

Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio

**Herramientas de software para
Big Data**

**Entregado como requisito para la obtención del título de
Licenciatura en Sistemas**

Guillermo Diotti - 285578

Santiago Pucciarelli - 245481

Juan Romero - 252446

Profesores:

Juan Andrés Rodriguez Pedreira

Alexis Javier Arriola Garcia

2025

Índice

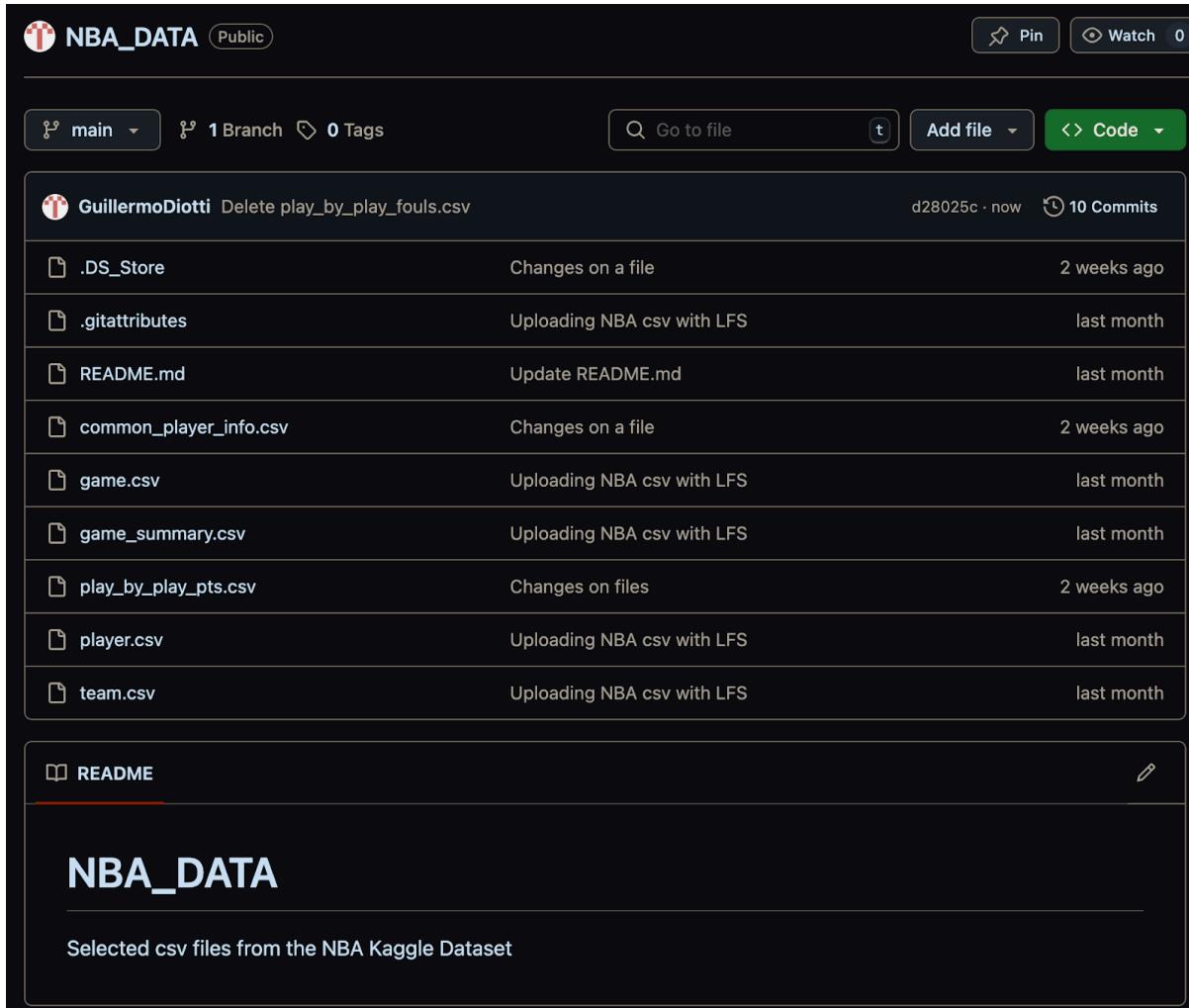
Parte 1.....	4
Introducción.....	4
Stack tecnológico.....	4
Preguntas a relevar.....	5
Preprocesamiento a la ingesta.....	5
La ingesta.....	6
Análisis Exploratorio.....	11
Descripción de los datos.....	11
Game.....	12
Game_summary.....	12
Common_player_info.....	13
Play_by_play_pts.....	14
Player.....	15
Team.....	16
Descripción de la exploración.....	16
Creación de la sesión.....	17
Carga de los dataframes.....	17
Visualización de las columnas.....	18
Implementación de schemas y carga de los dataframes con ellos.....	18
Conversión de datos mal representados.....	19
Eliminación de duplicados.....	19
Validación de datos no duplicados.....	19
Manejo de datos nulos.....	20
Verificación de la eliminación de nulos.....	20
Validación de la unicidad de la clave.....	21
Visualización de los datos finales.....	22
Carga de los dataframes finales.....	22
Validación de el correcto guardado de los datos.....	23
Modelado de datos.....	24
Descripción de la exploración.....	24
Creación de la sesión.....	24
Carga de los dataframes.....	25
Manipulación de los datos.....	25
common_player_info.....	25
game.....	25
play_pts.....	26
player.....	28
Eliminación y renombre de columnas.....	28
Carga del modelamiento final.....	29
Resultado final del modelado.....	29
game.....	30
common_player_info.....	30
play_pts.....	30

player.....	31
team.....	31
Esquema del modelo.....	32
Hive.....	33
Descripción del procesamiento.....	33
Creación de la sesión.....	33
Instalación de herramientas.....	33
Creación de la base de datos.....	34
Creación de las tablas hive.....	34
Script de creación de la tabla hive player.....	35
Script de creación de la tabla hive team.....	35
Script de creación de la tabla hive game.....	36
Script de creación de la tabla hive play_pts.....	36
Script de creación de la tabla hive player_info.....	37
Información de las tablas.....	37
Relevamiento de preguntas.....	38
Consulta 1.....	38
Consulta 2.....	39
Consulta 3.....	39
Consulta 4.....	40
Consulta 5.....	40
Visualización en Pandas.....	41
Visualizaciones de la consulta 1.....	42
Visualizaciones de la consulta 2.....	44
Visualizaciones de la consulta 3.....	46
Creación de tablas Hive para las consultas.....	47
Visualización en SuperSet.....	48
Visualización de la consulta 4.....	48
Visualización de la consulta 5.....	49
Parte 2.....	51
Stack Tecnológico.....	51
AWS DataSync.....	51
Talend.....	51
Talend es una plataforma de integración de datos, capaz de manejar procesos ETL (Extract, Transform, Load). Se puede conectar fácilmente a múltiples fuentes y destinos de datos.....	51
Amazon S3.....	51
Apache Spark.....	52
AWS Glue.....	52
Snowflake.....	52
Amazon Athena.....	52
Amazon Quicksight.....	52
Flujo de datos.....	53
Diagrama.....	54
Aprendizajes Finales.....	54

Parte 1

Introducción

En el presente obligatorio, se trabajará en torno a un dataset seleccionado por el equipo, siendo este el conjunto de datos de basketball de la NBA, seleccionado de la plataforma Kaggle. El dataset incluye una totalidad de 16 tablas (archivos csv). El mismo se puede acceder a través del siguiente enlace; https://github.com/GuilermoDiotti/NBA_DATA.



The screenshot shows a GitHub repository page for 'NBA_DATA'. At the top, it says 'Public' and has options to 'Pin' or 'Watch' the repository. Below that, it shows 'main' (1 Branch, 0 Tags), a search bar ('Go to file'), and buttons for 'Add file' and 'Code'. The main area lists 10 commits by 'GuillermoDiotti' from 'd28025c · now'. The commits include:

- .DS_Store: Changes on a file (2 weeks ago)
- .gitattributes: Uploading NBA csv with LFS (last month)
- README.md: Update README.md (last month)
- common_player_info.csv: Changes on a file (2 weeks ago)
- game.csv: Uploading NBA csv with LFS (last month)
- game_summary.csv: Uploading NBA csv with LFS (last month)
- play_by_play_pts.csv: Changes on files (2 weeks ago)
- player.csv: Uploading NBA csv with LFS (last month)
- team.csv: Uploading NBA csv with LFS (last month)

Below the commits, there's a link to 'README' and a large section titled 'NBA_DATA' containing the text: 'Selected csv files from the NBA Kaggle Dataset'.

Stack tecnológico

A continuación, se enumerarán brevemente las distintas herramientas y tecnologías que se usarán a lo largo del proyecto

- PySpark
- NiFi
- Git LFS
- Github

- Hadoop Distributed File System
- Hive
- SQLite
- DB Browser
- Apache SuperSet

Preguntas a relevar

- **¿Cuál es el top 10, de promedio de puntos de cada equipo en la temporada 2018?**
- **¿Cuántos jugadores, por cada posición, juegan en equipos fundados hace más de 60 años?**
- **¿Cuál es el top 3 de números de camiseta con más triples anotados por jugadores que hayan estudiado en UCLA?**
- **¿Cuál es la cantidad de veces que cada equipo de Nueva York haya ganado como local?**
- **¿Cuál es el promedio de puntos anotados por los jugadores actualmente inactivos por temporadas? Top 10**

Preprocesamiento a la ingesta

Para la ingesta de los datos, los mismos se deberán dejar en un lugar accesible a NiFi, siendo este lugar un repositorio en Github. El problema identificado, es que los archivos con extensión csv del dataset, resultaban demasiado pesados, imposibilitando su transferencia al repositorio. Luego de haber investigado sobre posibles soluciones, se ha optado por una de ellas, debiendo ejecutar una serie de pasos para posibilitar esta acción.

Como primer cometido, debió realizar un análisis exhaustivo de las consultas a relevar, identificando correctamente qué conjuntos de datos se precisarían para poder resolverlas. Identificados estos conjuntos de datos, se procede a la manipulación de los datos, reteniendo únicamente las tablas y columnas en ellas pertinentes.

Aquí es donde se ha aprovechado el hecho de que el dataset de Kaggle incluyera una base de datos SQLite. Por ende, se ha necesitado un software para manipular esta base de datos, DB Browser. En él, se redujeron en gran magnitud los registros, principalmente de la tabla “play_by_play”, la que presenta información de cada acción ocurrida en un partido, desde faltas, puntos anotados, pases y más.

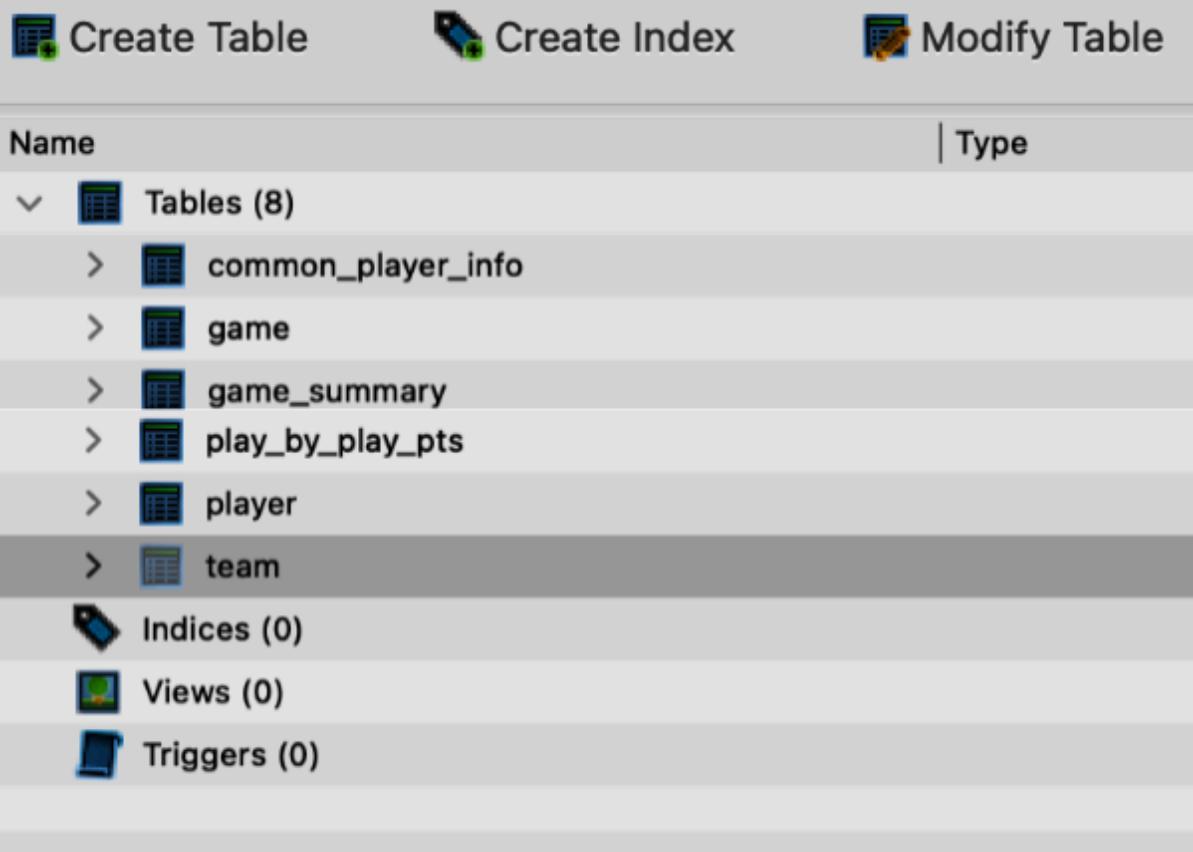
Tras un refinamiento de esta, se extrajeron las tuplas relacionadas con los puntos anotados y las faltas cometidas, ya que son las que ayudarán a resolver las preguntas previamente planteadas. Para este fraccionamiento, se ejecutaron dos consultas.

```

6
7   CREATE TABLE play_by_play_pts AS
8     SELECT *
9       FROM play_by_play
10      WHERE (homedescription LIKE '%PTS%' OR visitordescription LIKE '%PTS%')
11        AND neutraldescription IS NULL;
12

```

Tras todo este procesamiento, el resultado de conjuntos finales es el siguiente:



Name	Type
Tables (8)	
common_player_info	
game	
game_summary	
play_by_play_pts	
player	
team	
Indices (0)	
Views (0)	
Triggers (0)	

Cada tabla se extrajo de DB Browser como csv, lista para el envío a Github. Aquí es donde se ha utilizado otra tecnología investigada, resolviendo los últimos problemas de carga surgidos. La tecnología es git LFS (Large File Storage), garantizando que cada archivo se haya podido enviar correctamente.

La ingestá

El procedimiento de ingestá consistió, en términos generales, en acceder a los conjuntos de datos previamente definidos en este informe mediante Apache NiFi, para posteriormente almacenarlos en HDFS. Si bien a primera vista el flujo puede parecer lineal y directo, a continuación se detallan aspectos relevantes tanto del diseño en NiFi como del almacenamiento en Hadoop.

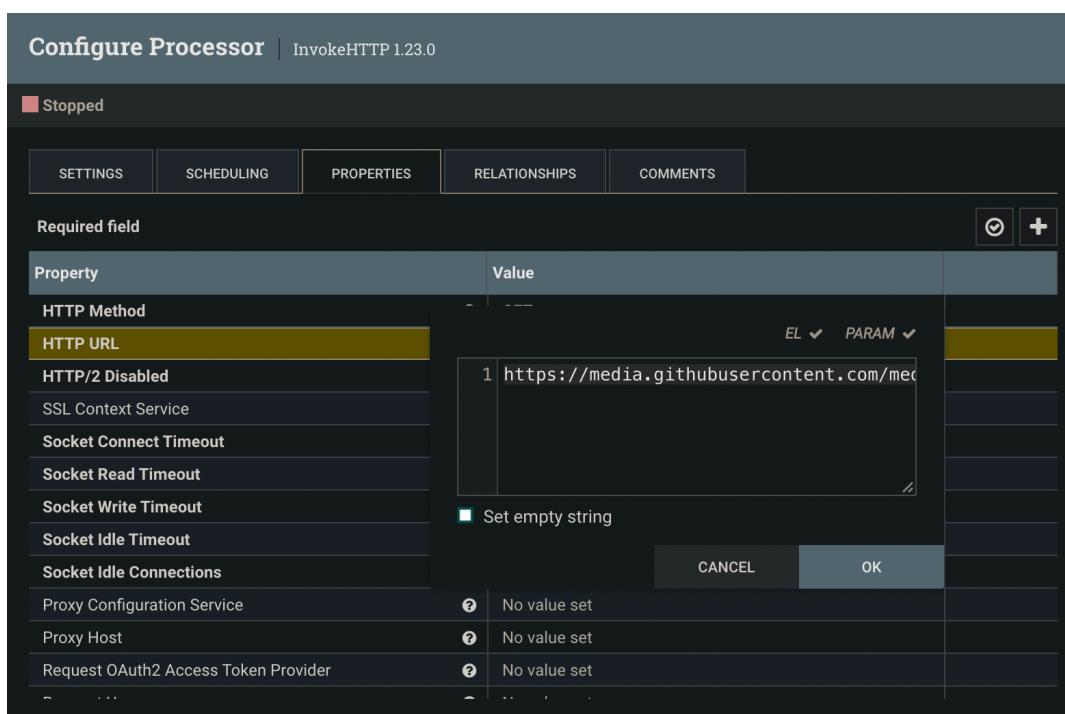
En NiFi se hizo uso de distintos processors, seleccionados estratégicamente para asegurar un flujo simplificado y alineado a buenas prácticas:

- **InvokeHTTP**

Este componente se utilizó para realizar solicitudes HTTP del tipo GET. Dado un endpoint

determinado, permite extraer el conjunto de datos requerido desde la fuente externa

Los “Path” mencionados siguen siguiente el formato; https://media.githubusercontent.com/media/GuillermoDiotti/NBA_DATA/refs/heads/main/common_player_info.csv



- **UpdateAttribute**

Su función es la de modificar o asignar atributos a los *FlowFiles*. En este caso particular, se utilizó para establecer de forma dinámica el nombre que luego tendrá el archivo en destino.

Configure Processor | UpdateAttribute 1.23.0

Stopped

SETTINGS	SCHEDULING	PROPERTIES	RELATIONSHIPS	COMMENTS
Required field				
Property	Value			
Delete Attributes Expression	No value set			
Store State	Do not store state			
Stateful Variables Initial Value	No value set			
Cache Value Lookup Cache Size	100			
filename	common_player_info.csv			

- **PutHDFS**

Finalmente, este procesor fue el encargado de persistir los datos en el sistema distribuido HDFS, asegurando su disponibilidad para posteriores etapas del proceso de refinamiento o análisis. El destino de esta transacción será desarrollado en detalle en la siguiente sección.

Configure Processor | PutHDFS 1.23.0

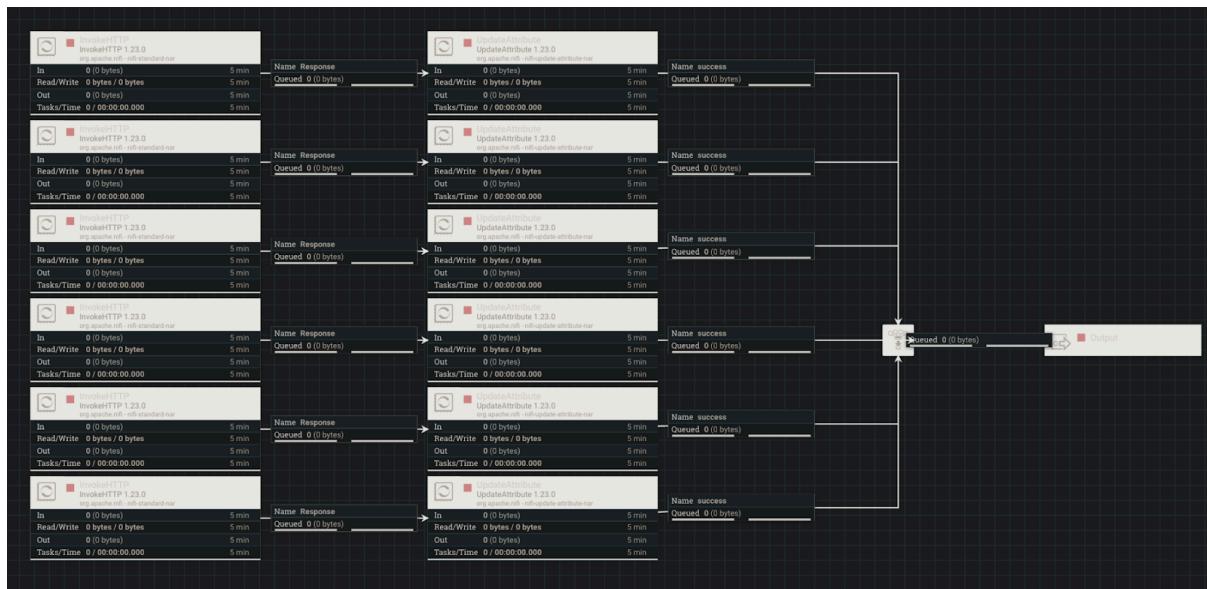
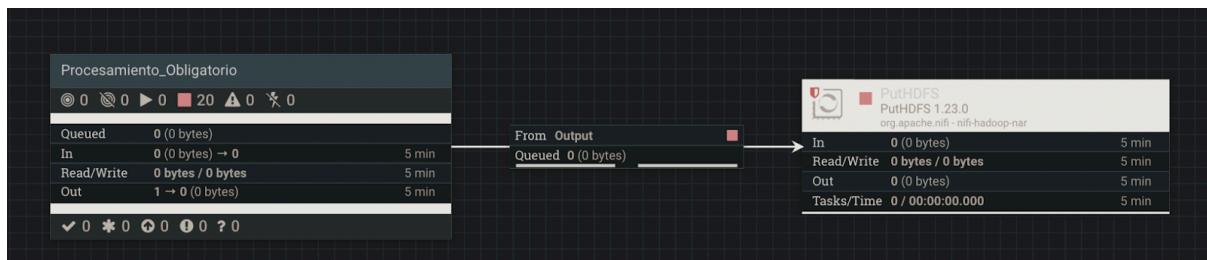
Stopped

SETTINGS	SCHEDULING	PROPERTIES	RELATIONSHIPS	COMMENTS
Required field				
Property	Value			
Kerberos Credentials Service	No value set			
Kerberos User Service	No value set			
Kerberos Principal	No value set			
Kerberos Keytab	No value set			
Kerberos Password	No value set			
Kerberos Relogin Period	4 hours			
Additional Classpath Resources				
Directory	<input type="text" value="1 /data_obligatorio_NBA/lnd/"/> <div style="display: flex; justify-content: space-between;"> EL PARAM </div> <input checked="" type="checkbox"/> Set empty string			
Conflict Resolution Strategy				
Writing Strategy				
Block Size				
IO Buffer Size				
Replication				

Tal como se ha mencionado, el flujo de trabajo incluye otros componentes que, en conjunto, promueven buenas prácticas en el uso de la herramienta. Uno de ellos es el Process Group (PG), que actúa como un contenedor capaz de agrupar múltiples componentes o flujos de manera modular y ordenada.

El Process Group implementado en este caso contiene los *processors* InvokeHTTP y UpdateAttribute, dejando fuera al PutHDFS, el cual permanece en el flujo principal. La comunicación entre el PG y el flujo principal se establece mediante entradas y salidas definidas con un componente adicional.

Para este propósito, se incorporó un Funnel, que permite combinar múltiples conexiones, en este caso, una por cada *dataframe* procesado, provenientes de InvokeHTTP y UpdateAttribute, canalizándolas todas hacia la salida (Output) del PG. Desde esta salida, ya en el flujo principal, se establece la conexión con el PutHDFS, completando así la ruta hacia el almacenamiento en HDFS.



Este flujo de trabajo permitió estructurar una ingesta automatizada, reproducible y mantenable, facilitando la integración de los datos con el ecosistema Big Data.

Ahora sí, tal como se mencionó previamente, se abordarán las acciones realizadas desde HDFS, describiendo la ruta especificada en el componente PutHDFS. En el sistema de archivos distribuido se trabajó con una estructura jerárquica basada en zonas, con el objetivo de mantener una arquitectura organizada, sostenible y escalable.

Se definió un directorio raíz denominado /data_obligatorio_NBA, ubicado dentro de /home/ort, que encapsula toda la información relacionada con el proyecto. A partir de este directorio, se estructuraron subdirectorios para representar distintas zonas del flujo de datos.

- **/data_obligatorio_NBA/Ind:** donde *Ind* corresponde a una abreviatura de **landing**. En esta zona se almacenan los datos en su forma cruda (*raw*), tal como fueron obtenidos durante el proceso de ingesta a través de Apache NiFi.
- **/data_obligatorio_NBA/rfn:** donde *rfn* corresponde a la abreviatura de **refinement**. En esta zona se almacenan los datos refinados, es decir, aquellos que han pasado por procesos de limpieza, transformación o enriquecimiento, y que están preparados para ser utilizados en análisis posteriores o integraciones con otras fuentes de datos. Esta “limpieza” es llevada a cabo con PySpark
- **/data_obligatorio_NBA/mld:** donde *mld* corresponde a la abreviatura de **modeled**. En esta zona se almacenan los datos preparados para su modelado. Los mismos posterior al refinado, fueron ajustados y adaptados para el modelo de visualización que se optará por el equipo
- **/data_obligatorio_NBA/anl:** donde *anl* corresponde a la abreviatura de **analytics**. En esta zona se almacenan los resultados de las consultas a relevar

Browse Directory										
<input style="width: 80%; border: 1px solid #ccc; padding: 2px; margin-right: 10px;" type="text" value="/data_obligatorio_NBA/"/> <input style="border: 1px solid #ccc; padding: 2px 10px;" type="button" value="Go!"/> 										
Show	25	entries								
□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name		
□	drwxr-xr-x	ort	supergroup	0 B	Jun 30 08:34	0	0 B	anl		
□	drwxr-xr-x	ort	supergroup	0 B	Jun 24 12:02	0	0 B	Ind		
□	drwxr-xr-x	ort	supergroup	0 B	Jun 30 09:20	0	0 B	mld		
□	drwxr-xr-x	ort	supergroup	0 B	Jun 28 13:38	0	0 B	rfn		

Showing 1 to 4 of 4 entries Previous 1 Next

Hadoop, 2023.

Análisis Exploratorio

El análisis exploratorio se lleva a cabo por medio de PySpark dentro de una Jupyter notebook. Esta analítica incluye limpiezas de los datos, comprensión y entendimiento de los

mismos y transformación de los anteriores, para lograr así una manipulación y manejo consistente de los datos

Esta etapa es realizada posteriormente a la ingesta de NiFi, y es la etapa antecedente a lo que sería el modelado, el cual se hablará en la siguiente sección.

A continuación se presentarán descripciones de las acciones cometidas por medio de este análisis exploratorio, comenzando primero por un entendimiento y familiarización de los datos seleccionados para una correcta comprensión

Descripción de los datos

Comenzando por una comprensión descriptiva de los datos. Los datos se distribuyen en 7 dataframes. Estos son:

- **game**
- **game_summary**
- **common_player_info**
- **play_by_play_pts**
- **player**
- **team**

Game

La tabla game tiene un registro por cada partido. Game tiene una totalidad de 7 columnas, atributos, mencionados y explicados a continuación:

Nombre	Descripción
game_id	Funcionando como un identificador de esta tabla
team_id_home	Identificador del equipo locatario que juega el partido
wl_home	Bandera la cual indica si el equipo locatario ganó el partido (W/L = Win/Loss)
pts_home	Puntaje final del equipo locatario
team_id_away	Identificador del equipo visitante que juega el partido

wl_away	Bandera la cual indica si el equipo visitante ganó el partido (W/L = Win/Loss)
pts_away	Puntaje final del equipo visitante

Game_summary

Esta tabla tiene información agregada del partido. Recordando los pasos seguidos en el preprocesamiento, esta tabla simplemente tiene:

Nombre	Descripción
game_id	Funcionando como un identificador de esta tabla
home_team_id	Identificador del equipo locatario que juega el partido
visitor_team_id	Identificador del equipo visitante que juega el partido
season	Temporada en años en la cual se llevó a cabo el partido

Common_player_info

La presente tabla tiene detalles agregados sobre los jugadores. Analizando y comprendiendo sus registros desde la fuente cruda, hemos notado que existen registros de usuarios que no cuentan con una tupla en esta tabla, por lo que es algo que se deberá manejar con cautela. Sus atributos son:

Nombre	Descripción
person_id	Identificador del jugador
first_name	Nombre del jugador
last_name	Apellido del jugador
birthdate	Año de nacimiento del jugador, en formato de timestamp
school	Escuela a la que atendió el jugador
country	País de origen del jugador

jersey	El/los dorsales del jugador. Visualizando esta columna, se pueden identificar dos escenarios. Por un lado jugadores con un único dorsal, representados aquí con un número. Por otro lado, jugadores con dos dorsales, siguiendo un formato “24-37”, dos valores seguidos por un guión.
position	Posición en la que juega el jugador Las diferentes posibles posiciones identificadas son: <ul style="list-style-type: none"> - Forward - Center - Guard - Forward-Guard - Guard-Forward - Forward-Center - Center-Forward
roosterstatus	Indicador que indica si un jugador se encuentra activo o no, indicando respectivamente “Active” o “Inactive” según sea el caso
games_played_current_season_flag	Bandera que indica si el equipo ha jugado algún partido en la temporada actual
team_id	Identificador del equipo al cual pertenece el jugador de la tupla
team_name	Nombre del equipo mencionado previamente
team_city	Ciudad del equipo mencionado
from_year	Año en el que el jugador comienza a activar su carrera en la NBA
to_year	Último año activo del jugador previo a su retiro. Nótese que todos los jugadores tienen un valor en esta columna, por más de que en la actualidad sigan marcando

Play_by_play_pts

La tabla Play_by_play_pts contiene un registro por cada evento en el que se hayan anotado puntos durante un partido.

Originalmente, estos eventos formaban parte de una tabla mucho más extensa llamada play_by_play, que registraba todos los eventos ocurridos en los encuentros (faltas, cambios, tiros, interrupciones, etc.). Debido al gran volumen de datos que eso implicaba, y al enfoque orientado al estudio de los puntajes, se decidió construir esta nueva versión refinada.

Esta tabla conserva únicamente los eventos que impactan directamente en el marcador, reduciendo significativamente su tamaño y permitiendo un análisis más ágil de los puntos anotados en los partidos.

A continuación, se describe el contenido de la tabla:

Nombre	Descripción
game_id	Identificador del partido. A su vez, parte de la clave compuesta de esta tabla
eventnum	Identificador numérico ascendente en orden de la ocurrencia de los eventos por partido. Junto a game_id, forman parte de la clave compuesta de el registro
period	Atributo numérico que indica en cual de los cuatro cuartos del partido ocurre el evento. Reglamentariamente en un partido se juegan cuatro cuartos. Sin embargo, se observan valores que trascienden hasta el octavo período. Si el partido termina en empate, se juegan periodos adicionales.
wctimestring	Hora de la ocurrencia del evento
homedescription	Contiene la descripción textual del evento cuando el equipo locatario anota puntos. Si el evento corresponde al equipo visitante, su valor será null. Las descripciones no siguen un formato estructurado ni estandarizado, lo cual dificultó considerablemente su interpretación. En su mayoría, cada registro incluye el tipo de anotación, el nombre del jugador que la realizó y, en algunos casos, el nombre del jugador que realizó la asistencia. Veamos un ejemplo: " <i>Payton Driving Layup (17 PTS)</i> " En este caso: "Payton" es el apellido del jugador que anota. "Driving Layup" indica el tipo de jugada que resultó en puntos. Los puntos resultados de esta jugada son 2. Este valor debió ser investigado. Se puede comprobar el incremento en la columna "scoremargin" "(17 PTS)" señala que Payton alcanza un total acumulado de 17 puntos en el partido al momento de ese evento.
visitordescription	Funciona de manera equivalente que la columna explicada anteriormente, homedescription
score	La columna visualiza el resultado del partido hasta el momento. Su formato de impresión es el siguiente; puntaje del equipo visitante, un guión separador, y resultado del equipo local
scoremargin	Scoremargin corresponde a la diferencia entre los valores de puntaje de ambos equipos, equipo local menos equipo visitante, obteniendo

	un valor positivo por ejemplo si el equipo locatario hasta el momento es el vitorioso
player1_id	Identificador del jugador que comete el evento
player1_name	Nombre completo del jugador que encadena el evento
player1_team_id	Equipo del jugador descrito anteriormente. Respecto a esta columna en relación con player1_id. A lo largo de los registros, se observan tuplas en donde la par id del jugador, id del equipo, difiere. Es decir una tupla estando el jugador en un equipo, y una tupla del mismo jugador estando en otro equipo distinto. Lo anterior da a atender que play_by_play_pts tiene filas de distintas temporadas.

Player

Tabla que representa a los jugadores de la NBA. Como bien se menciona respecto a “common_player_info”, no todos los jugadores representados en esta están vinculados también a una fila en esta otra tabla mencionada

Nombre	Descripción
id	Identificador del jugador
full_name	Nombre completo del jugador
first_name	Nombre de pila del jugador
last_name	Apellido del jugador
is_active	Bandera binaria que indica cuando un jugador está activo o no, reflejando 0 cuando no lo está y 1 cuando lo está

Team

Por último, team, los equipos de la NBA

Nombre	Descripción
id	Identificador del equipo
full_name	Nombre completo del equipo

city	Ciudad del equipo
state	Estado geográfico del equipo
year_founded	Año de fundación del equipo

Descripción de la exploración

El proceso de exploración de datos implica diversas operaciones imprescindibles, como la validación de claves primarias, el análisis de tipos de datos, así como la eliminación de registros duplicados. Estas tareas tienen como objetivo garantizar la calidad y consistencia de los datos antes de que se utilicen en etapas más avanzadas, como lo son el modelado.

Una duda que puede surgir, y que incluso nos ha surgido, es respecto al orden de estas etapas: ¿No sería más eficiente definir el modelo primero y refinar únicamente los subconjuntos de datos que se utilizarán? La respuesta es no. Realizar el refinamiento de forma previa aporta mayor flexibilidad: en caso de que se decida modificar el modelo en un escenario futuro, no será necesario rehacer todo el proceso de limpieza desde cero. Esta estrategia brinda un ahorro de recursos frente a estos posibles escenarios.

Enfocando concretamente acerca de las tareas tomadas:

Creación de la sesión

Previo a sumergirse en la etapa de exploración, se ejecutan ciertas líneas de código destinadas a crear la sesión e importar funciones imprescindibles.

El bloque de instrucciones importa SparkSession, punto de entrada principal para trabajar con Spark, junto con funciones y tipos de datos provistos por esta tecnología.

A continuación, se crea la instancia de dicha sesión, la cual será utilizada para la lectura, transformación y análisis de los datos.

```

from pyspark.sql import SparkSession

from pyspark.sql.functions import *
from pyspark.sql.types import *

spark = SparkSession \
    .builder \
    .appName("pyspark_Obl_NBA_rfn") \
    .getOrCreate()
  
```

Carga de los dataframes

Para la carga de los datos desde la carpeta lnd (landing), por cada dataframe, se ejecuta lo siguiente:

```
df_game = spark.read.csv('/data_obligatorio_NBA/lnd/game.csv', header=True, sep=',')
```

“Header = True” permite conservar los encabezados titulares de las columnas. Restante, “sep” hace referencia a los separadores entre los distintos conjuntos de elementos. En el caso de los csv, suele ser una coma “,”.

CARGA DE LOS CSV
<pre>[2]: df_game = spark.read.csv('/data_obligatorio_NBA/lnd/game.csv', header=True, sep=',') df_game_summary = spark.read.csv('/data_obligatorio_NBA/lnd/game_summary.csv', header=True, sep=',') df_common_player_info = spark.read.csv('/data_obligatorio_NBA/lnd/common_player_info.csv', header=True, sep=',') df_play_by_play_pts = spark.read.csv('/data_obligatorio_NBA/lnd/play_by_play_pts.csv', header=True, sep=',') df_player = spark.read.csv('/data_obligatorio_NBA/lnd/player.csv', header=True, sep=',') df_team = spark.read.csv('/data_obligatorio_NBA/lnd/team.csv', header=True, sep=',')</pre>

Visualización de las columnas

Para una lectura de las columnas, se utiliza la función “printSchema()”, precediendo de la variable que guarda el df. Esta impresión permite, línea por línea, identificar las columnas con su respectivo tipo.

Mencionando los tipos de los atributos, los datos son cargados a la Jupyter Notebook como string. Por ende, la salida seguiría la siguiente plantilla:

IMPRESION DE LOS ATRIBUTOS

```
df_game.printSchema()

root
|-- team_id_home: string (nullable = true)
|-- game_id: string (nullable = true)
|-- wl_home: string (nullable = true)
|-- pts_home: string (nullable = true)
|-- team_id_away: string (nullable = true)
|-- wl_away: string (nullable = true)
|-- pts_away: string (nullable = true)
```

Implementación de schemas y carga de los dataframes con ellos

Todos los datos son del tipo string. Sin embargo, sería más correcto que cada dato sea representado con el tipo que corresponda, dependiendo si es un dato numérico o una fecha. Aquí es donde entran los schemas. Estos artefactos, son similares a una plantilla donde se especifica para cada atributo, el tipo que se desea que tenga. Correspondiendo para un elemento de tipo string, en el esquema se pondrá StringType(), de int a IntegerType().

Las fechas son tratadas como TimestampType(), por defecto con el siguiente formato; (yyyy-mm-dd hh:mm:ss). Los identificadores, son adjudicados con IntegerType(), esto genera que los identificadores con ceros (0) por delante, sean omitidos

CREACION DE SCHEMAS, CONVIRTIENDO LOS ATRIBUTOS A TIPOS CORRESPONDIENTES

```
schema_game = StructType([
    StructField("team_id_home", IntegerType(), True),
    StructField("game_id", IntegerType(), True),
    StructField("wl_home", StringType(), True),
    StructField("pts_home", DoubleType(), True),
    StructField("team_id_away", IntegerType(), True),
    StructField("wl_away", StringType(), True),
    StructField("pts_away", DoubleType(), True)
])

# Lectura del archivo CSV con schema
df_game = spark.read.csv('/data_obligatorio_NBA/lnd/game.csv', header=True, sep=',', schema=schema_game)
```

Conversión de datos mal representados

Para evitar conflictos que podrían surgir y garantizar una correcta interpretación de los datos, se realizaron algunas conversiones de tipo.

En ciertos casos, los datos debieron ser definidos en el esquema con un tipo que no los representa de forma ideal, ya que de otro modo se imposibilitaba su lectura desde la fuente.

A continuación, se muestra un ejemplo de una de estas conversiones: los puntos de los equipos en la tabla game son cargados desde la fuente como valores de tipo double. No

obstante, desde la perspectiva del análisis, se desea que dichos atributos sean tratados como números enteros. Casos como este justifican las conversiones realizadas.

CONVERSION DE TIPOS DE DATOS MAL REPRESENTADOS EN LA FUENTE DE DATOS

```
df_game.select("pts_home", "pts_away").show(10)
df_game = df_game.withColumn("pts_home", df_game["pts_home"].cast("int"))
df_game = df_game.withColumn("pts_away", df_game["pts_away"].cast("int"))
df_game.select("pts_home", "pts_away").show(10)
```

Eliminación de duplicados

Para garantizar la consistencia de los datos y lectura adecuada, se hace uso de la función de spark dropDuplicates(). Gracias a esta función, se podrán desechar los registros duplicados

ELIMINACION DE DUPLICADOS

```
# eliminar duplicados
df_game = df_game.dropDuplicates()
df_game_summary = df_game_summary.dropDuplicates()
df_common_player_info = df_common_player_info.dropDuplicates()
df_play_by_play_pts = df_play_by_play_pts.dropDuplicates()
df_player = df_player.dropDuplicates()
df_team = df_team.dropDuplicates()
```

Validación de datos no duplicados

Al haber realizado el procedimiento de eliminado de duplicados, se someten los dataframes a una filtrado adicional creado para así garantizar que no permanezca ningún registro duplicado

VERIFICACION DE LA ELIMINACION DE LOS DUPLICADOS (NO DEBE HABER RETORNO)

```
df_game.groupBy("game_id") \
    .count() \
    .filter("count > 1") \
    .orderBy("count", ascending=False) \
    .show()
```

[Stage 14:>	(0 + 1) / 1]
+-----+-----+	
game_id count	
+-----+-----+	
+-----+-----+	

Manejo de datos nulos

Al igual que para los duplicados, Spark tiene definida la función dropna() para el caso de los registros nulos. La eliminación de registros nulos es una de las posibles decisiones. Sin embargo, frente a estos datos vacíos, se ha decidido conservarlos, para no omitir información que puede llegar a ser de relevancia en el futuro.

Para estos datos vacíos, se ha definido como estándar que, para el caso de pertenecer a una columna de tipo *string*, se les asigne la cadena "Undefined"; y, para el caso de pertenecer a una columna de tipo de dato *integer*, se les adjudique el valor -1.

Se destaca, a su vez, que se ha incluido “dentro de la misma bolsa” a los datos cuyo valor sea "" (cadena vacía).

ELIMINACION DE DATOS NULOS

```
for field in df_game.schema.fields:
    name = field.name
    dtype = field.dataType

    if isinstance(dtype, StringType):
        default_value = lit("Undefined")
        condition = col(name).isNull() | (trim(col(name)) == "")
    elif isinstance(dtype, IntegerType):
        default_value = lit(-1)
        condition = col(name).isNull()

    df_game = df_game.withColumn(
        name,
        when(col(name).isNull() |
            ((col(name).cast("string").isNotNull() & (trim(col(name)) == ""))), default_value)
        .otherwise(col(name))
    )
```

Verificación de la eliminación de nulos

Queda pendiente verificar esta gestión sobre registros nulos. La función implementada a continuación logra cumplir ese objetivo

VERIFICACION DE LA ELIMINACION DE NULOS

```
null_or_empty_counts = df_game.select([
    sum(
        when(
            col(c).isNull() |
            ((col(c).cast("string").isNotNull() & (trim(col(c)) == ""))), 1
        ).otherwise(0)
    ).alias(c)
    for c in df_game.columns
])

null_or_empty_counts.show()
```

team_id_home	game_id	wl_home	pts_home	team_id_away	wl_away	pts_away
0	0	0	0	0	0	0

Validación de la unicidad de la clave

Las denominadas claves son los campos identificadores de cada conjunto de datos. Por ende, los mismos deben ser únicos entre registros de un mismo conjunto. Es aquí donde viene la validación de la unicidad de la clave. Dada una función auxiliar de apoyo implementada, evalúa si por cada dataframe, la cantidad de registros es igual a la cantidad de registros distintos (DISTINCT). En caso contrario, se imprime un mensaje mencionando que no se cumple

```
def unique_key(df, col):
    tot_rows = df.count()
    primary_row = df.select(col).distinct().count()

    if tot_rows == primary_row:
        print(f"UNIQUE KEY")
    else:
        print(f"NOT UNIQUE KEY")

unique_key(df_game, "game_id")
```

UNIQUE KEY

Visualización de los datos finales

Una vez completadas todas las operaciones descriptas del refinamiento, se procede a imprimir las primeras filas de cada colección, visualizando así la estructura de los mismos, como resultado final del refinamiento

IMPRESIÓN DE LAS COLUMNAS DE LOS DF FINALES CON SUS TIPOS

```
df_game.printSchema()
root
 |-- team_id_home: integer (nullable = true)
 |-- game_id: integer (nullable = true)
 |-- wl_home: string (nullable = true)
 |-- pts_home: integer (nullable = true)
 |-- team_id_away: integer (nullable = true)
 |-- wl_away: string (nullable = true)
 |-- pts_away: integer (nullable = true)
```

VISUALIZACION DE LOS PRIMEROS DATOS DE LOS DF FINALES

```
df_game.show(10)
```

team_id_home	game_id	wl_home	pts_home	team_id_away	wl_away	pts_away
1610610026	24600156	L	74	1610610032	W	78
1610612738	24600187	L	57	1610610036	W	80
1610612747	24800213	W	89	1610612744	L	71
1610612765	24800254	W	89	1610612744	L	87
1610610029	24800326	L	80	1610610034	W	81
1610610025	44800212	L	85	1610612747	W	101
1610610036	25000004	W	100	1610610030	L	84
1610612755	25000213	L	86	1610612744	W	96
1610612738	25000260	W	85	1610612737	L	70
1610610024	25000270	L	72	1610612755	W	80

only showing top 10 rows

Carga de los dataframes finales

Ahora es cuando se logran cargar los resultados a HDFS, a una ubicación que sigue bajo la misma ruta raiz creada, pero difiere de directorio, siguiendo la definición de las zonas previamente definidas. Este nuevo directorio será rfn.

Los dataframes son almacenados bajo a una arquitectura PARQUET. PARQUET es un formato de almacenamiento de datos con una estructura de columna. Es decir, almacena cada columna por separado. Esto brinda varios beneficios para el manejo de varios volúmenes de datos, optimizando las lecturas al acceder únicamente a las columnas deseadas.

GUARDADO DE LOS DF EN FORMATO PARQUET

```

df_game.write \
  .mode("overwrite") \
  .parquet("/data_obligatorio_NBA/rfn/game_pqt")

df_game_summary.write \
  .mode("overwrite") \
  .parquet("/data_obligatorio_NBA/rfn/game_sum_pqt")

df_common_player_info.write \
  .mode("overwrite") \
  .parquet("/data_obligatorio_NBA/rfn/com_play_info_pqt")

df_play_by_play_pts.write \
  .mode("overwrite") \
  .parquet("/data_obligatorio_NBA/rfn/play_pts_pqt")

df_player.write \
  .mode("overwrite") \
  .parquet("/data_obligatorio_NBA/rfn/player_pqt")

df_team.write \
  .mode("overwrite") \
  .parquet("/data_obligatorio_NBA/rfn/team_pqt")
  
```

Validación de el correcto guardado de los datos

Al igual que con operaciones anteriores, esta carga viene acompañada de una validación posterior, comprobando así el éxito de la misma.

LECTURA DE LOS DF EN FORMATO PARQUET PARA COMPROBAR SU GUARDADO

```

df_parquet = spark.read.parquet("/data_obligatorio_NBA/rfn/game_pqt")
df_parquet.show(10)

+-----+-----+-----+-----+-----+-----+
|team_id_home| game_id|wl_home|pts_home|team_id_away|wl_away|pts_away|
+-----+-----+-----+-----+-----+-----+
| 1610610026|24600156|     L|      74| 1610610032|     W|      78|
| 1610612738|24600187|     L|      57| 1610610036|     W|      80|
| 1610612747|24800213|     W|      89| 1610612744|     L|      71|
| 1610612765|24800254|     W|      89| 1610612744|     L|      87|
| 1610610029|24800326|     L|      80| 1610610034|     W|      81|
| 1610610025|44800212|     L|      85| 1610612747|     W|      101|
| 1610610036|25000004|     W|     100| 1610610030|     L|      84|
| 1610612755|25000213|     L|      86| 1610612744|     W|      96|
| 1610612738|25000260|     W|      85| 1610612737|     L|      70|
| 1610610024|25000270|     L|      72| 1610612755|     W|      80|
+-----+-----+-----+-----+-----+-----+
only showing top 10 rows
  
```

Browse Directory

/data Obligatorio_NBA/rfn									Go!					
Show	25	entries								Search:				
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name						
drwxr-xr-x	ort	supergroup	supergroup	0 B	Jun 28 13:37	0	0 B	com_play_info_pqt						
drwxr-xr-x	ort	supergroup	supergroup	0 B	Jun 28 13:37	0	0 B	game_pqt						
drwxr-xr-x	ort	supergroup	supergroup	0 B	Jun 28 13:37	0	0 B	game_sum_pqt						
drwxr-xr-x	ort	supergroup	supergroup	0 B	Jun 28 13:38	0	0 B	play_pts_pqt						
drwxr-xr-x	ort	supergroup	supergroup	0 B	Jun 28 13:38	0	0 B	player_pqt						
drwxr-xr-x	ort	supergroup	supergroup	0 B	Jun 28 13:38	0	0 B	team_pqt						

Showing 1 to 6 of 6 entries

Previous 1 Next

Modelado de datos

Una de las grandes decisiones que se tuvo que tomar fue la de cómo modelar los datos. Existen extensas formas de como modelarlos, desde data vaults, modelos estrella, normalizada, entre otros. Cada diseño de modelado aporta sus características y beneficios, como también por contraparte posibles complicaciones.

El modelado implementado es el normalizado. El modelado normalizado organiza los datos en múltiples tablas relacionadas mediante claves, con el objetivo de reducir la redundancia y asegurar la integridad de los datos. Cada tabla representa una entidad específica (como jugadores, equipos o partidos), y las relaciones entre ellas se establecen mediante identificadores únicos. Entre sus principales beneficios se encuentran el menor uso de almacenamiento, facilidad para mantener y actualizar datos sin inconsistencias, y una estructura clara y lógica. A su vez, esta representación de los datos es la que implicaba menos transformaciones de realizar. Actualmente ya se contaba con tablas separadas e independientes, por más de que es cierto que hay ciertas operaciones a realizar con antelación. Sin embargo, su principal desventaja es el mayor costo al realizar consultas complejas, pudiendo requerir múltiples *joins*, lo que puede afectar el rendimiento frente a modelos como el estrella.

Descripción de la exploración

Tal como se menciona, para adecuar los dataframes al modelo normalizado, un procesamiento previo debe ser realizado, incluyendo renombre de variables, creación de columnas, eliminación de algunas de ellas inutilizadas, y más.

Creación de la sesión

Igual que en el refinement. Se crea la sesión de spark y se traen las dependencias imprescindibles y necesarias

Carga de los dataframes

Siguiendo el flujo, se cargan los df, sin embargo, esta ve en formato PARQUET, recordando que así fueron almacenados al culminar el refinamiento

Manipulación de los datos

Manipulación de los datos. Esta fase es extensa, que contempla operaciones importantes y sofisticadas

Para ilustrar las modificaciones codificadas, se separarán según su tabla.

common_player_info

Esta tabla almacena, entre otros datos, los números de camiseta de los jugadores. Tal como se detalla en secciones anteriores, la columna correspondiente puede contener uno o dos valores, separados por un guion, en caso de que un jugador haya utilizado más de una camiseta registrada. Se aplica una transformación sobre esta columna para extraer los números: el primero se almacena en la columna “jersey1” y, si existe un segundo número, este se guarda en “jersey2”. Para los jugadores que no tienen ningún número asignado o presentan solo uno, se asigna el valor “Undefined” tanto en “jersey2” como en ambas columnas, según corresponda.

```
split_col = split(col("jersey"), "-")

df_com_play_info_pqt = df_com_play_info_pqt.withColumn(
    "jersey1",
    when(split_col.getItem(0).isNotNull(), split_col.getItem(0).cast("string"))
)

df_com_play_info_pqt = df_com_play_info_pqt.withColumn(
    "jersey2",
    when(split_col.getItem(1).isNotNull(), split_col.getItem(1).cast("string"))
    .otherwise("Undefined")
)

|df_com_play_info_pqt.select("person_id", "jersey", "jersey1", "jersey2").show(80)
```

game

La tabla game no tiene la temporada del partido, pero esta sí está en la tabla game_summary. La temporada es considerada como el único campo de interés en esta segunda tabla, teniendo luego información ya contemplada en otros lados. Se hace un join entre estas dos tablas para guardar la season en game, con -1 cuando no haya una establecida

```

df_game_pqt = df_game_pqt.join(
    df_game_sum_pqt.select("game_id", "season"),
    on="game_id",
    how="left"
)

df_game_pqt = df_game_pqt.fillna({"season": -1})
  
```

play_pts

La tabla anterior con registros de cada jugada que aumente el puntaje de un partido, contiene homedescription y visitordescription, correspondiendo respectivamente a eventos del equipo local como visitante. Este almacenamiento se puede optimizar, conservado todo en una única columna y creando una nueva que indique a qué equipo corresponde el evento, recordemos que cuando una de estas descripciones tiene información, la restante está vacía.

```

df_play_pts_pqt = df_play_pts_pqt.withColumn(
    "description",
    when(col("homedescription") != "Undefined", col("homedescription"))
    .when(col("visitordescription") != "Undefined", col("visitordescription"))
)

df_play_pts_pqt = df_play_pts_pqt.withColumn("team_type",
    when(col("homedescription") != "Undefined", "home")
    .when(col("visitordescription") != "Undefined", "visitor")
)
  
```

Luego, “play_pts” será la columna que conserve cuantos puntos surgieron por el evento, ya sea por un tiro doble, triple, o de un punto

```
df_play_pts_pqt = df_play_pts_pqt.withColumn("description_lower", lower(col("description")))

df_play_pts_pqt = df_play_pts_pqt.withColumn("play_pts",
    when(col("description_lower").contains("3pt"), 3)
    .when(col("description_lower").rlike("1 of 2"), 1)
    .when(col("description_lower").rlike("2 of 2"), 1)
    .when(col("description_lower").rlike("free throw"), 1)
    .when(col("description_lower").rlike("dunk"), 2)
    .when(col("description_lower").rlike("layup"), 2)
    .when(col("description_lower").rlike("fadeaway"), 2)
    .when(col("description_lower").rlike("hook"), 2)
    .when(col("description_lower").rlike("jump(\s+\w+)*\s+shot"), 2)
    .when(col("description_lower").rlike("finger roll"), 2)
    .when(col("description_lower").rlike("tip shot"), 2)
    .when(col("description_lower").rlike("bank"), 2)
    .when(col("description_lower").rlike("no shot"), 2)
    .when(col("description_lower").rlike("shot"), 2)
    .otherwise(1)
)
```

Finalmente, una reescritura forzada de datos mal representados. “game” presenta los puntajes finales de cada partido para los dos equipos que lo jugaron. En la tabla que se está trabajando, play_by_play, están desglosados los puntajes por cada evento. Una conclusión que se puede sacar del relacionamiento entre estas, es que el puntaje del evento final de un partido (ordenado por orden de incidencia), deberá ser igual al puntaje final del partido. Esto se cumplía en su mayoría, exceptuando ciertos valores, que serán ajustados

```

df_game_pqt = df_game_pqt.withColumn("pts_away",
    when(col("game_id") == 11300032, 89)
    .when(col("game_id") == 11300033, 80)
    .when(col("game_id") == 11300034, 98)
    .when(col("game_id") == 12000009, 107)
    .when(col("game_id") == 20300778, 101)
    .when(col("game_id") == 29600070, 86)
    .when(col("game_id") == 29600332, 81)
    .when(col("game_id") == 29600370, 76)
    .otherwise(col("pts_away")))
)

df_game_pqt = df_game_pqt.withColumn("pts_home",
    when(col("game_id") == 11300032, 90)
    .when(col("game_id") == 11300033, 86)
    .when(col("game_id") == 11300034, 96)
    .when(col("game_id") == 12000009, 102)
    .when(col("game_id") == 20300778, 120)
    .when(col("game_id") == 29600070, 101)
    .when(col("game_id") == 29600332, 123)
    .when(col("game_id") == 29600370, 91)
    .otherwise(col("pts_home")))
)
  
```

player

Para el caso de la tabla de jugadores, el original `is_active`, de tipo binario que indica cuando un jugador está activo o no, se transforma en un campo de tipo string especificando para cada caso “Active” o “Inactive”, y en una nueva columna llamada “status”.

```

df_player_pqt = df_player_pqt.withColumn("status",
    when(col("is_active") == 1, "Active")
    .when(col("is_active") == 0, "Inactive")
)
  
```

Eliminación y renombre de columnas

Para mantener una eficiencia y efectividad de almacenamiento, se desechan los datos inutilizados, y renombran los conservados para mayor entendimiento.

ELIMINACION Y RENOMBRE DE COLUMNAS

```
df_com_play_info_pqt = df_com_play_info_pqt.withColumnRenamed("id", "player_id")
df_com_play_info_pqt = df_com_play_info_pqt.drop("first_name")
df_com_play_info_pqt = df_com_play_info_pqt.drop("last_name")
df_com_play_info_pqt = df_com_play_info_pqt.drop("birthdate")
df_com_play_info_pqt = df_com_play_info_pqt.drop("country")
df_com_play_info_pqt = df_com_play_info_pqt.drop("rosterstatus")
df_com_play_info_pqt = df_com_play_info_pqt.drop("games_played_current_season_flag")
df_com_play_info_pqt = df_com_play_info_pqt.drop("team_name")
df_com_play_info_pqt = df_com_play_info_pqt.drop("team_city")
df_com_play_info_pqt = df_com_play_info_pqt.drop("jersey")
df_com_play_info_pqt = df_com_play_info_pqt.drop("from_year")
df_com_play_info_pqt = df_com_play_info_pqt.drop("to_year")
```

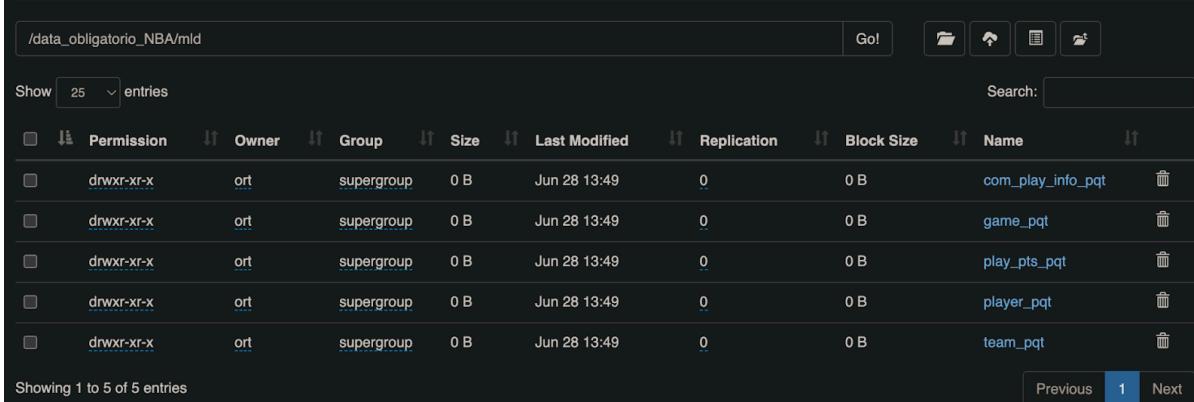
Carga del modelamiento final

Conservando una carga con formato PARQUET, se almacenan los datos modelados en la zona “mld”

```
df_player_pqt.write \
    .mode("overwrite") \
    .parquet("/data_obligatorio_NBA/mld/player_pqt")

df_team_pqt.write \
    .mode("overwrite") \
    .parquet("/data_obligatorio_NBA/mld/team_pqt")
```

Browse Directory



	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	ort	supergroup	0 B	Jun 28 13:49	0	0 B	com_play_info_pqt
□	drwxr-xr-x	ort	supergroup	0 B	Jun 28 13:49	0	0 B	game_pqt
□	drwxr-xr-x	ort	supergroup	0 B	Jun 28 13:49	0	0 B	play_pts_pqt
□	drwxr-xr-x	ort	supergroup	0 B	Jun 28 13:49	0	0 B	player_pqt
□	drwxr-xr-x	ort	supergroup	0 B	Jun 28 13:49	0	0 B	team_pqt

Showing 1 to 5 of 5 entries

Resultado final del modelado

El modelamiento comienza con 6 dataframes distintos. Atravesando las operaciones mencionadas, se termina este procedimiento con 5 de ellos. A continuación se describirá la estructura de cada uno

game

Nombre	Descripción
game_id	Funcionando como un identificador de esta tabla
team_id_home	Identificador del equipo locatario que juega el partido
pts_home	Puntaje final del equipo locatario
team_id_away	Identificador del equipo visitante que juega el partido
pts_away	Puntaje final del equipo visitante
season	Temporada en la que ocurre el juego, -1 cuando no hay una temporada registrada

common_player_info

Nombre	Descripción
id	Identificador del jugador
full_name	Nombre completo del jugador
first_name	Nombre de pila del jugador
last_name	Apellido del jugador
is_active	Bandera binaria que indica cuando un jugador está activo o no, reflejando 0 cuando no lo está y 1 cuando lo está

play_pts

Nombre	Descripción
game_id	Identificador del partido. A su vez, parte de la clave compuesta de esta tabla
eventnum	Identificador numérico ascendente en orden de la ocurrencia de los eventos por partido. Junto a game_id, forman parte de la clave compuesta de el registro
score_visitor	La columna visualiza el resultado total del puntaje del equipo visitante tras la ocurrencia del evento

team_type	Esta columna refleja si el evento que suma puntos fue realizado por el equipo local o visitante. De tal manera, refleja “home”, o “visitor”
play_pts	La columna visualiza el incremento de puntos del evento en particular
score_home	La columna visualiza el resultado total del puntaje del equipo local tras la ocurrencia del evento
player_id	Identificador del jugador que comete el evento
team_id	Equipo del jugador descrito anteriormente.

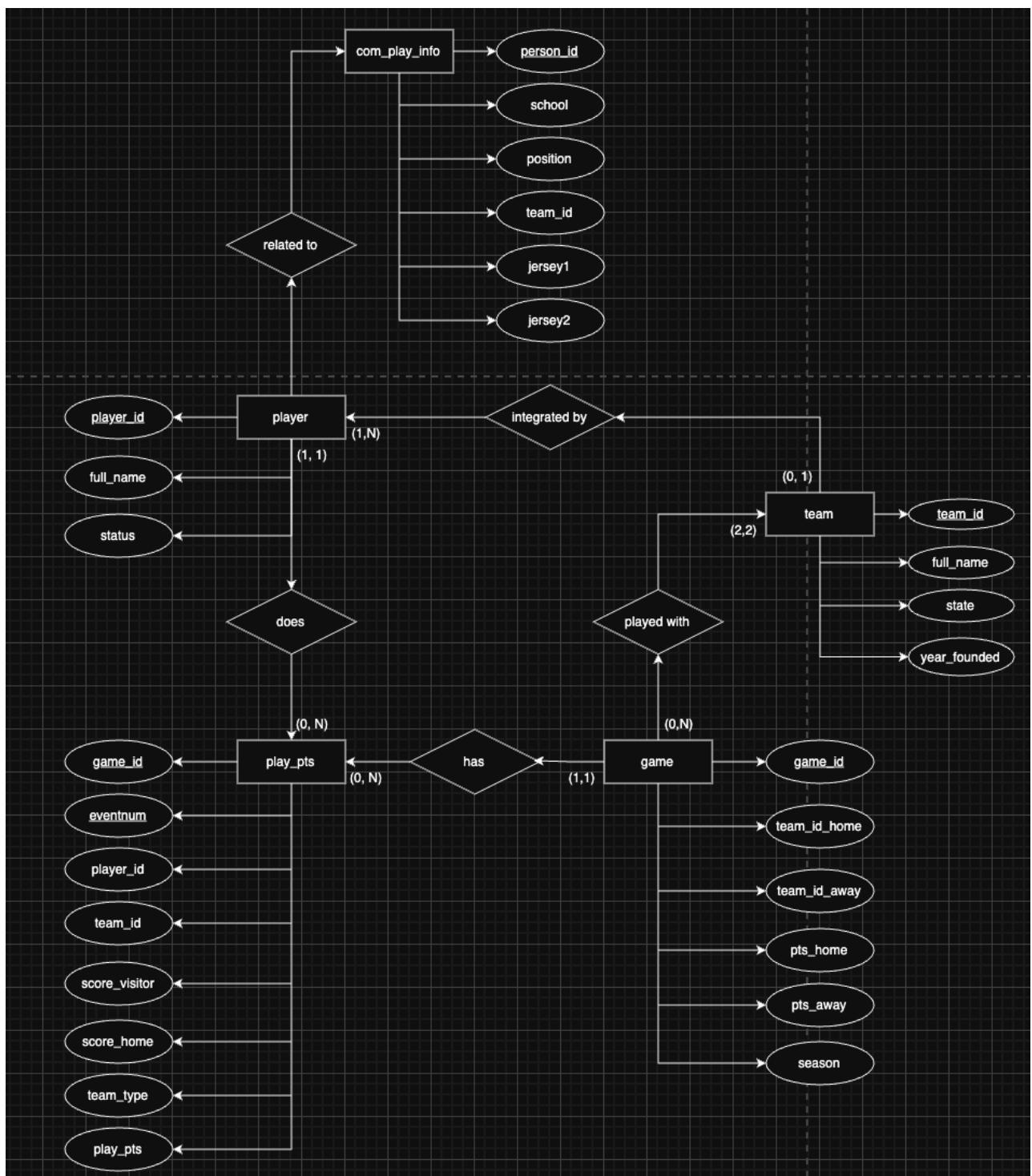
player

Nombre	Descripción
player_id	Identificador del jugador
full_name	Nombre completo del jugador
status	Cadena de texto que indica cuando un jugador está activo o no, reflejando “Inactive” cuando no lo está y “Active” cuando lo está

team

Nombre	Descripción
team_id	Identificador del equipo
full_name	Nombre completo del equipo
state	Estado geográfico del equipo
year_founded	Año de fundación del equipo

Esquema del modelo



En el esquema anterior se puede ver gráficamente el modelado. Se representa cada tabla junto a los atributos que presentan (los atributos subrayados son los que representan las claves únicas)

Hive

Hive es una herramienta que permite administrar y consultar grandes volúmenes de datos. Es la última macro tecnología implementada en el proyecto.

Descripción del procesamiento

Más abajo se hablará de todas las etapas encaminadas con la tecnología

Creación de la sesión

Como primer paso, al igual que en etapas anteriores, se procede con la creación de la sesión de Spark. Sin embargo, en esta instancia se incorpora la función enableHiveSupport(), lo que habilita el soporte de Hive dentro del entorno de Spark. Esto permite ejecutar instrucciones SQL sobre tablas Hive directamente desde la notebook, como si se estuviera trabajando dentro del propio entorno de Hive. Para realizar estas consultas, simplemente se utiliza el método spark.sql().

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import *
from pyspark.sql.types import *

spark = SparkSession \
    .builder \
    .appName("pyspark_Obl_NBA_hive") \
    .enableHiveSupport() \
    .getOrCreate()
```

Instalación de herramientas

La notebook que trabajará hive y responderá las consultas, reflejará también las visualizaciones de las consultas. Por ende, ciertas herramientas deben ser instaladas para esas visualizaciones posteriores. Este es el caso de la librería matplotlib por ejemplo. En sus inicios no se hablará en profundidad de la librería, sino que se dejará eso para la sección de visualización.

INSTALACION DE HERRAMIENTAS

```
!pip3 install pandas
import pandas as pd
!pip3 install matplotlib --user
import matplotlib.pyplot as plt
import builtins
```

Creación de la base de datos

Ahora sí, a sumergirse con el uso de Hive. Se creará una base de datos “obligatorioNBA”, será la base de datos destinada al proyecto.

Para asegurar su creación, se ejecuta el comando SHOW DATABASES, pudiendo ver todas las bases de datos creadas

```
spark.sql("DROP DATABASE IF EXISTS obligatorioNBA CASCADE")
spark.sql("CREATE DATABASE IF NOT EXISTS obligatorioNBA;")
```

```
2025-07-05T16:28:39,057 INFO [Thread-4] org.apache.hadoop.hive.ql.QLDriver
```

```
spark.sql("SHOW DATABASES").show()
```

namespace
big_data
default
obligatorionba

Creación de las tablas hive

Se creará una tabla Hive por cada uno de los cinco DataFrames. Para ello, es necesario definir qué campos almacenará cada tabla y los tipos de datos correspondientes. Esta estructura se especifica mediante lo que se conoce como un *script* de creación.

Las tablas serán creadas utilizando el formato PARQUET. El campo LOCATION dentro del script de creación indica la ruta en la que previamente fueron almacenados los DataFrames modelados. Al especificar esta ubicación, Hive puede vincular automáticamente los datos existentes con la estructura de la tabla, siempre y cuando la definición de los campos en el script coincida exactamente con la estructura de los DataFrames almacenados en HDFS.

Para optimizar aún más las consultas y el rendimiento de las tablas Hive, se podría haber utilizado la técnica de particionado. Las particiones permiten organizar los datos en subdirectorios según un criterio de partición (por ejemplo, por temporada o equipo), lo que resulta especialmente beneficioso cuando se trabaja con grandes volúmenes de información.

Esta técnica mejora la eficiencia de las consultas al permitir una lectura selectiva, sin necesidad de escanear todos los registros.

A pesar de ello, en este caso, no se aplicaron particiones. La razón principal es que las consultas que se llevaron a cabo no siguen un patrón de filtrado consistente por columna o campo específico, lo que haría que el uso de particiones no aportara beneficios significativos.

Script de creación de la tabla hive player

```
spark.sql("DROP TABLE IF EXISTS player")

spark.sql("USE obligatorioNBA")

spark.sql("""
CREATE EXTERNAL TABLE player
(
player_id INT,
full_name STRING,
status STRING
)
STORED AS PARQUET
LOCATION '/data_obligatorio_NBA/mld/player_pqt'
""")
```

Script de creación de la tabla hive team

```
spark.sql("DROP TABLE IF EXISTS team")

spark.sql("USE obligatorioNBA")

spark.sql("""
CREATE EXTERNAL TABLE team
(
team_id INT,
full_name STRING,
state STRING,
year_founded INT
)
STORED AS PARQUET
LOCATION '/data_obligatorio_NBA/mld/team_pqt'
""")
```

Script de creación de la tabla hive game

```
spark.sql("DROP TABLE IF EXISTS game")

spark.sql("USE obligatorioNBA")

spark.sql("""
CREATE EXTERNAL TABLE game
(
game_id INT,
team_id_home INT,
pts_home INT,
team_id_away INT,
pts_away INT,
season INT
)
STORED AS PARQUET
LOCATION '/data_obligatorio_NBA/mld/game_pqt'
""")
```

Script de creación de la tabla hive play_pts

```
spark.sql("DROP TABLE IF EXISTS play_pts")

spark.sql("USE obligatorioNBA")

spark.sql("""
CREATE EXTERNAL TABLE play_pts
(
game_id INT,
eventnum INT,
player_id INT,
team_id INT,
score_visitor INT,
score_home INT,
team_type STRING,
play_pts INT
)
STORED AS PARQUET
LOCATION '/data_obligatorio_NBA/mld/play_pts_pqt'
""")
```

Script de creación de la tabla hive player_info

SCRIPTS DE CREACION DE LAS TABLAS

```
spark.sql("DROP TABLE IF EXISTS player_info")

spark.sql("USE obligatorioNBA")

spark.sql("""
CREATE EXTERNAL TABLE player_info
(
person_id INT,
school STRING,
position STRING,
team_id INT,
jersey1 STRING,
jersey2 STRING
)
STORED AS PARQUET
LOCATION '/data_obligatorio_NBA/mld/com_play_info_pqt'
""")
```

Información de las tablas

Con DESCRIBE FORMATTED <nombre de la tabla>, se puede obtener información detallada de la tabla creada. En esta vista se incluyen los atributos y sus tipos de datos, la base de datos a la que pertenece, el nombre de la tabla, la fecha de creación y el tamaño ocupado en bytes, entre otros metadatos.

col_name	data_type	comment
game_id	int	null
team_id_home	int	null
pts_home	int	null
team_id_away	int	null
pts_away	int	null
season	int	null
# Detailed Table Information		
Catalog	spark_catalog	
Database	obligatorionba	
Table	game	
Owner	ort	
Created Time	Sat Jul 05 16:28:45 UTC 2025	
Last Access	UNKNOWN	
Created By	Spark 3.4.1	
Type	EXTERNAL	
Provider	hive	
Table Properties	[transient_lastDdlTime=1751732925]	
Statistics	535143 bytes	
Location	hdfs://localhost:9000/data_obligatorio_NBA/mld/game_pqt	
Serde Library	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe	
InputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat	
OutputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat	
Storage Properties	[serialization.format=1]	
Partition Provider	Catalog	

Relevamiento de preguntas

Consulta 1

CONSULTA 1

¿Cuál es el top 10, de promedio de puntos de cada equipo en la temporada 2018?

```
query1 = spark.sql("""
SELECT EQUIPO, ROUND(AVG(PUNTOS), 4) AS PROMEDIO
FROM (
    SELECT t.full_name AS EQUIPO, g pts_home AS PUNTOS
    FROM game g
    JOIN team t ON g.team_id_home = t.team_id
    WHERE g.season = 2018

    UNION ALL

    SELECT t.full_name AS EQUIPO, g pts_away AS PUNTOS
    FROM game g
    JOIN team t ON g.team_id_away = t.team_id
    WHERE g.season = 2018
)
GROUP BY EQUIPO
ORDER BY PROMEDIO DESC
LIMIT 10;
""")
```

query1.show()

EQUIPO	PROMEDIO
Milwaukee Bucks	117.1236
Golden State Warriors	116.6129
Sacramento Kings	115.3714
New Orleans Pelicans	114.9296
Los Angeles Clippers	114.725
Washington Wizards	114.3733
Philadelphia 76ers	114.2921
Portland Trail Blazers	114.0562
Houston Rockets	113.2778
Oklahoma City Thunder	113.2468

Consulta 2

CONSULTA 2

¿Cuántos jugadores, por cada posición, juegan en equipos fundados hace más de 60 años?

```
query2 = spark.sql("""
SELECT COUNT(p.*) AS CANTIDAD, p.position AS POSICION
FROM player_info p
JOIN team t ON p.team_id = t.team_id
WHERE year(current_date()) - t.year_founded >= 60
GROUP BY POSICION
""")

query2.show()
```

CANTIDAD	POSICION
32	Guard-Forward
516	Guard
1	Undefined
19	Center-Forward
491	Forward
9	Forward-Guard
185	Center
38	Forward-Center

Consulta 3

CONSULTA 3

¿Cuál es el top 3 de números de camiseta con más triples anotados por jugadores que hayan estudiado en UCLA?

```
query3 = spark.sql("""
SELECT pl.full_name AS NOMBRE, pi.jersey1 AS CAMISETA, COUNT(p.game_id) AS TRIPLES
FROM play_pts p
JOIN player_info pi ON p.player_id = pi.person_id
JOIN player pl ON p.player_id = pl.player_id
WHERE p.play_pts = 3
AND pi.school = "UCLA"
GROUP BY pi.jersey1, pl.full_name
ORDER BY triples DESC
LIMIT 3
""")

query3.show()
```

NOMBRE	CAMISETA	TRIPLES
Reggie Miller	31	1318
Russell Westbrook	0	1202
Matt Barnes	22	877

Consulta 4

CONSULTA 4

¿Cuál es la cantidad de veces que cada equipo de Nueva York haya ganado como local?

```
query4 = spark.sql("""
SELECT COUNT(g.team_id_home) AS CANTIDAD, t.full_name AS EQUIPO
FROM game g
JOIN team t ON g.team_id_home = t.team_id
WHERE g pts_home > g pts_away
AND t.state = "New York"
GROUP BY t.full_name
""")
```

```
query4.show()
```

CANTIDAD	EQUIPO
1023	Brooklyn Nets
1763	New York Knicks

Consulta 5

CONSULTA 5

¿Cuál es el promedio de puntos anotados por los jugadores actualmente inactivos por temporadas? Top 10

```
query5 = spark.sql("""
SELECT ROUND(AVG(pp.play_pts), 4) AS PROMEDIO, g.season AS TEMPORADA
FROM play_pts pp
JOIN game g ON g.game_id = pp.game_id
JOIN player p ON p.player_id = pp.player_id
WHERE p.status = "Inactive"
GROUP BY g.season
ORDER BY PROMEDIO DESC
LIMIT 10
""")
```

```
query5.show()
```

PROMEDIO	TEMPORADA
1.9616	2021
1.9505	2020
1.9274	2019
1.912	2018
1.912	2017
1.8772	2016
1.849	2015
1.8363	2014
1.8291	2013
1.8141	2011

Visualización en Pandas

Para las Consultas 1, 2 y 3, se realizan dos visualizaciones por cada una. La librería de Pandas utilizada es matplotlib, debiendo al inicio importarla con import matplotlib.pyplot as plt. Matplotlib es una de las bibliotecas más utilizadas en Python para la generación de gráficos y visualizaciones de datos. Permite crear una amplia variedad de visualizaciones, desde gráficos de líneas y barras hasta diagramas de dispersión, histogramas, tortas, mapas de calor y más.

Visualizaciones de la consulta 1

```
df = query1.toPandas()

# VISUALIZACION 1
plt.figure(figsize=(8,6))
plt.plot(df['EQUIPO'], df['PROMEDIO'], marker='o')
plt.xlabel('EQUIPO')
plt.ylabel('PROMEDIO')
plt.title('Top 10, de promedio de puntos de cada equipo en la temporada 2018')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

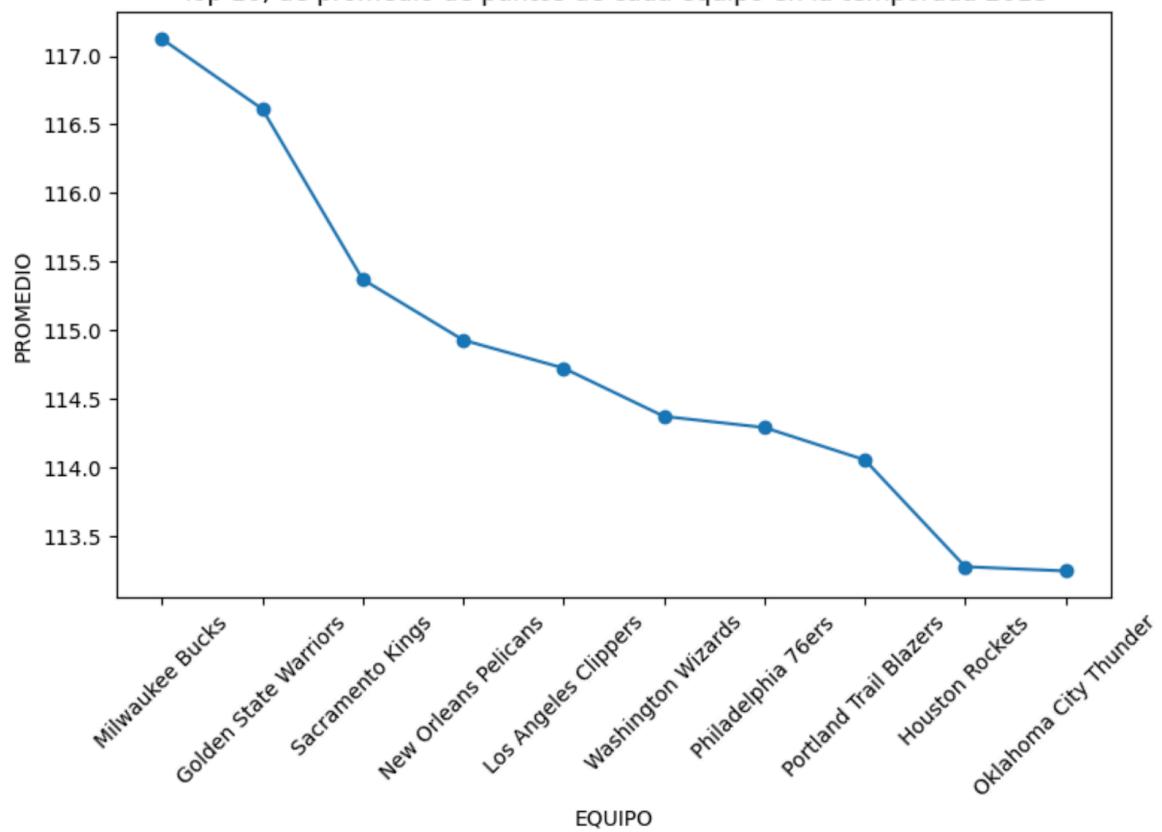
# VISUALIZACION 2
plt.figure(figsize=(10, 5))
plt.bar(df['EQUIPO'], df['PROMEDIO'], color='green')
plt.xlabel('Equipo')
plt.ylabel('Promedio de puntos')
plt.title('Top 10, de promedio de puntos de cada equipo en la temporada 2018')

min_prom = df['PROMEDIO'].min()
plt.ylim(min_prom * 0.95, df['PROMEDIO'].max() * 1.05)

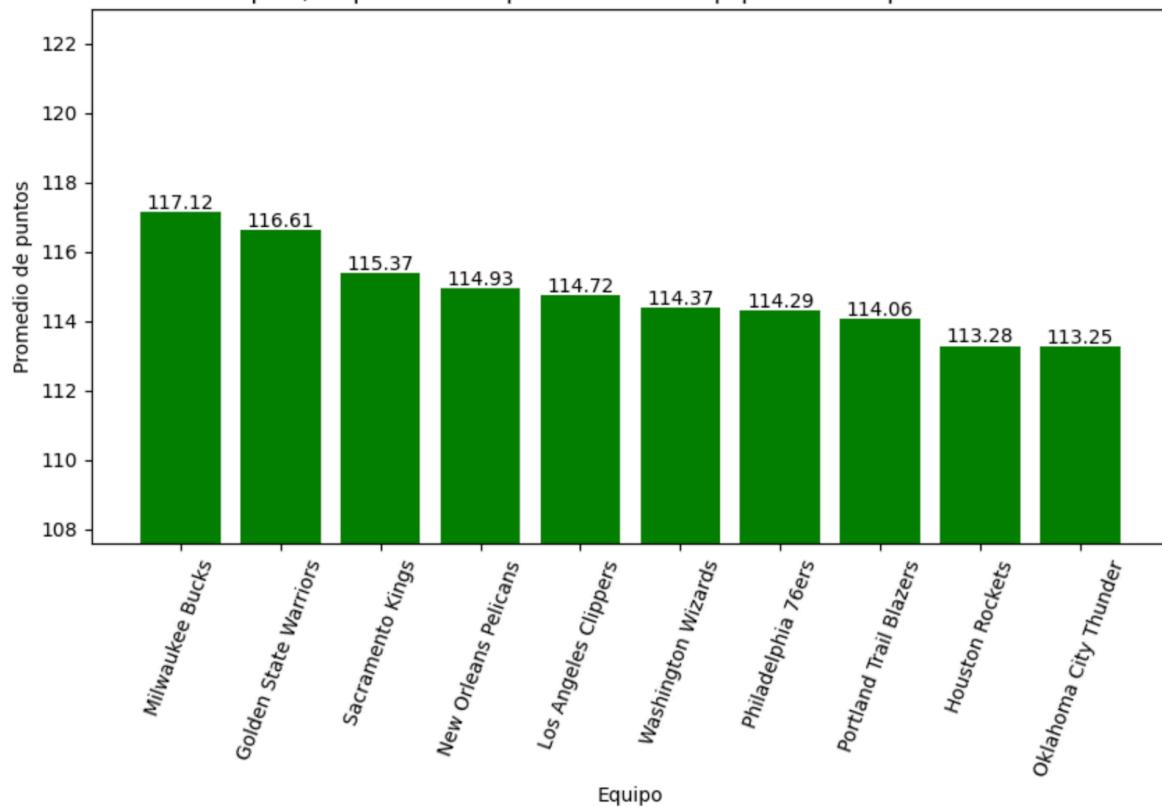
for i, v in enumerate(df['PROMEDIO']):
    plt.text(i, v + 0.1, f"{v:.2f}", ha='center')

plt.xticks(rotation=70)
plt.show()
```

Top 10, de promedio de puntos de cada equipo en la temporada 2018



Top 10, de promedio de puntos de cada equipo en la temporada 2018

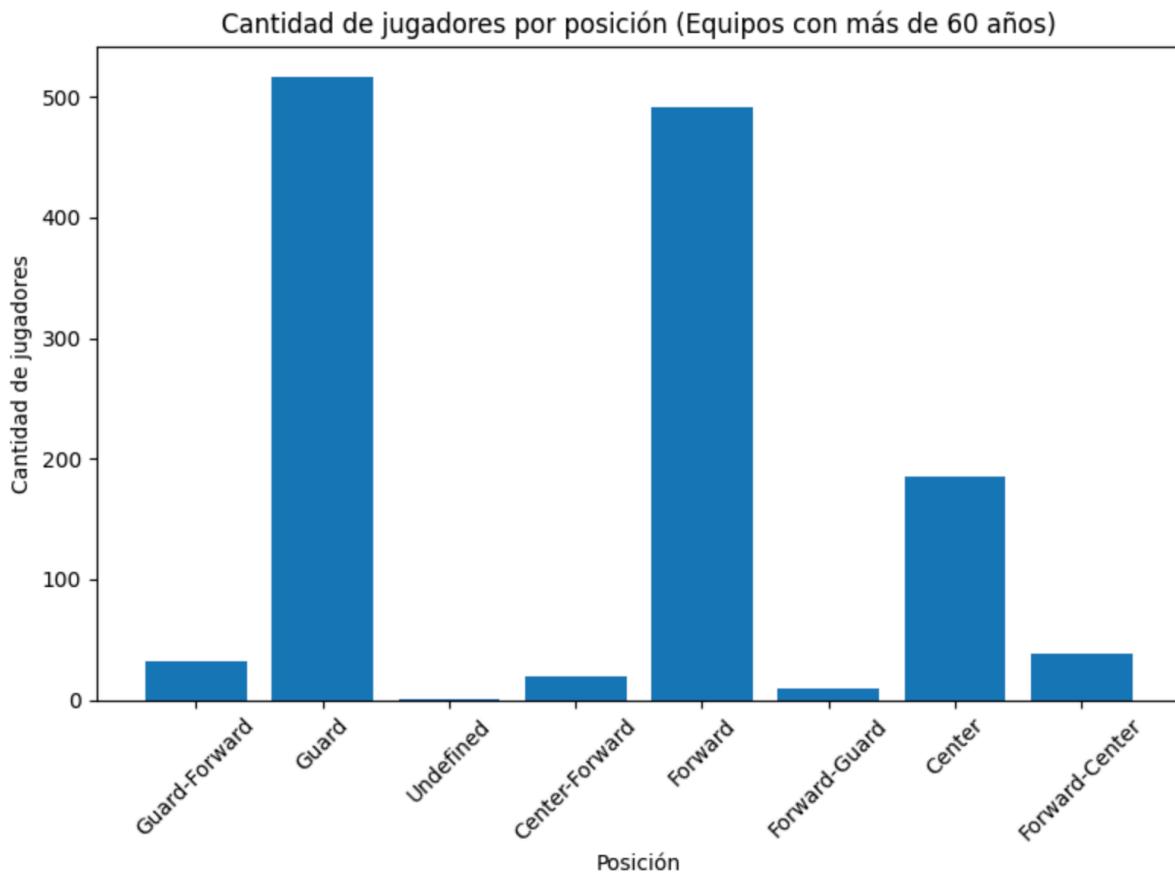


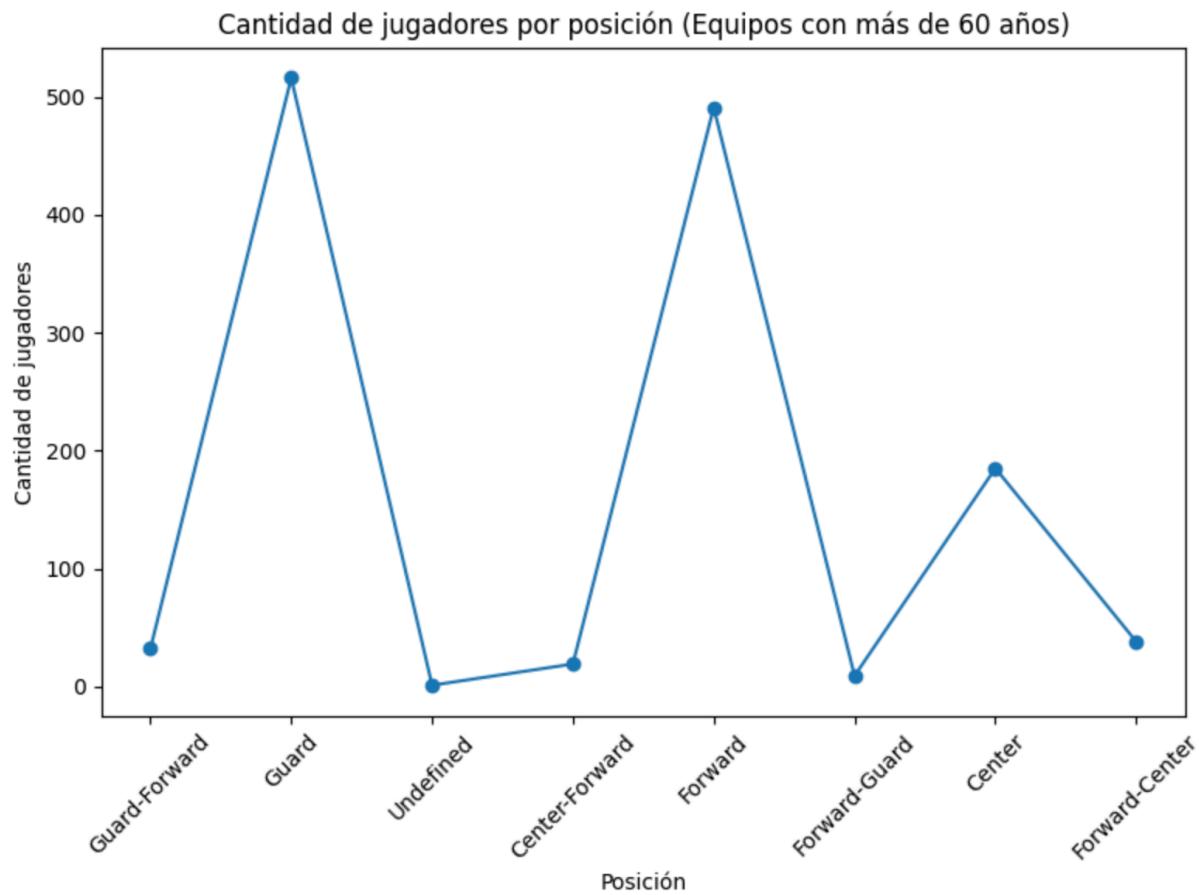
Visualizaciones de la consulta 2

```
df = query2.toPandas()

# VISUALIZACION 1
plt.figure(figsize=(8,6))
plt.bar(df['POSICION'], df['CANTIDAD'])
plt.xlabel('Posición')
plt.ylabel('Cantidad de jugadores')
plt.title('Cantidad de jugadores por posición (Equipos con más de 60 años)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# VISUALIZACION 2
plt.figure(figsize=(8,6))
plt.plot(df['POSICION'], df['CANTIDAD'], marker='o')
plt.xlabel('Posición')
plt.ylabel('Cantidad de jugadores')
plt.title('Cantidad de jugadores por posición (Equipos con más de 60 años)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```





Visualizaciones de la consulta 3

```
df = query3.toPandas()

# VISUALIZACION 1
labels = [f'{nombre} ({camiseta})' for nombre, camiseta in zip(df['NOMBRE'], df['CAMISETA'])]
allvals = df['TRIPLES'].astype(int).tolist()
total = builtins.sum(allvals)

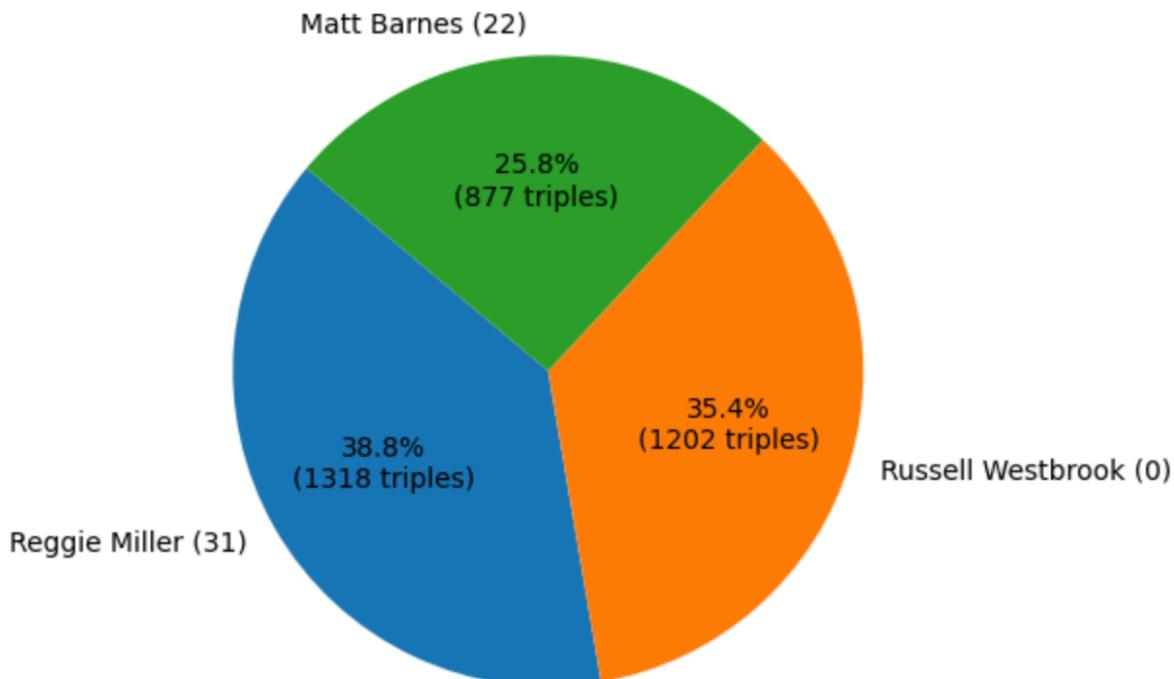
def func(pct):
    absolute = int(builtins.round(pct / 100 * total))
    return f'{pct:.1f}%\n{absolute} triples'

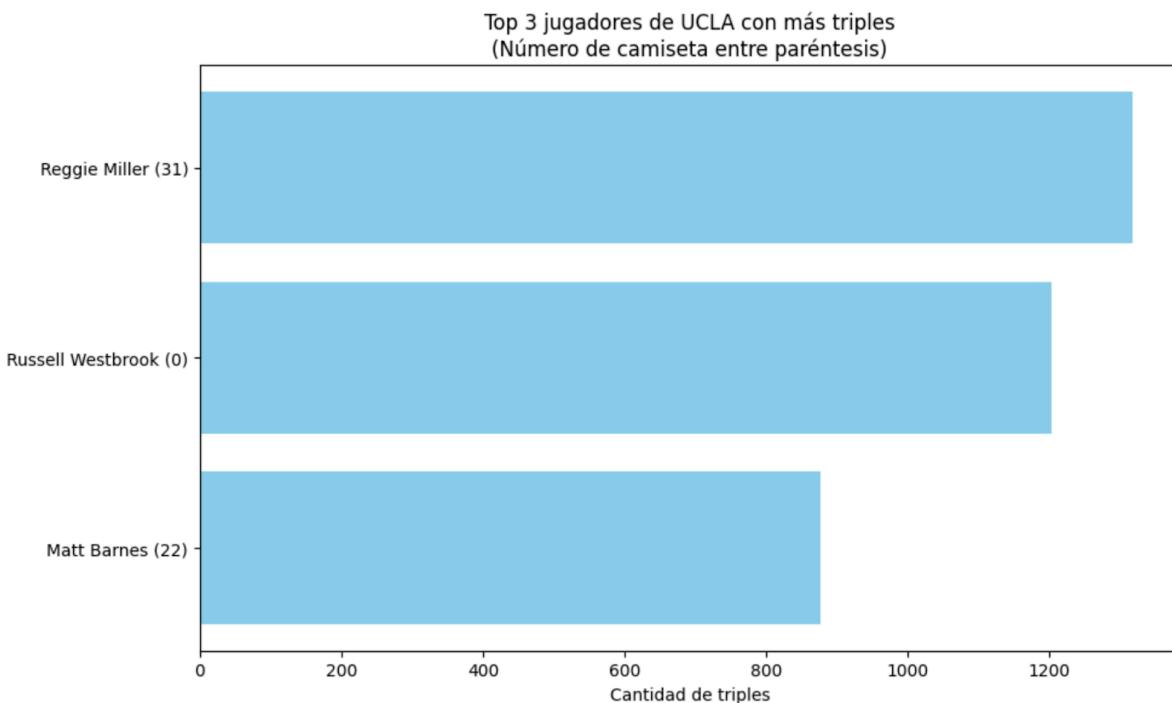
plt.pie(
    allvals,
    labels=labels,
    autopct=func,
    startangle=140
)
plt.title('Porcentaje y cantidad de triples por jugador (UCLA)\n(Número de camiseta en paréntesis)')
plt.tight_layout()
plt.show()

#VISUALIZACION 2
labels = [f'{nombre} ({camiseta})' for nombre, camiseta in zip(df['NOMBRE'], df['CAMISETA'])]
values = df['TRIPLES'].astype(int).tolist()

plt.figure(figsize=(10, 6))
plt.barh(labels, values, color='skyblue')
plt.xlabel("Cantidad de triples")
plt.title("Top 3 jugadores de UCLA con más triples\n(Número de camiseta entre paréntesis)")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

Porcentaje y cantidad de triples por jugador (UCLA)
 (Número de camiseta en paréntesis)





Creación de tablas Hive para las consultas

Para las respuestas de las consultas no visualizadas, es decir, la respuesta de la consulta 4 y la respuesta de la consulta 5, se crean tablas hive para así trabajar con su visualización con SuperSet. Primero, estas repuestas se guardan como PARQUET en HDFS, en un directorio de analítica de datos (anl), luego sí creando las tablas hive, recordando de especificar como location, el path donde están guardadas en HDFS

GUARDAR LAS RESPUESTAS COMO PARQUET

```
query4.write \
    .mode("overwrite") \
    .option("header", True) \
    .parquet("/data_obligatorio_NBA/anl/resultado_pregunta4")

query5.write \
    .mode("overwrite") \
    .option("header", True) \
    .parquet("/data_obligatorio_NBA/anl/resultado_pregunta5")
```

CREAR TABLAS HIVE PARA LAS RESPUESTAS

```

spark.sql("DROP TABLE IF EXISTS pregunta4")

spark.sql("USE obligatorioNBA")

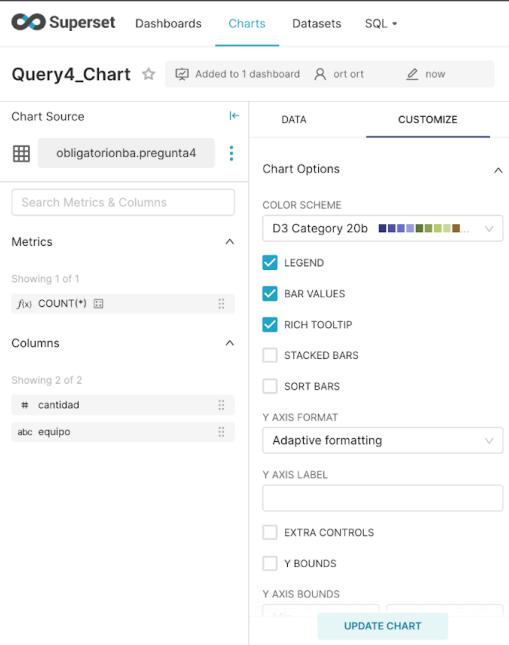
spark.sql("""
CREATE EXTERNAL TABLE pregunta4
(
CANTIDAD BIGINT,
EQUIPO STRING
)
STORED AS PARQUET
LOCATION '/data_obligatorio_NBA/anl/resultado_pregunta4'
""")
```

Visualización en SuperSet

Apache Superset es una plataforma de visualización de datos entre otras funcionalidades. Permite crear dashboards y realizar análisis exploratorios sobre datos. Es muy utilizado para analizar grandes volúmenes de datos conectándose a múltiples fuentes.

Para tener un acceso de las tablas creadas para las respuestas de las consultas, se necesitará conectar a la base de datos. Una vez conectado, se procederá con las visualizaciones, guardando luego los archivos resultantes en un dashboard

Visualización de la consulta 4



The screenshot shows the Apache Superset interface with the following details:

- Header:** Shows "Superset" logo, "Dashboards", "Charts" (which is the active tab), "Datasets", and "SQL".
- Query Selection:** "Query4_Chart" is selected, with a note "Added to 1 dashboard".
- Chart Source:** "obligatorionba.pregunta4" is selected.
- Metrics:** "COUNT(*)" is listed under "Showing 1 of 1".
- Columns:** "# cantidad" and "equipo" are listed under "Showing 2 of 2".
- Chart Options (Customize Tab):**
 - Color Scheme:** "D3 Category 20b" is selected.
 - Legend:** Checked.
 - Bar Values:** Checked.
 - Rich Tooltip:** Checked.
 - Stacked Bars:** Unchecked.
 - Sort Bars:** Unchecked.
 - Y Axis Format:** "Adaptive formatting" is selected.
 - Y Axis Label:** An empty input field.
 - Extra Controls:** Unchecked.
 - Y Bounds:** Unchecked.
 - Y Axis Bounds:** An empty input field.
- Buttons:** "UPDATE CHART" button at the bottom right.

Query5_Chart



Visualización de la consulta 5

Add the name of the chart

Chart Source

obligatorionba.pregunta5

Search Metrics & Columns

Metrics Showing 1 of 1 COUNT(*)

Columns Showing 2 of 2 # promedio # temporada

DATA CUSTOMIZE

SANKEY DIAG... View all charts

Time

Query

SOURCE / TARGET

- # promedio
- # temporada

METRIC

- # COUNT(promedio)

FILTERS

+ Drop columns/metrics here or click

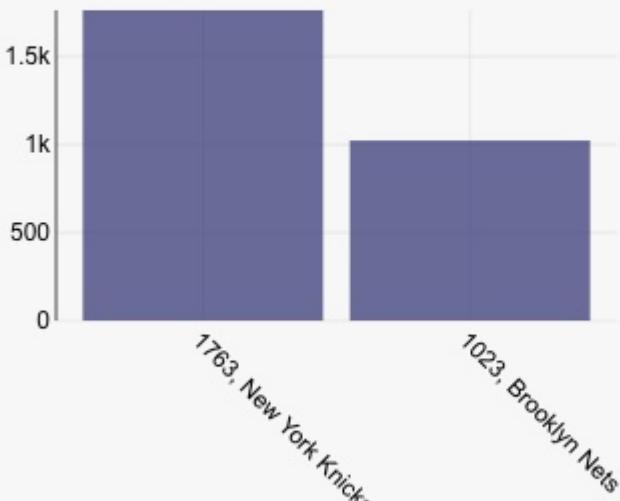
ROW LIMIT 10000

SORT BY METRIC

UPDATE CHART

Query4_Chart

● SUM(cantidad)



Parte 2

En esta sección se presenta una propuesta alternativa para la construcción de un Data Lake, utilizando un stack tecnológico distinto al utilizado en el obligatorio, exceptuando el motor de procesamiento Spark.

La solución sigue un enfoque de Data Lake con arquitectura ETL, donde los datos son extraídos desde distintas fuentes, luego transformados mediante herramientas de procesamiento, y finalmente almacenados en distintas zonas del Data Lake (landing, refined, modeled y analytics)

A continuación, se detalla el conjunto de tecnologías seleccionadas, la función que cumple cada una dentro, y un diagrama que representa el flujo desde las fuentes hasta los usuarios de negocio.

Stack Tecnológico

AWS DataSync

AWS DataSync es un servicio de Amazon de transferencia de datos diseñado para mover grandes volúmenes. Esta herramienta facilita la transacción entre sistemas on-premises (locales) y almacenamientos en la nube (como lo es Amazon S3).

La herramienta también permite una realización de una ingesta capaz de automatizarse, pudiendo generar un escenario de captura de datos periódica y automática.

El servicio de Amazon sustituiría lo que actualmente se encarga Apache NiFi, a su vez capaz de procesar datos de diversas fuentes, y en este caso por ejemplo, enviárslos a HDFS

Talend

Talend es una plataforma de integración de datos, capaz de manejar procesos ETL (Extract, Transform, Load). Se puede conectar fácilmente a múltiples fuentes y destinos de datos.

Al ser un intermediario entre la fuente de datos y un almacenamiento en la nube por ejemplo, se podrían aplicar buenas prácticas, y mejor dicho protocolos de seguridad, para garantizar una extracción y transferencia segura y confiable.

En el modelo original, la herramienta también sustituye a Apache NiFi

Amazon S3

Amazon S3 es un sistema de almacenamiento en la nube, comúnmente utilizado en el mercado como componente de un Data Lakes. Permite organizar los datos en buckets y carpetas según distintas zonas definidas, por ejemplo las utilizadas: landing, refined, modeled, analytics.

S3 funciona como un contenedor centralizado de los datos de un datalake, conteniendo tanto los datos crudos, manipulados y listos para el consumo final.

Esto sustituye a lo que vendría a ser HDFS (Hadoop Distributed File System

Apache Spark

Apache Spark es un motor de procesamiento distribuido para datos a gran escala. Es capaz de realizar transformaciones sobre ellos con Python.

Spark se usa para el procesado de datos raw que llegan al bucket landing de s3. Allí se pueden realizar operaciones como limpieza de datos, refinado, filtrado, y más.

Este motor no sería un nuevo stack inutilizado, en el modelo original también estaba presente

AWS Glue

AWS Glue es un conjunto de servicio que son capaces de la manipulación de los datasets almacenados en S3. Permite que otras herramientas posteriores sean capaz de la consulta sobre estos como si fueran tablas.

Glue sustituye a Hive en el flujo original.

Snowflake

Snowflake es una plataforma de datos en la nube que permite almacenar, consultar y transformar datos. Ofrece un entorno de SQL para la construcción de estos modelos

En el modelo original, como sustituyente de esta se usa Hive y Spark, modelando y estructurando con ellas

Amazon Athena

Amazon Athena es un servicio de consultas SQL que permite consultar directamente archivos almacenados en S3- De esta manera, permite una consulta y exploración simplificada.

En el modelo actual, Athena estaría asociado a Hive, cumplen una función similar sobre el DataLake.

Amazon Quicksight

Por último, Amazon QuickSight es una herramienta de visualización de datos, que permite la creación de dashboards conectados a las fuentes.

Quicksight reemplaza a Apache SuperSet, ambas cumplen el mismo rol, el del visualizado final para posterior análisis y comprensión.

Flujo de datos

En la sugerencia de DataLake, DataSync es utilizado para la transferencia entre archivos on-premise. Estos archivos fueron representados con una variabilidad, entre los que se incluye archivos xml, parquet, json, csv. El destino de esta carga es un bucket en la nube de Amazon S3, bucket que representa una zona de landing, es decir, una zona donde almacena los registros crudos de las fuentes de datos para una posterior manipulación y manejo con ellos.

Siguiendo, en la propuesta diseñada, Talend es utilizado para ser encargado de la extracción de datos desde las multiples bases de datos MySQL, SQL Server, PostgreSQL, entre otras. Al igual que con DataSync, esta ingesta dirige la ingesta hacia el bucket en la nube landing, siguiendo la misma metodología de arquitectura de zonas, manteniendo un flujo ordenado y eficiente.

Como bien se menciona, las anteriores desembocan en amazon S3, funcionando este como repositorio central organizado en distintas zonas: landing para el almacenaje de datos crudos, refined para los datos refinados, modeled para los modelados y por último analytics, ubicación final para ser consultados.

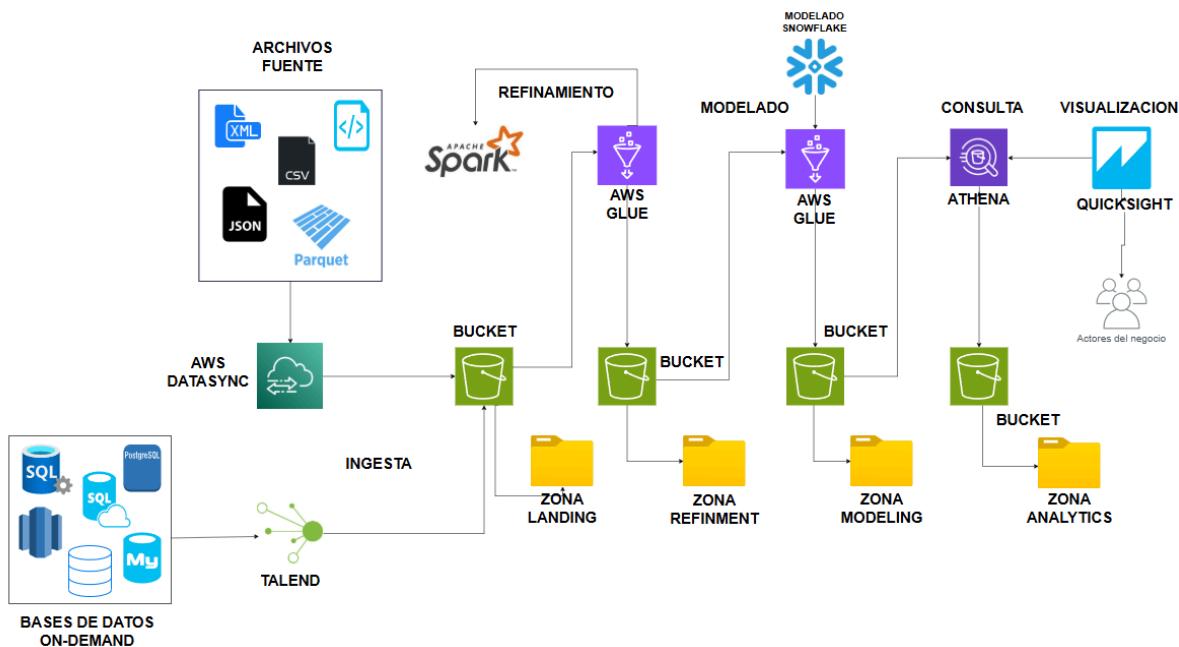
AWS Glue luego, analiza los buckets landing y refined para la construcción de tablas sobre los registros, siendo utilizadas de manera futura en la visualización y consulta

Con Snowflake, en la etapa del modelado brinda gran utilidad. Se construyen tablas agregadas, y kpis para análisis

Siguiendo el flujo, Athena se usa para la ejecución de consultas SQL sobre datos modelados en S3. A su vez, prepara los datos para visualizaciones surgidas sobre la zona analytics

Por último, QuickSight se utiliza en el modelo propuesto para la etapa final del visualizado, construyendo dashboards sobre los mismos y funcionando como punto de acceso para el análisis por parte de los usuarios del negocio, usuarios no técnicos

Diagrama



Aprendizajes Finales

El proyecto implementado atraviesa una amplia variedad de operaciones, tareas, metodologías y tecnologías. A lo largo del trabajo, se adquirió experiencia en la práctica de lo que verdaderamente implica construir un Data Lake, con sus beneficios y también sus desafíos.

Por un lado, permite una estructuración organizada y limpia del stack tecnológico utilizado y del flujo de datos diseñado. Tener una visualización clara de esto facilita en gran medida la comprensión del proceso completo, desde la ingesta hasta el consumo final.

Además, nos familiarizamos con la gran cantidad de herramientas necesarias para la construcción de un modelo de este tipo. Dado que existen múltiples alternativas para cada etapa del flujo, es fundamental evaluar el valor que aporta cada tecnología según el contexto. Esto pone en evidencia que construir un Data Lake no solo requiere conocimientos técnicos, sino también una inversión considerable, tanto en tiempo como en recursos.

En la etapa final de investigación, se destacó el hecho de que muchas de las tecnologías utilizadas son desarrolladas por Amazon, lo cual ilustra la participación de este proveedor en el mundo de la analítica en la nube.

En definitiva, este trabajo permitió comprender que el diseño de un Data Lake no es solamente un sobre lo técnico, deben tenerse presentes conceptos como costos, escalabilidad y mantenimiento.