

# Fifa practice by Crist Alcázar

We want to predict the *value* of the players using the information of the player.

The files of this project are in this [repository](https://github.com/kralcazar/practica_inteligencia):

[https://github.com/kralcazar/practica\\_inteligencia](https://github.com/kralcazar/practica_inteligencia)

## Import libraries

First of all we import the libraries we're gonna use.

```
In [1]: import os
import time
import datetime

from sklearn.model_selection import train_test_split
from sklearn import linear_model
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import pandas as pd
import numpy as np
```

## Read the data

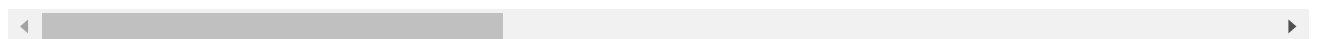
To read the data we're gonna use `pandas` and an utility from the library `os`.

```
In [2]: df = pd.read_csv(os.path.join("../", "in", "fifa.csv"))
df.head()
```

```
Out[2]:
```

|   | Unnamed: 0 | ID     | Name              | Age | Photo  | Nationality |                |
|---|------------|--------|-------------------|-----|--|-------------|----------------|
| 0 | 0          | 158023 | L. Messi          | 31  | https://cdn.sofifa.org/players/4/19/158023.png | Argentina   | https://cdn.so |
| 1 | 1          | 20801  | Cristiano Ronaldo | 33  | https://cdn.sofifa.org/players/4/19/20801.png  | Portugal    | https://cdn.so |
| 2 | 2          | 190871 | Neymar Jr         | 26  | https://cdn.sofifa.org/players/4/19/190871.png | Brazil      | https://cdn.so |
| 3 | 3          | 193080 | De Gea            | 27  | https://cdn.sofifa.org/players/4/19/193080.png | Spain       | https://cdn.so |
| 4 | 4          | 192985 | K. De Bruyne      | 27  | https://cdn.sofifa.org/players/4/19/192985.png | Belgium     | https://cdn.s  |

5 rows × 89 columns



## Data analysis

First of all we want to know what is happening to the data. To do so we increase the number of columns to show. After that we're gonna use a special instruction that will show us the data structure.

```
In [3]: pd.set_option('display.max_columns', None)
df.head()
```

Out[3]:

|   | Unnamed: 0 | ID     | Name              | Age | Photo  | Nationality |                |
|---|------------|--------|-------------------|-----|--|-------------|----------------|
| 0 | 0          | 158023 | L. Messi          | 31  | https://cdn.sofifa.org/players/4/19/158023.png | Argentina   | https://cdn.so |
| 1 | 1          | 20801  | Cristiano Ronaldo | 33  | https://cdn.sofifa.org/players/4/19/20801.png  | Portugal    | https://cdn.so |
| 2 | 2          | 190871 | Neymar Jr         | 26  | https://cdn.sofifa.org/players/4/19/190871.png | Brazil      | https://cdn.so |
| 3 | 3          | 193080 | De Gea            | 27  | https://cdn.sofifa.org/players/4/19/193080.png | Spain       | https://cdn.so |
| 4 | 4          | 192985 | K. De Bruyne      | 27  | https://cdn.sofifa.org/players/4/19/192985.png | Belgium     | https://cdn.s  |

## Cleaning non relevant columns

We can see that the two first columns do not provide any meaningful information so we can delete them. We will also clean all columns that, after being proved, do not contains relevant information because them worsen the score. The kepted columns have proved that benefits the final score.

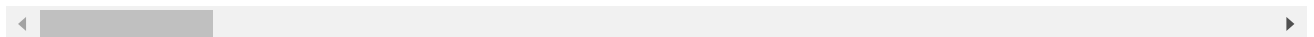
```
In [4]: df = df.iloc[:, 2:]
df.drop(['Name', 'Photo', 'Flag', 'Club Logo', 'Real Face', 'Body Type', 'Jersey Number'], axis=
df
```

Out[4]:

|       | Age | Nationality | Overall | Potential | Club                | Value   | Wage  | Special | Preferred Foot | International Reputation |
|-------|-----|-------------|---------|-----------|---------------------|---------|-------|---------|----------------|--------------------------|
| 0     | 31  | Argentina   | 94      | 94        | FC Barcelona        | €110.5M | €565K | 2202    | Left           | 5.0                      |
| 1     | 33  | Portugal    | 94      | 94        | Juventus            | €77M    | €405K | 2228    | Right          | 5.0                      |
| 2     | 26  | Brazil      | 92      | 93        | Paris Saint-Germain | €118.5M | €290K | 2143    | Right          | 5.0                      |
| 3     | 27  | Spain       | 91      | 93        | Manchester United   | €72M    | €260K | 1471    | Right          | 4.0                      |
| 4     | 27  | Belgium     | 91      | 92        | Manchester City     | €102M   | €355K | 2281    | Right          | 4.0                      |
| ...   | ... | ...         | ...     | ...       | ...                 | ...     | ...   | ...     | ...            | ...                      |
| 18202 | 19  | England     | 47      | 65        | Crewe Alexandra     | €60K    | €1K   | 1307    | Right          | 1.0                      |

|              | Age | Nationality | Overall | Potential | Club             | Value | Wage | Special | Preferred Foot | International Reputation |
|--------------|-----|-------------|---------|-----------|------------------|-------|------|---------|----------------|--------------------------|
| <b>18203</b> | 19  | Sweden      | 47      | 63        | Trelleborgs FF   | €60K  | €1K  | 1098    | Right          | 1.0                      |
| <b>18204</b> | 16  | England     | 47      | 67        | Cambridge United | €60K  | €1K  | 1189    | Right          | 1.0                      |
| <b>18205</b> | 17  | England     | 47      | 66        | Tranmere Rovers  | €60K  | €1K  | 1228    | Right          | 1.0                      |
| <b>18206</b> | 16  | England     | 46      | 66        | Tranmere Rovers  | €60K  | €1K  | 1321    | Right          | 1.0                      |

18207 rows × 80 columns



The data contains *NaN* (Not a Number). What it mean a *NaN*? What we do with *NaN*

## Handle NaNs

Let's see which columns contain NaNs.

```
In [5]: df.columns[df.isna().any()].tolist()
```

```
Out[5]: ['Club',
        'Preferred Foot',
        'International Reputation',
        'Weak Foot',
        'Skill Moves',
        'Work Rate',
        'Position',
        'Joined',
        'Loaned From',
        'Contract Valid Until',
        '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',
'GKDividing',
'GKHandling',
'GK Kicking',
'GK Positioning',
'GK Reflexes',
'Release Clause']

```

We can see that almost all players has a NaN. We can see multiple columns that always are a NaN as for example **Loaned From**. And some problematic player as David DeGea. And the rest of columns that we known that haves NaN as for example the **Club** columns what can we do?.

We can get all the rows with the column **Club** equals to NaN.

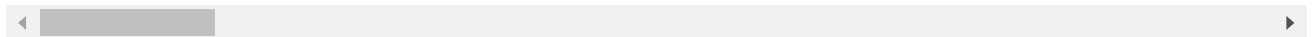
```
In [6]: df.loc[df.Club != df.Club]
```

Out[6]:

|              | Age | Nationality | Overall | Potential | Club | Value | Wage | Special | Preferred Foot | International Reputation | Weak Foot |
|--------------|-----|-------------|---------|-----------|------|-------|------|---------|----------------|--------------------------|-----------|
| <b>452</b>   | 24  | Argentina   | 80      | 85        | NaN  | €0    | €0   | 2122    | Right          | 2.0                      | 4.0       |
| <b>538</b>   | 33  | Sweden      | 80      | 80        | NaN  | €0    | €0   | 1797    | Right          | 2.0                      | 4.0       |
| <b>568</b>   | 26  | Russia      | 79      | 81        | NaN  | €0    | €0   | 1217    | Right          | 1.0                      | 3.0       |
| <b>677</b>   | 29  | Russia      | 79      | 79        | NaN  | €0    | €0   | 2038    | Right          | 2.0                      | 3.0       |
| <b>874</b>   | 29  | Russia      | 78      | 78        | NaN  | €0    | €0   | 1810    | Right          | 2.0                      | 3.0       |
| ...          | ... | ...         | ...     | ...       | ...  | ...   | ...  | ...     | ...            | ...                      | ...       |
| <b>17197</b> | 21  | India       | 55      | 64        | NaN  | €0    | €0   | 838     | Right          | 1.0                      | 2.0       |

|              | Age | Nationality | Overall | Potential | Club | Value | Wage | Special | Preferred Foot | International Reputation | Weak Foot |
|--------------|-----|-------------|---------|-----------|------|-------|------|---------|----------------|--------------------------|-----------|
| <b>17215</b> | 26  | Finland     | 55      | 57        | NaN  | €0    | €0   | 1366    | Right          | 1.0                      | 3.0       |
| <b>17339</b> | 23  | India       | 54      | 63        | NaN  | €0    | €0   | 1321    | Right          | 1.0                      | 3.0       |
| <b>17436</b> | 20  | India       | 54      | 67        | NaN  | €0    | €0   | 1270    | Right          | 1.0                      | 3.0       |
| <b>17539</b> | 21  | India       | 53      | 62        | NaN  | €0    | €0   | 1247    | Right          | 1.0                      | 3.0       |

241 rows × 80 columns



The problem of these players is that their club is not on FIFA.

So we can try to delete all the players with the club with *NaN*. To do so we can use the next instruction. We will also clean players without known position neither the goal keepers.

```
In [7]: # It's difficult to get the player's value without knowing their club
# These players also don't have information for ['Joined', 'Loaned From', 'Contract Valid
df = df.dropna(subset = ['Club'])
# Dropping the players who have NaN in position cleans a lot of NaNs scores like LS, ST, &
# It's difficult to get the player's value without knowing their positions
df = df.dropna(subset = ['Position'])
# For some reason the goal keeper information worsens the score, it also cleans a lot of I
df = df.drop(df[df['Position']=='GK'].index)
```

## Distribute non numerical columns

We can see that we removed all the players without a known club. Nevertheless we still have a problem, how to use this feature for an algorithm?

So we distribute the columns that do not have numerical values to become a unique column with value 0 if they do not have this value and 1 value if they do.

```
In [8]: clb = df.pop("Club")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(clb, prefix='clb').reset_index(drop=True)], axis=1)

prf = df.pop("Preferred Foot")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(prf, prefix='prf').reset_index(drop=True)], axis=1)

wr = df.pop("Work Rate")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(wr, prefix='wr').reset_index(drop=True)], axis=1)

pos = df.pop("Position")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(pos, prefix='pos').reset_index(drop=True)], axis=1)

nat = df.pop("Nationality")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(nat, prefix='nat').reset_index(drop=True)], axis=1)

lf = df.pop("Loaned From")
df = pd.concat([df.reset_index(drop=True), pd.get_dummies(lf, prefix='lf').reset_index(drop=True)], axis=1)

df.head()
```

Out[8]:

|   | Age | Overall | Potential | Value   | Wage  | Special | International<br>Reputation | Weak<br>Foot | Skill<br>Moves | Joined             | Contract<br>Valid<br>Until | Height |
|---|-----|---------|-----------|---------|-------|---------|-----------------------------|--------------|----------------|--------------------|----------------------------|--------|
| 0 | 31  | 94      | 94        | €110.5M | €565K | 2202    | 5.0                         | 4.0          | 4.0            | Jul 1,<br>2004     | 2021                       | 5      |
| 1 | 33  | 94      | 94        | €77M    | €405K | 2228    | 5.0                         | 4.0          | 5.0            | Jul 10,<br>2018    | 2022                       | 6      |
| 2 | 26  | 92      | 93        | €118.5M | €290K | 2143    | 5.0                         | 5.0          | 5.0            | Aug 3,<br>2017     | 2022                       | 5      |
| 3 | 27  | 91      | 92        | €102M   | €355K | 2281    | 4.0                         | 5.0          | 4.0            | Aug<br>30,<br>2015 | 2023                       | 5'11"  |
| 4 | 27  | 91      | 91        | €93M    | €340K | 2142    | 4.0                         | 4.0          | 4.0            | Jul 1,<br>2012     | 2020                       | 5      |

## Units and numerical formating

Now we fixed the non numerical columns but what about the **Values**, it does not have any NaN but it is not a number. How we convert the format?

First of all we define all the functions to pass from one format to the other:

```
In [9]: def value_to_int(x):
# From K and M to int.

ret_val = 0

# For non assigned clause release we will return 0 assuming that these players do not
if pd.isna(x) or x is np.nan or pd.isnull(x):
    return ret_val

# Remove € unit
x = x.replace('€', '')

# Removing and normalizing thousands and millions "shortcuts"
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
return int(ret_val)

def remove_lbs(x):
# Remove lbs unit
x = int(x.replace('lbs', ''))
return x

def sum_to_int(serie):
# Apply the sum in values like (89+2)
for index, value in serie.items():
    a = value.partition("+")[0]
```

```

        b = value.partition("+")[2]
        serie[index] = int(a) + int(b)
    return serie

def format_apostrophes_to_float(x):
    # Convert the british pound comma style to float
    return float(x.replace("'", "."))

def correct_nan_joined(age, joined):
    # Apply one single date for the players that do not have an assigned Joined date
    if pd.isna(joined) or joined is np.nan or pd.isnull(joined):
        # We assume that the players were being in club since they were young
        # 6 year old is the mean age for the players were joining a club
        # Other tested ages return a worse score
        return str(2020-int(age)+6)
    return joined

def stringDate_to_timestamp(str):
    # From date in string to timestamp numerical int
    value=0
    if len(str)==4: # For the single year date (2014)
        value= time.mktime(datetime.datetime.strptime(str, "%Y").timetuple())
    else: # For the full date (Jul 1, 2004)
        value=time.mktime(datetime.datetime.strptime(str, "%b %d, %Y").timetuple())
    return int(value)

```

Now we want to apply these functions to each value of the needed columns so we can train the prediction with them.

```

In [10]: # Apply for those that do not have Joined date
df['Joined'] = df.apply(lambda x: correct_nan_joined(x['Age'], x['Joined']), axis=1)
# Convert the string Joined dates to numerical dates
df["Joined"] = df["Joined"].apply(stringDate_to_timestamp)
# Convert the string Contract Valid Until dates to numerical dates
df["Contract Valid Until"] = df["Contract Valid Until"].apply(stringDate_to_timestamp)

# Format values to int with an unit in the value
df["Value"] = df["Value"].apply(value_to_int)
df["Wage"] = df["Wage"].apply(value_to_int)
df["Release Clause"] = df["Release Clause"].apply(value_to_int)
df["Weight"] = df["Weight"].apply(remove_lbs)
df["Height"] = df["Height"].apply(format_apostrophes_to_float)

# Apply the sum embedded in value
df[['LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW', 'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM',

# Show the results
df.head()

```

Out[10]:

|   | Age | Overall | Potential | Value     | Wage   | Special | International<br>Reputation | Weak<br>Foot | Skill<br>Moves | Joined     | Cont<br>Valid U |
|---|-----|---------|-----------|-----------|--------|---------|-----------------------------|--------------|----------------|------------|-----------------|
| 0 | 31  | 94      | 94        | 110500000 | 565000 | 2202    | 5.0                         | 4.0          | 4.0            | 1088632800 | 1609455         |
| 1 | 33  | 94      | 94        | 77000000  | 405000 | 2228    | 5.0                         | 4.0          | 5.0            | 1531173600 | 1640991         |
| 2 | 26  | 92      | 93        | 118500000 | 290000 | 2143    | 5.0                         | 5.0          | 5.0            | 1501711200 | 1640991         |
| 3 | 27  | 91      | 92        | 102000000 | 355000 | 2281    | 4.0                         | 5.0          | 4.0            | 1440885600 | 1672527         |
| 4 | 27  | 91      | 91        | 93000000  | 340000 | 2142    | 4.0                         | 4.0          | 4.0            | 1341093600 | 1577833         |

Now we have all columns with numerical and trainable values.

## Prediction

Finally we want to try to predict the **value**, to do so we store this column in the val variable.

```
In [11]: val = df.pop("Value")
```

Now we have the data we want to predict in a numeric format, the value information. So now we can use this data to train a model. But first we need to split the data to being able to know its true performance.

We assign a test size value, which it will be the non trained data for testing the results after the training. The recommendation is to set it at 20%.

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(df, val, test_size=0.20, random_state=
```

```
In [13]: len(X_train)
```

```
Out[13]: 12740
```

We have 12740 rows to train the model.

We will use a LinearRegression model because we need to obtain a continuous value from multidimensional dataset and can support some dominant classes and small variations in data without change its prediction too much.

The higher cost of the computing time is not an issue for this kind of training so the LinearRegression model for this problem is a good choice.

Let's do the train fitting the splitted data.

```
In [14]: reg = linear_model.LinearRegression().fit(X_train, y_train)
```

Predict the Value using the linear model of data testing splitted.

```
In [15]: preds = reg.predict(X_test)
```

Let's see the first row **predicted** value.

```
In [16]: preds[0]
```

```
Out[16]: 738540.7504741431
```

Let's see the first row **real** value.

```
In [17]: y_test[0]
```

```
Out[17]: 110500000
```

Let's see the coefficients.

```
In [18]: # The coefficients
print('Coefficients: \n', reg.coef_)
# The mean squared error
print('Mean squared error: %.2f' % mean_squared_error(y_test, preds))
```



```
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f' % r2_score(y_test, preds))
```

Coefficients:

```
[-56276.26181173  71126.96011495 -15303.90321458 ... 20011.34688828
  0.          -56102.01678503]
```

Mean squared error: 850784715073.95

Coefficient of determination: 0.98

Finally we get a metric  $R^2$  for the regression, we use the implementation from [scikit-learn](#).

```
In [19]: r2_score(preds, y_test)
```

```
Out[19]: 0.9785421951971136
```

We obtained a good score (97,85%)