Práctica 1
¿Cómo podemos capturar los datos de la web?

Tipología y ciclo de vida de los datos

Patricia Luengo Carretero
Alejandro Zarza Roa

Índice

1. Contexto

Para esta práctica se ha recolectado la información de la web de la liga femenina de baloncesto
https://baloncestoenvivo.feb.es/rankings/lfendesa/4/2022

La Liga Femenina Endesa es la máxima competición de baloncesto femenino en España, organizada por la Federación Española de Baloncesto (FEB) y patrocinada por Endesa.

La página web proporciona información sobre la competición, como la clasificación de los equipos, los resultados de los partidos, las estadísticas de los jugadores y los equipos, las noticias y novedades relacionadas con la competición, y otra información relevante para los aficionados al baloncesto femenino en España.

El sitio web elegido proporciona esta información porque es el sitio oficial de la Liga Femenina Endesa, donde la FEB y Endesa publican y actualizan información relacionada con la competición. Además, el sitio web es una fuente importante de información para los aficionados al baloncesto femenino en España, ya que permite seguir la evolución de los equipos y las jugadoras, y estar al tanto de las últimas noticias y novedades de la competición.

En definitiva, el sitio web es una herramienta clave para la promoción y difusión de la Liga Femenina Endesa y para fomentar el interés y la participación en el baloncesto femenino en España.

2. Título

"Estadísticas de jugadoras de la Liga Femenina de Baloncesto: Temporadas 2003-2023"

3. Descripción del dataset

El dataset contiene información estadística de cada jugadora de la Liga Femenina Endesa por temporada. Se compone de 20 atributos entre los que se encuentra información sobre su nacionalidad, equipo, puntos, rebotes,

asistencias, balones recuperados y perdidos, tapones a favor y en contra, faltas recibidas y cometidas, valoración, minutos jugados y porcentajes de tiros de 2, tiros de 3 y tiros libres. Además, se incluye el número de partidos jugados por cada jugadora en cada temporada.

Este conjunto de datos podría ser utilizado para analizar tendencias y estadísticas de jugadoras de la Liga Femenina Endesa de Baloncesto, así como para entrenar modelos de aprendizaje automático que permitan predecir resultados de partidos o tendencias de la liga. También podría ser utilizado por periodistas y fans de baloncesto para obtener información actualizada en tiempo real sobre los resultados y estadísticas de la liga.

4. Representación gráfica

5. Contenido

Este conjunto de datos contiene información estadística de jugadoras de la Liga Femenina Endesa de Baloncesto para las temporadas comprendidas entre la 2003/2004 y la 2022-2023.

El dataset contiene 20 atributos por cada jugadora y temporada, detallados a continuación:

- Temporada: Año de competición
- Jugadora: Nombre y apellidos de la jugadora
- Nacionalidad: Lugar de nacimiento
- Equipo: Equipo al que pertenece
- Puntos: Puntos anotados
- Rebotes Totales: Rebotes capturados en total
- Rebotes Ofensivos: Rebotes capturados en ataque
- Rebotes Defensivos: Rebotes capturados en defensa
- Asistencias: Asistencias de canasta realizadas
- Balones recuperados: Balones que se han recuperado
- Balones perdidos: Pérdidas de balón cometidas
- Tapones a favor: Tapones realizados
- Tapones en contra: Tapones recibidos
- Faltas cometidas: Faltas realizadas
- Valoración: Puntos de valoración obtenidos
- Minutos jugados: Minutos que ha disputado
- % Tiros de 2: Porcentaje de acierto en tiros de dos puntos
- % Tiros de 3: Porcentaje de acierto en tiros de tres puntos
- % Tiros Libres: Porcentaje de acierto en tiros libres
- Partidos: Partidos disputados

6. Propietario

Tal y como se recoge en la página web de la FEB el propietario de los datos corresponde a la Federación Española de Baloncesto.

No hemos encontrado estudios realizados por la FEB con respecto a sus datos, pero si citamos un artículo, “El boom de la inteligencia artificial:

detectando tendencias atípicas en la ACB" realizado por Adrià Arbués Sangüesa (2017):

"Con un conjunto de datos muy limitado (32 partidos de 17 equipos distintos), se pueden extraer conclusiones más que interesantes mediante el uso de inteligencia artificial. El "boom" de estos modelos está aún por llegar, pero los entrenadores se pueden frotar las manos con la cantidad de información que tendrán a su disposición."

En este artículo queda claro el gran potencial que tiene el conjunto de datos que estamos extrayendo.

Los pasos que hemos seguido para actuar de acuerdo con los principios éticos y legales han sido:

1. Comprobar los términos y condiciones de la página: Antes de realizar cualquier web scraping, se han leído las condiciones de la página. Entre los que se encuentran provocar daños mediante difamación, manipular los datos obtenidos, hacer propaganda de carácter racista, xenófobo, obsceno, pornográfico, apología del terrorismo o vulnerante de los derechos humanos.

2. Limitar la cantidad de datos extraídos: Para evitar cualquier mala interpretación de los datos o vulneración de privacidad la cantidad de datos extraídos se han limitado los que son necesarios.

3. Utilizar herramientas de web scraping éticas: Las consultas se han realizado asegurándonos de no sobrecargar la página web con demasiadas solicitudes.

4. No violar la privacidad de los usuarios: Tal y como se ha mencionado en puntos anteriores, y quedaba reflejado en los términos y condiciones de la página, los datos extraídos han sido única y exclusivamente los necesarios, asegurándonos de que no se violara la privacidad de los usuarios

5. Respetar los derechos de autor: En este punto, nos hemos asegurado de que la información extraída de la página no se haya utilizado de manera no autorizada.

6. Cumplir con las leyes y regulaciones aplicables: Todas las fuentes utilizadas en esta práctica han sido mencionadas y correctamente justificadas, respetando así las leyes de propiedad intelectual, privacidad y protección de datos.

7. Inspiración

El conjunto de datos que contiene estadísticas de las jugadoras de la Liga Femenina Endesa de baloncesto es interesante por varias razones. A continuación los desglosamos en varios puntos:

1. Recopilación de datos y unificación de los datos para su posterior manejo: Estos datos podrían ser útiles para analizar tendencias y estadísticas de equipos y jugadoras de la liga.

2. Análisis de rendimiento de los equipos: Al recopilar datos de partidos y resultados, se podría hacer un análisis del rendimiento de los equipos de la liga, lo que podría ser útil para entrenadores, analistas deportivos y fans de baloncesto.

3. Creación de modelos predictivos: Los datos obtenidos mediante web scraping podrían ser utilizados para entrenar modelos de aprendizaje automático que permitan predecir resultados de partidos o tendencias de la liga.

4. Obtención de información actualizada: Al automatizar la obtención de datos, se podría obtener información actualizada en tiempo real sobre los resultados y estadísticas de la liga, lo que podría ser útil para periodistas y fans.

Algunas de las preguntas que se pretenden responder son las siguientes:

- ¿Cuáles son las estadísticas de las jugadoras destacadas en la Liga Femenina Endesa?

- ¿Cómo se comparan las estadísticas de diferentes jugadoras en términos de eficiencia y contribución al equipo?

- ¿Cómo influyen las lesiones, la posición en la cancha y el tiempo de juego en el rendimiento de una jugadora en particular?

- ¿Hacia qué posición en la cancha se tiende a jugar más? ¿Qué posición es más importante a la hora de utilizar los recursos de un equipo?

8. Licencia

9. Código

El código ha sido implementado en Google Colab que nos ha permitido programar y ejecutar el código de forma colaborativa.

Para utilizar este entorno se han utilizado distintos paquetes que nos permiten utilizar el navegador Google Chrome y el Webdriver. Además de utilizar la librería Selenium, se han utilizado Pandas, Time y Csv. La descarga del archivo final se lleva a cabo sobre google drive.

def iniciar_chrome()

En esta primera función se implementan todas las opciones del navegador entre las que se ha incluido una opción que evita que Selenium sea detectado, se introduce el user agent, se crea el servicio y retorna el driver.

```
def iniciar_chrome():
  """
  Inicia el navegador con los parámetros indicados y nos devuelve el driver
  """
  # Opciones de Chrome
  chrome_options = Options()
  user_agent = "Mozilla/5.0 (Windows NT 10.; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36"
  chrome_options.add_argument(f"user-agent={user_agent}") # definimos el user agent
  chrome_options.add_argument('--headless')
  chrome_options.add_argument('--no-sandbox') # deshabilita el modo sandbox
  chrome_options.add_argument('--start_maximized') # maximizamos la ventana
  chrome_options.add_argument('--disable-web-security') # deshabilita la politica cross origin
  chrome_options.add_argument('--disable-extensions') # para que no cargue las extensiones
  chrome_options.add_argument('--disable-notifications') # bloquea las notificaciones de chrome
  chrome_options.add_argument('--allow-running-insecure-content') # desactiva el contenido no seguro
  chrome_options.add_argument('--no-default-browser-check') # evitar el aviso de que chrome no es el navegador principal
  chrome_options.add_argument('--no-first-run') # evita que se ejecuten tareas que se realizan por primera vez en el navegador
  chrome_options.add_argument('--no-proxy-server') # usar conexiones directas
  chrome_options.add_argument('--disable-blink-features=AutomationControlled') # evita que selenium sea detectado
  # Ruta del controlador de Chrome
  chrome_driver_path = '/content/chromedriver'

  # Crear un objeto Service para especificar la ruta del controlador
  service = Service(chrome_driver_path)

  # Crear una instancia del navegador Chrome
  driver = webdriver.Chrome(service=service, options=chrome_options)
  return driver
```

def select_context_navigation()

Para poder extraer los datos tenemos una lista desplegable en la que podemos realizar una selección. Para realizarla hemos implementado los id únicos siguientes:

- **temporada:** _ctl0_MainContentPlaceHolderMaster_temporadasDropDownList
- **competición:** _ctl0_MainContentPlaceHolderMaster_gruposDropDownList
- **atributos:** _ctl0_MainContentPlaceHolderMaster_rankingsDropDownList
- nacionalidad: _ctl0_MainContentPlaceHolderMaster_nacionalDropDownList

El resto de selectores se dejan tal y como aparecen en un principio ya que nos interesa recopilar los datos acumulados de la temporada.

Se han tenido en cuenta lapsos de tiempo entre cada opción del desplegable, ya que cada vez que elegimos uno de los atributos de la lista, la página web se refresca.

Esta función además de realizar la navegación autónoma devuelve como resultado la paginación una vez que se ha realizado la elección completa de los filtros.

```
def select_context_navigation(temporada, competicion, atributo, nacionalidad):
  """
  Selecciona las opciones en un desplegable y además devuelve el número de
  páginas por las que hay que navegar una vez se ha hecho la selección
  """
  # selección de la temporada
  dropdownT = Select(driver.find_element('id',
'_ctl0_MainContentPlaceHolderMaster_temporadasDropDownList'))
  dropdownT.select_by_visible_text(temporada)
  time.sleep(1) # tiempo de carga
```

```
  # selección del tipo de competición
  dropdownC = Select(driver.find_element('id',
'_ctl0_MainContentPlaceHolderMaster_gruposDropDownList'))
  dropdownC.select_by_visible_text(competicion)
  time.sleep(1) # tiempo de carga
  # selección de los atributos
  dropdownA = Select(driver.find_element('id',
'_ctl0_MainContentPlaceHolderMaster_rankingsDropDownList'))
  dropdownA.select_by_visible_text(atributo)
  time.sleep(1) # tiempo de carga
  # selección de los atributos
  dropdownN = Select(driver.find_element('id',
'_ctl0_MainContentPlaceHolderMaster_nacionalDropDownList'))
  dropdownN.select_by_visible_text(nacionalidad)
  time.sleep(1) # tiempo de carga

  # obtenemos el número de páginas del paginador
  pages =
len(driver.find_element(By.CLASS_NAME,'tabla-paginador').find_elements(By.TAG_NA
ME,'a'))

  return pages
```

def collect_attributes()

Esta función se encarga de recoger los datos, para ellos se ha utilizado una selección de la tabla mediante ID y después una selección de la fila mediante TAG, de esta forma vamos almacenando en un vector los datos de las filas.

Cuando termina, hacemos scroll para poder llegar a la paginación e ir haciendo clic en las distintas páginas mediante un LINK_TEXT. Para hacer esta navegación se ha implementado un bucle cuya salida depende del número de páginas obtenidas en la función anterior y que se han pasado como variable en esta.

La salida será una lista que contiene todos los datos recopilados.

```
def collect_attributes(lim):
  """
  Recopila los datos y los guarda en una lista
  """
  data = []
  page = 2
  while page <= lim + 2:
    time.sleep(2)
    selection = driver.find_element(By.ID,
'_ctl0_MainContentPlaceHolderMaster_rankingAcumuladosDataGrid').find_elements(B
y.TAG_NAME,'td')
    for sel in selection:
      if sel.text != '' and sel.text != '1 2' and sel.text != '1 2 3':
        data.append(sel.text)
    # Para hacer scroll
    driver.execute_script('window.scrollTo(0,document.body.scrollHeight)')
    # Damos tiempo
    time.sleep(2)
    # Hacemos clic en el siguiente enlace de paginación
    if page < lim + 2:
      button_page = driver.find_element(By.LINK_TEXT, str(page)).click()
    page = page + 1
  return data
```

def create df()

Esta función genera un dataframe a partir de los datos obtenidos de la función anterior.

Recibe como entradas la lista, la temporada a la que pertenecen los datos, el atributo escogido (Puntos, Rebotes Totales, etc.) y la nacionalidad de la jugadora.

La particularidad de este código es que se genera un primer dataframe con seis columnas cuando recibe el primer atributo de la lista y después, para el resto de atributos, genera dataframes con dos columnas que posteriormente se unirán mediante la columna Jugadora que es la que se ha tomado como referencia para la unión.

Además se ha realizado un tratamiento de duplicados, ya que había jugadoras que a mitad de temporada cambiaron de equipo, al ser pocas observaciones con respecto al global, en concreto cuatro, se ha optado por eliminarlas.

La salida de la función es un dataframe.

```
def create_df (lista, temporada, atributo, nacionalidad):
  """
  Generamos un dataframe con los datos con los datos de los atributos
  """
  if atributo == 'Puntos':
    df = pd.DataFrame(columns=['Temporada','Jugadora', 'Nacionalidad',
                                          'Equipo', atributo, 'Partidos'])
    for i in range(int(len(lista)/5)):
      dict_fila = {'Temporada': [temporada],
                   'Jugadora': [lista[5*i]],
                   'Nacionalidad': [nacionalidad],
                   'Equipo': [lista[5*i+1]],
                   atributo:[lista[5*i+2]],
                   'Partidos':[lista[5*i+3]]}
      df_fila = pd.DataFrame(dict_fila)
      df = pd.concat([df, df_fila], ignore_index = True)
  else:
    df = pd.DataFrame(columns=['Jugadora', atributo])
    for i in range(int(len(lista)/5)):
      dict_fila = {'Jugadora': [lista[5*i]],
                    atributo:[lista[5*i+2]]}
      df_fila = pd.DataFrame(dict_fila)
      df = pd.concat([df, df_fila], ignore_index = True)
  df = df.sort_values('Jugadora')
  # Tratamiento de jugadoras duplicadas que han cambiado de equipo durante
la liga
  if df["Jugadora"].duplicated(keep = False).any() == True:
    bool_series = df["Jugadora"].duplicated(keep = False)
    df = df[~bool_series]
  return df
```

def webscraper()

En esta última función realizamos el proceso recursivo de ir extrayendo datos e implementando un dataframe en función de las temporadas, los atributos y la nacionalidad.

Las variables de entrada se han codificado de la siguiente forma:

```
# variaciones que vamos a utilizar para extraer la información
attributes = ['Puntos', 'Rebotes Totales', 'Rebotes Ofensivos', 'Rebotes
Defensivos', 'Asistencias', 'Balones recuperados', 'Balones perdidos',
'Tapones a favor',
'Tapones en contra', 'Faltas recibidas', 'Faltas cometidas',
'Valoración', 'Minutos jugados', '% Tiros de 2', '% Tiros de 3', '% Tiros
Libres']

seasons = {'2022/2023':'Liga Regular Único',
           '2021/2022':'Liga Regular Único',
           '2020/2021':'Liga Regular Único',
           '2019/2020':'Liga Regular Único',
           '2018/2019':'Liga Regular',
           '2017/2018':'Liga Regular Único',
           '2016/2017':'Liga Regular Único',
           '2015/2016':'Liga Regular Único',
           '2014/2015':'Liga Regular Único',
           '2013/2014':'Liga Regular Grupo Unico',
           '2012/2013':'Liga Regular Único',
           '2011/2012':'LIGA REGULAR Único',
           '2010/2011':'REGULAR ÚNICO',
           '2009/2010':'REGULAR ÚNICO',
           '2008/2009':'REGULAR ÚNICO',
           '2007/2008':'REGULAR ÚNICO',
           '2006/2007':'REGULAR ÚNICO',
           '2005/2006':'REGULAR',
           '2004/2005':'REGULAR',
           '2003/2004':'LIGA REGULAR ÚNICO'}
nationality = ['Nacional', 'Extranjero']
```

La principal dificultad en este caso ha sido manejar las distintas etiquetas del desplegable para llamar a la Liga Regular de Baloncesto. Para ello se ha implementado un diccionario que recoge temporadas y nombre de la liga.

Cada vez que finaliza una temporada, el dataframe se escribe en un archivo csv para después volver a iniciar todo el proceso de extracción.

Como salida de la función quitamos el navegador.

```
def webscraper(seasons, attributes, nationality):
  """
  Función que realiza el raspado web
  """
  for key in seasons:
    df_season = pd.DataFrame(columns=['Jugadora'])
    for att in attributes:
      df_complete = pd.DataFrame()
      for nat in nationality:
        lista = collect_attributes(select_context_navigation(key,
seasons[key], att, nat))
        df_partial = create_df(lista, key, att, nat)
        df_complete = pd.concat([df_complete, df_partial], ignore_index =
True)
      df_complete = df_complete.sort_values('Jugadora')
      df_season = pd.merge(df_season, df_complete, on='Jugadora',
how='outer')
    df_season.to_csv(r'/content/drive/MyDrive/PR1Datos/prueba2.csv', mode
= 'a', header = True, index = False)
  return driver.quit()
```

La ejecución final se realiza de la siguiente forma:

```
# Ruta de la página web
url = 'https://baloncestoenvivo.feb.es/rankings/lfendesa/4/2022'
# Navegar a una página web
driver = iniciar_chrome()
driver.get(url)
# Lanzamos la función
webscraper(seasons, attributes, nationality)
```

Librerías y versiones utilizadas

```
-----
google          NA
pandas          1.5.3
selenium        4.9.0
session_info    1.0.0
-----
```

10. Dataset

Enlace al dataset en Zenodo

11. Vídeo

https://drive.google.com/file/d/1hBT9gZSHV52Z5sen-TuyRer2VlVuL-DV/view?usp=share_link

12. Contribuciones

Contribuciones	Firma
Investigación previa	PLC, AZR
Redacción de las respuestas	PLC, AZR
Desarrollo del código	PLC, AZR
Participación en el vídeo	PLC, AZR

Bibliografía

- Sangüesa, A. A. (s. f.). *Solobasket*. Solobasket. https://www.solobasket.com/liga-endesa/el-boom-de-la-inteligencia-artificial-detectando-tendencias-atipicas-en-la-acb
- Subirats, L., Calvo, M. (2018). Web Scraping. Editorial UOC.
- Masip, D. (2019). El lenguaje Python. Editorial UOC.
- Lawson, R. (2015). Web Scraping with Python. Packt Publishing Ltd. Chapter 2. Scraping the Data.

Webgrafía

- Federacion Española de Baloncesto. (s. f.). https://www.feb.es/
- Competiciones FEB. (s. f.). https://baloncestoenvivo.feb.es/
- (2023, 24 febrero). Usar Google Colab para Web scraping con Selenium [Vídeo]. YouTube. https://www.youtube.com/watch?v=MYD7hnxjiRw
- FRIKIdelTO. (2022, 27 febrero). 025 Web Scraping 4: Configurar CHROME y SCRAPING con SELENIUM [curso Python] [Vídeo]. YouTube. https://www.youtube.com/watch?v=348iLgXfgvk
- Artificial Corner. (2021, 12 abril). Curso de Web Scraping en Python | Web Scraping Dinámico con Selenium [Nivel Intermedio] [Vídeo]. YouTube. https://www.youtube.com/watch?v=CHiwaFEUB1Y