JAUME PARDO SOCIAS

GitHub: https://github.com/JaumePardo/FifaPractice

Fifa practice

Queremos predecir el valor de los jugadores usando la informacion de cada jugador. Por eso primero trataremos los datos, los normalizaremos y usaremos un modelo de regresión lineal para predecir el valor.

Importamos las librerias

Primero de todo debemos importar algunas librerias para poder trabajar con valores numericos y dataframes

In [1]:

```
import os

from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import pandas as pd
import numpy as np
import math
import warnings
warnings.filterwarnings('ignore')
```

Leemos los datos

Para leer los datos utilizamos la libreria `pandas`

In [2]:

```
df = pd.read_csv("fifa.csv")
df
```

Out[2]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	Overall	P
0	0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	
1	1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	
2	2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	
3	3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	
4	4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	
...	...	...	...	...	...	...	...	...	...
18202	18202	238813	J. Lundstram	19	https://cdn.sofifa.org/players/4/19/238813.png	England	https://cdn.sofifa.org/flags/14.png	47	
18203	18203	243165	N. Christoffersson	19	https://cdn.sofifa.org/players/4/19/243165.png	Sweden	https://cdn.sofifa.org/flags/46.png	47	
18204	18204	241638	B. Worman	16	https://cdn.sofifa.org/players/4/19/241638.png	England	https://cdn.sofifa.org/flags/14.png	47	
18205	18205	246268	D. Walker-Rice	17	https://cdn.sofifa.org/players/4/19/246268.png	England	https://cdn.sofifa.org/flags/14.png	47	
18206	18206	246269	G. Nugent	16	https://cdn.sofifa.org/players/4/19/246269.png	England	https://cdn.sofifa.org/flags/14.png	46	

18207 rows × 89 columns

Analisis de datos

Primeramente queremos saber que le estan pasando los datos. Para esto necesitamos poder mostrar todas las columnas de la tabla. Tambien mostraremos la estructura de los datos numericos.

In [3]:

```
pd.set_option('display.max_columns', None)
```

In [4]:

```
df.head()
```

Out[4]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	
0	0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	94	
1	1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	94	
2	2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	93	F
3	3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	93	M
4	4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	92	M

In [5]:

```
df.describe()
```

Out[5]:

	Unnamed: 0	ID	Age	Overall	Potential	Special	International Reputation	Weak Foot	Skill Moves	
count	18207.000000	18207.000000	18207.000000	18207.000000	18207.000000	18207.000000	18159.000000	18159.000000	18159.000000	1
mean	9103.000000	214298.338606	25.122206	66.238699	71.307299	1597.809908	1.113222	2.947299	2.361308	
std	5256.052511	29965.244204	4.669943	6.908930	6.136496	272.586016	0.394031	0.660456	0.756164	
min	0.000000	16.000000	16.000000	46.000000	48.000000	731.000000	1.000000	1.000000	1.000000	
25%	4551.500000	200315.500000	21.000000	62.000000	67.000000	1457.000000	1.000000	3.000000	2.000000	
50%	9103.000000	221759.000000	25.000000	66.000000	71.000000	1635.000000	1.000000	3.000000	2.000000	
75%	13654.500000	236529.500000	28.000000	71.000000	75.000000	1787.000000	1.000000	3.000000	3.000000	
max	18206.000000	246620.000000	45.000000	94.000000	95.000000	2346.000000	5.000000	5.000000	5.000000	

In [6]:

```
df = df.iloc[:, 2:]
```

Podemos ver como las dos primeras columnas no nos dan información importante para poder predecir el valor del jugador. Tambien las columnas del nombre del jugador, **Name** , la foto del jugador,**Photo**, y la cara, **Real Face** no son importantes ya que cada jugador tendrá su propio valor y no nos serviran para predecir. Tambien la foto de la bandera del pais, **Flag** , y el logo del club, **Club Logo**, no nos aportaran nueva informacion importante si ya tenemos la nacionalidad de cada jugador y el nombre de su club. Aparte la columna **Special** depende de las puntuaciones de las diferentes posiciones del jugador, por esto tampoco nos aportan nueva información. Tampoco las columnas de cuando se ha unido el jugador al club, **Joined** , y cuando terminará su contrato nos aportara información, **Contract Valid Until** , así que podemos eliminar todas estas columnas.

In [7]:

```
df = df.drop(columns = ["Name","Photo","Flag","Club Logo","Real Face","Joined","Special","Contract Valid
```

Handle NaNs

Podemos ver las columnas que tienen almenos un NaN.

In [8]:

```
df.columns[df.isna().any()].tolist()
```

Out[8]:

```
['Club',
 'Preferred Foot',
 'International Reputation',
 'Weak Foot',
 'Skill Moves',
 'Work Rate',
 'Body Type',
 'Position',
 'Jersey Number',
 'Loaned From',
 'Height',
 'Weight',
 'LS',
 'ST',
 'RS',
 'LW',
 'LF',
 'CF',
 'RF',
 'RW',
 'LAM',
 'CAM',
 'RAM',
 'LM',
 'LCM',
 'CM',
 'RCM',
 'RM',
 'LWB',
 'LDM',
 'CDM',
 'RDM',
 'RWB',
 'LB',
 'LCB',
 'CB',
 'RCB',
 'RB',
 'Crossing',
 'Finishing',
 'HeadingAccuracy',
 'ShortPassing',
 'Volleys',
 'Dribbling',
 'Curve',
 'FKAccuracy',
 'LongPassing',
 'BallControl',
 'Acceleration',
 'SprintSpeed',
 'Agility',
 'Reactions',
 'Balance',
 'ShotPower',
 'Jumping',
 'Stamina',
 'Strength',
 'LongShots',
 'Aggression',
 'Interceptions',
 'Positioning',
 'Vision',
 'Penalties',
 'Composure',
 'Marking',
 'StandingTackle',
 'SlidingTackle',
 'GKDiving',
 'GKHandling',
 'GKKicking',
 'GKPositioning',
 'GKReflexes',
 'Release Clause']
```

In [9]:

```
df[df.isna().any(axis=1)]
```

Out[9]:

	Age	Nationality	Overall	Potential	Club	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Work Rate	Body Type	Position	Je Nun
0	31	Argentina	94	94	FC Barcelona	€110.5M	€565K	Left	5.0	4.0	4.0	Medium/ Medium	Messi	RF	
1	33	Portugal	94	94	Juventus	€77M	€405K	Right	5.0	4.0	5.0	High/ Low	C. Ronaldo	ST	
2	26	Brazil	92	93	Paris Saint-Germain	€118.5M	€290K	Right	5.0	5.0	5.0	High/ Medium	Neymar	LW	
3	27	Spain	91	93	Manchester United	€72M	€260K	Right	4.0	3.0	1.0	Medium/ Medium	Lean	GK	
4	27	Belgium	91	92	Manchester City	€102M	€355K	Right	4.0	5.0	4.0	High/ High	Normal	RCM	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
18202	19	England	47	65	Crewe Alexandra	€60K	€1K	Right	1.0	2.0	2.0	Medium/ Medium	Lean	CM	
18203	19	Sweden	47	63	Trelleborgs FF	€60K	€1K	Right	1.0	2.0	2.0	Medium/ Medium	Normal	ST	
18204	16	England	47	67	Cambridge United	€60K	€1K	Right	1.0	3.0	2.0	Medium/ Medium	Normal	ST	
18205	17	England	47	66	Tranmere Rovers	€60K	€1K	Right	1.0	3.0	2.0	Medium/ Medium	Lean	RW	
18206	16	England	46	66	Tranmere Rovers	€60K	€1K	Right	1.0	3.0	2.0	Medium/ Medium	Lean	CM	

18207 rows × 79 columns

Club y Loaned From

Podemos ver que hay muchos de jugadores que tienen al menos un atriubuto a NaN. Por eso no podemos quitar todos estos jugadores ya que sino nos quedaríamos con pocos jugadores para hacer la predicción. Podemos ver muchas columnas que tienen casi siempre NaN así como la columna **Loaned From**. Algun problema con las columnas de David DeGea. También podemos ver como la columna club tiene NaN. Que podemos hacer?

In [10]:

```
df.loc[df.Club != df.Club]
```

Out[10]:

	Age	Nationality	Overall	Potential	Club	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Work Rate	Body Type	Position	Jersey Number	Loan Fr
452	24	Argentina	80	85	NaN	€0	€0	Right	2.0	4.0	4.0	Medium/ Medium	Normal	CM	5.0	N
538	33	Sweden	80	80	NaN	€0	€0	Right	2.0	4.0	2.0	High/ Medium	Normal	LCB	4.0	N
568	26	Russia	79	81	NaN	€0	€0	Right	1.0	3.0	1.0	Medium/ Medium	Normal	GK	12.0	N
677	29	Russia	79	79	NaN	€0	€0	Right	2.0	3.0	3.0	High/ High	Lean	RB	2.0	N
874	29	Russia	78	78	NaN	€0	€0	Right	2.0	3.0	3.0	High/ Medium	Stocky	ST	22.0	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
17197	21	India	55	64	NaN	€0	€0	Right	1.0	2.0	1.0	Medium/ Medium	Normal	GK	1.0	N
17215	26	Finland	55	57	NaN	€0	€0	Right	1.0	3.0	2.0	Medium/ High	Normal	RB	3.0	N
17339	23	India	54	63	NaN	€0	€0	Right	1.0	3.0	2.0	Medium/ Low	Normal	NaN	NaN	N
17436	20	India	54	67	NaN	€0	€0	Right	1.0	3.0	2.0	Medium/ Medium	Normal	NaN	NaN	N
17539	21	India	53	62	NaN	€0	€0	Right	1.0	3.0	2.0	High/ Medium	Lean	NaN	NaN	N

241 rows × 79 columns

El problema es que hay jugadores que su club no esta en el FIFA. Por ejemplo el jugador A. Granqvist. Como hay pocos jugadores en relacion al total, con el club igual a NaN, podemos intentar de eliminarlos. Lo podemos hacer utilizando la siguiente instruccion.

In [11]:

```
df = df.dropna(subset = ['Club'])
df
```

Out[11]:

	Age	Nationality	Overall	Potential	Club	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Work Rate	Body Type	Position	Je Nur
0	31	Argentina	94	94	FC Barcelona	€110.5M	€565K	Left	5.0	4.0	4.0	Medium/ Medium	Messi	RF	
1	33	Portugal	94	94	Juventus	€77M	€405K	Right	5.0	4.0	5.0	High/ Low	C. Ronaldo	ST	
2	26	Brazil	92	93	Paris Saint-Germain	€118.5M	€290K	Right	5.0	5.0	5.0	High/ Medium	Neymar	LW	
3	27	Spain	91	93	Manchester United	€72M	€260K	Right	4.0	3.0	1.0	Medium/ Medium	Lean	GK	
4	27	Belgium	91	92	Manchester City	€102M	€355K	Right	4.0	5.0	4.0	High/ High	Normal	RCM	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
18202	19	England	47	65	Crewe Alexandra	€60K	€1K	Right	1.0	2.0	2.0	Medium/ Medium	Lean	CM	
18203	19	Sweden	47	63	Trelleborgs FF	€60K	€1K	Right	1.0	2.0	2.0	Medium/ Medium	Normal	ST	
18204	16	England	47	67	Cambridge United	€60K	€1K	Right	1.0	3.0	2.0	Medium/ Medium	Normal	ST	
18205	17	England	47	66	Tranmere Rovers	€60K	€1K	Right	1.0	3.0	2.0	Medium/ Medium	Lean	RW	
18206	16	England	46	66	Tranmere Rovers	€60K	€1K	Right	1.0	3.0	2.0	Medium/ Medium	Lean	CM	

17966 rows × 79 columns

Ya hemos quitado todos los jugadores del que no sabemos el Club. Aun así seguimos teniendo un problema, como utilizamos esta información para un algoritmo? Primeramente vamos a cambiar los valores de los jugadores que han sido cedidos por un club. Para poder saber a que club pertenecen realmente vamos a cambiar su club actual por el club que lo han cedido. Por esto primero rellenaremos con un 0 a todos los jugadores que no han sido cedidos

In [12]:

```
df[df.isna()["Loaned From"]==False]
```

Out[12]:

	Age	Nationality	Overall	Potential	Club	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Work Rate	Body Type	Position	Jers Numb
28	26	Colombia	88	89	FC Bayern München	€69.5M	€315K	Left	4.0	3.0	4.0	Medium/ Medium	Normal	LAM	10
38	30	Argentina	88	88	Milan	€57M	€245K	Right	4.0	4.0	3.0	High/ Medium	Normal	LS	9
91	29	Brazil	85	85	Guangzhou Evergrande Taobao FC	€37M	€235K	Right	3.0	3.0	3.0	High/ High	Lean	LDM	9
166	24	Brazil	83	90	Guangzhou Evergrande Taobao FC	€36.5M	€18K	Left	2.0	4.0	4.0	High/ Medium	Normal	CAM	24
176	24	Croatia	83	89	Chelsea	€35M	€165K	Right	3.0	4.0	4.0	High/ High	Normal	LCM	17
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
17978	21	England	51	57	Hamilton Academical FC	€50K	€3K	Right	1.0	2.0	2.0	Medium/ Medium	Lean	ST	16
17979	21	China PR	51	60	Guizhou Hengfeng FC	€60K	€2K	Right	1.0	2.0	2.0	Medium/ Medium	Normal	CM	8
18026	21	China PR	50	59	Guizhou Hengfeng FC	€50K	€2K	Right	1.0	2.0	2.0	Medium/ Medium	Lean	LM	29
18031	20	China PR	50	61	Stabæk Fotball	€40K	€2K	Right	1.0	3.0	2.0	Medium/ Medium	Normal	RB	98
18056	19	Italy	50	65	Ascoli	€60K	€3K	Left	1.0	3.0	2.0	Medium/ Medium	Lean	CM	27

1264 rows × 79 columns

In [13]:

```
df.iloc[38]["Club"]
```

Out[13]:

```
'Milan'
```

In [14]:

```
df.iloc[38]["Loaned From"]
```

Out[14]:

```
'Juventus'
```

Para cambiar el club que lo han cedido por el club original, primero vamos a rellenar los NaN de la columnas de cedidos por 0.

In [15]:

```
df["Loaned From"].fillna(0,inplace = True)
```

Con la siguiente instrucción cambiaremos los valores del club a los que estan en la columna Loaned From, si estos no estan con un 0.

In [16]:

```
df['Club'] = np.where(df['Loaned From'] != 0, df['Loaned From'], df["Club"])
```

Comprobamos que el procedimiento ha funcionado y un jugador que tenia el Club y el Loaned From diferente, ahora lo tienen igual

In [17]:

```
df.iloc[38]["Club"]
```

Out[17]:

```
'Juventus'
```

In [18]:

```
df.iloc[38]["Loaned From"]
```

Out[18]:

```
'Juventus'
```

Finalmente hacemos un one hot encoding con la columna club. Es decir, crearemos una columna para cada valor posible de la columa club y se indicara si el jugador esta en este club con un 1 y sino con un 0. Al final, cada jugador solo tendrà un 1 en una de estas columnas de clubs.

In [19]:

```
clb = df.pop("Club")

df = pd.concat([df.reset_index(drop=True), pd.get_dummies(clb, prefix='clb').reset_index(drop=True)], axi
df.head()
```

Out[19]:

	Age	Nationality	Overall	Potential	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Work Rate	Body Type	Position	Jersey Number	Loaned From	He
0	31	Argentina	94	94	€110.5M	€565K	Left	5.0	4.0	4.0	Medium/ Medium	Messi	RF	10.0	0	
1	33	Portugal	94	94	€77M	€405K	Right	5.0	4.0	5.0	High/ Low	C. Ronaldo	ST	7.0	0	
2	26	Brazil	92	93	€118.5M	€290K	Right	5.0	5.0	5.0	High/ Medium	Neymar	LW	10.0	0	
3	27	Spain	91	93	€72M	€260K	Right	4.0	3.0	1.0	Medium/ Medium	Lean	GK	1.0	0	
4	27	Belgium	91	92	€102M	€355K	Right	4.0	5.0	4.0	High/ High	Normal	RCM	7.0	0	

Value, Release Clause, Wage

Tambien tenemos un problema en nuestra variable objetivo **Value**, no tiene ningun NaN pero no es un numero. Por esto crearemos una función para convertir de formato texto a formato numérico.

In [20]:

```
df["Value"].isna().sum()
```

Out[20]:

```
0
```

In [21]:

```
def value_to_float(x):
    """
    From K and M to float.

    """
    x = x.replace('€', '')
    ret_val = 0.0

    if type(x) == float or type(x) == int:
        ret_val = x
    if 'K' in x:
        if len(x) > 1:
            ret_val = float(x.replace('K', ''))
        ret_val = ret_val *1000
    if 'M' in x:
        if len(x) > 1:
            ret_val = float(x.replace('M', ''))
        ret_val = ret_val * 1000000.0
    return ret_val
```

Tambien podemos observar como la columna de **Release Clause** tiene valores NaN. Estos valores son debidos a que su clausula de liberación ya ha terminado. Por esto podemos cambiar todos los NaN por la media ya que es una columna importante y hay bastantes jugadores con NaN.

In [22]:

```
df["Release Clause"].isna().sum()
```

Out[22]:

```
1323
```

In [23]:

```
mitjanaReleaseClause = df[df["Release Clause"].isna()==False]["Release Clause"].apply(value_to_float).mea
df["Release Clause"].fillna(str(mitjanaReleaseClause),inplace = True)
```

Ahora ya si que podemos aplicar la función sobre todas las columnas con valores en euros.

In [24]:

```
df["Value"] = df["Value"].apply(value_to_float)
df["Wage"] = df["Wage"].apply(value_to_float)
df["Release Clause"] = df["Release Clause"].apply(value_to_float)
df.head()
```

Out[24]:

	Age	Nationality	Overall	Potential	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Work Rate	Body Type	Position	Jersey Number	Loan Fr
0	31	Argentina	94	94	110500000.0	565000.0	Left	5.0	4.0	4.0	Medium/ Medium	Messi	RF	10.0	
1	33	Portugal	94	94	77000000.0	405000.0	Right	5.0	4.0	5.0	High/ Low	C. Ronaldo	ST	7.0	
2	26	Brazil	92	93	118500000.0	290000.0	Right	5.0	5.0	5.0	High/ Medium	Neymar	LW	10.0	
3	27	Spain	91	93	72000000.0	260000.0	Right	4.0	3.0	1.0	Medium/ Medium	Lean	GK	1.0	
4	27	Belgium	91	92	102000000.0	355000.0	Right	4.0	5.0	4.0	High/ High	Normal	RCM	7.0	

Nationality, Work Rate y Position

Podemos hacer el mismo procedimiento que en el club pero con la nacionalidad. Aplicar un one hot encoding para cada valor de nacionalidad.

In [25]:

```
nat = df.pop("Nationality")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(nat, prefix='nat').reset_index(drop=True)], axi
```

Para la variable categorica **Work Rate** realizaremos un one hot encoding para poder pasarlo a variables numericas y que el algoritmo las pueda utilizar. Tambien podemos ver que solo hay 48 variables con el **Work Rate** a NaN así que los podemos eliminar.

In [26]:

```
df["Work Rate"].value_counts()
```

Out[26]:

```
Medium/ Medium    9685
High/ Medium      3131
Medium/ High      1660
High/ High        1007
Medium/ Low        840
High/ Low          686
Low/ Medium        440
Low/ High          435
Low/ Low            34
Name: Work Rate, dtype: int64
```

In [27]:

```
df["Work Rate"].isna().sum()
```

Out[27]:

```
48
```

In [28]:

```
df = df.dropna(subset = ['Work Rate'])
WR = df.pop("Work Rate")
```

```
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(WR, prefix='wr').reset_index(drop=True)], axis=
```

Para la variable categorica **Position** tambien realizaremos un one hot encoding para poder pasarlo a variables numericas y que el algoritmo las pueda utilizar. Tambien podemos ver que no hay ningun valor NaN, así que podemos hacer el one hot encoding sin problemas.

In [29]:

```
df["Position"].isna().sum()
```

Out[29]:

```
0
```

In [30]:

```
pos = df.pop("Position")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(pos, prefix='pos').reset_index(drop=True)], axi
```

Preferred Foot y Loaned From

Podemos observar que el columna del pie dominante solo puede tener dos valores posibles: zurdo o diesto. Por esto para pasarlo a valores numericos basta aplicar una función lambda que si el valor es diestro se indique con un 1 y en caso contrario con un 0. Podemos ver que solo hay 48 NaN y por esto los podemos eliminar

In [31]:

```
df["Preferred Foot"].isna().sum()
```

Out[31]:

```
0
```

In [32]:

```
df["Preferred Foot"].value_counts()
```

Out[32]:

```
Right    13756
Left      4162
Name: Preferred Foot, dtype: int64
```

In [33]:

```
df["Preferred Foot"] = df["Preferred Foot"].apply(lambda x: 1 if x=="Right" else 0)
```

Tambien cambiaremos los datos de la columna **Loaned From**. En este caso si la columna ya tenia un 0 (un NaN anteriormente) lo dejaremos con un 0 y si tenia el nombre de un equipo le pondremos un 1.

In [34]:

```
df["Loaned From"] = df["Loaned From"].apply(lambda x: 1 if x!=0 else 0)
```

In [35]:

```
df["Loaned From"].value_counts()
```

Out[35]:

```
0    16654
1     1264
Name: Loaned From, dtype: int64
```

Body Type

En el caso de la variable **BodyType**, podemos observar que hay clases que solo aparecen una vez. Estas clases las podemos substituir por la clase moda, que en nuestro caso sera Normal. Eso lo hacemos aplicando una funcion tractamentBodyType a todos los valores de esta columna. Finalmente haremos un one hot encoding con esta variable.

In [36]:

```
df["Body Type"].value_counts()
```

Out[36]:

```
Normal                 10436
Lean                    6351
Stocky                  1124
Akinfenwa                  1
C. Ronaldo                 1
Shaqiri                    1
Messi                      1
PLAYER_BODY_TYPE_25        1
Courtois                   1
Neymar                     1
Name: Body Type, dtype: int64
```

In [37]:

```
moda = df["Body Type"].mode()[0]
```

In [38]:

```
def tratamientoBodyType(x):
  if x == "Normal" or x == "Lean" or x == "Stocky":
    return x
  return moda
```

In [39]:

```
df["Body Type"] = df["Body Type"].apply(tratamientoBodyType)
```

In [40]:

```
body = df.pop("Body Type")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(body, prefix='body').reset_index(drop=True)], a
```

Weight

La variable del peso es una cadena de texto que tendremos que transformar a numerica. El problema es que al final de cada medida tiene puesto las unidades que usa para representar el peso. Podemos ver que todo esta en libras, por esto basta eliminar el sufijo "lbs". Para esto utilizaremos la funcion rstrip en todos los valores de la columna **Weight**

In [41]:

```
print(type(df.iloc[0]["Weight"]))
df["Weight"].isna().sum()
```

```
<class 'str'>
```

Out[41]:

```
0
```

In [42]:

```
df = df.dropna(subset = ['Weight'])
```

In [43]:

```
def tratamientoWeight(x):
  x=x.rstrip("lbs")
  return float(x)
```

In [44]:

```
df["Weight"] = df["Weight"].apply(tratamientoWeight)
```

Height

Tambien tendremos que cambiar los valores de la columna **Height**. Podemos ver que los datos de la altura estan en pies y pulgadas separados por una coma. Podemos crear una función para separar los dos numeros por la coma, multiplicarlos por su conversion a cm y sumarlos.

In [45]:

```
def ConvertirAltura(x):
  x=x.split("'")
  pie = float(x[0]) * 30.48
  pulgada = float(x[1]) * 2.54
  return pie + pulgada
```

In [46]:

```
df["Height"].isna().sum()
```

Out[46]:

```
0
```

In [47]:

```
df["Height"] = df["Height"].apply(ConvertirAltura)
```

Porteros

Ahora arreglaremos las variables que nos indican la puntuacion concreta de los jugadores en una posicion dentro del FIFA. Esta puntuacion se puede mejorar con entrenamiento, por esto tiene un numero + otro numero, que la suma será el maximo que puede obtener el jugador con el entrenamiento. Por esto sumaremos estos dos valores y lo converitemos a float. Aparte podemos observar que todos los porteros tienen NaN en estas columnas. Por esto calcularemos la media de todos los otros jugadores que no tengan NaN en estas columnas, y se las añadiremos a los porteros.

In [48]:

```
def tractamentPuntuacioPosicions(x):
  x=x.split('+')
  return float(x[0]) + float(x[1])
```

Primero separaremos dos datasets, aquellos que tienen NaN en las columnas de la puntuacion (los porteros) y los que no. Despues calcularemos la media de todos los que no son porteros de todas estas columnas y pondremos estos valores en los porteros

In [49]:

```
dfSensePorters = df[df.isna()["LS"]==False]
dfPorters = df[df.isna()["LS"]==True]
```

In [50]:

```
df.head()
```

Out[50]:

	Age	Overall	Potential	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Jersey Number	Loaned From	Height	Weight	LS	ST	RS
0	31	94	94	110500000.0	565000.0	0	5.0	4.0	4.0	10.0	0	170.18	159.0	88+2	88+2	88+2
1	33	94	94	77000000.0	405000.0	1	5.0	4.0	5.0	7.0	0	187.96	183.0	91+3	91+3	91+3
2	26	92	93	118500000.0	290000.0	1	5.0	5.0	5.0	10.0	0	175.26	150.0	84+3	84+3	84+3
3	27	91	93	72000000.0	260000.0	1	4.0	3.0	1.0	1.0	0	193.04	168.0	NaN	NaN	NaN
4	27	91	92	102000000.0	355000.0	1	4.0	5.0	4.0	7.0	0	180.34	154.0	82+3	82+3	82+3

In [51]:

```
dfSensePorters["LS"] = dfSensePorters["LS"].apply(tractamentPuntuacioPosicions)
dfSensePorters["ST"] = dfSensePorters["ST"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RS"] = dfSensePorters["RS"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LW"] = dfSensePorters["LW"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LF"] = dfSensePorters["LF"].apply(tractamentPuntuacioPosicions)
dfSensePorters["CF"] = dfSensePorters["CF"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RF"] = dfSensePorters["RF"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RW"] = dfSensePorters["RW"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LAM"] = dfSensePorters["LAM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["CAM"] = dfSensePorters["CAM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RAM"] = dfSensePorters["RAM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LM"] = dfSensePorters["LM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LCM"] = dfSensePorters["LCM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["CM"] = dfSensePorters["CM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RCM"] = dfSensePorters["RCM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RM"] = dfSensePorters["RM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LWB"] = dfSensePorters["LWB"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LDM"] = dfSensePorters["LDM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["CDM"] = dfSensePorters["CDM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RDM"] = dfSensePorters["RDM"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RWB"] = dfSensePorters["RWB"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LB"] = dfSensePorters["LB"].apply(tractamentPuntuacioPosicions)
dfSensePorters["LCB"] = dfSensePorters["LCB"].apply(tractamentPuntuacioPosicions)
dfSensePorters["CB"] = dfSensePorters["CB"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RCB"] = dfSensePorters["RCB"].apply(tractamentPuntuacioPosicions)
dfSensePorters["RB"] = dfSensePorters["RB"].apply(tractamentPuntuacioPosicions)
```

In [52]:

```
dfPorters["LS"] = dfSensePorters["LS"].mean()
dfPorters["ST"] = dfSensePorters["ST"].mean()
dfPorters["RS"] = dfSensePorters["RS"].mean()
dfPorters["LW"] = dfSensePorters["LW"].mean()
dfPorters["LF"] = dfSensePorters["LF"].mean()
dfPorters["CF"] = dfSensePorters["CF"].mean()
dfPorters["RF"] = dfSensePorters["RF"].mean()
dfPorters["RW"] = dfSensePorters["RW"].mean()
dfPorters["LAM"] = dfSensePorters["LAM"].mean()
dfPorters["CAM"] = dfSensePorters["CAM"].mean()
dfPorters["RAM"] = dfSensePorters["RAM"].mean()
dfPorters["LM"] = dfSensePorters["LM"].mean()
dfPorters["LCM"] = dfSensePorters["LCM"].mean()
dfPorters["CM"] = dfSensePorters["CM"].mean()
dfPorters["RCM"] = dfSensePorters["RCM"].mean()
dfPorters["RM"] = dfSensePorters["RM"].mean()
dfPorters["LWB"] = dfSensePorters["LWB"].mean()
dfPorters["LDM"] = dfSensePorters["LDM"].mean()
dfPorters["CDM"] = dfSensePorters["CDM"].mean()
dfPorters["RDM"] = dfSensePorters["RDM"].mean()
dfPorters["RWB"] = dfSensePorters["RWB"].mean()
dfPorters["LB"] = dfSensePorters["LB"].mean()
dfPorters["LCB"] = dfSensePorters["LCB"].mean()
dfPorters["CB"] = dfSensePorters["CB"].mean()
dfPorters["RCB"] = dfSensePorters["RCB"].mean()
dfPorters["RB"] = dfSensePorters["RB"].mean()
```

In [53]:

```
dfPorters.head()
```

Out[53]:

	Age	Overall	Potential	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Jersey Number	Loaned From	Height	Weight	LS	
3	27	91	93	72000000.0	260000.0	1	4.0	3.0	1.0	1.0	0	193.04	168.0	59.842647	59.8426
9	25	90	93	68000000.0	94000.0	1	3.0	3.0	1.0	1.0	0	187.96	192.0	59.842647	59.8426
18	26	89	92	58000000.0	240000.0	1	3.0	4.0	1.0	22.0	0	187.96	187.0	59.842647	59.8426
19	26	89	90	53500000.0	240000.0	0	4.0	2.0	1.0	1.0	0	198.12	212.0	59.842647	59.8426
22	32	89	89	38000000.0	130000.0	1	5.0	4.0	1.0	1.0	0	193.04	203.0	59.842647	59.8426

Finalmente volveremos a juntar los dos datasets en uno.

In [54]:

```
df = pd.concat([dfSensePorters.reset_index(drop=True),dfPorters.reset_index(drop=True)])
```

In [55]:

```
df.head()
```

Out[55]:

	Age	Overall	Potential	Value	Wage	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Jersey Number	Loaned From	Height	Weight	LS	ST	RS	
0	31	94	94	110500000.0	565000.0	0	5.0	4.0	4.0	10.0	0	170.18	159.0	90.0	90.0	90.0	9
1	33	94	94	77000000.0	405000.0	1	5.0	4.0	5.0	7.0	0	187.96	183.0	94.0	94.0	94.0	9
2	26	92	93	118500000.0	290000.0	1	5.0	5.0	5.0	10.0	0	175.26	150.0	87.0	87.0	87.0	9
3	27	91	92	102000000.0	355000.0	1	4.0	5.0	4.0	7.0	0	180.34	154.0	85.0	85.0	85.0	9
4	27	91	91	93000000.0	340000.0	1	4.0	4.0	4.0	10.0	0	172.72	163.0	86.0	86.0	86.0	9

Normalización y predicción

Finalmente queremos intentar predecir la columna **value** a partir de todos los otros parametros de entrada. Separaremos la variable objetivo de las variables dependientes y despues separaremos el dataframe original en una parte de entrenamiento y otra de testeo. Esto nos servira para comprobar si el modelo generaliza bien y no tiene un alto error de varianza.

Despues normalizaremos tanto la parte de test como la de entrenamiento con los datos de entrenamiento. Normalizamos los datos ya que así el modelo podrá entrenar mas rápido ya que aprenderá a la misma velocidad para todas las variables dependientes. Aparte le dara la misma importancia a todos los datos.

In [56]:

```
val = df.pop("Value")
```

In [57]:

```
X_train, X_test, y_train, y_test = train_test_split(df, val, test_size=0.33, random_state=42)
```

Haremos la normalizacion estandar utilizando esta formula:

$$Z = \frac{x - \mu}{\sigma}$$

No normalizaremos las columnas que solo tienen valores binarios o booleanos ya que ya lo estan.

In [58]:

```
for i in X_test.columns:
  if i == "Loaned" or i == "Preferred Foot":
    pass
  elif i.startswith("clb_"):
    break
  else:
    X_test[i] = (X_test[i]-X_train[i].mean())/X_train[i].std()
    X_train[i] = (X_train[i]-X_train[i].mean())/X_train[i].std()
```

In [59]:

```
y_test = (y_test-y_train.mean())/y_train.std()
y_train = (y_train-y_train.mean())/y_train.std()
```

Ahora podemos entrenar nuestro modelo de regresion lineal y hacer una prediccion sobre la parte de testeo

In [60]:

```
reg = linear_model.LinearRegression().fit(X_train, y_train)
```

Utilizamos la metrica R^2 para la regresión, y usamos la implementación de sickit-learn. La metrica R^2 determina la calidad del modelo para generalizar los resultados, y la proporción de variación de los resultados que puede explicar. Sale de calcular el cuadrado del coeficiente

$$R^2 = \frac{\sigma_{XY}^2}{\sigma_X^2 \sigma_Y^2}$$

Donde:

- σ_{XY} es la covarianza de (X, Y)
- σ_X^2 es la Varianza de la variable X
- σ_Y^2 es la Varianza de la variable Y

de correlación de Pearson:

El R^2 adquiere valores entre 0 y 1. Existen casos dentro de la definición computacional de R^2 donde este valor puede tomar valores negativos si predice peor que predeciendo de forma aleatoria

In [61]:

```
preds = reg.predict(X_test)
r2_score(preds, y_test)
```

Out[61]:

```
-0.0018628953143737803
```

In [62]:

```
preds = reg.predict(X_train)
r2_score(preds, y_train)
```

Out[62]:

```
0.9618875584127543
```

Podemos obsevar que el modelo no sabe generalizar bien ya que nos hace una predicción muy buena prediciendo en el dataset del train con un R^2 de 0.9618, pero en el dataset del test sale una predicción de -0.0018. Es un claro caso de overfitting. Si miramos nuestro dataset podemos observar que tenemos muchas columnas que no nos aportan mucha información ya que hay muchos registros que tienen 0. Podemos observar que estas columnas son las que hemos creado a partir del one hot encoding. Para solucionarlo eliminaremos esas columnas que tienen menos de 30 jugadores con esos valores.

In [63]:

```
X_train, X_test, y_train, y_test = train_test_split(df, val, test_size=0.33, random_state=42)
```

In [64]:

```
for i in X_test.columns:
  if i == "Loaned" or i == "Preferred Foot":
    pass
  elif i.startswith("clb_") or i.startswith("nat_") or i.startswith("pos_") or i.startswith("WR_") or i.st
    if df[i].sum()<=30:
      X_train.pop(i)
      X_test.pop(i)
  else:
    X_test[i] = (X_test[i]-X_train[i].mean())/X_train[i].std()
    X_train[i] = (X_train[i]-X_train[i].mean())/X_train[i].std()
```

In [65]:

```
reg = linear_model.LinearRegression().fit(X_train, y_train)
```

In [66]:

```
preds = reg.predict(X_test)
```

In [67]:

```
r2_score(preds, y_test)
```

Out[67]:

```
0.9698045715448264
```