



NBA STATISTICS ANALYSIS



Roberto Manzano López

INDICE:

1. Objetivo	pág. 3
2. Herramientas	pág. 3
3. Estado del arte	pág. 4
4. Datos	pág. 5
5. El proyecto	pág. 7
5.1. Limpieza de datos	pág. 8
5.2. Modificado del dataframe	pág. 11
5.2.1. Modifing_dataframe	pág. 11
5.2.2. Modifing_dataframe2	pág. 13
5.3. Exploración de los datos	pág. 21
5.3.1. Evolución	pág. 21
5.3.2. Distribución de los datos	pág. 24
5.3.3. Correlación de los datos	pág. 27
5.4. Modelado	pág. 30
5.5. Visualización	pág. 40
5.5.1. Dashboard 1: Evolución	pág. 40
5.5.2. Dashboard 2: Puntos, rebotes y asistencias	pág. 41
5.5.3. Dashboard 3: Victorias y %FG	pág. 42
6. Conclusiones	pág. 42



I. OBJETIVO

El objetivo de este proyecto es realizar un análisis de las principales estadísticas generadas en los partidos de baloncesto de la NBA e intentar predecir para un partido, qué equipo ganará.

El trabajo muestra la utilización de diferentes técnicas y procesos que un data scientist debe conocer para poder extraer información útil de los datos. En el desarrollo del proyecto se lleva a cabo la limpieza de los datos, su preparación, una exploración de los mismos, se realiza “ingeniería del dato” para obtener información adicional y finalmente obtener un modelo, además de una completa visualización que permite comprender de manera más profunda y detallada el conjunto de datos con los que se va a trabajar.

2. HERRAMIENTAS

Para la realización del proyecto se han utilizado Jupyter Notebook y Tableau. Jupyter Notebook es una aplicación web que permite crear y compartir documentos con código, en concreto código Python, lenguaje en el que se desarrolla el trabajo por completo, más concretamente Python 3. Gracias a esta herramienta se pueden crear pequeñas celdas con código y ejecutarlas independientemente, facilitando mucho la interacción.

El sistema operativo utilizado ha sido Ubuntu por la gran compatibilidad que tiene con la mayoría de programas utilizados por un data science, a excepción de la herramienta de visualización escogida, Tableau, que ha sido utilizada sobre Windows 11.

Para instalar Jupyter Notebook basta con seguir estos sencillos pasos:

- Abrir la consola (Ctrl+Alt+T)
- sudo apt update; sudo apt upgrade
- sudo apt install python3-pip python3-dev
- sudo -H pip3 install --upgrade pip
- pip --version
- sudo -H pip3 install virtualenv
- mkdir jupyter
- cd jupyter
- virtualenv jupyter
- source jupyter/bin/activate
- pip install jupyter

Con esto finaliza la instalación de Jupyter Notebook. Para iniciarla se debe abrir la consola y ejecutar la sentencia “jupyter notebook”, esto abrirá en el navegador la aplicación y estará lista para su uso.

Para instalar Tableau hay que ir a su página web y seguir los pasos que indican:
<https://www.tableau.com/es-es>

3. ESTADO DEL ARTE

Antes de continuar, se debe conocer un poco el tema que se va a tratar, el baloncesto. NBA son las siglas de National Basketball Association, es decir, asociación nacional de baloncesto. Es la liga de baloncesto estadounidense, participan 30 equipos divididos en dos conferencias, Este y Oeste, cada una cuenta con 15 equipos. Cada equipo juega durante la temporada un total de 82 partidos (que pueden ser más si se clasifican para playoffs). Durante la temporada los equipos no juegan el mismo número de veces contra cada uno, si no que juegan un mayor número de partidos con los equipos de su misma conferencia. Esto se debe a la gran superficie que ocupa EEUU, que imposibilita que los equipos puedan desplazarse tanta distancia y tan a menudo, ya que la liga se juega entre los meses de Octubre (finales) y Junio (mediados) hay una gran cantidad de partidos a la semana, por ejemplo, un equipo puede llegar a jugar 2, 3 o incluso 4 partidos una misma semana.

En cuanto a la liga no es necesario que se conozca mucho más para el objetivo, pero si es necesario que comentemos un poco más sobre baloncesto. En un partido de baloncesto juegan 5 jugadores de un equipo contra 5 jugadores del otro equipo, el partido está dividido en 4 cuartos de 12 minutos (10 min para el resto del mundo) de duración y gana el equipo que más puntos anote en la canasta rival. En caso de empate se juegan tantas prórrogas de 5 min como fueran necesarias hasta terminar desempatando.

Es también importante conocer el desarrollo de una temporada. Como se ha dicho cada equipo juega un total de 82 partidos durante una temporada, estos 82 partidos pertenecen a lo que se conoce como “regular season”, es decir, temporada regular. Al final de la temporada se obtiene la tabla de clasificación con el total de partidos jugados, ganados y perdidos. Cada victoria se cuenta como 1 y cada derrota como 0, a medida que avanza la temporada se van sumando las victorias de cada equipo y la tabla de clasificación va cambiando hasta que todos los equipos han jugado los 82 partidos y obtenemos la tabla de clasificación final. Aquí termina la temporada regular, y empiezan los playoffs, a esta fase pasan los 8 mejores de cada conferencia (más recientemente la liga ha introducido una nueva fase llamada play-in en la que

los equipos clasificados en las posiciones 7, 8, 9 y 10 se disputan 2 plazas en los playoffs). En este trabajo solo se atenderá a los datos correspondientes a la temporada regular, dejando de lado los playoffs.

Esta decisión se debe a que durante la temporada regular existe una uniformidad con la cantidad de partidos y quienes los juegan, mientras que en los playoffs no participan todos los equipos y la dinámica de enfrentamientos para cada uno puede cambiar respecto de la otro equipo. Esto se debe a que cada ronda de clasificación en los playoffs la gana el mejor de 7 partidos. Es decir, en cada ronda hay partidos de octavos, cuartos, semis y final (la terminología americana es distinta, ellos en lugar de semis lo llaman “final de conferencia”, y a la final “The finals”), en los que los dos equipos que se enfrentan juegan al mejor de 7 partidos (repartidos entre ambos canchas, 4 para el equipo que mejor clasificación final obtuvo y 3 para el otro), es decir el primero que gane 4 se clasifica y el otro termina la competición hasta la próxima temporada. Como los playoffs cambian cada año y dependen de cómo se ha desarrollado la temporada regular se ha tomado la decisión de utilizar solo los partidos de temporada regular, que no cambian y cada año es lo mismo (salvo por alguna excepción a lo largo de la historia, que comentaremos más adelante si fuera necesario).

Durante un partido se generan diferentes estadísticas que permiten saber cómo se desarrolló el partido para cada equipo, algunas de estas estadísticas son: los puntos anotados por cada equipo, las asistencias repartidas, los rebotes cogidos, el porcentaje de acierto, etc... Más adelante se comentará en detalle el significado de cada estadística.

4. DATOS

Antes de descargar los datos se ha creado la carpeta “data”, que se utilizará para guardar los datos iniciales y los dataframes que vayan generando a partir de ellos.

Los datos han sido obtenidos de un repositorio en kaggle “<https://www.kaggle.com/nathanlauga/nba-games/version/8?select=games.csv>”, y “<https://www.kaggle.com/datasets/nathanlauga/nba-games/version/8?select=teams.csv>” que previamente fueron obtenidos de la API de la NBA.

El grueso del trabajo se realiza con el archivo games.csv que corresponde con el primer enlace. En este archivo están los datos que abarcan todos los partidos jugados desde la temporada 2003-2004 hasta la temporada 2018-2019, un total de registros de 25024. Las variables con las que cuenta inicialmente son un total de 21, y son las siguientes:

- GAME_DATE_EST: Fecha del partido.
- GAME_ID: Id del partido (único para cada partido).
- GAME_STATUS_TEXT: Estado del partido.
- HOME_TEAM_ID: Id del equipo que juega como local.
- VISITOR_TEAM_ID: Id del equipo que juega como visitante.
- SEASON: Año de la temporada.
- TEAM_ID_home: Id del equipo que juega como local.
- PTS_home: Puntos anotados por el equipo local.
- FG_PCT_home: Porcentaje de acierto en canastas de 2 puntos por el equipo local. Los lanzamientos de 2 puntos son los que se realizan estando situado en cualquier punto entre la canasta y la línea de triple.
- FT_PCT_home: Porcentaje de acierto en tiros libres por el equipo local. Del total de tiros libres (lanzamientos situados desde la línea de personal, lugar desde el cual se ejecutan los tiros si durante el lanzamiento se hace falta al jugador), % de acertados.
- FG3_PCT_home: Porcentaje de acierto en canastas de 3 puntos para el equipo local. Los lanzamientos de 3 puntos son los que se realizan estando situado en cualquier sitio que quede por detrás de la línea de triple.
- AST_home: Asistencias realizadas por el equipo local. Se cuenta como asistencia el pase previo a una canasta, si esta se consigue justo en la siguiente acción tras el pase.
- REB_home: Rebotes obtenido por el equipo local. Un rebote se consigue si atrapas el balón después de un tiro que no acaba en canasta.
- TEAM_ID_away: Id del equipo que juega como visitante.
- PTS_away: Puntos anotados por el equipo visitante.
- FG_PCT_away: Porcentaje de acierto en canastas de 2 puntos por el equipo visitante. Los lanzamientos de 2 puntos son los que se realizan estando situado en cualquier punto entre la canasta y la línea de triple.

- FT_PCT_away: Porcentaje de acierto en tiros libres por el equipo visitante. Del total de tiros libres (lanzamientos situados desde la línea de personal, lugar desde el cual se ejecutan los tiros si durante el lanzamiento se hace falta al jugador), % de acertados.
- FG3_PCT_away: Porcentaje de acierto en canastas de 3 puntos para el equipo visitante. Los lanzamientos de 3 puntos son los que se realizan estando situado en cualquier sitio que quede por detrás de la línea de triple.
- AST_away: Asistencias realizadas por el equipo local. Se cuenta como asistencia el pase previo a una canasta, si esta se consigue justo en la siguiente acción tras el pase.
- REB_away: Rebotes obtenido por el equipo local. Un rebote se consigue si atrapas el balón después de un tiro que no acaba en canasta.
- HOME_TEAM_WINS: Indica con un 1 si el equipo local ha ganado ese partido o con un 0 si el equipo local ha perdido ese partido.

Inicialmente se tiene un dataset formado por 21 columnas y 25024 filas.

Ahora que se conocen los datos con los que se va a trabajar, y están descargados y guardados y en la carpeta “data” es importante saber cómo se ha ordenado el documento y cuál ha sido la forma de trabajar.

5. EL PROYECTO

El proyecto está dividido en notebooks de jupyter. Cada notebook tiene un objetivo, y su nomenclatura está determinada por la tarea que se realiza en él. De esta forma se tienen los siguientes notebooks, según cual debe ser su orden de ejecución:

- data_clean.ipynb
- modifying_dataframe.ipynb
- modifying_dataframe2.ipynb
- data_exploration.ipynb
- model_creation.ipynb
- (create_data.py)

5.1 LIMPIEZA DE LOS DATOS: DATA_CLEAN.IPYNB

El primer notebook es data_clean.ipynb, en él se realiza la limpieza de los datos. Se cargan los datos, se eliminan las columnas no deseadas, se comprueban los huecos y se toman las decisiones de qué hacer con los mismos. A continuación se detallan las acciones realizadas:

Carga de los datos.

```
In [5]: # Cargamos los datos: leemos el csv y obtenemos un dataframe
df_nba_games = pd.read_csv('data/games.csv')
df_nba_games
```

	GAME_DATE_EST	GAME_ID	GAME_STATUS_TEXT	HOME_TEAM_ID	VISITOR_TEAM_ID	SEASON	TEAM_ID_home	PTS_home	FG_PCT_home	FT_PCT_
0	2021-11-17	22100213	Final	1610612766	1610612764	2021	1610612766	97.0	0.438	
1	2021-11-17	22100214	Final	1610612765	1610612754	2021	1610612765	97.0	0.425	
2	2021-11-17	22100215	Final	1610612737	1610612738	2021	1610612737	110.0	0.506	
3	2021-11-17	22100216	Final	1610612751	1610612739	2021	1610612751	109.0	0.458	
4	2021-11-17	22100217	Final	1610612748	1610612740	2021	1610612748	113.0	0.483	
...
25019	2014-10-06	11400007	Final	1610612737	1610612740	2014	1610612737	93.0	0.419	
25020	2014-10-06	11400004	Final	1610612741	1610612764	2014	1610612741	81.0	0.338	
25021	2014-10-06	11400005	Final	1610612747	1610612743	2014	1610612747	98.0	0.448	
25022	2014-10-05	11400002	Final	1610612761	1610612758	2014	1610612761	99.0	0.440	
25023	2014-10-04	11400001	Final	1610612748	1610612740	2014	1610612748	86.0	0.431	

25024 rows × 21 columns

Se modifica el tipo de dato de la columna “GAME_DATE_EST” y se seleccionan solo los valores tipo numéricos y tipo fecha.

```
In [4]: # Cambiamos el tipo de dato que tiene GAME_DATE_EST
df_nba_games['GAME_DATE_EST'] = pd.to_datetime(df_nba_games['GAME_DATE_EST'], format = "%Y-%m-%d")
```

```
In [5]: # Nos quedamos solo con los valores numericos y con date que lo necesitaremos mas adelante
df_nba_games_only_numerics = df_nba_games.select_dtypes(include=[np.number, np.datetime64])
df_nba_games_only_numerics
```

	GAME_DATE_EST	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	SEASON	TEAM_ID_home	PTS_home	FG_PCT_home	FT_PCT_home	FG3_PCT_home
0	2021-11-17	22100213	1610612766	1610612764	2021	1610612766	97.0	0.438	0.500	0.313
1	2021-11-17	22100214	1610612765	1610612754	2021	1610612765	97.0	0.425	0.750	0.286
2	2021-11-17	22100215	1610612737	1610612738	2021	1610612737	110.0	0.506	0.833	0.351
3	2021-11-17	22100216	1610612751	1610612739	2021	1610612751	109.0	0.458	0.840	0.375
4	2021-11-17	22100217	1610612748	1610612740	2021	1610612748	113.0	0.483	0.824	0.375
...
25019	2014-10-06	11400007	1610612737	1610612740	2014	1610612737	93.0	0.419	0.821	0.421
25020	2014-10-06	11400004	1610612741	1610612764	2014	1610612741	81.0	0.338	0.719	0.381
25021	2014-10-06	11400005	1610612747	1610612743	2014	1610612747	98.0	0.448	0.682	0.500
25022	2014-10-05	11400002	1610612761	1610612758	2014	1610612761	99.0	0.440	0.771	0.333
25023	2014-10-04	11400001	1610612748	1610612740	2014	1610612748	86.0	0.431	0.679	0.333

Se comprueban los huecos en blanco, cuántos hay y a que columnas pertenecen

```
In [8]: # Cuantos valores NaN hay para una columna:
for column in columns_with_nan:
    count_nan = df_nba_games_only_numerics[column].isna().sum()
    print ('Column: ' + str(column) + ' has : ' + str(count_nan) + ' of NaN')
```

Column: PTS_home has :99 of NaN
 Column: FG_PCT_home has :99 of NaN
 Column: FT_PCT_home has :99 of NaN
 Column: FG3_PCT_home has :99 of NaN
 Column: AST_home has :99 of NaN
 Column: REB_home has :99 of NaN
 Column: PTS_away has :99 of NaN
 Column: FG_PCT_away has :99 of NaN
 Column: FT_PCT_away has :99 of NaN
 Column: FG3_PCT_away has :99 of NaN
 Column: AST_away has :99 of NaN
 Column: REB_away has :99 of NaN

Se observan 99 registros vacíos, correspondientes a las variables más importantes de los partidos. Antes de tomar una decisión se comprueba a qué partidos corresponden dichos registros para tratar de determinar su importancia.

```
In [14]: df_columnas_y_filas_conflictivas
```

index	GAME_DATE_EST	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	SEASON	TEAM_ID_home	PTS_home	FG_PCT_home	FT_PCT_home	...	AST
17548	17548	2003-10-24	10300116	1610612753	2003	1610612753	NaN	NaN	NaN	...	
17549	17549	2003-10-24	10300108	1610612737	2003	1610612737	NaN	NaN	NaN	...	
17550	17550	2003-10-24	10300109	1610612738	2003	1610612738	NaN	NaN	NaN	...	
17551	17551	2003-10-24	10300113	1610612759	2003	1610612759	NaN	NaN	NaN	...	
17552	17552	2003-10-24	10300112	1610612749	2003	1610612749	NaN	NaN	NaN	...	
...
17642	17642	2003-10-09	10300019	1610612743	2003	1610612743	NaN	NaN	NaN	...	
17643	17643	2003-10-09	10300022	1610612757	2003	1610612757	NaN	NaN	NaN	...	
17644	17644	2003-10-08	10300013	1610612759	2003	1610612759	NaN	NaN	NaN	...	
17651	17651	2003-10-08	10300015	1610612747	2003	1610612747	NaN	NaN	NaN	...	
17652	17652	2003-10-07	10300006	1610612747	2003	1610612747	NaN	NaN	NaN	...	

Se observa que todos esos registros (partidos) corresponden a la temporada 2003. ¿Cuánto suponen 99 partidos para una temporada completa en la NBA?

```
In [17]: # Efectivamente son 99 partidos de la temporada 2003, vamos a ver que supone eso para esa temporada
df_nba_games_only_numerics.groupby('SEASON').count()
```

SEASON	index	GAME_DATE_EST	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	TEAM_ID_home	PTS_home	FG_PCT_home	FT_PCT_home	FG3_PCT_home
2003	1385		1385	1385	1385	1385	1286	1286	1286	1286
2004	1362		1362	1362	1362	1362	1362	1362	1362	1362
2005	1432		1432	1432	1432	1432	1432	1432	1432	1432
2006	1419		1419	1419	1419	1419	1419	1419	1419	1419
2007	1411		1411	1411	1411	1411	1411	1411	1411	1411
2008	1425		1425	1425	1425	1425	1425	1425	1425	1425
2009	1424		1424	1424	1424	1424	1424	1424	1424	1424
2010	1422		1422	1422	1422	1422	1422	1422	1422	1422
2011	1104		1104	1104	1104	1104	1104	1104	1104	1104
2012	1420		1420	1420	1420	1420	1420	1420	1420	1420
2013	1427		1427	1427	1427	1427	1427	1427	1427	1427
2014	1418		1418	1418	1418	1418	1418	1418	1418	1418
2015	1416		1416	1416	1416	1416	1416	1416	1416	1416
2016	1405		1405	1405	1405	1405	1405	1405	1405	1405
2017	1382		1382	1382	1382	1382	1382	1382	1382	1382
2018	1378		1378	1378	1378	1378	1378	1378	1378	1378
2019	1241		1241	1241	1241	1241	1241	1241	1241	1241
2020	1264		1264	1264	1264	1264	1264	1264	1264	1264
2021	289		289	289	289	289	289	289	289	289

Los partidos afectados son 99, y el total de partidos correspondientes a la temporada 2003 es de 1385, eso supone un 7,1%. Dado que es un % relativamente bajo y que no se quiere “adulterar” los datos con la media ni ninguna otra forma, se toma la decisión de eliminar esos registros conflictivos y no trabajar con ellos.

Se aprecian otras dos anomalías en el dataset: el número de partidos de la temporada 2011 y 2021.

Es importante conocer que durante la historia de la liga de la NBA se han producido diversos “lockouts” o “huelgas” que hicieron que las temporadas se vieran “comprobadas”.

- 1995: Fue el primer lockout de la historia, duró del 1 de julio al 12 de septiembre, con lo que no afectó al inicio de la temporada 95-96.
- 1996: Sólo duró unas horas. La disputa estaba en el reparto de los beneficios de los derechos televisivos.
- 1998-99: El más largo de todos, 204 días de lockout, el mayor tiempo que haya estado parada nunca la NBA. Comenzó el 1 de julio de 1998 hasta el 20 de enero de 1999, lo que provocó que la temporada regular pasase de 82 a 50 partidos por equipo.
- 2011: Fue del 1 de julio al 8 de diciembre, 161 días. Afectó a la temporada regular dejándola en 66 partidos.

Ahora se entiende por que la temporada correspondiente al año 2011 tiene muchos menos partidos que el resto. Por lo tanto se considera que para el propósito de este proyecto no afecta y es como una temporada más.

Y por supuesto la temporada 2021 corresponde al año de la pandemia provocada por la COVID-19. Se toma la decisión de eliminar todos los registros correspondientes a la temporada 2021 y no trabajar con esos datos, ya que la situación mundial afectó tanto al normal desarrollo de los partidos de la liga que podría afectar a todo el estudio realizado durante el proyecto.

Como se mencionó anteriormente el proyecto se centra en los partidos correspondientes a la temporada regular, y en el dataset actual aún quedan los partidos correspondientes a playoffs y pretemporada, por lo tanto se eliminan filtrando por la fecha de inicio y de fin de cada temporada regular que se encuentra en el dataset.

Con todas estas acciones se obtiene el dataframe “df_all_rs” que se guarda en la carpeta “data”.

In [54]:	df_all_rs											
Out[54]:	level_0	Index	GAME_DATE_EST	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	SEASON	TEAM_ID_home	PTS_home	FG_PCT_home	FT_PCT_home	
0	16359	16359	2004-04-14	20301188	1610612746	1610612760	2003	1610612746	87.0	0.423	0.727	
1	16360	16360	2004-04-14	20301184	1610612759	1610612743	2003	1610612759	93.0	0.424	0.679	
2	16361	16361	2004-04-14	20301181	1610612754	1610612741	2003	1610612754	101.0	0.420	0.794	
3	16362	16362	2004-04-14	20301177	1610612764	1610612740	2003	1610612764	78.0	0.375	0.714	
4	16363	16363	2004-04-14	20301179	1610612752	1610612739	2003	1610612752	90.0	0.481	0.714	
...	
19393	19332	19332	2018-10-17	21800011	1610612758	1610612762	2018	1610612758	117.0	0.516	0.667	
19394	19333	19333	2018-10-17	21800012	1610612746	1610612743	2018	1610612746	98.0	0.398	0.833	
19395	19334	19334	2018-10-17	21800013	1610612756	1610612742	2018	1610612756	121.0	0.543	0.875	
19396	19335	19335	2018-10-16	21800001	1610612738	1610612755	2018	1610612738	105.0	0.433	0.714	
19397	19336	19336	2018-10-16	21800002	1610612744	1610612760	2018	1610612744	108.0	0.442	0.944	

In [55]: # DATAFRAME DEFINITIVO CON EL QUE TRABAJAR -> df_all_rs
df_all_rs.to_csv('data/df_all_rs.csv')

5.2 MODIFICADO DEL DATAFRAME MODIFYING_DATAFRAME.ipynb Y MODIFYING_DATAFRAME2.ipynb

5.2.1 MODIFYING_DATAFRAME.ipynb

En este notebook se agregan los datos del otro archivo que se ha guardado previamente en "data" como "teams.csv".

Se cargan los datos y se seleccionan solo el id del equipo, la abreviación (corresponde a las siglas de la ciudad de la que es equipo) y el nombre.

In [3]: df_teams = pd.read_csv('data/teams.csv')
df_teams = df_teams[['TEAM_ID', 'ABBREVIATION', 'NICKNAME']]

Out[3]:	TEAM_ID	ABBREVIATION	NICKNAME
0	1610612737	ATL	Hawks
1	1610612738	BOS	Celtics
2	1610612740	NOP	Pelicans
3	1610612741	CHI	Bulls
4	1610612742	DAL	Mavericks
5	1610612743	DEN	Nuggets
6	1610612745	HOU	Rockets
7	1610612746	LAC	Clippers
8	1610612747	LAL	Lakers

Se unen ambos dataframes y se eliminan las columnas que se han introducido al unir los datos y que no son necesarias.

```
In [4]: # unimos los dataframes de los partidos y de los equipos para tener el nombre de los equipos en el dataframe
df_merge = pd.merge(left=df,right=df_teams, left_on='HOME_TEAM_ID', right_on='TEAM_ID')
df_merge = pd.merge(left=df_merge,right=df_teams, left_on='VISITOR_TEAM_ID', right_on='TEAM_ID')
df_merge
```

Out[4]:

level_0	GAME_DATE_EST	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	SEASON	TEAM_ID_home	PTS_home	FG_PCT_home	FT_PCT_home	... FG3
0	16359	2004-04-14	20301188	1610612746	1610612760	2003	1610612746	87.0	0.423	0.727 ...
1	17528	2003-10-31	20300025	1610612746	1610612760	2003	1610612746	105.0	0.404	0.833 ...
2	15714	2005-01-12	20400514	1610612746	1610612760	2004	1610612746	103.0	0.494	0.815 ...
3	16215	2004-11-03	20400015	1610612746	1610612760	2004	1610612746	114.0	0.629	0.677 ...
4	13607	2006-04-16	20501201	1610612746	1610612760	2005	1610612746	98.0	0.455	0.781 ...
...
19393	24713	2014-11-24	21400198	1610612766	1610612746	2014	1610612766	92.0	0.413	0.800 ...
19394	23023	2015-12-30	21500474	1610612766	1610612746	2015	1610612766	117.0	0.432	0.889 ...
19395	21279	2017-02-11	21600808	1610612766	1610612746	2016	1610612766	102.0	0.435	0.750 ...
19396	20481	2017-11-18	21700228	1610612766	1610612746	2017	1610612766	102.0	0.391	0.793 ...
19397	18534	2019-02-05	21800796	1610612766	1610612746	2018	1610612766	115.0	0.419	0.846 ...

19398 rows × 27 columns

```
In [5]: #eliminamos las columnas que hemos metido de mas al hacer los merges
df_merge = df_merge.drop(['TEAM_ID_x', 'TEAM_ID_y', 'TEAM_ID_home'], axis=1)
df_merge
```

Se ordenan las columnas y se renombran otras:

```
In [6]: #ordenamos las columnas a nuestro antojo
df_merge = df_merge[['GAME_DATE_EST', 'GAME_ID', 'HOME_TEAM_ID', 'VISITOR_TEAM_ID', 'ABBREVIATION_x', 'NICKNAME_x', \
                     'ABBREVIATION_y', 'NICKNAME_y', 'SEASON', 'PTS_home', 'FG_PCT_home', 'FT_PCT_home', \
                     'FG3_PCT_home', 'AST_home', 'REB_home', 'PTS_away', 'FG_PCT_away', 'FT_PCT_away', \
                     'FG3_PCT_away', 'AST_away', 'REB_away', 'HOME_TEAM_WINS']]
```

Out[6]:

	GAME_DATE_EST	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	ABBREVIATION_x	NICKNAME_x	ABBREVIATION_y	NICKNAME_y	SEASON	PTS_home
0	2004-04-14	20301188	1610612746	1610612760	LAC	Clippers	OKC	Thunder	2003	87.
1	2003-10-31	20300025	1610612746	1610612760	LAC	Clippers	OKC	Thunder	2003	105.
2	2005-01-12	20400514	1610612746	1610612760	LAC	Clippers	OKC	Thunder	2004	103.
3	2004-11-03	20400015	1610612746	1610612760	LAC	Clippers	OKC	Thunder	2004	114.
4	2006-04-16	20501201	1610612746	1610612760	LAC	Clippers	OKC	Thunder	2005	98.
...
19393	2014-11-24	21400198	1610612766	1610612746	CHA	Hornets	LAC	Clippers	2014	92.
19394	2015-12-30	21500474	1610612766	1610612746	CHA	Hornets	LAC	Clippers	2015	117.
19395	2017-02-11	21600808	1610612766	1610612746	CHA	Hornets	LAC	Clippers	2016	102.
19396	2017-11-18	21700228	1610612766	1610612746	CHA	Hornets	LAC	Clippers	2017	102.
19397	2019-02-05	21800796	1610612766	1610612746	CHA	Hornets	LAC	Clippers	2018	115.

19398 rows × 22 columns

```
In [7]: #Renombramos algunas columnas
df_merge = df_merge.rename(columns={'GAME_DATE_EST' : 'GAME_DATE',
                                    'ABBREVIATION_x' : 'HOME_TEAM_ABBREVIATION',
                                    'NICKNAME_x' : 'HOME_TEAM_NICKNAME',
                                    'ABBREVIATION_y' : 'VISITOR_TEAM_ABBREVIATION',
                                    'NICKNAME_y' : 'VISITOR_TEAM_NICKNAME',})
```

Se ordenan las filas por fecha e id de partido.

```
In [8]: #ordenamos por game date y game_id
df_merge = df_merge.sort_values(['GAME_DATE', 'GAME_ID'])
df_merge
```

Out[8]:

	GAME_DATE	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	HOME_TEAM_ABBREVIATION	HOME_TEAM_NICKNAME	VISITOR_TEAM_ABBREVIATION	VISITOR_TEAM_NICKNAME
11562	2003-10-28	20300001	1610612755	1610612748	PHI	76ers	MIA	Bulls
12344	2003-10-28	20300002	1610612759	1610612756	SAS	Spurs	PHX	Suns
5505	2003-10-28	20300003	1610612747	1610612742	LAL	Lakers	DAL	Mavericks
11151	2003-10-29	20300004	1610612738	1610612748	BOS	Celtics	MIA	Heat
16941	2003-10-29	20300005	1610612752	1610612753	NYK	Knicks	ORL	Pelicans
...
242	2019-04-10	21801226	1610612749	1610612760	MIL	Bucks	OKC	Rockets
5243	2019-04-10	21801227	1610612759	1610612742	SAS	Spurs	DAL	Mavericks
3105	2019-04-10	21801228	1610612743	1610612750	DEN	Nuggets	MIN	Timberwolves
1971	2019-04-10	21801229	1610612746	1610612762	LAC	Clippers	UTA	Jazz
10592	2019-04-10	21801230	1610612757	1610612758	POR	Trail Blazers	SAC	Kings

19398 rows × 22 columns

Se crea la columna “VISITOR_TEAM_WINS” que indica con 1 si el equipo que juega como visitante gana el partido y con 0 si pierde.

```
In [12]: #Añadimos la columna VISITOR_TEAM_WINS que necesitaremos mas adelante
df['VISITOR_TEAM_WINS'] = np.nan
df
```

Se crea una función para crear los datos y junto con un map se aplica y se guardan en una lista, que se utiliza para llenar la columna creada. Por último se guarda el dataframe generado “df_merged_renamed_dropped_sorted_2_rs” en la carpeta “data” para su posterior uso.

```
In [13]: def fill_visitor_team_wins(n):
    if n==1:
        return 0
    else:
        return 1

In [14]: home_team_wins = list(df['HOME_TEAM_WINS'].values)
visitor_team_wins = list(map(fill_visitor_team_wins, home_team_wins))
df['VISITOR_TEAM_WINS'] = pd.Series(visitor_team_wins)
df
```

Out[14]:

	GAME_DATE	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	HOME_TEAM_ABBREVIATION	HOME_TEAM_NICKNAME	VISITOR_TEAM_ABBREVIATION	VISITOR_TEAM_NICKNAME	VISITOR_TEAM_WINS
0	2003-10-28	20300001	1610612755	1610612748	PHI	76ers	MIA	Bulls	0
1	2003-10-28	20300002	1610612759	1610612756	SAS	Spurs	PHX	Suns	0
2	2003-10-28	20300003	1610612747	1610612742	LAL	Lakers	DAL	Mavericks	0
3	2003-10-29	20300004	1610612738	1610612748	BOS	Celtics	MIA	Heat	0
4	2003-10-29	20300005	1610612752	1610612753	NYK	Knicks	ORL	Pelicans	0
...
19393	2019-04-10	21801226	1610612749	1610612760	MIL	Bucks	OKC	Rockets	0
19394	2019-04-10	21801227	1610612759	1610612742	SAS	Spurs	DAL	Wizards	0
19395	2019-04-10	21801228	1610612743	1610612750	DEN	Nuggets	MIN	Timberwolves	0
19396	2019-04-10	21801229	1610612746	1610612762	LAC	Clippers	UTA	Jazz	0
19397	2019-04-10	21801230	1610612757	1610612758	POR	Trail Blazers	SAC	Bucks	0

19398 rows × 23 columns

```
In [15]: df.to_csv('data/df_merged_renamed_dropped_sorted_2_rs.csv')
```

5.2.2 MODIFYING DATAFRAME2.IPYNB

En este notebook se quiere agregar nuevos datos al dataframe que darán un valor adicional que ningún otro dataframe tiene. Las nuevas variables que se van a agregar son las siguientes:

- PERCENT_VIC_UNITL_THIS_GAME_HOME_TEAM: Es el % de victorias hasta ese partido para el equipo local.
- PERCENT_VIC_UNITL_THIS_GAME_VISITOR_TEAM: Es el % de victorias hasta ese partido para el equipo visitante.
- PERCENT_VIC_LAST5_GAMES_HOME_TEAM: Es el % de victorias en los últimos 5 partidos para el equipo local.

- PERCENT_VIC_LAST5_GAMES_VISITOR_TEAM: Es el % de victorias en los últimos 5 partidos para el equipo visitante.
- PERCENT_VIC_LAST10_GAMES_HOME_TEAM: Es el % de victorias en los últimos 10 partidos para el equipo local.
- PERCENT_VIC_LAST10_GAMES_VISITOR_TEAM: Es el % de victorias en los últimos 10 partidos para el equipo visitante.
- AVG_POINTS_UNTIL_THIS_GAME_HOME_TEAM: Es la media de puntos obtenidos por el equipo local hasta ese partido.
- AVG_POINTS_UNTIL_THIS_GAME_VISITOR_TEAM: Es la media de puntos obtenidos por el equipo visitante hasta ese partido.
- AVG_POINTS_LAST5_HOME_TEAM: Es la media de puntos obtenidos por el equipo local en los últimos 5 partidos.
- AVG_POINTS_LAST5_VISITOR_TEAM: Es la media de puntos obtenidos por el equipo visitante en los últimos 5 partidos.
- AVG_POINTS_LAST10_HOME_TEAM: Es la media de puntos obtenidos por el equipo local en los últimos 10 partidos.
- AVG_POINTS_LAST10_VISITOR_TEAM: Es la media de puntos obtenidos por el equipo visitante en los últimos 10 partidos.
- AVG_FGPERCENT_UNTIL_THIS_GAME_HOME_TEAM: Es el % de acierto en tiros de 2 para el equipo local hasta ese partido.
- AVG_FGPERCENT_UNTIL_THIS_GAME_VISITOR_TEAM: Es el % de acierto en tiros de 2 para el equipo visitante hasta ese partido.
- AVG_FGPERCENT_LAST5_HOME_TEAM: Es el % de acierto en tiros de 2 para el equipo local en los últimos 5 partidos.
- AVG_FGPERCENT_LAST5_VISITOR_TEAM: Es el % de acierto en tiros de 2 para el equipo visitante en los últimos 5 partidos.
- AVG_FGPERCENT_LAST10_HOME_TEAM: Es el % de acierto en tiros de 2 para el equipo local en los últimos 10 partidos.

- AVG_FGPERCENT_LAST10_VISITOR_TEAM: Es el % de acierto en tiros de 2 para el equipo visitante en los últimos 10 partidos.
- AVG_FG3PCT_UNTIL_THIS_GAME_HOME_TEAM: Es el % de acierto en tiros de 3 para el equipo local hasta ese partido.
- AVG_FG3PCT_UNTIL_THIS_GAME_VISITOR_TEAM: Es el % de acierto en tiros de 3 para el equipo visitante hasta ese partido.
- AVG_FG3PCT_LAST5_HOME_TEAM: Es el % de acierto en tiros de 3 para el equipo local en los últimos 5 partidos.
- AVG_FG3PCT_LAST5_VISITOR_TEAM: Es el % de acierto en tiros de 3 para el equipo visitante en los últimos 5 partidos.
- AVG_FG3PCT_LAST10_HOME_TEAM: Es el % de acierto en tiros de 3 para el equipo local en los últimos 10 partidos.
- AVG_FG3PCT_LAST10_VISITOR_TEAM: Es el % de acierto en tiros de 3 para el equipo visitante en los últimos 10 partidos.

Estos nuevos datos se generan a partir de los que ya se tienen. Para ello se utiliza el módulo “create_data.py” creado previamente y que tiene todas las funciones necesarias para crearlos:

- home_visitor_df: Para un game id dado devuelve, el equipo de casa, el visitante y un dataframe auxiliar con todos los datos de la temporada correspondiente a dicho equipo.
- porcentaje_victorias_home_team: Calcula el porcentaje de victorias que lleva en toda la temporada hasta ese partido para el equipo que juega en casa.
- porcentaje_victorias_visitor_team: Calcula el porcentaje de victorias que lleva en toda la temporada hasta ese partido para el equipo que juega como visitante.
- porcentaje_victorias_last5_home_team: Calcula el porcentaje de victorias que lleva en los últimos 5 partidos jugados para el equipo que juega en casa.
- porcentaje_victorias_last5_visitor_team: Calcula el porcentaje de victorias que lleva en los últimos 5 partidos jugados para el equipo que juega como visitante.

- porcentaje_victorias_last10_home_team: Calcula el porcentaje de victorias que lleva en los últimos 10 partidos jugados para el equipo que juega en casa.
- porcentaje_victorias_last10_visitor_team: Calcula el porcentaje de victorias que lleva en los últimos 10 partidos jugados para el equipo que juega como visitante.
- avg_points_home_team: Calcula la media de puntos que lleva en toda la temporada hasta ese partido para el equipo que juega en casa.
- avg_points_visitor_team: Calcula la media de puntos que lleva en toda la temporada hasta ese partido para el equipo que juega como visitante.
- avg_points_last5_home_team: Calcula la media de puntos que lleva en los últimos 5 partidos para el equipo que juega en casa.
- avg_points_last5_visitor_team: Calcula la media de puntos que lleva en los últimos 5 partidos para el equipo que juega como visitante.
- avg_points_last10_home_team: Calcula la media de puntos que lleva en los últimos 10 partidos para el equipo que juega en casa.
- avg_points_last10_visitor_team: Calcula la media de puntos que lleva en los últimos 10 partidos para el equipo que juega como visitante.
- avg_fg_home_team: Calcula la media del porcentaje de acierto en tiros de campo que lleva en toda la temporada hasta ese partido para el equipo que juega en casa.
- avg_fg_visitor_team: Calcula la media del porcentaje de acierto en tiros de campo que lleva en toda la temporada hasta ese partido para el equipo que juega como visitante.
- avg_fg_last5_home_team: Calcula la media del porcentaje de acierto en tiros de campo de los últimos 5 partidos para el equipo que juega en casa.
- avg_fg_last5_visitor_team: Calcula la media del porcentaje de acierto en tiros de campo de los últimos 5 partidos para el equipo que juega como visitante.
- avg_fg_last10_home_team: Calcula la media del porcentaje de acierto en tiros de campo de los últimos 10 partidos para el equipo que juega en casa.
- avg_fg_last10_visitor_team: Calcula la media del porcentaje de acierto en tiros de campo de los últimos 10 partidos para el equipo que juega como visitante.

- avg_fg3_home_team: Calcula la media del porcentaje de acierto en tiros de tres puntos que lleva en toda la temporada hasta ese partido para el equipo que juega en casa.
- avg_fg3_visitor_team: Calcula la media del porcentaje de acierto en tiros de tres puntos que lleva en toda la temporada hasta ese partido para el equipo que juega como visitante.
- avg_fg3_last5_home_team: Calcula la media del porcentaje de acierto en tiros de tres puntos de los últimos 5 partidos para el equipo que juega en casa.
- avg_fg3_last5_visitor_team: Calcula la media del porcentaje de acierto en tiros de tres puntos de los últimos 5 partidos para el equipo que juega como visitante.
- avg_fg3_last10_home_team: Calcula la media del porcentaje de acierto en tiros de tres puntos de los últimos 10 partidos para el equipo que juega en casa.
- avg_fg3_last10_visitor_team: Calcula la media del porcentaje de acierto en tiros de tres puntos de los últimos 10 partidos para el equipo que juega como visitante.

Se comienza cargando los datos de la carpeta “data”, del último archivo .csv que se generó anteriormente, “df_merged_renamed_dropped_sorted_2_rs.csv”:

In [4]:								
Out[4]:								
	GAME_DATE	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	HOME_TEAM_ABBREVIATION	HOME_TEAM_NICKNAME	VISITOR_TEAM_ABBREVIATION	VIS
0	2003-10-28	20300001	1610612755	1610612748	PHI	76ers	MIA	
1	2003-10-28	20300002	1610612759	1610612756	SAS	Spurs	PHX	
2	2003-10-28	20300003	1610612747	1610612742	LAL	Lakers	DAL	
3	2003-10-29	20300004	1610612738	1610612748	BOS	Celtics	MIA	
4	2003-10-29	20300005	1610612752	1610612753	NYK	Knicks	ORL	
...
19393	2019-04-10	21801226	1610612749	1610612760	MIL	Bucks	OKC	
19394	2019-04-10	21801227	1610612759	1610612742	SAS	Spurs	DAL	
19395	2019-04-10	21801228	1610612743	1610612750	DEN	Nuggets	MIN	
19396	2019-04-10	21801229	1610612746	1610612762	LAC	Clippers	UTA	
19397	2019-04-10	21801230	1610612757	1610612758	POR	Trail Blazers	SAC	

19398 rows × 23 columns

Una vez se han cargado los datos, se quiere generar una lista de listas. Cada lista corresponde a una temporada, y contienen los “GAME_ID” de cada partido de esa temporada, de manera que se tenga una lista con todos los “GAME_ID” de todos los partidos, pero dividido por temporada.

```
In [5]: df_rs_03 = df.query("GAME_DATE >= '2003-10-28' and GAME_DATE <='2004-04-14'")  
df_rs_04 = df.query("GAME_DATE >= '2004-11-02' and GAME_DATE <='2005-04-20'")  
df_rs_05 = df.query("GAME_DATE >= '2005-11-01' and GAME_DATE <='2006-04-19'")  
df_rs_06 = df.query("GAME_DATE >= '2006-10-31' and GAME_DATE <='2007-04-18'")  
df_rs_07 = df.query("GAME_DATE >= '2007-10-30' and GAME_DATE <='2008-04-16'")  
df_rs_08 = df.query("GAME_DATE >= '2008-10-28' and GAME_DATE <='2009-04-16'")  
df_rs_09 = df.query("GAME_DATE >= '2009-10-27' and GAME_DATE <='2010-04-14'")  
df_rs_10 = df.query("GAME_DATE >= '2010-10-26' and GAME_DATE <='2011-04-13'")  
df_rs_11 = df.query("GAME_DATE >= '2011-12-25' and GAME_DATE <='2012-04-26'")  
df_rs_12 = df.query("GAME_DATE >= '2012-10-30' and GAME_DATE <='2013-04-17'")  
df_rs_13 = df.query("GAME_DATE >= '2013-10-29' and GAME_DATE <='2014-04-16'")  
df_rs_14 = df.query("GAME_DATE >= '2014-10-28' and GAME_DATE <='2015-04-15'")  
df_rs_15 = df.query("GAME_DATE >= '2015-10-27' and GAME_DATE <='2016-04-13'")  
df_rs_16 = df.query("GAME_DATE >= '2016-10-25' and GAME_DATE <='2017-04-12'")  
df_rs_17 = df.query("GAME_DATE >= '2017-10-17' and GAME_DATE <='2018-04-11'")  
df_rs_18 = df.query("GAME_DATE >= '2018-10-16' and GAME_DATE <='2019-04-10'")  
  
In [6]: game_ids_03 = list(df_rs_03['GAME_ID'].values)  
game_ids_04 = list(df_rs_04['GAME_ID'].values)  
game_ids_05 = list(df_rs_05['GAME_ID'].values)  
game_ids_06 = list(df_rs_06['GAME_ID'].values)  
game_ids_07 = list(df_rs_07['GAME_ID'].values)  
game_ids_08 = list(df_rs_08['GAME_ID'].values)  
game_ids_09 = list(df_rs_09['GAME_ID'].values)  
game_ids_10 = list(df_rs_10['GAME_ID'].values)  
game_ids_11 = list(df_rs_11['GAME_ID'].values)  
game_ids_12 = list(df_rs_12['GAME_ID'].values)  
game_ids_13 = list(df_rs_13['GAME_ID'].values)  
game_ids_14 = list(df_rs_14['GAME_ID'].values)  
game_ids_15 = list(df_rs_15['GAME_ID'].values)  
game_ids_16 = list(df_rs_16['GAME_ID'].values)  
game_ids_17 = list(df_rs_17['GAME_ID'].values)  
game_ids_18 = list(df_rs_18['GAME_ID'].values)  
  
game_ids_all = [game_ids_03, game_ids_04, game_ids_05, game_ids_06, game_ids_07, game_ids_08, game_ids_09, \  
                 game_ids_10, game_ids_11, game_ids_12, game_ids_13, game_ids_14, game_ids_15, game_ids_16, \  
                 game_ids_17, game_ids_18, ]
```

Se importa el módulo “create_data”.

```
In [7]: #2) Funciones -> Modulo create_data.py  
import create_data
```

Se crean tantas listas como nuevas columnas se van a generar. En estas listas se guardarán los datos correspondientes a cada una de las columnas nuevas. Se crean con el mismo nombre que va a tener cada columna.

```
In [20]: %time  
  
#3) Creamos las listas  
porcentajes_victorias_team_home_rs_df_total = []  
porcentajes_victorias_team_visitor_rs_df_total = []  
porcentajes_victorias_last5_team_home_rs_df_total = []  
porcentajes_victorias_last5_team_visitor_rs_df_total = []  
porcentajes_victorias_last10_team_home_rs_df_total = []  
porcentajes_victorias_last10_team_visitor_rs_df_total = []  
avg_points_team_home_rs_df_total = []  
avg_points_team_visitor_rs_df_total = []  
avg_points_last5_team_home_rs_df_total = []  
avg_points_last5_team_visitor_rs_df_total = []  
avg_points_last10_team_home_rs_df_total = []  
avg_points_last10_team_visitor_rs_df_total = []  
avg_fg_team_home_rs_df_total = []  
avg_fg_team_visitor_rs_df_total = []  
avg_fg5_team_home_rs_df_total = []  
avg_fg5_team_visitor_rs_df_total = []  
avg_fg10_team_home_rs_df_total = []  
avg_fg10_team_visitor_rs_df_total = []  
avg_fg3_team_home_rs_df_total = []  
avg_fg3_team_visitor_rs_df_total = []  
avg_fg35_team_home_rs_df_total = []  
avg_fg35_team_visitor_rs_df_total = []  
avg_fg310_team_home_rs_df_total = []  
avg_fg310_team_visitor_rs_df_total = []
```

En la misma celda se hace un bucle que recorre la lista de listas de “GAME_ID”, y de la misma forma que se hizo anteriormente con la columna “VISITOR_TEAMS_WINS”, con un map y guardando los datos en una lista, se generan todos los nuevos datos que a continuación se añadirán al dataframe.

```

for game_ids in game_ids_all:
    porcentajes_victorias_team_home_rs_df = list(map(create_data.porcentaje_victorias_home_team, game_ids))
    porcentajes_victorias_team_home_rs_df_total.extend(porcentajes_victorias_team_home_rs_df)

    porcentajes_victorias_team_visitor_rs_df = list(map(create_data.porcentaje_victorias_visitor_team, game_ids))
    porcentajes_victorias_team_visitor_rs_df_total.extend(porcentajes_victorias_team_visitor_rs_df)

    porcentajes_victorias_last5_team_home_rs_df = list(map(create_data.porcentaje_victorias_last5_home_team, game_ids))
    porcentajes_victorias_last5_team_home_rs_df_total.extend(porcentajes_victorias_last5_team_home_rs_df)

    porcentajes_victorias_last5_team_visitor_rs_df = list(map(create_data.porcentaje_victorias_last5_visitor_team, game_ids))
    porcentajes_victorias_last5_team_visitor_rs_df_total.extend(porcentajes_victorias_last5_team_visitor_rs_df)

    porcentajes_victorias_last10_team_home_rs_df = list(map(create_data.porcentaje_victorias_last10_home_team, game_ids))
    porcentajes_victorias_last10_team_home_rs_df_total.extend(porcentajes_victorias_last10_team_home_rs_df)

    porcentajes_victorias_last10_team_visitor_rs_df = list(map(create_data.porcentaje_victorias_last10_visitor_team, game_ids))
    porcentajes_victorias_last10_team_visitor_rs_df_total.extend(porcentajes_victorias_last10_team_visitor_rs_df)

    avg_points_team_home_rs_df = list(map(create_data.avg_points_home_team, game_ids))
    avg_points_team_home_rs_df_total.extend(avg_points_team_home_rs_df)

    avg_points_team_visitor_rs_df = list(map(create_data.avg_points_visitor_team, game_ids))
    avg_points_team_visitor_rs_df_total.extend(avg_points_team_visitor_rs_df)

    avg_points_last5_team_home_rs_df = list(map(create_data.avg_points_last5_home_team, game_ids))
    avg_points_last5_team_home_rs_df_total.extend(avg_points_last5_team_home_rs_df)

    avg_points_last5_team_visitor_rs_df = list(map(create_data.avg_points_last5_visitor_team, game_ids))
    avg_points_last5_team_visitor_rs_df_total.extend(avg_points_last5_team_visitor_rs_df)

    avg_points_last10_team_home_rs_df = list(map(create_data.avg_points_last10_home_team, game_ids))
    avg_points_last10_team_home_rs_df_total.extend(avg_points_last10_team_home_rs_df)

    avg_points_last10_team_visitor_rs_df = list(map(create_data.avg_points_last10_visitor_team, game_ids))
    avg_points_last10_team_visitor_rs_df_total.extend(avg_points_last10_team_visitor_rs_df)

    avg_fg_team_home_rs_df = list(map(create_data.avg_fg_home_team, game_ids))
    avg_fg_team_home_rs_df_total.extend(avg_fg_team_home_rs_df)

    avg_fg_team_visitor_rs_df = list(map(create_data.avg_fg_visitor_team, game_ids))
    avg_fg_team_visitor_rs_df_total.extend(avg_fg_team_visitor_rs_df)

```

```

avg_fg5_team_home_rs_df = list(map(create_data.avg_fg_last5_home_team, game_ids))
avg_fg5_team_home_rs_df_total.extend(avg_fg5_team_home_rs_df)

avg_fg5_team_visitor_rs_df = list(map(create_data.avg_fg_last5_visitor_team, game_ids))
avg_fg5_team_visitor_rs_df_total.extend(avg_fg5_team_visitor_rs_df)

avg_fg10_team_home_rs_df = list(map(create_data.avg_fg_last10_home_team, game_ids))
avg_fg10_team_home_rs_df_total.extend(avg_fg10_team_home_rs_df)

avg_fg10_team_visitor_rs_df = list(map(create_data.avg_fg_last10_visitor_team, game_ids))
avg_fg10_team_visitor_rs_df_total.extend(avg_fg10_team_visitor_rs_df)

avg_fg3_team_home_rs_df = list(map(create_data.avg_fg3_home_team, game_ids))
avg_fg3_team_home_rs_df_total.extend(avg_fg3_team_home_rs_df)

avg_fg3_team_visitor_rs_df = list(map(create_data.avg_fg3_visitor_team, game_ids))
avg_fg3_team_visitor_rs_df_total.extend(avg_fg3_team_visitor_rs_df)

avg_fg35_team_home_rs_df = list(map(create_data.avg_fg3_last5_home_team, game_ids))
avg_fg35_team_home_rs_df_total.extend(avg_fg35_team_home_rs_df)

avg_fg35_team_visitor_rs_df = list(map(create_data.avg_fg3_last5_visitor_team, game_ids))
avg_fg35_team_visitor_rs_df_total.extend(avg_fg35_team_visitor_rs_df)

avg_fg310_team_home_rs_df = list(map(create_data.avg_fg3_last10_home_team, game_ids))
avg_fg310_team_home_rs_df_total.extend(avg_fg310_team_home_rs_df)

avg_fg310_team_visitor_rs_df = list(map(create_data.avg_fg3_last10_visitor_team, game_ids))
avg_fg310_team_visitor_rs_df_total.extend(avg_fg310_team_visitor_rs_df)

```

Esta acción puede llevar un tiempo largo de ejecución.

```

CPU times: user 6h 39min 37s, sys: 14min 22s, total: 6h 54min
Wall time: 6h 54min 29s
Parser   : 127 ms

```

Una vez se han generado los nuevos datos y están guardados en listas, se procede a crear las columnas y a llenarlas.

```
In [21]: #1) Añadimos todas las columnas
df['PERCENT_VIC_UNITL_THIS_GAME_HOME_TEAM'] = np.nan
df['PERCENT_VIC_UNITL_THIS_GAME_VISITOR_TEAM'] = np.nan
df['PERCENT_VIC_LAST5_GAMES_HOME_TEAM'] = np.nan
df['PERCENT_VIC_LAST5_GAMES_VISITOR_TEAM'] = np.nan
df['PERCENT_VIC_LAST10_GAMES_HOME_TEAM'] = np.nan
df['PERCENT_VIC_LAST10_GAMES_VISITOR_TEAM'] = np.nan
df['AVG_POINTS_UNTIL_THIS_GAME_HOME_TEAM'] = np.nan
df['AVG_POINTS_UNTIL_THIS_GAME_VISITOR_TEAM'] = np.nan
df['AVG_POINTS_LAST5_HOME_TEAM'] = np.nan
df['AVG_POINTS_LAST5_VISITOR_TEAM'] = np.nan
df['AVG_POINTS_LAST10_HOME_TEAM'] = np.nan
df['AVG_POINTS_LAST10_VISITOR_TEAM'] = np.nan
df['AVG_FGPERCENT_UNTIL_THIS_GAME_HOME_TEAM'] = np.nan
df['AVG_FGPERCENT_UNTIL_THIS_GAME_VISITOR_TEAM'] = np.nan
df['AVG_FGPERCENT_LAST5_HOME_TEAM'] = np.nan
df['AVG_FGPERCENT_LAST5_VISITOR_TEAM'] = np.nan
df['AVG_FGPERCENT_LAST10_HOME_TEAM'] = np.nan
df['AVG_FGPERCENT_LAST10_VISITOR_TEAM'] = np.nan
df['AVG_FG3PCT_UNTIL_THIS_GAME_HOME_TEAM'] = np.nan
df['AVG_FG3PCT_UNTIL_THIS_GAME_VISITOR_TEAM'] = np.nan
df['AVG_FG3PCT_LAST5_HOME_TEAM'] = np.nan
df['AVG_FG3PCT_LAST5_VISITOR_TEAM'] = np.nan
df['AVG_FG3PCT_LAST10_HOME_TEAM'] = np.nan
df['AVG_FG3PCT_LAST10_VISITOR_TEAM'] = np.nan
```

```
In [22]: #4) Introducimos los datos
df['PERCENT_VIC_UNITL_THIS_GAME_HOME_TEAM'] = pd.Series(porcentajes_victorias_team_home_rs_df_total)
df['PERCENT_VIC_UNITL_THIS_GAME_VISITOR_TEAM'] = pd.Series(porcentajes_victorias_team_visitor_rs_df_total)
df['PERCENT_VIC_LAST5_GAMES_HOME_TEAM'] = pd.Series(porcentajes_victorias_last5_team_home_rs_df_total)
df['PERCENT_VIC_LAST5_GAMES_VISITOR_TEAM'] = pd.Series(porcentajes_victorias_last5_team_visitor_rs_df_total)
df['PERCENT_VIC_LAST10_GAMES_HOME_TEAM'] = pd.Series(porcentajes_victorias_last10_team_home_rs_df_total)
df['PERCENT_VIC_LAST10_GAMES_VISITOR_TEAM'] = pd.Series(porcentajes_victorias_last10_team_visitor_rs_df_total)
df['AVG_POINTS_UNTIL_THIS_GAME_HOME_TEAM'] = pd.Series(avg_points_team_home_rs_df_total)
df['AVG_POINTS_UNTIL_THIS_GAME_VISITOR_TEAM'] = pd.Series(avg_points_team_visitor_rs_df_total)
df['AVG_POINTS_LAST5_HOME_TEAM'] = pd.Series(avg_points_last5_team_home_rs_df_total)
df['AVG_POINTS_LAST5_VISITOR_TEAM'] = pd.Series(avg_points_last5_team_visitor_rs_df_total)
df['AVG_POINTS_LAST10_HOME_TEAM'] = pd.Series(avg_points_last10_team_home_rs_df_total)
df['AVG_POINTS_LAST10_VISITOR_TEAM'] = pd.Series(avg_points_last10_team_visitor_rs_df_total)
df['AVG_FGPERCENT_UNTIL_THIS_GAME_HOME_TEAM'] = pd.Series(avg_fg_team_home_rs_df_total)
df['AVG_FGPERCENT_UNTIL_THIS_GAME_VISITOR_TEAM'] = pd.Series(avg_fg_team_visitor_rs_df_total)
df['AVG_FGPERCENT_LAST5_HOME_TEAM'] = pd.Series(avg_fg5_team_home_rs_df_total)
df['AVG_FGPERCENT_LAST5_VISITOR_TEAM'] = pd.Series(avg_fg5_team_visitor_rs_df_total)
df['AVG_FGPERCENT_LAST10_HOME_TEAM'] = pd.Series(avg_fg10_team_home_rs_df_total)
df['AVG_FGPERCENT_LAST10_VISITOR_TEAM'] = pd.Series(avg_fg10_team_visitor_rs_df_total)
df['AVG_FG3PCT'] = pd.Series(avg_fg3_team_home_rs_df_total)
df['AVG_FG3PCT_UNTIL_THIS_GAME_HOME_TEAM'] = pd.Series(avg_fg3_team_visitor_rs_df_total)
df['AVG_FG3PCT_LAST5_HOME_TEAM'] = pd.Series(avg_fg35_team_home_rs_df_total)
df['AVG_FG3PCT_LAST5_VISITOR_TEAM'] = pd.Series(avg_fg35_team_visitor_rs_df_total)
df['AVG_FG3PCT_LAST10_HOME_TEAM'] = pd.Series(avg_fg310_team_home_rs_df_total)
df['AVG_FG3PCT_LAST10_VISITOR_TEAM'] = pd.Series(avg_fg310_team_visitor_rs_df_total)
```

Se reordenan por última vez las columnas.

```
In [25]: df = df[['GAME_ID', 'HOME_TEAM_ID', 'VISITOR_TEAM_ID', 'HOME_TEAM_ABBREVIATION', 'HOME_TEAM_NICKNAME', \
'VISITOR_TEAM_ABBREVIATION', 'VISITOR_TEAM_NICKNAME', 'SEASON', 'GAME_DATE', \
'FG_PCT_home', 'FT_PCT_home', 'FG3_PCT_home', 'PERCENT_VIC_UNITL_THIS_GAME_HOME_TEAM', \
'PERCENT_VIC_LAST5_GAMES_HOME_TEAM', 'PERCENT_VIC_LAST10_GAMES_HOME_TEAM', \
'AVG_POINTS_UNTIL_THIS_GAME_HOME_TEAM', 'AVG_POINTS_LAST5_HOME_TEAM', 'AVG_POINTS_LAST10_HOME_TEAM', \
'AVG_FGPERCENT_UNTIL_THIS_GAME_HOME_TEAM', 'AVG_FGPERCENT_LAST5_HOME_TEAM', \
'AVG_FGPERCENT_LAST10_HOME_TEAM', 'AVG_FGPERCENT_LAST10_HOME_TEAM', 'AVG_FG3PCT_UNTIL_THIS_GAME_HOME_TEAM', \
'AVG_FG3PCT_LAST5_HOME_TEAM', 'AVG_FG3PCT_LAST10_HOME_TEAM', 'AST_home', 'REB_home', \
'FG_PCT_away', 'FT_PCT_away', 'FG3_PCT_away', 'PERCENT_VIC_UNITL_THIS_GAME_VISITOR_TEAM', \
'PERCENT_VIC_LAST5_GAMES_VISITOR_TEAM', 'PERCENT_VIC_LAST10_GAMES_VISITOR_TEAM', \
'AVG_POINTS_UNTIL_THIS_GAME_VISITOR_TEAM', 'AVG_POINTS_LAST5_VISITOR_TEAM', 'AVG_POINTS_LAST10_VISITOR_TEAM', \
'AVG_FGPERCENT_UNTIL_THIS_GAME_VISITOR_TEAM', 'AVG_FGPERCENT_LAST5_VISITOR_TEAM', \
'AVG_FGPERCENT_LAST10_VISITOR_TEAM', 'AVG_FGPERCENT_LAST10_VISITOR_TEAM', \
'AVG_FG3PCT_UNTIL_THIS_GAME_VISITOR_TEAM', 'AVG_FG3PCT_LAST5_VISITOR_TEAM', 'AVG_FG3PCT_LAST10_VISITOR_TEAM', \
'AST_away', 'REB_away', 'HOME_TEAM_WINS', 'VISITOR_TEAM_WINS', 'PTS_home', 'PTS_away']]
```

Finalmente se muestra por pantalla el dataframe definitivo con el que se trabajará, y se guarda para su posterior uso.

```
In [26]: pd.options.display.max_columns = None
df
```

	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	HOME_TEAM_ABBREVIATION	HOME_TEAM_NICKNAME	VISITOR_TEAM_ABBREVIATION	VISITOR_TEAM_NICKNAME
0	20300001	1610612755	1610612748	PHI	76ers	MIA	
1	20300002	1610612759	1610612756	SAS	Spurs	PHX	
2	20300003	1610612747	1610612742	LAL	Lakers	DAL	
3	20300004	1610612738	1610612748	BOS	Celtics	MIA	
4	20300005	1610612752	1610612753	NYK	Knicks	ORL	
...
19393	21801226	1610612749	1610612760	MIL	Bucks	OKC	
19394	21801227	1610612759	1610612742	SAS	Spurs	DAL	
19395	21801228	1610612743	1610612750	DEN	Nuggets	MIN	Tin
19396	21801229	1610612746	1610612762	LAC	Clippers	UTA	
19397	21801230	1610612757	1610612758	POR	Trail Blazers	SAC	

19398 rows × 9 columns

```
In [27]: df.to_csv('df_complete.csv')
```

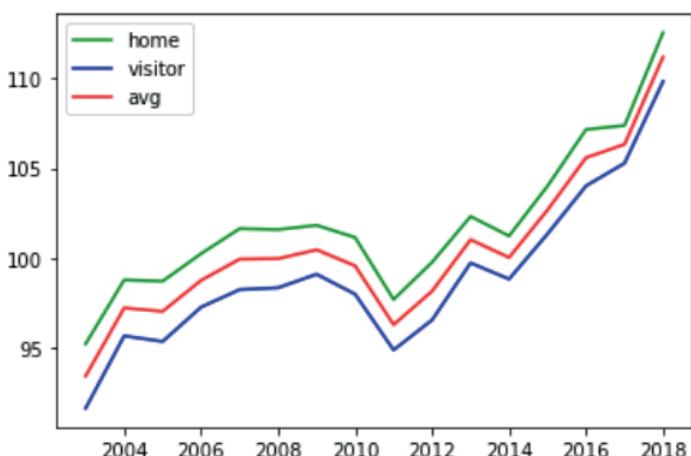
5.3 EXPLORACION DE LOS DATOS

Una vez se tienen los datos definitivos con los que se va a trabajar, se realiza una exploración de los mismos. Al igual que al principio se han explorado los datos con el objetivo de encontrar huecos vacíos o irregularidades en el conjunto de datos, ahora se pretende navegar en ellos para tener una mejor comprensión sobre los registros con los que se va a trabajar.

Lo primero que se desea conocer es cómo han variado los datos de las principales y más importantes estadísticas a lo largo de los últimos años. Para ello se van a mostrar y se comentan a continuación unos gráficos que nos permitan saber cómo ha sido esta evolución.

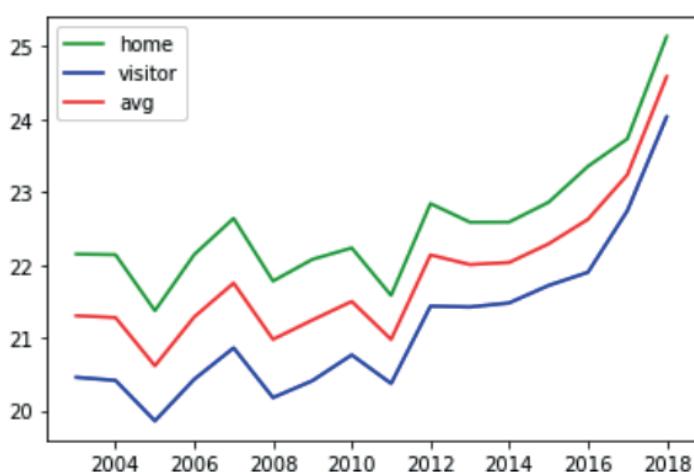
5.3.1 EVOLUCION

- Puntos:



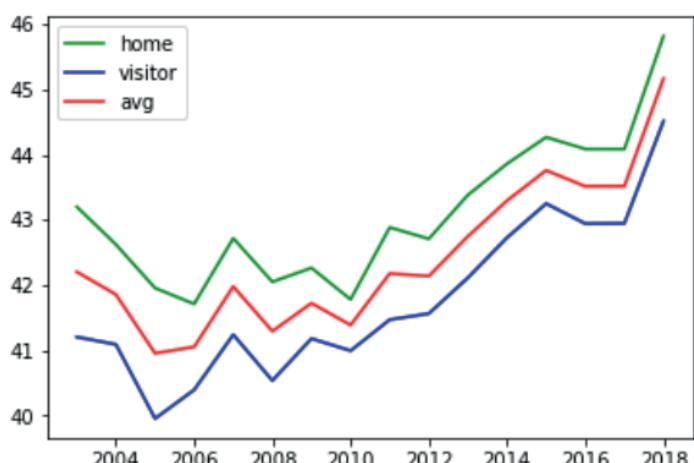
Lo primero que se aprecia al ver este gráfico es que en los últimos 15 años han ido aumentando la cantidad de puntos que se anotan, se ha pasado de una media de casi 95 puntos por partido a más de 110, esto indica que cada vez es más difícil ganar un partido, ya que se debe aumentar la anotación para poder ganarlo. También se aprecia cómo el equipo que juega en casa tiende a anotar más que el que juega como visitante, es decir, jugar en casa es una ventaja y en ocasiones puede ser determinante. La competitividad entre el equipo local y el visitante ha aumentado, la diferencia de puntos entre ambos va disminuyendo con los años, esto se aprecia atendiendo a la distancia que hay entre la línea que marca el equipo local (verde) y la línea que marca el equipo visitante (azul).

- Asistencias:



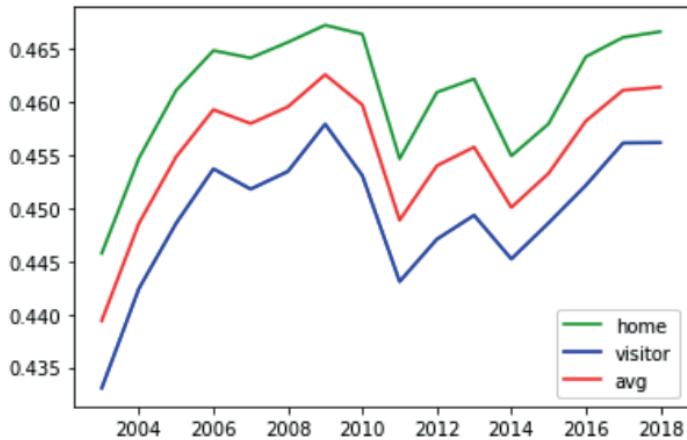
El número de asistencias por partido también ha aumentado, aunque en este caso su evolución ha sido más leve, se ha pasado de una media de 21 asistencias por partido a casi 25. Está totalmente relacionado con el gráfico de los puntos, si se reparten más asistencias también es normal que aumenten los puntos anotados. Al igual que antes, se observa una vez más como jugar como local puede ser una ventaja.

- Rebotes:



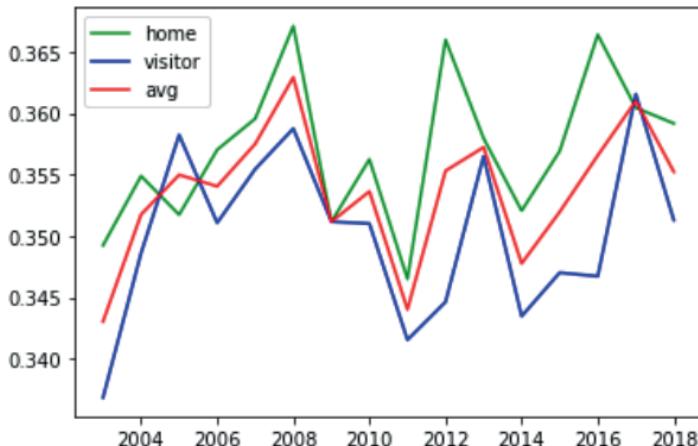
El número de rebotes por partido apenas ha aumentado, de una media de 42 se ha pasado a 45. Una vez más jugar como local tiene ventaja.

- %FG:



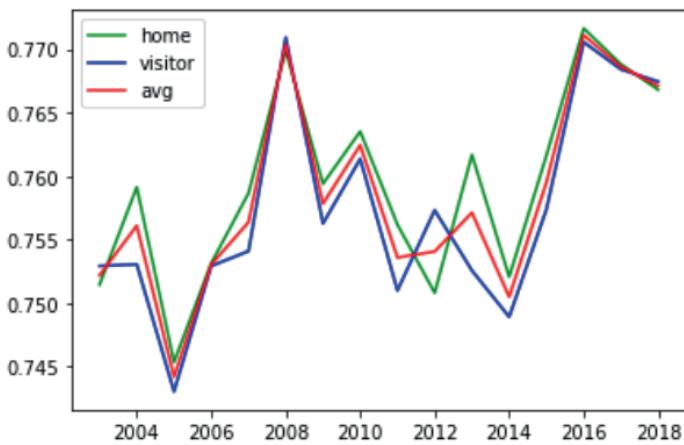
El porcentaje de tiros de 2 prácticamente no ha cambiado, aunque ha variado ligeramente con pequeñas subidas y bajadas no es especialmente relevante. El equipo local vuelve a tener “ventaja” ya que tiende a tener mayor acierto que el equipo visitante.

- %FG3:



Este gráfico nos muestra la evolución en el porcentaje de tiros de 3. La relevancia que puede tener no es significativa.

- %FT:

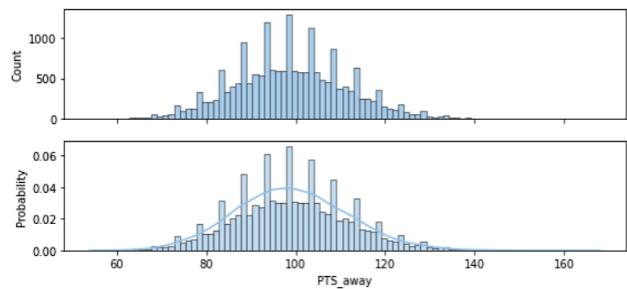
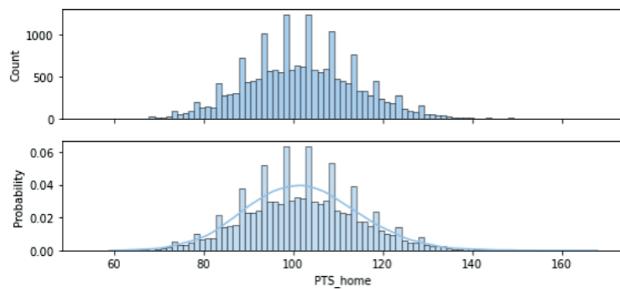


Este evolutivo muestra la evolución en el porcentaje de acierto para los tiros libres. Se observa como aquí no existe una clara ventaja para el equipo que juega como local, aunque si está ligeramente por encima del equipo visitante.

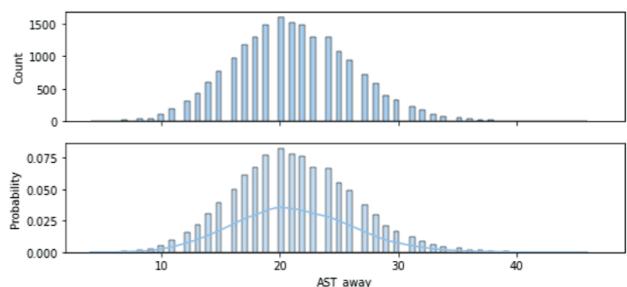
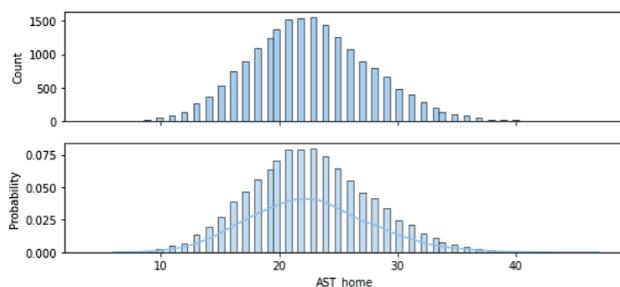
5.3.2 DISTRIBUCION DE LOS DATOS

Este apartado se centra fundamentalmente en conocer cómo es la distribución de las estadísticas más importantes.

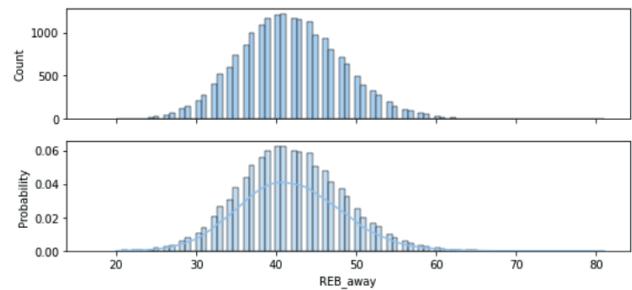
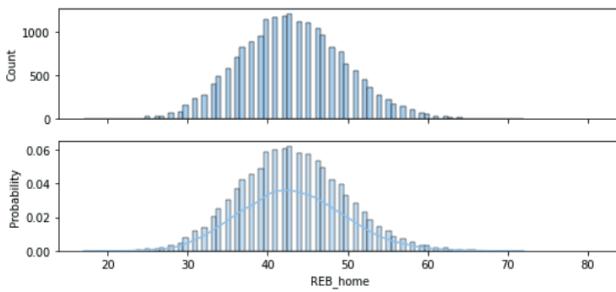
- Puntos:



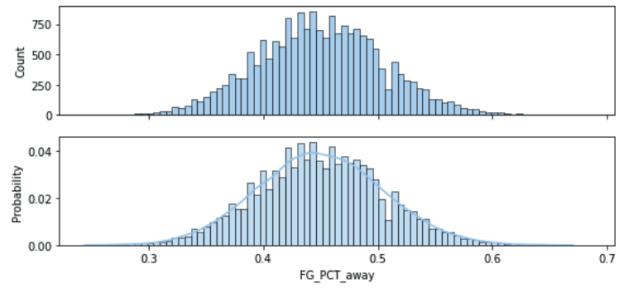
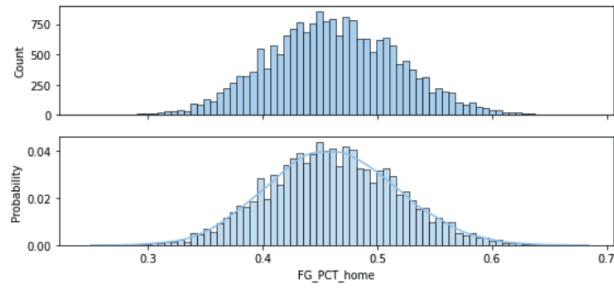
- Asistencias:



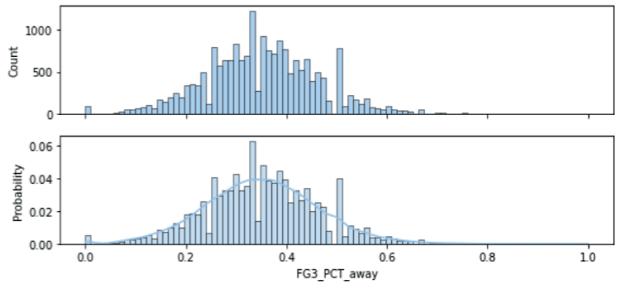
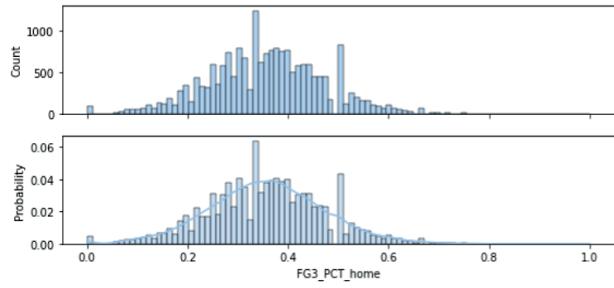
● Rebotes:



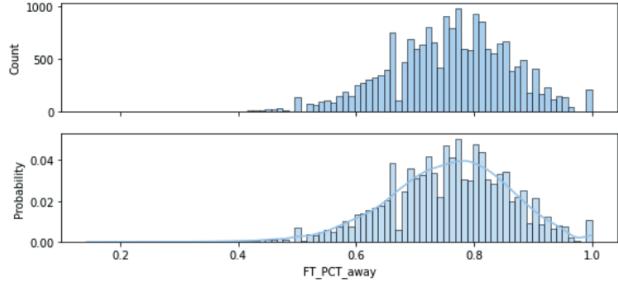
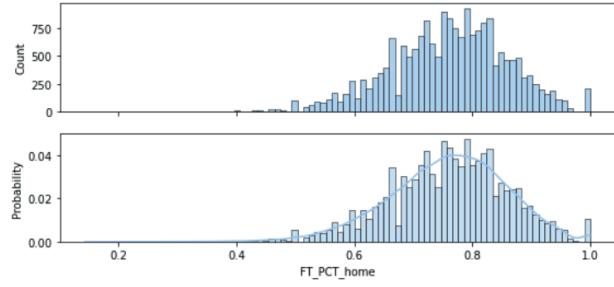
● % FG:



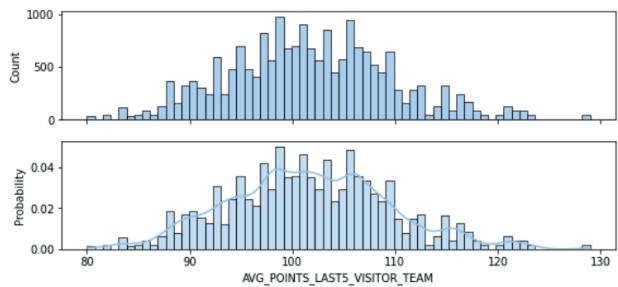
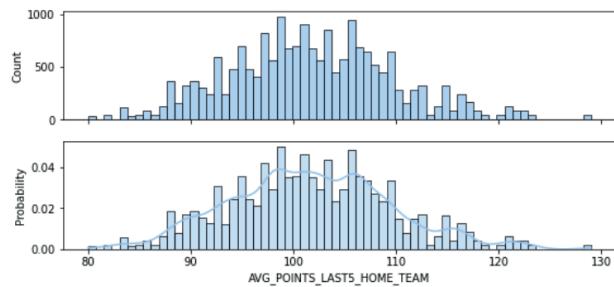
● % FG3:



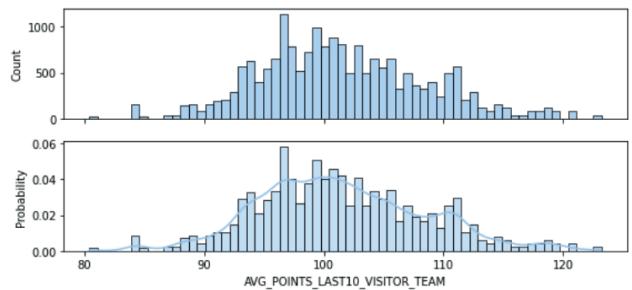
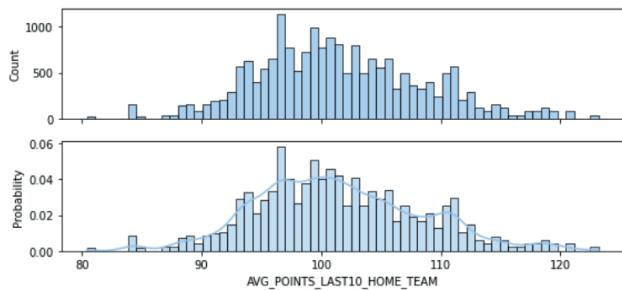
● % FT:



● Media puntos últimos 5 partidos:



- Media puntos últimos 10 partidos:



Como se aprecia todas las variables siguen distribuciones normales. Aunque todas siguen distribuciones normales bastante centradas, se observa que algunas variables tienen un desplazamiento.

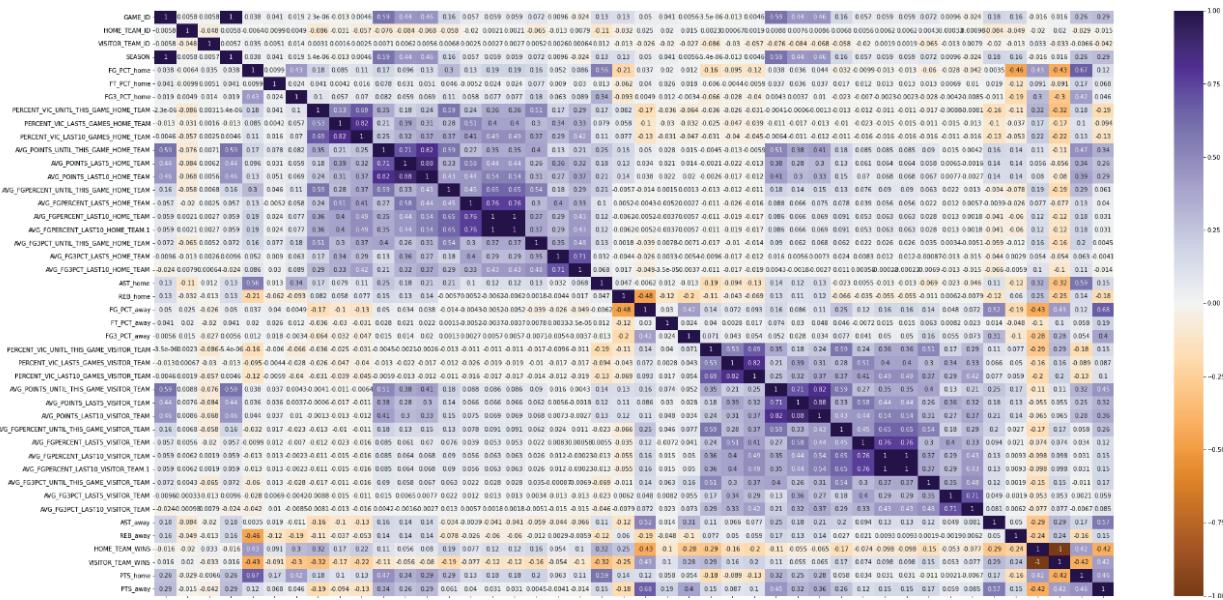
El %FG3 (%acerto en triple) está desplazado hacia la izquierda, es decir hacia cero, esto indica la complejidad de anotar un tiro de 3 en un partido de la NBA, pocas veces el porcentaje supera 50%, que sería lo mismo que anotar la mitad de los tiros intentados, algo por lo que se ve muy difícil de conseguir.

El %FT (tiros libres) está muy desplazado a la derecha, es decir hacia 1, que sería el ideal supuesto en el que se encestarían todos los tiros libres. Esto se puede concluir que es algo normal, igual que se observaba que anotar un triple es algo complejo debido entre otras cuestiones a que el partido está en juego, para los tiros libres esto no ocurre. El tiro libre es siempre igual, no cambia, y además el partido está “parado” por lo que la concentración es máxima, es por eso que el porcentaje de acierto en tiros libres es tan elevado.

La media de anotación en los últimos 5 partidos y en los últimos 10 aunque también siguen distribuciones normales estas son más irregulares. Esto indica cómo pueden cambiar los resultados de un equipo en un corto periodo de partidos, debido al alto volumen de partidos, y a los constantes desplazamientos que sufren los equipos (por la lejanía que existe entre la mayoría de ciudades que tienen equipo en la liga).

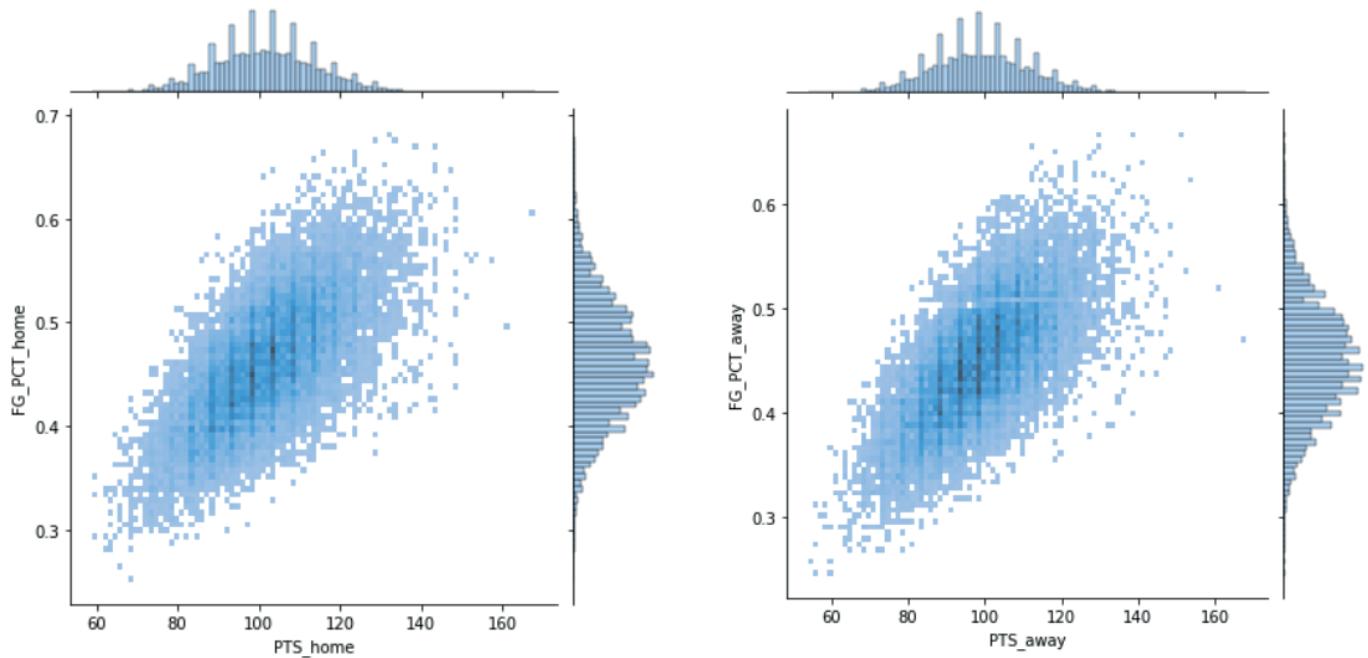
5.3.3 CORRELACION DE LOS DATOS

- Matriz de correlación:



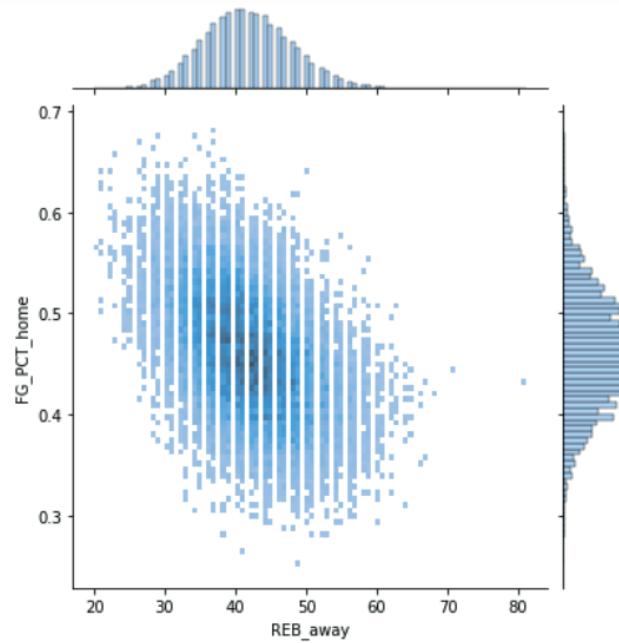
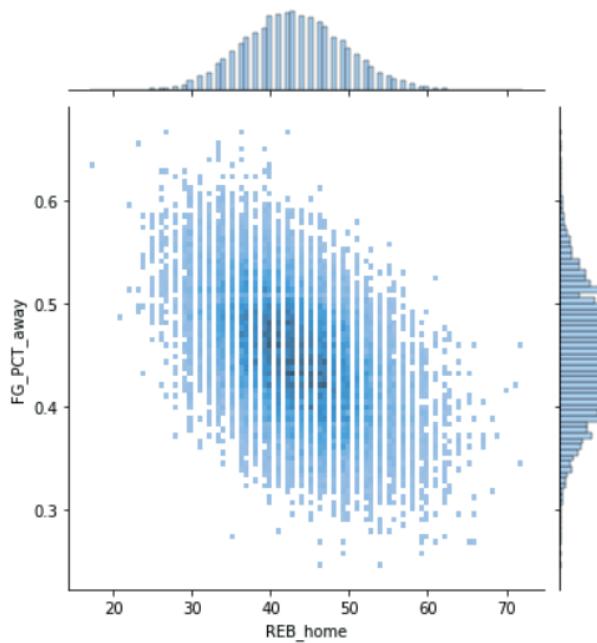
La matriz de correlación que se tiene es de gran tamaño debido a la multitud de variables de que se disponen. Atendiendo a los colores, las variables más correlacionadas son aquellas cuya tonalidad en el gráfico es menos blanca. A continuación se estudia más detalladamente la relación entre algunas de estas variables.

- % FG - PTS:



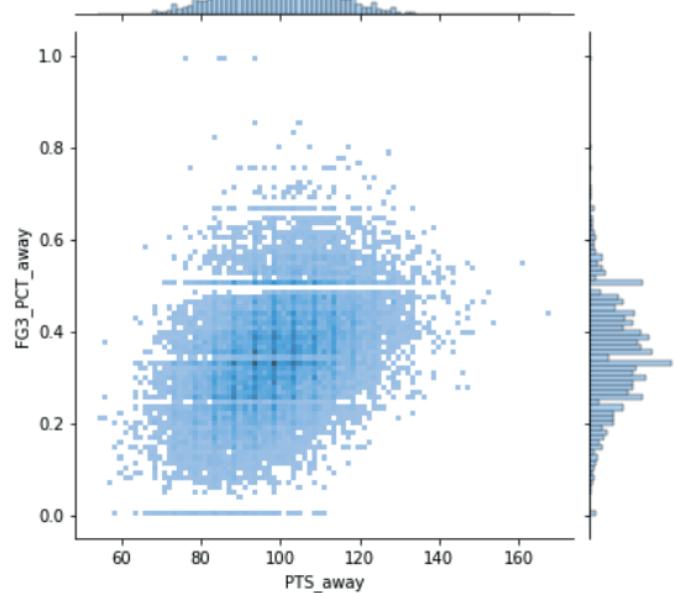
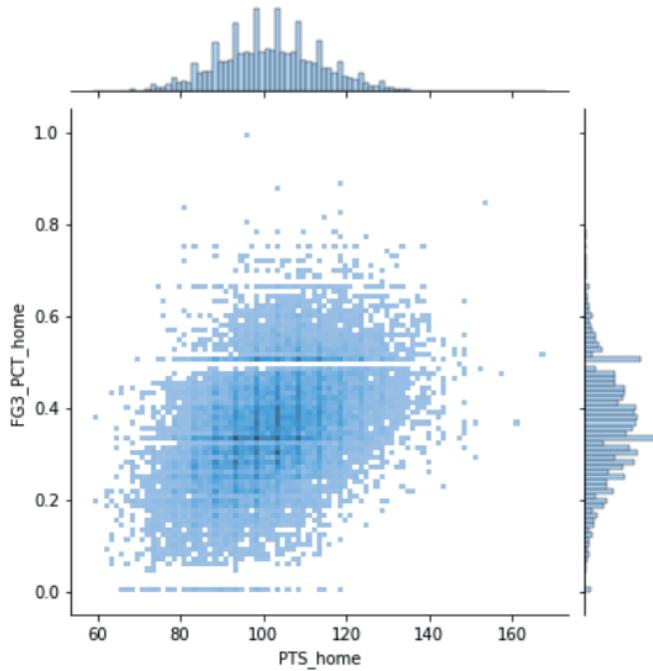
La conclusión aquí es evidente, cuanto mayor es el porcentaje de acierto en tiros de campo mayor es la puntuación obtenida en un partido. Esto ocurre tanto si el equipo es local como visitante.

● % FG - REB:

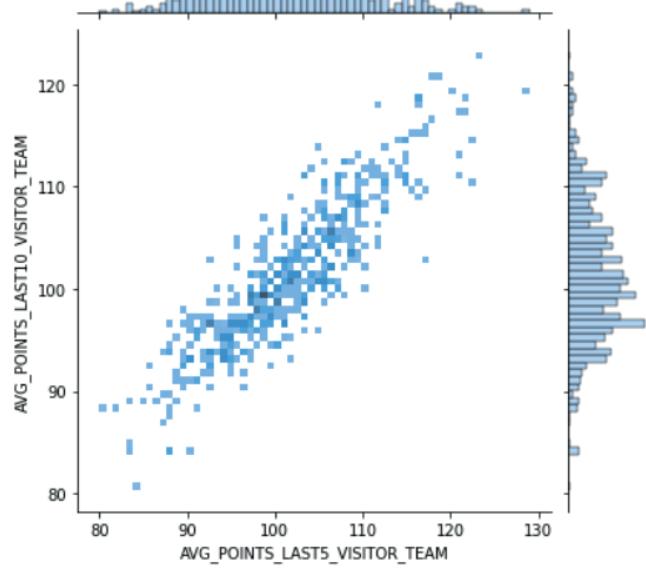
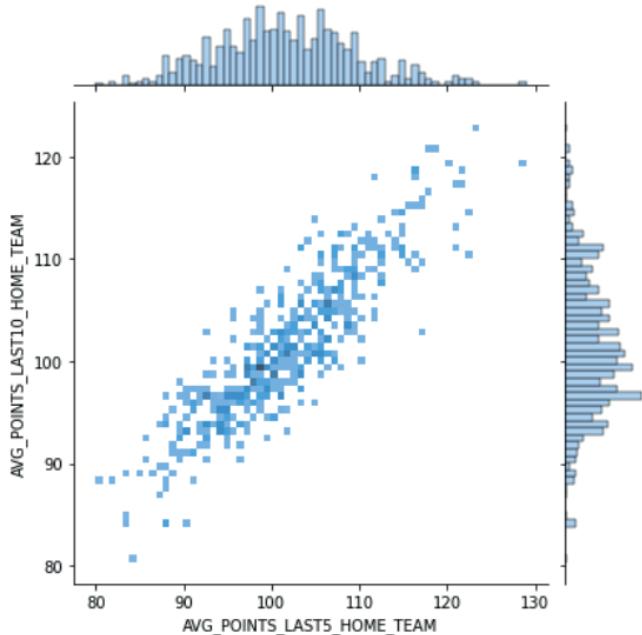


Aquí se observa como cuanto mayor es el porcentaje de acierto en tiros de campo menor es el número de rebotes del equipo rival, algo totalmente lógico ya que si se anota, se impide directamente capturar el rebote.

● % FG3 - PTS:

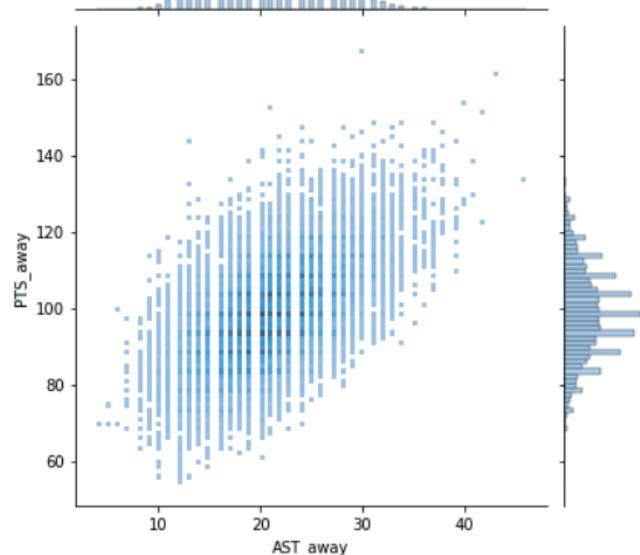
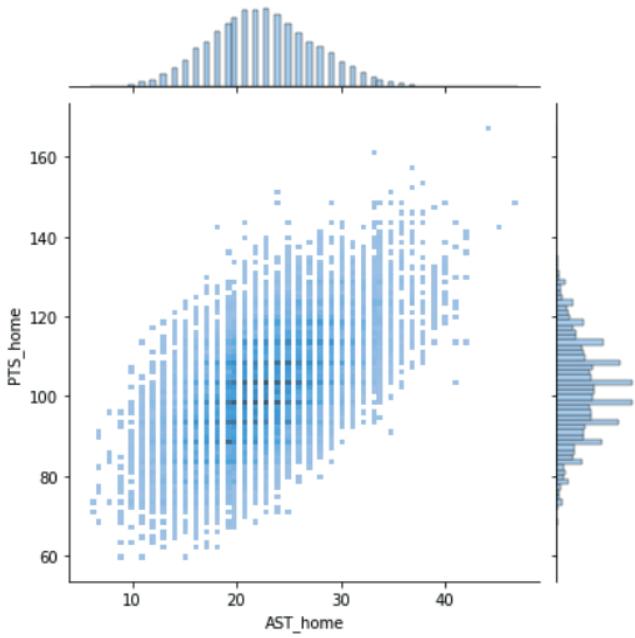


- Media de puntos últimos 5 partidos - Media de puntos últimos 10 partidos:



Poco se puede comentar más que la clara relación lineal que existe entre ambas variables.

- AST - PTS:



La relación que hay entre las asistencias y los puntos anotados es lógica, cuantos más pases de canasta se realicen mayor será la anotación del equipo. Por eso la figura del base en el equipo, y de jugadores que sepan distribuir bien el balón son tan importantes y demandados, por que en ocasiones aun con una anotación baja individualmente, pueden suponer la diferencia con un alto número de pases de canasta.

5.4 MODELADO: MODEL_CREATION.IPYNB

En este notebook se estudian diferentes opciones para generar un modelo que permita saber qué equipo ganará un partido basándose en las estadísticas estudiadas. Se utilizan varias métricas para medir el rendimiento del modelo generado, de manera que ayuden a determinar qué algoritmo se adapta mejor y qué parámetros maximizan su rendimiento, con el fin de obtener los mejores resultados posibles.

Para poder comparar los distintos modelos generados se utiliza:

- Función “evaluate_ROC”: que muestra la curva ROC:

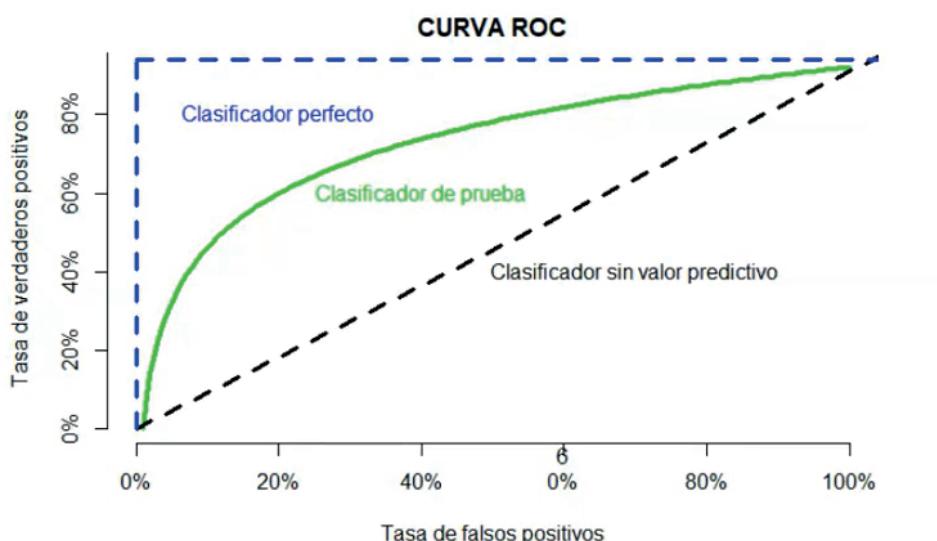
```
def evaluate_ROC(estimator, data, target):
    X_train, X_test, y_train, y_test = train_test_split(data,target)

    estimator.fit(X_train, y_train)
    y_hat = estimator.predict(X_test)
    probs = estimator.predict_proba(X_test)[:,1]

    # plotear curva roc
    fpr, tpr, thresholds = roc_curve(y_test, probs)
    f, ax = plt.subplots(1,1)
    f.set_size_inches(8,7)
    ax.plot(fpr,tpr)
    ax.plot([0,1], [0,1], c='grey')

    print(f'acc: {accuracy_score(y_test, y_hat):.3} \
recall: {recall_score(y_test, y_hat):.3} \
precision:{precision_score(y_test, y_hat):.3} \
f1:{f1_score(y_test, y_hat):.3} \
auc:{roc_auc_score(y_test, probs):.3}' )
```

Esto permite poder comparar de manera rápida a través de un gráfico, cómo de bueno es el modelo que se ha generado.



Un espacio ROC se define por FPR y VPR como ejes x e y respectivamente, y representa los intercambios entre verdaderos positivos y falsos positivos. Dado que VPR es equivalente a sensibilidad y FPR es igual a 1-especificidad. Cada resultado de predicción o instancia de la matriz de confusión representa un punto en el espacio ROC.

El resultado óptimo posible de predicción se sitúa en el punto de la esquina superior izquierda, ese punto representa un 100% de sensibilidad (ningún falso negativo) y un 100% también de especificidad (ningún falso positivo). Este punto representa la clasificación perfecta. Por el contrario, una clasificación totalmente aleatoria daría un punto a lo largo de la línea diagonal.

- La matriz de confusión:

Esta matriz pone en relación las predicciones realizadas por el algoritmo y los resultados correctos que debería haber mostrado. De la matriz se extraen 4 métricas que ayudan a medir cuán bueno es el modelo.

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) <i>Type II Error</i>	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) <i>Type I Error</i>	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
	Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$	

- El reporte de clasificación:

Sirve la métricas anteriores en formato tabla para facilitar su lectura.

	precision	recall	f1-score	support
0	0.83	0.79	0.81	1959
1	0.86	0.89	0.88	2891
accuracy			0.85	4850
macro avg	0.85	0.84	0.84	4850
weighted avg	0.85	0.85	0.85	4850

Lo primero es cargar las librerías que se van a utilizar durante el notebook:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

from sklearn.metrics import mean_squared_error, mean_absolute_error, roc_auc_score, roc_curve, f1_score, \
precision_score, recall_score, accuracy_score, precision_recall_curve, auc

from sklearn.model_selection import GridSearchCV, ParameterGrid

from sklearn.metrics import classification_report, confusion_matrix

import multiprocessing

from sklearn.model_selection import RepeatedKFold, RepeatedStratifiedKFold
```

Lo siguiente es cargar el dataframe generado y guardado anteriormente:

```
pd.options.display.max_columns = None
df = pd.read_csv('data/df_complete.csv')
df = df.drop('Unnamed: 0', axis=1)
df
```

	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	HOME_TEAM_ABBREVIATION	HOME_TEAM_NICKNAME	VISITOR_TEAM_ABBREVIATION	VISITOR_TEAM_NICKNAME
0	20300001	1610612755	1610612748	PHI	76ers	MIA	Heat
1	20300002	1610612759	1610612756	SAS	Spurs	PHX	Suns
2	20300003	1610612747	1610612742	LAL	Lakers	DAL	Cowboys
3	20300004	1610612738	1610612748	BOS	Celtics	MIA	Heat
4	20300005	1610612752	1610612753	NYK	Knicks	ORL	Pelicans
...
19393	21801226	1610612749	1610612760	MIL	Bucks	OKC	Rockets
19394	21801227	1610612759	1610612742	SAS	Spurs	DAL	Cowboys
19395	21801228	1610612743	1610612750	DEN	Nuggets	MIN	Timberwolves
19396	21801229	1610612746	1610612762	LAC	Clippers	UTA	Jazz
19397	21801230	1610612757	1610612758	POR	Trail Blazers	SAC	Bucks

19398 rows × 49 columns

Los algoritmos escogidos para el estudio son:

- Gaussian Naive Bayes (GNB)
- Logistic Regression
- Support Vector Classification (SVC)
- Decision Tree Classifier
- Random Forest Classifier

Para obtener los mejores hiperparámetros en cada algoritmo y maximizar su precisión se han utilizado GridSearchCV y RandomSearchCV.

La selección de datos empleada ha sido:

Para data:

- game_date,
- fg_pct_home, ft_pct_home, fg3_pct_home,
- fg_pct_away, ft_pct_away, fg3_pct_away,
- percent_vic_untl_this_game_home_team,
- percent_vic_untl_this_game_visitor_team,
- percent_vic_last5_games_home_team,
- percent_vic_last5_games_visitor_team,
- percent_vic_last10_games_home_team,
- percent_vic_last10_games_visitor_team,
- avg_points_until_this_game_home_team,
- avg_points_until_this_game_visitor_team,
- avg_points_last5_home_team,
- avg_points_last5_visitor_team,
- avg_points_last10_home_team,
- avg_points_last10_visitor_team,
- avg_fgpercent_until_this_game_home_team,
- avg_fgpercent_until_this_game_visitor_team,
- avg_fgpercent_last5_home_team,
- avg_fgpercent_last5_visitor_team,
- avg_fgpercent_last10_home_team,
- avg_fgpercent_last10_visitor_team,
- avg_fg3pct_until_this_game_home_team,
- avg_fg3pct_until_this_game_visitor_team,
- avg_fg3pct_last5_home_team,
- avg_fg3pct_last5_visitor_team,
- avg_fg3pct_last10_home_team,
- avg_fg3pct_last10_visitor_team,
- ast_home, reb_home, ast_away y reb_away.

Y como target (objetivo):

- home_team_wins

Para poder comparar más fácilmente los resultados de cada algoritmo y método, se crea un dataframe donde se añade dicha información.

```
#Dataframe de resultados de accuracy
columnas_opt = ['Tecnica','Gaussian','Logistic Regression','SVC','Decission Tree Classifier','Random Forest Classifier']
df_opt = pd.DataFrame(columns = columnas_opt)
df_opt.set_index("Tecnica",inplace=True)
df_opt.loc['Standard'] = [0, 0, 0, 0, 0]
df_opt.loc['GridSearch'] = [0, 0, 0, 0, 0]
df_opt.loc['RandomSearch'] = [0, 0, 0, 0, 0]
df_opt.head()

columnas_ps = ['Estimator', 'Params', 'Score']
df_params_scores = pd.DataFrame(columns = columnas_ps)

df_opt
```

	Gaussian	Logistic Regression	SVC	Decission Tree Classifier	Random Forest Classifier
Tecnica					
Standard	0	0	0	0	0
GridSearch	0	0	0	0	0
RandomSearch	0	0	0	0	0

A continuación se seleccionan las variables que se van a utilizar, se normaliza, se separan los datos en “train” y “test” y se “estandarizan”.

```
X = df.iloc[:,9:43].values
y = df.iloc[:,43].values
y=y.astype('int')

X_copy = X.copy()
X_norm = (X_copy - X_copy.mean()) / X_copy.std()

X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.25, random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Se empieza primero probando cada algoritmo con los hiperparámetros que tiene por defecto, con ánimo de poder ver cuánta mejora existe más tarde cuando se entre el modelo con unos hiperparametros mejor ajustados.

1. Default Parameters

```
gnb = GaussianNB()
logR = LogisticRegression()
svc = SVC()
dtc = DecisionTreeClassifier()
rfc = RandomForestClassifier()

models = [gnb,logR,svc,dtc,rfc]

col = 0
for model in models:
    model.fit(X_train,y_train)
    df_opt.iloc[0,col] = model.score(X_test,y_test)
    col += 1

df_opt.head()
```

	Gaussian	Logistic Regression	SVC	Decission Tree Classifier	Random Forest Classifier
Tecnica					
Standard	0.798557	0.862268	0.862887	0.763299	0.856701
GridSearch	0	0	0	0	0
RandomSearch	0	0	0	0	0

La siguiente prueba se realiza con un GridSearch que irá comprobando la mejor combinación de hiperparametros que es capaz de encontrar para los hiperparametros propuestos. Cada algoritmo tiene sus propios hiperparametros y dentro de cada uno no todos son compatibles con todos, ni todos tienen los mismo fines. En la selección que se ha realizado se ha escogido una combinación de hiperparametros que no incurra en error durante la ejecución. Esto es importante debido al largo tiempo de espera que puede llevar al ordenador realizar esta tarea, ya que deberá hacer una gran número de combinaciones con cada hiperparametro de cada uno de los algoritmos.

Aunque GridSearch puede ser de gran ayuda para determinar la mejor combinación, es necesario realizar un bucle que sirva al GridSearch de distintos modelos, de esta forma obtendremos los resultados de todos los Gridsearch sobretodos los algoritmos directamente. A continuación se guardan los resultados.

2. GridSearchCV

```
#Gaussian
var_smoothing = [0.00000001, 0.00000001, 0.000000001]
gnb_grid = dict(var_smoothing = var_smoothing)

#Logistic Regression
C = np.logspace(-4, 4, 50)
solver = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

logR_grid = dict(C = C,
                  solver = solver)

#SVC
C = np.logspace(-5, 7, 20)
kernel = ['poly', 'rbf', 'sigmoid']
gamma = ['scale', 'auto']

svc_grid = dict(C = C,
                 kernel = kernel,
                 gamma = gamma)

#Decission Tree Classifier
ccp_alpha = np.linspace(0, 5, 10)
criterion = ["gini", "entropy"]
max_features = ["auto", "sqrt", "log2"]

dtc_grid = dict(ccp_alpha = ccp_alpha,
                criterion = criterion,
                max_features = max_features,)

# Random Forest Classifier
n_estimators = [50, 100, 150, 200, 250, 300, 350, 400, 450, 500]
max_depth = [None, 3, 10, 20]
criterion = ['gini', 'entropy']
max_features = ["auto", "sqrt", "log2"]

rfc_grid = dict(n_estimators=n_estimators,
                max_depth=max_depth,
                criterion=criterion,
                max_features=max_features)
```

```

models = [gnb,logR,svc,dtc,rfc]
grids = [gnb_grid,logR_grid,svc_grid,dtc_grid,rfc_grid]
col = 0
fila_aux = 0

for ind in range(0,len(models)):
    cv = RepeatedStratifiedKFold(n_splits=10,
                                n_repeats=3,
                                random_state=1)

    grid_search = GridSearchCV(estimator=models[col],
                                param_grid=grids[col],
                                n_jobs=-1,
                                cv=cv,
                                scoring='accuracy',
                                error_score=0)

    grid_clf_acc = grid_search.fit(X_train, y_train)
    df_opt.iloc[1,col] = grid_clf_acc.score(X_test,y_test)

    df_params_scores.loc[fila_aux, 'Estimator'] = grid_clf_acc.estimator
    df_params_scores.loc[fila_aux, 'Params'] = str(grid_clf_acc.best_params_)
    df_params_scores.loc[fila_aux, 'Score'] = grid_clf_acc.best_score_

    col += 1
    fila_aux += 1

df_opt.head()

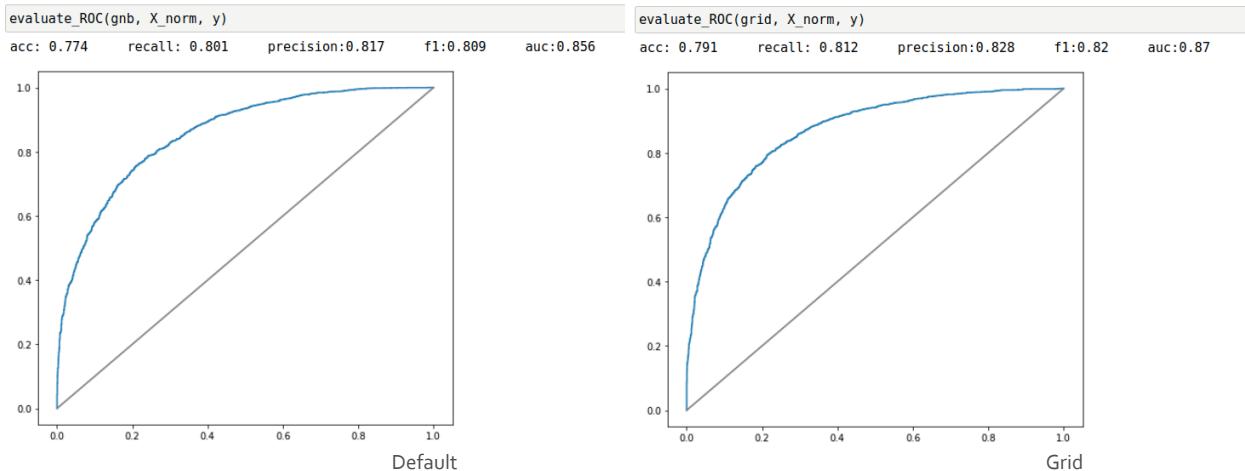
```

	Gaussian	Logistic Regression	SVC	Decision Tree Classifier	Random Forest Classifier
Tecnica					
Standard	0.798557	0.862268	0.862887	0.763299	0.856701
GridSearch	0.798557	0.863299	0.862268	0.742062	0.856289
RandomSearch	0	0	0	0	0

```
df_params_scores.to_csv('data/df_params_scores_grid.csv')
```

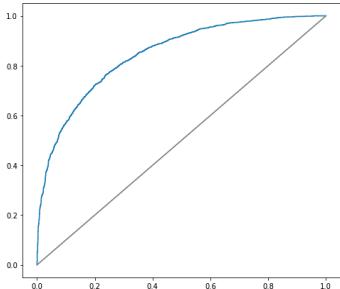
Para la curva ROC de cada modelo se han ejecutado individualmente los GridSearch correspondientes por separado. Los resultado se muestran a continuación:

● Gaussian Naive Bayes (GNB)



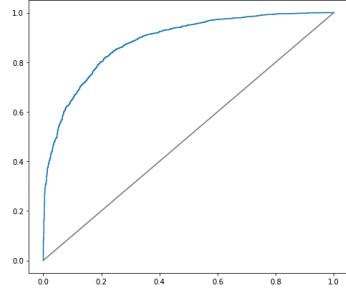
● Logistic Regression:

```
evaluate_ROC(logr, X_norm, y)
acc: 0.767      recall: 0.851      precision:0.772      f1:0.81      auc:0.845
/home/dsc/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfsgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.
Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result()
```



Default

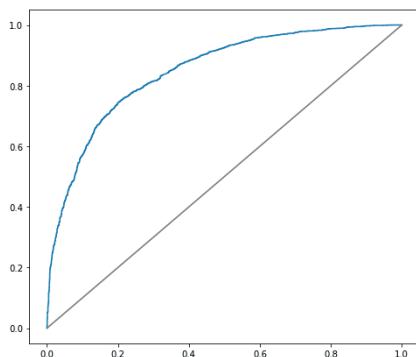
```
logr = LogisticRegression(max_iter=500, C=232.995, solver='sag')
evaluate_ROC(logr, X_norm, y)
acc: 0.808      recall: 0.867      precision:0.817      f1:0.841      auc:0.884
/home/dsc/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
```



Grid

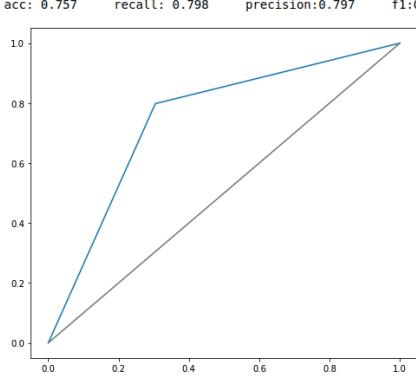
● Support Vector Classification (SVC):

```
evaluate_ROC(svc, X_norm, y)
acc: 0.77      recall: 0.852      precision:0.784      f1:0.816      auc:0.848
```

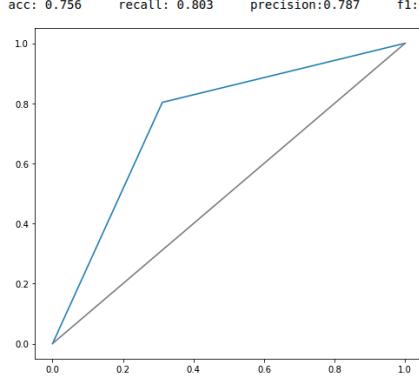


Default

```
evaluate_ROC(grid, X_norm, y)
acc: 0.756      recall: 0.803      precision:0.787      f1:0.795      auc:0.746
```



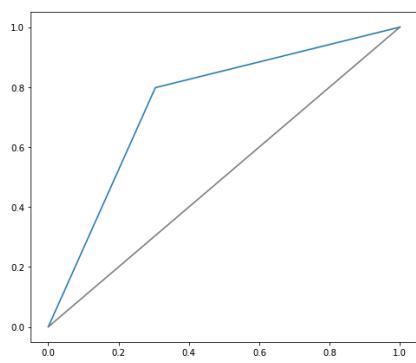
Default



Grid

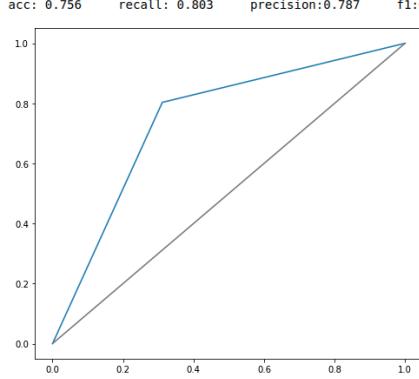
● Decision Tree Classifier:

```
evaluate_ROC(dtc, X_norm, y)
acc: 0.757      recall: 0.798      precision:0.797      f1:0.797      auc:0.747
```



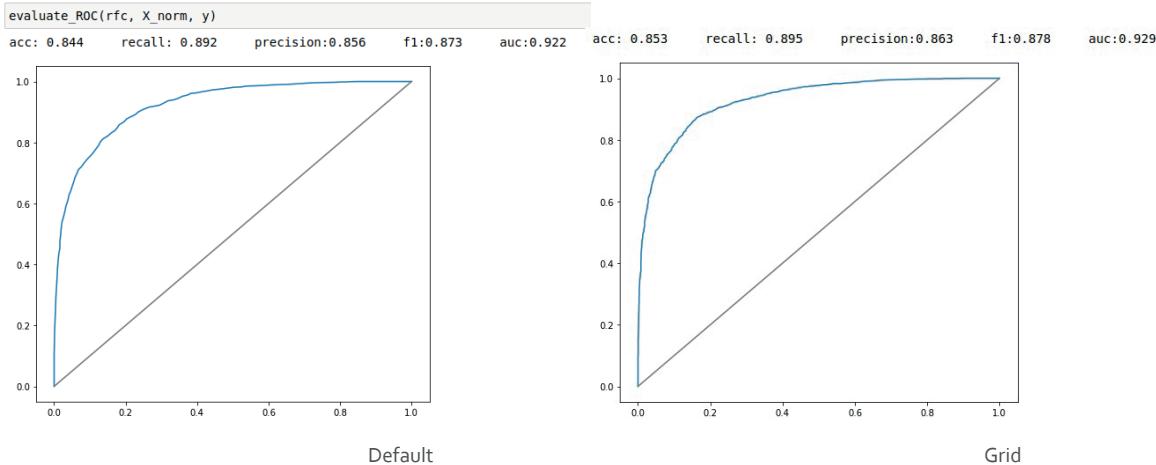
Default

```
evaluate_ROC(grid, X_norm, y)
acc: 0.756      recall: 0.803      precision:0.787      f1:0.795      auc:0.746
```



Grid

- Random Forest Classifier:



El siguiente intento que se va a realizar para obtener el mejor modelo posible es el RandomSearch, que a diferencia que el GridSearch que prueba todas las combinaciones posibles (con la cantidad de recursos que eso conlleva), el RandomSearch hace una búsqueda “aleatoria” entre todas las posibilidades.

Al igual que anteriormente, nos servimos del mismo bucle que hemos usado con el GridSearch, y guardamos los resultados.

3. RandomizedSearchCV

```
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

columnas_ps = ['Estimator', 'Params', 'Score']
df_params_scores_random = pd.DataFrame(columns = columnas_ps)

col = 0
for ind in range(0,len(models)):
    cv = RepeatedStratifiedKFold(n_splits=10,
                                n_repeats=3,
                                random_state=1)
    n_iter_search = 3
    random_search = RandomizedSearchCV(models[col],
                                         param_distributions=grids[col],
                                         n_iter=n_iter_search,
                                         cv=cv)

    random_search.fit(X_train,y_train)
    df_opt.iloc[2,col] = random_search.score(X_test,y_test)

    df_params_scores_random.loc[filas_aux, 'Estimator'] = random_search.estimator
    df_params_scores_random.loc[filas_aux, 'Params'] = str(random_search.best_params_)
    df_params_scores_random.loc[filas_aux, 'Score'] = random_search.best_score_

    col += 1
    filas_aux += 1

df_opt.head()
```

Tecnica	Gaussian	Logistic Regression	SVC	Decission Tree Classifier	Random Forest Classifier
Standard	0.798557	0.862268	0.862887	0.763299	0.856701
GridSearch	0.798557	0.863299	0.862268	0.742062	0.856289
RandomSearch	0.798557	0.862268	0.780206	0.733402	0.855052

Una vez que se tienen los resultados guardados, se cargan y se juntan para visualizarlos en conjunto y ver qué opción ha reportado mejores resultados.

```
df_params_scores_random.to_csv('data/df_params_scores_random.csv')

df_grid = pd.read_csv('data/df_params_scores_grid.csv')
df_grid = df_grid.drop('Unnamed: 0', axis=1)

df_random = pd.read_csv('data/df_params_scores_random.csv')
df_random = df_random.drop('Unnamed: 0', axis=1)

df_params = pd.concat([df_grid, df_random], ignore_index=True)
df_params = df_params.sort_values(by='Score', ascending=False)
df_params
```

	Estimator	Params	Score
1	LogisticRegression()	{'C': 0.013257113655901081, 'solver': 'lbfgs'}	0.865549
6	LogisticRegression()	{'solver': 'liblinear', 'C': 5.428675439323859}	0.865296
2	SVC()	{'C': 0.26366508987303555, 'gamma': 'auto', 'k...}	0.864563
4	RandomForestClassifier()	{'criterion': 'entropy', 'max_depth': None, 'm...}	0.852947
9	RandomForestClassifier()	{'n_estimators': 500, 'max_features': 'sqrt', ...}	0.852259
0	GaussianNB()	{'var_smoothing': 1e-08}	0.796994
5	GaussianNB()	{'var_smoothing': 1e-08}	0.796994
7	SVC()	{'kernel': 'sigmoid', 'gamma': 'auto', 'C': 88...}	0.780978
3	DecisionTreeClassifier()	{'ccp_alpha': 0.0, 'criterion': 'entropy', 'ma...}	0.744180
8	DecisionTreeClassifier()	{'max_features': 'log2', 'criterion': 'entropy...}	0.742988

Con esta tabla se aprecia claramente qué opciones son las mejores. Los que mejor resultado han obtenido han sido la Regresión logística con un 86.55% y 86.52% de acierto y Support Vector Classifier con un 86.45% de acierto.

```
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.25, random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

modelo_final = LogisticRegression(C = 0.013257113655901081, solver = 'lbfgs')
modelo_final.fit(X_train, y_train)
```

```
y_pred = modelo_final.predict(X_test)

print("Accuracy %: ")
print(np.mean(y_pred == y_test)*100)
print("-----")
print("Matrix Confusion: ")
print(confusion_matrix(y_test,y_pred))
print("-----")
print("Clasification report: ")
print(classification_report(y_test,y_pred))
```

```
Accuracy %:
86.3298969072165
-----
Matrix Confusion:
[[1600  359]
 [ 304 2587]]
-----
Clasification report:
      precision    recall  f1-score   support

          0       0.84      0.82      0.83     1959
          1       0.88      0.89      0.89     2891

   accuracy                           0.86      4850
  macro avg       0.86      0.86      0.86      4850
weighted avg       0.86      0.86      0.86      4850
```

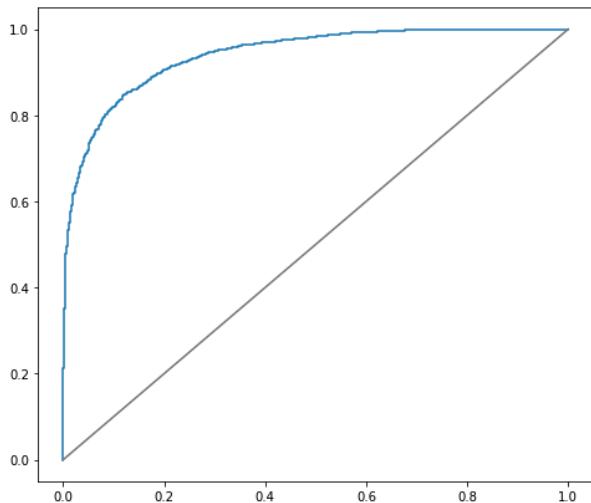
S
=

```
X = df.iloc[:,9:43].values
y = df.iloc[:,43].values
y=y.astype('int')

X_copy = X.copy()
X_norm = (X_copy - X_copy.mean()) / X_copy.std()

eval_ROC(X_norm, y)

acc: 0.863      recall: 0.895      precision:0.878      f1:0.886      auc:0.941
```



A continuación se guarda el modelo para futuros fines

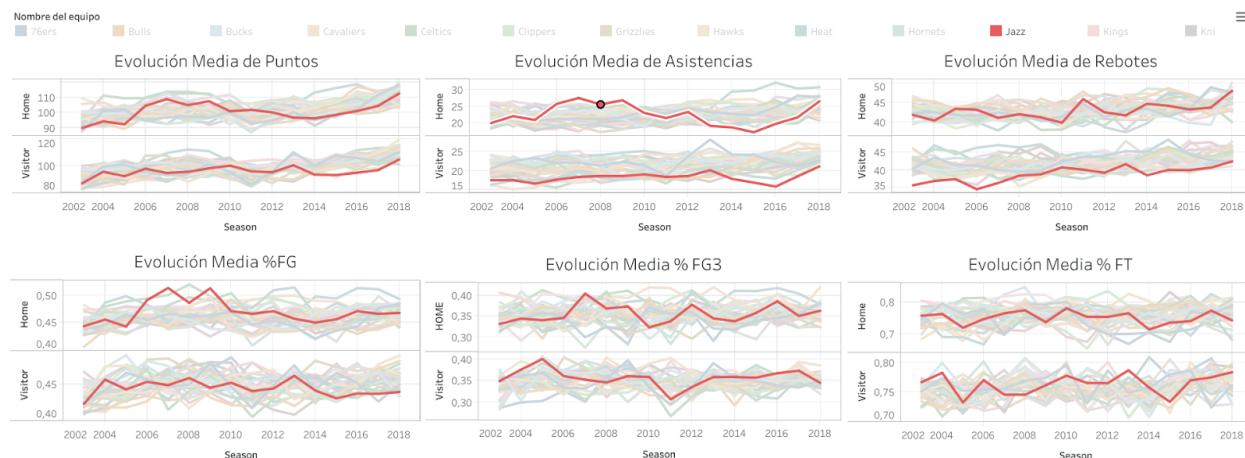
```
import joblib
joblib.dump(modelo_final, 'modelo_final.pkl')
```

5.5 VISUALIZACION TABLEAU

Para la visualización se ha provisto de 3 dashboards que permiten al usuario navegar y conocer más profundamente las estadísticas a lo largo de los años de los diferentes equipos que conforman la liga de la NBA.

5.5.1 DASHBOARD 1: EVOLUCION

En él se muestra la evolución de las 6 principales estadísticas: puntos, asistencias, rebotes, % tiros de 2, % tiros de 3 y % tiros libres. Cada variable está dividida en dos gráficos distintos, uno para los partidos en casa y otro para los de visitante, con esto se puede comparar fácilmente como puede cambiar un equipo según si juega en casa o fuera de casa. El dashboard incluye un pequeño filtro que permite seleccionar al usuario el equipo que quiere ver. Debido al gran número de equipos que hay, es un dashboard pensado para ver individualmente y de uno en uno los equipos. Ejemplo con la selección de equipo de "Jazz".

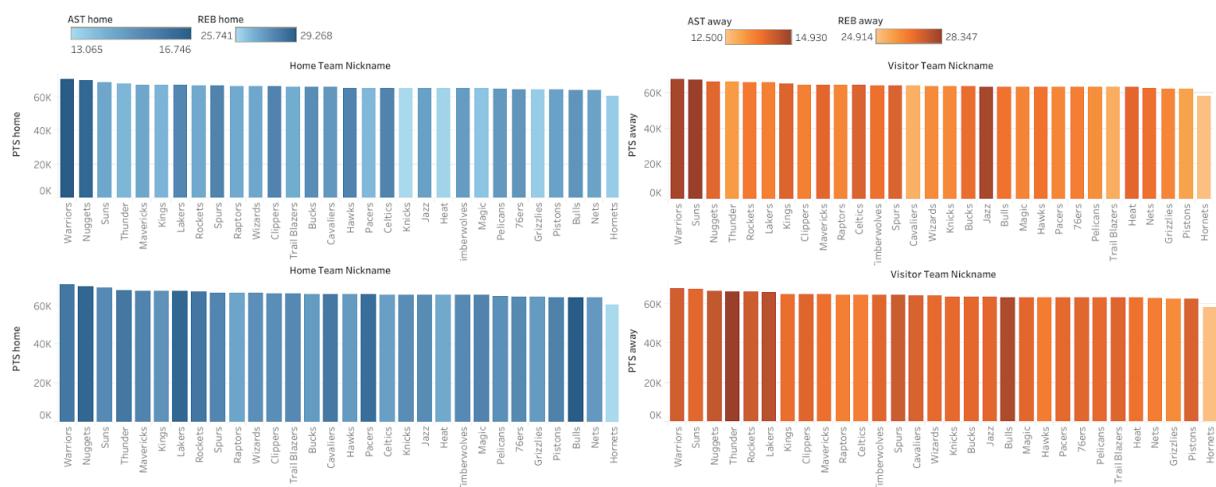


5.5.2 DASHBOARD 2: PUNTOS, REBOTES Y ASISTENCIAS

Este dashboard se divide en dos partes claramente separadas por el color, en azul si se juega en casa y en naranja si se juega fuera de casa. Tanto para casa como para fuera de casa se tienen dos gráficos de barras que muestran el total de puntos anotados por dicho equipo (en orden descendente). Además cada tonalidad indica para el gráfico que está por encima la cantidad de asistencia repartidas en total por dicho equipo, y en el de debajo la cantidad de rebotes capturados.

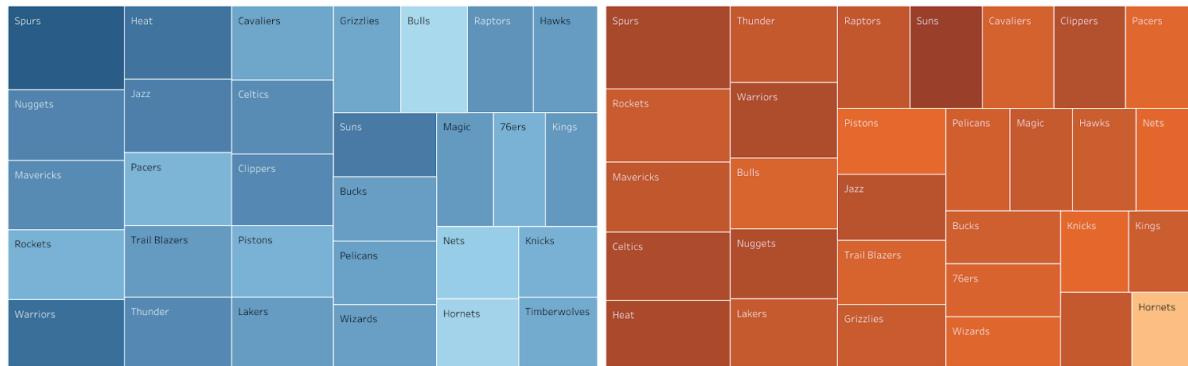
De esta forma cuanto más a la izquierda este el equipo mayor cantidad de puntos habrá anotado históricamente (en los últimos 15 años), y cuanto más oscura sea su barra más asistencia y rebotes habrá conseguido.

Esto es importante para entender la trascendencia que puede tener esto a la hora de ser un equipo competitivo. Se puede apreciar como un equipo que está muy a la izquierda y con tonalidad oscura son los "Spurs", comparando esto con los resultados del equipo en los últimos 15 años (4 Títulos y 5 finales) se entiende de la importancia de que un equipo debe meter muchos puntos, pero igual de importante puede ser repartir asistencias o capturar rebotes.



5.5.3 DASHBOARD 3: VICTORIAS Y % FG

El último dashboard está también dividido en dos, azul si se juega en casa y naranja si es fuera de casa. En él se muestra la cantidad de partidos ganados históricamente (últimos 15 años) por cada franquicia de la liga, la tonalidad indica cuánto más oscura un mayor % de acierto con tiros de campo (estadística muy ligada siempre a la cantidad de victorias que puede obtener un equipo).



El color muestra la media FG PCT en casa. El tamaño muestra la cantidad de partidos ganados en casa.

Prom. FG PCT home
0,44355 [0,48227]

El color muestra la media FG PCT fuera de casa. El tamaño muestra la cantidad de partidos ganados fuera de casa.

FG PCT away
265,45 [299,87]

6. CONCLUSIONES:

Gracias a la metodología que provee la ciencia de datos, se ha podido realizar un análisis completo de las estadísticas más importantes generadas por los equipos de la NBA en los últimos 15 años.

Se ha podido determinar con una buena precisión uno de los objetivos que se buscaban: pronosticar para un partido si un equipo ganará (intentando predecir la columna "Home team wins"). Esto se ha conseguido gracias a una limpieza completa de los datos, a la toma de decisiones (como elegir solo los partidos de "regular season", o prescindir de la temporada del COVID), a la ingeniería del dato realizada incluyendo variables que ningún otro proyecto o trabajo ha incluido y a las técnicas de trabajo que permiten optimizar los resultados que se pretendían conseguir.

Por otra parte también se ha conseguido el otro objetivo que planteaba este proyecto, realizar un análisis de las estadísticas a lo largo de los años de estudio propuesto para este trabajo (últimos 15 años). Gracias a la exploración de los datos realizada se ha podido conocer cómo se distribuyen y qué correlación existe entre las variables, pudiendo obtener conclusiones muy útiles tanto para la parte de predicción como de análisis de las estadísticas. Esto conjunto con los dashboards de Tableau propuestos dan una visión global del comportamiento de las estadísticas principales a través del tiempo, y cómo influyen en los diferentes equipos, permitiendo entender mejor porque unos equipos ganan más de otros.



¡MUCHAS GRACIAS!



Roberto Manzano López