EDA: partidos de ligas de Europa

Walter Andrés Castañeda Díaz
Universidad Nacional de Colombia

RESUMEN

Uno de los pasos más imprescindibles en un proyecto de ciencia de datos es el análisis exploratorio de datos. El análisis exploratorio en una base de datos se realiza con el fin de poder comprender y analizar que puede estar ocurriendo en dichos datos y que decisiones poder tomar frente a estas. El propósito de este proyecto es de explicar los pasos que se realizaron para la limpieza, análisis y tratamiento de una base de datos asociados a algunos partidos realizados en distintas ligas europeas de fútbol. Luego, se describirán los intentos(fracasos) que se realizaron, y finalmente las posibles ideas a desarrollar.

METODOLOGIA

A continuación se realizará la limpieza de dos tablas que contienen algunos partidos realizados en distintas ligas europeas de fútbol, y los atributos(características) de los equipos que jugaron dichos partidos. Luego, se unirán esta tablas para aplicar algunos algoritmos de machine learning, y predecir el número de goles que anotará el equipo en casa, mediante los atributos de dichos equipos y los visitantes. El primer paso a realizar es la obtención de la base de datos e importar las librerías posibles a realizar durante el proyecto.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sqlite3
import requests
import re
import nltk
import warnings
warnings.filterwarnings('ignore')
from pyod.models.knn import KNN
import time
import seaborn as sns
import math
import pylab
import scipy.stats as stats

path="D:\\SoccerGames\\archive\\database.sqlite"
```

Para poder leer cada una de las tablas en la base de datos, se debe realizar una conexión a sql.

```
con=sqlite3.connect(path)
cursor=con.cursor()
cursor.execute("SELECT name FROM sqlite_master WHERE type='table
 ';")
```

Dicha base de datos contiene 8 tablas con nombres como 'sqlite_sequence', 'Player_Attributes','Player','Match','League', 'Country', 'Team',,'Team_Attributes'. Primero se realizará la limpieza de la tabla llamada 'Match', con la que se puede obtener con el siguiente código:

```
query_match=" SELECT * FROM Match"
data_match=pd.read_sql_query(query_match,con)
data_match.head()
```

luego se observaron que columnas, que tipo de columnas y la dimensión que tiene dicha tabla.

```
data_match.shape
data_match.head()
data_match.columns
data_match.dtypes
```

Dado que hay columnas que tienen información sobre tasas de apuestas (las cuales no nos interesa), se eliminan.

```
lista1=["B365H","B365D","B365A","BWH","BWD","BWA",
       "IWH","IWD","IWA","LBH","LBD","LBA","PSH",
       "PSD","PSA","WHH","WHD","WHA","SJH","SJD",
       "SJA","VCH","VCD","VCA","GBH","GBD","GBA",
       "BSH","BSD","BSA"]

def eliminar_columnas(lista):
    for i in lista:
        del(data_match[i])

eliminar_columnas(lista1)
```

A continuación se procede a eliminar los valores nulos que puedan aparecer en la tabla mediante el siguiente código:

```
for i in data_match.columns:
    print(i+":",data_match[i].isnull().sum())

data_match.dropna(how="any",inplace=True)
```

luego de haber visto que algunas columnas como lo son: 'corner','foulcommit', 'card','shoton' y 'shotoff' estaban en formato html(posiblemente la información se obtuvo mediante web scraping), se determinó que la mejor forma de obtener esta información limpia fue mediante las funciones:

```
class otras_estadisticas:
    def corners(x):
        corners= data_match["corner"]
        soup3=soup=BeautifulSoup(corners.iloc[x],"lxml")
        return(len(soup3.find_all("elapsed")))

    def foul(x):
        fouls= data_match["foulcommit"]
        soup4=BeautifulSoup(fouls.iloc[x],"lxml")
        return(len(soup4.find_all("elapsed")))
```

```
def card(x):
    card=data_match["card"]
    soup5= BeautifulSoup(card.iloc[x],"lxml")
    return(len(soup5.find_all("elapsed")))

def cross(x):
    cross=data_match["cross"]
    soup5= BeautifulSoup(cross.iloc[x],"lxml")
    return(len(soup5.find_all("elapsed")))

def shoton(x):
    shoton=data_match["shoton"]
    soup5= BeautifulSoup(shoton.iloc[x],"lxml")
    return(len(soup5.find_all("elapsed")))

def shotoff(x):
    shotoff=data_match["shotoff"]
    soup5= BeautifulSoup(shotoff.iloc[x],"lxml")
    return(len(soup5.find_all("elapsed")))
```

dado que lo que se pretende hacer, es que la información sobre dichas variables coincidan partidos de la vida real, se creo la siguiente función para examinar si la información de las tablas coinciden con la de dichos partidos

```
def imp_todas_estadisticas(x):
    print("Tiros␣de␣esquina:",otras_estadisticas.corners(x))
    print("Faltas␣hechas:",otras_estadisticas.foul(x))
    print("Tarjetas:",otras_estadisticas.card(x))
    print("Pases␣cruzados:",otras_estadisticas.cross(x))
    print("Tiros␣a␣puerta:",otras_estadisticas.shoton(x))
    print("Tiros␣a␣fuera:",otras_estadisticas.shotoff(x))
```

Se observó que había otra columna que también tenia texto en formato html, pero decidí eliminarla.

```
del(data_match["possession"])
```

Ya con la información de dichas columnas limpias, se procede a adjuntar la información a la tabla mediante el código:

```
listass=[otras_estadisticas.corners,
         otras_estadisticas.foul,
         otras_estadisticas.card,
         otras_estadisticas.cross,
         otras_estadisticas.shoton,
         otras_estadisticas.shotoff]
kkk=["Tiros_esquina","Faltas",
     "Tarjetas","Pases_cruzados",
     "Tiros_a_puerta","Tiros_a_fuera"]

def valor(titulo,declase):
    l=[]
    for i in range(0,data_match.shape[0]):
        a=declase(i)
        l.append(a)
    data_match[titulo]=l

valor("Tiros_esquina",otras_estadisticas.corners)
valor("Faltas",otras_estadisticas.foul)
valor("Tarjetas",otras_estadisticas.card)
valor("Pases_cruzados",otras_estadisticas.cross)
valor("Tiros_a_puerta",otras_estadisticas.shoton)
valor("Tiros_a_fuera",otras_estadisticas.shotoff)
```

Ahora, dado que en un partido de fútbol la probabilidad que hallan cero faltas, cero tarjeta, cero tiros a puerta u otras estadísticas es nula, entonces se verifican y se eliminan dichos registros que contienen esta información errónea mediante el siguiente código:

```
for l in kkk:
    serie=data_match[l]
    print(serie[serie==0].count())

def f_eliminar(x,data):
    filas_a_eliminar=[]
    k=0
    for i in range(0,len(data[x])):
        if data[x].iloc[i]==0:
            filas_a_eliminar.append(i)
    indices=data.index[filas_a_eliminar]
    return(data.drop(indices,axis=0,inplace=True))

for l in kkk:
    f_eliminar(l,data_match)
```

Se observa que todavía en la tabla están incluidas las columnas con texto formato html, las cuales se proceden a eliminar.

```
lista2=["cross","goal","shoton","shotoff",
        "foulcommit","card","corner"]

eliminar_columnas(lista2)
```

Además, como existen columnas con atributos de jugadores, que no serán útiles por el momento, también serán eliminadas.

```
lista3=data_match.columns[11:55]

eliminar_columnas(lista3)
```

Se eliminan aquellos registros de que estén duplicados.

```
data_match.drop_duplicates()
```

Ya realizado la limpieza de lo tabla de partidos, se procede a limpiar la tabla que contiene los datos de los atributos de cada equipo.
Inicialmente se importa la tabla en la base de datos

```
query_team_atr="␣SELECT␣*␣FROM␣Team_Attributes␣"
data_team_atr=pd.read_sql_query(query_team_atr,con)
```

Se observa las columnas, los valores nulos y los valores duplicados en la tabla.

```
data_team_atr.columns
data_team_atr.isna().sum()
data_team_atr.duplicated().sum()
```

Dado que hay una sola columna llamada ’buildUpPlayDribbling’ que contiene mas del 60% de sus registros como valores nulos, se procede a eliminar dicha columna.

```
data_team_atr["buildUpPlayDribbling"].isna().sum()
len(data_team_atr["buildUpPlayDribbling"])
del(data_team_atr["buildUpPlayDribbling"])
```

Al observar la tabla, hay ciertas columnas que no quiero considerar, que se eliminan mediante el código:

```
for i in data_team_atr.columns:
    if "Class" in i:
        del(data_team_atr[i])
```

Ahora lo que se realiza es unir las dos tablas ya limpias, esto con el fin de que en dado caso de implementar algún modelo, tener la información en una única tabla. Primero se tomaran como llaves foráneas las columnas 'away_team_api_id' para la primera tabla y 'team_api_id' para la segunda. Primero se ajusta y se verifica que las columnas 'awa_team_api_id' y 'team_api_id' tengan los mismo valores únicos. Luego se hallan los índices que no compartan valores de estos.

```
ids_equi_vis=sorted(data_match["away_team_api_id"].unique())
ids_team=sorted(data_team_atr["team_api_id"].unique())

print(len(ids_equi_vis))
print(len(ids_team))

ids_equi_vis
ids_team

lista_ids=[]
jj=0
for i in range(0,len(ids_equi_vis)):
    while ids_team[jj]<ids_equi_vis[i]:
        lista_ids.append(ids_team[jj])
        ids_team.pop(jj)
    jj=jj+1

lista_ids.append(ids_equi_vis[-1])
ids_team.pop(-1)


len(lista_ids)

print(len(ids_equi_vis),len(ids_team))
```

Se eliminan las filas con dichos índices.

```
for i in range(0,len(lista_ids)):
    data_team_atr.drop(data_team_atr[data_team_atr["team_api_id"
     ]==lista_ids[i]].index,inplace=True)
```

Luego se procede a eliminar las columnas "id"y team_fifa_api_id".

```
lista4=["id","team_fifa_api_id"]
for i in lista4:
    del(data_team_atr[i])
```

Dado que ambas tablas tienen una columna llamada "date", a una de ellas se le cambia el nombre para no presentar confusiones y no presenten problemas. Se unen dichas tablas mediante el código:

```
data_team_atr.rename(columns{"date":"fecha_equipo"},inplace=True)
```

```
data_match=data_match.merge(data_team_atr,left_on=
"home_team_api_id",right_on="team_api_id")
```

Como lo que se pretende es que las fechas de los atributos de los equipos coincidan con aquel las fechas que se jugaron los partidos, entonces se procede a encontrar los índices donde estas fechas no coinciden. Luego se procede a eliminar dichos registros mediante el código:

```
t=[]
for i in range(0,data_match.shape[0]):
    if int(data_match["fecha_equipo"][i][:4])==2010 :
        if int(data_match["date"][i][:4])>(int(data_match["
         fecha_equipo"][i][:4])):
            t.append(i)
    elif int(data_match["fecha_equipo"][i][:4])==2015 :
        if int(data_match["date"][i][:4])<int(data_match["
         fecha_equipo"][i][:4]):
            t.append(i)
    elif int(data_match["date"][i][:4])!=int(data_match["
     fecha_equipo"][i][:4]) :
        t.append(i)

data_match.drop(t,inplace=True)
```

con esto se obtienen los atributos de los equipos que jugaron en casa. Ahora se procede de la misma forma para obtener los atributos de los equipos que jugaron de visitante. Para ello, inicialmente se cambian los nombres de algunas columnas y luego se procede a eliminar otras. Para unir ahora estas tablas se usan como llaves foráneas las columnas "away_team_api_id"y "team_api_id". Se realiza el mismo procedimiento anterior(no se pega el código completo ya que es el mismo procedimiento).

```
data_match=data_match.merge(data_team_atr,left_on="
 away_team_api_id",right_on="team_api_id")
```

Ahora, se crea una nueva tabla con la cual se realizará un análisis de las estadísticas y algunas visualizaciones de los datos contenidos en dicha tabla.

```
data_new=data_match[columnas_new_data]
```

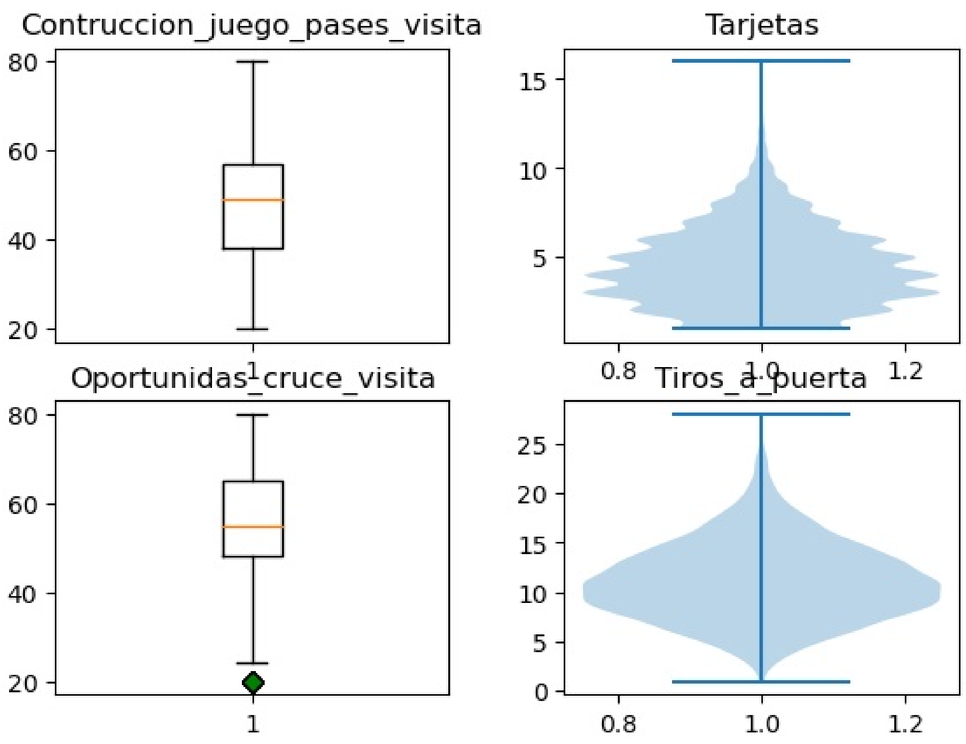
Tiros_a_fuera	Vel.juego_casa	Contruccion_juego_pases_casa	...	Agresividad_def_casa	Ancho_defensa_casa	Vel.juego_visita
10	61	44	...	56	50	62
10	61	52	...	55	48	62
4	61	52	...	55	48	62
15	48	50	...	44	51	62
9	55	49	...	54	50	62
13	64	49	...	44	49	62
15	51	41	...	48	60	62
15	51	41	...	48	60	62
8	45	43	...	42	48	62
7	45	43	...	42	48	62

ISe realizan diagramas de cajas y de violín para obtener información de las variables respecto a la normalidad y determinar la existencia de valores atípicos

```
def diagramas_caja(x):
    columna_0=columnas[x].reshape(2,2)
    fig1 ,axs1 = plt.subplots(2, 2)
    green_diamond = dict(markerfacecolor='g',marker='D')
    for i in range(0,2):
```

```
        for j in range(0,2):
            a=data_new[columna_0[i,j]].index
            axs1[i,j].boxplot(np.array(data_new[columna_0[i,j]]),
             flierprops=green_diamond)
            axs1[i,j].set_title(columna_0[i,j])

def diagramas_violin(x):
    columna_0=columnas[x].reshape(2,2)
    fig1 ,axs1 = plt.subplots(2, 2)
    for i in range(0,2):
        for j in range(0,2):
            a=data_new[columna_0[i,j]].index
            axs1[i,j].violinplot(data_new[columna_0[i,j]].tolist
             ())
            axs1[i,j].set_title(columna_0[i,j])
```



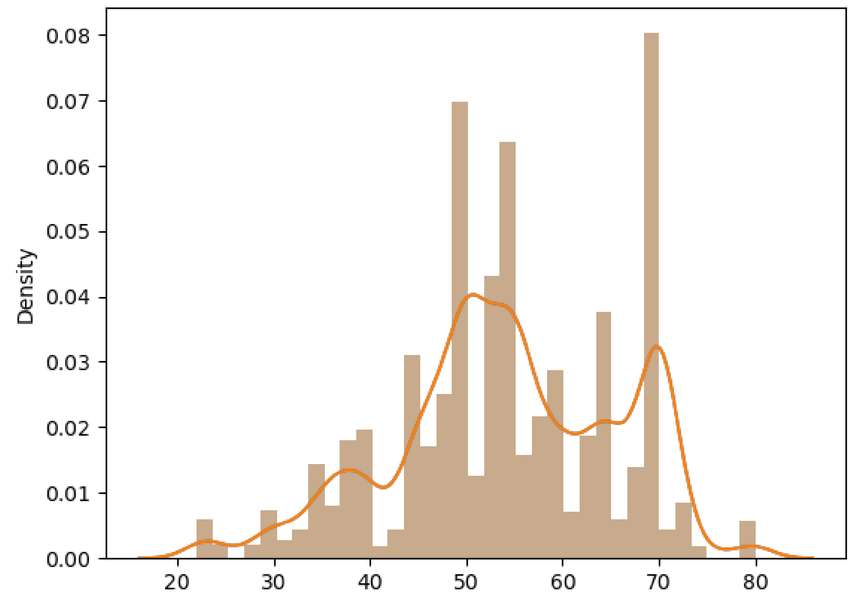
Dado que la mayoría de las variables tienen valores atípicos, se procede a eliminar estos mediante el siguiente código:

```
def eliminar_atipicos(columna):
    longitud=1
    data_new.index=np.array(
    range(len(data_new[columna])))
    while longitud>0:
        z = np.abs(stats.zscore(data_new[columna]))
        threshold = 3
        valores_atipicos=np.where(z > 3)[0]
        valores_atipicos=valores_atipicos.tolist()
        data_new.drop(valores_atipicos,inplace=True)
        data_new.index=np.array(range(len(data_new[columna])))
        longitud=len(valores_atipicos)

for i in data_new.columns:
    eliminar_atipicos(i)
```

Luego, se realiza diagramas de densidad de algunas variables mediante la función:

```
def diagrama_densidad(i):
    sns.distplot(x = data_new[i], kde = True)
    sns.distplot(x = data_new[i], kde = True)
    plt.show
```



Se realizan pruebas de normalidad.

```
alpha=0.05
for i in data_new.columns:
    a,b=stats.shapiro(data_new[i])
    print("Statistics",a,"p-value",b)
    if alpha>= b:
        print(i)
        print("Se␣rechaza␣H0,␣por␣tanto␣no␣hay␣normalidad")
    else:
        print("No␣se␣rechaza␣H0,␣existe␣normalidad")
```

Luego de verificar mediante varias transformaciones y no poder determinar la causa de la no normalidad, se procese a trabajar con modelos no perimétricos. Para ello, se aplicaran 2 modelos tradicionales de machine learning como son K-nearest neighbors y Random Forest para tratar de predecir el número de goles que anotarán los equipos que juegan en casa. Primero se importan las librerías para implementar dichos algoritmos.

```
from matplotlib.colors import ListedColormap
import matplotlib.patches as mpatches
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from scipy.stats import kendalltau
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

Primero se realiza un test de kendal para saber la relación entre el número de goles con el resto de las variables que me ayudarán a predecir, las cuales que se encuentran en las lista ”data_atr_pre_gc”.

```
for i in data_atr_pre_gc:
    coef, p = kendalltau(data_new["Goles_casa"], data_new[i])
    print("Goles_casa␣:␣"+ i + "␣:=␣",coef)
```

Primero se aplicará k-nearest neighbors. Para ello se pondrán los nombre a dos variables como ”X”a un dataframe de las variables que ayudaran a predecir y a ”y”el target. Luego, se divide las tablas en los datos de entrenamiento y de prueba pero normalizadas mediante MinMaxScaler().

```
X_train, X_test, y_train,y_test=train_test_split(X,y,random_state
 =0)

scaler=MinMaxScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)
```

Se aplica K-nearest neighbors con k=10, mediante el código:

```
n_neighbors=10
knn=KNeighborsClassifier(n_neighbors)
knn.fit(X_train,y_train)

pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.24	0.32	0.28	389
1	0.38	0.50	0.43	603
2	0.31	0.25	0.28	443
3	0.12	0.05	0.08	202
4	0.00	0.00	0.00	101
5	0.33	0.04	0.07	27
accuracy			0.31	1765
macro avg	0.23	0.19	0.19	1765
weighted avg	0.28	0.31	0.29	1765

Se observa que el modelo es realmente malo, por lo tanto se procede a implementar el modelo de Random Forest.

```
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print("Exactitud de Random Forest es ",metrics.accuracy_score(
 y_test, y_pred))
```

```
Exactitud de Random Forest es  0.3008498583569405
```

RESUMEN DE INTENTOS

A continuación se realizará una lista de los intentos superados:

- El primer obstáculo era poder cargar una base de datos en formato SQ-Lite, debido a que solo sabia importar archivos de tipo csv o xlsx.
- Al momento de obtener la información de aquellas columnas que contenían texto en formato html no sabia como poder obtener dicha información.
- Para la unión de las tablas fue complicado debido a que se buscaba que las fechas de los atributos de los equipos coincidiera con la temporada en se jugaban los partidos.

A continuación se dará una breve lista de los fracasos:

- No se pudo disminuir el tiempo de ejecución de algunos códigos.
- No se logró aumentar la exactitud de los modelos, debido a una poca profundización en lo que puede suceder en las variables.

Las herramientas que principalmente he utilizado para la realización de este proyecto fueron la siguientes paginas web:

- `https://www.geeksforgeeks.org/` donde se encuentran diferentes ejemplos de códigos con python y otras herramientas de programación.
- `https://www.w3schools.com/` igual que el anterior pagina se encuentra distintos ejemplos de codigo con python
- `https://www.aprendemachinelearning.com/` es una pagina web donde explican como funcionan diferentes algoritmos de machine learning.
- `https://nasa.github.io/nasa-latex-docs/html/examples/listing.html` en esta pagina se explica como poder introducir texto como si fuese en formato de python.
- `https://www.overleaf.com/latex/templates/postersifisc/zgnpqwtnrrrx` es la plantilla sobre la cual se realizo el informe.

Otras herramientas fueron tomadas de otros cursos o lecturas anteriores a este proyecto.

El conjunto de datos fue extraído de la pagina `https://www.kaggle.com/datasets/hugomathien/soccer`. Los datos contienen

- +25.000 partidos de fútbol en algunas ligas europeas
- +10,000 jugadores
- 11 países europeos con su campeonato líder
- Temporadas 2008 a 2016
- Atributos de jugadores y equipos* extraídos de la serie de videojuegos FIFA de EA Sports, incluidas las actualizaciones semanales
- Línea de equipo con formación de escuadrón (coordenadas X, Y)
- Eventos de partidos detallados (tipos de goles, posesión, córner, centros, faltas, tarjetas, etc.) para +10,000 partidos, entre otros.

IDEAS POSIBLES A DESARROLLAR

A continuación habrá una lista de las posibles mejoras a este proyecto:

- Poder realizar un análisis más profundo a lo que sucede con las variables, especialmente si es necesario adquirir más datos. Esto se tendria pensado mediante Web Scraping.
- Intentar implementar otros algoritmos tradicionales de machine learning, y si no es posible aumentar la predicción con estos usar redes neuronales.
- Tratar de predecir otros variables como faltas, tarjetas u otras.