "Mediocentro ofensivo", "Francia/Martinica", "22", "VfB Stuttgart", "25.00 mill.", "42.00 mill.", "68.0", "+17.00 mill."],
["Michael Olise", "Extremo derecho", "Francia/Inglaterra", "22", "Bayern Múnich", "50.00 mill.", "65.00 mill.", "30.0", "+15.00 mill."],
["Hugo Ekitiké", "Delantero centro", "Francia/Camerún", "22", "Eintracht Fráncfort ", "15.00 mill.", "30.00 mill.", "100.0", "+15.00 mill."],
["Angelo Stiller", "Pivote", "Alemania", "23", "VfB Stuttgart", "14.00 mill.", "28.00 mill.", "100.0", "+14.00 mill."],
["Waldemar Anton", "Defensa central", "Alemania", "28", "Borussia Dortmund", "10.00 mill.", "24.00 mill.", "140.0", "+14.00 mill."],
["Yan Couto", "Lateral derecho", "Brasil/Portugal", "22", "Borussia Dortmund", "12.00 mill.", "25.00 mill.", "108.3", "+13.00 mill."],
["Maximilian Mittelstädt", "Lateral izquierdo", "Alemania", "27", "VfB Stuttgart", "5.00 mill.", "17.00 mill.", "240.0", "+12.00 mill."],
["Paul Wanner", "Mediocentro ofensivo", "Alemania/Austria", "18", "1.FC Heidenheim 1846", "3.50 mill.", "15.00 mill.", "328.6", "+11.50 mill."],
["Deniz Undav", "Delantero centro", "Alemania/Turquía", "28", "VfB Stuttgart", "17.00 mill.", "28.00 mill.", "64.7", "+11.00 mill."],
["Jeanuël Belocian", "Defensa central", "Francia/Guadalupe", "19", "Bayer 04 Leverkusen", "9.00 mill.", "20.00 mill.", "122.2", "+11.00 mill."],
["Xavi Simons", "Mediocentro ofensivo", "Países Bajos/Surinam", "21", "RB Leipzig", "70.00 mill.", "80.00 mill.", "14.3", "+10.00 mill."],
["Alejandro Grimaldo", "Lateral izquierdo", "España", "29", "Bayer 04 Leverkusen", "35.00 mill.", "45.00 mill.", "28.6", "+10.00 mill."],
["Jamie Leweling", "Extremo derecho", "Alemania/Ghana", "23", "VfB Stuttgart", "4.50 mill.", "13.00 mill.", "188.9", "+8.50 mill."],
["Hiroki Ito", "Defensa central", "Japón", "25", "Bayern Múnich", "22.00 mill.", "30.00 mill.", "36.4", "+8.00 mill."],
["Ermedin Demirovic", "Delantero centro", "Bosnia y Herzegovina", "26", "VfB Stuttgart", "20.00 mill.", "28.00 mill.", "40.0", "+8.00 mill."],
["David Raum", "Lateral izquierdo", "Alemania", "26", "RB Leipzig", "17.00 mill.", "25.00 mill.", "47.1", "+8.00 mill."],
["Jonathan Burkardt", "Delantero centro", "Alemania", "24", "1.FSV Mainz 05", "7.00 mill.", "15.00 mill.", "114.3", "+8.00 mill."],
["Konstantinos Koulierakis", "Defensa central", "Grecia", "20", "VfL Wolfsburgo", "4.50 mill.", "12.00 mill.", "166.7", "+7.50 mill."],
["Karim Adeyemi", "Extremo izquierdo", "Alemania/Nigeria", "22", "Borussia Dortmund", "28.00 mill.", "35.00 mill.", "25.0", "+7.00 mill."],
```
```

```
["Jamie Gittens", "Extremo izquierdo", "Inglaterra/Barbados", "20", "Borussia Dortmund", "28.00 mill.", "35.00 mill.", "25.0", "+7.00 mill."],
["Igor Matanovic", "Delantero centro", "Croacia/Alemania", "21", "Eintracht Fráncfort  ", "1.00 mill.", "8.00 mill.", "700.0", "+7.00 mill."],
["Romano Schmid", "Mediocentro ofensivo", "Austria", "24", "SV Werder Bremen", "6.50 mill.", "13.00 mill.", "100.0", "+6.50 mill."],
["Robin Hranac", "Defensa central", "República Checa", "24", "TSG 1899 Hoffenheim", "1.70 mill.", "8.00 mill.", "370.6", "+6.30 mill."],
["Robert Andrich", "Pivote", "Alemania", "30", "Bayer 04 Leverkusen", "11.00 mill.", "17.00 mill.", "54.5", "+6.00 mill."],
["Mohamed Amoura", "Delantero centro", "Argelia", "24", "VfL Wolfsburgo", "10.00 mill.", "16.00 mill.", "60.0", "+6.00 mill."],
["Merlin Röhl", "Mediocentro ofensivo", "Alemania", "22", "SC Friburgo", "6.00 mill.", "12.00 mill.", "100.0", "+6.00 mill."],
["Eren Dinkçi", "Extremo derecho", "Turquía/Alemania", "22", "SC Friburgo", "6.00 mill.", "12.00 mill.", "100.0", "+6.00 mill."],
["Justin Njinmah", "Extremo derecho", "Alemania/Nigeria", "23", "SV Werder Bremen", "3.00 mill.", "9.00 mill.", "200.0", "+6.00 mill."],
["Felix Agu", "Lateral derecho", "Alemania/Nigeria", "25", "SV Werder Bremen", "600 mil", "6.00 mill.", "900.0", "+5.40 mill."],
["Loïs Openda", "Delantero centro", "Bélgica/Marruecos", "24", "RB Leipzig", "55.00 mill.", "60.00 mill.", "9.1", "+5.00 mill."],
["Exequiel Palacios", "Mediocentro", "Argentina", "26", "Bayer 04 Leverkusen", "40.00 mill.", "45.00 mill.", "12.5", "+5.00 mill."],
["Edmond Tapsoba", "Defensa central", "Burkina Faso", "25", "Bayer 04 Leverkusen", "40.00 mill.", "45.00 mill.", "12.5", "+5.00 mill."],
["Victor Boniface", "Delantero centro", "Nigeria", "23", "Bayer 04 Leverkusen", "40.00 mill.", "45.00 mill.", "12.5", "+5.00 mill."],
["Donyell Malen", "Extremo derecho", "Países Bajos/Surinam", "25", "Borussia Dortmund", "35.00 mill.", "40.00 mill.", "14.3", "+5.00 mill."],
["Nico Schlotterbeck", "Defensa central", "Alemania", "24", "Borussia Dortmund", "35.00 mill.", "40.00 mill.", "14.3", "+5.00 mill."],
["Piero Hincapié", "Defensa central", "Ecuador", "22", "Bayer 04 Leverkusen", "35.00 mill.", "40.00 mill.", "14.3", "+5.00 mill."],
["Amine Adli", "Extremo izquierdo", "Marruecos/Francia", "24", "Bayer 04 Leverkusen", "25.00 mill.", "30.00 mill.", "20.0", "+5.00 mill."],
["Jonathan Tah", "Defensa central", "Alemania/Costa de Marfil", "28", "Bayer 04 Leverkusen", "25.00 mill.", "30.00 mill.", "20.0", "+5.00 mill."],
["Chris Führich", "Extremo izquierdo", "Alemania", "26", "VfB Stuttgart", "17.00 mill.", "22.00 mill.", "29.4", "+5.00 mill."],
["Antonio Nusa", "Extremo izquierdo", "Noruega/Nigeria", "19", "RB Leipzig", "17.00 mill.", "22.00 mill.", "29.4", "+5.00 mill."],
["Julian Ryerson", "Lateral derecho", "Noruega/Estados Unidos", "26", "Borussia Dortmund", "15.00 mill.", "20.00 mill.", "33.3", "+5.00 mill."],
["Nadiem Amiri", "Mediocentro", "Alemania/Afganistán", "28", "1.FSV Mainz 05", "4.00 mill.", "9.00 mill.", "125.0", "+5.00 mill."],
["Robin Hack", "Extremo izquierdo", "Alemania", "26", "Borussia Mönchengladbach", "3.00 mill.", "8.00 mill.", "166.7", "+5.00 mill."]



    # ... (agrega todos los demás registros aquí)
]

df = pd.DataFrame(data, columns=["Nombre", "Posición", "Nacionalidad", "Edad", "Equipo", "Valor de Mercado en 01/01/2024", "Valor de Mercado Actual", "Incremen

# Convierte el DataFrame a una tabla HTML
html_table = df.to_html(index=False, classes=['table', 'table-striped'])

# Muestra la tabla HTML usando display y HTML
display(HTML(html_table))

# O alternativamente, puedes usar display con HTML como argumento
# display(HTML(html_table))

# Guardar el DataFrame en un archivo CSV
df.to_csv("valores_mercado_bundesliga.csv", index=False)
print("Archivo 'valores_mercado_bundesliga.csv' guardado exitosamente.")

"""Este código utiliza Python para trabajar con datos tabulares (jugadores de fútbol en este caso) y genera tanto una visualización en HTML como un archivo HTM

##1. Importación de Módulos
```
import pandas as pd
from IPython.display import display, HTML
```
* pandas: Biblioteca para manejar y analizar datos tabulares.
* IPython.display: Permite mostrar contenido HTML (útil en notebooks como Jupyter).

##2. Creación del DataFrame
```
data = [
    ["Lamine Yamal", "Extremo derecho", "España", "Guinea Ecuatorial", ...],
    # Más registros...
]
df = pd.DataFrame(data, columns=["Nombre", "Posición", "Nacionalidad", "Segundo Nacionalidad", ...])
```
* data: Lista de listas que contiene información sobre los jugadores.
* pd.DataFrame: Convierte esta lista en una estructura tabular (DataFrame) con las columnas especificadas.

##3. Visualización HTML
```
html_table = df.to_html(index=False, classes=['table', 'table-striped'])
display(HTML(html_table))
```
* to_html: Convierte el DataFrame en código HTML, que genera una tabla:
  * index=False: No incluye la columna de índice.
  * classes=['table', 'table-striped']: Aplica clases CSS para formatear la tabla.
* display(HTML(...)): Renderiza el contenido HTML para mostrarlo (útil en notebooks).

##4. Exportación a Excel
```
df.to_excel("jugadores_liga_espanolaa.xlsx", index=False)
print("Datos exportados exitosamente a jugadores_liga_espanolaa.xlsx")
```
* to_excel: Exporta los datos del DataFrame a un archivo Excel.

##5. Eliminación de una Columna
```
df = df.drop(columns=["Segundo Nacionalidad"])
drop: Elimina la columna "Segundo Nacionalidad" del DataFrame.
```

##6. Generación y Guardado del Archivo HTML
```
html_table = df.to_html(index=False, classes=['table', 'table-striped'])
with open('jugadores_liga_espanola.html', 'w', encoding='utf-8') as f:
    f.write(html_table)
```
* Se vuelve a generar una tabla HTML (sin la columna eliminada).
* open: Abre/crea un archivo en modo escritura (w) con codificación UTF-8.
* f.write: Escribe el HTML en el archivo jugadores_liga_espanola.html.

##7. Visualización Final (Opcional)

```display(HTML(html_table))```

Esto muestra la tabla HTML generada en un entorno interactivo como Jupyter Notebook.
"""
```
```

```
import pandas as pd
from IPython.display import display, HTML

# Primero, crea el DataFrame con tus datos
data = [
    ["Lamine Yamal", "Extremo derecho", "España", "Guinea Ecuatorial", "17", "FC Barcelona", "60,00 mill. €", "150,00 mill. €", "150,00 %", "+90,00 mill. €"],
    ["Vinicius Junior", "Extremo izquierdo", "Brasil", "España", "24", "Real Madrid CF", "150,00 mill. €", "200,00 mill. €", "33,3 %", "+50,00 mill. €"],
    ["Pau Cubarsí", "Defensa central", "España", "", "17", "FC Barcelona", "1,50 mill. €", "40,00 mill. €", "2566,7 %", "+38,50 mill. €"],
    ["Fermín López", "Mediocentro", "España", "", "21", "FC Barcelona", "15,00 mill. €", "50,00 mill. €", "233,3 %", "+35,00 mill. €"],
    ["Federico Valverde", "Mediocentro", "Uruguay", "España", "26", "Real Madrid CF", "100,00 mill. €", "130,00 mill. €", "30,0 %", "+30,00 mill. €"],
    ["Arda Güler", "Extremo derecho", "Turquía", "", "19", "Real Madrid CF", "15,00 mill. €", "45,00 mill. €", "200,0 %", "+30,00 mill. €"],
    ["Álex Baena", "Extremo izquierdo", "España", "", "23", "Villarreal CF", "25,00 mill. €", "50,00 mill. €", "100,0 %", "+25,00 mill. €"],
    ["Nico Williams", "Extremo izquierdo", "España", "Ghana", "22", "Athletic Club", "50,00 mill. €", "70,00 mill. €", "40,0 %", "+20,00 mill. €"],
    ["Brahim Díaz", "Mediocentro ofensivo", "Marruecos", "España", "25", "Real Madrid CF", "20,00 mill. €", "40,00 mill. €", "100,0 %", "+20,00 mill. €"],
    ["Cristhian Mosquera", "Defensa central", "España", "Colombia", "20", "Valencia CF", "10,00 mill. €", "30,00 mill. €", "200,0 %", "+20,00 mill. €"],
    ["Orri Óskarsson", "Delantero centro", "Islandia", "", "20", "Real Sociedad", "2,00 mill. €", "20,00 mill. €", "900,0 %", "+18,00 mill. €"],
    ["Isaac Romero", "Delantero centro", "España", "", "24", "Sevilla FC", "400 mil €", "18,00 mill. €", "4400,0 %", "+17,60 mill. €"],
    ["Andriy Lunin", "Portero", "Ucrania", "", "25", "Real Madrid CF", "8,00 mill. €", "25,00 mill. €", "212,5 %", "+17,00 mill. €"],
    ["Endrick", "Delantero centro", "Brasil", "", "18", "Real Madrid CF", "45,00 mill. €", "60,00 mill. €", "33,3 %", "+15,00 mill. €"],
    ["Giorgi Mamardashvili", "Portero", "Georgia", "", "24", "Valencia CF", "30,00 mill. €", "45,00 mill. €", "50,0 %", "+15,00 mill. €"],
    ["Marc Casadó", "Pivote", "España", "", "21", "FC Barcelona", "600 mil €", "15,00 mill. €", "2400,0 %", "+14,40 mill. €"],
    ["Johnny Cardoso", "Pivote", "Estados Unidos", "Italia", "23", "Real Betis Balompié", "6,00 mill. €", "20,00 mill. €", "233,3 %", "+14,00 mill. €"],
    ["Logan Costa", "Defensa central", "Cabo Verde", "Francia", "23", "Villarreal CF", "5,00 mill. €", "18,00 mill. €", "260,0 %", "+13,00 mill. €"],
    ["Thierno Barry", "Delantero centro", "Francia", "Guinea", "22", "Villarreal CF", "1,00 mill. €", "14,00 mill. €", "1300,0 %", "+13,00 mill. €"],
    ["Yáser Asprilla", "Extremo derecho", "Colombia", "", "20", "Girona FC", "6,00 mill. €", "18,00 mill. €", "200,0 %", "+12,00 mill. €"],
    ["Beñat Prados", "Mediocentro", "España", "", "23", "Athletic Club", "3,00 mill. €", "15,00 mill. €", "400,0 %", "+12,00 mill. €"],
    ["Ilias Akhomach", "Extremo derecho", "Marruecos", "España", "20", "Villarreal CF", "3,00 mill. €", "15,00 mill. €", "400,0 %", "+12,00 mill. €"],
    ["Alexander Sørloth", "Delantero centro", "Noruega", "", "28", "Atlético de Madrid", "14,00 mill. €", "25,00 mill. €", "78,6 %", "+11,00 mill. €"],
    ["Rodrygo", "Extremo derecho", "Brasil", "España", "23", "Real Madrid CF", "100,00 mill. €", "110,00 mill. €", "10,0 %", "+10,00 mill. €"],
    ["Eduardo Camavinga", "Mediocentro", "Francia", "Congo", "22", "Real Madrid CF", "90,00 mill. €", "100,00 mill. €", "11,1 %", "+10,00 mill. €"],
    ["Aurélien Tchouaméni", "Pivote", "Francia", "Camerún", "24", "Real Madrid CF", "90,00 mill. €", "100,00 mill. €", "11,1 %", "+10,00 mill. €"],
    ["Dani Olmo", "Mediocentro ofensivo", "España", "", "26", "FC Barcelona", "50,00 mill. €", "60,00 mill. €", "20,0 %", "+10,00 mill. €"],
    ["Martín Zubimendi", "Pivote", "España", "", "25", "Real Sociedad", "50,00 mill. €", "60,00 mill. €", "20,0 %", "+10,00 mill. €"],
    ["Raphinha", "Extremo izquierdo", "Brasil", "Italia", "27", "FC Barcelona", "50,00 mill. €", "60,00 mill. €", "20,0 %", "+10,00 mill. €"],
    ["Dani Vivian", "Defensa central", "España", "", "25", "Athletic Club", "20,00 mill. €", "30,00 mill. €", "50,0 %", "+10,00 mill. €"],
    ["Aitor Paredes", "Defensa central", "España", "", "24", "Athletic Club", "12,00 mill. €", "22,00 mill. €", "83,3 %", "+10,00 mill. €"],
    ["Pepelu", "Pivote", "España", "", "26", "Valencia CF", "12,00 mill. €", "22,00 mill. €", "83,3 %", "+10,00 mill. €"],
    ["Loïc Badé", "Defensa central", "Francia", "Costa de Marfil", "24", "Sevilla FC", "10,00 mill. €", "20,00 mill. €", "100,0 %", "+10,00 mill. €"],
    ["Beñat Turrientes", "Mediocentro", "España", "", "22", "Real Sociedad", "10,00 mill. €", "20,00 mill. €", "100,0 %", "+10,00 mill. €"],
    ["Alberto Moleiro", "Mediocentro ofensivo", "España", "Cuba", "21", "UD Las Palmas", "10,00 mill. €", "20,00 mill. €", "100,0 %", "+10,00 mill. €"],
    ["Mika Mármol", "Defensa central", "España", "", "23", "UD Las Palmas", "5,00 mill. €", "15,00 mill. €", "200,0 %", "+10,00 mill. €"],
    ["Joan García", "Portero", "España", "", "23", "RCD Espanyol", "400 mil €", "10,00 mill. €", "2400,0 %", "+9,60 mill. €"],
    ["Héctor Fort", "Lateral derecho", "España", "", "18", "FC Barcelona", "1,00 mill. €", "10,00 mill. €", "900,0 %", "+9,00 mill. €"],
    ["Enzo Boyomo", "Defensa central", "Camerún", "Francia", "23", "CA Osasuna", "1,60 mill. €", "10,00 mill. €", "525,0 %", "+8,40 mill. €"],
    ["Conor Gallagher", "Mediocentro", "Inglaterra", "Irlanda", "24", "Atlético de Madrid", "42,00 mill. €", "50,00 mill. €", "19,0 %", "+8,00 mill. €"],
    ["Gorka Guruzeta", "Delantero centro", "España", "", "28", "Athletic Club", "10,00 mill. €", "18,00 mill. €", "80,0 %", "+8,00 mill. €"],
    ["Hugo Duro", "Delantero centro", "España", "", "25", "Valencia CF", "8,00 mill. €", "16,00 mill. €", "100,0 %", "+8,00 mill. €"],
    ["Álvaro Djaló", "Extremo izquierdo", "España", "Guinea-Bissau", "25", "Athletic Club", "7,00 mill. €", "15,00 mill. €", "114,3 %", "+8,00 mill. €"],
    ["Jon Pacheco", "Defensa central", "España", "", "23", "Real Sociedad", "8,00 mill. €", "15,00 mill. €", "87,5 %", "+7,00 mill. €"],
    ["Juanlu Sánchez", "Lateral derecho", "España", "", "21", "Sevilla FC", "8,00 mill. €", "15,00 mill. €", "87,5 %", "+7,00 mill. €"],
    ["Carlos Vicente", "Extremo derecho", "España", "", "25", "Deportivo Alavés", "1,20 mill. €", "8,00 mill. €", "566,7 %", "+6,80 mill. €"],
    ["Óscar Mingueza", "Lateral derecho", "España", "", "25", "RC Celta de Vigo", "6,00 mill. €", "12,00 mill. €", "100,0 %", "+6,00 mill. €"],
    ["Kirian Rodríguez", "Mediocentro ofensivo", "España", "", "28", "UD Las Palmas", "6,00 mill. €", "12,00 mill. €", "100,0 %", "+6,00 mill. €"],
    ["Kike Salas", "Defensa central", "España", "", "22", "Sevilla FC", "2,00 mill. €", "8,00 mill. €", "300,0 %", "+6,00 mill. €"],
    ["Jorge Herrando", "Defensa central", "España", "", "23", "CA Osasuna", "800 mil €", "6,00 mill. €", "650,0 %", "+5,20 mill. €"]
    # ... (agrega todos los demás registros aquí)
]

df = pd.DataFrame(data, columns=["Nombre", "Posición", "Nacionalidad", "Segundo Nacionalidad", "Edad", "Equipo", "Valor de Mercado en 01/01/2024", "Valor de Me

# Convierte el DataFrame a una tabla HTML
html_table = df.to_html(index=False, classes=['table', 'table-striped'])

# Muestra la tabla HTML usando display y HTML
display(HTML(html_table))

# O alternativamente, puedes usar display con HTML como argumento
# display(HTML(html_table))


print("Datos exportados exitosamente a jugadores_liga_espanolaa.xlsx")

"""# Depuracion del codigo, limpieza de segunda nacionalidad."""

import pandas as pd
from IPython.display import display, HTML

# Tu código para crear el DataFrame (suponiendo que lo has hecho antes)

# Eliminar la columna "País de origen"
df = df.drop(columns=["Segundo Nacionalidad"])

# Generar el archivo HTML
html_table = df.to_html(index=False, classes=['table', 'table-striped'])

# Guardar el contenido HTML en un archivo
with open('jugadores_liga_espanola.html', 'w', encoding='utf-8') as f:
    f.write(html_table)

# Mostrar la tabla HTML (opcional)
display(HTML(html_table))

# Exportar el DataFrame a un archivo Excel
df.to_excel('jugadores_liga_espanola.xlsx', index=False)

print("Datos exportados exitosamente a jugadores_liga_espanolaa.xlsx")

# Guardar el DataFrame en un archivo CSV
df.to_csv("valores_mercado_actualizados.csv", index=False)
print("Archivo 'valores_mercado_actualizados.csv' guardado exitosamente.")

"""Este código procesa datos financieros de valores de mercado y los convierte de cadenas de texto con formato (por ejemplo, "mil €" o "mill. €") a valores ent

##1. Definición de la Función convertir_valor
```
def convertir_valor(valor):
```
```

```
    if "mil €" in valor:
        return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)  # Convierte a euros completos
    elif "mill. €" in valor:
        return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)  # Convierte a euros completos
    else:
        return None
```
Detalles de la Función
* Propósito: Convierte un valor financiero en formato de texto a un entero en euros.
* Parámetro valor: Una cadena que representa el valor (ej., "10 mil €" o "1,5 mill. €").
* Proceso Interno:
  * Si el valor contiene "mil €", se elimina esta parte del texto con replace, convirtiendo el número a un flotante y multiplicándolo por 1,000 para obtener eu
  * Si contiene "mill. €", el proceso es similar, pero se multiplica por 1,000,000.
  * Si no contiene ninguna de estas cadenas, la función retorna None (valor desconocido).
* int(float(...)):
  * Convierte la cadena a flotante para manejar valores decimales (ej., "1,5 mill. €").
  * Finalmente, lo transforma a entero porque los valores se consideran redondeados a euros completos.
##2. Aplicación de la Función a las Columnas
```
df["Valor de Mercado en 01/01/2024"] = df["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
df["Valor de Mercado Actual"] = df["Valor de Mercado Actual"].apply(convertir_valor)
df["Incremento (€)"] = df["Incremento (€)"].apply(convertir_valor)
```

Detalles de Aplicación
* df["Columna"].apply(función):
  * Aplica la función convertir_valor a cada elemento de la columna especificada en el DataFrame df.
* Columnas Procesadas:
  * "Valor de Mercado en 01/01/2024": Convierte valores iniciales de mercado.
  * "Valor de Mercado Actual": Convierte los valores actuales.
  * "Incremento (€)": Convierte las diferencias de mercado en euros.
* bEfecto del Proceso:
  * Las cadenas como "10 mil €" se convierten en el entero 10000.
  * Las cadenas como "1,5 mill. €" se convierten en 1500000.

##3. Contexto y Resultados
* Antes de la Conversión: Las columnas contienen valores en formato de texto, por ejemplo:
css

```["10 mil €", "1,5 mill. €", "500 mil €"]```

* Después de la Conversión: Los valores se convierten a enteros:
csharp

```[10000, 1500000, 500000]```

* Uso Posterior: Con los datos convertidos a números, puedes realizar cálculos matemáticos o análisis (como sumas, promedios, etc.) sin errores.
"""

```
# Función para convertir los valores de mercado a euros completos (enteros)
def convertir_valor(valor):
    if "mil €" in valor:
        return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)  # Convierte a euros completos
    elif "mill. €" in valor:
        return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)  # Convierte a euros completos
    else:
        return None

# Aplicar la función de conversión a las columnas de valores de mercado
df["Valor de Mercado en 01/01/2024"] = df["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
df["Valor de Mercado Actual"] = df["Valor de Mercado Actual"].apply(convertir_valor)
df["Incremento (€)"] = df["Incremento (€)"].apply(convertir_valor)
```

"""Este código proporciona una funcionalidad interactiva para graficar el crecimiento en el valor de mercado de un jugador a lo largo del tiempo.
##1. Importación de módulos
```
import plotly.graph_objects as go
from datetime import datetime, timedelta
import pandas as pd
```
* plotly.graph_objects: Se usa para crear gráficos interactivos, en este caso, una gráfica de barras.
* datetime y timedelta: Módulos para manejar fechas y horas.
* pandas: Biblioteca para trabajar con datos estructurados, como tablas.

##2. Definir fechas
```
fecha_inicio = datetime(2024, 1, 1)
fecha_hoy = datetime.today()
```
* fecha_inicio: Define la fecha de inicio del seguimiento, en este caso, el 1 de enero de 2024.
* fecha_hoy: Obtiene la fecha actual para marcar el fin del seguimiento.

##3. Función graficar_jugador
La función toma el nombre del jugador, calcula la evolución de su valor de mercado y genera una gráfica interactiva.

####a. Entrada del usuario

```nombre_jugador = input("Escribe el nombre del jugador: ")```

* Pide al usuario que ingrese el nombre del jugador.

####b. Filtrar al jugador

```jugador = df[df['Nombre'] == nombre_jugador]```
* Busca el registro del jugador en un dataframe (df) que contiene datos de jugadores.
* df: Se asume que es una tabla donde cada fila representa a un jugador con sus atributos, como el nombre y valores de mercado inicial y actual.
####c. Validación de datos
```
if not jugador.empty:
    try:
        valor_inicial = float(jugador['Valor de Mercado en 01/01/2024'].values[0])
        valor_actual = float(jugador['Valor de Mercado Actual'].values[0])
    except ValueError:
        print("Error: Los valores de mercado deben ser numéricos.")
        return
```

```
* Si el jugador existe en la tabla:
  * Obtiene su valor de mercado inicial y actual.
  * Convierte estos valores a flotantes para asegurarse de que son números.
* Si hay un error en los datos (por ejemplo, si no son números), muestra un mensaje de error y termina la función.
####d. Generar fechas y valores
```
fechas = pd.date_range(fecha_inicio, fecha_hoy, freq='MS')
valores = [valor_inicial + (valor_actual - valor_inicial) * (i / (len(fechas) - 1)) for i in range(len(fechas))]
```
* fechas: Crea una lista de fechas, una para el inicio de cada mes desde la fecha inicial hasta la fecha actual.
* valores: Calcula una interpolación lineal entre el valor inicial y el valor actual para cada mes.

Fórmula:

valor en el mes
i
=
valor inicial
+
(
valor actual
−
valor inicial
)
×
i
n
−
1
valor en el mes i=valor inicial+(valor actual−valor inicial)×
i/n−1


Donde:

* i es el índice del mes.
* n es el número total de meses.

Esto asegura que el valor aumente uniformemente cada mes.

####e. Crear la gráfica
```
fig = go.Figure(data=go.Bar(x=fechas, y=valores))
fig.update_layout(title=f'Evolución del Valor de Mercado de {nombre_jugador}',
                  xaxis_title='Fecha',
                  yaxis_title='Valor de Mercado (mill. €)')
fig.show()
```
* go.Bar: Crea una gráfica de barras con las fechas en el eje X y los valores interpolados en el eje Y.
* update_layout: Personaliza el título de la gráfica y los ejes.
* fig.show(): Muestra la gráfica interactiva.

####f. Mensaje si el jugador no existe
```
else:
    print("Jugador no encontrado.")
```
* Si el nombre del jugador no está en el dataframe, muestra un mensaje de error.

####4. Ejecutar la función
```graficar_jugador()```
* Llama a la función para iniciar el flujo del programa.
"""

```python
import plotly.graph_objects as go
from datetime import datetime, timedelta
import pandas as pd

# Fecha de inicio y fecha actual
fecha_inicio = datetime(2024, 1, 1)
fecha_hoy = datetime.today()

# Función para graficar el crecimiento en valor de mercado
def graficar_jugador():
    # Pedir el nombre del jugador al usuario
    nombre_jugador = input("Escribe el nombre del jugador: ")

    # Filtrar al jugador
    jugador = df[df['Nombre'] == nombre_jugador]

    if not jugador.empty:
        # Obtener valores inicial y actual, convirtiéndolos a float si es necesario
        try:
            valor_inicial = float(jugador['Valor de Mercado en 01/01/2024'].values[0])
            valor_actual = float(jugador['Valor de Mercado Actual'].values[0])
        except ValueError:
            print("Error: Los valores de mercado deben ser numéricos.")
            return

        # Generar fechas mensuales desde el inicio hasta hoy
        fechas = pd.date_range(fecha_inicio, fecha_hoy, freq='MS')  # 'MS' para inicio de cada mes
        valores = [valor_inicial + (valor_actual - valor_inicial) * (i / (len(fechas) - 1)) for i in range(len(fechas))]

        # Crear la gráfica de barras
        fig = go.Figure(data=go.Bar(x=fechas, y=valores))
        fig.update_layout(title=f'Evolución del Valor de Mercado de {nombre_jugador}',
                          xaxis_title='Fecha',
                          yaxis_title='Valor de Mercado (mill. €)')
        fig.show()
    else:
        print("Jugador no encontrado.")

# Llamar a la función
graficar_jugador()

"""#DATOS IMPORTADOS DESDE EL CSV CREADO"""
```

```
import pandas as pd

# Enlace directo al archivo raw en GitHub
file_path = 'https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv'

# Cargar el archivo CSV desde GitHub
data = pd.read_csv(file_path)

# Mostrar el DataFrame completo
pd.set_option('display.max_rows', None)
data

"""Este código crea una aplicación interactiva para graficar la evolución del valor de mercado de un jugador a lo largo del tiempo. A continuación, se desglosa

####1. Importación de módulos
```
import plotly.graph_objects as go
from datetime import datetime
import pandas as pd
```
* plotly.graph_objects: Se utiliza para generar gráficos interactivos. Aquí se crea una gráfica de barras.
* datetime: Maneja fechas y horas.
* pandas: Facilita la manipulación de datos tabulares (como tablas o CSVs).

####2. Cargar datos desde GitHub
```
file_path = 'https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv'
data = pd.read_csv(file_path)
```
* file_path: Contiene la URL del archivo CSV alojado en un repositorio de GitHub.
* pd.read_csv: Carga los datos del CSV en un DataFrame, que es una estructura de datos tabular similar a una hoja de cálculo.

####3. Conversión de valores de mercado
```
def convertir_valor(valor):
    if isinstance(valor, str):
        if "mil €" in valor:
            return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)
        elif "mill. €" in valor:
            return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)
    return None
```
Esta función convierte los valores de mercado en cadenas de texto como "20 mil €" o "3.5 mill. €" a números enteros representados en euros completos.

Proceso:
1. Si el valor contiene "mil €", lo convierte a miles de euros multiplicando por 1,000.
2. Si contiene "mill. €", lo convierte a millones de euros multiplicando por 1,000,000.
3. Si el valor no tiene formato esperado, devuelve None.

####4. Aplicar la conversión al DataFrame
```
if 'Valor de Mercado en 01/01/2024' in data.columns and 'Valor de Mercado Actual' in data.columns:
    data["Valor de Mercado en 01/01/2024"] = data["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
    data["Valor de Mercado Actual"] = data["Valor de Mercado Actual"].apply(convertir_valor)
else:
    print("Las columnas 'Valor de Mercado en 01/01/2024' o 'Valor de Mercado Actual' no existen en el archivo CSV.")
```
* Comprueba si las columnas Valor de Mercado en 01/01/2024 y Valor de Mercado Actual están presentes en el DataFrame.
* Aplica la función convertir_valor a estas columnas para convertir los valores al formato numérico.
* Si faltan las columnas, muestra un mensaje de error.

####5. Fechas clave
```
fecha_inicio = datetime(2024, 1, 1)
fecha_hoy = datetime.today()
```
Define:

* fecha_inicio: Fecha inicial del análisis (1 de enero de 2024).
* fecha_hoy: Fecha actual, usada como punto final del análisis.

####6. Función para graficar la evolución
```
def graficar_jugador():
    nombre_jugador = input("Escribe el nombre del jugador: ")
    jugador = data[data['Nombre'] == nombre_jugador]
```
* Solicita al usuario que ingrese el nombre del jugador.
* Filtra los datos del jugador correspondiente usando data[data['Nombre'] == nombre_jugador].

#####a. Validación del jugador:
```
if not jugador.empty:
    valor_inicial = jugador['Valor de Mercado en 01/01/2024'].values[0]
    valor_actual = jugador['Valor de Mercado Actual'].values[0]
```
* Comprueba si el jugador existe en el DataFrame.
* Extrae sus valores de mercado inicial y actual.

#####b. Generación de fechas y valores
```
fechas = pd.date_range(fecha_inicio, fecha_hoy, freq='MS')
valores = [valor_inicial + (valor_actual - valor_inicial) * (i / (len(fechas) - 1)) for i in range(len(fechas))]
```
* pd.date_range: Genera una lista de fechas mensuales desde la fecha de inicio hasta la fecha actual.
* valores: Interpola los valores de mercado entre el inicial y el actual para cada fecha.
Fórmula de interpolación:

valor en el mes
i
=
valor inicial
+
(
valor actual
```

−
valor inicial
)
×
i /
n
−
1

Donde:

* i es el índice del mes.
* n es el número total de meses.

#####c. Crear la gráfica
```
fig = go.Figure(data=go.Bar(x=fechas, y=valores))
fig.update_layout(title=f'Evolución del Valor de Mercado de {nombre_jugador}',
                  xaxis_title='Fecha',
                  yaxis_title='Valor de Mercado (€)',
                  xaxis=dict(tickformat="%Y-%m"))
fig.show()
```
* go.Bar: Genera una gráfica de barras con las fechas en el eje X y los valores interpolados en el eje Y.
* update_layout: Personaliza el título, etiquetas de ejes y formato de las fechas.

#####d. Mensaje si el jugador no existe
```
else:
    print("Jugador no encontrado.")
```
* Muestra un mensaje de error si el jugador no está en el DataFrame.
#####7. Llamada a la función
```
graficar_jugador()```

* Inicia la ejecución del programa.
"""

```python
import plotly.graph_objects as go
from datetime import datetime
import pandas as pd

# Enlace directo al archivo raw en GitHub
file_path = 'https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv'

# Cargar el archivo CSV desde GitHub
data = pd.read_csv(file_path)

# Función para convertir los valores de mercado a euros completos (enteros)
def convertir_valor(valor):
    if isinstance(valor, str):
        if "mil €" in valor:
            return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)  # Convierte a euros completos
        elif "mill. €" in valor:
            return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)  # Convierte a euros completos
    return None

# Verificar que las columnas existen en el DataFrame antes de aplicar la función
if 'Valor de Mercado en 01/01/2024' in data.columns and 'Valor de Mercado Actual' in data.columns:
    # Aplicar la función de conversión a las columnas de valores de mercado
    data["Valor de Mercado en 01/01/2024"] = data["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
    data["Valor de Mercado Actual"] = data["Valor de Mercado Actual"].apply(convertir_valor)
else:
    print("Las columnas 'Valor de Mercado en 01/01/2024' o 'Valor de Mercado Actual' no existen en el archivo CSV.")

# Fecha de inicio y fecha actual
fecha_inicio = datetime(2024, 1, 1)
fecha_hoy = datetime.today()

# Función para graficar el crecimiento en valor de mercado
def graficar_jugador():
    # Pedir el nombre del jugador al usuario
    nombre_jugador = input("Escribe el nombre del jugador: ")

    # Filtrar al jugador
    jugador = data[data['Nombre'] == nombre_jugador]

    if not jugador.empty:
        # Obtener valores inicial y actual
        valor_inicial = jugador['Valor de Mercado en 01/01/2024'].values[0]
        valor_actual = jugador['Valor de Mercado Actual'].values[0]

        # Generar fechas mensuales desde el inicio hasta hoy
        fechas = pd.date_range(fecha_inicio, fecha_hoy, freq='MS')  # 'MS' para inicio de cada mes
        valores = [valor_inicial + (valor_actual - valor_inicial) * (i / (len(fechas) - 1)) for i in range(len(fechas))]

        # Crear la gráfica de barras
        fig = go.Figure(data=go.Bar(x=fechas, y=valores))
        fig.update_layout(title=f'Evolución del Valor de Mercado de {nombre_jugador}',
                          xaxis_title='Fecha',
                          yaxis_title='Valor de Mercado (€)',
                          xaxis=dict(tickformat="%Y-%m"))
        fig.show()
    else:
        print("Jugador no encontrado.")

# Llamar a la función
graficar_jugador()

import plotly.graph_objects as go
import pandas as pd

# Enlace directo al archivo raw en GitHub
file_path = 'https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv'
```
```

```
# Cargar el archivo CSV desde GitHub
df = pd.read_csv(file_path)

# Función para convertir los valores de mercado a euros completos (enteros)
def convertir_valor(valor):
    if isinstance(valor, str):
        if "mil €" in valor:
            return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)  # Convierte a euros completos
        elif "mill. €" in valor:
            return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)  # Convierte a euros completos
        else:
            # Eliminar otros caracteres no numéricos (como € o comas) y convertir a float
            valor = valor.replace("€", "").replace(",", "").strip()
            try:
                return float(valor)
            except ValueError:
                return None
    return valor

# Función para convertir las columnas a valores numéricos
def convertir_a_numerico(df, columnas):
    for columna in columnas:
        df[columna] = df[columna].apply(convertir_valor)

# Convertir las columnas a valores numéricos
convertir_a_numerico(df, ['Valor de Mercado en 01/01/2024', 'Valor de Mercado Actual'])

# Calcular el porcentaje de aumento para cada jugador
df['Porcentaje Aumento'] = ((df['Valor de Mercado Actual'] - df['Valor de Mercado en 01/01/2024'])
                            / df['Valor de Mercado en 01/01/2024']) * 100

# Filtrar jugadores con datos válidos
df_filtrado = df.dropna(subset=['Porcentaje Aumento'])

# Crear la gráfica de barras
fig = go.Figure(data=go.Bar(x=df_filtrado['Nombre'], y=df_filtrado['Porcentaje Aumento']))
fig.update_layout(title='Porcentaje de Aumento en el Valor de Mercado de los Jugadores durante la Temporada',
                  xaxis_title='Jugadores',
                  yaxis_title='Porcentaje de Aumento (%)',
                  xaxis_tickangle=-45)  # Inclina los nombres de los jugadores

# Mostrar la gráfica
fig.show()

"""

---


#CODIGO DEFINITIVO PARA AÑADIR DATOS AL CSV DE LALIGA DEL GITHUB (SE ELIMINARIAN LAS IMAGENES)

Para añadir datos a un archivo CSV en GitHub mediante un script automatizado en Python, puedes usar la API de GitHub. Esto requiere autenticación con un Token

Aquí tienes una guía paso a paso para implementar este método:

#Pasos para Configurar y Ejecutar el Código
1. Generar un Token de Acceso Personal en GitHub:

* Entra a https://github.com/settings/tokens.
* Haz clic en Generate new token.
* Asigna permisos de repo para tener acceso de lectura y escritura al repositorio.
* Copia el token, ya que lo necesitarás para autenticarte en el script.
2. Instalar la Librería requests:

* Si no la tienes instalada, instálala usando:
* pip install requests
3. Código para Modificar el CSV en GitHub:

* Este código cargará el archivo CSV desde GitHub, añadirá nuevos datos y subirá el archivo actualizado de vuelta al repositorio."""

import pandas as pd
import requests
import base64
import json

# Configuración del repositorio
owner = "AndersonP444"
repo = "PROYECTO-SIC-JAKDG"
file_path = "valores_mercado_actualizados (3).csv"
branch = "main"
github_token = "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"  # Reemplaza con tu token personal

# URL del archivo CSV en GitHub
github_api_url = f"https://api.github.com/repos/{owner}/{repo}/contents/{file_path}"

# Función para cargar el archivo CSV desde GitHub
def cargar_csv_github():
    headers = {"Authorization": f"token {github_token}"}
    response = requests.get(github_api_url, headers=headers)
    if response.status_code == 200:
        file_content = base64.b64decode(response.json()["content"])
        with open("temp.csv", "wb") as file:
            file.write(file_content)
        return pd.read_csv("temp.csv"), response.json()["sha"]
    else:
        raise Exception(f"Error al cargar archivo: {response.status_code}")

# Función para subir el archivo CSV modificado a GitHub
def subir_csv_github(df, sha):
    # Convertir el DataFrame a CSV en base64
    csv_data = df.to_csv(index=False).encode("utf-8")
    b64_content = base64.b64encode(csv_data).decode("utf-8")

    # Crear el commit de actualización
    commit_message = "Actualización de datos de mercado"
    headers = {"Authorization": f"token {github_token}"}
```

```
    payload = {
        "message": commit_message,
        "content": b64_content,
        "sha": sha,
        "branch": branch
    }

    # Realizar la solicitud PUT para actualizar el archivo
    response = requests.put(github_api_url, headers=headers, data=json.dumps(payload))
    if response.status_code == 200:
        print("Archivo actualizado exitosamente en GitHub.")
    else:
        raise Exception(f"Error al subir archivo: {response.status_code}")

# Cargar el CSV actual de GitHub
data, sha = cargar_csv_github()

# Solicitar datos del usuario y añadir al DataFrame
while True:
    # Recibir datos del usuario en un solo input
    datos_jugador = input("\nIngrese los datos del jugador en el siguiente formato:\nNombre, Posición, Nacionalidad, Edad, Equipo, Valor de Mercado en 01/01/20

    # Separar los datos por comas
    datos_separados = [dato.strip() for dato in datos_jugador.split(",")]

    # Asegurarse de que la longitud de los datos sea correcta
    if len(datos_separados) != 9:
        print("Error: Asegúrese de ingresar exactamente 9 campos separados por comas.")
        continue

    # Asignar cada valor a su respectiva variable y mantener como str
    nombre, posicion, nacionalidad, edad, equipo, valor_inicial, valor_actual, incremento_porcentaje, incremento_euros = datos_separados

    # Formatear los valores
    valor_inicial = f"{valor_inicial} €"
    valor_actual = f"{valor_actual} €"
    incremento_porcentaje = f"{incremento_porcentaje} %"
    incremento_euros = f"{incremento_euros} €"

    # Añadir el nuevo jugador al DataFrame
    nuevo_jugador = pd.DataFrame([{
        "Nombre": nombre,
        "Posición": posicion,
        "Nacionalidad": nacionalidad,
        "Edad": edad,
        "Equipo": equipo,
        "Valor de Mercado en 01/01/2024": valor_inicial,
        "Valor de Mercado Actual": valor_actual,
        "Incremento (%)": incremento_porcentaje,
        "Incremento (€)": incremento_euros
    }])
    data = pd.concat([data, nuevo_jugador], ignore_index=True)

    # Preguntar si el usuario desea añadir otro jugador
    agregar_mas = input("¿Desea agregar otro jugador? (si/no): ").lower()
    if agregar_mas != 'si':
        break

# Subir el archivo actualizado a GitHub
subir_csv_github(data, sha)

"""

---


#CODIGO DEFINITIVO PARA AÑADIR DATOS AL CSV DE LA BUNDESLIGA DEL GITHUB (SE ELIMINARIAN LAS IMAGENES)

Para añadir datos a un archivo CSV en GitHub mediante un script automatizado en Python, puedes usar la API de GitHub. Esto requiere autenticación con un Token

Aquí tienes una guía paso a paso para implementar este método:

#Pasos para Configurar y Ejecutar el Código
1. Generar un Token de Acceso Personal en GitHub:

* Entra a https://github.com/settings/tokens.
* Haz clic en Generate new token.
* Asigna permisos de repo para tener acceso de lectura y escritura al repositorio.
* Copia el token, ya que lo necesitarás para autenticarte en el script.
2. Instalar la Librería requests:

* Si no la tienes instalada, instálala usando:
* pip install requests
3. Código para Modificar el CSV en GitHub:

* Este código cargará el archivo CSV desde GitHub, añadirá nuevos datos y subirá el archivo actualizado de vuelta al repositorio."""

import pandas as pd
import requests
import base64
import json

# Configuración del repositorio
owner = "AndersonP444"
repo = "PROYECTO-SIC-JAKDG"
file_path = "valores_mercado_bundesliga_actualizado_v2.csv"
branch = "main"
github_token = "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"  # Reemplaza con tu token personal

# URL del archivo CSV en GitHub
github_api_url = f"https://api.github.com/repos/{owner}/{repo}/contents/{file_path}"

# Función para cargar el archivo CSV desde GitHub
def cargar_csv_github():
    headers = {"Authorization": f"token {github_token}"}
    response = requests.get(github_api_url, headers=headers)
```

```
    if response.status_code == 200:
        file_content = base64.b64decode(response.json()["content"])
        with open("temp.csv", "wb") as file:
            file.write(file_content)
        return pd.read_csv("temp.csv"), response.json()["sha"]
    else:
        raise Exception(f"Error al cargar archivo: {response.status_code}")

# Función para subir el archivo CSV modificado a GitHub
def subir_csv_github(df, sha):
    # Convertir el DataFrame a CSV en base64
    csv_data = df.to_csv(index=False).encode("utf-8")
    b64_content = base64.b64encode(csv_data).decode("utf-8")

    # Crear el commit de actualización
    commit_message = "Actualización de datos de mercado"
    headers = {"Authorization": f"token {github_token}"}
    payload = {
        "message": commit_message,
        "content": b64_content,
        "sha": sha,
        "branch": branch
    }

    # Realizar la solicitud PUT para actualizar el archivo
    response = requests.put(github_api_url, headers=headers, data=json.dumps(payload))
    if response.status_code == 200:
        print("Archivo actualizado exitosamente en GitHub.")
    else:
        raise Exception(f"Error al subir archivo: {response.status_code}")

# Cargar el CSV actual de GitHub
data, sha = cargar_csv_github()

# Solicitar datos del usuario y añadir al DataFrame
while True:
    # Recibir datos del usuario en un solo input
    datos_jugador = input("\nIngrese los datos del jugador en el siguiente formato:\nNombre, Posición, Nacionalidad, Edad, Equipo, Valor de Mercado en 01/01/20

    # Separar los datos por comas
    datos_separados = [dato.strip() for dato in datos_jugador.split(",")]

    # Asegurarse de que la longitud de los datos sea correcta
    if len(datos_separados) != 9:
        print("Error: Asegúrese de ingresar exactamente 9 campos separados por comas.")
        continue

    # Asignar cada valor a su respectiva variable y mantener como str
    nombre, posicion, nacionalidad, edad, equipo, valor_inicial, valor_actual, incremento_porcentaje, incremento_euros = datos_separados

    # Formatear los valores
    valor_inicial = f"{valor_inicial} €"
    valor_actual = f"{valor_actual} €"
    incremento_porcentaje = f"{incremento_porcentaje} %"
    incremento_euros = f"{incremento_euros} €"

    # Añadir el nuevo jugador al DataFrame
    nuevo_jugador = pd.DataFrame([{
        "Nombre": nombre,
        "Posición": posicion,
        "Nacionalidad": nacionalidad,
        "Edad": edad,
        "Equipo": equipo,
        "Valor de Mercado en 01/01/2024": valor_inicial,
        "Valor de Mercado Actual": valor_actual,
        "Incremento (%)": incremento_porcentaje,
        "Incremento (€)": incremento_euros
    }])
    data = pd.concat([data, nuevo_jugador], ignore_index=True)

    # Preguntar si el usuario desea añadir otro jugador
    agregar_mas = input("¿Desea agregar otro jugador? (si/no): ").lower()
    if agregar_mas != 'si':
        break

# Subir el archivo actualizado a GitHub
subir_csv_github(data, sha)

"""#Explicación del Código
1. Cargar CSV de GitHub:

* La función cargar_csv_github descarga el archivo CSV desde el repositorio de GitHub usando la API y lo carga en un DataFrame.
* Se obtiene también el valor sha del archivo, que es necesario para realizar una actualización en GitHub.
2. Modificar los Datos del CSV:

* El código solicita al usuario los datos del jugador en un solo input.
Añade los datos al DataFrame como una fila nueva, manteniendo el formato de texto para los valores monetarios y de porcentaje.
3. Subir CSV Modificado a GitHub:

* La función subir_csv_github convierte el DataFrame actualizado a formato CSV en base64.
* Envía una solicitud PUT a la API de GitHub con el archivo codificado en base64 para actualizar el CSV en el repositorio.
* Incluye el sha del archivo para que GitHub identifique qué versión del archivo está siendo actualizada.
#Notas y Advertencias
* Almacenamiento del Token: Evita incluir el token de GitHub directamente en tu código. Usa variables de entorno para mayor seguridad.
* Rate Limiting: La API de GitHub tiene límites de tasa para solicitudes autenticadas. Evita hacer demasiadas actualizaciones en poco tiempo para no superar es

Este código automatiza la descarga, modificación y actualización del archivo CSV en GitHub, permitiéndote realizar todo el proceso en un único flujo de trabajo

---
#ELIMINAR UNA FILA MEDIANTE SU INDICE (POR SI SE REPITE UN JUGADOR O UN ERROR)
NOTA: CUANDO SE VAYA A ELIMINAR UNA FILA, LE RESTAS 2 AL INDICE. EJEMPLO FILA A ELIMINAR 62, COLOCAS 60 Y SE ELIMINARA LA 62

Para eliminar una fila en un archivo CSV de GitHub por su índice y guardar el cambio en el archivo original, el proceso consiste en:

1. Descargar el archivo CSV desde GitHub.
2. Eliminar la fila correspondiente del DataFrame.
```

```
3. Subir el archivo actualizado nuevamente a GitHub utilizando la API.
"""

import pandas as pd
import requests
import base64
import json

# Configuración del repositorio y token de GitHub
owner = "AndersonP444"
repo = "PROYECTO-SIC-JAKDG"
file_path = "valores_mercado_actualizados (3).csv"
branch = "main"
github_token = "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"  # Reemplaza con tu token personal

# URL del archivo CSV en GitHub
github_api_url = f"https://api.github.com/repos/{owner}/{repo}/contents/{file_path}"

# Función para cargar el archivo CSV desde GitHub
def cargar_csv_github():
    headers = {"Authorization": f"token {github_token}"}
    response = requests.get(github_api_url, headers=headers)
    if response.status_code == 200:
        file_content = base64.b64decode(response.json()["content"])
        with open("temp.csv", "wb") as file:
            file.write(file_content)
        return pd.read_csv("temp.csv"), response.json()["sha"]
    else:
        raise Exception(f"Error al cargar archivo: {response.status_code}")

# Función para subir el archivo CSV modificado a GitHub
def subir_csv_github(df, sha):
    # Convertir el DataFrame a CSV en base64
    csv_data = df.to_csv(index=False).encode("utf-8")
    b64_content = base64.b64encode(csv_data).decode("utf-8")

    # Crear el commit de actualización
    commit_message = "Actualización de datos tras eliminar una fila"
    headers = {"Authorization": f"token {github_token}"}
    payload = {
        "message": commit_message,
        "content": b64_content,
        "sha": sha,
        "branch": branch
    }

    # Realizar la solicitud PUT para actualizar el archivo
    response = requests.put(github_api_url, headers=headers, data=json.dumps(payload))
    if response.status_code == 200:
        print("Archivo actualizado exitosamente en GitHub.")
    else:
        raise Exception(f"Error al subir archivo: {response.status_code}")

# Cargar el CSV actual de GitHub
data, sha = cargar_csv_github()

# Solicitar el índice de la fila a eliminar
try:
    index_to_remove = int(input(f"Ingrese el índice de la fila que desea eliminar (0 a {len(data) - 1}): "))
    if 0 <= index_to_remove < len(data):
        data = data.drop(index_to_remove).reset_index(drop=True)  # Eliminar y reajustar el índice
        subir_csv_github(data, sha)  # Subir el CSV actualizado a GitHub
    else:
        print(f"Índice fuera de rango. Debe estar entre 0 y {len(data) - 1}.")
except ValueError:
    print("Error: Por favor, ingrese un número válido para el índice.")

"""##Explicación del Código
1. Cargar CSV de GitHub:

* La función cargar_csv_github descarga el archivo CSV en formato base64 desde GitHub, lo decodifica y lo carga en un DataFrame.
* Se obtiene el sha del archivo, necesario para hacer una actualización en GitHub.
2. Eliminar Fila por Índice:

* Se solicita al usuario el índice de la fila a eliminar.
* data.drop(index_to_remove).reset_index(drop=True) elimina la fila especificada y reorganiza los índices del DataFrame.
3. Subir CSV Modificado a GitHub:

* La función subir_csv_github convierte el DataFrame actualizado a formato CSV y lo codifica en base64.
* Envía una solicitud PUT a la API de GitHub para actualizar el archivo en el repositorio, especificando el sha para confirmar la versión.
##Notas
* Token de GitHub: Para mayor seguridad, almacena tu token en una variable de entorno.
* Límites de Solicitud: La API de GitHub tiene límites de tasa; evitar muchas solicitudes en corto tiempo.

Este código permite gestionar y sincronizar cambios en GitHub de manera automatizada, eliminando filas por índice en un archivo CSV y actualizando el archivo d

---
#  Gráfica que compara los valores de mercado iniciales y actuales de dos jugadores seleccionados por el usuario.
"""

import plotly.graph_objects as go
import pandas as pd

# Enlace directo al archivo raw en GitHub
file_path = 'https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv'

# Cargar el archivo CSV desde GitHub
data = pd.read_csv(file_path)

# Función para convertir los valores de mercado a euros completos (enteros)
def convertir_valor(valor):
    if isinstance(valor, str):
        if "mil €" in valor:
            return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)  # Convierte a euros completos
        elif "mill. €" in valor:
            return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)  # Convierte a euros completos
```

```
    return None

# Verificar que las columnas existen en el DataFrame antes de aplicar la función
if 'Valor de Mercado en 01/01/2024' in data.columns and 'Valor de Mercado Actual' in data.columns:
    # Aplicar la función de conversión a las columnas de valores de mercado
    data["Valor de Mercado en 01/01/2024"] = data["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
    data["Valor de Mercado Actual"] = data["Valor de Mercado Actual"].apply(convertir_valor)
else:
    print("Las columnas 'Valor de Mercado en 01/01/2024' o 'Valor de Mercado Actual' no existen en el archivo CSV.")

# Función para graficar la comparación de valores de mercado entre dos jugadores
def comparar_jugadores():
    # Pedir el nombre de los jugadores al usuario
    jugador1 = input("Escribe el nombre del primer jugador: ")
    jugador2 = input("Escribe el nombre del segundo jugador: ")

    # Filtrar los datos de los jugadores seleccionados
    jugador_data = data[data['Nombre'].isin([jugador1, jugador2])]

    if len(jugador_data) == 2:
        # Obtener los nombres y valores de mercado inicial y actual
        nombres = jugador_data['Nombre'].values
        valores_iniciales = jugador_data['Valor de Mercado en 01/01/2024'].values
        valores_actuales = jugador_data['Valor de Mercado Actual'].values

        # Crear la gráfica de barras para comparación
        fig = go.Figure()
        fig.add_trace(go.Bar(name='Valor Inicial', x=nombres, y=valores_iniciales, marker_color='lightblue'))
        fig.add_trace(go.Bar(name='Valor Actual', x=nombres, y=valores_actuales, marker_color='darkblue'))

        # Configurar el diseño de la gráfica
        fig.update_layout(title='Comparación de Valor de Mercado Inicial y Actual',
                          xaxis_title='Jugadores',
                          yaxis_title='Valor de Mercado (€)',
                          barmode='group')
        fig.show()
    else:
        print("No se encontraron ambos jugadores en los datos. Verifica los nombres e inténtalo de nuevo.")

# Llamar a la función
comparar_jugadores()

"""---
# Gráfica que indica el incremento o decremento de dos jugadores en una tabla (desde el 01/01/2024 hasta su valor actual)
"""

import plotly.graph_objects as go
import pandas as pd

# Enlace directo al archivo raw en GitHub
file_path = 'https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv'

# Cargar el archivo CSV desde GitHub
data = pd.read_csv(file_path)

# Función para convertir los valores de mercado a euros completos (enteros)
def convertir_valor(valor):
    if isinstance(valor, str):
        if "mil €" in valor:
            return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)  # Convierte a euros completos
        elif "mill. €" in valor:
            return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)  # Convierte a euros completos
    return None

# Verificar que las columnas existen en el DataFrame antes de aplicar la función
if 'Valor de Mercado en 01/01/2024' in data.columns and 'Valor de Mercado Actual' in data.columns:
    # Aplicar la función de conversión a las columnas de valores de mercado
    data["Valor de Mercado en 01/01/2024"] = data["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
    data["Valor de Mercado Actual"] = data["Valor de Mercado Actual"].apply(convertir_valor)
else:
    print("Las columnas 'Valor de Mercado en 01/01/2024' o 'Valor de Mercado Actual' no existen en el archivo CSV.")

# Función para graficar la comparación de valores de mercado entre dos jugadores
def comparar_jugadores():
    # Pedir el nombre de los jugadores al usuario
    jugador1 = input("Escribe el nombre del primer jugador: ")
    jugador2 = input("Escribe el nombre del segundo jugador: ")

    # Filtrar los datos de los jugadores seleccionados
    jugador_data = data[data['Nombre'].isin([jugador1, jugador2])]

    if len(jugador_data) == 2:
        # Obtener los nombres y valores de mercado inicial y actual
        nombres = jugador_data['Nombre'].values
        valores_iniciales = jugador_data['Valor de Mercado en 01/01/2024'].values
        valores_actuales = jugador_data['Valor de Mercado Actual'].values

        # Crear la gráfica de líneas para comparación
        fig = go.Figure()

        fig.add_trace(go.Scatter(
            x=['01/01/2024', 'Actual'],
            y=[valores_iniciales[0], valores_actuales[0]],
            mode='lines+markers+text',
            name=nombres[0],
            text=[valores_iniciales[0], valores_actuales[0]],
            textposition='top center',
            line=dict(shape='spline', smoothing=1.3, color='royalblue', width=4),
            marker=dict(size=10)
        ))

        fig.add_trace(go.Scatter(
            x=['01/01/2024', 'Actual'],
            y=[valores_iniciales[1], valores_actuales[1]],
            mode='lines+markers+text',
            name=nombres[1],
```

```
            text=[valores_iniciales[1], valores_actuales[1]],
            textposition='top center',
            line=dict(shape='spline', smoothing=1.3, color='firebrick', width=4),
            marker=dict(size=10)
        ))

        # Configurar el diseño de la gráfica
        fig.update_layout(
            title={
                'text': 'Crecimiento del Valor de Mercado de los Jugadores',
                'y':0.9,
                'x':0.5,
                'xanchor': 'center',
                'yanchor': 'top'
            },
            xaxis_title='Fecha',
            yaxis_title='Valor de Mercado (€)',
            template='plotly_dark',  # Tema oscuro
            paper_bgcolor='rgb(30, 30, 30)',  # Color de fondo del papel
            plot_bgcolor='rgb(30, 30, 30)',  # Color de fondo del gráfico
            font=dict(
                family="Courier New, monospace",
                size=14,
                color="white"
            )
        )

        fig.show()
    else:
        print("No se encontraron ambos jugadores en los datos. Verifica los nombres e inténtalo de nuevo.")

# Llamar a la función
comparar_jugadores()



import pandas as pd
import requests
import base64
import json

# Configuración del repositorio
owner = "AndersonP444"
repo = "PROYECTO-SIC-JAKDG"
file_path = "valores_mercado_actualizados.csv"
branch = "main"
github_token = "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"  # Reemplaza con tu token personal

# URL del archivo CSV en GitHub
github_api_url = f"https://api.github.com/repos/{owner}/{repo}/contents/{file_path}"

# Función para cargar el archivo CSV desde GitHub
def cargar_csv_github():
    headers = {"Authorization": f"token {github_token}"}
    response = requests.get(github_api_url, headers=headers)
    if response.status_code == 200:
        file_content = base64.b64decode(response.json()["content"])
        with open("temp.csv", "wb") as file:
            file.write(file_content)
        return pd.read_csv("temp.csv"), response.json()["sha"]
    else:
        raise Exception(f"Error al cargar archivo: {response.status_code}")

# Función para subir el archivo CSV modificado a GitHub
def subir_csv_github(df, sha):
    # Convertir el DataFrame a CSV en base64
    csv_data = df.to_csv(index=False).encode("utf-8")
    b64_content = base64.b64encode(csv_data).decode("utf-8")

    # Crear el commit de actualización
    commit_message = "Eliminación de la columna 'Jugador'"
    headers = {"Authorization": f"token {github_token}"}
    payload = {
        "message": commit_message,
        "content": b64_content,
        "sha": sha,
        "branch": branch
    }

    # Realizar la solicitud PUT para actualizar el archivo
    response = requests.put(github_api_url, headers=headers, data=json.dumps(payload))
    if response.status_code == 200:
        print("Archivo actualizado exitosamente en GitHub.")
    else:
        raise Exception(f"Error al subir archivo: {response.status_code}")

# Cargar el CSV actual de GitHub
data, sha = cargar_csv_github()

# Eliminar la columna 'Jugador' si existe
if 'Jugador' in data.columns:
    data = data.drop(columns=['Jugador'])
    print("Columna 'Jugador' eliminada con éxito.")
else:
    print("La columna 'Jugador' no existe en el archivo.")

# Subir el archivo actualizado a GitHub
subir_csv_github(data, sha)

import pandas as pd
import requests
import base64
import json
from IPython.display import display, HTML

# Configuración del repositorio
```

```python
owner = "AndersonP444"
repo = "PROYECTO-SIC-JAKDG"
file_path = "valores_mercado_actualizados.csv"
branch = "main"
github_token = "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"  # Reemplaza con tu token personal

# URL del archivo CSV en GitHub
github_api_url = f"https://api.github.com/repos/{owner}/{repo}/contents/{file_path}"

# Función para cargar el archivo CSV desde GitHub
def cargar_csv_github():
    headers = {"Authorization": f"token {github_token}"}
    response = requests.get(github_api_url, headers=headers)
    if response.status_code == 200:
        file_content = base64.b64decode(response.json()["content"])
        with open("temp.csv", "wb") as file:
            file.write(file_content)
        return pd.read_csv("temp.csv"), response.json()["sha"]
    else:
        raise Exception(f"Error al cargar archivo: {response.status_code}")

# Función para subir el archivo CSV modificado a GitHub
def subir_csv_github(df, sha):
    # Convertir el DataFrame a CSV en base64
    csv_data = df.to_csv(index=False).encode("utf-8")
    b64_content = base64.b64encode(csv_data).decode("utf-8")

    # Crear el commit de actualización
    commit_message = "Actualización de imágenes de jugadores"
    headers = {"Authorization": f"token {github_token}"}
    payload = {
        "message": commit_message,
        "content": b64_content,
        "sha": sha,
        "branch": branch
    }

    # Realizar la solicitud PUT para actualizar el archivo
    response = requests.put(github_api_url, headers=headers, data=json.dumps(payload))
    if response.status_code == 200:
        print("Archivo actualizado exitosamente en GitHub.")
    else:
        raise Exception(f"Error al subir archivo: {response.status_code}")

# Cargar el CSV actual de GitHub
data, sha = cargar_csv_github()

# Asegurarse de que la columna "Jugador" esté al principio
if 'Jugador' not in data.columns:
    data.insert(0, 'Jugador', '')  # Crear la columna "Jugador" al principio del DataFrame

# Solicitar la URL de la imagen para cada jugador existente
for i, row in data.iterrows():
    nombre = row["Nombre"]

    # Preguntar por la URL de la imagen
    url_imagen = input(f"Ingrese la URL de la imagen para {nombre} (o escriba 'stop' para finalizar): ")

    if url_imagen.lower() == "stop":
        print("Proceso detenido por el usuario.")
        break  # Detener el ciclo si el usuario ingresa 'stop'

    if url_imagen:
        data.at[i, "Jugador"] = url_imagen  # Asignar la URL de la imagen a la columna "Jugador"

# Función para mostrar el DataFrame con las imágenes renderizadas en Colab
def mostrar_imagenes(data):
    # Crear una nueva columna en el DataFrame con etiquetas HTML para mostrar las imágenes
    data['Jugador'] = data['Jugador'].apply(lambda url: f'<img src="{url}" width="50">')

    # Mostrar el DataFrame con las imágenes renderizadas
    display(HTML(data.to_html(escape=False)))  # escape=False permite que se rendericen las imágenes HTML

# Llamar a la función para mostrar las imágenes en el DataFrame
mostrar_imagenes(data)

# Subir el archivo actualizado a GitHub
subir_csv_github(data, sha)

import pandas as pd
import requests
import base64
import json
from IPython.display import display, HTML

# Configuración del repositorio
owner = "AndersonP444"
repo = "PROYECTO-SIC-JAKDG"
file_path = "valores_mercado_bundesliga.csv"
branch = "main"
github_token = "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"  # Reemplaza con tu token personal

# URL del archivo CSV en GitHub
github_api_url = f"https://api.github.com/repos/{owner}/{repo}/contents/{file_path}"

# Función para cargar el archivo CSV desde GitHub
def cargar_csv_github():
    headers = {"Authorization": f"token {github_token}"}
    response = requests.get(github_api_url, headers=headers)
    if response.status_code == 200:
        file_content = base64.b64decode(response.json()["content"])
        with open("temp.csv", "wb") as file:
            file.write(file_content)
        return pd.read_csv("temp.csv"), response.json()["sha"]
    else:
        raise Exception(f"Error al cargar archivo: {response.status_code}")
```

```
# Función para subir el archivo CSV modificado a GitHub
def subir_csv_github(df, sha):
    # Convertir el DataFrame a CSV en base64
    csv_data = df.to_csv(index=False).encode("utf-8")
    b64_content = base64.b64encode(csv_data).decode("utf-8")

    # Crear el commit de actualización
    commit_message = "Actualización de imágenes de jugadores"
    headers = {"Authorization": f"token {github_token}"}
    payload = {
        "message": commit_message,
        "content": b64_content,
        "sha": sha,
        "branch": branch
    }

    # Realizar la solicitud PUT para actualizar el archivo
    response = requests.put(github_api_url, headers=headers, data=json.dumps(payload))
    if response.status_code == 200:
        print("Archivo actualizado exitosamente en GitHub.")
    else:
        raise Exception(f"Error al subir archivo: {response.status_code}")

# Cargar el CSV actual de GitHub
data, sha = cargar_csv_github()

# Asegurarse de que la columna "Jugador" esté al principio
if 'Jugador' not in data.columns:
    data.insert(0, 'Jugador', '')  # Crear la columna "Jugador" al principio del DataFrame

# Solicitar la URL de la imagen para cada jugador existente
for i, row in data.iterrows():
    nombre = row["Nombre"]

    # Preguntar por la URL de la imagen
    url_imagen = input(f"Ingrese la URL de la imagen para {nombre} (o escriba 'stop' para finalizar): ")

    if url_imagen.lower() == "stop":
        print("Proceso detenido por el usuario.")
        break  # Detener el ciclo si el usuario ingresa 'stop'

    if url_imagen:
        data.at[i, "Jugador"] = url_imagen  # Asignar la URL de la imagen a la columna "Jugador"

# Función para mostrar el DataFrame con las imágenes renderizadas en Colab
def mostrar_imagenes(data):
    # Crear una nueva columna en el DataFrame con etiquetas HTML para mostrar las imágenes
    data['Jugador'] = data['Jugador'].apply(lambda url: f'<img src="{url}" width="50">')

    # Mostrar el DataFrame con las imágenes renderizadas
    display(HTML(data.to_html(escape=False)))  # escape=False permite que se rendericen las imágenes HTML

# Llamar a la función para mostrar las imágenes en el DataFrame
mostrar_imagenes(data)

# Subir el archivo actualizado a GitHub
subir_csv_github(data, sha)

"""#BOT
Para construir un bot que extraiga y procese datos de los archivos valores_mercado_actualizados.csv y valores_mercado_bundesliga.csv desde tu repositorio de Gi

A continuación, te muestro cómo implementar este bot que descarga y muestra la información relevante desde los archivos CSV en GitHub.

##Pasos para Crear el Bot
1. Acceder a los Archivos en GitHub:
* Usaremos URLs directos a los archivos en formato raw para descargarlos.
2. Leer y Procesar los Archivos CSV:
* Cargaremos los archivos CSV en DataFrames usando pandas.
3. Realizar Análisis de Datos:
* Implementaremos un ejemplo básico que muestra información relevante, como la lista de jugadores, el valor de mercado, o cualquier otra información que necesi
##Instalación de Dependencias
Si aún no tienes instaladas las dependencias necesarias, puedes instalarlas con:
"""

pip install pandas requests

"""##Código para Crear el Bot
Aquí tienes el código en Python para implementar este bot:
"""

import pandas as pd
import requests
import io
# URLs a los archivos CSV en GitHub
urls = {
    "La Liga": "https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv",
    "Bundesliga": "https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_bundesliga.csv"
}

# Función para cargar un archivo CSV desde GitHub
def cargar_datos_csv(url):
    response = requests.get(url)
    if response.status_code == 200:
        # Usa io.StringIO en lugar de pd.compat.StringIO
        data = pd.read_csv(io.StringIO(response.text))
        print(f"Datos cargados desde {url}")
        return data
    else:
        print(f"Error al acceder al archivo: {response.status_code}")
        return None

# Función para mostrar un resumen de los datos de la liga
def mostrar_resumen_liga(datos, liga_nombre):
    print(f"\n--- {liga_nombre} ---")
    if datos is not None:
```

```
        print(f"Total de registros en {liga_nombre}: {len(datos)}\n")

        # Mostrar las primeras filas
        print("Primeros 5 registros:")
        print(datos.head(), "\n")

        # Mostrar estadísticas de valores de mercado si existen las columnas correspondientes
        if "Valor de Mercado Actual" in datos.columns:
            print("Estadísticas de Valor de Mercado Actual:")
            print(datos["Valor de Mercado Actual"].describe())
        else:
            print("No se encontró la columna 'Valor de Mercado Actual' en el archivo.")
    else:
        print("No se pudo cargar los datos.")

# Cargar y mostrar los datos de cada liga
for liga, url in urls.items():
    datos_liga = cargar_datos_csv(url)
    mostrar_resumen_liga(datos_liga, liga)

"""Para mostrar las tablas como un display HTML en Python, puedes usar el entorno de Jupyter Notebook junto con pandas.DataFrame.to_html() y IPython.display.di

Si estás trabajando fuera de un entorno de notebook y necesitas una vista previa HTML en un navegador, generaremos el HTML directamente en un archivo o bien co

##Método con Jupyter Notebook (si estás usando Jupyter)
Aquí te muestro cómo modificar el código para ver el contenido en formato HTML usando IPython.display.display.
"""

import pandas as pd
import requests
import io
from IPython.display import display, HTML

# URLs a los archivos CSV en GitHub
urls = {
    "La Liga": "https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_actualizados.csv",
    "Bundesliga": "https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/valores_mercado_bundesliga.csv"
}

# Función para cargar un archivo CSV desde GitHub
def cargar_datos_csv(url):
    response = requests.get(url)
    if response.status_code == 200:
        data = pd.read_csv(io.StringIO(response.text))
        print(f"Datos cargados desde {url}")
        return data
    else:
        print(f"Error al acceder al archivo: {response.status_code}")
        return None

# Mostrar datos como HTML en el notebook
def mostrar_tabla_html(datos, liga_nombre):
    if datos is not None:
        display(HTML(f"<h2>{liga_nombre}</h2>"))
        display(HTML(datos.to_html(index=False)))
    else:
        display(HTML(f"<h2>{liga_nombre}</h2><p>Error al cargar los datos.</p>"))

# Cargar y mostrar los datos de cada liga
for liga, url in urls.items():
    datos_liga = cargar_datos_csv(url)
    mostrar_tabla_html(datos_liga, liga)

"""##Explicación
Jupyter Notebook (display con HTML):

Esta opción te permite mostrar directamente las tablas HTML en un notebook usando display(HTML(...)).

##Explicación del Código
1. URLs de Archivos en GitHub:
* Definimos un diccionario urls con el nombre de cada liga y la URL del archivo CSV en GitHub. Usamos la URL raw para acceder a los datos directamente.
2. Función cargar_datos_csv:
* Esta función toma la URL de un archivo CSV, hace una solicitud GET para descargar el archivo, y si tiene éxito, lo carga en un DataFrame de pandas.
* Imprime un mensaje de éxito o error según el estado de la solicitud.
3. Función mostrar_resumen_liga:
* Esta función toma un DataFrame de pandas y un nombre de liga.
* Imprime un resumen de los datos, incluyendo el total de registros y los primeros cinco registros.
* Si existe la columna Valor de Mercado Actual, muestra estadísticas descriptivas de esta columna.
4. Iteración sobre las Ligas:
* Iteramos sobre el diccionario urls, cargamos los datos de cada liga y mostramos el resumen usando mostrar_resumen_liga.
##Ejecución del Código
Este código descargará y analizará los archivos CSV de tu repositorio en GitHub, mostrando los primeros registros y un resumen del valor de mercado si está dis

---
#Coordenadas de la localizacion de los estadios de los equipos de LaLiga

Este código utiliza la librería Folium para crear un mapa interactivo centrado en el estadio de un club de fútbol, según las coordenadas proporcionadas. El cód

####1. Definición de los Datos de los Clubes
```
clubs = {
    "Athletic Club": {"estadio": "San Mamés, Bilbao", "coordenadas": (43.2641, -2.9495)},
    "Atlético de Madrid": {"estadio": "Wanda Metropolitano, Madrid", "coordenadas": (40.4362, -3.5995)},
    ...
}
```
Aquí, se define un diccionario clubs que contiene la información de varios clubes de fútbol. Para cada club, se guarda:

* El nombre del estadio.
* Las coordenadas geográficas del estadio (en formato (latitud, longitud)).

Por ejemplo, para el club "Athletic Club", el estadio es "San Mamés" en Bilbao, y sus coordenadas son (43.2641, -2.9495).

####2. Función obtener_coordenadas
```
def obtener_coordenadas(club):
```
```

```
    if club in clubs:
        return clubs[club]['coordenadas']
    else:
        return None
```
Esta función toma como argumento el nombre de un club y verifica si ese club está presente en el diccionario clubs. Si el club existe, devuelve las coordenadas

####3. Entrada del Usuario
```club = input("Introduce el nombre del club: ")```

Se solicita al usuario que ingrese el nombre de un club de fútbol. El nombre del club se guarda en la variable club.

####4. Verificación y Creación del Mapa
```
if club in clubs:
    estadio = clubs[club]['estadio']
    coordenadas = obtener_coordenadas(club)

    if coordenadas:
        mapa = folium.Map(location=coordenadas, zoom_start=17)
        folium.Marker(location=coordenadas, popup=estadio).add_to(mapa)
        mapa.save("mapa.html")
        display(mapa)
    else:
        print("No se pudieron obtener las coordenadas del estadio.")
```
Detalles de esta sección:
* Verificación del Club: Primero, se comprueba si el club ingresado por el usuario está en el diccionario clubs.
* Si el club está en la lista, se obtiene el nombre del estadio y las coordenadas llamando a la función obtener_coordenadas.
* Si las coordenadas se encuentran correctamente:
  * Crear un mapa: Se utiliza Folium para crear un mapa centrado en las coordenadas del estadio, con un nivel de zoom inicial de 17 (es decir, un zoom alto par
  * Añadir un marcador: Se agrega un marcador en el mapa en las coordenadas del estadio. Este marcador muestra el nombre del estadio al hacer clic sobre él.
  * Guardar el mapa: El mapa generado se guarda como un archivo HTML, llamado mapa.html.
  * Mostrar el mapa: Se utiliza la función display de IPython para mostrar el mapa interactivo dentro del entorno de ejecución (esto se usa principalmente en n
* Si no se pueden obtener las coordenadas del estadio, se muestra un mensaje de error.

####5. Manejo del Caso cuando el Club no Existe
```
else:
    print("Club no encontrado.")
```
Si el club ingresado por el usuario no está en el diccionario clubs, se muestra el mensaje "Club no encontrado".
"""

```python
import folium
from IPython.display import display

# Lista de clubes y sus estadios con coordenadas (latitud, longitud)
clubs = {
    "Athletic Club": {"estadio": "San Mamés, Bilbao", "coordenadas": (43.2641, -2.9495)},
    "Atlético de Madrid": {"estadio": "Wanda Metropolitano, Madrid", "coordenadas": (40.4362, -3.5995)},
    "FC Barcelona": {"estadio": "Camp Nou, Barcelona", "coordenadas": (41.3809, 2.1228)},
    "Real Betis": {"estadio": "Benito Villamarín, Sevilla", "coordenadas": (37.3565, -5.9810)},
    "Celta de Vigo": {"estadio": "Abanca Balaídos, Vigo", "coordenadas": (42.2110, -8.7393)},
    "Deportivo Alavés": {"estadio": "Mendizorroza, Vitoria-Gasteiz", "coordenadas": (42.8370, -2.6879)},
    "Elche": {"estadio": "Manuel Martínez Valero, Elche", "coordenadas": (38.2670, -0.6630)},
    "Espanyol": {"estadio": "RCDE Stadium, Cornellà de Llobregat", "coordenadas": (41.3476, 2.0755)},
    "Getafe": {"estadio": "Coliseum Alfonso Pérez, Getafe", "coordenadas": (40.3259, -3.7148)},
    "Granada": {"estadio": "Nuevo Los Cármenes, Granada", "coordenadas": (37.1386, -3.6093)},
    "Girona": {"estadio": "Montilivi, Girona", "coordenadas": (41.9549, 2.8174)},
    "Las Palmas": {"estadio": "Gran Canaria, Las Palmas", "coordenadas": (28.1248, -15.4300)},
    "Mallorca": {"estadio": "Son Moix, Palma de Mallorca", "coordenadas": (39.5895, 2.6301)},
    "Osasuna": {"estadio": "El Sadar, Pamplona", "coordenadas": (42.7960, -1.6371)},
    "Rayo Vallecano": {"estadio": "Estadio de Vallecas, Madrid", "coordenadas": (40.3918, -3.6587)},
    "Real Madrid": {"estadio": "Santiago Bernabéu, Madrid", "coordenadas": (40.4531, -3.6884)},
    "Real Sociedad": {"estadio": "Reale Arena, San Sebastián", "coordenadas": (43.3012, -1.9730)},
    "Sevilla": {"estadio": "Ramón Sánchez Pizjuán, Sevilla", "coordenadas": (37.3841, -5.9706)},
    "Valencia": {"estadio": "Mestalla, Valencia", "coordenadas": (39.4745, -0.3582)},
    "Villarreal": {"estadio": "La Cerámica, Villarreal", "coordenadas": (39.9448, -0.1035)}
}

# Función para obtener las coordenadas del estadio
def obtener_coordenadas(club):
    if club in clubs:
        return clubs[club]['coordenadas']
    else:
        return None

# Input del usuario
club = input("Introduce el nombre del club: ")

if club in clubs:
    estadio = clubs[club]['estadio']
    coordenadas = obtener_coordenadas(club)

    if coordenadas:
        # Crear un mapa centrado en las coordenadas del estadio con mayor zoom
        mapa = folium.Map(location=coordenadas, zoom_start=17)

        # Añadir un marcador para el estadio
        folium.Marker(location=coordenadas, popup=estadio).add_to(mapa)

        # Guardar y mostrar el mapa
        mapa.save("mapa.html")
        display(mapa)
    else:
        print("No se pudieron obtener las coordenadas del estadio.")
else:
    print("Club no encontrado.")

"""#Coordenadas de la localización de los estadios de los equipos de la Bundesliga"""

import folium
from IPython.display import display

# Lista de clubes de la Bundesliga y sus estadios con coordenadas (latitud, longitud)
```
```

```
clubs = {
    "1. FC Heidenheim": {"estadio": "Voith-Arena", "coordenadas": (48.6643, 10.1393)},
    "1. FC Union Berlin": {"estadio": "Stadion An der Alten Forsterei", "coordenadas": (52.5404, 13.4763)},
    "1. FSV Mainz 05": {"estadio": "MEWA Arena", "coordenadas": (49.9844, 8.2244)},
    "Bayer 04 Leverkusen": {"estadio": "BayArena", "coordenadas": (51.0376, 7.0147)},
    "Borussia Dortmund": {"estadio": "Signal Iduna Park", "coordenadas": (51.4925, 7.4514)},
    "Borussia Mönchengladbach": {"estadio": "Borussia-Park", "coordenadas": (51.1745, 6.3853)},
    "Eintracht Frankfurt": {"estadio": "Deutsche Bank Park", "coordenadas": (50.0682, 8.6454)},
    "FC Augsburg": {"estadio": "WWK Arena", "coordenadas": (48.3225, 10.8853)},
    "FC Bayern Munich": {"estadio": "Allianz Arena", "coordenadas": (48.2188, 11.6247)},
    "FC St. Pauli": {"estadio": "Millerntor-Stadion", "coordenadas": (53.5544, 9.9672)},
    "Holstein Kiel": {"estadio": "Holstein-Stadion", "coordenadas": (54.3208, 10.1400)},
    "RB Leipzig": {"estadio": "Red Bull Arena", "coordenadas": (51.3458, 12.3748)},
    "SC Freiburg": {"estadio": "Europa-Park Stadion", "coordenadas": (47.9887, 7.8926)},
    "SV Werder Bremen": {"estadio": "Weserstadion", "coordenadas": (53.0667, 8.8378)},
    "TSG Hoffenheim": {"estadio": "PreZero Arena", "coordenadas": (49.2787, 8.8571)},
    "VfB Stuttgart": {"estadio": "MHP Arena", "coordenadas": (48.7833, 9.1783)},
    "VfL Bochum": {"estadio": "Vonovia Ruhrstadion", "coordenadas": (51.4893, 7.2216)},
    "VfL Wolfsburg": {"estadio": "Volkswagen Arena", "coordenadas": (52.4345, 10.8068)}
}

# Función para obtener las coordenadas del estadio
def obtener_coordenadas(club):
    if club in clubs:
        return clubs[club]['coordenadas']
    else:
        return None

# Input del usuario
club = input("Introduce el nombre del club: ")

if club in clubs:
    estadio = clubs[club]['estadio']
    coordenadas = obtener_coordenadas(club)

    if coordenadas:
        # Crear un mapa centrado en las coordenadas del estadio
        mapa = folium.Map(location=coordenadas, zoom_start=15)

        # Añadir un marcador para el estadio
        folium.Marker(location=coordenadas, popup=estadio).add_to(mapa)

        # Mostrar el mapa
        display(mapa)
    else:
        print("No se pudieron obtener las coordenadas del estadio.")
else:
    print("Club no encontrado.")

"""

---
# CODIGO PARA LA PAGINA DE STREAMLIT (APP.PY)
"""

import streamlit as st
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
from datetime import datetime, timedelta
import requests
from streamlit_lottie import st_lottie

# Configuración inicial de la página
st.set_page_config(
    page_title="Análisis Futbolístico",
    page_icon="⚽",
    layout="wide"
)

# Función para cargar animaciones Lottie
def load_lottieurl(url):
    r = requests.get(url)
    if r.status_code != 200:
        return None
    return r.json()

# Cargar datos
@st.cache_data
def load_data():
    spain_data = pd.read_csv('https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/CSV%20DESPUES%20DEL%20PROCESAMIENTO%20DE%20DATOS/valores_m
    bundesliga_data = pd.read_csv('https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/CSV%20DESPUES%20DEL%20PROCESAMIENTO%20DE%20DATOS/valo
    return spain_data, bundesliga_data

# Función para convertir valores de mercado
def convertir_valor(valor):
    if isinstance(valor, str):
        if "mil €" in valor:
            return int(float(valor.replace(" mil €", "").replace(",", ".")) * 1_000)
        elif "mill. €" in valor:
            return int(float(valor.replace(" mill. €", "").replace(",", ".")) * 1_000_000)
    return None

# Función para convertir URLs a imágenes
def convertir_urls_a_imagenes(df):
    df_copy = df.copy()
    for col in df_copy.columns:
        if df_copy[col].astype(str).str.startswith('http').any():
            df_copy[col] = df_copy[col].apply(lambda url: f'<img src="{url}" width="50">' if isinstance(url, str) and url.startswith('http') else url)
    return df_copy

# Función para generar valores mensuales interpolados
def generar_valores_mensuales(valor_inicial, valor_final):
    fecha_inicio = datetime(2024, 1, 1)
    fecha_actual = datetime.now()
    meses = []
```

```
    valores = []

    fecha_actual = fecha_actual.replace(day=1)
    fecha = fecha_inicio
    while fecha <= fecha_actual:
        meses.append(fecha.strftime('%B %Y'))
        fecha += timedelta(days=32)
        fecha = fecha.replace(day=1)

    num_meses = len(meses)
    for i in range(num_meses):
        valor = valor_inicial + (valor_final - valor_inicial) * (i / (num_meses - 1))
        valores.append(valor)

    return meses, valores

# Cargar datos
spain_data, bundesliga_data = load_data()

# Procesar datos de España
spain_data["Valor de Mercado en 01/01/2024"] = spain_data["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
spain_data["Valor de Mercado Actual"] = spain_data["Valor de Mercado Actual"].apply(convertir_valor)

# Procesar datos de Bundesliga
bundesliga_data["Valor de Mercado en 01/01/2024"] = bundesliga_data["Valor de Mercado en 01/01/2024"].apply(convertir_valor)
bundesliga_data["Valor de Mercado Actual"] = bundesliga_data["Valor de Mercado Actual"].apply(convertir_valor)

# Sidebar con menú principal
st.sidebar.title("Menú Principal")
menu_principal = st.sidebar.radio(
    "Seleccione una sección:",
    ["Introducción", "Objetivos", "Metodología", "Herramientas", "Resultados", "Conclusiones"]
)

# Selector de liga
liga_seleccionada = st.sidebar.selectbox(
    "Seleccione la liga:",
    ["LaLiga", "Bundesliga", "Comparativa"]
)

# Función para convertir URLs a imágenes
def convertir_urls_a_imagenes(df):
    df_copy = df.copy()
    for col in df_copy.columns:
        if df_copy[col].astype(str).str.startswith('http').any():
            df_copy[col] = df_copy[col].apply(lambda url: f'<img src="{url}" width="50">' if isinstance(url, str) and url.startswith('http') else url)
    return df_copy

# Función para convertir URLs a imágenes
def convertir_urls_a_imagenes(df):
    df_copy = df.copy()
    for col in df_copy.columns:
        if df_copy[col].astype(str).str.startswith('http').any():
            df_copy[col] = df_copy[col].apply(lambda url: f'<img src="{url}" width="50">' if isinstance(url, str) and url.startswith('http') else url)
    return df_copy

# Código principal
if menu_principal == "Introducción":
    st.title("Introducción")
    st.write("""
    La industria del fútbol ha evolucionado significativamente, convirtiéndose en un mercado
    donde el valor de los jugadores es un indicador crucial de su desempeño y potencial.
    """)

    # Cargar animación Lottie
    lottie_url = "https://lottie.host/embed/3d48d4b9-51ad-4b7d-9d28-5e248cace11/Rz3QtSCq3.json"
    lottie_coding = load_lottieurl(lottie_url)
    if lottie_coding:
        st_lottie(lottie_coding, height=200, width=300)

    # Mostrar datos según la liga seleccionada
    if liga_seleccionada == "LaLiga":
        data_to_show = spain_data
        title = "Datos de Jugadores de LaLiga"
    elif liga_seleccionada == "Bundesliga":
        data_to_show = bundesliga_data
        title = "Datos de Jugadores de Bundesliga"
    else:
        st.subheader("Comparativa entre LaLiga y Bundesliga")

        # Mostrar tablas en Comparativa
        def generar_tabla_html(data):
            data_con_imagenes = convertir_urls_a_imagenes(data)
            return data_con_imagenes.to_html(escape=False, index=False)

        # Tabla de LaLiga
        st.write("### LaLiga")
        tabla_laliga_html = generar_tabla_html(spain_data)
        st.markdown(tabla_laliga_html, unsafe_allow_html=True)

        # Tabla de Bundesliga
        st.write("### Bundesliga")
        tabla_bundesliga_html = generar_tabla_html(bundesliga_data)
        st.markdown(tabla_bundesliga_html, unsafe_allow_html=True)

    # Mostrar tabla individual con imágenes (si no es Comparativa)
    if liga_seleccionada != "Comparativa":
        with st.container():
            st.subheader(title)
            data_con_imagenes = convertir_urls_a_imagenes(data_to_show)
            st.markdown(data_con_imagenes.to_html(escape=False), unsafe_allow_html=True)

if menu_principal == "Metodología":
    st.title("Metodología")

    if liga_seleccionada == "Comparativa":
```

```
st.subheader("Análisis Comparativo: LaLiga vs Bundesliga")

# 1. Gráfica de violín
st.subheader("1. Distribución General de Valores de Mercado")
fig_violin = go.Figure()

fig_violin.add_trace(go.Violin(
    y=spain_data['Valor de Mercado Actual'],
    name='LaLiga',
    box_visible=True,
    meanline_visible=True,
    line_color='blue',
    fillcolor='rgba(0, 0, 255, 0.3)',
    opacity=0.7
))

fig_violin.add_trace(go.Violin(
    y=bundesliga_data['Valor de Mercado Actual'],
    name='Bundesliga',
    box_visible=True,
    meanline_visible=True,
    line_color='green',
    fillcolor='rgba(0, 255, 0, 0.3)',
    opacity=0.7
))

fig_violin.update_layout(
    title="Distribución de Valores de Mercado por Liga",
    yaxis_title="Valor de Mercado (€)",
    xaxis_title="Ligas",
    violingap=0.5,
    violingroupgap=0.3,
    showlegend=True
)

st.plotly_chart(fig_violin)

# 2. Gráfica de dispersión
st.subheader("2. Relación Edad vs Valor de Mercado")
fig_scatter = go.Figure()

fig_scatter.add_trace(go.Scatter(
    x=spain_data['Edad'],
    y=spain_data['Valor de Mercado Actual'],
    mode='markers',
    name='LaLiga',
    marker=dict(
        size=10,
        color='blue',
        opacity=0.6
    )
))

fig_scatter.add_trace(go.Scatter(
    x=bundesliga_data['Edad'],
    y=bundesliga_data['Valor de Mercado Actual'],
    mode='markers',
    name='Bundesliga',
    marker=dict(
        size=10,
        color='green',
        opacity=0.6
    )
))

fig_scatter.update_layout(
    title="Relación entre Edad y Valor de Mercado",
    xaxis_title="Edad",
    yaxis_title="Valor de Mercado (€)",
    showlegend=True
)

st.plotly_chart(fig_scatter)

# 3. Comparativa Individual
st.subheader("3. Comparativa Individual de Jugadores")
col1, col2 = st.columns(2)
with col1:
    jugador_laliga = st.selectbox("Selecciona un jugador de LaLiga:", spain_data['Nombre'].unique())
with col2:
    jugador_bundesliga = st.selectbox("Selecciona un jugador de Bundesliga:", bundesliga_data['Nombre'].unique())

datos_laliga = spain_data[spain_data['Nombre'] == jugador_laliga].iloc[0]
datos_bundesliga = bundesliga_data[bundesliga_data['Nombre'] == jugador_bundesliga].iloc[0]

# 3.1 Gráfica de evolución temporal
meses_laliga, valores_laliga = generar_valores_mensuales(
    datos_laliga['Valor de Mercado en 01/01/2024'],
    datos_laliga['Valor de Mercado Actual']
)
meses_bundesliga, valores_bundesliga = generar_valores_mensuales(
    datos_bundesliga['Valor de Mercado en 01/01/2024'],
    datos_bundesliga['Valor de Mercado Actual']
)

fig_evolucion = go.Figure()

fig_evolucion.add_trace(go.Scatter(
    x=meses_laliga,
    y=valores_laliga,
    mode='lines+markers',
    name=f"{jugador_laliga} (LaLiga)",
    line=dict(color='red', width=3),
    marker=dict(size=10)
))
```

```
fig_evolucion.add_trace(go.Scatter(
    x=meses_bundesliga,
    y=valores_bundesliga,
    mode='lines+markers',
    name=f"{jugador_bundesliga} (Bundesliga)",
    line=dict(color='blue', width=3),
    marker=dict(size=10)
))

fig_evolucion.update_layout(
    title='Evolución Mensual del Valor de Mercado',
    xaxis_title='Mes',
    yaxis_title='Valor de Mercado (€)',
    hovermode='x unified',
    showlegend=True
)

st.plotly_chart(fig_evolucion)

# 3.2 Gráfica de barras comparativa
fig_barras = go.Figure(data=[
    go.Bar(name='Valor Inicial',
          x=['LaLiga', 'Bundesliga'],
          y=[datos_laliga['Valor de Mercado en 01/01/2024'],
             datos_bundesliga['Valor de Mercado en 01/01/2024']],
          marker_color=['rgba(255, 0, 0, 0.7)', 'rgba(0, 0, 255, 0.7)']),
    go.Bar(name='Valor Actual',
          x=['LaLiga', 'Bundesliga'],
          y=[datos_laliga['Valor de Mercado Actual'],
             datos_bundesliga['Valor de Mercado Actual']],
          marker_color=['rgba(255, 0, 0, 0.9)', 'rgba(0, 0, 255, 0.9)'])
])

fig_barras.update_layout(
    title=f'Comparación de Valores: {jugador_laliga} vs {jugador_bundesliga}',
    barmode='group',
    yaxis_title='Valor de Mercado (€)'
)

st.plotly_chart(fig_barras)

# 4. Análisis de Variación Porcentual
variacion_laliga = ((datos_laliga['Valor de Mercado Actual'] -
                     datos_laliga['Valor de Mercado en 01/01/2024']) /
                     datos_laliga['Valor de Mercado en 01/01/2024'] * 100)

variacion_bundesliga = ((datos_bundesliga['Valor de Mercado Actual'] -
                         datos_bundesliga['Valor de Mercado en 01/01/2024']) /
                         datos_bundesliga['Valor de Mercado en 01/01/2024'] * 100)

# Gráfica de variación porcentual
fig_variacion = go.Figure(data=[
    go.Bar(
        x=['LaLiga', 'Bundesliga'],
        y=[variacion_laliga, variacion_bundesliga],
        marker_color=['red', 'blue'],
        text=[f"{variacion_laliga:.1f}%", f"{variacion_bundesliga:.1f}%"],
        textposition='auto',
    )
])

fig_variacion.update_layout(
    title='Variación Porcentual del Valor de Mercado',
    yaxis_title='Variación (%)',
    showlegend=False
)

st.plotly_chart(fig_variacion)

# Análisis detallado
st.write(f"""
### Análisis Comparativo Detallado

#### 1. Distribución General (Gráfica de Violín)
- Muestra la concentración de valores en diferentes rangos
- Permite identificar patrones de valoración en cada liga
- Revela la dispersión y simetría de los valores

#### 2. Relación Edad-Valor (Gráfica de Dispersión)
- Visualiza la correlación entre edad y valor de mercado
- Identifica tendencias de valoración por edad
- Permite comparar políticas de valoración entre ligas

#### 3. Análisis Individual

**{jugador_laliga} (LaLiga)**
- Valor inicial (Enero 2024): €{datos_laliga['Valor de Mercado en 01/01/2024']:,}
- Valor actual: €{datos_laliga['Valor de Mercado Actual']:,}
- Variación porcentual: {variacion_laliga:.2f}%
- Tendencia: {'Positiva ↑' if variacion_laliga > 0 else 'Negativa ↓' if variacion_laliga < 0 else 'Estable →'}

**{jugador_bundesliga} (Bundesliga)**
- Valor inicial (Enero 2024): €{datos_bundesliga['Valor de Mercado en 01/01/2024']:,}
- Valor actual: €{datos_bundesliga['Valor de Mercado Actual']:,}
- Variación porcentual: {variacion_bundesliga:.2f}%
- Tendencia: {'Positiva ↑' if variacion_bundesliga > 0 else 'Negativa ↓' if variacion_bundesliga < 0 else 'Estable →'}

#### 4. Factores Influyentes

**Rendimiento Deportivo**
- Participación en competiciones
- Estadísticas individuales
- Impacto en resultados del equipo

**Factores Externos**
- Lesiones o tiempo de inactividad
```

```
        - Situación contractual
        - Edad y potencial de desarrollo
        - Demanda en el mercado

        #### 5. Conclusiones del Análisis
        - {'El jugador de LaLiga muestra una tendencia más pronunciada' if abs(variacion_laliga) > abs(variacion_bundesliga) else 'El jugador de Bundesliga mue
        - Diferencias entre ligas: {'Los valores sugieren una valoración más alta en LaLiga' if datos_laliga['Valor de Mercado Actual'] > datos_bundesliga['Val
        - Oportunidades de mercado: {'Potencial de inversión en crecimiento' if variacion_laliga > 0 or variacion_bundesliga > 0 else 'Momento de cautela en in
        """)

    else:
        # Para las otras visualizaciones (Evolución Individual, Comparación entre Jugadores, etc.)
        visualizacion = st.selectbox(
            "Seleccione tipo de visualización:",
            ["Evolución Individual", "Comparación entre Jugadores"]
        )

        # Selección de datos según la liga
        if liga_seleccionada == "LaLiga":
            data = spain_data
        elif liga_seleccionada == "Bundesliga":
            data = bundesliga_data

        # Visualización: Evolución Individual
        if visualizacion == "Evolución Individual":
            st.subheader(f"Evolución Individual del Valor de Mercado - {liga_seleccionada}")
            nombre_jugador = st.selectbox("Selecciona un jugador:", data['Nombre'].unique())

            jugador = data[data['Nombre'] == nombre_jugador]
            if not jugador.empty:
                valor_inicial = jugador['Valor de Mercado en 01/01/2024'].iloc[0]
                valor_final = jugador['Valor de Mercado Actual'].iloc[0]

                meses, valores = generar_valores_mensuales(valor_inicial, valor_final)

                fig = go.Figure()
                fig.add_trace(go.Scatter(
                    x=meses,
                    y=valores,
                    mode='lines+markers',
                    name=nombre_jugador,
                    line=dict(width=3),
                    marker=dict(size=10)
                ))

                fig.update_layout(
                    title=f'Evolución Mensual del Valor de Mercado de {nombre_jugador}',
                    xaxis_title='Mes',
                    yaxis_title='Valor de Mercado (€)',
                    hovermode='x unified',
                    showlegend=True
                )
                st.plotly_chart(fig)

                df_mensual = pd.DataFrame({
                    'Mes': meses,
                    'Valor de Mercado (€)': [f"€{int(v):,}" for v in valores]
                })
                st.write("Valores mensuales:")
                st.dataframe(df_mensual)

        # Visualización: Comparación entre Jugadores
        elif visualizacion == "Comparación entre Jugadores":
            if liga_seleccionada == "Comparativa":
                st.subheader("Comparación entre Jugadores de LaLiga y Bundesliga")
                col1, col2 = st.columns(2)
                with col1:
                    jugador1 = st.selectbox("Jugador de LaLiga:", spain_data['Nombre'].unique())
                with col2:
                    jugador2 = st.selectbox("Jugador de Bundesliga:", bundesliga_data['Nombre'].unique())

                if jugador1 and jugador2:
                    fig = go.Figure()

                    # Datos LaLiga
                    datos_jugador1 = spain_data[spain_data['Nombre'] == jugador1]
                    valor_inicial1 = datos_jugador1['Valor de Mercado en 01/01/2024'].iloc[0]
                    valor_final1 = datos_jugador1['Valor de Mercado Actual'].iloc[0]
                    meses1, valores1 = generar_valores_mensuales(valor_inicial1, valor_final1)

                    # Datos Bundesliga
                    datos_jugador2 = bundesliga_data[bundesliga_data['Nombre'] == jugador2]
                    valor_inicial2 = datos_jugador2['Valor de Mercado en 01/01/2024'].iloc[0]
                    valor_final2 = datos_jugador2['Valor de Mercado Actual'].iloc[0]
                    meses2, valores2 = generar_valores_mensuales(valor_inicial2, valor_final2)

                    fig.add_trace(go.Scatter(
                        x=meses1,
                        y=valores1,
                        mode='lines+markers',
                        name=f"{jugador1} (LaLiga)",
                        line=dict(width=3),
                        marker=dict(size=10)
                    ))

                    fig.add_trace(go.Scatter(
                        x=meses2,
                        y=valores2,
                        mode='lines+markers',
                        name=f"{jugador2} (Bundesliga)",
                        line=dict(width=3),
                        marker=dict(size=10)
                    ))

                    fig.update_layout(
                        title='Comparación de Valores de Mercado entre Ligas',
```

```
                        xaxis_title='Mes',
                        yaxis_title='Valor de Mercado (€)',
                        hovermode='x unified',
                        showlegend=True
                    )
                    st.plotly_chart(fig)

                    # Análisis
                    st.write(f"""
                    ### Análisis de la Comparación:
                    - **{jugador1} (LaLiga)**:
                        - Valor Inicial: €{int(valor_inicial1):,}
                        - Valor Actual: €{int(valor_final1):,}
                    - **{jugador2} (Bundesliga)**:
                        - Valor Inicial: €{int(valor_inicial2):,}
                        - Valor Actual: €{int(valor_final2):,}

                    Este análisis resalta las diferencias en las trayectorias de los jugadores seleccionados,
                    permitiendo observar cómo han evolucionado sus valores de mercado a lo largo del tiempo.
                    """)
            else:
                st.subheader(f"Comparación entre Jugadores - {liga_seleccionada}")
                col1, col2 = st.columns(2)
                with col1:
                    jugador1 = st.selectbox("Primer jugador:", data['Nombre'].unique())
                with col2:
                    jugador2 = st.selectbox("Segundo jugador:", data['Nombre'].unique())

                if jugador1 and jugador2:
                    fig = go.Figure()

                    for jugador in [jugador1, jugador2]:
                        datos_jugador = data[data['Nombre'] == jugador]
                        valor_inicial = datos_jugador['Valor de Mercado en 01/01/2024'].iloc[0]
                        valor_final = datos_jugador['Valor de Mercado Actual'].iloc[0]

                        meses, valores = generar_valores_mensuales(valor_inicial, valor_final)

                        fig.add_trace(go.Scatter(
                            x=meses,
                            y=valores,
                            mode='lines+markers',
                            name=jugador,
                            line=dict(width=3),
                            marker=dict(size=10)
                        ))

                    fig.update_layout(
                        title=f'Comparación de Valores de Mercado - {liga_seleccionada}',
                        xaxis_title='Mes',
                        yaxis_title='Valor de Mercado (€)',
                        hovermode='x unified',
                        showlegend=True
                    )
                    st.plotly_chart(fig)

                    # Análisis
                    st.write(f"""
                    ### Análisis de la Comparación:
                    Comparando los valores de mercado de **{jugador1}** y **{jugador2}** en la misma liga,
                    es posible observar diferencias significativas en sus trayectorias.

                    Estos patrones pueden estar relacionados con:
                    - **Rendimiento reciente**.
                    - **Impacto en sus equipos**.
                    - **Expectativas futuras en el mercado de fichajes**.
                    """)



elif menu_principal == "Objetivos":
    st.title("Objetivos del Proyecto")
    st.write("""
    ### Objetivos Principales:
    - Analizar y visualizar el valor de mercado de los jugadores en LaLiga y Bundesliga
    - Evaluar el incremento porcentual del valor de mercado a lo largo del tiempo
    - Identificar patrones y tendencias en la valoración de jugadores
    - Comparar las valoraciones entre las diferentes ligas
    """)

elif menu_principal == "Herramientas":
    st.title("Herramientas y Tecnologías")
    col1, col2 = st.columns(2)

    with col1:
        st.header("Tecnologías Principales")
        st.write("""
        - Python
        - Pandas
        - Streamlit
        - Plotly
        - Google Colab
        - Jupiter Notebook
        """)

    with col2:
        st.header("Bibliotecas Adicionales")
        st.write("""
        - Matplotlib
        - Seaborn
        - Streamlit-Lottie
        """)

elif menu_principal == "Resultados":
    st.title("Resultados")
```

```
    if liga_seleccionada == "Comparativa":
        tab1, tab2, tab3 = st.tabs(["Estadísticas Generales", "Análisis Comparativo", "Recomendaciones"])

        with tab1:
            st.header("Estadísticas Generales")
            col1, col2 = st.columns(2)

            with col1:
                st.subheader("LaLiga")
                st.dataframe(spain_data[['Valor de Mercado en 01/01/2024', 'Valor de Mercado Actual']].describe())

            with col2:
                st.subheader("Bundesliga")
                st.dataframe(bundesliga_data[['Valor de Mercado en 01/01/2024', 'Valor de Mercado Actual']].describe())

        with tab2:
            st.header("Análisis Comparativo")
            fig = go.Figure()

            fig.add_trace(go.Box(
                y=spain_data['Valor de Mercado Actual'],
                name='LaLiga'
            ))

            fig.add_trace(go.Box(
                y=bundesliga_data['Valor de Mercado Actual'],
                name='Bundesliga'
            ))

            fig.update_layout(
                title='Distribución de Valores de Mercado por Liga',
                yaxis_title='Valor de Mercado (€)'
            )
            st.plotly_chart(fig)

        with tab3:
            st.header("Recomendaciones")
            st.write("""
            Basadas en el análisis comparativo:
            - Recomendaciones para clubes de ambas ligas
            - Estrategias de inversión considerando diferencias entre mercados
            - Oportunidades de mercado en ambas ligas
            """)
    else:
        tab1, tab2, tab3 = st.tabs(["Estadísticas Generales", "Análisis de Tendencias", "Recomendaciones"])

        with tab1:
            st.header("Estadísticas Generales")
            if liga_seleccionada == "LaLiga":
                st.dataframe(spain_data[['Valor de Mercado en 01/01/2024', 'Valor de Mercado Actual']].describe())
            else:
                st.dataframe(bundesliga_data[['Valor de Mercado en 01/01/2024', 'Valor de Mercado Actual']].describe())

        with tab2:
            st.header("Análisis de Tendencias")
            fig = go.Figure()

            if liga_seleccionada == "LaLiga":
                fig.add_trace(go.Box(
                    y=spain_data['Valor de Mercado en 01/01/2024'],
                    name='Enero 2024'
                ))
                fig.add_trace(go.Box(
                    y=spain_data['Valor de Mercado Actual'],
                    name='Actual'
                ))
            else:
                fig.add_trace(go.Box(
                    y=bundesliga_data['Valor de Mercado en 01/01/2024'],
                    name='Enero 2024'
                ))
                fig.add_trace(go.Box(
                    y=bundesliga_data['Valor de Mercado Actual'],
                    name='Actual'
                ))

            fig.update_layout(
                title=f'Distribución de Valores de Mercado - {liga_seleccionada}',
                yaxis_title='Valor de Mercado (€)'
            )
            st.plotly_chart(fig)

        with tab3:
            st.header("Recomendaciones")
            st.write(f"""
            Basadas en el análisis de datos de {liga_seleccionada}:
            - Recomendaciones para clubes
            - Estrategias de inversión
            - Oportunidades de mercado
            """)

else:  # Conclusiones
    st.title("Conclusiones")
    if liga_seleccionada == "Comparativa":
        st.write("""
        ### Principales Hallazgos:
        - Comparativa entre LaLiga y Bundesliga muestra patrones interesantes en la valoración de jugadores
        - Las diferencias entre mercados ofrecen oportunidades únicas de inversión
        - La gestión basada en datos puede mejorar significativamente las estrategias de los equipos en ambas ligas
        """)
    else:
        st.write(f"""
        ### Principales Hallazgos en {liga_seleccionada}:
        - El análisis de datos en el fútbol ofrece insights valiosos para la toma de decisiones
        - Las tendencias del mercado muestran patrones significativos en la valoración de jugadores
        - La gestión basada en datos puede mejorar significativamente las estrategias de los equipos
```

```
        """)

# Footer
st.sidebar.markdown("---")
st.sidebar.info("ANÁLISIS DE LAS ESTADÍSTICAS QUE TIENEN MAYOR CORRELACIÓN CON EL VALOR DE MERCADO DE LOS JUGADORES DE FUTBOL EN ESPAÑA Y ALEMANIA")

"""ARCHIVO REQUERIMENTS TXT"""

streamlit==1.31.1
pandas==2.2.0
plotly==5.18.0
requests==2.31.0
streamlit-lottie==0.0.5
plotly-express==0.4.1

!pip install streamlit

!pip install streamlit-lottie

"""#CODIGO QUE MUESTRA LA TABLA DE LOS JUGADORES DE LA LIGA
Este codigo muestra la tabla extraida desde el repositorio de github
"""

import pandas as pd
from IPython.core.display import display, HTML

# URL del archivo CSV en tu repositorio de GitHub
url = "https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/CSV%20DESPUES%20DEL%20PROCESAMIENTO%20DE%20DATOS/valores_mercado_actualizados_con

# Cargar los datos
data = pd.read_csv(url)

# Verificar si hay columnas que contienen URLs de imágenes
def convertir_urls_a_imagenes(df):
    df_copy = df.copy()
    for col in df_copy.columns:
        if df_copy[col].astype(str).str.startswith('http').any():
            df_copy[col] = df_copy[col].apply(lambda url: f'<img src="{url}" width="50">' if isinstance(url, str) and url.startswith('http') else url)
    return df_copy

# Convertir columnas con URLs en HTML para imágenes
data_con_imagenes = convertir_urls_a_imagenes(data)

# Mostrar la tabla con imágenes en Colab
display(HTML(data_con_imagenes.to_html(escape=False)))

"""#CODIGO QUE MUESTRA LA TABLA DE LOS JUGADORES DE LA BUNDESLIGA
Este codigo muestra la tabla extraida desde el repositorio de github
"""

import pandas as pd
from IPython.core.display import display, HTML

# URL del archivo CSV en tu repositorio de GitHub
url = "https://raw.githubusercontent.com/AndersonP444/PROYECTO-SIC-JAKDG/main/CSV%20DESPUES%20DEL%20PROCESAMIENTO%20DE%20DATOS/valores_mercado_bundesliga_con_e

# Cargar los datos
data = pd.read_csv(url)

# Función para convertir URLs a imágenes en cualquier columna
def convertir_urls_a_imagenes(df):
    df_copy = df.copy()
    for col in df_copy.columns:
        # Detectar columnas que tienen URLs válidas
        if df_copy[col].astype(str).str.startswith('http').any():
            df_copy[col] = df_copy[col].apply(
                lambda url: f'<img src="{url}" width="50">' if isinstance(url, str) and url.startswith('http') else url
            )
    return df_copy

# Convertir columnas con URLs en HTML para imágenes
data_con_imagenes = convertir_urls_a_imagenes(data)

# Mostrar la tabla con imágenes en Colab
print("Tabla de Valores de Mercado con Imágenes:")
display(HTML(data_con_imagenes.to_html(escape=False)))
```