DESARROLLO EN PYTHON DE UN *WEB SCRAPER* DE DATOS DE LOS PARTIDOS DE LA LFP

Roger Cervantes Sentenà & Rodrigo Rico Gómez

TIPOLOGÍA Y CICLO DE VIDA DE LOS DATOS (PRA 1)

WEB SCRAPING

Máster Universitario en Ciencia de Datos

Universitat Oberta de Catalunya

9 de noviembre de 2020

Introducción

En este documento se presenta el trabajo llevado a cabo por los autores del mismo en la primera práctica de la asignatura *Tipología y ciclo de vida de los datos*. En ella, se pretende realizar un recolector de datos que extraiga la información interesante de una *web* para luego poder disponer de ella en formato CSV. Popularmente, este proceso es conocido con el nombre de *web scraping*.

En cada uno de los siguientes apartados se contesta a cada una de las preguntas planteadas en el enunciado de la práctica. Se comienza estableciendo el contexto en el cual se enmarca la realización del trabajo y porqué se ha elegido ese sitio *web* como fuente de los datos. Luego se realizará una descripción del *dataset* que se desea obtener como producto final, previa definición del título. Posteriormente, se mostrará el contenido, es decir, los campos o atributos que componen el *dataset*. Por último, dedicaremos un apartado a mostrar nuestro agradecimiento al propietario del sitio *web* que hemos usado para extraer la información necesaria, otro apartado a explicar en qué medida nos hemos sentido inspirados por este tema y por qué creemos que puede resultar de utilidad nuestro trabajo. Se concluirá especificando la licencia bajo la cuál publicamos nuestro *dataset*.

Al final de este documento se adjuntan dos anexos en los que se presenta el código de nuestro *web scraper* y una vista previa de los tres *datasets* generados.

1. Contexto

El trabajo desarrollado en esta práctica se enmarca dentro de la disciplina del *web scraping*. El *web scraping* es una técnica empleada para obtener información útil de un sitio *web* disponible en Internet, para ser utilizada posteriormente en un proyecto de datos [1]. En este caso, nos centramos en la etapa de extracción de la información, por tanto, la materia prima del trabajo es el sitio *web* seleccionado, y el producto final son los datos estructurados en tres *dataset* en formato *CSV*.

Concretamente, la temática del proyecto serán los datos de los partidos de la LFP (Liga de Fútbol Profesional de España). Por tanto, es necesario encontrar una fuente que nos provea de dichos datos en Internet. La primera pregunta que cualquier recolector de datos debe hacerse es si existe una página oficial que permita la descarga de los datos necesarios mediante una *API* [1]. Al no existir ninguna herramienta de tipo *API* que nos permita obtener la información, el *web scraping* se vuelve una alternativa interesante [1]. En este contexto de incapacidad de obtener la información de los partidos de la LFP a través de medios preparados para ello (*APIs*), presentamos en este trabajo el diseño de un *web scraper* que cumpla dicha función.

El sitio *web* elegido para obtener los datos es una página dedicada a recoger los datos principales de cada partido (local, visitante, resultado, fecha y hora, etc.), y que contiene enlaces a páginas que contienen información más detallada de cada partido en particular (como pueden ser las estadísticas comparativas, las alineaciones, los cambios, las tarjetas, etc.). Esta estructura se puede visualizar de forma esquemática en la Fig. 1.

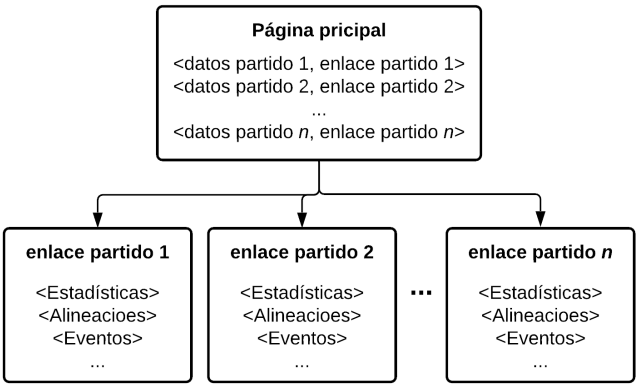

Figura 1: Estructura básica del sitio web.

Dada esta estructura, es posible acceder a toda la información que contiene el sitio *web* mediante técnicas de *web scraping*. El desarrollo del código se ha realizado en lenguaje *Python*, apoyándonos en la librería especializada BeautifulSoap para acceder a los distintos campos de la estructura *html* de las páginas *web* visitadas. Se puede acceder a la página principal a través de este enlace. Un ejemplo de página *web* de uno de los partidos se puede consultar aquí: (partido Granada - Athletic de Bilbao).

Tras inspeccionar e investigar varios sitios *web* alternativos que nos puedan aportar la misma información, se ha llegado a la conclusión de que éste es el mejor de los que se han barajado, debido a su estructura sencilla (Fig. 1) y al fácil acceso de la información interesante desde el formato *html*.

2. Definición del título

Por razones expuestas en el apartado 3, se ha decidido almacenar los datos de salida en tres *datasets* distintos. El título asignado a cada uno de ellos va en relación con los datos que alberga (Cuadro 1).

Datos	Título del dataset
Datos y estadísticas de cada partido	Partidos
Alineaciones	Alineaciones
Eventos del partido	Eventos

Cuadro 1: Título de cada *dataset*

Los nombres asignados a cada *dataset* hacen referencia a la entidad que representan. A continuación, se explica la existencia de los tres *datasets* y su descripción.

3. Descripción del *dataset*

Aunque el objetivo de la práctica es la obtención de un único *dataset*, dada la cantidad y la utilidad de la información encontrada, se ha decidido ampliar el número de *datasets* a tres. En el apartado anterior se ha comentado que los datos que extraeremos del sitio *web* serán las estadísticas de cada partido, las alineaciones y los eventos. Ante esta cantidad de información existen dos opciones:

- Almacenar toda ella en un mismo *dataset*.
- Separarla en función de sus peculiaridades.

Cada una de ellas tiene ventajas e inconvenientes. En *data warehousing*, a la primera opción se le conoce como estructura desnormalizada [2], que resulta más eficiente frente a consultas complejas pero almacena gran cantidad de información redundante, lo cual puede derivar en inconsistencias y falta de estabilidad [2]. A la segunda opción se le denomina estructura normalizada [2], y, aunque no es tan eficiente frente a consultas complejas como la anterior, su estructura resulta mucho más intuitiva y estable.
Como ejemplo, imaginemos que almacenamos toda la información en el mismo *dataset*, tendríamos en la misma tabla los eventos y las estadísticas. Si durante un partido ocurren alrededor de 20 eventos, tendríamos que almacenar todos los datos de las estadísticas 20 veces. La redundancia de información sería tal, que merece la pena dedicar un *dataset* a guardar cada partido como una fila con sus estadísticas correspondientes. Y en otro *dataset* distinto, tener un registro de cada evento ocurrido, donde el partido se encuentre referenciado por un *ID*.

Dicho esto, es preciso realizar una descripción breve de los datos almacenados en cada uno de los *datasets*. La información detallada de cada uno de los campos se encuentra en el apartado 5.

Partidos:
El objetivo de este *dataset* es que guarde en cada fila la información de un partido. Dicha información serán los datos principales que describen el encuentro: equipo local, equipo visitante, fecha y hora, estadio, árbitro, etc. y las estadísticas comparativas de cada equipo: remates, faltas, tarjetas, saques de esquina, etc.

Alineaciones:
En este *dataset* cada fila es la participación de un jugador en un partido. Los atributos caracterizan cómo ha sido el rendimiento de dicho jugador en cada encuentro. El más conocido es la calificación que ha obtenido el jugador en el partido.

Eventos:
Por último, en este *dataset* cada fila representará, como su nombre indica, un evento ocurrido en un partido determinado. Los atributos que alberga sirven para caracterizar cada evento: partido en el que ocurrió, minuto, jugador o jugadores involucrados, etc.

4. Representación gráfica

En esta sección se muestra una imagen que describe visualmente el contenido de los tres *datasets*. Se puede consultar el esquema visual en la Fig. 2.

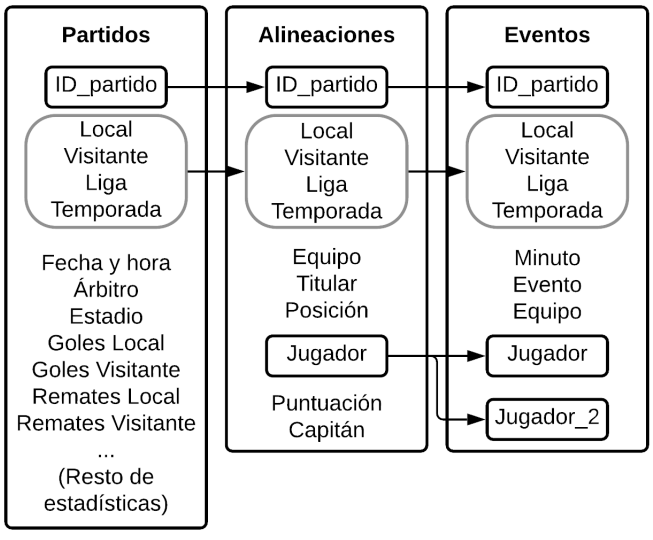

Figura 2: Esquema de los tres *datasets*.

Se ha considerado de interés remarcar aquellos campos que tienen en común varios *datasets*. Conocer esta información resulta de utilidad a la hora de hacer consultas a los datos almacenados. Especialmente cuando se trata de consultas complejas, ya que al dividir la información en tres *datasets* estamos perjudicando el rendimiento.

5. Contenido

Una vez descrita la estructura de la información almacenada en tres *datasets* es necesario realizar una descripción detallada de cada uno de los campos que integran cada *dataset*. En el caso del *dataset* 'partidos', que contiene todas las estadísticas de los partidos, no se mostrarán en su totalidad dentro del texto, se pueden consultar en el anexo II que contiene una tabla con todos los campos presentes en cada *dataset*.
Antes de proceder a describir cada *dataset*, conviene establecer una leyenda para conocer el formato en el que pueden aparecer los datos Cuadro 2.

str	Cadena de caracteres
num	Número en formato decimal
str	Número entero
datetime	Fecha y hora
bool	Booleano que alerga True o False

Cuadro 2: Leyenda con los tipos de datos.

5.1. Campos de cada *dataset*

Comenzamos mostrando, precisamente, la descripción de los principales campos (o atributos) que contiene el *dataset* 'partidos', que se pueden consultar en el Cuadro 3.

ID (*int*)	Identificador del partido
Local (*str*)	Nombre del equipo local
Visitante (*str*)	Nombre del equipo visitante
Liga (*str*)	Primera División
Fecha y hora (*datetime*)	Cadena que contiene la fecha y la hora
Temporada (*str*)	Año de la competición
Árbitro (*str*)	Nombre del árbitro
Estadio (*str*)	Nombre del estadio donde se ha jugado el partido
Sistema Local (*str*)	Sistema de juego del equipo local
Sistema Visitante (*str*)	Sistema de juego del equipo visitante
(Estadísticas)	Consultar anexo II

Cuadro 3: Campos que alberga el *dataset* 'partidos'

Por su parte, los campos que alberga el *dataset* 'alineaciones' se exponen en el Cuadro 4.

ID (*int*)	Identificador del partido
Local (*str*)	Indica el equipo local del partido
Visitante (*str*)	Indica el equipo visitante del partido
Liga (*str*)	Primera División
Temporada (*str*)	Año de la competición
Equipo (*str*)	Equipo al que pertenece el jugador
Titular (*bool*)	Si el jugador es o no titular
Posición (*str*)	Posición del jugador en el sistema de juego
Jugador (*str*)	Nombre del jugador
Puntuación (*num*)	Calificación obtenida durante el partido
Capitán (*bool*)	Indica si es o no el capitán ('C' o null)

Cuadro 4: Campos que alberga el *dataset* 'alineaciones'

Y para terminar, se muestran en el Cuadro 5 los campos que contiene el *dataset* 'eventos':

ID (*int*)	Identificador del partido en el que se produce
Local (*str*)	Indica el equipo local del partido
Visitante (*str*)	Indica el equipo visitante del partido
Liga (*str*)	Primera División
Temporada (*str*)	Año de la competición
Minuto (*int*)	Minuto en el cual se produce el evento
Evento (*str*)	Tipo de evento (tarjeta, cambio, gol, penalti, etc)
Equipo (*str*)	Si el jugador pertenece al equpio local o visitante
Jugador (*str*)	Nombre del jugador que comete el evento
Jugador_2 (*str*)	En caso de cambio, jugador que entra

Cuadro 5: Campos que alberga el *dataset* 'eventos'

5.2. Recogida de los datos

En este apartado, se describirá la estructura propia del *web scraper* diseñado para extraer los datos del sitio *web* y que constituye la aportación fundamental de este trabajo. Para evitar ensuciar el texto con código, que puede consultarse tanto en el Anexo II como en el directorio de

GitHub adjunto a este documento, se mostrará el funcionamiento del *web scraper* en forma de pseudocódigo.
El contenido del *web scraper* se encuentra en el *script* `LFP.py` adjunto, donde se define la clase `LFP_scraper`, que contiene todas las funciones empleadas para recolectar los datos de la *web*. La función troncal que articula todo el proceso y que ensambla el resto de las funciones de la clase es `scraper()`.

5.2.1. Función `scraper()`

Esta función constituye el esqueleto del proceso de *scraping*. Toma como entrada el nombre de los tres ficheros *CSV* que se deben generar y el número de partidos que se desea *srcapear*, aunque este último parámetro es opcional, por defecto se recogen 1000 partidos, siempre, lo cuál provocará que se recojan todos los partidos que se han jugado hasta la fecha de recolección. Como salida, devuelve los tres archivos *CSV* finales que contienen los datos de partidos, alineaciones y eventos. A continuación, se muestra el pseudocódigo explicativo de su funcionamiento interno:

Function scraper($file\ names, n$) :
    $html \leftarrow$ download_html($url + subdomain$)
    $bs \leftarrow$ BeautifulSoup($html$, $parser \leftarrow html$)
    $links \leftarrow$ get_match_links($html$)
    $links \leftarrow (links_{(1)}, ..., links_{(n)})$

    $c \leftarrow 0$
    for $match$ in $links$:
        $c \leftarrow c + 1$
        $link \leftarrow$ find 'href' on $match$
        get_match($link$)

    all_data_2_csv($file\ names$)

La estructura de la función es sencilla, comienza obteniendo los *links* de los partidos a partir de la estructura *html* de la página principal, usando la función `get_match_links()`. Posteriormente itera sobre los n primeros *links* y extrae de ellos la información correspondiente mediante la función `get_match()`. Finalmente guarda todos los datos en los correspondientes archivos *CSV*.
Gran parte del proceso se delega en la función `get_match()`, que extrae los datos de cada partido a través de su *link*.

5.2.2. Función `get_match()`

Se encarga de recibir la *url* de un partido concreto y extraer de ella la información, estadísticas, alineaciones y eventos. Para cada una de esas tareas se apoya en funciones auxiliares: las funciones de recolección directa de la información (`get_match_details()`, `get_match_events()` y `get_match_lineups()`). Cada una, recolecta directamente de la página *web* del partido la información que rellenará los *datasets* de 'partidos', 'eventos' y 'alineaciones' respectivamente.

Function get_match(url) :
    $Id \leftarrow$ substring from url
    $html \leftarrow$ download_html(url)
    $bs \leftarrow$ BeautifulSoup($html$, $parser \leftarrow html$)
    $info \leftarrow$ get_match_details(Id, bs, url)

    $events \leftarrow$ find 'matchEvents' on bs
    if $events$ is not empty :
        get_match_events($info$, $events$)

    $lineups \leftarrow$ find 'matchLineups' on bs
    if $lineups$ is not empty :
        get_match_lineups($info$, $lineups$)

    return $info$

5.2.3. Funciones de recolección

Como se ha comentado antes, estas funciones son las encargadas de extraer la información que rellenará los tres *datasets*, por tanto, serán tres, una para cada uno. Se trata de tres funciones con una estructura bastante engorrosa, ya que deben acceder a partes concretas de la estructura *html* de la página *web*. Por tanto, se mostrará un pseudocódigo muy simplificado.

Extracción de partidos
Comenzaremos por la función de extracción de los datos de los partidos (`get_match_details()`). Recibe como parámetros el objeto `bs`, que no es más que la estructura *html* procesada mediante la librería de *python* `BeautifulSoup`. Y almacena la información obtenida en la variable `data`.

Function get_match_details(Id, bs, url) :
    if $dataMatch$ is empty :
        get_match_cols(bs, url)

    $info \leftarrow$ find 'matchInfo' on bs
    $statistics \leftarrow$ find 'matchStatistics' on $info$
    if $statistics$ is empty :
        $statistical\ features \leftarrow -1$

    else :
        $row \leftarrow$ find features on ($info$ or $statistics$)
    append row to $dataMatch$

La función comienza 'preguntando' si el partido al que se quiere acceder es el primero cuya información se va a almacenar, en tal caso, se guarda una primera fila que será la cabecera con los nombres de las columnas. La función encargada de extraer el nombre de las columnas de la página *web* es `get_match_cols()`, su código puede consultarse en el anexo II. El siguiente condicional se asegura de que el partido se ha jugado, si, por el contrario, ha sido aplazado, únicamente se guardaran los datos principales

(local, visitante, fecha y hora, estadio, etc.) y en los datos referentes a estadísticas, se especificará que el partido ha sido aplazado. Si el partido ha sido jugado, se busca la información y se almacena en la variable `dataMatch`, que es una lista donde cada elemento es una fila con los datos de cada partido, siendo el primer elemento la fila que contiene los nombre de las columnas. Posteriormente esta lista se enviará al archivo final 'partidos.csv'.

Extracción de eventos

La función `get_match_details()` realiza la extracción de los datos de cada uno de los eventos de cada partido. Para ello toma como argumentos las variables *info* y *events*, declaradas en la función `get_match()`.

Function get_match_events(*info*, *events*) :
  if *dataEvent* is empty :
    event ← [ ]
    event ← column names
    append *event* to *dataEvent*

  matchEvents ← find 'matchEvent' on *events*
  for *event* in *matchEvents*
    find *features* on *event*
    currentEvent ← [ ]
    for *feature* in *features*
      append *feature* to *currentEvent*

    append *currentEvent* to *dataEvent*

Al igual que en la función anterior, el primer paso es comprobar si el evento que se quiere guardar es el primero. Si lo es, se almacenará en la variable `dataEvent` una fila con el nombre de las columnas del *dataset* 'eventos'. Una diferencia sustancial con respecto a la función anterior es que, en este caso, el nombre de los atributos no se encuentra disponible en la página *web* y debe introducirse manualmente (`column names`). En caso de no ser el primer dato registrado, se itera sobre el total de eventos del partido y de cada evento se extraen los atributos (variable `features`) que buscamos para luego almacenarlos como una fila en la variable `dataEvent`, que es una lista donde cada elemento es una fila del *datset* 'eventos'. Cabe destacar que en el proceso de extracción de los valores de cada atributo se han empleado funciones auxiliares diseñadas previamente cuyo código puede consultarse en el anexo II.

Cabe destacar que en esta parte de la recolección ha sido necesario corregir un *bug* existente en la página *web*, que provocaba que en los eventos en los que a priori debería participar un solo jugador (todos menos los cambios), aparecieran dos, el segundo sin sentido alguno. La solución ha conseguido eliminar el segundo jugador para evitar posibles confusiones del futuro usuario de los *datasets*.

Extracción de alineaciones

La estructura de la función `get_match_lineups()` es idéntica a la anterior:

Function get_match_lineups(*info*, *lineups*) :
  if *dataLineup* is empty :
    lineup ← [ ]
    lineup ← column names
    append *lineup* to *dataLineup*

  matchLineup ← find 'matchLineupsValues' on *events*
  get_match_lineups_players(*info*, $matchLineup_{(1)}$)
  get_match_lineups_players(*info*, $matchLineup_{(2)}$)

En este caso, los datos son almacenados en la variable `dataLineup`. Para extraer los datos, se usa la función `get_match_lineups_players()`, cuyo código puede consultarse en el anexo II.

En la ejecución del proceso de *scraping* se usan más funciones, sin embargo son pequeñas secciones de código que realizan operaciones muy específicas, y no contribuyen a visualizar el funcionamiento global del *web scraper*. Por tanto, no se muestra pseudocódigo de su estructura.

6. Agradecimientos

En nombre de los autores de este trabajo, nos gustaría expresar nuestro más sincero agradecimiento a FCSTATS.com, que es la organización propietaria de las páginas *web* de donde se han recogido los datos. Además, nos gustaría dejar constancia de que se han tenido en cuenta las especificaciones que el propietario ha dejado reflejadas en el archivo 'robots.txt' del sitio *web*.

Al no disponer de un capítulo en la memoria dedicado a presentar el código en Python (puede consultarse en el anexo I o en la página de GitHub), hemos considerado necesario hacer referencia al material bibliográfico de apoyo para desarrolla todo el *web scraper*: [3], [4] y [5].

7. Inspiración

El motor principal que nos ha llevado a recoger este tipo de datos ha sido nuestra afición por este deporte. Con este trabajo no solo publicamos datos abiertos sobre la liga española, con los que se pueden hacer consultas específicas o globales y elaborar gráficos interesantes, sino que también presentamos un método de *web scraping* a partir del cuál es posible obtener la información de otras temporadas u otras ligas almacenada en la estructura *html* de la *web*.

Con respecto a los *datasets* publicados, de cada uno de ellos es posible obtener la siguiente información:

- 'partidos.csv': se podrán realizar consultas concretas sobre resultados, estadio, árbitro, etc. Aunque la parte más interesante es aquella destinada a recoger las estadísticas de cada equipo durante el partido. Se puede estudiar la evolución temporal de las mismas, la correlación entre ellas o incluso la correlación entre ellas y el rendimiento de ciertos jugadores, si usamos también el dataset 'alineaciones.csv'.

- 'alineaciones.csv': con estos datos es posible realizar análisis del rendimiento de los jugadores y cómo in-

fluye éste en las estadísticas de cada partido o si se ve influenciado por ciertas circunstancias como el sistema de juego, la posición o partir como titular.

- 'eventos.csv': en este *dataset* se recoge información más detallada de circunstancias que se dan en cada partido. Como tarjetas, goles, cambios, etc. Lo interesante de este juego de datos es que podemos conocer la autoría de cada uno de estos eventos, y estudiar el impacto que éstos tienen sobre el rendimiento del jugador, el resultado o las estadísticas de los partidos.

En conclusión, queda de manifiesto la gran cantidad de posibilidades de análisis que puede barajar el futuro usuario de éstos datos. Aparte de facilitar estos datos al público (consultar publicación), queremos expresar nuestro deseo de que más personas contribuyan a la publicación abierta de *datasets* de ámbito deportivo.

8. Licencia

Este trabajo queda publicado bajo la licencia **CC-BY-SA 4.0** que habilita a cualquier persona a utilizar el contenido de esta obra para cualquier fin, sea o no comercial. Las únicas restricciones que se deben cumplir son que:

- Se deben citar los autores del contenido: Roger Cervantes Sentenà y Rodrigo Rico Gómez.
- Cualquier trabajo elaborado utilizando este contenido debe ser publicado bajo esta misma licencia: CC-BY-SA 4.0.

De esta forma, damos plena libertad en el uso del contenido a la vez que nos aseguramos que se reconoce nuestra autoría en cualquier trabajo derivado.

9. Publicación

El producto de este trabajo, los tres *datasets* han sido publicados en el repositorio *Zenodo* bajo la licencia especificada en el apartado anterior y con el DOI: 10.5281/zenodo.4263326. La publicación se encuentra disponible en este enlace: publicación de los datasets en Zenodo.

Referencias

[1] Subirats, L. & Calvo, M., *Web scraping*, Recursos de aprendizaje de la asignatura: *Tipología y ciclo de vida de los datos*, (2019), Editorial UOC, Barcelona, España.

[2] Abelló-Gamazo, A. & Curto-Díaz, J. & Rius-Gavidia, À. & Serra-Vizern, M. & Samos-Jiménez, J. & Vidal-Gil, J. & Díaz-Arias, D., *Introducción a las bases de datos analíticas*, Recursos de aprendizaje de la asignatura: *Diseño y uso de bases de datos analíticas*, (2020), Editorial UOC, Barcelona, España.

[3] Mitchell, R., *Web Scraping with Python, 2nd Edition*, (2018), O'Reilly Media Inc., ISBN: 9781491985571. URL oficial.

[4] Masip, D., *El lenguaje Python*, Recursos de aprendizaje de la asignatura: *Tipología y ciclo de vida de los datos*, (2010), Editorial UOC, Barcelona, España.

[5] Lawson, R., *Web Scraping with Python, 2nd Edition*, (2015), O'Reilly Media Inc., ISBN: 9781782164364. URL oficial.

Contribuciones	**Firma**
Investigación previa	RCS, RRG
Redacción de las respuestas	RCS, RRG
Desarrollo código	RCS, RRG

Anexo I(a): Código LFP.py

```
import urllib.request
import time
from bs4 import BeautifulSoup

class LFP_Scraper():

        # Inicializamos la classe
        def __init__(self):
                self.url = "https://es.fcstats.com"
                self.subdomain = "/partidos,primera-division-espana,19,1.php"

                self.dataMatches = []
                self.dataEvents = []
                self.dataLineups = []

        # Descargamos la pagina seleccionada
        def download_html(self, url):
                response = urllib.request.urlopen(url)
                html = response.read()
                return html

        # Obtenemos la lista de partidos
        def get_match_links(self, html):

                # Get link for all match
                bs = BeautifulSoup(html, 'html.parser')
                match_Links = bs.findAll("td", {"class": "matchResult"})

                return match_Links

        def get_match_cols(self, bs, url_Match):

                cols_general = ['ID', 'Local', 'Visitante', 'Liga', 'Temporada',
                                'Fecha-hora', 'Arbitro', 'Estadio',
                                'Sistema_Local', 'Sistema_Visitante']
                cols_goles = ['Goles_Local', 'Goles_Visitante',
                              'Goles_1er_tiempo_Local',
                              'Goles_1er_tiempo_Visitante']

                # Podemos obtener los nombres de las columnas a partir
                  del primer partido:
                matchStatistics = bs.find("div", {"id": "matchStatistics"})
                stats = matchStatistics.findAll("div")

                cols_stats = []
                for i in range(1, 64, 4):
                        cols_stats.append(stats[i].findAll("div")[1].text.strip()
                                                .replace(' ','_')
                                                .replace('[%]','(por)'))

                # Nos interesa que cada estadistica se guarde por separado LOCAL
                  y VISITANTE:
                cols_lv = []
                for col in cols_stats:
                        cols_lv.append(col + '_Local')
                        cols_lv.append(col + '_Visitante')

                cols = cols_general + cols_goles + cols_lv
```

```
        # Store the data
        self.dataMatches.append(cols)

def get_match_details(self, match_Id, bs, url_Match):
        #print("get_match_details")

        # Pintamos la cabecera la primera vez
        if len(self.dataMatches) == 0: self.get_match_cols(bs, url_Match)

        matchInfo = bs.find("div", {"id": "matchInfo"})
        matchStatistics = matchInfo.find("div", {"id": "matchStatistics"})

        if matchStatistics == None:
                principal_info = bs.find("div", {"id": "matchInfo"})
                a = principal_info.findAll("a")
                match_row = [match_Id, a[2].text.strip(), a[3].text.strip(),
                             "".join(a[0].text.strip().split()),
                             principal_info.findAll("div")[1].text.strip()[19:28],
                             a[1].text.strip().replace(',', ' -')]
                aplazado = ['Sin disputar'] * 4 + ['-1'] * 36
                self.dataMatches.append(match_row + aplazado)
                return match_row + aplazado
        else:
                matchSystems = bs.find("div", {"id": "matchLineups"})
                matchSystems = matchSystems.findAll("div",
                                        {"class": "matchLineupsValues"})[0]
                matchSystems = matchSystems.findAll("div")

                stats = matchStatistics.findAll("div")

                principal_info = bs.find("div", {"id": "matchInfo"})
                matchResult = bs.find("td", {"id": "matchResult"})
                [golesL, golesV] = matchResult.text.strip().split(':')
                a = principal_info.findAll("a")
                d = principal_info.findAll("div")
                [goles_1erT_L, goles_1erT_V] = d[6].text.strip()[-3:].split(':')

                match_row = [match_Id, a[2].text.strip(), a[3].text.strip(),
                             "".join(a[0].text.strip().split()),
                             d[1].text.strip()[19:28],
                             a[1].text.strip().replace(',', ' -'),
                             d[3].text.strip()[9:].replace(',', ' -'),
                             d[4].text.strip()[9:], matchSystems[0].text,
                             matchSystems[1].text,
                             golesL, golesV, goles_1erT_L, goles_1erT_V]

                for i in range(1, 64, 4):
                        match_row.append(stats[i].findAll("div")[0].text.strip())
                        match_row.append(stats[i].findAll("div")[2].text.strip())

                self.dataMatches.append(match_row)

                return match_row[0:6]

def get_event_home_away(self, event_minute_home):
        event_minute_home = event_minute_home.text.strip()
        if event_minute_home != "":
                event_home_away = 'Local'
        else:
                event_home_away = 'Visitante'
```

```
        return event_home_away

def clear_event_type(self, event_Type):

        # Get de class
        event_Type = event_Type['class']
        # Get second component
        event_Type = event_Type[1]
        # Remove 'eventIcon_' String
        event_Type = event_Type[10:]

        list_Event_Type = {
                '1': 'Gol',
                '2': 'Autogol',
                '3': 'Gol Penalti',
                '4': 'Penalti Fallado',
                '5': 'Tarjeta Amarillla',
                '6': 'Tarjeta Roja',
                '7': 'Cambio'
        }

        # Find value in list. Default return value
        event_Type = list_Event_Type.get(event_Type, event_Type)

        return event_Type

def clear_event_minute(self, event_home_away, event_minute_home,
                       event_minute_away):
        if event_home_away == "Local":
                event_minute = event_minute_home.text.strip()
        else:
                event_minute = event_minute_away.text.strip()

        return event_minute.replace('\'', '')

def clear_event_player(self, event_home_away, event_player_home,
                       event_player_away):

        if event_home_away == "Local":
                event_player = event_player_home.text.strip()
        else:
                event_player = event_player_away.text.strip()

        start = event_player.find('(')
        stop = event_player.find(')')
        if len(event_player) > stop:
                event_player = event_player[0: start:] +
                               event_player[stop + 1::]

        return event_player.strip()

def clear_event_2_player(self, event_home_away, event_player_home,
                         event_player_away, event_type):
        if event_home_away == "Local":
                event_2_player = event_player_home.text.strip()
        else:
                event_2_player = event_player_away.text.strip()

        # If is a event type in list, remove a web bug (second player)
        if event_type in ('Cambio'):
```

```
                start = event_2_player.find('(')
                stop = event_2_player.find(')')
                if len(event_2_player) > stop:
                        event_2_player = event_2_player[start + 1:stop]
        else:
                event_2_player = ''

        return event_2_player.strip()

def get_match_events(self, match_info, match_events):

        # print('***** matchEvents *****')
        # Pintamos la cabecer
        if len(self.dataEvents) == 0:
                current_event = []
                current_event.append('ID')
                current_event.append('Local')
                current_event.append('Visitante')
                current_event.append('Liga')
                current_event.append('Temporada')
                current_event.append('Minuto')
                current_event.append('Evento')
                current_event.append('Equipo')
                current_event.append('Jugador')
                current_event.append('Jugador_2')
                # Store the data
                self.dataEvents.append(current_event)

        match_event = match_events.findAll("div", {"class": "matchEvent"})
        for event in match_event:

                # Get all Divs
                divs = event.findAll('div')

                event_home_away = self.get_event_home_away(divs[1])
                event_team = ''
                if event_home_away == 'Local': event_team = match_info[1]
                if event_home_away == 'Visitante': event_team = match_info[2]

                event_minute = self.clear_event_minute(event_home_away,
                                                       divs[1], divs[3])
                event_type = self.clear_event_type(divs[2])
                event_player = self.clear_event_player(event_home_away,
                                                       divs[0], divs[4])
                event_2_player = self.clear_event_2_player(event_home_away,
                                                           divs[0], divs[4],
                                                           event_type)

                # Create Current Event
                current_event = []
                current_event.append(match_info[0])
                current_event.append(match_info[1])
                current_event.append(match_info[2])
                current_event.append(match_info[3])
                current_event.append(match_info[4])
                current_event.append(event_minute)
                current_event.append(event_type)
                current_event.append(event_team)
                current_event.append(event_player)
                current_event.append(event_2_player)
```

```
                # Store the data
                self.dataEvents.append(current_event)

        return True

def get_match_lineups_system(self, match_info, matchLineups):

        # print('***** get_match_lineups_system *****')

        # Split 2 divs
        divs = matchLineups.findAll("div")

        # Home System
        current_lineups = []
        current_lineups.append(match_info[0])
        current_lineups.append(match_info[1])
        current_lineups.append('Sistema')
        current_lineups.append(divs[0].text.strip())
        # Store the data
        self.dataLineups.append(current_lineups)

        # Away System
        current_lineups = []
        current_lineups.append(match_info[0])
        current_lineups.append(match_info[2])
        current_lineups.append('Sistema')
        current_lineups.append(divs[1].text.strip())
        # Store the data
        self.dataLineups.append(current_lineups)

        return True

def get_match_lineups_players(self, match_info, matchLineups, titular):

        list_Position = {
                'G': 'Portero',
                'D': 'Defensa',
                'M': 'Mediocentro',
                'F': 'Delantero'
        }

        # print('***** get_match_lineups_players *****')
        divs_team = matchLineups.findAll("div", recursive=False)

        for team in [0, 1]:
                if team == 0: home_away = match_info[1]
                if team == 1: home_away = match_info[2]

                divs_player = divs_team[team].findAll("div")
                for div in divs_player:

                        position = div.find("span", {"class": "lineupPosition"})
                        if position is None: position = ""
                        else:
                                position = position.text.strip()
                                position = list_Position.get(position, position)


                        rating = div.find("span", {"class": "lineupRating"})
                        if rating is None: rating = ""
                        else: rating = rating.text.strip()
```

```
                        captain = div.find("span", {"class": "lineupCaptain"})
                        if captain is None: captain = ""
                        else: captain = captain.text.strip()

                        player = div.contents[2]
                        if player is None: player = ""
                        else: player = player.strip()

                        # Curren Player
                        current_lineups = []
                        current_lineups.append(match_info[0])
                        current_lineups.append(match_info[1])
                        current_lineups.append(match_info[2])
                        current_lineups.append(match_info[3])
                        current_lineups.append(match_info[4])
                        current_lineups.append(home_away)
                        current_lineups.append(titular)
                        current_lineups.append(position)
                        current_lineups.append(player)
                        current_lineups.append(rating)
                        current_lineups.append(captain)
                        # Store the data
                        self.dataLineups.append(current_lineups)

        return True

def get_match_lineups(self, match_info, matchLineups):

        #print('***** matchLineups *****')

        # Anadimos las cabeceras
        if len(self.dataLineups) == 0:
                current_lineups = []
                current_lineups.append('ID')
                current_lineups.append('Local')
                current_lineups.append('Visitante')
                current_lineups.append('Liga')
                current_lineups.append('Temporada')
                current_lineups.append('Equipo')
                current_lineups.append('Titular')
                current_lineups.append('Posicion')
                current_lineups.append('Jugador')
                current_lineups.append('Puntuacion')
                current_lineups.append('Capitan')

                # Store the data
                self.dataLineups.append(current_lineups)

        match_lineup = matchLineups.findAll("div",
                                        {"class": "matchLineupsValues"})
        self.get_match_lineups_players(match_info, match_lineup[1], 'Si')
        self.get_match_lineups_players(match_info, match_lineup[2], 'No')

        return True

def get_match(self, url_Match):
        #print('***** get_match *****')
        match_Id = url_Match.split(sep=',')[3]
        match_Id = match_Id[0:-4]
```

```
        # Download HTML
        html = self.download_html(self.url + '/' + url_Match)
        bs = BeautifulSoup(html, 'html.parser')

        # Get the data of the data match in a row:
        # Obtenemos el id y los dos equipos
        match_info = self.get_match_details(match_Id, bs, url_Match)

        matchEvents = bs.find("div", {"id": "matchEvents"})
        if matchEvents is not None:
                self.get_match_events(match_info, matchEvents)

        matchLineups = bs.find("div", {"id": "matchLineups"})
        if matchLineups is not None:
                self.get_match_lineups(match_info, matchLineups)

        return match_info

# Pasamos una estructura a CSV
def data2csv(self, data, filename):
        # Overwrite to the specified file.
        # Create it if it does not exist.
        file = open("../csv/" + filename, "wb+")

        # Dump all the data with CSV format
        for i in range(len(data)):
                new_line = ""
                for j in range(len(data[i])):
                        new_line += data[i][j] + ","
                new_line += "\n"
                file.write(new_line.encode('utf8'))
        file.close()

# Pasamos todas las estructuras a csv
def all_data_2_csv(self, fileMatchsName, fileEventsName, fileLineupsName):
        self.data2csv(self.dataMatches, fileMatchsName)
        self.data2csv(self.dataEvents, fileEventsName)
        self.data2csv(self.dataLineups, fileLineupsName)

# Funcion principal, le pasamos los 3 nombres de los ficheros y opcional si
  queremos coger solo X partidos.
def scrape(self, output_fileMatches, output_fileEvents, output_fileLineups,
           num_de_partidos=1000):
        print("Web Scraping de la LFP desde '" + self.url +
              self.subdomain + "'")
        print("Este proceso puede tardar alrededor de 5 minutos.\n")

        # Start timer
        start_time = time.time()

        # Download HTML
        html = self.download_html(self.url + self.subdomain)
        bs = BeautifulSoup(html, 'html.parser')

        # Get the links of each match
        match_links = self.get_match_links(html)

        # Cogemos los X primeros para pruebas, o todos si no pasamos parametro
        match_links = match_links[:num_de_partidos]

        # Bucle para todos los partidos seleccionados
```

```
        contador = 0
        num_partidos = len(match_links)
        for match in match_links:
                contador += 1
                print('Procesando partido numero ' + str(contador) + ' de ' +
                      str(num_partidos))

                match_link = match.find('a').get('href')
                self.get_match(match_link)

        # Pasamos los datos a CSV
        self.all_data_2_csv(output_fileMatches, output_fileEvents,
                            output_fileLineups)

        # Show elapsed time
        end_time = time.time()
        print("\nelapsed time: " + str(round(((end_time - start_time) / 60), 2)) +
              " minutes")
```

Anexo I(b): Código main.py

```
from LFP import LFP_Scraper

output_fileMatches = "Partidos.csv"
output_fileEvents = "Eventos.csv"
output_fileLineups = "Alineaciones.csv"

scraper = LFP_Scraper()
scraper.scrape(output_fileMatches, output_fileEvents, output_fileLineups)
```

Anexo II(a): partidos.csv

ID	Local	Visitante	Liga	Temporada	Fecha-hora	Árbitro	Estadio	Sistema_Local	Sistema_Visitante
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	12 Septiembre 2020 - 14:00	Valentin Pizarro Gomez - Spain	Estadio Municipal de IpurÃºa	4-4-2	4-3-3
473359	Granada	Athletic Bilbao	PrimeraDivision(EspaÃ±a)	2020/2021	12 Septiembre 2020 - 16:30	Antonio Mateu - Spain	Estadio Nuevo Los CÃ¡rmenes	4-2-3-1	4-2-3-1
473361	CÃ¡diz	Osasuna	PrimeraDivision(EspaÃ±a)	2020/2021	12 Septiembre 2020 - 19:00	Isidro Diaz de Mera Escuderos - Spain	Estadio RamÃ³n de Carranza	4-2-3-1	4-4-2
473367	Deportivo AlavÃ©s	Real Betis	PrimeraDivision(EspaÃ±a)	2020/2021	13 Septiembre 2020 - 12:00	Pablo Gonzales Fuertes - Spain	Estadio de Mendizorroza	3-1-4-2	4-2-3-1
473362	Real Valladolid	Real Sociedad	PrimeraDivision(EspaÃ±a)	2020/2021	13 Septiembre 2020 - 14:00	Mario Melero Lopez - Spain	Estadio Municipal JosÃ© Zorrilla	4-2-3-1	4-1-4-1
473363	FC Barcelona	Elche	PrimeraDivision(EspaÃ±a)	2020/2021	13 Septiembre 2020 - 15:00	Sin disputar	Sin disputar	Sin disputar	Sin disputar
473364	Real Madrid	Getafe	PrimeraDivision(EspaÃ±a)	2020/2021	13 Septiembre 2020 - 15:00	Sin disputar	Sin disputar	Sin disputar	Sin disputar
473365	Villarreal	Huesca	PrimeraDivision(EspaÃ±a)	2020/2021	13 Septiembre 2020 - 16:30	Javier Estrada Fernandez - Spain	Estadio de la CerÃ¡mica	4-4-1-1	4-2-3-1

Goles_Local	Goles_Visitante	Goles_1er_tiempo_Local	Goles_1er_tiempo_Visitante	Remates_a_puerta_Local	Remates_a_puerta_Visitante	Remates_fuera_Local	Remates_fuera_Visitante	Remates_Local
0	0	0	0	1	1	5	3	7
2	0	0	0	2	2	2	7	4
0	2	0	1	3	4	6	2	11
0	1	0	0	2	4	3	5	7
1	1	1	0	3	2	5	6	8
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	0	1	3	2	5	3	10

Remates_Visitante	Remates_rechazados_Local	Remates_rechazados_Visitante	Tiros_dentro_del_Ãrea_Local	Tiros_dentro_del_Ãrea_Visitante	Tiros_desde_fuera_del_Ãrea_Local	Tiros_desde_fuera_del_Ãrea_Visitante
4	1	0	3	4	4	0
9	0	0	2	7	2	2
6	2	0	5	4	6	2
11	2	2	6	4	1	7
8	0	0	8	4	0	4
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
7	2	2	7	4	3	3

Faltas_Local	Faltas_Visitante	CÃ³rneres_Local	CÃ³rneres_Visitante	Fueras_de_juego_Local	Fueras_de_juego_Visitante	PosesiÃ³n_de_balÃ³n_(por)_Local	PosesiÃ³n_de_balÃ³n_(por)_Visitante	Tarjetas_amarillas_Local
11	8	2	3	1	4	49	51	2
8	10	1	2	5	0	40	60	3
12	18	6	2	2	0	52	48	1
18	17	6	4	8	3	41	59	3
13	16	5	2	1	1	38	62	4
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
9	15	4	4	4	3	63	37	1

Tarjetas_amarillas_Local	Tarjetas_amarillas_Visitante	Tarjetas_rojas_Local	Tarjetas_rojas_Visitante	Paradas_Local	Paradas_Visitante	Pases_totales_Local	Pases_totales_Visitante	Pases_acierte_Local
2	1	0	0	1	1	237	249	136
3	4	0	0	2	0	320	442	224
1	2	0	0	2	3	322	302	218
3	3	0	0	3	2	289	422	222
4	1	0	0	1	1	227	364	153
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
1	0	0	0	1	2	434	244	374

Pases_totales_Visitante	Pases_acierte_Local	Pases_acierte_Visitante	Pases_acierte_(por)_Local	Pases_acierte_(por)_Visitante
249	136	143	57	57
442	224	341	70	77
302	218	182	68	60
422	222	330	77	78
364	153	282	67	77
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
244	374	185	86	76

Anexo II(b): alienaciones.csv

ID	Local	Visitante	Liga	Temporada	Equipo	Titular	Posicion	Jugador	Puntuacion	Capitan
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Portero	1. Marko DmitroviÄ‡	69	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Defensa	3. Pedro Bigas	69	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Defensa	4. Paulo Oliveira	69	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Defensa	6. Sergio Ãlvarez	69	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Defensa	20. RÃ³ber	69	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Mediocentro	8. Papakouli Diop	67	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Mediocentro	10. Edu ExpÃ³sito	63	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Mediocentro	14. Takashi Inui	62	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Mediocentro	21. Pedro LeÃ³n	69	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Delantero	9. Sergi Enrich	66	C
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	Eibar	Si	Delantero	17. Kike	69	

Anexo II(c): eventos.csv

ID	Local	Visitante	Liga	Temporada	Minuto	Evento	Equipo	Jugador	Jugador_2
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	13	Tarjeta Amarillla	Eibar	Pedro Bigas	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	37	Tarjeta Amarillla	Celta de Vigo	Nolito	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	41	Tarjeta Amarillla	Eibar	Sergio Ãlvarez	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	61	Cambio	Eibar	Damian KÄ…dzior	Takashi Inui
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	68	Cambio	Eibar	Recio	Sergi Enrich
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	73	Tarjeta Amarillla	Celta de Vigo	Lucas Olaza	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	75	Tarjeta Amarillla	Celta de Vigo	David JuncÃ	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	77	Cambio	Celta de Vigo	Miguel Baeza	Brais MÃ©ndez
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	78	Tarjeta Amarillla	Celta de Vigo	Okay YokuÅŸlu	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	81	Tarjeta Amarillla	Eibar	Papakouli Diop	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	84	Cambio	Eibar	Rafa Soares	Pedro LeÃ³n
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	85	Cambio	Celta de Vigo	Santi Mina	Emre Mor
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	87	Tarjeta Amarillla	Eibar	Papakouli Diop	
473360	Eibar	Celta de Vigo	PrimeraDivision(EspaÃ±a)	2020/2021	87	Tarjeta Roja	Eibar	Papakouli Diop	
473359	Granada	Athletic Bilbao	PrimeraDivision(EspaÃ±a)	2020/2021	10	Tarjeta Amarillla	Granada	Yangel Herrera	
473359	Granada	Athletic Bilbao	PrimeraDivision(EspaÃ±a)	2020/2021	15	Tarjeta Amarillla	Athletic Bilbao	Oier Zarraga	
473359	Granada	Athletic Bilbao	PrimeraDivision(EspaÃ±a)	2020/2021	45	Tarjeta Amarillla	Granada	GermÃ¡n SÃ¡nchez	
473359	Granada	Athletic Bilbao	PrimeraDivision(EspaÃ±a)	2020/2021	49	Gol	Granada	Yangel Herrera	
473359	Granada	Athletic Bilbao	PrimeraDivision(EspaÃ±a)	2020/2021	52	Tarjeta Amarillla	Athletic Bilbao	IÃ±igo MartÃnez	
473359	Granada	Athletic Bilbao	PrimeraDivision(EspaÃ±a)	2020/2021	53	Gol	Granada	Luis Milla	