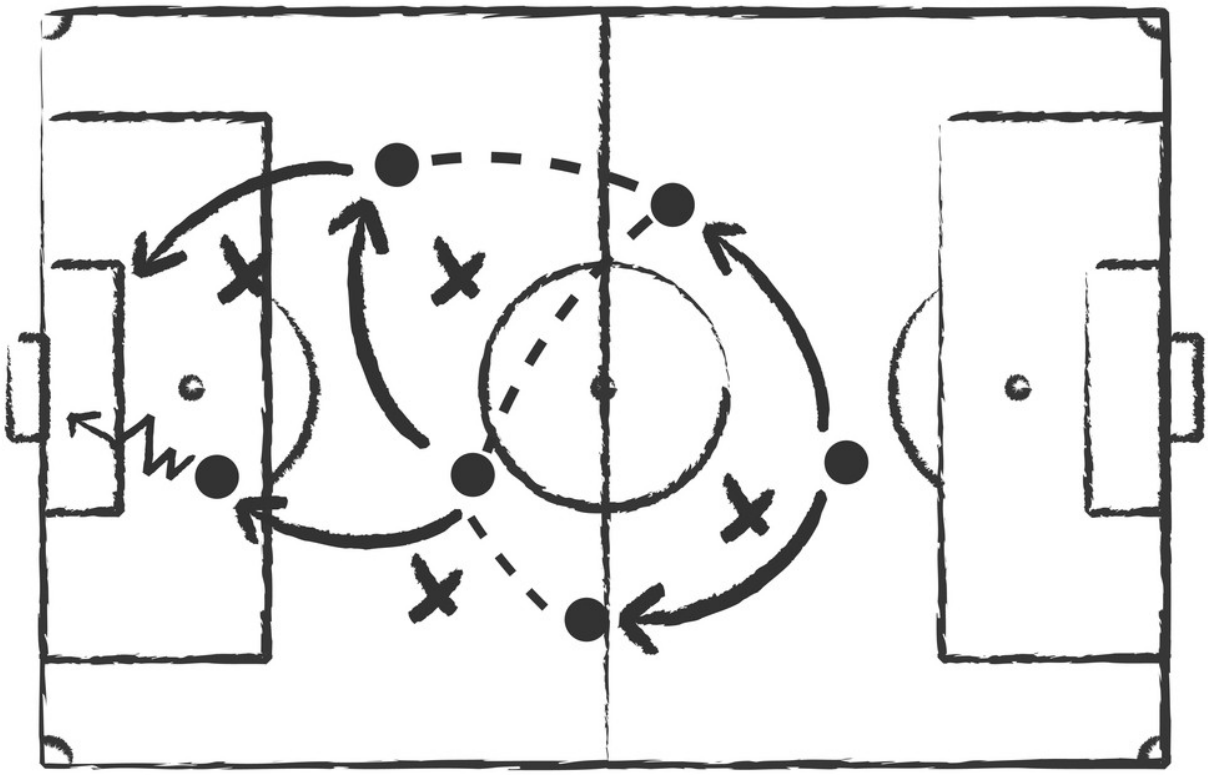
23/01/2022

Grandes Volumenes de Datos

Práctica Final

Análisis de mercado de fútbol



Víctor Hernández Sanz 21835807

Ramón Íñiguez Bascuas 21802766

Rubén Ortiz Nieto 21841860

Índice

Introducción

En este proyecto pretendemos mostrar los resultados de cruzar distintos conjuntos de datos acerca de equipos, clubes y jugadores, con la intención de crear una herramienta que sea de utilidad para los equipos para visualizar las estadísticas acerca de sus partidos, temporadas y jugadores, así como ayudarles a sacar conclusiones sobre posibles futuros fichajes ofreciendo listados configurables sobre los jugadores atendiendo a filtros como su posición, pie de preferencia, o incluso nacionalidad, ya que son factores que pueden definir el rendimiento de un jugador o de un club.

Requisitos previos

Es necesario tener pyspark instalado, además de realizar una serie de pip installs:

- pip install flask
- pip install pandas
- pip install flask-cors
- pip install streamlit
- pip install plotly
- pip install git+https://github.com/octosport/octopy.git

Cómo ejecutar la aplicación:

Abrir el proyecto, ejecutar el fichero "app.py" y esperar a que se abran todos los servicios necesarios, como se muestra a continuación:

```
 * Restarting with watchdog (windowsapi)
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
 * Debugger is active!
 * Debugger PIN: 463-710-645
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Después recomendamos que se abra una ventana en CMD y ejecutar el comando *"streamlit run main.py"* que automáticamente despliega una pestaña en el navegador con la nuestra web.

```
PS C:\Users\victo\Documents\GitHub\valor-mercado-jugador-futbol> streamlit run .\main.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8502
  Network URL: http://192.168.1.64:8502
```

Configuración necesaria (IMPORTANTE):

Una vez descargado el proyecto, es necesario meter el archivo "gcs-connector-latest-hadoop2.jar" dentro de la carpeta "%SPARK_HOME%\jars" para poder acceder al google bucket.

Enlace de descarga:
"https://storage.googleapis.com/hadoop-lib/gcs/gcs-connector-latest-hadoop2.jar "

Tecnologías empleadas:

Durante el desarrollo de esta práctica hemos empleado varias tecnologías que nos ayudan a obtener nuestros objetivos de una manera más innovadora, permitiendo ahorrar tiempo, almacenamiento y optimizando recursos.

Google Cloud Platform: Google Cloud es una plataforma que ha reunido todas las aplicaciones de desarrollo web que Google estaba ofreciendo por separado. Es utilizada para crear ciertos tipos de soluciones a través de la tecnología almacenada en la nube y destaca por la rapidez y la escalabilidad de su infraestructura.

Apache Spark: Es un framework de computación en clúster open-source. Apache Spark se puede considerar un sistema de computación en clúster de propósito general y orientado a la velocidad. Proporciona APIs en Java, Scala, Python y R.

Streamlit: Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science.

Machine Learning: es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan.

DataSets:

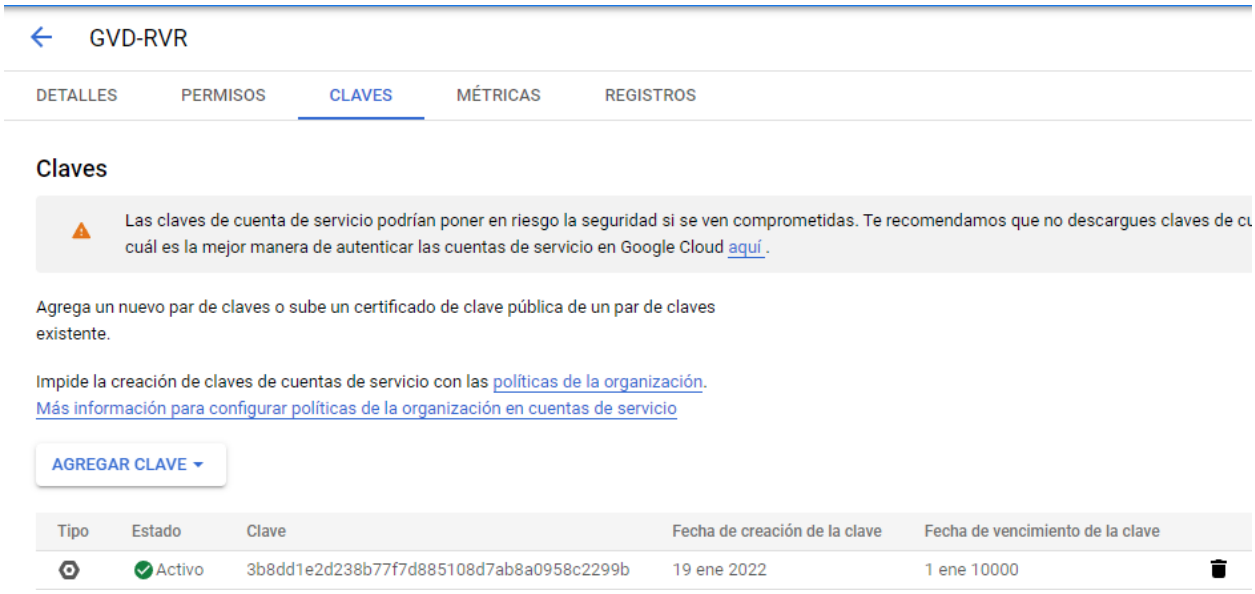
Durante la realización de ésta práctica, hemos decidido emplear dos datasets, uno que contiene información sobre partidos, equipos, jugadores y en qué posición, juegan, cuál es su pie de preferencia, etc (Football Data from Transfermarkt | Kaggle) y de manera adicional hemos usado un dataset (Football Players Market Value Prediction | Kaggle) con información relativa al valor de mercado de algunos jugadores.

Mientras trabajabamos con los datos, nos dimos cuenta que no todos los jugadores se encuentran en ambas tablas o que no todos los jugadores disponen de información en todos los casos, y debido a esto se pueden encontrar “null” o “None” en algunos de los campos sin información.

Para la lectura de datos hemos implementado dos maneras distintas, en cloud y en local:

- **Cloud**: Nuestra aplicación utiliza por defecto la lectura de datos desde el bucket “data_football” de Google Cloud Platform, y para tener acceso a este bucket, se ha configurado de manera que sea accesible públicamente a través de internet. En la siguiente imagen se puede ver todos los dataset con la configuración correspondiente:

Para poder utilizar la información del bucket, hay que crearse una cuenta de servicio dentro de “IAM y administración > Cuentas de servicio” con nombre único y generar una clave, en nuestro caso de tipo JSON:



Una vez realizados estos dos pasos, es necesario adjuntar la clave en formato JSON que hemos generado dentro de la carpeta “data” y con la inicialización del servicio y la llamada a la función “load_data_bucket()” se tendría acceso a la información de la nube.

Inicialización del servicio:

```
# Service creation (with json keyfile) to access google cloud bucket
spark._jsc.hadoopConfiguration().set('fs.gs.auth.service.account.enable', 'true')
spark._jsc.hadoopConfiguration().set('google.cloud.auth.service.account.json.keyfile', 'data/gvd-rvr-3b8dd1e2d238.json')
```

Función “load_data_bucket()”:

```
# Load data from google cloud bucket
def load_data_bucket():
    global df_appearances, df_clubs, df_competitions, df_games, df_leagues, df_players, df_final
    bucket_name='data_football'
    df_appearances = spark.read.csv(f'gs://{bucket_name}/appearances.csv', inferSchema=True, header=True, sep=',')
    df_clubs = spark.read.csv(f'gs://{bucket_name}/clubs.csv', inferSchema=True, header=True, sep=',')
    df_competitions = spark.read.csv(f'gs://{bucket_name}/competitions.csv', inferSchema=True, header=True, sep=',')
    df_games = spark.read.csv(f'gs://{bucket_name}/games.csv', inferSchema=True, header=True, sep=',')
    df_leagues = spark.read.csv(f'gs://{bucket_name}/leagues.csv', inferSchema=True, header=True, sep=',')
    df_players = spark.read.csv(f'gs://{bucket_name}/players.csv', inferSchema=True, header=True, sep=',')
    df_final = spark.read.csv(f'gs://{bucket_name}/Final.csv', inferSchema=True, header=True, sep=';')
```

- **Local**: Para la lectura de datos en local hemos utilizado la función “load_data_local()” que saca los dataset de la carpeta “data”. En principio esta función no se utiliza pero la hemos implementado en caso de que no haya conexión a internet o el bucket de google falle.

```
# Load data from 'data' folder
def load_data_local():
    global df_appearances, df_clubs, df_competitions, df_games, df_leagues, df_players, df_final
    df_appearances = spark.read.csv('data/appearances.csv', inferSchema=True, header=True, sep=',')
    df_clubs = spark.read.csv('data/clubs.csv', inferSchema=True, header=True, sep=',')
    df_competitions = spark.read.csv('data/competitions.csv', inferSchema=True, header=True, sep=',')
    df_games = spark.read.csv('data/games.csv', inferSchema=True, header=True, sep=',')
    df_leagues = spark.read.csv('data/leagues.csv', inferSchema=True, header=True, sep=',')
    df_players = spark.read.csv('data/players.csv', inferSchema=True, header=True, sep=',')
    df_final = spark.read.csv('data/Final.csv', inferSchema=True, header=True, sep=';')
```

En ambos casos, empleamos una función que se llama spark.read.csv(), que obtiene como parámetro la ruta en formato Google Storage ('gs://{bucket_name}/{name}.csv') o Local (spark.read.csv('data/{name}.csv') de los ficheros que queremos leer. Una vez hecho esto, ya disponemos en un DataFrame de Pandas el conjunto de datos relacionados.

KPIs:

Toda la información de los KPIs que vamos a mostrar, proviene de una llamada a la APIs que hemos codificado, estas funciones se definen en el fichero “api/app.py” estas APIs son:

+ *jugador_resume():*

Este KPI, permite mostrar a los equipos cual es el rendimiento de sus jugadores, recopilando el número de goles, asistencias, y la media de minutos jugados.

```
''' Show player name, total goals and assists and a media of minutes played '''
@app.route('/api/jugador_resume', methods=['GET'])
def jugador_resume():
    df_appearances.createOrReplaceTempView('sqlAppeareances')
    df_players.createOrReplaceTempView('sqlPlayers')
    result = spark.sql('''
    SELECT FIRST(sqlAppeareances.player_id) AS player_id, sqlPlayers.name, SUM(sqlAppeareances.goals), SUM(sqlAppeareances.assists),
           round(AVG(sqlAppeareances.minutes_played), 2) AS media_minutes_played, COUNT(sqlPlayers.player_id) AS games_played
    FROM sqlPlayers
    JOIN sqlAppeareances ON sqlPlayers.player_id = sqlAppeareances.player_id
    GROUP BY sqlPlayers.name
    ORDER BY player_id ''').toJSON().collect()
    return json.dumps(result)
```

+ *jugador_position_foot_price():*

Este KPI une los datasets de “players” y “Final” para sacar el nombre de los jugadores y el precio de ese jugador, filtrándose por el “Pie bueno” y la “posición” de juego que el usuario selecciona en el selectbox. Esto sirve para poder fichar a los jugadores basándose en la posición que necesite el club y la preferencia por el pie bueno de ese jugador.

Por ejemplo, un delantero izquierdo con pie derecho bueno puede tener mejor rendimiento y mejor tiro a puerta que uno con el pie bueno izquierdo.

```
''' Take players where foot and position is introduced by the user '''
@app.route('/api/jugador_position_foot_price', methods=['GET'])
def jugador_position_foot_price():
    df_players.createOrReplaceTempView('sqlPlayers')
    df_final.createOrReplaceTempView('sqlFinal')
    result = spark.sql('''
    SELECT sqlPlayers.pretty_name, sqlPlayers.position, sqlPlayers.foot, sqlFinal.Player, sqlFinal.Market_value, sqlFinal.Club_x
    FROM sqlPlayers
    JOIN sqlFinal ON sqlPlayers.pretty_name = sqlFinal.Player
    WHERE position = \"{}\" AND foot = \"{}\" '''.format(position, foot)).toJSON().collect()
    return json.dumps(result)
```

+ KPIs auxiliares para jugador_position_foot_price():

get_data_foot() selecciona todos los valores únicos de la columna ‘foot’, y el KPI ***foot_selected()*** mediante el método POST recoge el valor que el usuario introduce para pasarlo a la función ***jugador_position_foot_price()***.

```
''' Get the different values for "foot" variable '''
@app.route('/api/get_data_foot', methods=['GET'])
def get_data_foot():
    rows = df_players.select('foot').distinct().collect()
    return json.dumps([row['foot'] for row in rows])

''' Take value of "foot" from selectBox '''
@app.route('/api/foot_selected', methods=['POST'])
def foot_selected():
    global foot
    request_data = json.loads(request.data)
    foot = request_data['foot']
    return json.dumps({'message': 'success'})
```

get_data_position() selecciona todos los valores únicos de la columna 'position', y el KPI ***position_selected()*** mediante el método POST recoge el valor que el usuario introduce para pasarlo a la función ***jugador_position_foot_price()***.

```
''' Get the different values for "position" variable '''
@app.route('/api/get_data_position', methods=['GET'])
def get_data_position():
    rows = df_players.select('position').distinct().orderBy('position', ascending=False).collect()
    return json.dumps([row['position'] for row in rows])

''' Take value of "position" from selectBox '''
@app.route('/api/position_selected', methods=['POST'])
def position_selected():
    global position
    request_data = json.loads(request.data)
    position = request_data['position']
    return json.dumps({'message': 'success'})
```

+ *seleccion_inglesa():*

Mediante esta función representamos a los 11 jugadores de la selección inglesa, limitando al mejor jugador para cada posición en función a su valor de mercado.

```
''' Take players from England of each position with their market price value  '''
@app.route('/api/seleccion_inglesa', methods=['GET'])
def seleccion_inglesa():
    df_players.createOrReplaceTempView('sqlPlayers')
    df_final.createOrReplaceTempView('sqlFinal')
    result = spark.sql('''
    SELECT FIRST(sqlPlayers.pretty_name) AS pretty_name, FIRST(sqlPlayers.sub_position), FIRST(sqlFinal.Market_value), FIRST(sqlFinal.Nation)
    FROM sqlPlayers
    JOIN sqlFinal ON sqlPlayers.pretty_name = sqlFinal.Player
    WHERE Nation = "ENG"
    GROUP BY sqlPlayers.sub_position''').toJSON().collect()
    return json.dumps(result)
```

+ *most_expensive_per_position():*

Este KPI muestra los 15 jugadores más caros ordenados de manera descendente. De esta manera se puede ver el top 15 de salarios más caros en todas las ligas.

```
''' Displays the 15 most expensive players (biggest market value) along its player id, club id, country of citizenship
    team, DOB, position, sub-position and market value '''
@app.route('/api/most_expensive_players', methods=['GET'])
def most_expensive_per_position():
    df_players.createOrReplaceTempView('players')
    df_clubs.createOrReplaceTempView('clubs')
    query = spark.sql('''
        SELECT players.pretty_name AS Name, highest_market_value_in_gbp AS Market_Value
        FROM players
        JOIN clubs ON clubs.club_id = players.current_club_id
        ORDER BY highest_market_value_in_gbp DESC
        LIMIT 15
    ''').toJSON().collect()
    return json.dumps(query)
```

+ *roaster_value():*

Con este KPI se obtiene el nombre del equipo, el número de jugadores, el valor del equipo y la media del valor equipo. Con esto se puede conocer la desviación típica del precio de cada jugador.

```
''' Team name, number of players, team value and average of team value '''
@app.route('/api/roaster_value', methods=['GET'])
def roaster_value():
    df_clubs.createOrReplaceTempView('sqlClubs')
    result = spark.sql(''' SELECT sqlClubs.pretty_name, sqlClubs.squad_size, sqlClubs.total_market_value,
    sqlClubs.total_market_value / sqlClubs.squad_size AS avg_player_value
    FROM sqlClubs
    WHERE squad_size != 0
    ORDER BY squad_size ASC
    ''').toJSON().collect()
    return json.dumps(result)
```

+ *local_victories():*

Esta función recibe como parámetro el nombre del club del que se quiere obtener la información y retorna los resultados de los partidos ganados jugados en casa, junto a los identificadores y nombres de los equipos.

```
''' Results of all games won as local '''
@app.route('/api/local_victories', methods=['GET'])
def local_victories():
    df_clubs.createOrReplaceTempView('sqlClubs')
    df_games.createOrReplaceTempView('sqlGames')
    result = spark.sql(''' SELECT sqlClubs.pretty_name, sqlGames.home_club_id AS ID_local, sqlGames.away_club_id AS ID_visitante, sqlGames.home_club_goals AS Goles_local,
    sqlGames.away_club_goals AS Goles_visitante
    FROM sqlGames
    JOIN sqlClubs ON sqlGames.home_club_id = sqlClubs.club_id
    WHERE sqlClubs.pretty_name =\"{}\" AND sqlGames.away_club_goals < sqlGames.home_club_goals '''.format(pretty_name)).toJSON().collect()
    return json.dumps(result)
```

+ *KPIs auxiliares para local_victories()*

get_data_club_name() selecciona todos los valores únicos de la columna 'pretty_name', es decir, el nombre del club, y el KPI ***pretty_name_selected()*** mediante el método POST recoge el valor que el usuario introduce para pasarlo a la función ***local_victories()***.

```
''' Get the different team names for "sqlClubs.pretty_name" variable '''
@app.route('/api/get_data_club_name', methods=['GET'])
def get_data_club_name():
    rows = df_clubs.select('pretty_name').distinct().orderBy('pretty_name', ascending=True).collect()
    return json.dumps([row['pretty_name'] for row in rows])

''' Take value of "pretty_name" from selectBox '''
@app.route('/api/pretty_name_selected', methods=['POST'])
def pretty_name_selected():
    global pretty_name
    request_data = json.loads(request.data)
    pretty_name = request_data['pretty_name']
    return json.dumps({"message": "success"})
```

Predicción de resultados:

get_predictions() es una función localizada en el fichero "***predictions.py***" que emplea el resultado de los partidos de las últimas 5 temporadas de la English Premier League (dataset "prediction_data/epl.csv"), para entrenar un modelo que calcula el coeficiente de victoria para cada equipo, dependiendo de si juegan en el campo local o juegan en el campo del rival como visitante.

Los valores que retorna esta función pertenecen al conjunto de los números Reales, pudiendo ser tanto valores positivos como valores negativos, por lo que si se obtiene para un equipo en coeficiente de 1.2 como local, es más probable que gane el partido si se enfrenta contra un equipo con un coeficiente como visitante de -1.2 , ya que valores más altos en el coeficiente representan más efectividad en los partidos, y por tanto más posibilidad de victoria.

El modelo se ha entrenado con el algoritmo "LogisticModel( )" de la librería "octopy".

```
27  def get_predictions():
28      #get the data in the train date range
29      data.date = pd.DatetimeIndex(data.date)
30      #We use the results of the last 5 seasons to train our model
31      # that's why datetime.datetime(2016,3,4) stands for
32      data_train = data.loc[(data.date < datetime.datetime(2021,3,1))
33       & (data.date >= (datetime.datetime(2016,3,4)))]
34
35      #train the model
36      model = LogisticModel()
37      model.fit(data_train.home,data_train.away,data_train.home_goals,data_train.away_goals)
38
39      #get the table from the
40      teams_coef = model.get_coef()[['home wins']].sort_values('home wins',ascending=False) .round(2)
41      teams_coef.index = [x[0] for x in teams_coef.index.values.ravel()]
42
43      teams_names = teams_coef.index # recoge unicamente los valores de la primera columna ( nombres equipo )
44      #print(teams_names)
```

Para poder usar los resultados que nos aporta esta librería, hemos necesitado convertir el dataset en varias listas de tuplas, para poder ordenar los valores por equipos y mostrar de manera gráfica mediante *streamlit* los datos para los equipos como locales y visitantes como la misma variable, ya que los valores retornados son tratados para la librería como equipos diferentes y los nombra con una nomenclatura de *"away_teamname"* y "

home_teamname", de la que nos tenemos que deshacer mediante expresiones regulares para almacenar los dos resultados bajo el mismo nombre.

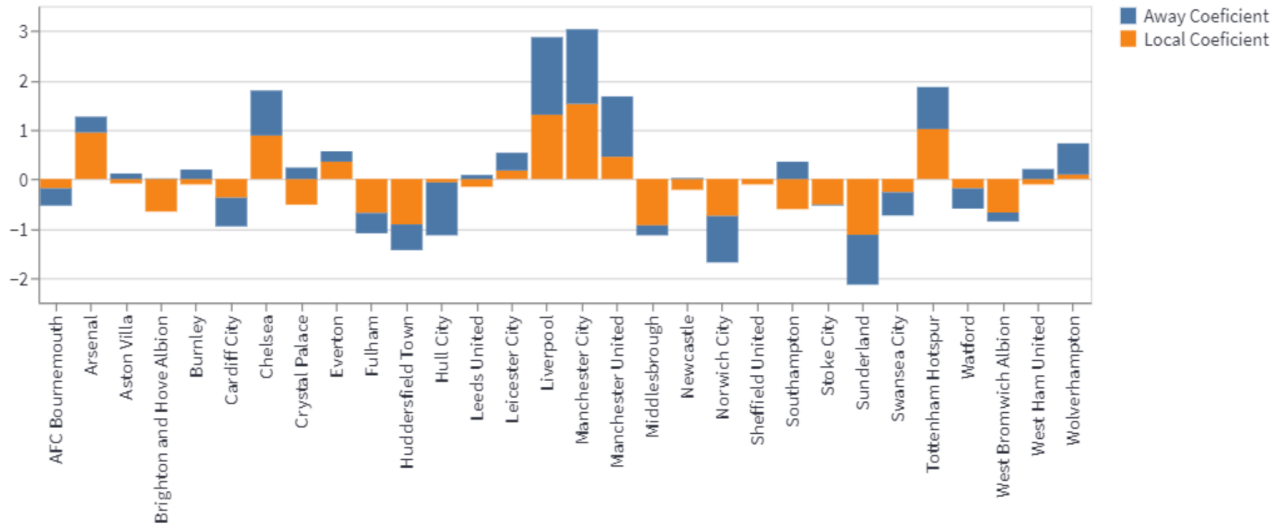
```
48      list_away= []
49      i =0
50      for t in teams_names:
51          if t[0] == "a": # si la primera letra es a de away
52              #Eliminamos el indicador que es local del nombre
53              t = re.sub('away_', '', t)
54              #Invertimos el numero para indicar que cuando estos equipos
55              # juegan fuera este es su coeficiente de victoria
56              list_away.append((t,  -1*teams_coef.iloc[i][0]))
57          i+=1
```

Es importante mencionar que hemos ordenado alfabéticamente los equipos y los resultados en las respectivas listas antes de separarlas, ya que una vez desechemos los nombres, no podríamos saber a qué equipo corresponden esos valores.

```
60      list_home= []
61      j =0
62      for t in teams_names:
63          if t[0] == "h": #Si la primera letra es h de home
64              #Eliminamos el indicador que es local del nombre
65              t = re.sub('home_', '', t)
66              list_home.append((t,teams_coef.iloc[j][0]))
67          j+=1
68      #print(list_home)
69
70      #Ordenamos las dos listas de igual manera para
71      # que los valores de las tuplas se correpondan al mismo equipo
72      list_away.sort()
73      list_home.sort()
74      teams_names_sorted = []
75      for x in teams_names:
76          # si la primera letra es h de home
77          if x[0] == "h":
78              #Eliminamos el indicador que es local del nombre
79              x = re.sub('home_', '', x)
80              teams_names_sorted.append(x)
81      teams_names_sorted.sort()
```

Como resultado de ejecutar esta función, retorna tres listas, la primera retorna la lista de valores que nos aporta la librería, el segundo elemento, se corresponde con la lista ordenada alfabéticamente de tuplas con los coeficientes de local y visitante, respectivamente. Y el último elemento retorna la lista de nombres de los clubes ordenada.

```
83      tuplas_final = []
84      for a in range(len(list_home)):
85          #guardamos en la lista de tuplas el coeficiente
86          # como locales junto con el coeficiente como visitante
87          tuplas_final.append((list_home[a][1], list_away[a][1]))
88
89
90      return teams_coef, tuplas_final, teams_names_sorted
```



Como podemos ver en la gráfica que empleamos para representar estos valores las barras naranjas representan los coeficientes para la victoria como Local, mientras que las barras azules representan los coeficientes para la victoria como visitante. Es decir, lo ideal es establecerse en rangos altos para ambos coeficientes, indicando esto que hay más posibilidades de ganar tanto en casa como fuera de ella, como sucede con el Liverpool o el Manchester City.

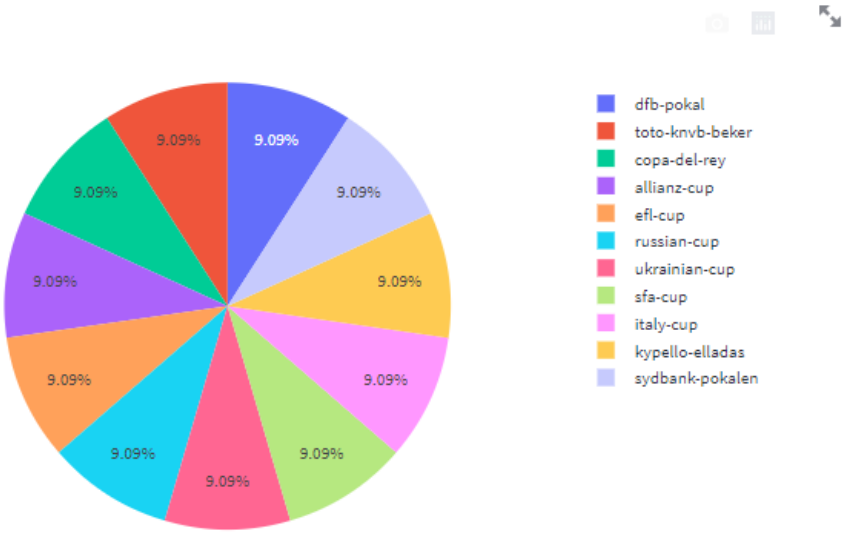
Interfaz

Al iniciar la aplicación, lo primero que se puede apreciar al terminar de cargar es la gráfica que representa, el porcentaje de información que disponemos sobre las ligas mundiales. Siendo muy equitativo, únicamente disponemos de más volumen de información para los resultados de la Premier League Inglesa.

Después de ésta gráfica se muestra una gráfica equivalente pero con las ligas domésticas de cada país, como podemos ver, disponemos de los mismos datos para todas las competiciones.

Copas domesticas

A continuacion se muestran el numero de copas domesticas disponibles.



En la gráfica que se muestra, hay dos selectbox en los que sí se despliega se puede seleccionar el valor que va a tomar la tabla de abajo.

Selector de variables

Selecciona el pie de preferencia

Left

Selecciona la posicion del jugador

Midfield

Jugadores seleccionando "Pie bueno" y "Posición"

En esta tabla el usuario puede elegir el pie bueno y la posición de juego para todos los jugadores dentro del dataset de jugadores.

	pretty_name	position	foot	Player	Market_value	Club_x
0	Talisca	Midfield	Left	Talisca	€16.00m	nan
1	Callum Wilson	Midfield	Left	Callum Wilson	€24.00m	Bournen
2	Allan	Midfield	Left	Allan	€32.00m	Napoli
3	Allan	Midfield	Left	Allan	€32.00m	Napoli
4	Paulinho	Midfield	Left	Paulinho	€18.50m	Leverkus
5	Robin Gosens	Midfield	Left	Robin Gosens	€20.00m	Atalanta
6	Mohamed Ihattaren	Midfield	Left	Mohamed Ihattaren	€20.00m	nan
7	Adrien Rabiot	Midfield	Left	Adrien Rabiot	€25.50m	Juventu
8	Sergio Canales	Midfield	Left	Sergio Canales	€20.00m	Betis
9	Nabil Fekir	Midfield	Left	Nabil Fekir	€32.00m	Betis

Desarrollo técnico:

En el “main.py” se declaran 2 funciones auxiliares “init_foot()” y “init_position()” que establecen los valores posibles para el selectbox. Con la función “data_request_foot_position()” se hacen llamadas a la API del “app.py” que responden con los valores de pie de preferencia y posición. Una vez que disponemos los datos, los convertimos en DataFrame de pandas y con la función de “jugador_position_foot_price_table()” rellenamos la tabla.

```python
#Table that shows players and price with "foot" and "position" given by the user
def jugador_position_foot_price_table(data):
    st.header('Jugadores seleccionando "Pie bueno" y "Posición"')
    st.write('En esta tabla el usuario puede elegir el pie bueno y la posición de juego para todos los '+
    'jugadores dentro del dataset de jugadores.')
    res = load_json_data(data)
    df = pd.DataFrame(res).astype(str)
    st.write(df)

#Fill Selectbox with "foot" choices
def init_foot(foot):
    global foot_selected
    foot_selected = st.selectbox('Selecciona el pie de preferencia', json.loads(foot))

#Fill Selectbox with "position" choices
def init_position(position):
    global position_selected
    position_selected = st.selectbox('Selecciona la posicion del jugador', json.loads(position))

#GET request calls for "jugador_position_foot_price_table"
def data_request_foot_position():
    st.header('Selector de variables')
    r_foot = requests.get('http://127.0.0.1:5000/api/get_data_foot').content
    r_position = requests.get('http://127.0.0.1:5000/api/get_data_position').content
    init_foot(r_foot)
    init_position(r_position)
```

En la imagen siguiente se puede ver que desde la página principal de la aplicación se muestra la tabla obtenida con la información sobre el rendimiento de los jugadores.

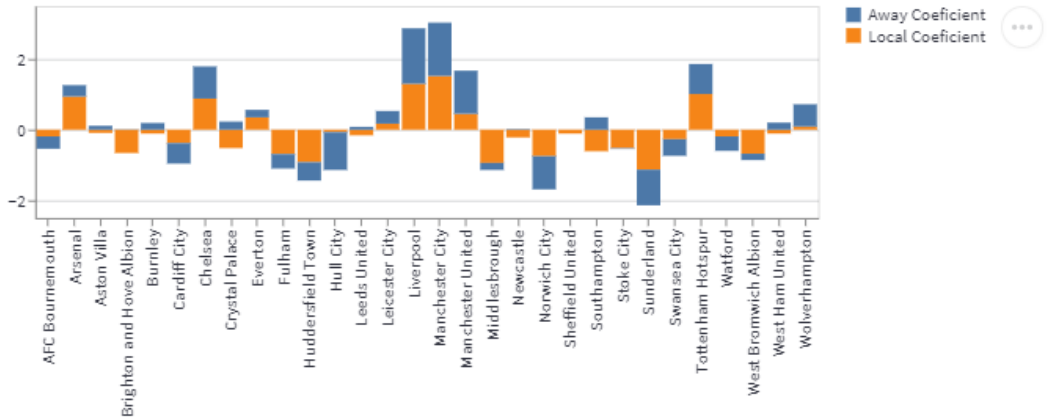
Información general jugadores

En esta tabla se muestran todos los jugadores ordenados por ID junto a la suma de goles, suma de asistencias y una media de los minutos jugados por partido.

	player_id	name	sum(goals)	sum(assists)	media_minutes_p
0	10	miroslav-klose	24	16	56.38
1	26	roman-weidenfeller	0	0	87.53
2	65	dimitar-berbatov	13	2	60.07
3	80	tom-starke	0	0	90.0
4	132	tomas-rosicky	2	1	34.48
5	215	roque-santa-cruz	8	1	49.5
6	410	sebastian-kehl	1	2	77.06
7	488	gerhard-tremmel	0	0	75.8
8	532	claudio-pizarro	29	10	36.42
9	568	martin-stranzl	0	0	79.41

Predicción para victorias

Se muestran los coeficientes predecidos para las victorias como local y como visitante para todos los equipos de la Premiere League, siendo los valores más positivos más probabilidad de ganar y los más negativos más posibilidades de perder.



A continuación se aprecia la gráfica de las predicciones de victoria para los equipos de la Premier League y la selección de los 11 jugadores ingleses de las distintas posiciones.

	Local Coeficient	Away Coeficient
AFC Bournemouth	-0.19	-0.35
Arsenal	0.94	0.32
Aston Villa	-0.09	0.11
Brighton and Hove Albion	-0.66	0.01
Burnley	-0.11	0.19
Cardiff City	-0.38	-0.58
Chelsea	0.88	0.91
Crystal Palace	-0.52	0.23
Everton	0.35	0.21
Fulham	-0.69	-0.41

11 Jugadores ingleses

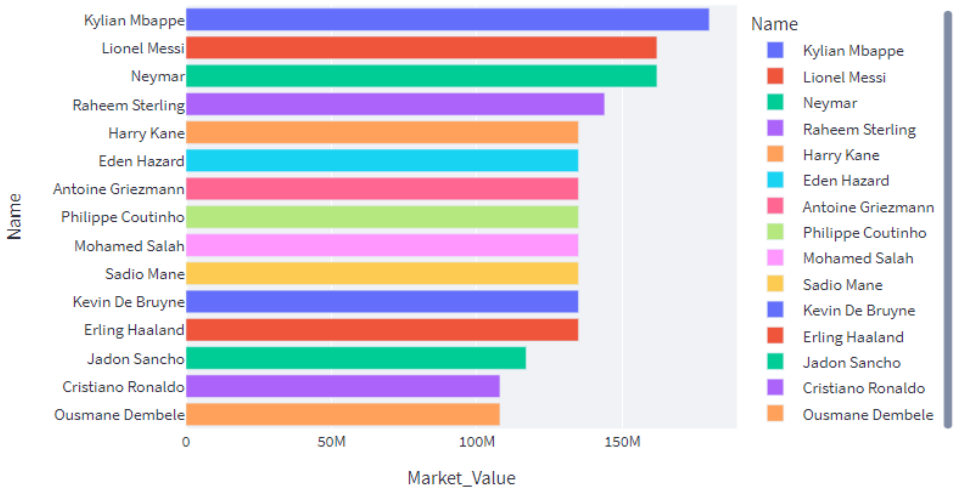
En esta tabla se muestran 11 jugadores ingleses de todas las sub posiciones junto al precio de cada uno.

	pretty_name	first(sub_position)	first(Market_value)	first(Nation)
0	Jesse Lingard	Attacking Midfield	€17.50m	ENG
1	Lewis Cook	Central Midfield	€16.00m	ENG
2	Fikayo Tomori	Centre-Back	€16.00m	ENG
3	Tammy Abraham	Centre-Forward	€40.00m	ENG
4	Declan Rice	Defensive Midfield	€49.50m	ENG
5	Bukayo Saka	Left Midfield	€20.00m	ENG
6	Jadon Sancho	Left Winger	€117.00m	ENG
7	Reece James	Left-Back	€22.50m	ENG
8	Callum Wilson	Right Midfield	€24.00m	ENG
9	Jarrod Bowen	Right Winger	€16.00m	ENG

Aquí se muestra una comparativa de los jugadores con mayor valor de mercado.

Jugadores más caros del mercado

A continuacion se muestran los 15 jugadores más caros del mercado.



Equipo con el total de jugadores, su valor y la media de su valor

En esta tabla se muestran los equipos con la cantidad de jugadores, el valor de la totalidad del equipo y también la media del valor del equipo.

	pretty_name	squad_size	total_market_value	avg_player_value
0	Desportivo Aves	1	180.0	180.0
1	Panthrakikos Komotini	5	23.0	4.6
2	Cf Uniao Madeira	5	nan	nan
3	Gfc Ajaccio	9	450.0	50.0
4	Ao Platanias	9	225.0	25.0
5	Aok Kerkyra	9	9.0	1.0
6	Sbv Excelsior Rotterdam	14	3.06	0.21857142857142858
7	Denizlispor	14	2.0	0.14285714285714285
8	Roda Jc Kerkrade	16	4.05	0.253125
9	Ado Den Haag	16	3.92	0.245

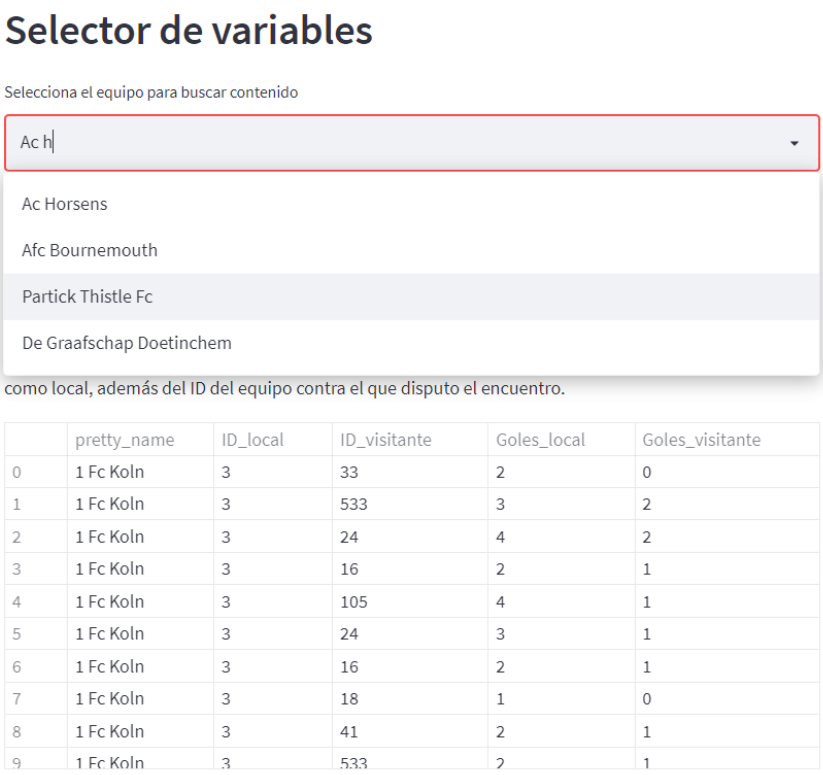
Desarrollo técnico:

La función siguiente obtiene como parámetro los datos de los jugadores con mayor valor de mercado, los transforma a un DataFrame de pandas e indicamos que use como índice para la gráfica sus nombres, mediante *df.set_index('Name')* . Para que los datos puedan ser filtrados como números y no como string aplicamos *pd.to_numeric(df["Market _Value"])* al campo donde se encuentran los precios de valor de mercado.

Una vez convertido el DataFrame tal como lo necesitamos, los mostramos mediante *streamlit* via *st.bar_char(df)* con su respectivo título y descripción.

```
def most_expensive_players(data):
    res = load_json_data(data)
    df = pd.DataFrame(res).astype(str)
    df = df.set_index('Name')
    df['Market_Value'] = pd.to_numeric(df['Market_Value'])
    st.header('Jugadores más caros del mercado')
    st.write('En esta gráfica se puede ver los 15 jugadores más caros del mercado.')
    st.bar_chart(df)
```

Abajo hay otro selectBox en el que seleccionar el equipo para actualizar la tabla de abajo y mostrar el resultado de todos los partidos ganados como local.



Conclusiones

Esta práctica nos ha demostrado que usar grandes conjuntos de datos e interaccionar con estos *DataFrames* de Spark aporta una gran escalabilidad y versatilidad a la hora de no disponer de unas máquinas con una potencia de cómputo extremadamente elevada, y con una capacidad en memoria principal limitada.

Una de las posibles mejoras de la aplicación sería realizar las peticiones al servidor paralelamente ya que bajaría muchísimo el tiempo que se tarda en pedir los datos al servidor y pintarlos en la web, ya que ahora mismo no hay problema puesto que se hacen pocas peticiones pero a medida que escale y se vuelva más complicada la aplicación esto podría ser un aspecto clave para que funcione correctamente.