Herramienta de predicción de partidos de la English Premier League

Daniel González Millán

IMF Business School & Universidad de Nebrija

Índice

Resumen

En este proyecto se va a desarrollar una herramienta que sea capaz de predecir el resultado de los partidos de la liga inglesa de fútbol, la English Premier League.

Para ello, nuestra herramienta será capaz de descargarse los datos de una página web que contiene los resultados históricos de esta competición, así como una serie de estadísticas que nos pueden ser de gran utilidad. Para esto, usaremos los paquetes BeautifulSoup, Requests y Pandas que nos permitirá leer la página web, obtener los datos que necesitamos, cargarlos en un dataframe y finalmente descargarlos en un archivo CSV.

Seguidamente se hará exploración del dataset (valores faltantes, adaptar las variables al tipo que queremos o modificarlas en caso de ser necesario) y se adaptarán los datos de forma que sean útiles para el tipo de modelo que vamos a utilizar.

Tras esto, procederemos a modelar nuestro problema para resolverlo.En este caso usaremos dos algoritmos de ensamblados distintos, uno será un algoritmo de *bagging* (Random Forests) y el otro será un algoritmo de *boosting* (XGBoost). Para ambos casos utilizaremos técnicas y medidores diferentes (tanto de eventos que podemos conocer antes del partido, como de eventos que iremos conociendo durante el transcurso y al final de este) para finalmente hacer una comparativa entre ellos y ser capaces de discernir qué modelo es el más adecuado para nuestro problema.

Finalmente, el problema se planteará de dos maneras diferentes, la primera como un problema de adivinar el resultado (es decir, si el equipo va a ganar, perder o empatar), mientras que la segunda será un problema de saber si el equipo va a ganar o no (en este caso, tanto perder como empatarse considerarán el mismo resultado).

Al final del proyecto se discutirán ambas perspectivas y se obtendrán conclusiones de ambas en función de los resultados obtenidos en cuanto a velocidad, precisión y demás.

1. Introducción

El fútbol es considerado como el deporte más popular en el mundo. Cada 4 años, la emisión deportiva más vista suele ser la final del campeonato mundial de fútbol. Hoy en día estamos acostumbrados a encontrarnos en casi cualquier ámbito de nuestra vida (salud, ocio, educación…) el uso del *Big Data* y del *Data Science* para mejorar nuestra experiencia o para la toma de decisiones basada en datos y, en este aspecto, el fútbol no es una excepción.

Durante el desarrollo de este trabajo, realizaremos una herramienta que tratará de predecir los resultados de un partido de fútbol, en nuestro caso nos centraremos en la liga inglesa de fútbol.

La English Premier League (EPL) es la máxima competición de fútbol de inglaterra, en ella, cada año se enfrentan 20 equipos durante 38 jornadas, es decir, cada equipo se enfrenta dos veces a cada uno de los 19 equipos restantes dos veces, una en su su estadio y la otra en el estadio del rival.

Esta liga es considerada la liga más antigua del mundo, ya que, aunque la competición que está vigente hoy en día se fundó en 1992, es hija de la antigua liga inglesa de fútbol, la English Football League (EFL) fundada en 1881.

La EPL es la liga más vista del mundo, se ve en casi cualquier rincón de nuestro planeta y tiene una audiencia potencial de 4.7 billones de personas.

Por cada partido de fútbol, al finalizar, se asignan 3 puntos al equipo ganador y 0 puntos al equipo perdedor, en caso de empatar, se asignará un punto a cada equipo. Al final de la temporada, los equipos se ordenarán en la tabla teniendo en cuenta el número de puntos que han conseguido sumar durante las 38 jornadas, resultando campeón el equipo que más puntos haya obtenido (en caso de haber empates a puntos, se utilizarán criterios secundarios como el de resultado directo entre los equipos, más goles a favor o menos goles en contra).

Cada temporada, los primeros 7 equipos juegan competición europea (la competición varía dependiendo de si quedas entre las posiciones 1-4, 5-6 o 7) y los tres últimos equipos descenderán a la segunda división. Los 3 huecos libres que dejan estos tres equipos se ocuparán por los tres mejores equipos de la segunda división, que ascenderán de categoría. Esto es importante, ya que cada temporada habrá una rotación de tres equipos, por lo que, en nuestro set de datos, habrá equipos que tendrán más partidos disputados (aquellos que no han descendido en ninguna de las temporadas) y habrá equipos que tendrán menos (aquellos que solo han estado una temporada).

Aunque los equipos que vamos a tratar disputan competiciones diferentes (FA Cup, Copa de la liga o incluso competiciones a nivel europeo), nosotros nos centraremos solamente en la EPL.

2. Business Case

Hoy en día y desde hace muchos años, el análisis de datos ha llegado a los deportes para quedarse.

Por un lado, los equipos tienen equipos de científicos de datos por detrás, analizan en cada partido los km recorridos por cada jugador, la velocidad media, su cansancio, en número de pases, etc. Todo para poder analizar el desempeño individual de cada jugador y ser capaz de gestionar mejor sus esfuerzos, decidir cuándo es más óptimo sustituir a un jugador o la influencia real de este en el juego. Esto está llegando a provocar cambios en la forma de jugar de los jugadores (Adam, D. (2022)), dejan de tirar a puerta desde cierta distancia por la baja probabilidad de anotar a partir de una cierta distancia, los entrenadores se centran más en dónde quieren que sus jugadores tengan la posesión (cuanto más tengan el balón lejos de su propia portería, menos probable que el equipo contrario anote) y los jugadores deciden la jugada o a quién pasar en función de estos datos también.

Esto lo podemos observar en equipos de fútbol como el Brentford FC, que utiliza un algoritmo interno encargado de calificar a los jugadores de muchas ligas diferentes y se utiliza para reclutar a estrellas emergentes con bajo valor en el mercado. También tenemos casos como el Liverpool FC, que ha creado un modelo encargado de evaluar si el trabajo realizado por un jugador determinado influye directamente en la probabilidad de hacer un gol.

Por todo esto, cada vez se usa menos la intuición de los jugadores o el entrenador para dejar paso a decisiones sustentadas en datos.

Obviamente esto ha cambiado el paradigma del fútbol para bien, pero también para mal, hay que tener en cuenta que todas y cada una de las acciones de un jugador durante los partidos de fútbol son monitorizadas, así como los entrenamientos. Se hacen procesado de imágenes post partido en la que se mide la influencia de cada uno de los jugadores en el partido, así como se mide su esfuerzo. Esto provoca que

se optimice el rendimiento de un equipo, pero también puede añadir presión a los jugadores a la hora de que los equipos de analistas tengan una gran cantidad de control sobre la medición de su rendimiento o incluso su influencia en el campo de fútbol que puede rozar la ética.

Por otro lado, las ligas profesionales también tienen equipos que realizan análisis sobre los partidos, las asistencias a los estadios o la cantidad de espectadores en la televisión. Este tipo de recolección de datos les resulta de gran utilidad para fijar los horarios, vender los paquetes de derechos televisivos a los distintos canales internacionales o incluso a la designación o no de un árbitro en un partido si la liga pudiera detectar mediante el uso de datos que un árbitro es determinante en el resultado de un determinado equipo.

Por último, las casas de apuestas mueven mucho dinero. Según el portal del internet *The Business Research Company* (https://www.thebusinessresearchcompany.com/report/sports-betting-global-market-report) se espera que el mercado de las apuestas crezca de 89.5 billones de dólares en 2021 a 99.2 billones de dólares en 2022, con un crecimiento esperado para 2026 de 144.34 billones de dólares.

Las apuestas son algo que existe desde hace muchos años pero se han popularizado mucho más en los últimos años desde la aparición de las casas de apuestas online, ya que estas han dotado de mayor accesibilidad el hacer una apuesta.Se puede apostar en casi cualquier deporte, obviamente uno de los mercados más grandes es el fútbol, ya que es el deporte de mayor éxito global. Se pueden hacer apuestas de casi cualquier ítem o estadística del partido (tiros a puerta, número de córners, resultado final…) y además, con la modalidad online, se pueden hacer apuestas en directo.

Para tratar de asegurar el máximo beneficio a la hora de fijar una cuota, las casas de apuestas utilizan multitud de datos de los equipos, jugadores, entrenadores y cualquier ítem que les pueda ser útil relacionado con el partido en cuestión.

Por otro lado, para tratar de ganar dinero con las apuestas, tienes que conseguir acertar aquello que apostaste, muchos profesionales de las apuestas utilizan hoy en día sus propios programas externos, que tratan de tener una precisión mayor a los de las casas de apuestas para poder así ganar mucho dinero.

3. Objetivos

El objetivo primordial de este proyecto es ser capaz de utilizar las herramientas necesarias para crear una herramienta que sea capaz de predecir los resultados de la jornada de la EPL con la mayor precisión posible.

Para ello, nuestra herramienta tiene que ser capaz de dos cosas principalmente:

- La herramienta tiene que ser capaz de descargarse los datos con los resultados de la EPL, así como estadísticas relativas al aforo, si se juega en casa o fuera y también datos relativos al partido (número de corners, faltas, tarjetas…) para poder alimentar y entrenar al modelo.
- La herramienta aplicará el modelo que más se ajuste a la resolución del problema, dando el mejor resultado posible. Para ello, se usarán diferentes modelos y diferentes métricas, calculando su precisión y encajando cada una de ellas dentro del problema, para así poder determinar qué modelo es el que se ajusta mejor.

Plantearemos el problema desde dos perspectivas distintas, la de tratar de acertar el resultado final (victoria, empate o derrota) y la de discernir si el equipo es capaz de ganar o no. Haremos estos dos planteamientos ya que, al contar una victoria como 3 puntos, mientras que un empate solo vale 1 punto y una derrota 0, la victoria en la liga tiene mucha más importancia que empatar, ya que hay menos diferencia en cuanto a puntos entre empatar o perder que entre empatar o ganar.

Plantear el problema como ganar o no, nos debería proporcionar a priori más precisión que plantearlo como sistema de victoria-empate-derrota, ya que tenemos dos resultados posibles frente a 3.

Por lo tanto, para estos dos problemas, nuestro objetivo será ver, para los modelos de *bagging* y *boosting* que vamos a aplicar, cuál es el mejor modelo a aplicar para cada caso y si nos conviene plantear el problema de una manera o de otra. Para

ello, tendremos en cuenta los resultados obtenidos, el tiempo que ha llevado cada uno y qué solución encaja mejor en nuestro problema.

4. Metodología

El entorno de desarrollo que utilizaremos para el proyecto será JupyterLab.

De acuerdo con la página web de Jupyter (https://jupyter.org/): *"JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality"*.

Es decir, a priori, JupyterLab es la evolución natural de Jupyter Notebook. Permite desarrollar notebooks, programar y trabajar con datos. Es por ello por lo que ha sido el entorno seleccionado para este proyecto, ya que necesitaremos desarrollar una herramienta que sea capaz de obtener los datos a partir de la página web, trabajar con la base de datos para adaptarla a lo que necesitamos y se capaz de aplicarles modelos de Machine Learning para poder llevar a cabo nuestras predicciones.

4.1. Web Scraping

El web scraping consiste en obtener datos de una página web mediante HTTP (Hyper Text Protocol), que nos permitirá descargarnos el código fuente de la página y, a partir de este, obtener los datos que necesitemos de esta. Es una práctica que es totalmente legal, pero siempre viene bien mirar los Términos y Condiciones de la página web con respecto a sus datos, para estar seguros de no violar ninguna de ellas y no tener problemas a la hora de realizar esta práctica.

Para desarrollar esta parte, importaremos dos módulos que serán necesarios:

```
import requests
from bs4 import BeautifulSoup
```

- Por un lado tenemos requests, que nos permite hacer peticiones HTTP usando python.
- Por otro lado, tenemos BeautifulSoup, que nos servirá para parsear los datos de la página y extraer todo aquello que vayamos a necesitar.

La página web que vamos a usar es (decir aquí qué es) (https://fbref.com/en/comps/9/Premier-League-Stats) en la que nos encontramos una página de este estilo:

Rk	Squad	MP	W	D	L	GF	GA	GD	Pts	Pts/MP	xG	xGA	xGD	xGD/90	Last 5	Attendance	Top Team Scorer	Goalkeeper	Notes
1	Arsenal	13	11	1	1	31	11	+20	34	2.62	24.1	11.2	+12.9	+0.99	W W D W W	60,135	Gabriel Jesus, Martinelli - 5	Aaron Ramsdale	
2	Manchester City	13	10	2	1	39	12	+27	32	2.46	26.0	9.9	+16.1	+1.24	W L W W W	53,205	Erling Haaland - 18	Ederson	
3	Newcastle Utd	14	7	6	1	28	11	+17	27	1.93	23.3	14.1	+9.3	+0.66	D W W W W	52,164	Miguel Almirón - 8	Nick Pope	
4	Tottenham	14	8	2	4	27	18	+9	26	1.86	22.3	15.5	+6.9	+0.49	W L L W L	61,667	Harry Kane - 11	Hugo Lloris	
5	Manchester Utd	13	7	2	4	18	19	-1	23	1.77	16.3	15.9	+0.3	+0.03	D W D W L	73,729	Marcus Rashford - 4	David de Gea	
6	Brighton	13	6	3	4	22	17	+5	21	1.62	20.8	14.4	+6.4	+0.50	L D L W W	31,400	Leandro Trossard - 7	Robert Sánchez	
7	Chelsea	13	6	3	4	17	16	+1	21	1.62	15.1	18.4	-3.2	-0.25	W D D L L	39,901	Raheem Sterling, Kai Havertz - 3	Edouard Mendy	
8	Liverpool	13	5	4	4	25	16	+9	19	1.46	22.3	18.4	+3.8	+0.30	W W L L W	53,211	Mohamed Salah, Roberto Firmino - 6	Alisson	
9	Fulham	14	5	4	5	23	24	-1	19	1.36	18.3	24.8	-6.5	-0.46	D W W D L	22,675	Aleksandar Mitrović - 9	Bernd Leno	
10	Crystal Palace	13	5	4	4	15	17	-2	19	1.46	12.8	17.2	-4.4	-0.34	D W L W W	24,691	Wilfried Zaha - 6	Vicente Guaita	
11	Brentford	14	3	7	4	21	24	-3	16	1.14	19.0	18.7	+0.3	+0.02	W D L D D	17,066	Ivan Toney - 8	David Raya	
12	Leeds United	13	4	3	6	19	22	-3	15	1.15	19.2	18.4	+0.9	+0.07	L L L W W	36,441	Rodrigo - 7	Illan Meslier	
13	Aston Villa	14	4	3	7	14	21	-7	15	1.07	15.6	18.6	-3.0	-0.21	L L W L W	41,682	Leon Bailey, Danny Ings - 3	Emiliano Martínez	
14	Leicester City	14	4	2	8	23	25	-2	14	1.00	15.9	20.8	-5.0	-0.35	D W W L W	31,630	James Maddison - 6	Danny Ward	
15	West Ham	14	4	2	8	12	15	-3	14	1.00	17.2	13.8	+3.4	+0.25	D L W L L	62,442	Gianluca Scamacca, Jarrod Bowen... - 2	Łukasz Fabiański	
16	Everton	14	3	5	6	11	14	-3	14	1.00	14.6	23.0	-8.4	-0.60	L L W D L	39,209	Anthony Gordon - 3	Jordan Pickford	
17	Bournemouth	14	3	4	7	15	32	-17	13	0.93	9.2	23.4	-14.1	-1.01	D L L L L	10,397	Philip Billing - 4	Neto	
▼ 18	Southampton	14	3	3	8	12	24	-12	12	0.86	13.6	17.3	-3.7	-0.26	D W D L L	30,681	Che Adams - 3	Gavin Bazunu	
▼ 19	Wolves	14	2	4	8	8	22	-14	10	0.71	14.2	17.5	-3.3	-0.24	W L L D L	31,189	Rúben Neves - 4	José Sá	
▼ 20	Nott'ham Forest	14	2	4	8	10	30	-20	10	0.71	14.2	22.9	-8.7	-0.62	L D W L D	29,039	Taiwo Awoniyi - 3	Dean Henderson	

En ella, podemos observar que aparece una tabla con las estadísticas de los equipos de la liga y, si pulsamos sobre el nombre del equipo, nos aparecerá una nueva página en la que obtendremos una serie de tablas con los datos de la temporada del equipo.

Podemos encontrar varias secciones diferentes como “shooting”, “passing”, “scores and fixtures” y “shooting”. Nosotros nos vamos a centrar en estas dos últimas, que contienen los datos que vamos a necesitar.

La tabla de “scores and fixtures” contiene datos como goles a favor, goles en contra, posesión o incluso el nombre del árbitro o del capitán del equipo. Tiene una forma de este estilo:

Scores & Fixtures 2022-2023 Arsenal: All Competitions Share & Export ▾ Glossary

Date	Time	Comp	Round	Day	Venue	Result	GF	GA	Opponent	xG	xGA	Poss	Attendance	Captain	Formation	Referee	Match Report	Notes
2022-08-05	20:00 (05:00)	Premier League	Matchweek 1	Fri	Away	W	2	0	Crystal Palace	1.0	1.2	44	25,286	Martin Ødegaard	4-3-3	Anthony Taylor	Match Report	
2022-08-13	15:00 (00:00)	Premier League	Matchweek 2	Sat	Home	W	4	2	Leicester City	2.7	0.5	50	60,033	Martin Ødegaard	4-2-3-1	Darren England	Match Report	
2022-08-20	17:30 (02:30)	Premier League	Matchweek 3	Sat	Away	W	3	0	Bournemouth	1.3	0.3	57	10,423	Martin Ødegaard	4-2-3-1	Craig Pawson	Match Report	
2022-08-27	17:30 (02:30)	Premier League	Matchweek 4	Sat	Home	W	2	1	Fulham	2.6	0.8	71	60,164	Martin Ødegaard	4-2-3-1	Jarred Gillett	Match Report	
2022-08-31	19:30 (04:30)	Premier League	Matchweek 5	Wed	Home	W	2	1	Aston Villa	2.4	0.4	59	60,012	Martin Ødegaard	4-2-3-1	Robert Jones	Match Report	
2022-09-04	16:30 (01:30)	Premier League	Matchweek 6	Sun	Away	L	1	3	Manchester Utd	1.3	1.5	60	73,431	Martin Ødegaard	4-2-3-1	Paul Tierney	Match Report	
2022-09-08	18:45 (02:45)	Europa Lg	Group stage	Thu	Away	W	2	1	Zürich	2.3	1.2	69	17,070	Granit Xhaka	4-3-3	Mohammed Al Hakim	Match Report	
2022-09-11	14:00 (23:00)	Premier League	Matchweek 7	Sun	Home				Everton									Match Postponed
2022-09-18	12:00 (21:00)	Premier League	Matchweek 8	Sun	Away	W	3	0	Brentford	1.5	0.5	63	17,122	Granit Xhaka	4-2-3-1	David Coote	Match Report	
2022-10-01	12:30 (21:30)	Premier League	Matchweek 9	Sat	Home	W	3	1	Tottenham	2.4	1.6	64	60,278	Martin Ødegaard	4-2-3-1	Anthony Taylor	Match Report	
2022-10-06	20:00 (05:00)	Europa Lg	Group stage	Thu	Home	W	3	0	Bodø/Glimt	3.4	0.8	50	59,724	Granit Xhaka	4-3-3	Harm Osmers	Match Report	
2022-10-09	16:30 (01:30)	Premier League	Matchweek 10	Sun	Home	W	3	2	Liverpool	2.7	1.1	43	60,059	Martin Ødegaard	4-2-3-1	Michael Oliver	Match Report	
2022-10-13	18:45 (02:45)	Europa Lg	Group stage	Thu	Away	W	1	0	Bodø/Glimt	0.7	0.5	55	7,922	Martin Ødegaard	4-2-3-1	Irfan Peljto	Match Report	
2022-10-16	14:00 (23:00)	Premier League	Matchweek 11	Sun	Away	W	1	0	Leeds United	0.5	1.8	53	36,700	Martin Ødegaard	4-2-3-1	Chris Kavanagh	Match Report	
2022-10-19	19:30 (04:30)	Premier League	Matchweek 12	Wed	Home				Manchester City									Match Postponed
2022-10-20	18:00 (03:00)	Europa Lg	Group stage	Thu	Home	W	1	0	PSV Eindhoven	2.7	0.3	62	52,200	Granit Xhaka	4-2-3-1	Alejandro Hernández	Match Report	
2022-10-23	14:00 (23:00)	Premier League	Matchweek 13	Sun	Away	D	1	1	Southampton	1.0	0.7	59	31,145	Martin Ødegaard	4-3-3	Robert Jones	Match Report	
2022-10-27	18:45 (02:45)	Europa Lg	Group stage	Thu	Away	L	0	2	PSV Eindhoven	1.4	0.8	68	35,000	Martin Ødegaard	4-3-3	Marco Di Bello	Match Report	
2022-10-30	14:00 (00:00)	Premier League	Matchweek 14	Sun	Home	W	5	0	Nott'ham Forest	2.3	0.4	69	60,263	Martin Ødegaard	4-3-3	Simon Hooper	Match Report	
2022-11-03	20:00 (06:00)	Europa Lg	Group stage	Thu	Home	W	1	0	Zürich	1.6	0.4	55	48,500	Gabriel Jesus	4-3-3	Erik Lambrechts	Match Report	

Por otro lado, tenemos la tabla de “shooting” esta contiene datos más concretos sobre el partido (tiros a puerta, número de pases, número de faltas…) y tiene la siguiente forma:

Date	Time	Comp	Round	Day	Venue	Result	GF	GA	Opponent	Gls	Sh	SoT	SoT%	G/Sh	G/SoT	Dist	FK	PK	PKatt	xG	npxG	npxG/Sh	G-xG	np:G-xG
2022-08-05	20:00 (05:00)	Premier League	Matchweek 1	Fri	Away	W	2	0	Crystal Palace	1	10	2	20.0	0.10	0.50	14.6	1	0	0	1.0	1.0	0.10	0.0	0.0
2022-08-13	15:00 (00:00)	Premier League	Matchweek 2	Sat	Home	W	4	2	Leicester City	4	19	7	36.8	0.21	0.57	13.0	0	0	0	2.7	2.7	0.16	+1.3	+1.3
2022-08-20	17:30 (02:30)	Premier League	Matchweek 3	Sat	Away	W	3	0	Bournemouth	3	14	6	42.9	0.21	0.50	14.8	0	0	0	1.3	1.3	0.10	+1.7	+1.7
2022-08-27	17:30 (02:30)	Premier League	Matchweek 4	Sat	Home	W	2	1	Fulham	2	22	8	36.4	0.09	0.25	15.5	1	0	0	2.6	2.6	0.12	-0.6	-0.6
2022-08-31	19:30 (04:30)	Premier League	Matchweek 5	Wed	Home	W	2	1	Aston Villa	2	22	8	36.4	0.09	0.25	16.3	1	0	0	2.4	2.4	0.12	-0.4	-0.4
2022-09-04	16:30 (01:30)	Premier League	Matchweek 6	Sun	Away	L	1	3	Manchester Utd	1	16	3	18.8	0.06	0.33	18.6	1	0	0	1.3	1.3	0.08	-0.3	-0.3
2022-09-08	18:45 (02:45)	Europa Lg	Group stage	Thu	Away	W	2	1	Zürich	2	18	8	44.4	0.11	0.25	15.4	0	0	0	2.3	2.3	0.13	-0.3	-0.3
2022-09-18	12:00 (21:00)	Premier League	Matchweek 8	Sun	Away	W	3	0	Brentford	3	13	7	53.8	0.23	0.43	18.2	0	0	0	1.5	1.5	0.12	+1.5	+1.5
2022-10-01	12:30 (21:30)	Premier League	Matchweek 9	Sat	Home	W	3	1	Tottenham	3	22	9	40.9	0.14	0.33	18.6	1	0	0	2.4	2.4	0.11	+0.6	+0.6
2022-10-06	20:00 (05:00)	Europa Lg	Group stage	Thu	Home	W	3	0	Bodø/Glimt	3	18	6	33.3	0.17	0.50	14.7	1	0	0	3.4	3.4	0.19	-0.4	-0.4
2022-10-09	16:30 (01:30)	Premier League	Matchweek 10	Sun	Home	W	3	2	Liverpool	3	10	6	60.0	0.20	0.33	14.3	0	1	1	2.7	2.3	0.24	+0.3	-0.3

Para obtener los datos de estas tablas tendremos primero que descargar el HTML de la página a nuestro Notebook. Para ello, requests tiene el método “get”, que nos permite pasarle el link de la página y este se encargará de descargar el HTML de la página, por lo tanto, realizaremos la siguiente acción:

```
standings_url = "https://fbref.com/en/comps/9/Premier-League-Stats"
datos = requests.get(standings_url)
```

Con esto tendremos descargado el HTML de la página en nuestro Notebook, sin embargo, si quisiéramos ver lo que contiene, podemos usar “datos.text” para visualizarlo, no obstante esto nos va a devolver una grandísima cantidad de caracteres inteligibles, por lo que tenemos que hacer un poco de limpieza.

El siguiente paso es obtener todos los links de los 20 equipos de la competición, para poder acceder, para cada uno de ellos, a su página propia, en la que encontraremos sus correspondientes tablas de “scores and fixtures” y “shooting”. Lo primero que tenemos que observar es cómo obtener estos links. Para ello, lo que haremos será usar el elemento “inspeccionar” de google Chrome, que te permite ver qué elemento está relacionado con qué tag de la parte del código HTML de la página.

Si nos fijamos bien, para cada equipo de fútbol hay un elemento que hace referencia a él (elemento href) del tipo “a”, esta “a” viene del inglés *anchor*. Este es el elemento ancla de la página que hace referencia al link propio de cada equipo para poder acceder a sus datos.

Entonces nuestro siguiente paso será obtener todos estos elementos ancla para todos los equipos de la página. Para realizar esta parte, utilizaremos BeautifulSoup, que nos permitirá parsear el HTML y poder trabajar con él de manera más cómoda:

```
soup = BeautifulSoup(datos.text)
standings_table = soup.select('table.stats_table')[0]
links = standings_table.find_all('a')
links = [l.get("href") for l in links]
links = [ l for l in links if '/squads' in l]
```

Con la primera parte, lo que hacemos es inicializar BeautifulSoup utilizando su clase y pasándole como argumento nuestro HTML. Tras esto, seleccionaremos el tag de la tabla (table) que tiene la clase “stats_table”, de tal manera que en nuestra variable “standings_table” tendremos la parte del HTML correspondiente a esa tabla.

Finalmente obtendremos los links referentes a cada uno de los equipos buscando todos los elementos *anchor* (o elementos 'a') de la tabla e iterando sobre ellos. Obteniendo los siguientes resultados:

```
links
```

```
['/en/squads/18bb7c10/Arsenal-Stats',
 '/en/squads/b8fd03ef/Manchester-City-Stats',
 '/en/squads/b2b47a98/Newcastle-United-Stats',
 '/en/squads/361ca564/Tottenham-Hotspur-Stats',
 '/en/squads/19538871/Manchester-United-Stats',
 '/en/squads/d07537b9/Brighton-and-Hove-Albion-Stats',
 '/en/squads/cff3d9bb/Chelsea-Stats',
 '/en/squads/822bd0ba/Liverpool-Stats',
 '/en/squads/fd962109/Fulham-Stats',
 '/en/squads/47c64c55/Crystal-Palace-Stats',
 '/en/squads/cd051869/Brentford-Stats',
 '/en/squads/5bfb9659/Leeds-United-Stats',
 '/en/squads/8602292d/Aston-Villa-Stats',
 '/en/squads/a2d435b3/Leicester-City-Stats',
 '/en/squads/7c21e445/West-Ham-United-Stats',
 '/en/squads/d3fd31cc/Everton-Stats',
 '/en/squads/4ba7cbea/Bournemouth-Stats',
 '/en/squads/33c895d4/Southampton-Stats',
 '/en/squads/8cec06e1/Wolverhampton-Wanderers-Stats',
 '/en/squads/e4a775cb/Nottingham-Forest-Stats']
```

Una vez tenemos esto, solo tenemos que convertir nuestros links en URLS completas para poder empezar a trabajar con ellas y poder empezar a extraer los datos de cada equipo, esto lo haremos con los siguientes comandos:

```
urls_equipos = [f"https://fbref.com{l}" for l in links]
urls_Arsenal = urls_equipos[0]
datos_Arsenal = requests.get(urls_Arsenal)
```

Al igual que hicimos al principio, hemos usado Requests para obtener la información de la URL correspondiente, en este caso, al Arsenal.

Ahora nos centraremos en obtener nuestra tabla de "scores and fixtures", para ello,usaremos la librería Pandas. Esta librería nos permite obtener el trozo de HTML que queremos y parsearlo directamente como un Dataframe mediante el método "read_html", que buscará entre todas las tablas del HTML la cadena de caracteres "scores and fixtures" y tomará esa tabla. Por lo tanto, el resultado que obtendremos

será directamente una Dataframe de la tabla de “scores and fixtures”, con todas sus columnas y todos sus datos. Para ello, el código a ejecutar será el siguiente:

```
import pandas as pd
matches = pd.read_html(datos_Arsenal.text, match="Scores & Fixtures")
matches[0]
```

El resultado matches nos devuelve una lista de la que tendremos que tomar el primer elemento, que es el Dataframe que queremos obtener, que tendrá un aspecto como este:

	Date	Time	Comp	Round	Day	Venue	Result	GF	GA	Opponent	xG	xGA	Poss	Attendance	Captain	Formation	Referee	Match Report
0	2022-08-05	20:00	Premier League	Matchweek 1	Fri	Away	W	2.0	0.0	Crystal Palace	1.0	1.2	44.0	25286.0	Martin Ødegaard	4-3-3	Anthony Taylor	Match Report
1	2022-08-13	15:00	Premier League	Matchweek 2	Sat	Home	W	4.0	2.0	Leicester City	2.7	0.5	50.0	60033.0	Martin Ødegaard	4-2-3-1	Darren England	Match Report
2	2022-08-20	17:30	Premier League	Matchweek 3	Sat	Away	W	3.0	0.0	Bournemouth	1.3	0.3	57.0	10423.0	Martin Ødegaard	4-2-3-1	Craig Pawson	Match Report
3	2022-08-27	17:30	Premier League	Matchweek 4	Sat	Home	W	2.0	1.0	Fulham	2.6	0.8	71.0	60164.0	Martin Ødegaard	4-2-3-1	Jarred Gillett	Match Report
4	2022-08-31	19:30	Premier League	Matchweek 5	Wed	Home	W	2.0	1.0	Aston Villa	2.4	0.4	59.0	60012.0	Martin Ødegaard	4-2-3-1	Robert Jones	Match Report
5	2022-09-04	16:30	Premier League	Matchweek 6	Sun	Away	L	1.0	3.0	Manchester Utd	1.3	1.5	60.0	73431.0	Martin Ødegaard	4-2-3-1	Paul Tierney	Match Report
6	2022-09-08	18:45	Europa Lg	Group stage	Thu	Away	W	2.0	1.0	ch Zürich	2.3	1.2	69.0	17070.0	Granit Xhaka	4-3-3	Mohammed Al Hakim	Match Report

Realizando la misma operación para cada uno de los equipos de la liga, obtendremos sus datasets con los datos que necesitamos.

No obstante, en esta tabla no tenemos todos los datos que queremos obtener, en esta tabla tenemos datos básicos de los partidos como “goles a favor”, “goles en contra”, “posesión”, etc.

Sin embargo, si queremos datos un poco más elaborados de lo que ocurrió en el partido, como el número de tiros, pases … Estos datos se encuentran en la tabla “shooting”, por lo que también queremos obtener esta tabla para todos los equipos de la liga.

Para ello, haremos algo parecido a lo que hicimos con la tabla anterior, buscaremos los enlaces que nos lleven a la página de la tabla, volviendo a tomar todos los

elementos “anchor”, de estos tomaremos los elementos “href” que contengan “all_comps/shooting” y esto nos devolverá cuatro enlaces, que si observamos, son los mismos:

```
soup = BeautifulSoup(datos_Arsenal.text)
links = soup.find_all('a')
links = [l.get("href") for l in links]
links = [l for l in links if l and 'all_comps/shooting/' in l]
links
```

```
['/en/squads/18bb7c10/2022-2023/matchlogs/all_comps/shooting/Arsenal-Match-Logs-All-Competitions',
 '/en/squads/18bb7c10/2022-2023/matchlogs/all_comps/shooting/Arsenal-Match-Logs-All-Competitions',
 '/en/squads/18bb7c10/2022-2023/matchlogs/all_comps/shooting/Arsenal-Match-Logs-All-Competitions',
 '/en/squads/18bb7c10/2022-2023/matchlogs/all_comps/shooting/Arsenal-Match-Logs-All-Competitions']
```

A partir de aquí, tomaremos el primero, por ejemplo, el primero y haremos lo mismo que antes, esta vez buscando por la palabra clave “shooting” y nos quedaremos con el primer elemento de la lista:

```
datos_shooting = requests.get(f"https://fbref.com{links[0]}")
shooting = pd.read_html(datos_shooting.text, match="Shooting")[0]
shooting
```

que nos dará lugar al Dataframe deseado:

										For Arsenal	...	Standard							Expected		Unnamed: 25_level_0
	Date	Time	Comp	Round	Day	Venue	Result	GF	GA	Opponent	...	Dist	FK	PK	PKatt	xG	npxG	npxG/Sh	G-xG	np:G-xG	Match Report
0	2022-08-05	20:00	Premier League	Matchweek 1	Fri	Away	W	2	0	Crystal Palace	...	14.6	1	0	0	1.0	1.0	0.10	0.0	0.0	Match Report
1	2022-08-13	15:00	Premier League	Matchweek 2	Sat	Home	W	4	2	Leicester City	...	13.0	0	0	0	2.7	2.7	0.16	1.3	1.3	Match Report
2	2022-08-20	17:30	Premier League	Matchweek 3	Sat	Away	W	3	0	Bournemouth	...	14.8	0	0	0	1.3	1.3	0.10	1.7	1.7	Match Report
3	2022-08-27	17:30	Premier League	Matchweek 4	Sat	Home	W	2	1	Fulham	...	15.5	1	0	0	2.6	2.6	0.12	-0.6	-0.6	Match Report
4	2022-08-31	19:30	Premier League	Matchweek 5	Wed	Home	W	2	1	Aston Villa	...	16.3	1	0	0	2.4	2.4	0.12	-0.4	-0.4	Match Report
5	2022-09-04	16:30	Premier League	Matchweek 6	Sun	Away	L	1	3	Manchester Utd	...	18.6	1	0	0	1.3	1.3	0.08	-0.3	-0.3	Match Report

No obstante, si nos fijamos bien, podemos observar que este Dataframe es multinivel, lo cual nos puede hacer las cosas algo más complicadas a la hora de hacer cálculos. Teniendo en cuenta que no necesitamos ese nivel extra, lo que

podemos hacer es eliminarlo para que nos quede un Dataframe del estilo del de “scores and fixtures”:

```
shooting.columns = shooting.columns.droplevel()
shooting
```

	Date	Time	Comp	Round	Day	Venue	Result	GF	GA	Opponent	...	Dist	FK	PK	PKatt	xG	npxG	npxG/Sh	G-xG	np:G-xG	Match Report
0	2022-08-05	20:00	Premier League	Matchweek 1	Fri	Away	W	2	0	Crystal Palace	...	14.6	1.0	0	0	1.0	1.0	0.10	0.0	0.0	Match Report
1	2022-08-13	15:00	Premier League	Matchweek 2	Sat	Home	W	4	2	Leicester City	...	13.0	0.0	0	0	2.7	2.7	0.16	1.3	1.3	Match Report
2	2022-08-20	17:30	Premier League	Matchweek 3	Sat	Away	W	3	0	Bournemouth	...	14.8	0.0	0	0	1.3	1.3	0.10	1.7	1.7	Match Report
3	2022-08-27	17:30	Premier League	Matchweek 4	Sat	Home	W	2	1	Fulham	...	15.5	1.0	0	0	2.6	2.6	0.12	-0.6	-0.6	Match Report
4	2022-08-31	19:30	Premier League	Matchweek 5	Wed	Home	W	2	1	Aston Villa	...	16.3	1.0	0	0	2.4	2.4	0.12	-0.4	-0.4	Match Report
5	2022-09-04	16:30	Premier League	Matchweek 6	Sun	Away	L	1	3	Manchester Utd	...	18.6	1.0	0	0	1.3	1.3	0.08	-0.3	-0.3	Match Report

Tras esto, ya tenemos nuestras dos tablas con todos los datos que vamos a necesitar para poder generar nuestros modelos y poder realizar predicciones. Nos quedaría un último paso, que consiste en unir estas dos tablas. Para ello, si nos fijamos en ambas tablas, tenemos la suerte de que tiene campos “Date” y “Time” que son únicos para cada partido (no puede haber dos partidos que se jueguen el mismo día) y también tienen el mismo formato, por lo que podemos unir ambas tablas usando el campo “Date”.

```
shooting.columns = shooting.columns.droplevel()
shooting
```

	Date	Time	Comp	Round	Day	Venue	Result	GF	GA	Opponent	...	Dist	FK	PK	PKatt	xG	npxG	npxG/Sh	G-xG	np:G-xG	Match Report
0	2022-08-05	20:00	Premier League	Matchweek 1	Fri	Away	W	2	0	Crystal Palace	...	14.6	1.0	0	0	1.0	1.0	0.10	0.0	0.0	Match Report
1	2022-08-13	15:00	Premier League	Matchweek 2	Sat	Home	W	4	2	Leicester City	...	13.0	0.0	0	0	2.7	2.7	0.16	1.3	1.3	Match Report
2	2022-08-20	17:30	Premier League	Matchweek 3	Sat	Away	W	3	0	Bournemouth	...	14.8	0.0	0	0	1.3	1.3	0.10	1.7	1.7	Match Report
3	2022-08-27	17:30	Premier League	Matchweek 4	Sat	Home	W	2	1	Fulham	...	15.5	1.0	0	0	2.6	2.6	0.12	-0.6	-0.6	Match Report
4	2022-08-31	19:30	Premier League	Matchweek 5	Wed	Home	W	2	1	Aston Villa	...	16.3	1.0	0	0	2.4	2.4	0.12	-0.4	-0.4	Match Report
5	2022-09-04	16:30	Premier League	Matchweek 6	Sun	Away	L	1	3	Manchester Utd	...	18.6	1.0	0	0	1.3	1.3	0.08	-0.3	-0.3	Match Report

Con esto, habremos obtenido todos los datos que necesitamos para nuestro proyecto. No obstante, estos son los datos para la temporada actual y solamente para el Arsenal, por lo tanto, para nuestro proyecto no es suficiente.

A partir de aquí, lo único que nos queda es realizar esta misma tarea para todos los equipos de la liga y para varias temporadas distintas. Realizar este mismo proceso para cada uno de los equipos y cada una de las temporadas requeriría de demasiado tiempo y esfuerzo innecesario, por lo que vamos a automatizar toda esta parte para que nuestro programa recoja los datos de cada temporada para cada uno de los 20 equipos que se encuentran en la primera división inglesa (EPL) en dicha temporada.

Para ello, primero vamos a definir una lista con los años de los que queremos obtener datos, en este caso, desde la temporada actual hasta la 2017, es decir 5 temporadas y lo que va de esta (una media temporada). Inicializamos también una lista que llamaremos “total_matches” que contendrá un Dataframe por equipo y por año y la variable “standings_url”, que contendrá la raíz del directorio de nuestra página y que iremos modificando para acceder a la página correspondiente de la tabla “scores and fixtures” o “shooting” para cada uno de los equipos y para cada una de las temporadas.

```
years = list(range(2022, 2016, -1))
total_partidos = []
standings_url = "https://fbref.com/en/comps/9/Premier-League-Stats"
years
```

```
[2022, 2021, 2020, 2019, 2018, 2017]
```

Por lo tanto, el bucle encargado de la obtención de los partidos será el siguiente:

```
for season in years:
    datos = requests.get(standings_url)
    soup = BeautifulSoup(datos.text)
    standings_table = soup.select('table.stats_table')[0]

    links = [l.get("href") for l in standings_table.find_all('a')]
    links = [l for l in links if '/squads/' in l]
    team_urls = [f"https://fbref.com{l}" for l in links]

    previous_season = soup.select("a.prev")[0].get("href")
    standings_url = f"https://fbref.com{previous_season}"

    for team_url in team_urls:
        team_name = team_url.split("/")[-1].replace("-Stats", "").replace("-", " ")
        datos = requests.get(team_url)
        matches = pd.read_html(datos.text, match="Scores & Fixtures")[0]
        soup = BeautifulSoup(datos.text)
        links = [l.get("href") for l in soup.find_all('a')]
        links = [l for l in links if l and 'all_comps/shooting/' in l]
        datos = requests.get(f"https://fbref.com{links[0]}")
        shooting = pd.read_html(datos.text, match="Shooting")[0]
        shooting.columns = shooting.columns.droplevel()
        try:
            team_data = matches.merge(shooting[["Date", "Sh", "SoT", "Dist", "FK", "PK", "PKatt"]], on="Date")
        except ValueError:
            continue
        team_data = team_data[team_data["Comp"] == "Premier League"]

        team_data["Season"] = season
        team_data["Team"] = team_name
        total_matches.append(team_data)
        time.sleep(2)
```

En este bucle, primero vamos recorriendo los años en orden inverso (empezando por la temporada actual y yendo hacia atrás) y vamos obteniendo el link referente a cada temporada. Seguidamente, iremos recorriendo, para la temporada en la que estemos, el segundo bucle, que es el que irá recorriendo los equipos de dicha temporada. Una vez aquí, para cada equipo obtendremos las tablas “scores and fixtures” y “shooting” y las uniremos, como hicimos en los pasos anteriores. Finalmente, obtendremos un Dataframe para cada equipo, para cada temporada, que uniremos a la lista de “total_matches”.

Dos cosas a tener en cuenta en este bucle son:

- Hacemos la parte del “split” para obtener los nombres de los equipos correctamente, ya que estos pueden venir con caracteres del tipo “/” o “-”.
- Al final ponemos un time.sleep(2). Esto lo que hace es que nuestro bucle pare dos segundos antes de seguir, esto lo hacemos para no saturar la página con demasiadas peticiones, lo cual es una buena práctica a la hora de hacer web scraping. En muchas páginas no

importa, pero otras podrían quedar colgadas o anular el acceso por hacer demasiadas peticiones al segundo.

Una vez terminado este bucle, tendremos nuestro Dataframe,que tendrá un tamaño de 4092 filas por 27 columnas. Esto se debe a que tenemos:

5 temporadas x 20 equipos x 38 jornadas = 3800 entradas

Si a estas entradas les sumamos que llevamos 15 jornadas de esta temporada, tenemos:

15 jornadas x 20 equipos = 300 entradas

Si nos fijamos, esto sumaría un total de 4100 entradas, por lo que tendríamos 8 entradas que no están. Esto lo veremos en profundidad en el siguiente punto.

4.2. Generación del Dataset definitivo

Una vez hemos obtenido los datos de nuestro modelo mediante web scraping, es hora de estudiar un poco el Dataset y prepararlo para el estudio que vamos a realizar. Lo primero que vamos a ver es qué forma tiene:

```
PL_Dataframe.info(verbose = False)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4092 entries, 1 to 16
Columns: 27 entries, Date to Team
dtypes: float64(13), int64(1), object(13)
memory usage: 895.1+ KB
```

Podemos observar que el dataframe tiene 4092 entradas y 27 columnas como comentamos en el punto anterior. Cada una de estas entradas es un partido de la Premier League, es decir, tenemos recogidas en esas 4092 entradas cada uno de

los partidos de cada uno de los equipos de la Premier League durante 5 temporadas, más los partidos disputados de la temporada actual.

Ahora queremos ver qué equipos se encuentran en el Dataframe y cuantos registros tienen:

```
[12]: PL_Dataframe["Team"].value_counts()

[12]: West Ham United            205
      Newcastle United           205
      Southampton                205
      Leicester City             205
      Everton                    205
      Tottenham Hotspur          205
      Chelsea                    204
      Brighton and Hove Albion   204
      Manchester United          204
      Arsenal                    204
      Manchester City            204
      Liverpool                  204
      Crystal Palace             204
      Burnley                    190
      Wolverhampton Wanderers    167
      Watford                    152
      Aston Villa                129
      Bournemouth                129
      Fulham                      91
      Leeds United                90
      Sheffield United            76
      Huddersfield Town           76
      West Bromwich Albion        76
      Norwich City                76
      Brentford                   53
      Swansea City                38
      Stoke City                  38
      Cardiff City                38
      Nottingham Forest           15
      Name: Team, dtype: int64
```

Hay un total de 29 equipos, que es el número total de equipos que han estado en la EPL durante estos últimos 6 años. Al haber ascensos y descensos, vemos que no todos los equipos tienen el mismo número de entradas. De la cantidad de entradas que tenemos, podemos deducir que equipos como el West Ham, el Newcastle o el Everton han estado durante todas las temporadas y que, el Nottingham Forest ha estado solamente la última temporada.

Esto nos complicará un poco las cosas a la hora de trabajar con el Dataset, ya que habrá de algunos equipos (aquellos que han estado todas las temporadas) que tendremos una gran cantidad de información y habrá otros de los que apenas tendremos datos (como el Swansea, Stoke city o Nottingham Forest, por ejemplo).

Por otro lado, queremos ver también cuántas entradas tiene cada jornada, para poder observar los 8 datos que vimos anteriormente que faltaban:

```
PL_Dataframe["Round"].value_counts()
```

```
Matchweek 6     120
Matchweek 1     120
Matchweek 16    120
Matchweek 3     120
Matchweek 4     120
Matchweek 11    120
Matchweek 10    120
Matchweek 2     120
Matchweek 9     120
Matchweek 15    120
Matchweek 14    120
Matchweek 13    120
Matchweek 5     120
Matchweek 12    118
Matchweek 8     114
Matchweek 25    100
Matchweek 23    100
```

Como podemos observar, de las jornadas 1 a la 15, casi todas tienen 120 entradas (6 temporadas x 20 equipos), que corresponden con las jornadas que ya se han disputado esta temporada. No obstante, podemos apreciar que en las jornadas 12 y 8 faltan 2 y 6 entradas respectivamente, es decir, que esas jornadas hubo 1 y 3 partidos que no se pudieron disputar, por lo que en estas jornadas estarían aquellos datos que no tenemos.

No obstante, si nos vamos a la clasificación actual:

Rk	Squad	MP	W	D	L	GF	GA	GD	Pts	Pts/MP	xG	xGA	xGD	xGD/90	Last 5
1	Arsenal	14	12	1	1	33	11	+22	37	2.64	26.2	11.8	+14.3	+1.02	W D W W W
2	Manchester City	14	10	2	2	40	14	+26	32	2.29	27.6	11.2	+16.4	+1.17	L W W W L
3	Newcastle Utd	15	8	6	1	29	11	+18	30	2.00	24.3	14.3	+9.9	+0.66	W W W W W
4	Tottenham	15	9	2	4	31	21	+10	29	1.93	24.1	16.5	+7.6	+0.50	L L W L W
5	Manchester Utd	14	8	2	4	20	20	0	26	1.86	18.8	17.3	+1.5	+0.11	W D W L W
6	Liverpool	14	6	4	4	28	17	+11	22	1.57	24.2	19.5	+4.7	+0.34	W L L W W
7	Brighton	14	6	3	5	23	19	+4	21	1.50	21.5	15.9	+5.6	+0.40	D L W W L
8	Chelsea	14	6	3	5	17	17	0	21	1.50	15.4	19.3	-3.9	-0.28	D D L L L
9	Fulham	15	5	4	6	24	26	-2	19	1.27	19.6	27.3	-7.6	-0.51	W W D L L
10	Brentford	15	4	7	4	23	25	-2	19	1.27	20.3	20.3	0.0	0.00	D L D D W
11	Crystal Palace	14	5	4	5	15	18	-3	19	1.36	14.0	18.3	-4.3	-0.31	W L W W L
12	Aston Villa	15	5	3	7	16	22	-6	18	1.20	17.1	19.3	-2.2	-0.14	L W L W W
13	Leicester City	15	5	2	8	25	25	0	17	1.13	17.6	22.2	-4.6	-0.31	W W L W W
14	Bournemouth	15	4	4	7	18	32	-14	16	1.07	11.6	24.2	-12.6	-0.84	L L L L W
15	Leeds United	14	4	3	7	22	26	-4	15	1.07	20.3	20.2	+0.1	+0.01	L L W W L
16	West Ham	15	4	2	9	12	17	-5	14	0.93	18.6	15.5	+3.1	+0.20	L W L L L
17	Everton	15	3	5	7	11	17	-6	14	0.93	15.4	25.4	-10.0	-0.66	L W D L L
▼ 18	Nott'ham Forest	15	3	4	8	11	30	-19	13	0.87	15.3	24.1	-8.8	-0.59	D W L D W
▼ 19	Southampton	15	3	3	9	13	27	-14	12	0.80	14.7	19.2	-4.5	-0.30	W D L L L
▼ 20	Wolves	15	2	4	9	8	24	-16	10	0.67	14.9	19.6	-4.8	-0.32	L L D L L

podemos observar que hay 8 equipos que están con un partido menos, por lo tanto, parece que nuestro Dataset tiene todos los datos correctos y que hemos hecho scrape de la página correctamente.

Una vez explorados los datos, vamos a preparar el Dataset para modelizarlo. Lo primero de lo que nos encargaremos será de comprobar las variables que tenemos y ver si estas son las adecuadas para trabajar con ellas y, en caso de no serlo, realizaremos cambios.

Las columnas con las que cuenta nuestro Dataset son las siguientes:

```
PL_Dataframe.dtypes
```

```
Date             object
Time             object
Comp             object
Round            object
Day              object
Venue            object
Result           object
GF              float64
GA              float64
Opponent         object
xG              float64
xGA             float64
Poss            float64
Attendance      float64
Captain          object
Formation        object
Referee          object
Match Report     object
Notes           float64
Sh              float64
SoT             float64
Dist            float64
FK              float64
PK              float64
PKatt           float64
Season            int64
Team             object
```

Con el comando “dtypes” podemos ver las columnas de nuestro dataset y además podremos observar el tipo de estas, esto nos ayudará a entender el tipo de datos que contienen y a cambiar el tipo de algunas de ellas para poder ajustar el tipo al que requiere nuestro modelo de Machine Learning.

De estas, eliminaremos “Comp”, que se refiere a la competición en la que estamos, ya que solo hemos tomado partidos de la EPL y también eliminaremos “Match Report” y “notes” ya que ambas no aportan ningún tipo de datos.

A partir de aquí usaremos una serie de variables que describiremos a continuación:

- Round: Hace referencia a la jornada en la que se disputa el partido.
- Day: El día de la semana en el que se disputa el partido.
- Venue: Si el partido se disputa jugando en su estadio o en el del rival.
- Opponent: El nombre del rival
- Referee: Nombre del árbitro
- Season: Temporada en la que nos encontramos:

- Team: Equipo que juega el partido.

Todas estas estadísticas las podemos obtener antes de los partidos, cuando se fija una jornada se sabe qué equipos van a jugar, la fecha o quién va a ser el árbitro. Más adelante esto será importante.

Por otro lado, tenemos las estadísticas que se calculan una vez acabado el partido, que son:

- GF: Goles a favor
- GA: Goles en contra
- xG: Goles esperados a favor
- xGA: Goles esperados en contra
- Poss: Posesión del balón (porcentaje del tiempo total que un equipo tuvo el balón)
- Attendance: Asistencia al estadio
- Captain: Capitán del equipo
- Formation: Esquema de juego del equipo
- Sh: Tiros realizados
- SoT: Tiros a puerta (aquellos cuya trayectoria inicial va hacia la portería)
- FK: Faltas tiradas.
- PK: Penaltis tirados
- PKatt: Penaltis ejecutados

Todos estos datos se van recolectando durante el transcurso del partido y se obtienen de forma definitiva una vez acaba este.

A partir de aquí y conociendo las variables que tenemos, lo que haremos será cambiar el tipo de algunas de ellas, para transformarlas a variables que sean aptas para nuestros modelos.

Primero transformaremos las siguientes variables:

```
PL_Dataframe["Date"] = pd.to_datetime(PL_Dataframe["Date"])
PL_Dataframe["Venue_Code"] = PL_Dataframe["Venue"].astype("category").cat.codes
PL_Dataframe["Opponent_Code"] = PL_Dataframe["Opponent"].astype("category").cat.codes
PL_Dataframe["Hour"] = PL_Dataframe["Time"].str.replace(":.*", "", regex=True).astype("int")
PL_Dataframe["Day_Code"] = PL_Dataframe["Date"].dt.dayofweek
PL_Dataframe["Season_Code"] = PL_Dataframe["Season"].astype("category").cat.codes
PL_Dataframe["Round_Code"] = PL_Dataframe["Round"].astype("category").cat.codes
PL_Dataframe["Referee_Code"] = PL_Dataframe["Referee"].astype("category").cat.codes
PL_Dataframe["Team_Code"] = PL_Dataframe["Team"].astype("category").cat.codes
```

Las cuales las convertiremos en tipo integer, lo que nos permitirá usarlas en nuestro modelo. Para la hora simplemente hemos retirado todo aquello que no es el número de la hora y hemos convertido la hora en integer. Para el resto, lo que hemos hecho es una relación entre los valores de las variables y unos integers, de tal manera que, por ejemplo, para la variable “Opponent” hemos asignado a cada equipo un único número integer y hemos metido esto en la variable “Opponent_Code”. Esto lo hemos realizado mediante el comando “astype(“category”).cat.codes” que se encarga de transformar nuestra variable en categórica según los diferentes tipos de valores que tiene y luego le asignará un código a cada uno de estos valores.

Los modelos de Machine Learning trabajan con variables del tipo numérico, por lo tanto, todas las variables que son del tipo “float64” o “int64” son aptas para nuestro modelo, mientras que aquellas que son del tipo “object” no nos sirven, es por ello que las variables de este último tipo las hemos cambiado buscando la manera de asociarlas a una variable numérica, mientras que hemos dejado el resto tal y como estaban.

Finalmente, solo nos queda definir nuestra variable target, que es aquella que queremos predecir, en este caso será el resultado. Para esta variable, vamos a plantear el problema de dos maneras distintas:

- Definir el problema prediciendo el resultado del equipo, es decir, si va a ganar, perder o empatar. En este caso, nuestro sistema definirá nuestro target como 0 en caso de que el equipo empate, 1 en caso de perder y 2 en caso de ganar.

- Definir el problema de tal manera que queremos saber si el equipo va a ganar o no. En este caso, nuestro target se definirá como 0 en caso de empate o derrota y 1 en caso de victoria.

Esto nos originará dos problemas diferentes y con resultados distintos.

4.3. Modelos a aplicar

Antes de explicar los modelos que vamos a aplicar en nuestro proyecto, hay que observar el tipo de problema que tenemos.

Nuestro problema consiste en predecir el resultado de los partidos de la English Premier League, en función de los datos que hemos obtenido y recopilado en el dataset que hemos estado trabajando anteriormente, por lo tanto, estamos ante un **problema de clasificación**, ya que nuestro objetivo es predecir una variable cualitativa Y, en nuestro caso, el resultado del partido . Si observamos bien, muchas de nuestra variables, como por ejemplo, puede ser la variable de los oponentes (“Opponent_Code”) son variables que no tiene por qué tener una relación lineal con el resto de variable, ya que el valor de esta no está linealmente relacionado con un mejor o peor desempeño (como por ejemplo sí puede ser el número de faltas o el número de goles).

Estas variables son de tipo *integer* pero la realidad es que ese código es simplemente una relación que hemos establecido para que nuestro modelo pueda usar ese tipo de variables, pero no tienen un significado numérico más allá del de designar un valor concreto de la categoría (por ejemplo, el código 15 se refiere al Tottenham Hotspurs, pero no quiere decir que este sea ni mejor ni peor que un equipo con el código 3, ese 15 solo hace referencia a que, para la categoría de “Opponent_Code” el código 15 se refiere al Tottenham Hotspurs, nada más).

Debido a esto, tenemos que descartar cualquier modelo que sea lineal o que utilice relaciones lineales. Por lo tanto, modelos como el de regresión logística quedan descartados.

Por otro lado, al tener en torno a 20 dimensiones, modelos del tipo KNN (K Nearest Neighbours) tampoco son adecuados para este tipo de problema, ya que el cálculo de vecinos con las distancias no queda muy claro.

Los modelos que vamos a aplicar son los de Random Forests y XGBoost. Estos son unos algoritmos que están formados por combinación de otros algoritmos más sencillos, a este tipo de algoritmos se les conoce como **algoritmos ensamblados.**

Dentro de los algoritmos ensamblados, existen una serie de algoritmos que se conocen como algoritmos de *bagging*, que es el tipo de algoritmo al que pertenece Random Forests. Esta clase de algoritmos utilizan otros algoritmos simples corriendo en paralelo, de manera que se aprovecha de varios algoritmos siendo ejecutados a la vez para eliminar sesgos utilizando la independencia que existe dentro de los algoritmos simples que usa este tipo de modelos.

Por otro lado, tenemos los algoritmos de Boosting, que son parecidos a los anteriores, también usan algoritmos más sencillos, pero en este caso, de manera secuencial, esto provoca que el propio algoritmo vaya aprendiendo de los errores anteriores, para así poder proporcionar un resultado mejor.

Para entender mejor estos algoritmos, hace falta aclarar dos cosas.

4.3.1. Muestreo bootstrap

Lo primero es saber qué es el muestreo *bootstrap.* Como podemos ver en Davison & Hinkley (1997), el bootstrap es un método estadístico que sirve para aproximar las

características de una distribución en el muestreo de un estadístico. Es decir, nosotros tomaremos una muestra inicial de nuestra población, a partir de aquí, volveremos a tomar una muestra de nuestra muestra inicial, la cual usaremos para obtener el estadístico que necesitemos y será el que usemos como estimador de la población. Obviamente, al ser el estimador obtenido de la muestra de una muestra, este estimador no será muy preciso. Hay que tener en cuenta que, cuando se realiza este tipo de muestreos, se tiene que hacer con reposición, es decir, cada vez que obtengo un elemento para la segunda muestra, este ha sido devuelto a la primera, pudiendo volver a ser seleccionable para la segunda muestra y dando lugar a que en nuestra segunda muestra unelemente pueda aparecer repetido múltiples veces.

Realizando este proceso de forma iterativa muchas veces obtendremos un gran número de estimadores, que los usaremos para generar lo que se denomina la *distribución de bootstrap*, de la cual podremos obtener el estadístico centrar y su desviación para obtener el valor que queríamos estimar y la confianza de este.

Obviamente es un método que no tiene sentido sin una capacidad alta de computación y sin una gran cantidad de datos, ya que su precisión aumenta cuantas más repeticiones y más subgrupos se realicen.

4.3.2. Árboles de decisión

Lo segundo que hay que conocer es qué es un *árbol de decisión* o *decision tree.*

Este es un tipo de algoritmo de aprendizaje supervisado en el que el modelo se constituye de dos partes: las hojas y las ramas.

Las hojas (leaves) representan a cada una de las etiquetas de clase, es decir, la decisión final del árbol, las opciones que tendrán estas hojas coincidirán con los valores posibles de nuestra variable objetivo o *target.*

Las ramas (branches o nodos) conforman las reglas básicas que se aplican para llegar a una decisión final u otra.

4.3.3. Random Forests

Para Breiman (2001), el algoritmo de Random Forests se define como un conjunto de clasificadores de tal manera que:

- Tiene árboles de decisión como algoritmo base.
- Cada uno de estos árboles se entrena a partir de un subconjunto de datos usando un muestreo de tipo *bootstrap.*
- Cada uno de estos árboles se entrenará también con un subconjunto de las variables.

En resumen, como nuestro problema es de clasificación, nuestro algoritmo funciona como una votación, se ejecutan varios algoritmos a la vez, provocando resultados diferentes. A partir de aquí, nuestro algoritmo seleccionará como buena “aquella salida que tenga más votaciones”. Por lo que podemos suponer que lo que “escoja la mayoría” será la solución correcta.

Usaremos este algoritmo primero y principalmente porque **no es un algoritmo lineal**. Esta parte es imprescindible, ya que hemos asociado nombres de equipos, de árbitros, etc, a números y no queremos que nuestro sistema recoja relaciones lineales, ya que en este caso, el código 1 se refiere a un equipo y el 2 a otro, pero eso no significa que el equipo 1 sea mejor que el 2 ni todo lo contrario.

Los parámetros de los que más nos vamos a preocupar en este modelo son los siguientes:

- Número de estimadores: Aparece en nuestro modelo como “n_estimators”, que son la cantidad de árboles que conformarán nuestro modelo.
- Mínimo número de muestras para hacer separación: Que aparecerá en nuestro modelo como “min_samples_split”, que es el mínimo número de muestras que se podrá tener en un árbol para realizar una bifurcación.
- Random State: Que aparecerá como “random_state” y lo inicializamos a 1 para que siempre que realicemos el mismo experimento obtengamos los mismos resultados.

Las ventajas que nos va a ofrecer este modelo son básicamente:

- Es un algoritmo que no tiende a sobreajustar.
- Es un algoritmo con un gran rendimiento y muy rápido de calcular, debido a que es una combinación de Decision Trees corriendo en paralelo.

La gran desventaja de este modelo es que no es interpretable, pero en nuestro caso no necesitamos una interpretación de nuestro modelo, solo necesitamos que realice la predicción correctamente, por lo que en nuestro caso no importa esa desventaja.

Finalmente, hay que hacer una consideración antes de aplicar el modelo. Tenemos una gran cantidad de predictores a nuestra disposición, sin embargo, hay que tener en cuenta, como comentamos anteriormente, que algunas de ellas las podemos obtener antes de que comience el partido y otras las obtendremos durante el transcurso de este y al final. Por lo tanto, empezaremos utilizando las primeras, ya que, al ser un modelo predictivo, podemos tratar de predecir el resultado de partido sabiendo qué dos equipos se enfrentan entre ellos o quién es el árbitro, pero sin embargo no lo podemos saber por el número de goles, ya que ese dato no lo tendremos hasta que acabe el partido.

4.3.4. XGBoost

Dentro de los algoritmos ensamblados, también tenemos otro tipo, los conocidos como algoritmos de boosting. Este tipo de algoritmos, a diferencia de los de bagging, lo que hacen es correr diferentes modelos simples de forma secuencial, en vez de hacerlo en paralelo. La ventaja que nos aporta esto es que los algoritmos "aprenden de los errores de sus predecesores", es decir, al ejecutarse secuencialmente, tienen en cuenta los errores cometidos por los algoritmos ejecutados anteriormente y eso les lleva a mejorar sus resultados.

El algoritmo que nos va a ocupar en este trabajo es el XGBoost (Extreme Gradient Boosting), es uno de los algoritmos que ha demostrado tener mejor rendimiento y está muy optimizado en términos de velocidad de entrenamiento, consiste en un sistema de decision trees ejecutándose linealmente (parecido a Random Forests , pero en vez de ejecutarse en paralelo, lo hará en secuencial).

El modelo consta de multitud de parámetros, pero nosotros solo nos vamos a centrar en aquellos más importantes:

- max_depth: O, lo que es lo mismo, la profundidad máxima de un árbol. Cuanto mayor sea este número, más complejo será nuestro modelo y, por lo tanto, más tenderá a sobreajustar.
- eta: Este es un peso que se le da a la acción anterior para tener en cuenta la siguiente, esto nos permite tomar más o menos en cuenta la iteración anterior de nuestro modelo. Es una contracción del tamaño de paso utilizada para evitar el sobreajuste.
- subsample: Proporción de submuestras de entrenamiento.
- Lambda y alpha: Regularizaciones L2 y L1 para los pesos. Aumentar estos valores hará que el modelo sea más conservativo, ayudan a no sobreajustar.

Los puntos fuertes de este modelo son que rellena perfectamente los huecos así como trata muy bien los valores atípicos, por lo que no hace falta una labor de preprocesado para usarlo. Al igual que Random Forests, tiene un alto rendimiento, por lo que su ejecución será más rápida que la de otros modelos.

Por estas razones y, teniendo en cuenta que no necesitamos interpretación del modelo (ya que este modelo es de tipo “caja negra” y no deja clara la interpretación del modelo con respecto a nuestro problema) usaremos XGBoost para nuestro proyecto.

4.3.5. Evaluación de los modelos

Finalmente, para poder comprobar el desempeño de los modelos empleados, usaremos una métrica del paquete *sklearn* llamada *accuracy_score*.

Esta métrica es muy útil en modelos de clasificación con etiquetas múltiples, como es nuestro caso, ya que compara los resultados esperados de nuestro modelo con el resultado actual obtenido y nos devuelve el porcentaje de acierto.

Hemos escogido esta muestra ya que realmente lo único que nos interesa es ser precisos en la obtención del resultado, ya que otras que pesaran más un resultado que otro o aquellas que lo hacen mediante error cuadrático medio no son aplicables en nuestro problema ya que este consiste en adivinar el resultado de un partido, no el marcador (en este caso, no es lo mismo predecir un 3-0 y que el partido quede 4-0 o 1-0, se puede entender que si el resultado final es 4-0, no nos hemos quedado tan lejos como sí nos habríamos quedado con el 1-0).

La fórmula que utiliza nuestra métricas, por lo tanto es la siguiente:

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

Donde “nsamples” es el número de muestras, cada una de las $y\hat{}_i$ son los valores predichos por el modelo y cada una de las y_i son los valores reales de nuestro

target y, por últimos, la función **1** es aquella que asigna el valor 1 si ambos resultados son iguales y 0 si no lo son (Toda esta información la podemos encontrar en el manual de sklearn: https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score) .

5. Resultados

Como hemos explicado anteriormente, hemos aplicado nuestros modelos de Random Forests y de XGBoost a dos tipos de problema, uno en el que tenemos como target tres opciones (victoria - empate - derrota) y otro en el que tenemos dos (victoria y empate o derrota).

5.1. Random Forests con W-D-L

Lo primero que haremos será importar los módulos necesarios:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

A partir de aquí, dividiremos nuestro Dataset entre el training y el test, lo haremos tomando los partidos de este año como test y los partidos de años anteriores como el training, lo cual nos deja aproximadamente un 80% - 20% entre training y test.

Nuestro problema es un problema temporal, es decir, estamos prediciendo el futuro con valores pasados, por lo tanto, es importante que todos los valores del train set sean anteriores en tiempo a los valores del test set, ya que no tiene sentido tratar de predecir el resultado de un partido con resultados que aún no se habían producido.

Definiremos también las variables que vamos a usar como predictores:

```
train = PL_Dataframe[PL_Dataframe["Date"] < '2022-01-01']
test = PL_Dataframe[PL_Dataframe["Date"] >= '2022-01-01']
predictors = ["Venue_Code", "Opponent_Code", "Hour", "Day_Code", "Round_Code", "Referee_Code", "Team_Code"]
```

A partir de aquí, lo que hemos hecho ha sido probar la combinación de cada uno de los predictores con los parámetros optimizados para el modelo usando "GridSearchCV", que nos permite encontrar los parámetros óptimos de un set de

parámetros previamente dado, este realiza una serie de validaciones cruzadas para obtener el valor óptimo de los parámetros del modelo.

Para ello, hemos ejecutado el modelo optimizando los parámetros y luego hemos comprobado el rendimiento de cada uno de ellos para las distintas combinaciones de predictores.

De esta manera hemos descartado usar la temporada como predictor, ya que nos reducía la precisión del modelo y hemos mantenido el resto de predictores, arrojando como modelo con mayor precisión el siguiente:

```
clf_grid = GridSearchCV(rf, params, cv=3, n_jobs=-1)
```

```
clf_grid.fit(train[predictors], train["Target"])
```

```
GridSearchCV(cv=3,
             estimator=RandomForestClassifier(min_samples_split=49,
                                              n_estimators=99, random_state=1),
             n_jobs=-1,
             param_grid={'min_samples_split': [5, 6, 7, 8, 9, 10, 11, 12, 13,
                                               14, 15, 16, 17, 18, 19, 20, 21,
                                               22, 23, 24, 25, 26, 27, 28, 29,
                                               30, 31, 32, 33, 34, ...],
                         'n_estimators': [10, 11, 12, 13, 14, 15, 16, 17, 18,
                                          19, 20, 21, 22, 23, 24, 25, 26, 27,
                                          28, 29, 30, 31, 32, 33, 34, 35, 36,
                                          37, 38, 39, ...]})
```

```
clf_grid.best_params_
```

```
{'min_samples_split': 37, 'n_estimators': 74}
```

en el que la función “fit” se encarga de encajar nuestros datos con parámetros optimizados en nuestro modelo, es decir, entrenará nuestro modelo con los datos de entrenamiento (train[predictors]) y los comparará con respecto a las salidas de los datos de entrenamiento (train[“Target”]).

El objeto que devuelve, en este caso “clf_grid”, contiene los parámetros optimizados para este modelo, esto lo podemos ver cuando ejecutamos “clf_grid.best_params_”.

Con estos datos, ya solo tenemos que aplicar nuestro modelo a los datos de test y comprobar la precisión de nuestro modelo, que nos devuelve unos valores de:

```
accuracy = accuracy_score(test["Target"], preds)
```

```
accuracy
```

```
0.5029154518950437
```

más de un 50%. Si pensamos en que cada partido hay tres posibles resultados y pensamos en cada uno de ellos como resultados equiprobables, habría un 33% de probabilidad de acertar el resultado de un partido al azar, por lo que nuestro modelo estaría en torno al 17% por encima de esto.

5.2. Random Forests con W-D/L

En este caso, como explicamos anteriormente, queremos predecir si un equipo va a ganar o no. Es decir, empate o derrota contará como el mismo resultado.

Tanto los sets para train y test como los predictores que vamos a usar son los mismos que los del modelo anterior, lo único que cambia en este caso es el target, que solo tendrá dos opciones (ganar o no ganar).

Repitiendo las mismas operaciones de antes obtendremos el siguiente resultado:

```
clf_grid.fit(train[predictors], train["Target"])
```

```
GridSearchCV(cv=3,
             estimator=RandomForestClassifier(min_samples_split=10,
                                              n_estimators=50, random_state=1),
             n_jobs=-1,
             param_grid={'min_samples_split': [5, 6, 7, 8, 9, 10, 11, 12, 13,
                                               14, 15, 16, 17, 18, 19, 20, 21,
                                               22, 23, 24, 25, 26, 27, 28, 29,
                                               30, 31, 32, 33, 34, ...],
                         'n_estimators': [10, 11, 12, 13, 14, 15, 16, 17, 18,
                                          19, 20, 21, 22, 23, 24, 25, 26, 27,
                                          28, 29, 30, 31, 32, 33, 34, 35, 36,
                                          37, 38, 39, ...]})
```

```
clf_grid.best_params_
```

```
{'min_samples_split': 41, 'n_estimators': 75}
```

```
preds = clf_grid.predict(test[predictors])
```

y una precisión de

```
accuracy = accuracy_score(test["Target"], preds)
```

```
accuracy
```

```
0.651702786377709
```

más de un 65%, si tenemos en cuenta que, en este caso, solo hay dos opciones posibles, ganar o no ganar, suponiendo que ambas opciones son equiprobables tendríamos que la probabilidad de acertar al azar sería de un 50%, por lo que habríamos estado un 15% por encima.

5.3. XGBoost con W-D-L

Como siempre, lo primero que haremos será importar los módulos necesarios:

```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
```

Una vez los tenemos, vamos a definir los parámetros que utilizaremos para obtener el modelo con mejores parámetros gracias a GridSearchCV:

```
estimator = XGBClassifier(
    objective= 'binary:logistic',
    nthread=4,
    seed=42
)
```

```
parameters = {
    'max_depth': range (2, 10, 1),
    'n_estimators': range(60, 220, 40),
    'learning_rate': [0.1, 0.01, 0.05]
}
```

```
grid_search = GridSearchCV(
    estimator=estimator,
    param_grid=parameters,
    n_jobs = 10,
    cv = 10,
    verbose=True
)
```

Tras esto, ya podemos ajustar nuestro modelo:

```
grid_search.fit(train[predictors], train["Target"])
```

```
Fitting 10 folds for each of 96 candidates, totalling 960 fits
[Parallel(n_jobs=10)]: Using backend LokyBackend with 10 concurrent workers.
[Parallel(n_jobs=10)]: Done  30 tasks      | elapsed:   13.0s
[Parallel(n_jobs=10)]: Done 180 tasks      | elapsed:   55.5s
[Parallel(n_jobs=10)]: Done 430 tasks      | elapsed:  3.0min
[Parallel(n_jobs=10)]: Done 780 tasks      | elapsed:  5.9min
[Parallel(n_jobs=10)]: Done 960 out of 960 | elapsed:  8.0min finished
GridSearchCV(cv=10, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=None, gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, ma...
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None, nthread=4,
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     predictor=None, random_state=None, ...),
             iid='deprecated', n_jobs=10,
             param_grid={'learning_rate': [0.1, 0.01, 0.05],
                         'max_depth': range(2, 10),
                         'n_estimators': range(60, 220, 40)},
```

y obtener los mejores parámetros:

```
grid_search.best_params_
```

```
{'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}
```

obteniendo una precisión de

```
accuracy = accuracy_score(test["Target"], preds)
```

```
accuracy
```

```
0.5247813411078717
```

más del 52%, un poco superior a lo obtenido con Random Forests.

5.4. XGBoost con W-D/L

En este caso, todo es igual que el apartado anterior, solo cambiaremos nuestro target, que en vez de tener las 3 opciones, solo tendrá 2 (victoria o no victoria).

Volvemos a buscar los mejores parámetros para el modelo:

```
grid_search.fit(train[predictors], train["Target"])
```

```
Fitting 10 folds for each of 96 candidates, totalling 960 fits
[Parallel(n_jobs=10)]: Using backend LokyBackend with 10 concurrent workers.
[Parallel(n_jobs=10)]: Done  30 tasks      | elapsed:    9.4s
[Parallel(n_jobs=10)]: Done 180 tasks      | elapsed:   23.5s
[Parallel(n_jobs=10)]: Done 430 tasks      | elapsed:   59.5s
[Parallel(n_jobs=10)]: Done 780 tasks      | elapsed:  1.8min
[Parallel(n_jobs=10)]: Done 960 out of 960 | elapsed:  2.4min finished
GridSearchCV(cv=10, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=None, gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, ma...
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None, nthread=4,
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     predictor=None, random_state=None, ...),
             iid='deprecated', n_jobs=10,
             param_grid={'learning_rate': [0.1, 0.01, 0.05],
                         'max_depth': range(2, 10),
                         'n_estimators': range(60, 220, 40)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)
```

A partir de aquí, al igual que antes, ajustaremos nuestro modelo y obtendremos su precisión, obteniendo como resultado:

```
grid_search.best_params_
```

```
{'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 180}
```

```
preds = grid_search.predict(test[predictors])
```

```
accuracy = accuracy_score(test["Target"], preds)
```

```
accuracy
```

```
0.6865889212827988
```

una precisión de casi el 69%, mejorando también algo los resultados obtenidos para el mismo problema con Random Forests.

5.5. Cálculo de medias

Hasta aquí hemos trabajado con los datos de los que podíamos disponer antes del partido, sin embargo, ¿Qué podemos hacer si queremos usar datos como el número de faltas, la posesión o los goles anotados?

A priori, este tipo de datos nos pueden ayudar mucho más que los anteriores, ya que están directamente relacionados con lo que sucede en el juego, por lo que podríamos pensar que pueden ser mejores predictores que la hora del partido o que quién es el árbitro. Sin embargo, no podemos utilizar variables de entrenamiento que luego no podemos usar para predecir, ya que, hasta que no termina el partido, no podemos conocer el número total de corners o de faltas, por ejemplo.

Para poder usar estos datos en nuestro modelo, lo que vamos a hacer es un cálculo medio de la racha del equipo, es decir, vamos a calcular una media de los valores desempeñados en los últimos partidos y usaremos esas medias para ser capaces de predecir los valores del partido que se va a disputar y así poder realizar la predicción.

Para hacer el cálculo, hemos escogido hacer el promedio de las variables de:

- Goles a favor (GF)
- Goles en contra (GA)
- Disparos (Sh)
- Disparos a puerta (SoT)
- Distancia recorrida (Dist)
- Faltas (FK)
- Penaltis (PK)
- Intentos de penaltis (PKatt)

Como todas estas variables son de tipo “float64”, no es necesario hacer ninguna transformación para introducirla en nuestro modelo. Por lo tanto, lo primero que haremos será definir una función que se encargue de hacer las medias de estos datos para los n partidos anteriores:

```
def calculo_medias(group, cols, new_cols, n):
    group = group.sort_values("Date")
    stats_medias = group[cols].rolling(n, closed='left').mean()
    group[new_cols] = stats_medias
    group = group.dropna(subset=new_cols)
    return group
```

```
cols = ["GF", "GA", "Sh", "SoT", "Dist", "FK", "PK", "PKatt"]
new_cols = [f"{c}_rolling" for c in cols]
```

En esta función, lo que hacemos es ordenar los valores por fechas, calcular la media de la estadística concreta (GF, GA, Sh, etc) para una ventana de valor “n” y obtendremos las nuevas columnas. Es importante el parámetro “closed=’left’” ya que esto hace que no tomemos el último valor de la ventana, es decir, el del partido actual, ya que si no, en el cálculo de la media estaríamos usando los valores del partido que aún no se ha disputado, lo cual no tiene sentido, por lo tanto, dejando este parámetro así, para hacer el cálculo de la media para los goles a favor de la jornada 35 y con una ventana n=3, usará los valores de goles a favor de las jornadas 33 y 34.

A partir de aquí, lo que haremos será aplicar esta función a todos nuestros equipos y de nuevo volver a aplicar los mismos modelos:

```
matches_rolling = matches.groupby("Team").apply(lambda x: calculo_medias(x, cols, new_cols, 3)
```

En este caso, hemos tomado el rango de partidos a contar hacia atrás de 3.
Ahora, volveremos a aplicar nuestro modelo para nuestros dos casos, el del sistema W-D-L y el del sistema W-D/L.

5.5.1. Cálculo de medias con W-D-L

Aplicando de nuevo nuestro modelo de **Random Forests** hemos obtenido el siguiente resultado:

```
matches_rolling = matches_rolling.droplevel('Team')
```

```
matches_rolling.index = range(matches_rolling.shape[0])
```

```
def make_predictions(data, predictors):
    train = data[data["Date"] < '2022-01-01']
    test = data[data["Date"] > '2022-01-01']
    rf.fit(train[predictors], train["Target"])
    preds = rf.predict(test[predictors])
    combined = pd.DataFrame(dict(actual=test["Target"], predicted=preds), index=test.index)
    error = accuracy_score(test["Target"], preds)
    return combined, error
```

```
combined, error = make_predictions(matches_rolling, predictors + new_cols)
```

```
error
```

```
0.6017699115044248
```

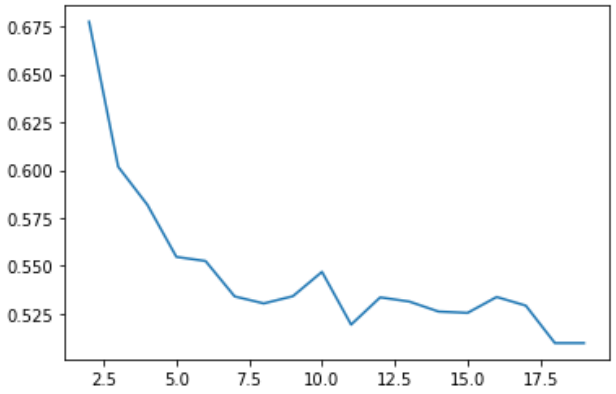
Es decir, ahora tenemos un 60% de error lo que, comparado con el 50% obtenido anteriormente supone una mejora sustancial del 10%.

No obstante, esto está hecho para un parámetro concreto, así que vamos a probar para un cierto rango:

```
for n in list(range(2,20)):
    matches_rolling = matches.groupby("Team").apply(lambda x: rolling_averages(x, cols, new_cols, n))
    matches_rolling = matches_rolling.droplevel('Team')
    combined, error = make_predictions(matches_rolling, predictors + new_cols)
    error_list.append(error)
    n_list.append(n)
```

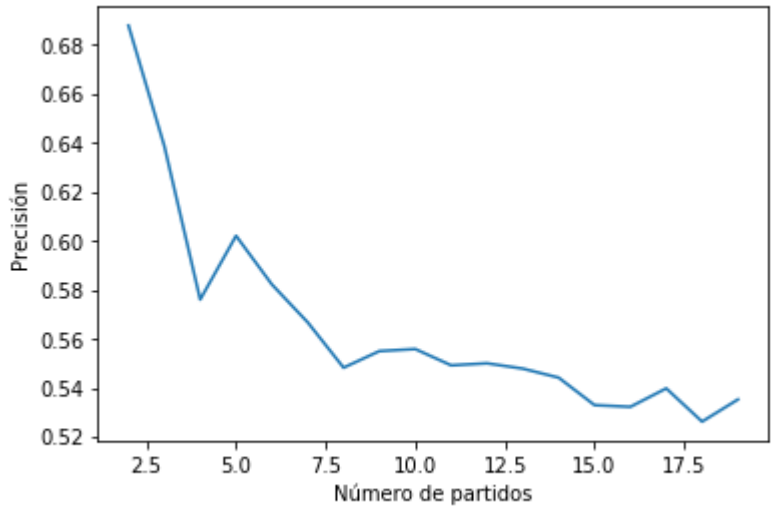
```
from matplotlib import pyplot as plt
plt.plot(n_list, error_list)
```

```
[<matplotlib.lines.Line2D at 0x1bbafbed7c0>]
```



Hemos tomado el rango de 2 hasta 19, como podemos observar, la tendencia general es que, cuanto más grande se hace la ventana, menor es la precisión, por lo tanto, la mejor ventana para tomar en este sistema es la de 2, que nos arroja una precisión del 68%, es decir 18 puntos porcentuales por encima de lo que habíamos conseguido con el anterior método.

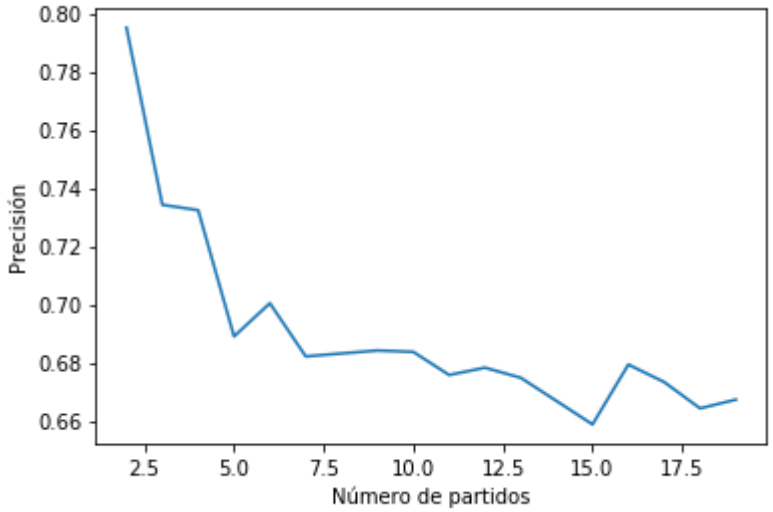
Ahora, probando el mismo método para **XGBoost:**



Obteniendo un 69% de precisión, obteniendo una precisión mejor tomando dos partidos y luego bajando el rendimiento del modelo, al igual que pasaba con Random Forests.

5.5.2. Cálculo de medias con W-D/L

Para este modelo, vamos a aplicar exactamente lo mismo, cambiando nuestro target.
En esta ocasión, vamos a probar directamente con varias ventanas diferentes, lo que nos arrojará el siguiente resultado para **Random Forests**:



Como podemos observar, al igual que en punto anterior, la tendencia es que, a más grande es la ventana, menor es la precisión que obtenemos, obteniendo la mayor precisión con una ventana de tamaño 2, y una precisión de casi el 80%, es decir, una mejora en torno al 15%.

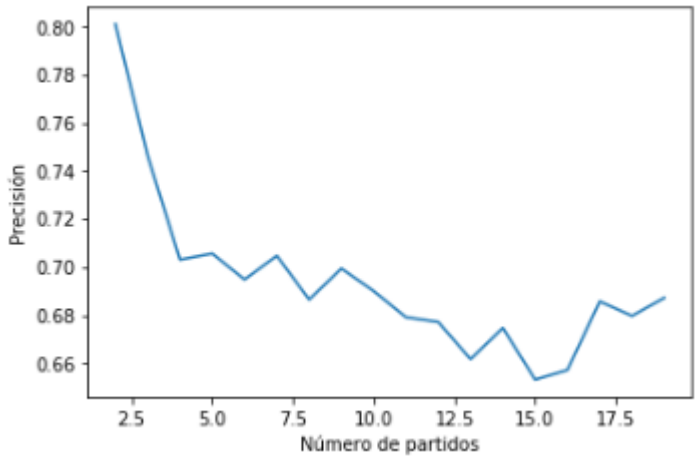
Ahora, realizaremos la misma operación para **XGBoost**:

```python
def make_predictions_grid(data, predictors):
    train = data[data["Date"] < '2022-01-01']
    test = data[data["Date"] > '2022-01-01']
    grid_search.fit(train[predictors], train["Target"])
    preds = grid_search.predict(test[predictors])
    combined = pd.DataFrame(dict(actual=test["Target"], predicted=preds), index=test.index)
    error = accuracy_score(test["Target"], preds)
    return combined, error
```

```python
error_list = []
n_list = []
```

```python
for n in list(range(2,20)):
    matches_rolling = matches.groupby("Team").apply(lambda x: rolling_averages(x, cols, new_cols, n))
    matches_rolling = matches_rolling.droplevel('Team')
    combined, error = make_predictions_grid(matches_rolling, predictors + new_cols)
    error_list.append(error)
    n_list.append(n)
```

Obteniendo los siguientes resultados:



Vemos que hay una mejora con respecto al anterior, ya que la precisión aumenta hasta el 80%, mientras que lo que habíamos obtenido si el cálculo de medias es del 69%.

En los cuatro casos hemos tomado una ventana de 2 a 20 porque, como podemos observar en las gráficas, la tendencia general es a ir perdiendo precisión conforme aumentamos el tamaño de la ventana, por lo que se ha considerado que no era necesario probar con valores más grandes.

Por otro lado, no se han tomado valores más pequeños porque 2 es lo mínimo que se puede tomar ya que, como mencionamos al principio, si tomamos los valores medios para la jornada 30 y una ventana n=2, al tener el parámetro “closed=’left’”, no tomaremos el valor de la jornada que estamos calculando, por lo tanto en vez de usar la jornada 30 y 29, solo usaremos la 29, ya que, como dijimos al principio, al estar prediciendo datos a futuro, no tiene sentido que usemos datos de partidos que aún no se han disputado.

6. Discusión

Como hemos hablado durante el desarrollo del proyecto, el problema se ha planteado desde dos perspectivas diferentes, la del sistema W-D-L y la del sistema W-D/L. Vamos a exponer los resultados de ambos y a compararlos.

6.1. Sistema W-D-L

Para este sistema, los resultados obtenidos usando las diferentes técnicas, son los siguientes en cuanto a precisión:

Precisión	Sin cálculo de medias	Con cálculo de medias
Random Forests	50,3%	68%
XGBoost	52,5%	69%

Para ambos modelos, observamos una clara mejora cuando osamos los parámetros del cambio de medias, vemos que Random Forests mejora en torno al 18%, mientras que XGBoost lo hace en torno a un 17%. Es lógico pensar que al usar más parámetros, nuestro resultado va a ser mejor, pero no tiene por qué ser así.

La conclusión que obtenemos es que, para este sistema de W-D-L, el uso de estadística del partido hace que el modelo se mejor y pase de ser un modelo aceptable (al estar en torno al 50% sería un modelo que es más preciso que tratar de adivinar los partidos al azar) a ser un modelo que nos asegura una tasa de éxito bastante alta (casi un 70% por ciento de los resultados se aciertan, es decir, más de 2 de cada 3).

Por otro lado, viendo el rendimiento de ambos, parece que XGBoost es un modelo que se adapta mejor a los datos, ya que ofrece una precisión algo mayor que Random Forests, sin embargo, esta diferencia no es muy grande y Random Forests

es más rápido de ejecutarse, ya que al hace varias tareas en paralelo y no de manera secuencial. Es por esto que podríamos considerar que ambos pueden ser usados indistintamente ya que Random Forests gana en velocidad, mientras que XGBoost gana en rendimiento.

6.2. Sistema W-D/L

Para este sistema, también vamos a recoger los resultados en la siguiente tabla:

Precisión	Sin cálculo de medias	Con cálculo de medias
Random Forests	65,2%	80,1%
XGBoost	68,7%	80,2%

En este caso, podemos observar que XGBoost tiene una mejor precisión que Random Forests cuando no usamos el cálculo de medias (en torno a un 3,5%) mientras que, sin embargo, al usar el cálculo de medias obtienen una precisión aproximada del 80% ambos. Vemos que este sistema es más preciso que el anterior, lo cual a priori es como tiene que ser, ya que hemos eliminado un posible resultado, pasando de tener un sistema de tres resultados posibles, a dos.

En este caso, al tener los mismos resultados en cuanto a precisión, afirmamos que el método Random Forests es más adecuado para la resolución de nuestro problema, ya que hemos obtenido los mismos resultados, pero en un tiempo de ejecución bastante menor.

Finalmente, aunque este sistema sea más preciso que el de W-D-L, creo que el porcentaje de mejora no justifica la simplificación del problema, por lo tanto, en líneas generales, preferiría utilizar el sistema W-D-L, ya que los resultados obtenidos se refieren directamente a las opciones que puede tener un partido de fútbol. Si miramos nuestro dataset, la cantidad de empates que hay en la muestra es

de aproximadamente un 24% de los resultados, mientras que la mejora que hemos obtenido es del 10% obviando este 24% de resultados.

7. Conclusiones

Después de haber tratado los datos, haberlos modelizado y haber discutido sobre los resultados, las conclusiones principales obtenidas son las siguientes:

1. Aunque se pueden hacer predicciones solamente usando datos de ítems que podemos saber antes del partido, los modelos obtenidos con este tipo de datos son bastante pobres (son efectivos, ya que superan a tratar de adivinar los resultados de forma azarosa, pero no por mucho). Es por ello, que hemos visto que, aunque se pueden hacer predicciones sin ellos, es necesario introducir datos sobre sucesos del partido (corners, faltas, goles a favor, en contra …) para poder obtener un modelo fiable.
2. Al usar datos de partidos, hemos observado que la mayor precisión la encontramos cuando usamos datos que tienen solamente una ventana de 2 partidos anteriores, mientras que, si vamos ampliando esta ventana, nuestra precisión va bajando. Esto se debe a que el modelo se sobreajusta, ya que al entrenarse se enfocará mucho en las tendencias de los equipos y sin embargo será incapaz de predecir si un equipo va a perder contra otro, se centrará más en asumir que si viene de una buena racha, el equipo va a seguir ganando. Esto nos deja patente que, aunque los equipos de fútbol suelen tener rachas durante una liga, no es bueno para nuestro modelos basarnos mucho en este tipo de rachas, ya que no son tan significativas en el largo plazo de lo que puede suceder en el próximo partido (pueden ser efectivas para equipos de muy arriba de la tabla o muy abajo de la tabla, ya que estos van a tener rachas de victorias o derrotas respectivamente, pero no son efectivos en cuanto a la tendencia de los equipos en general).
3. Aunque el modelo de W-D/L tiene una precisión mayor que el de W-D-L, teniendo en cuenta que, pese a haber simplificado el problema, haciéndolo pasar de dos a 3 opciones, solo hemos mejorado nuestra precisión de un 70% a un 80% (es decir, hemos acertado 1 de cada 10 partidos más en el modelo W-D/L con respecto al modelo W-D-L), no considero que merezca la pena simplificar tanto nuestro problema en pos de obtener esa mejora en la precisión. Es cierto que en circunstancias concretas en las que sea de

especial interés saber si un equipo gana o no (por ejemplo, un equipo que al final de temporada necesite ganar sus tres últimos partidos para poder salvarse) podría ser una forma de plantear el problema válida, en líneas generales es mucho mejor que nuestro target se ajuste al de las 3 posibles opciones de un partido.

4. Finalmente, hemos comparado dos algoritmos ensamblados en nuestro proyecto para nuestro problema, que son Random Forests y XGBoost, hemos visto que ambos ofrecen una precisión muy similar, pero sin embargo, hemos visto que Random Forests se ejecuta de manera más rápida, por lo que, en reglas generales, nos decantamos por este algoritmo.

Este problema se puede seguir tratando de mejorar en cuanto a precisión. Algunos de los siguientes pasos para mejorar nuestro modelo y poder hacerlo más sofisticado serían los siguientes:

- Utilizar datos de otras competiciones, por ejemplo, los equipos de la English Premier League juegan también dos copas diferentes durante el año (FA Cup y la Copa de la Liga) y los equipos que quedaron más altos en la tabla la temporada pasada juegan competiciones europeas. Esto puede servir para medir si nuestro equipo solamente está fuerte en la liga, mientras que en otras competiciones no lo está tanto, así como podría suceder que los equipos que jueguen varias competiciones tengan menor rendimiento las semanas que tienen varios partidos, ya que tienen un esfuerzo mayor. Por otro lado, al introducir más competiciones, tendríamos más partidos en la base de datos, por lo que podríamos alimentar más nuestro modelo. El problema de esto es que nuestra base de datos se haría gigantesca ya que habría que introducir equipos de otras divisiones en el caso de las copas y equipos de otras ligas en el caso de las competiciones europeas (equipos de los cuales, en ambos casos, tendríamos menos información que la que tenemos de los equipos de la Premier League).
- Utilizar una métrica que sea porcentaje de victorias contra los equipos de tabla alta y de tabla baja. Hay equipos a los que se les da mejor las grandes

citas y otros que pierden muchos puntos en partidos, a priori, más sencillos. Una métrica de este tipo podría ayudar a hacer que el algoritmo fuera capaz de detectar este tipo de partidos, que son un poco “outliers” de las tendencias generales de los equipos. Puede parecer una métrica poco útil pero hay muchos partidos de este tipo durante el año (equipos de la misma ciudad, equipos con rivalidades históricas o equipos que históricamente ganan a otro equipo aunque esa temporada pierdan casi siempre contra el resto).

- Introducir una métrica que pondere con mayor peso los datos más recientes y con menor peso los datos más antiguos. Un equipo puede ir mejorando o empeorando durante la temporada y esto quedaría reflejado en nuestro modelo, así como un equipo hace 3 años podría ser excepcional y esta temporada, debidos a cambios en la plantilla o por cualquier otro motivo, ser peor. También podríamos tener el caso contrario, un ejemplo lo tenemos en el Leicester City, un equipo que no suele estar en el EPL y que, cuando lo está, no suele ocupar las plazas más altas, sin embargo, este equipo fue capaz de salir campeón en la temporada 2015-2016.
- Utilizar una base de datos con los jugadores. Esto nos permite muchas posibilidades, desde sinergias entre jugadores, jugadores que tienen una mayor influencia sobre el juego o sobre los resultados etc. Sin embargo, esto requiere de muchísimo trabajo, ya que no puede ser una base de datos cerrada, ya que todos los años se fichan jugadores de otras ligas, por lo que de aquellos jugadores que nunca hubieran jugado en la EPL apenas tendríamos datos. Este podría ser un sistema muy bueno si el problema fuera para las mayores 5 ligas europeas, ya que la mayoría de traspasos entre clubes se producen entre equipos de estas ligas.
- Por último, también podríamos incluir una serie de métricas en referencias al oponente, es decir, por ejemplo cuando el Arsenal se enfrente al Chelsea, no solo mirar la racha de los últimos partidos del Arsenal, sino mirar también la racha de los últimos partidos del Chelsea para hacer las predicciones (es decir, sus goles a favor y en contra, faltas, etc). De esta manera, nuestras predicciones no solo se basarán como hasta ahora en el desempeño del equipo que estamos tratando de predecir, si no que también se basarán en el desempeño del equipo rival y se podrá tener en cuenta que si dos equipos lo

están haciendo bien, pero uno mejor que el otro, es más probable que el que lo está haciendo mejor se lleve la victoria o que ambos empaten.

Referencias

3.3. Metrics and scoring: quantifying the quality of predictions. (s/f). Scikit-Learn. Recuperado el 10 de diciembre de 2022, de https://scikit-learn.org/stable/modules/model_evaluation.html

Adam, D. (2022). Science and the World Cup: how big data is transforming football. *Nature*, *611*(7936), 444–446. https://doi.org/10.1038/d41586-022-03698-1

Breiman, L. (2001). Berkeley.edu. https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

Chapagain, A. (2019). *Hands-On Web Scraping with Python: Perform advanced scraping operations using various Python libraries and tools such as Selenium, Regex, and others*. Packt Publishing.

Heydt, M. (2018). *Python Web Scraping Cookbook: Over 90 proven recipes to get you scraping with Python, microservices, Docker, and AWS*. Packt Publishing.

History and time are key to power of football, says Premier League chief. (n.d.). *Times (London, England: 1788)*. Retrieved November 26, 2022, from https://www.thetimes.co.uk/article/history-and-time-are-key-to-power-of-football-says-premier-league-chief-3d3zf5kb35m

Johnston, B., & Mathur, I. (2019). *Applied Supervised Learning with Python: Use scikit-learn to build predictive models from real-world datasets and prepare yourself for the future of machine learning*. Packt Publishing.

Mitchell, R. (2018). *Web scraping with python: Collecting more data from the modern web* (2nd ed.). O'Reilly Media.

Premier League football news, fixtures, scores & results. (n.d.). Premierleague.com. Retrieved November 26, 2022, from https://www.premierleague.com/

Project jupyter. (n.d.). Jupyter.org. Retrieved November 26, 2022, from https://jupyter.org/

Sports betting market analysis, size and trends global forecast to 2022-2030. (n.d.). Thebusinessresearchcompany.com. Retrieved November 26, 2022, from https://www.thebusinessresearchcompany.com/report/sports-betting-global-market-report

Verdhan, V. (2020). *Supervised learning with python: Concepts and practical implementation using python* (1st ed.). APress.

Wade, C. (2020). *Hands-On Gradient Boosting with XGBoost and scikit-learn: Perform accessible machine learning and extreme gradient boosting with Python*. Packt Publishing.

Wikipedia contributors. (2022, November 25). *Premier League*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Premier_League&oldid=1123781091

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: Principles and techniques for data scientists*. O'Reilly Media.

Zinoviev, D. (2016). *Data Science Essentials in Python*. Pragmatic Bookshelf.