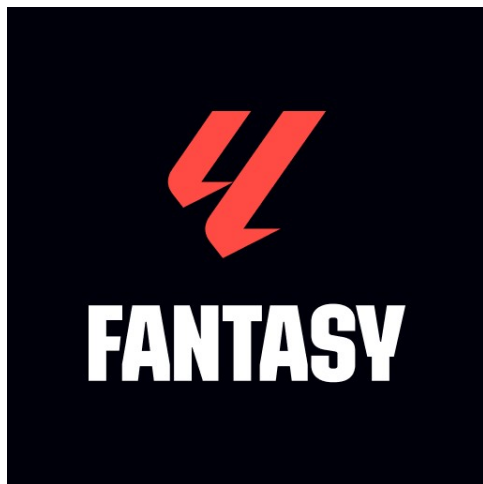


IES Clara del Rey

Procesamiento de datos Big Data

TRABAJO FINAL

LALIGA FANTASY 24/25



Darío Panadero Fernández

LALIGA FANTASY 24/25.....	1
1. INTRODUCCIÓN.....	3
OBJETIVO DEL PROYECTO.....	3
ALCANCE DEL PROYECTO.....	3
HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS.....	3
2. DESCARGA Y EXTRACCIÓN DE DATOS.....	5
PRINCIPALES INCONVENIENTES.....	5
3. NIFI: REESTRUCTURACIÓN DIRECTORIO LALIGA.....	7
4. SPARK: ANÁLISIS ESQUEMA JSON.....	11
ATRIBUTOS GENERALES DEL JUGADOR:.....	15
ESTADÍSTICAS DE TEMPORADAS Y PARTIDOS:.....	15
IMÁGENES ASOCIADAS:.....	16
INFORMACIÓN DEL EQUIPO:.....	16
5. HDFS-HIVE: ANÁLISIS DE LOS CSV.....	16
SUBIDA HDFS.....	16
CREACIÓN DE TABLAS.....	18
ANÁLISIS HIVE.....	21
6. PYTHON (JUPYTERLAB) - VISUALIZACIÓN GRÁFICA.....	24
RANKINGS DE JUGADORES.....	24
COMPARATIVA DE POSICIONES.....	26
VALORES DE MERCADO.....	29

1. INTRODUCCIÓN

Este proyecto está basado en buena parte en el siguiente proyecto del Aula Virtual

https://aulavirtual33.educa.madrid.org/ies.claradelrey.madrid/pluginfile.php/253223/mod_folder/content/0/Trabajos%20realizados/LaLigaFantasy_NCE.zip?forcedownload=1

Principalmente me he inspirado en los apartados de Spark y Python-JupyterLab. El resto del proyecto tiene variaciones ya que extraigo los datos de una API diferente, la cual además de archivos JSON también genera 2 archivos CSV

Debido a esto último se ha añadido una etapa, en la cual se suben los CSV a HDFS para analizarlos con la herramienta HIVE

OBJETIVO DEL PROYECTO

El objetivo principal de este proyecto es realizar un análisis estadístico detallado y la generación de gráficos a partir de los datos de la liga fantasy de fútbol. Esto incluye el procesamiento de grandes volúmenes de información relacionada con jugadores, equipos y sus estadísticas, con el fin de identificar patrones, tendencias y otros insights relevantes que puedan ser útiles tanto para la gestión de equipos en la plataforma como para el análisis deportivo en general.

ALCANCE DEL PROYECTO

El proyecto abarca las siguientes etapas clave:

- **Extracción de Datos:** Obtener información estructurada desde fuentes específicas, como archivos JSON y APIs relacionadas con la liga fantasy.
- **Almacenamiento de Datos:** Organizar y almacenar la información recolectada en una infraestructura adecuada para el análisis, utilizando herramientas como HDFS y Apache Hive.
- **Procesamiento de Datos:** Limpiar, transformar y normalizar los datos con herramientas de procesamiento como Apache NiFi y Apache Spark, para garantizar la consistencia y calidad de los mismos.
- **Visualización de Datos:** Generar gráficos y representaciones visuales utilizando JupyterLab (bibliotecas como Matplotlib y Seaborn) para comunicar de manera efectiva los resultados y hallazgos del análisis.

HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

El proyecto se apoya en una combinación de herramientas y tecnologías para llevar a cabo las distintas etapas del flujo de trabajo:

NIFI

Apache NiFi es una plataforma de código abierto diseñada para automatizar y gestionar el flujo de datos entre sistemas heterogéneos en tiempo real.

En este proyecto, emplearemos NiFi para organizar y procesar archivos JSON y CSV de jugadores de fútbol. Su interfaz gráfica intuitiva simplifica la creación de flujos de trabajo para manipular y procesar datos en diferentes formatos.

A través de NiFi, diseñaremos un flujo que recopilará archivos desde múltiples ubicaciones, aplicará filtros y enrutará los datos según criterios específicos, almacenándolos en directorios organizados. Esto permitirá una gestión eficiente de los datos, facilitando su análisis y visualización posterior.

SPARK

En esta sección, exploraremos las características principales de los archivos de jugadores utilizando Apache Spark. Al cargar los datos en un DataFrame, analizaremos su estructura y el esquema que proporcionan.

Este análisis inicial es esencial para:

- Comprender cómo están organizados los datos y qué estadísticas se registran para cada jugador en cada partido.
- Identificar la información clave que se incluirá en los gráficos y análisis posteriores.

Esta etapa nos ayudará a enfocar los esfuerzos en los datos más relevantes para el proyecto.

HDFS-HIVE

HDFS (Hadoop Distributed File System) y Apache Hive son componentes clave en este proyecto para el almacenamiento y análisis de datos estructurados.

Los dos archivos CSV generados se han subido a HDFS, garantizando un almacenamiento distribuido y escalable. Posteriormente, estos datos se han cargado en tablas Hive, lo que facilita su consulta y análisis mediante un lenguaje similar a SQL.

Hive permite estructurar los datos en tablas, lo que resulta ideal para analizar estadísticas clave de jugadores y su rendimiento. Este enfoque optimiza la manipulación de grandes volúmenes de datos, permitiendo un análisis eficiente y compatible con otras herramientas del ecosistema Hadoop.

PYTHON-JUPYTERLAB

El entorno de desarrollo JupyterLab ha sido clave para implementar el análisis, los gráficos y las conclusiones relacionadas con los jugadores, el mercado de valores, y otras características del juego fantasy.

Las principales librerías utilizadas incluyen:

- **Pandas:** Para la carga, limpieza, transformación y análisis estadístico de los datos.
- **JSON:** Para cargar, transformar o tratar con archivos de formato JSON en el lenguaje Python
- **Matplotlib:** Para la creación de gráficos estáticos como líneas, barras, dispersión, e histogramas.
- **Seaborn:** Basado en Matplotlib, permite generar gráficos estadísticos visualmente atractivos y fáciles de interpretar.

Con estas herramientas, hemos llevado a cabo un análisis detallado de los jugadores y del mercado fantasy, cubriendo aspectos como rankings, rendimiento por posición, estrategias de compra-venta, y la evolución del valor de mercado de los jugadores. Este enfoque analítico es esencial para maximizar el rendimiento de los equipos en el juego.

2. DESCARGA Y EXTRACCIÓN DE DATOS

PRINCIPALES INCONVENIENTES

Una de las mayores dificultades de este proyecto ha sido encontrar una fuente de datos actualizada que incluya todas las métricas y estadísticas relevantes de los jugadores en cada jornada. Además, el valor de mercado de los jugadores se actualiza diariamente alrededor de las 00:15, lo que implica un constante cambio y movimiento en los datos.

La Liga no ofrece una API ni archivos descargables que permitan realizar análisis personalizados o medir estadísticas detalladas. Únicamente proporciona los resultados visibles en la aplicación **Fantasy**, lo que limita el acceso directo a la información necesaria.

Sin embargo, en GitHub hay dos repositorios de web scraping de este juego:

- diegoparrilla/**marca-fantasy-scraper**
- alxgarci/**marca-fantasy-api-scraper-updated**

El primer repositorio data del 2020 y el segundo es una evolución del mismo cuya última versión es del 20 de septiembre de 2023.

En mi caso me decanté por el primero. Por un lado, este repositorio además de carpetas JSON para los jugadores de cada equipo también genera 2 archivos CSV.

Por otra parte, al ejecutar el archivo **fantasy-scraper.py** correspondiente a la versión actualizada nunca era capaz de extraer todos los archivos de la API. La vez que más archivos me generó fue en torno a un 25-30%, lo cual es una muestra de datos menor que la que me generaba el desarrollado por Diego Parrilla.

No obstante, este tampoco me ha podido generar todos los jugadores de la temporada actual ya que, al ser la API de 2020, solo me sirven aquellos que continúan en la liga a día de hoy. En total, en torno a unos 120 jugadores.

El enlace de su repositorio GitHub es el siguiente:

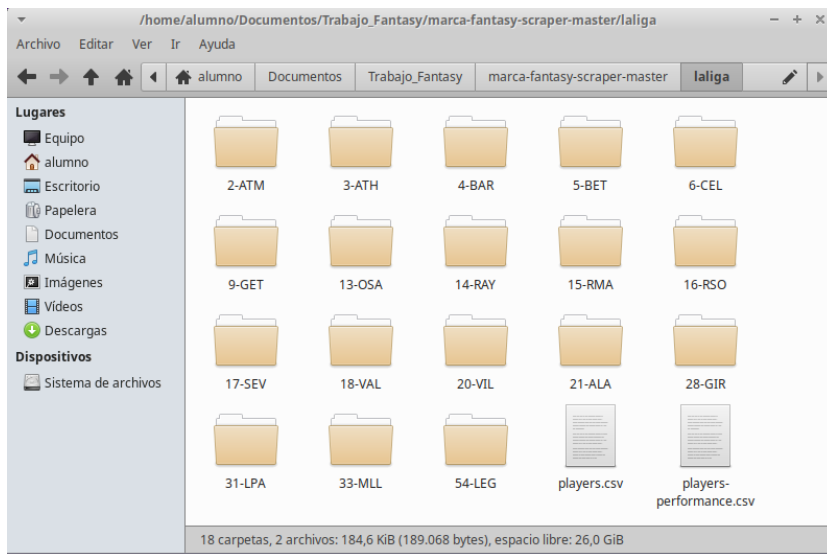
<https://github.com/diegoparrilla/marca-fantasy-scraper>

Para empezar nuestra descarga de datos, debemos pulsar donde indica Download ZIP. El nombre del archivo que descarga es marca-fantasy-scraper-master.zip.

3. NIFI: REESTRUCTURACIÓN DIRECTORIO LALIGA

El propósito de emplear *Apache NiFi* en este proyecto es organizar los archivos **JSON** de los jugadores en una estructura de directorios que facilite su manejo durante el análisis y la generación de gráficos.

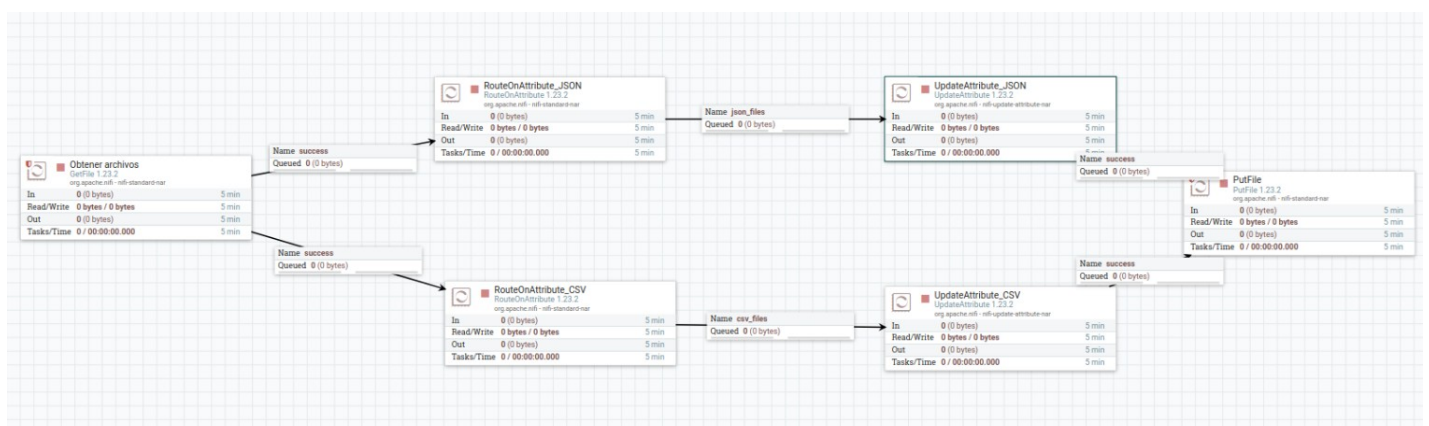
Actualmente, los archivos están ubicados en una carpeta principal llamada */laliga*, que contiene tanto archivos .csv como un subdirectorio denominado */json_players*. Dentro de este subdirectorio, existen carpetas separadas para cada equipo. Cada una de estas carpetas almacena los archivos JSON correspondientes a los jugadores de ese equipo. Esta organización jerárquica busca simplificar el acceso y procesamiento de los datos en las etapas posteriores del proyecto



El principal inconveniente de esta estructura es que complica la búsqueda y el acceso a la información al tener que navegar entre múltiples carpetas.


Reorganizar los archivos en un número menor de directorios puede simplificar el código necesario para generar estadísticas o gráficos, reduciendo la cantidad de líneas, bucles y mejorando la eficiencia del procesamiento. Aunque la separación por equipos tiene su lógica y utilidad, considero que para futuros análisis será más práctico agrupar todos los archivos JSON de jugadores en una única carpeta y los archivos CSV en otra.

Para lograr esta reorganización, utilizaremos Apache NiFi. El flujo que implementaremos es sencillo, pero permite observar las capacidades de la herramienta y cómo facilita la manipulación y organización de datos.



Los procesadores utilizados son los siguientes:

1 **GetFile**: Recorrerá la carpeta donde se encuentran todos los subdirectorios de los equipos, y recogerá las rutas de todos los archivos .json.

	Obtener archivos GetFile 1.23.2 org.apache.nifi - nifi-standard-nar
In	0 (0 bytes) 5 min
Read/Write	0 bytes / 0 bytes 5 min
Out	0 (0 bytes) 5 min
Tasks/Time	0 / 00:00:00.000 5 min

Configure Processor | GetFile 1.23.2

Stopped

SETTINGS | SCHEDULING | PROPERTIES | RELATIONSHIPS | COMMENTS

Required field

Property	Value
Input Directory	
File Filter	/** *
Path Filter	
Batch Size	1 /home/alumno/Documentos/Trabajo_Fantasy/marca-fantasy-scraper-master/laliga/
Keep Symlinks	
Recursive	
Polling	
Ignore Hidden	
Minimum File Size	0 B
Maximum File Size	No value set

EL

PARAM

Set empty string

CANCEL

OK


CANCEL



APPLY

Recurse Subdirectories	?	true
------------------------	---	------

2 **RouteOnAttribute**: Utilizaremos dos **RouteOnAttribute**, uno para filtrar los archivos que sean de formato .json, y otro para filtrar los archivos .csv.

	RouteOnAttribute_JSON RouteOnAttribute 1.23.2 org.apache.nifi - nifi-standard-nar
In	0 (0 bytes) 5 min
Read/Write	0 bytes / 0 bytes 5 min
Out	0 (0 bytes) 5 min
Tasks/Time	0 / 00:00:00.000 5 min

	RouteOnAttribute_CSV RouteOnAttribute 1.23.2 org.apache.nifi - nifi-standard-nar
In	0 (0 bytes) 5 min
Read/Write	0 bytes / 0 bytes 5 min
Out	0 (0 bytes) 5 min
Tasks/Time	0 / 00:00:00.000 5 min

json_files	?	\${filename:endsWith('.json')}	
csv_files	?	\${filename:endsWith('.csv')}	

2 **UpdateAttribute**: Cada **RouteOnAttribute** mandará los archivos filtrados a este procesador, que los actualizará.

	UpdateAttribute_JSON UpdateAttribute 1.23.2 org.apache.nifi - nifi-update-attribute-nar
In	0 (0 bytes) 5 min
Read/Write	0 bytes / 0 bytes 5 min
Out	0 (0 bytes) 5 min
Tasks/Time	0 / 00:00:00.000 5 min


target_directory



EL ✓ PARAM ✓

1 /home/alumno/Documentos/Trabajo_Fantasy/marca-fantasy-scraper-master/laliga/json_players/

☐ Set empty string

	UpdateAttribute_CSV UpdateAttribute 1.23.2 org.apache.nifi - nifi-update-attribute-nar
In	0 (0 bytes) 5 min
Read/Write	0 bytes / 0 bytes 5 min
Out	0 (0 bytes) 5 min
Tasks/Time	0 / 00:00:00.000 5 min

target_directory



EL ✓ PARAM ✓

1 /home/alumno/Documentos/Trabajo_Fantasy/marca-fantasy-scraper-master/laliga/csv/


☐ Set empty string

CANCEL

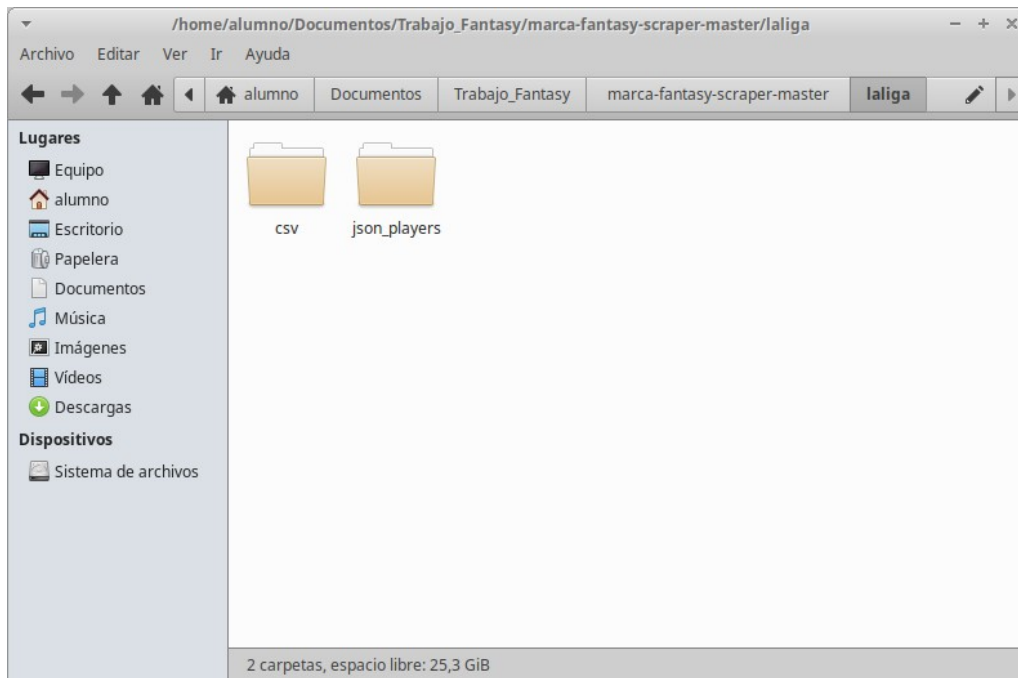
OK

PLY

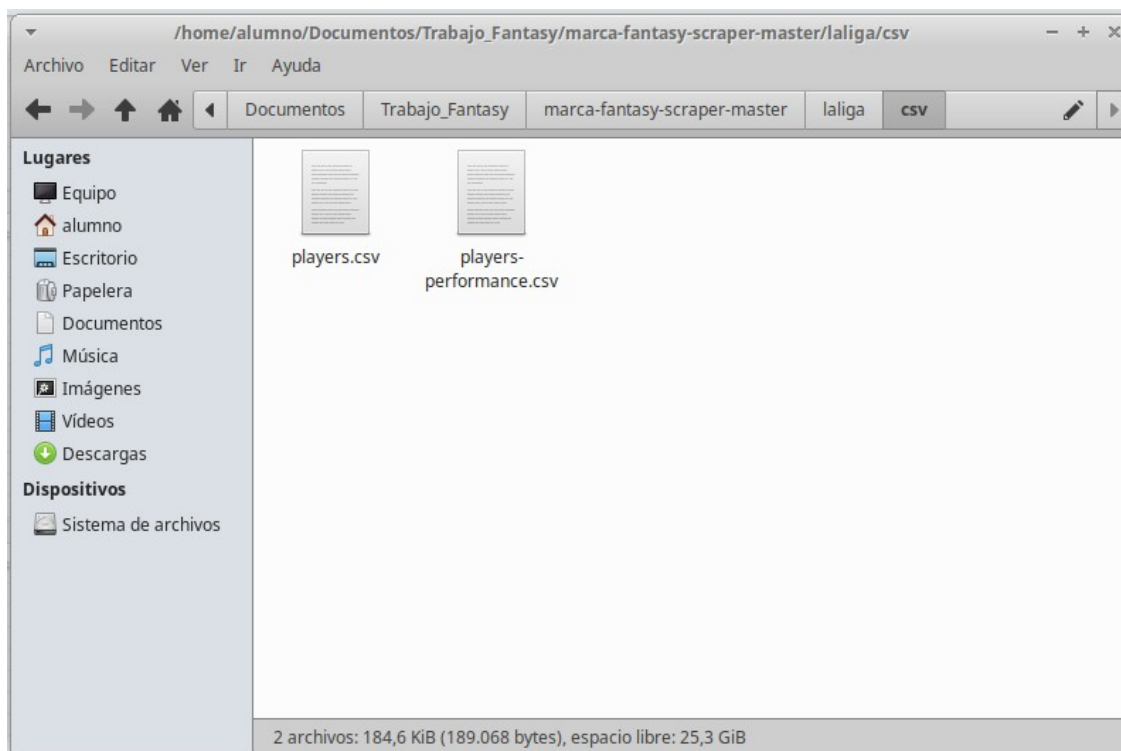
1 **PutFile**: Cada *UpdateAttribute* mandará los archivos filtrados a este procesador, que los guardará en una carpeta única para los JSON de todos los jugadores, y una carpeta única para los CSV (players.csv y players-performance.csv).

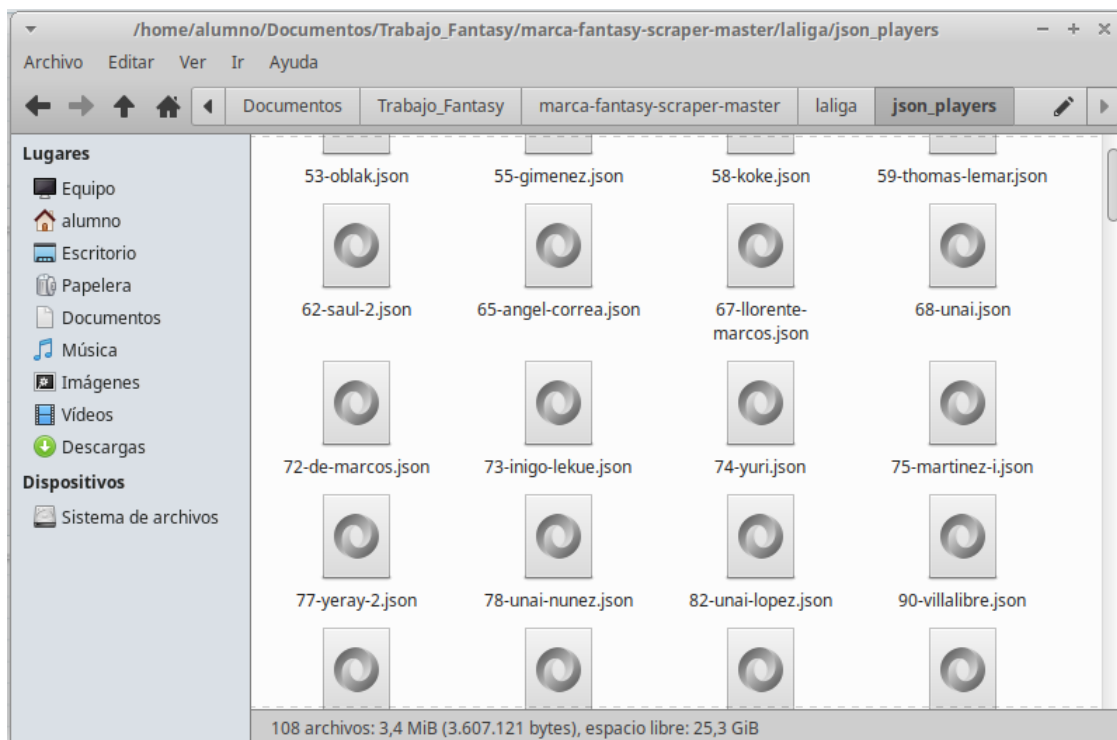
	PutFile PutFile 1.23.2 org.apache.nifi - nifi-standard-nar
In	0 (0 bytes) 5 min
Read/Write	0 bytes / 0 bytes 5 min
Out	0 (0 bytes) 5 min
Tasks/Time	0 / 00:00:00.000 5 min

Una vez ejecutamos este flujo Nifi, la estructura de la carpeta *laliga* es la siguiente



Asimismo, podemos observar que dentro de cada una de estas carpetas están los archivos correspondientes:



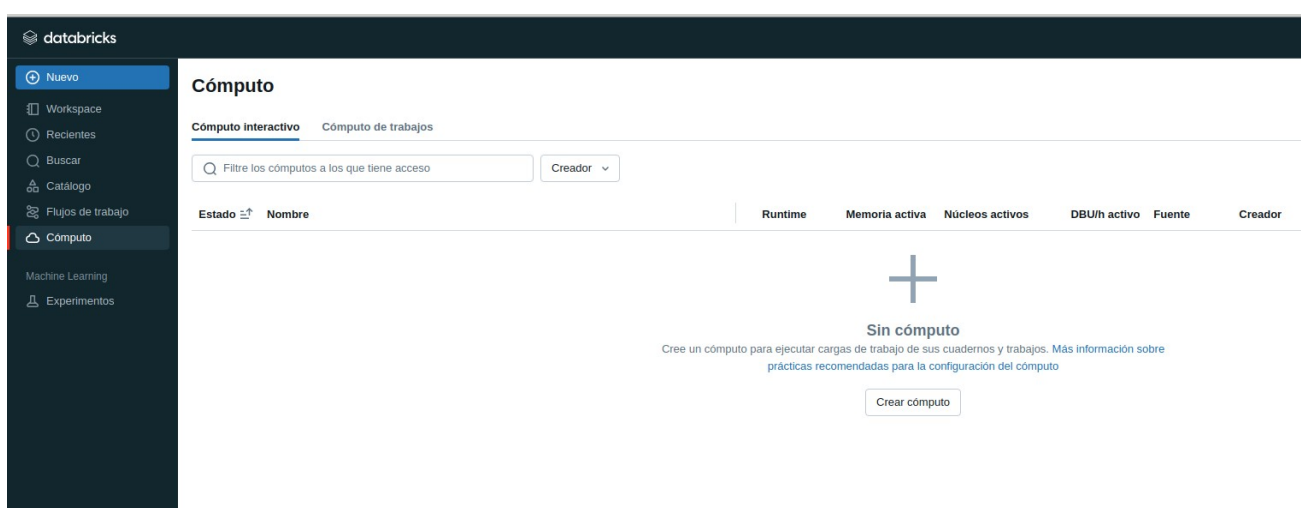


De este modo, nos resultará más sencillo analizar los archivos a posteriori.

4. SPARK: ANÁLISIS ESQUEMA JSON

En esta sección, analizaremos las características de los archivos JSON de los jugadores, enfocándonos en el esquema y las estadísticas clave que se registran, lo que nos ayudará a decidir qué gráficos y datos mostrar más adelante.

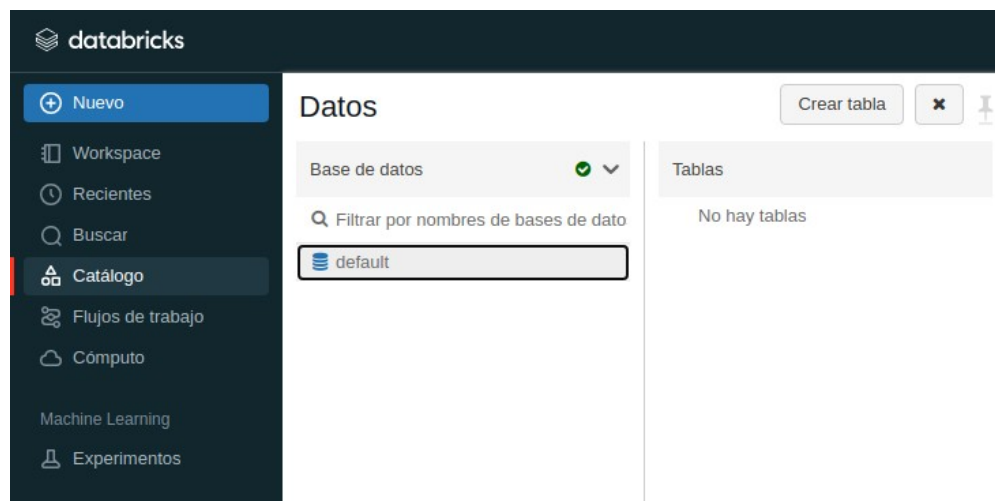
Usaremos **Spark** en *Databricks*, donde configuraremos un clúster para cargar y analizar los datos. Para ello, solo necesitamos ir a la sección "Compute" en el panel izquierdo.



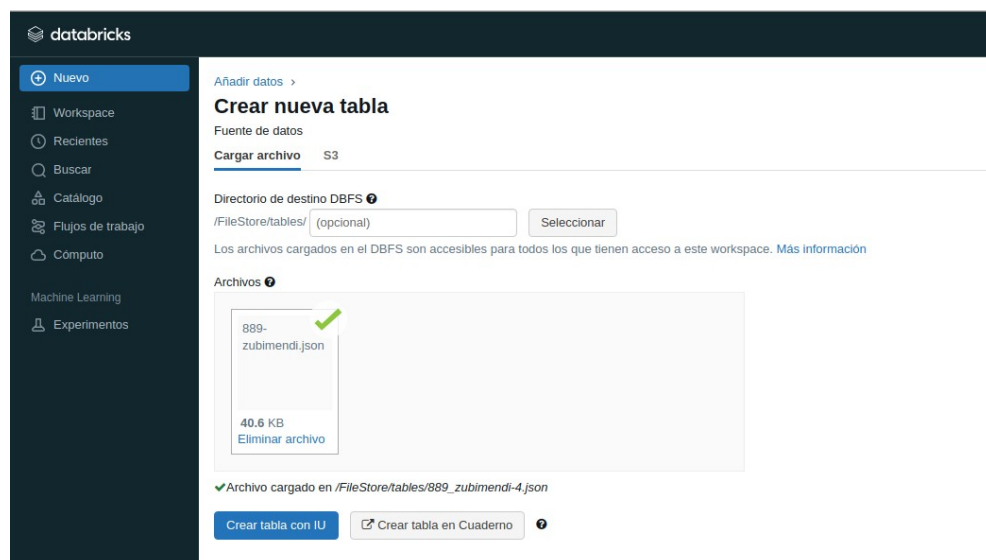
Una vez estamos aquí situados, creamos un cómputo/cluster y esperamos que se inicie.

Crear cómputo

Para poder analizar los archivos, primero tendremos que cargarlos como tablas. Nos dirigimos a la sección Catalog, elegimos la base de datos default, y pinchamos en **Crear Tabla**.



Aquí elegimos los archivos que queremos cargar y seguimos las instrucciones.



Tras haber elegido el/los archivos que queremos cargar, podemos crear un notebook/cuaderno desde el cual analizar el archivo:



Lo primero que haremos en el notebook es importar el archivo JSON que hemos cargado como ejemplo, leerlo y crear un dataframe para comprobar que los datos se recogen correctamente.

itasySpark Python

ivo Editar Ver Ejecutar Ayuda Última edición hace 2 minutos

Ejecutar todo Dario Panadero Fe

▶

✓

Hace 2 minutos (3 s)

2

```
1 from pyspark.sql import SparkSession
2
3 #Creamos una sesión de Spark
4 spark = SparkSession.builder.appName("LecturaJSON").getOrCreate()
5
6 #Indicamos la ruta del JSON en DBFS(DataBricks File System)
7 ruta_dbfs_json = "/FileStore/tables/889_zubinendi.json"
8
9 #Leemos el JSON en un DataFrame
10 df = spark.read.option("multiline", "true").json(ruta_dbfs_json)
11
12 #Mostramos el DataFrame
13 df.show()
```

Salida Terminal Consola de depuración

▶ df: pyspark.sql.dataframe.DataFrame = [averagePoints: double, id: string ... 13 campos adicionales]

averagePoints	id	images	lastSeasonPoints	marketValue	name	nickname	playerStats	playerStatus	points	position	positionId	slug	team	weekPoints
6.176470588235294	889	[[https://assets-...	183]	46061036	Martin Zubinendi	Zubinendi	[[false, [[1, 0],...	ok	105]	Centrocampista	3]	zubinendi	[[https://assets-f...	6]

No obstante, es más interesante a nivel estructural mostrar las estadísticas básicas usando el comando ***'dataframe.describe()'***. Esto nos muestra los nombres de las columnas del archivo y el tipo de dato que incluye

Finalmente, vamos a mostrar el esquema; la parte más interesante del apartado de Spark en este proyecto. Para esto, se usará la funcion ***'dataframe.printSchema()'***.

En la siguiente página se muestra el esquema que devuelve la función

root

```
-- averagePoints: double (nullable = true)
-- id: string (nullable = true)
-- images: struct (nullable = true)
|  -- beat: struct (nullable = true)
|  |  -- 1024x1024: string (nullable = true)
|  |  -- 128x128: string (nullable = true)
|  |  -- 2048x2048: string (nullable = true)
|  |  -- 256x256: string (nullable = true)
|  |  -- 512x512: string (nullable = true)
|  |  -- 64x64: string (nullable = true)
|  -- big: struct (nullable = true)
|  |  -- 1024x1113: string (nullable = true)
|  |  -- 128x139: string (nullable = true)
|  |  -- 2048x2225: string (nullable = true)
|  |  -- 256x278: string (nullable = true)
|  |  -- 512x556: string (nullable = true)
|  |  -- 64x70: string (nullable = true)
|  -- transparent: struct (nullable = true)
|  |  -- 1024x1024: string (nullable = true)
|  |  -- 128x128: string (nullable = true)
|  |  -- 2048x2048: string (nullable = true)
|  |  -- 256x256: string (nullable = true)
|  |  -- 512x512: string (nullable = true)
|  |  -- 64x64: string (nullable = true)
-- lastSeasonPoints: long (nullable = true)
-- marketValue: long (nullable = true)
-- name: string (nullable = true)
-- nickname: string (nullable = true)
-- playerStats: array (nullable = true)
|  -- element: struct (containsNull = true)
|  |  -- isInIdealFormation: boolean (nullable = true)
|  |  -- stats: struct (nullable = true)
|  |  |  -- ball_recovery: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- effective_clearance: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- goal_assist: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- goals: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- goals_conceded: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- marca_points: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- mins_played: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- offtarget_att_assist: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- own_goals: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- pen_area_entries: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- penalty_conceded: array (nullable = true)
|  |  |  |  -- element: long (containsNull = true)
|  |  |  -- penalty_failed: array (nullable = true)
```

```

|-- poss_lost_all: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- red_card: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- saves: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- second_yellow_card: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- total_scoring_att: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- won_contest: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- yellow_card: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- totalPoints: long (nullable = true)
|-- weekNumber: long (nullable = true)
-- playerStatus: string (nullable = true)
-- points: long (nullable = true)
-- position: string (nullable = true)
-- positionId: long (nullable = true)
-- slug: string (nullable = true)
-- team: struct (nullable = true)
|   |-- badgeColor: string (nullable = true)
|   |-- badgeGray: string (nullable = true)
|   |-- badgeWhite: string (nullable = true)
|   |-- dspId: long (nullable = true)
|   |-- id: string (nullable = true)
|   |-- name: string (nullable = true)
|   |-- shortName: string (nullable = true)
|   |-- slug: string (nullable = true)
|   |-- store: string (nullable = true)
-- weekPoints: long (nullable = true)

```

Esta estructura contiene un montón de apartados y subapartados. Mejor realicemos un resumen para poder comprenderlo más fácilmente

ATRIBUTOS GENERALES DEL JUGADOR:

- **averagePoints:** Puntos promedio del jugador (decimal).
- **id:** Identificador único del jugador (texto).
- **name** y **nickname:** Nombre y apodo del jugador.
- **position** y **positionId:** Posición en el campo y su identificador.
- **playerStatus:** Estado actual del jugador (activo, lesionado, etc.).
- **marketValue:** Valor de mercado actual (número entero).
- **points** y **weekPoints:** Puntos totales acumulados y puntos obtenidos en la última semana.

ESTADÍSTICAS DE TEMPORADAS Y PARTIDOS:

- **lastSeasonPoints:** Puntos obtenidos en la última temporada.
- **playerStats:** Lista de estadísticas por partido, incluyendo:
 - **isInIdealFormation:** Si fue parte de la formación ideal de la semana.
 - **stats:** Detalles de desempeño en distintas categorías como:
 - **goals, goal_assist, own_goals:** Goles, asistencias y goles en propia puerta.
 - **mins_played:** Minutos jugados.
 - **penalty_conceded, penalty_save, penalty_won:** Estadísticas relacionadas con penales.

- **red_card, yellow_card, second_yellow_card:** Tarjetas obtenidas.
- **saves y effective_clearance:** Salvadas y despejes efectivos.

IMÁGENES ASOCIADAS:

- **images:** Contiene tres categorías de imágenes del jugador:
 - **beat, big, transparent:** Varias resoluciones para cada categoría (desde 64x64 hasta 2048x2048).

INFORMACIÓN DEL EQUIPO:

- **team:** Datos del equipo al que pertenece, como:
 - **name y shortName:** Nombre completo y abreviado.
 - **badgeColor, badgeGray, badgeWhite:** Distintos formatos del escudo del equipo.
 - **store:** URL o identificador relacionado con la tienda del equipo.

Este esquema proporciona una visión completa del rendimiento, estado y detalles visuales del jugador, lo que permite un análisis detallado de su impacto en la liga fantasy.

Es interesante destacar que todos estas “stats” están formadas por un array de dos valores. Por ejemplo:

```
"stats": {
  "mins_played": [55, 1],
  "goals": [2, 5],
  "goal_assist": [0, 0],
}
```

El primer valor del array representa la *literalidad del campo*, es decir, que ha jugado 55 minutos, que ha marcado 2 goles, y que no ha dado asistencias de gol. Mientras que, el segundo valor de cada array representa la *conversión* a puntos *fantasy* en base a esas estadísticas.

5. HDFS-HIVE: ANÁLISIS DE LOS CSV

SUBIDA HDFS

Por otro lado, nos disponemos a subir al sistema HDFS los archivos CSV players y players-performance, para posteriormente volcar sus datos en una tabla Hive y analizar los datos. Primero de todo, debemos iniciar los procesos HDFS en nuestro terminal, y comprobar que se han iniciado correctamente mediante ‘jps’

```
hadoop@eco-hadoop1:~$ jps
14548 Jps
hadoop@eco-hadoop1:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [eco-hadoop1]
Starting datanodes
Starting secondary namenodes [eco-hadoop1]
Starting resourcemanager
Starting nodemanagers
hadoop@eco-hadoop1:~$ jps
15041 NameNode
15586 ResourceManager
15322 SecondaryNameNode
15162 DataNode
15694 NodeManager
16079 Jps
```


Ahora, procedemos a crear una carpeta en la cual se almacenarán los archivos:

```
hdfs dfs -mkdir -p /data/fantasy/laliga
```

Comprobamos que se ha creado correctamente

```
hadoop@eco-hadoop1:~$ hdfs dfs -ls /data/fantasy
Found 1 items
drwxr-xr-x    - hadoop supergroup          0 2025-01-13 13:59 /data/fantasy/laliga
hadoop@eco-hadoop1:~$
```

Ahora es el momento de subir los archivos a esa carpeta

```
hdfs dfs -put /home/hadoop/Documentos/Trabajo_Fantasy/laliga/csv/players.csv
/data/fantasy/laliga/
```

```
hdfs dfs -put /home/hadoop/Documentos/Trabajo_Fantasy/laliga/csv/players-performance.csv
/data/fantasy/laliga/
```

Igual que con la carpeta, nos aseguramos de que los archivos se subieron correctamente

```
hadoop@eco-hadoop1:~$ hdfs dfs -ls /data/fantasy/laliga
Found 2 items
-rw-r--r--    1 hadoop supergroup    158140 2025-01-13 14:08 /data/fantasy/laliga/players-performance.csv
-rw-r--r--    1 hadoop supergroup    30928 2025-01-13 14:06 /data/fantasy/laliga/players.csv
hadoop@eco-hadoop1:~$
```

Finalmente, aprovechamos el **Browse Directory** que nos ofrece Hadoop Home para comprobar visualmente que los archivos mantienen su contenido una vez subidos a HDFS

The screenshot shows the Hadoop Browse Directory interface. A modal window titled "File information - players-performance.csv" is open, displaying details for the file. The modal includes a "Download" button, links to "Head the file (first 32K)" and "Tail the file (last 32K)", and a "Block information" section. The block information shows: Block ID: 1073744080, Block Pool ID: BP-292605565-10.0.2.15-1720712920174, Generation Stamp: 3256, Size: 158140, and Availability: eco-hadoop1. Below this, the "File contents" section shows a preview of the CSV data, including columns like ID, SLUG, EQUIPO_ID, SEMANA, PUNTOS_TOTALES, MIN_JUGADOS, GOLES, ASISTENCIA, S_GOL, ASISTENCIAS_SIN_GOL, LLEGADAS_AREA, PENALTIS_PROVOCADOS, PENALTIS_PARADOS, PARADAS, DESPEJES, PENALTIS_FALLADOS, GOLES_EN_PROPIA, GOLES_EN_CONTRA, TARJETAS_AMARILLAS, SEGUNDAS_AMARILLAS, TARJETAS_ROJAS, TIROS_A_PUERTA, REGATES, BALONES_RECUPERADOS, POSESIONES_PERDIDAS, PUNTOS_MARCA, MIN_JUGADOS_PUNTOS, GOLES_PUNTOS, ASISTENCIAS_GOL_PUNTOS, ASISTENCIAS_SIN_GOL_PUNTOS, LLEGADAS_AREA_PUNTOS, PENALTIS_PROVOCADO_PUNTOS, PENALTIS_PARADOS_PUNTOS, PARADAS_PUNTOS, DESPEJES_PUNTOS, P.

Mostramos que se haya creado, y nos situamos en ella

```
0: jdbc:hive2://> show databases;
OK
+-----+
| database_name |
+-----+
| bigdata      |
| default      |
| ejemplo      |
| fantasydb    |
+-----+
4 rows selected (1.066 seconds)
```

USE fantasydb;

Una vez estamos en la base de datos, creamos ambas tablas con todos los campos de los **csv** correspondientes:

TABLA PLAYERS

//Crear la tabla con todos los campos del archivo 'players.csv'

```
CREATE TABLE IF NOT EXISTS players (
id INT,
equipo_id STRING,
puntos INT,
puntos_media DOUBLE,
nombre STRING,
apodo STRING,
slug STRING,
posicion_id INT,
posicion STRING,
valor_mercado BIGINT,
estado_jugador STRING,
-- Columnas SEMANA_1 a SEMANA_38
semana_1 INT,
semana_2 INT,
semana_3 INT,
semana_4 INT,
...
semana_36 INT,
semana_37 INT,
semana_38 INT,
-- Columnas Agregadas SEMANA_1_AGGR a SEMANA_38_AGGR
semana_1_aggr INT,
semana_2_aggr INT,
semana_3_aggr INT,
semana_4_aggr INT,
...
semana_36_aggr INT,
semana_37_aggr INT,
semana_38_aggr INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

TABLA PLAYERS_PERFORMANCE

//Crear la tabla con todos los campos del archivo 'players-performance.csv'

```
CREATE TABLE IF NOT EXISTS players_performance (  
  ID INT,  
  slug STRING,  
  equipo_id STRING,  
  semana INT,  
  puntos_totales INT,  
  min_jugados INT,  
  goles INT,  
  asistencias_gol INT,  
  asistencias_sin_gol INT,  
  llegadas_area INT,  
  penaltis_provocados INT,  
  penaltis_parados INT,  
  paradas INT,  
  despejes INT,  
  penaltis_fallados INT,  
  goles_en_propia INT,  
  goles_en_contra INT,  
  tarjetas_amarillas INT,  
  segundas_amarillas INT,  
  tarjetas_rojas INT,  
  tiros_a_puerta INT,  
  regates INT,  
  balones_recuperados INT,  
  posesiones_perdidas INT,  
  puntos_marca INT,  
  min_jugados_puntos INT,  
  goles_puntos INT,  
  asistencias_gol_puntos INT,  
  asistencias_sin_gol_puntos INT,  
  llegadas_area_puntos INT,  
  penaltis_provocados_puntos INT,  
  penaltis_parados_puntos INT,  
  paradas_puntos INT,  
  despejes_puntos INT,  
  penaltis_fallados_puntos INT,  
  goles_en_propia_puntos INT,  
  goles_en_contra_puntos INT,  
  tarjetas_amarillas_puntos INT,  
  segundas_amarillas_puntos INT,  
  tarjetas_rojas_puntos INT,  
  tiros_a_puerta_puntos INT,  
  regates_puntos INT,  
  balones_recuperados_puntos INT,  
  posesiones_perdidas_puntos INT,  
  puntos_marca_puntos INT  
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

Finalmente, se cargan los datos de los archivos en las tablas que acabamos de crear

```
LOAD DATA INPATH '/data/fantasy/laliga/players.csv' INTO TABLE players;
```

```
0: jdbc:hive2://> LOAD DATA INPATH '/data/fantasy/laliga/players.csv' INTO TABLE players;  
25/01/13 17:33:47 [HiveServer2-Background-Pool: Thread-196]: WARN metastore.ObjectStore: data  
Loading data to table fantasydb.players  
25/01/13 17:33:48 [HiveServer2-Background-Pool: Thread-196]: WARN metastore.ObjectStore: data  
OK  
No rows affected (1.178 seconds)  
0: jdbc:hive2://> █
```

```
LOAD DATA INPATH '/data/fantasy/laliga/players-performance.csv' INTO TABLE players_performance;
```

```
0: jdbc:hive2://> LOAD DATA INPATH '/data/fantasy/laliga/players-performance.csv' INTO TABLE players_performance;
Loading data to table fantasydb.players_performance
25/01/15 20:00:00 [HiveServer2-Background-Pool: Thread-202]: WARN metastore.ObjectStore: datanucleus.autoStartMechan
25/01/15 20:00:00 [HiveServer2-Background-Pool: Thread-202]: WARN metastore.ObjectStore: datanucleus.autoStartMechan
OK
No rows affected (0.295 seconds)
0: jdbc:hive2://>
```

ANÁLISIS HIVE

En esta sección, se describen las principales consultas realizadas para analizar las estadísticas de los jugadores y extraer información relevante para el proyecto. Estas consultas se han llevado a cabo utilizando las tablas generadas en Apache Hive tras la carga de los datos desde HDFS. A continuación, se presentan algunos ejemplos de las consultas más representativas:

1. Jugadores con el mejor promedio de puntos

Se realizó una consulta para identificar a los jugadores con el promedio de puntos más alto por jornada y sus respectivos valores de mercado. Esta métrica es fundamental para detectar a los jugadores más valiosos en términos de rendimiento. La consulta emplea funciones de agregación para calcular el promedio de puntos por jugador:

//Top 15 jugadores con más media de puntuación

```
SELECT NOMBRE, EQUIPO_ID, PUNTOS_MEDIA, VALOR_MERCADO
FROM players
ORDER BY PUNTOS_MEDIA DESC
LIMIT 15;
```

nombre	equipo_id	puntos_media	valor_mercado
"Vinicius Júnior"	RMA	9.5	115355760
"Antoine Griezmann"	ATM	8.235294117647058	106983361
"Fede Valverde"	RMA	8.058823529411764	101978402
"Jan Oblak"	ATM	7.5625	47195223
"Rodrygo Silva de Goes"	RMA	7.357142857142857	58760342
"Marcos Llorente"	ATM	7.083333333333333	50339706
"Alejandro Remiro Gargallo"	RSO	7.055555555555555	32967729
"Thibaut Courtois"	RMA	7.0	53244101
"Oihan Sancet Tirapu"	ATH	6.933333333333334	58859779
"Pedro González"	BAR	6.777777777777778	82901227
"Ante Budimir"	OSA	6.647058823529412	61576983
"Luis Milla Manzanares"	GET	6.588235294117647	44426184
"Iago Aspas Juncal"	CEL	6.5625	47251849
"Iñaki Williams Arthuer"	ATH	6.352941176470588	63182767
"Martín Zubimendi"	RSO	6.176470588235294	46061036

15 rows selected (55.299 seconds)

2. Media de puntos de todos los jugadores

Otra consulta fundamental en el análisis estadístico consistió en calcular la media de puntos obtenidos por todos los jugadores en el juego fantasy. Esto permitió evaluar el rendimiento promedio y sirvió como referencia para identificar a los jugadores que destacan por encima o por debajo del estándar general:

//Media de puntos de todos los jugadores

```
SELECT AVG(PUNTOS_MEDIA) AS media_puntos FROM players;
```

```
+-----+
|  media_puntos  |
+-----+
| 3.495727528039619 |
+-----+
1 row selected (32.574 seconds)
0: jdbc:hive2://>
```

- La consulta devuelve un valor único, **media_puntos**, que representa el promedio de los puntos obtenidos por todos los jugadores en el sistema.
- Este indicador global proporciona una perspectiva inicial sobre la distribución de los puntos en la liga fantasy.

3. Top 10 jugadores por minutos totales

Una consulta destacada del análisis consistió en identificar a los 10 jugadores con mayor cantidad de minutos jugados en total. Esto es relevante porque permite observar qué jugadores son piezas fundamentales para sus equipos y tienen un papel consistente en el campo.

//Top 10 jugadores por minutos totales

```
SELECT ID, slug AS JUGADOR, SUM(min_jugados) AS
total_minutos
FROM players_performance
GROUP BY ID, slug
ORDER BY total_minutos
DESC LIMIT 10;
```

```
+-----+-----+-----+
| id | jugador | total_minutos |
+-----+-----+-----+
| 988 | sivera | 1530 |
| 274 | remiro | 1530 |
| 644 | s-herrera | 1530 |
| 184 | soria-david | 1530 |
| 853 | l-milla | 1515 |
| 75 | martinez-i | 1512 |
| 243 | federico-valverde | 1473 |
| 889 | zubimendi | 1445 |
| 53 | oblak | 1440 |
| 566 | kounde | 1435 |
+-----+-----+-----+
10 rows selected (54.619 seconds)
0: jdbc:hive2://>
```

Como podemos observar, en esta estadística destacan los porteros, ya que los 4 primeros y 5 de los 10 jugadores con más minutos totales ocupan esta posición en el campo (Sivera, Remiro, Sergio Herrera, David Soria y Jan Oblak). Esto se debe a que en la posición de portero no suele haber tantas rotaciones entre partido y partido (ni tantas lesiones) como en los jugadores de campo.

4. 10 jugadores con más tarjetas amarillas recibidas

Este apartado se centra en analizar a los jugadores con mayor número de tarjetas amarillas acumuladas. Aunque esta métrica suele ser vista como negativa, también puede indicar roles defensivos clave o jugadores que asumen riesgos estratégicos para frenar al rival.

//Top 10 jugadores por N° de tarjetas amarillas

```
SELECT ID, SLUG AS JUGADOR, SUM(TARJETAS_AMARILLAS)
AS TOTAL_AMARILLAS
FROM players_performance
GROUP BY ID, SLUG
ORDER BY TOTAL_AMARILLAS DESC
LIMIT 10;
```



id	jugador	total_amarillas
873	lucas-torro	6
310	dani-parejo	6
193	mauro-arambarri	5
82	unai-lopez	5
637	jon-moncayola	5
192	dakonam	5
247	vinicius-1	5
574	albiol	5
265	igor-zubeldia	4
995	yeremi	4

10 rows selected (88.227 seconds)

Roles en el equipo:

- En línea con lo que devuelve esta consulta, se destaca que la mayoría de jugadores con más tarjetas son o bien defensas o mediocentros con alta responsabilidad defensiva, además de jugadores con tendencia a protestar (Vinicius Jr, Yeremi Pino o Dani Parejo)
- Es posible que estos jugadores también sean los encargados de realizar faltas tácticas en momentos críticos.

Impacto en el rendimiento fantasy:

- Las tarjetas amarillas afectan negativamente la puntuación en el fantasy, ya que generalmente restan puntos al jugador.
- Identificar a los jugadores más propensos a recibir tarjetas puede ser un factor a tener en cuenta a la hora de seleccionar jugadores para tu plantilla. Pues al margen de la penalización en cuanto a puntos que ello supone, cada 5 tarjetas amarillas los jugadores se pierden 1 partido por sanción

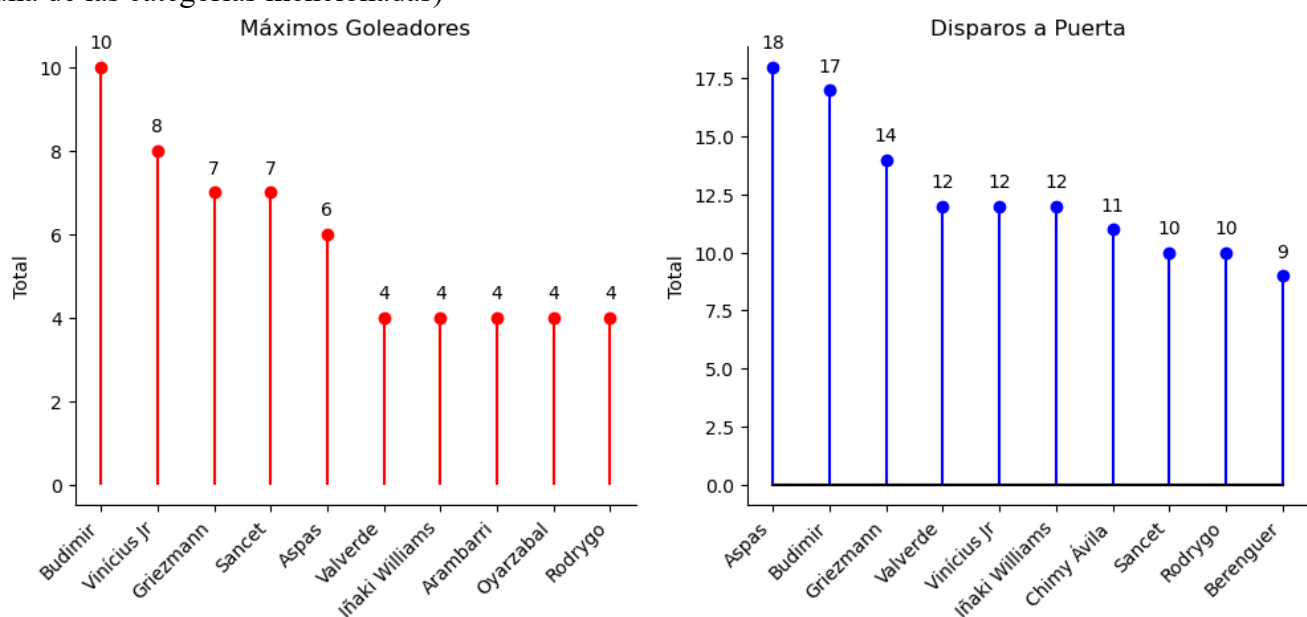
6. PYTHON (JUPYTERLAB) - VISUALIZACIÓN GRÁFICA

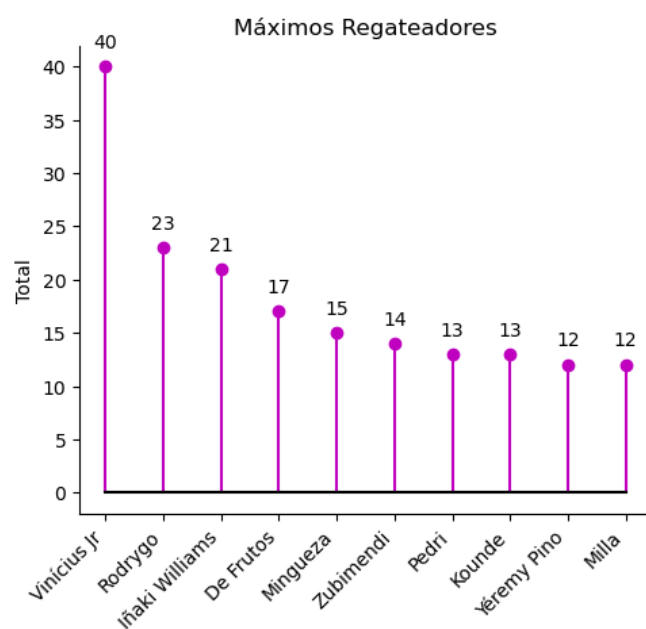
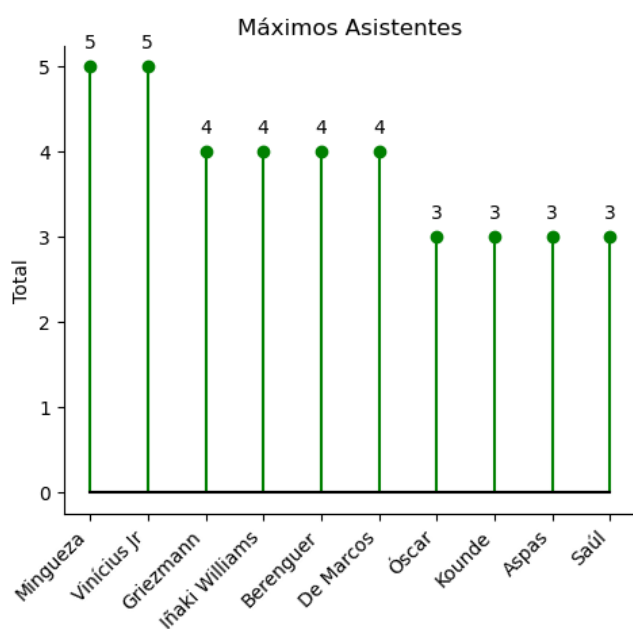
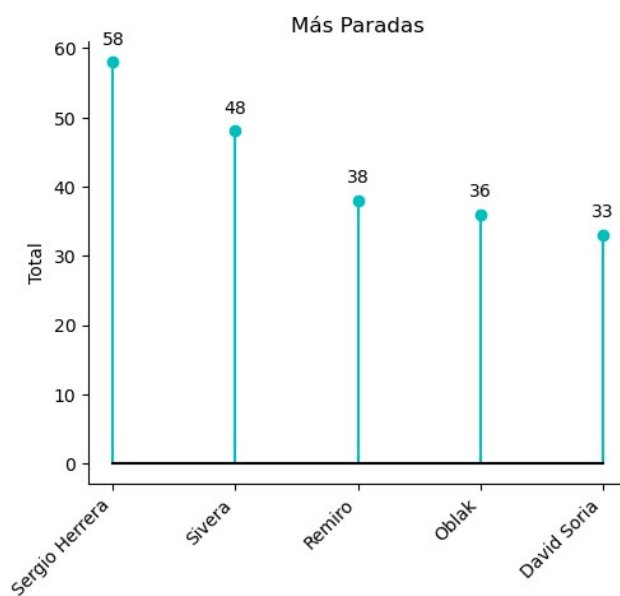
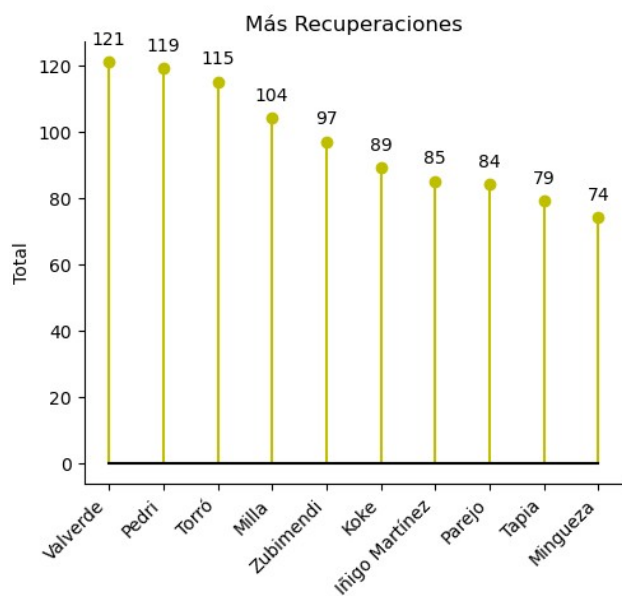
Hemos llegado a la sección más apasionante de este proyecto, la generación de gráficos en base a los datos y análisis que hemos obtenido

RANKINGS DE JUGADORES

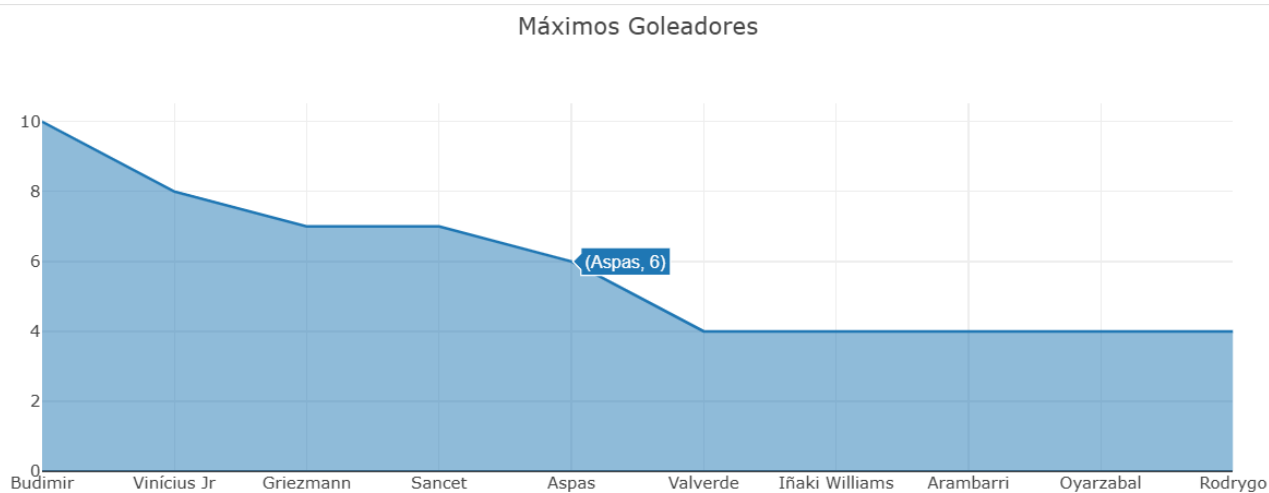
El primer objetivo en esta etapa será encontrar algo que siempre genera interés, no solo en el mundo del fantasy, sino también entre los aficionados de LaLiga: los jugadores más destacados. Para comenzar, nos centraremos en identificar a los máximos goleadores, los que más disparos realizan a puerta, los mejores asistentes, regateadores, recuperadores y aquellos con más paradas.

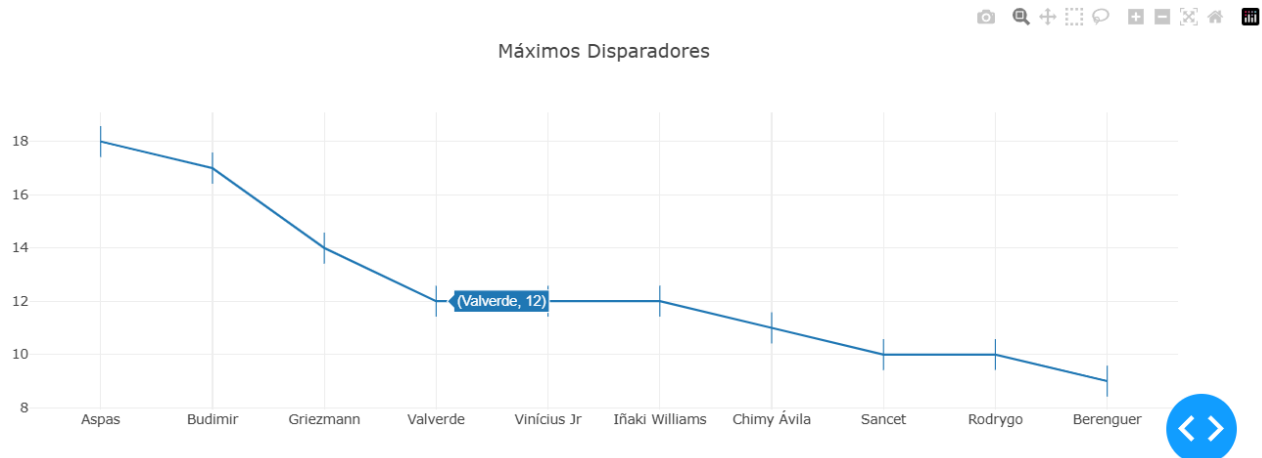
Como resultado, obtenemos una figura compuesta por 6 subplots (Cada uno de ellos es el ranking de cada una de las categorías mencionadas)





Otra forma de realizar este gráfico es mediante el uso de **Dash**





COMPARATIVA DE POSICIONES

Como managers de fútbol, debemos elegir una alineación (por ejemplo, 4-4-2 o 4-3-3) considerando que el portero es obligatorio, y los otros números indican defensas, centrocampistas y delanteros. Aunque los delanteros suelen ser importantes por marcar goles, esta aplicación también ofrece estadísticas defensivas. Esto plantea la pregunta: ¿es siempre mejor tener más delanteros en nuestra alineación? Para responder, realizaremos un análisis de las puntuaciones medias de los delanteros:

- Importamos las bibliotecas necesarias: `os`, `pandas`, y `json`, junto con `json_normalize` de `pandas`.
- Definimos la carpeta de archivos JSON como `./json_players`.
- Iteramos sobre los archivos de la carpeta, procesando solo los que tengan extensión `.json`.
- Normalizamos cada archivo con `json_normalize` y almacenamos los resultados en una lista de `DataFrames`.
- Combinamos los `DataFrames` en uno solo usando `pd.concat`.
- Calculamos la puntuación promedio por posición con `groupby` y `mean`.
- Visualizamos los resultados en un gráfico de barras con `matplotlib.pyplot`.

El código para generar el gráfico se ve así:

```
import os
import pandas as pd
import json
from pandas import json_normalize
import matplotlib.pyplot as plt

# Especifica la ruta de la carpeta que contiene los archivos JSON
carpeta = '../..marca-fantasy-scraper-master/laliga/json_players'

# Lista para almacenar los DataFrames de cada archivo
dataframes = []

# Recorre todos los archivos en la carpeta
for filename in os.listdir(carpeta):
    if filename.endswith('.json'):
        filepath = os.path.join(carpeta, filename)

        # Lee el archivo JSON y normaliza la estructura
        with open(filepath, 'r') as file:
            data = json.load(file)
            df_temp = json_normalize(data)
```

```

# Agrega el DataFrame a la lista
dataframes.append(df_temp)

# Concatena todos los DataFrames en uno solo
df = pd.concat(dataframes, ignore_index=True)

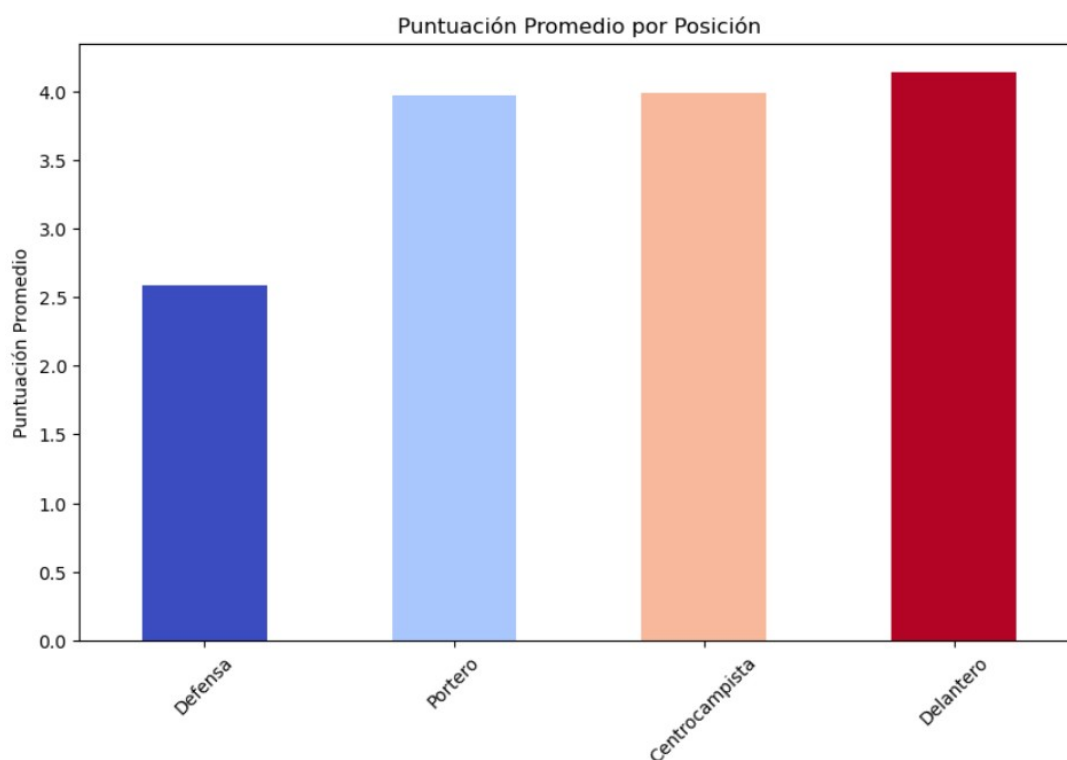
# Calcula la puntuación promedio por posición
avg_points_by_position = df.groupby('position')['averagePoints'].mean().sort_values()

# Define una lista de colores para las barras
colores = plt.cm.get_cmap("coolwarm", len(avg_points_by_position))

# Visualiza los resultados con un gráfico de barras y asigna colores
plt.figure(figsize=(10, 6))
avg_points_by_position.plot(kind='bar', color=colores(range(len(avg_points_by_position))))
plt.title('Puntuación Promedio por Posición')
plt.xlabel('Posición')
plt.ylabel('Puntuación Promedio')
plt.xticks(rotation=45)
plt.show()

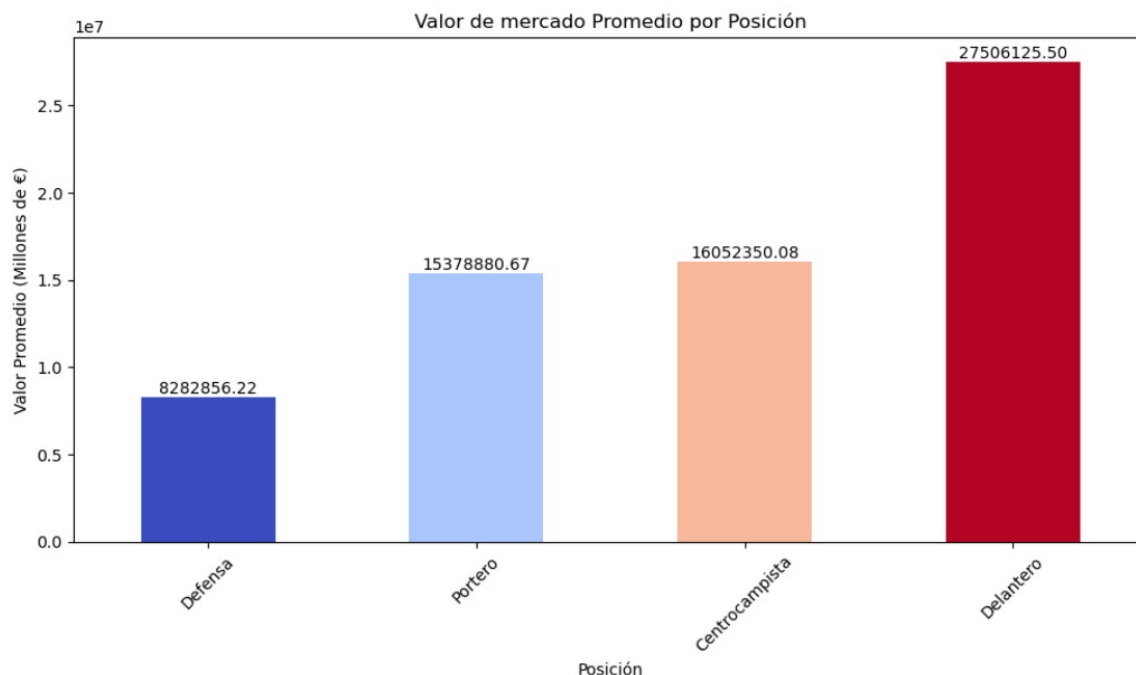
```

Este enfoque nos permite evaluar las posiciones con mejor rendimiento para optimizar nuestras alineaciones.



Según vemos, efectivamente los *delanteros* tienen una mayor puntuación promedio, seguidos muy de cerca por los *centrocampistas* y *porteros*

No obstante, esto no es todo. También tenemos que analizar cuál es el valor de mercado promedio para cada posición para poder determinar que posición es la '*más rentable*' del juego a nivel general.



Claramente los jugadores más caros a nivel global parecen ser los delanteros, con unos 27,5M de media. Le siguen muy de lejos porteros (15,38M) y centrocampistas (16,05M), y finalmente se sitúan los defensas con un valor promedio de 8,28M.

En sintonía con esta información, podemos concluir que estadísticamente las posiciones más rentables son las de Portero y Centrocampista. Por tanto, nos interesaría tener alineaciones con el mayor número posible de centrocampistas (3-5-2, 4-5-1 o incluso una 3-4-3), debido a que tienen prácticamente la misma puntuación media que los delanteros y su valor de mercado promedio es notablemente inferior (en torno a un 58,3% del valor de los atacantes)

Vamos a mostrar el código empleado para generar este gráfico

```
import os
import pandas as pd
import json
from pandas import json_normalize
import matplotlib.pyplot as plt

# Especifica la ruta de la carpeta que contiene los archivos JSON
carpeta = '../..\\marca-fantasy-scraper-master\\laliga\\json_players'

# Lista para almacenar los DataFrames de cada archivo
dataframes = []

# Recorre todos los archivos en la carpeta
for filename in os.listdir(carpeta):
    if filename.endswith('.json'):
        filepath = os.path.join(carpeta, filename)

        # Lee el archivo JSON y normaliza la estructura
        with open(filepath, 'r') as file:
            data = json.load(file)
            df_temp = json_normalize(data)

        # Agrega el DataFrame a la lista
        dataframes.append(df_temp)
```

```

# Concatena todos los DataFrames en uno solo
df = pd.concat(dataframes, ignore_index=True)

# Calcula el valor de mercado promedio por posición
avg_value_by_position = df.groupby('position')['marketValue'].mean().sort_values()

# Define el mapa de colores (puedes cambiar 'coolwarm' por cualquier otro mapa)
cmap = plt.get_cmap("coolwarm", len(avg_value_by_position))

# Genera una lista de colores a partir del mapa
colores = [cmap(i) for i in range(len(avg_value_by_position))]

# Visualiza los resultados con un gráfico de barras y asigna los colores
plt.figure(figsize=(10, 6))
ax = avg_value_by_position.plot(kind='bar', color=colores)

# Añade etiquetas encima de cada barra
for i, value in enumerate(avg_value_by_position):
    ax.text(i, value, f'{value:.2f}', ha='center', va='bottom', fontsize=10)

# Configura los títulos y etiquetas
plt.title('Valor de mercado Promedio por Posición')
plt.xlabel('Posición')
plt.ylabel('Valor Promedio (Millones de €)')
plt.xticks(rotation=45) # Ajusta la rotación de las etiquetas del eje x
plt.tight_layout() # Ajusta automáticamente los márgenes para evitar cortes

# Muestra el gráfico
plt.show()

```

VALORES DE MERCADO

Para finalizar el análisis gráfico, atendemos la relación entre el valor de mercado de los jugadores y su puntuación media en el juego Fantasy. Este análisis es clave para comprender si existe una correlación entre el rendimiento de los jugadores y su precio, lo que puede ser una herramienta útil para optimizar las estrategias de compra y venta en el mercado del Fantasy.

Se genera un gráfico de dispersión utilizando **Plotly Express**, donde:

- El eje X representa la puntuación media de los jugadores.
- El eje Y representa el último valor de mercado.
- Cada punto en el gráfico representa a un jugador, y se incluye un hover interactivo que muestra su nombre.

Antes de generar el gráfico se eliminan jugadores con una puntuación media igual a 0, ya que estos no aportan información relevante para el análisis, y se organiza la información en un DataFrame para facilitar el procesamiento.

El código para generarlo es el siguiente:

```
import os
import json
import pandas as pd
import plotly.express as px

# Ruta de la carpeta que contiene archivos JSON
ruta_carpeta = "../../../marca-fantasy-scraper-master/laliga/json_players"

# Lista para almacenar los datos de cada jugador
jugadores = []

# Iterar sobre cada archivo en la carpeta
for archivo_json in os.listdir(ruta_carpeta):
    # Excluir archivos o carpetas específicas
    if archivo_json == '.ipynb_checkpoints':
        continue

    # Combinar la ruta completa
    ruta_completa = os.path.join(ruta_carpeta, archivo_json)

    # Leer el archivo JSON
    with open(ruta_completa, 'r') as f:
        datos_jugador = json.load(f)

    # Calcular la puntuación media del jugador
    puntuacion_media = datos_jugador.get('averagePoints', 0)

    # Extraer el nombre del jugador y el valor de mercado
    nombre_jugador = datos_jugador.get('nickname', 'Desconocido')

    # Tomar marketValue directamente como valor numérico
    ultimo_valor_mercado = datos_jugador.get('marketValue', None)

    # Agregar los datos a la lista
    jugadores.append({
        'Nombre': nombre_jugador,
        'PuntuacionMedia': puntuacion_media,
        'UltimoValorMercado': ultimo_valor_mercado
    })

# Crear un DataFrame a partir de la lista
df = pd.DataFrame(jugadores)

# Eliminar los jugadores con puntuación media igual a cero
df = df[df['PuntuacionMedia'] > 0]

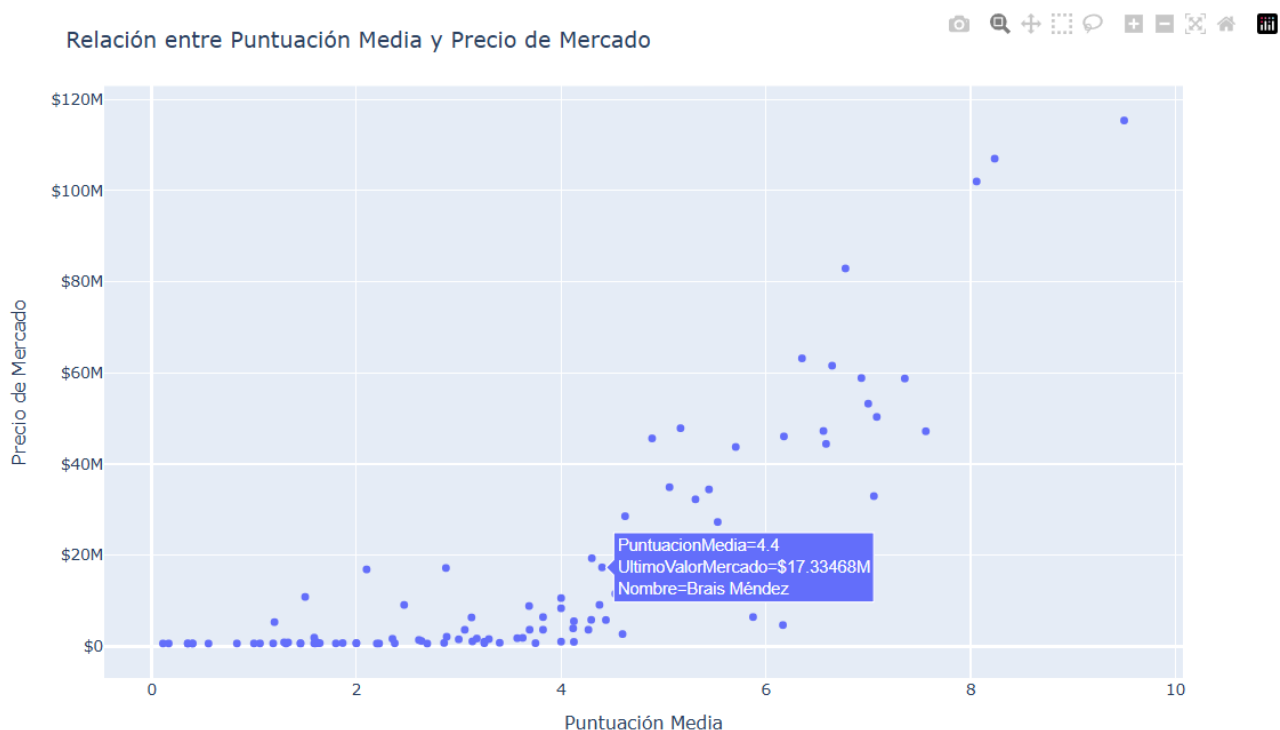
# Configurar la visualización interactiva con Plotly Express
fig = px.scatter(df,
                 x='PuntuacionMedia',
                 y='UltimoValorMercado',
                 hover_data=['Nombre'])

# Configurar el eje y para que se vea el precio completo
fig.update_yaxes(tickprefix="$")
```

```
# Personalizar el diseño del gráfico
fig.update_layout(
    title='Relación entre Puntuación Media y Precio de Mercado',
    xaxis_title='Puntuación Media',
    yaxis_title='Precio de Mercado',
    hovermode='closest',
    width=1000,
    height=600
)

# Mostrar el gráfico interactivo
fig.show()

# Guardar el gráfico interactivo en un archivo HTML
fig.write_html("relacion_puntuacion_precio.html")
```



Patrones generales:

En la mayoría de los casos, los jugadores con puntuaciones medias altas tienden a tener un mayor valor de mercado.

Sin embargo, también se observan anomalías, como jugadores con un valor elevado pero puntuaciones medias bajas. Esto suele ocurrir con jugadores con mucha popularidad, o que se les presupone un rendimiento mayor del demostrado hasta ahora (Koke, Oyarzabal o Koundé)

Por otro lado, jugadores como Sergio Herrera o Marcos Llorente presentan un caso totalmente contrario

Aplicaciones prácticas:

- **Identificación de oportunidades de mercado:** Jugadores con un valor de mercado relativamente bajo y una puntuación media alta podrían ser adquisiciones estratégicas para maximizar la eficiencia en el presupuesto.

- **Estrategias de venta:** Los jugadores con valores de mercado elevados pero un rendimiento medio bajo podrían ser candidatos para vender antes de que su precio disminuya.

Este análisis se ha visto claramente condicionado por no haber podido extraer datos de todos los jugadores, ya que muchos de los jugadores importantes de la competición no aparecen. Sin embargo, la dinámica es exactamente la misma y los patrones identificados se seguirían aplicando en su mayoría en caso de tener disponibles los datos del resto de jugadores.