Made by: Omar Kamand.

Subject: Artificial Intelligence (Machine Learning Part).

Project: Project GitHub link.

Final Project: Fifa practice

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

Import libraries

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

In [1]:
```
import os

from sklearn.model_selection import train_test_split # ML library, import only train and test
from sklearn import linear_model # Import for the model
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # To calculate efficiency and score of the model
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt
import pandas as pd # to translate from file to table (df=dataframe)
import numpy as np # Numeric python
```

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")) # Import dataframe
df.head() # See first 5 rows
```

Out[2]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	Club	...	Composure	Marking
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	FC Barcelona	...	96.0	33.0
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	Juventus	...	95.0	28.0
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	Paris Saint-Germain	...	94.0	27.0
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	Manchester United	...	68.0	15.0
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	Manchester City	...	88.0	68.0

5 rows × 89 columns

Now we select one columns. The result will be an object from the `pandas` library a `series` object. Thi object can be converted to a more standard `numpy` array.

In [3]:
```
name = df["Name"] # name = column "Name"
name # print name
```

Out[3]:
```
0                 L. Messi
1        Cristiano Ronaldo
2                Neymar Jr
3                   De Gea
4             K. De Bruyne
               ...
18202         J. Lundstram
18203    N. Christoffersson
18204             B. Worman
18205       D. Walker-Rice
18206             G. Nugent
Name: Name, Length: 18207, dtype: object
```

Data analysis

First of all we want to known 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 [4]:
```
pd.set_option('display.max_columns', None) # Show all columns.
```

In [5]:
```
df.head()
```

Out[5]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	Club	Club Logo
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	FC Barcelona	https://cdn.sofifa.org/teams/2/light/241.png
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	Juventus	https://cdn.sofifa.org/teams/2/light/45.png
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	Paris Saint-Germain	https://cdn.sofifa.org/teams/2/light/73.png
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	Manchester United	https://cdn.sofifa.org/teams/2/light/11.png
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	Manchester City	https://cdn.sofifa.org/teams/2/light/10.png

In [6]:
```
df.describe() # Statistics of every column
```

Out[6]:

	Unnamed: 0	ID	Age	Overall	Potential	Special	International Reputation	Weak Foot	Skill Moves	Jersey Number	Crossing	Finishing
count	18207.000000	18207.000000	18207.000000	18207.000000	18207.000000	18207.000000	18159.000000	18159.000000	18159.000000	18147.000000	18159.000000	18159.000000
mean	9103.000000	214298.338606	25.122206	66.238699	71.307299	1597.809908	1.113222	2.947299	2.361308	19.546096	49.734181	45.550911
std	5256.052511	29965.244204	4.669943	6.908930	6.136496	272.586016	0.394031	0.660456	0.756164	15.947765	18.364524	19.525820
min	0.000000	16.000000	16.000000	46.000000	48.000000	731.000000	1.000000	1.000000	1.000000	1.000000	5.000000	2.000000
25%	4551.500000	200315.500000	21.000000	62.000000	67.000000	1457.000000	1.000000	3.000000	2.000000	8.000000	38.000000	30.000000
50%	9103.000000	221759.000000	25.000000	66.000000	71.000000	1635.000000	1.000000	3.000000	2.000000	17.000000	54.000000	49.000000
75%	13654.500000	236529.500000	28.000000	71.000000	75.000000	1787.000000	1.000000	3.000000	3.000000	26.000000	64.000000	62.000000
max	18206.000000	246620.000000	45.000000	94.000000	95.000000	2346.000000	5.000000	5.000000	5.000000	99.000000	93.000000	95.000000

We make a backup of the df for avoiding the troubles.

In [7]:
```
#df2 = df.copy()
#df = df2.copy()
```

We drop all the columns that we don't need.

In [8]:
```
to_drop = ['Unnamed: 0', 'ID', 'Name', 'Photo', 'Flag', 'Club Logo', 'Real Face', 'Jersey Number', 'Joined', 'Contract Valid Until', 'Work Rate', 'Bod
df = df.drop(columns=to_drop)
df
```

Out[8]:

	Age	Nationality	Overall	Potential	Club	Value	Wage	Special	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Position	Loaned From	Height	Weight	LS	ST	RS	LW	LF
0	31	Argentina	94	94	FC Barcelona	€110.5M	€565K	2202	Left	5.0	4.0	4.0	RF	NaN	5'7	159lbs	88+2	88+2	88+2	92+2	93+2
1	33	Portugal	94	94	Juventus	€77M	€405K	2228	Right	5.0	4.0	5.0	ST	NaN	6'2	183lbs	91+3	91+3	91+3	89+3	90+3
2	26	Brazil	92	93	Paris Saint-Germain	€118.5M	€290K	2143	Right	5.0	5.0	5.0	LW	NaN	5'9	150lbs	84+3	84+3	84+3	89+3	89+3
3	27	Spain	91	93	Manchester United	€72M	€260K	1471	Right	4.0	3.0	1.0	GK	NaN	6'4	168lbs	NaN	NaN	NaN	NaN	NaN
4	27	Belgium	91	92	Manchester City	€102M	€355K	2281	Right	4.0	5.0	4.0	RCM	NaN	5'11	154lbs	82+3	82+3	82+3	87+3	87+3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18202	19	England	47	65	Crewe Alexandra	€60K	€1K	1307	Right	1.0	2.0	2.0	CM	NaN	5'9	134lbs	42+2	42+2	42+2	44+2	44+2
18203	19	Sweden	47	63	Trelleborgs FF	€60K	€1K	1098	Right	1.0	2.0	2.0	ST	NaN	6'3	170lbs	45+2	45+2	45+2	39+2	42+2
18204	16	England	47	67	Cambridge United	€60K	€1K	1189	Right	1.0	3.0	2.0	ST	NaN	5'8	148lbs	45+2	45+2	45+2	45+2	46+2
18205	17	England	47	66	Tranmere Rovers	€60K	€1K	1228	Right	1.0	3.0	2.0	RW	NaN	5'10	154lbs	47+2	47+2	47+2	47+2	46+2
18206	16	England	46	66	Tranmere Rovers	€60K	€1K	1321	Right	1.0	3.0	2.0	CM	NaN	5'10	176lbs	43+2	43+2	43+2	45+2	44+2

18207 rows × 77 columns

We can see that the two first columns do not provide any meaningful information so we can delete them.

df = df.iloc[:, 2:] # dataframe is no from 2 second column on (rows remain the same). df

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

Handle NaNs

Which NaNs do we have?

In [9]:
```
def check_na():
    na_counts = df.isna().sum()
    for column, na_count in zip(df.columns, na_counts):
        if na_count:
            print(column + ': ' + str(na_count))
```

In [10]:
```
check_na()
```

```
Club: 241
Preferred Foot: 48
International Reputation: 48
Weak Foot: 48
Skill Moves: 48
Position: 60
Loaned From: 16943
Height: 48
Weight: 48
LS: 2085
ST: 2085
RS: 2085
LW: 2085
LF: 2085
CF: 2085
RF: 2085
RW: 2085
LAM: 2085
CAM: 2085
RAM: 2085
LM: 2085
LCM: 2085
CM: 2085
RCM: 2085
RM: 2085
LWB: 2085
LDM: 2085
CDM: 2085
RDM: 2085
RWB: 2085
LB: 2085
LCB: 2085
CB: 2085
RCB: 2085
RB: 2085
Crossing: 48
Finishing: 48
HeadingAccuracy: 48
ShortPassing: 48
Volleys: 48
Dribbling: 48
Curve: 48
FKAccuracy: 48
LongPassing: 48
BallControl: 48
Acceleration: 48
SprintSpeed: 48
Agility: 48
Reactions: 48
Balance: 48
ShotPower: 48
Jumping: 48
Stamina: 48
Strength: 48
LongShots: 48
Aggression: 48
Interceptions: 48
Positioning: 48
Vision: 48
Penalties: 48
Composure: 48
Marking: 48
StandingTackle: 48
SlidingTackle: 48
GKDiving: 48
GKHandling: 48
GKKicking: 48
GKPositioning: 48
GKReflexes: 48
Release Clause: 1564
```

Here we can see how many NaNs a column has. In order to treat the data we have to find a solution for the NaNs. We start by analyzing each column that contains NaNs:

1. Club:

We have 241 NaNs that represent the players that play in clubs that are not registered in FIFA. We delete them.

In [11]:
```
df = df.dropna(subset = ['Club'])
```

2. Multiple columns with 48 NaNs:

We have multiple columns that have 48 NaNs, we suppose that these missing values are for the same 48 players. If that is the case, then we delete those players.

In [12]:
```
df = df.dropna(subset = ['Height'])
```

In [13]:
```
check_na()
```

```
Loaned From: 16654
LS: 1992
ST: 1992
RS: 1992
LW: 1992
LF: 1992
CF: 1992
RF: 1992
RW: 1992
LAM: 1992
CAM: 1992
RAM: 1992
LM: 1992
LCM: 1992
CM: 1992
RCM: 1992
RM: 1992
LWB: 1992
LDM: 1992
CDM: 1992
RDM: 1992
RWB: 1992
LB: 1992
LCB: 1992
CB: 1992
RCB: 1992
RB: 1992
Release Clause: 1275
```

3. Loaned From:

We delete this column due to the elevated number of NaNs in it.

In [14]:
```
df = df.drop(columns='Loaned From')
```

In [15]:
```
check_na()
```

```
LS: 1992
ST: 1992
RS: 1992
LW: 1992
LF: 1992
CF: 1992
RF: 1992
RW: 1992
LAM: 1992
CAM: 1992
RAM: 1992
LM: 1992
LCM: 1992
CM: 1992
RCM: 1992
RM: 1992
LWB: 1992
LDM: 1992
CDM: 1992
RDM: 1992
RWB: 1992
LB: 1992
LCB: 1992
CB: 1992
RCB: 1992
RB: 1992
Release Clause: 1275
```

4. Multiple columns with 1992 NaNs:

We have multiple columns that have 1992 NaNs, we suppose that these missing values are for the same 1992 players. If that is the case, we want to see what these rows have in common.

In [16]:
```
df[df['LS'].isna()]
```

Out[16]:

	Age	Nationality	Overall	Potential	Club	Value	Wage	Special	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Position	Height	Weight	LS	ST	RS	LW	LF	CF	RF
3	27	Spain	91	93	Manchester United	€72M	€260K	1471	Right	4.0	3.0	1.0	GK	6'4	168lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	25	Slovenia	90	93	Atlético Madrid	€68M	€94K	1331	Right	3.0	3.0	1.0	GK	6'2	192lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	26	Germany	89	92	FC Barcelona	€58M	€240K	1328	Right	3.0	4.0	1.0	GK	6'2	187lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
19	26	Belgium	89	90	Real Madrid	€53.5M	€240K	1311	Left	4.0	2.0	1.0	GK	6'6	212lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
22	32	Germany	89	89	FC Bayern München	€38M	€130K	1473	Right	5.0	4.0	1.0	GK	6'4	203lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18178	18	Sweden	48	65	Dalkurd FF	€50K	€1K	738	Right	1.0	2.0	1.0	GK	6'0	176lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18180	22	Scotland	48	58	St. Johnstone FC	€40K	€1K	987	Right	1.0	2.0	1.0	GK	6'1	172lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18183	44	England	48	48	Cambridge United	€0	€1K	774	Right	1.0	2.0	1.0	GK	6'2	190lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18194	18	Italy	47	65	Lecce	€50K	€1K	731	Right	1.0	3.0	1.0	GK	6'3	187lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18198	18	England	47	70	Burton Albion	€60K	€1K	792	Right	1.0	2.0	1.0	GK	5'11	154lbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1992 rows × 76 columns

We can see that all the NaNs have something in common, which is the position (all of them are goalkeepers). To be sure, if we count the number of the GK it should be 1992.

In [17]:
```
len(df.loc[df.Position == 'GK'])
```

Out[17]: 1992

Therefore we will delete these columns.

In [18]:
```
to_drop = ['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']
df = df.drop(columns=to_drop)
```

In [19]:
```
df
```

Out[19]:

	Age	Nationality	Overall	Potential	Club	Value	Wage	Special	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Position	Height	Weight	Crossing	Finishing	HeadingAccuracy
0	31	Argentina	94	94	FC Barcelona	€110.5M	€565K	2202	Left	5.0	4.0	4.0	RF	5'7	159lbs	84.0	95.0	70.0
1	33	Portugal	94	94	Juventus	€77M	€405K	2228	Right	5.0	4.0	5.0	ST	6'2	183lbs	84.0	94.0	89.0
2	26	Brazil	92	93	Paris Saint-Germain	€118.5M	€290K	2143	Right	5.0	5.0	5.0	LW	5'9	150lbs	79.0	87.0	62.0
3	27	Spain	91	93	Manchester United	€72M	€260K	1471	Right	4.0	3.0	1.0	GK	6'4	168lbs	17.0	13.0	21.0
4	27	Belgium	91	92	Manchester City	€102M	€355K	2281	Right	4.0	5.0	4.0	RCM	5'11	154lbs	93.0	82.0	55.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18202	19	England	47	65	Crewe Alexandra	€60K	€1K	1307	Right	1.0	2.0	2.0	CM	5'9	134lbs	34.0	38.0	40.0
18203	19	Sweden	47	63	Trelleborgs FF	€60K	€1K	1098	Right	1.0	2.0	2.0	ST	6'3	170lbs	23.0	52.0	52.0
18204	16	England	47	67	Cambridge United	€60K	€1K	1189	Right	1.0	3.0	2.0	ST	5'8	148lbs	25.0	40.0	46.0
18205	17	England	47	66	Tranmere Rovers	€60K	€1K	1228	Right	1.0	3.0	2.0	RW	5'10	154lbs	44.0	50.0	39.0
18206	16	England	46	66	Tranmere Rovers	€60K	€1K	1321	Right	1.0	3.0	2.0	CM	5'10	176lbs	41.0	34.0	46.0

17918 rows × 50 columns

5. Release Clause:

This feature is critical, since it could possibly lower the relation between other features and the output due to the fact that usual a player is bought with the Release Clause price. Since this feature is critical, I decided to make 2 different models and compare them to see the impact that the Release Clause had on the dataframe.

In [20]:
```
df = df.dropna(subset = ['Release Clause'])
df2 = df.drop(columns=['Release Clause'])
```

In [21]:
```
check_na()
```

Now we convert the values of **Nationality, Club, Preffered Foot** and **Position** to hot-econding style.

In [22]:
```
def column_encoder(data, col):
    col_pref = data.pop(col)
    data = pd.concat([data.reset_index(drop=True), pd.get_dummies(col_pref, prefix=col).reset_index(drop=True)], axis=1, sort=False)

    return data
```

In [23]:
```
columns_to_encode = ['Nationality', 'Club', 'Preferred Foot', 'Position'] #, 'Work Rate'
for column_to_encode in columns_to_encode:
    df = column_encoder(df, column_to_encode)
```

We turn the value of **Value, Wage, Height, Weight** and **Release Clause** into a numeric value.

In [24]:
```
def format_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 '\'' in x or 'lbs' in x:
        x = x.replace('\'' , '')
        x = x.replace('lbs' , '')
        ret_val = float(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
```

In [25]:
```
df['Value'] = df['Value'].apply(format_to_float)
df['Wage'] = df['Wage'].apply(format_to_float)
df['Height'] = df['Height'].apply(format_to_float)
df['Weight'] = df['Weight'].apply(format_to_float)
df['Release Clause'] = df['Release Clause'].apply(format_to_float)
```

Prediction

Finally we want to try to predict the **value** but we only fixed one column (**club**), so we're gonna try to fix the value only with the club information.

In [26]:
```
val = df.pop("Value")
#to_drop = [c for c in df.columns if not c.startswith('clb')]
#df = df.drop(to_drop, axis=1)
```

Now we have only the club information, in hot-econding style. At the same time 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 known its true performance.

In [27]:
```
df2 = df.drop(columns=['Release Clause'])
```

Finally we get a metric R^2 for the regression, we use the implementation from sickit-learn.

Model 1.

In this model we **keep** the column 'Release Clause'.

We train, test and get a metric R^2 for the regression.

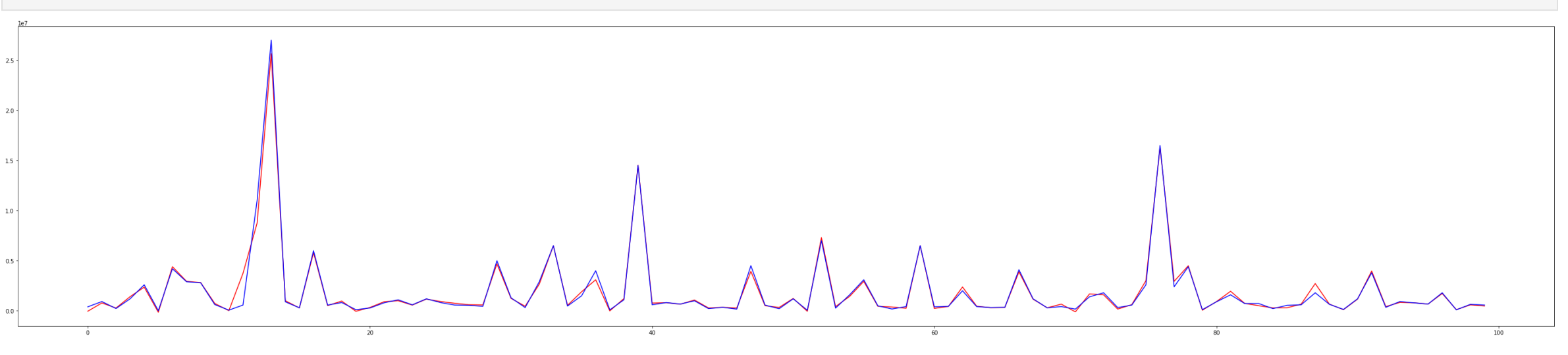
In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(df, val, test_size=0.33, random_state=0)
reg = linear_model.LinearRegression().fit(X_train, y_train)
preds = reg.predict(X_test)

print('R2 score: ' + str(r2_score(preds, y_test)))
#print('Accuracy: ' + str(reg.score(X_test,y_test)))
```

```
R2 score: 0.9937387847609508
```

A graphic representation of some of the data, comparing the prediction with the test value.

In [29]:
```
fig = plt.figure(figsize=(50,10))
fig= plt.plot(preds[0:100], 'r')
fig= plt.plot(y_test.values[0:100], 'b')
```



Model 2.

In this model we **remove** the column 'Release Clause'.

We train, test and get a metric R^2 for the regression.

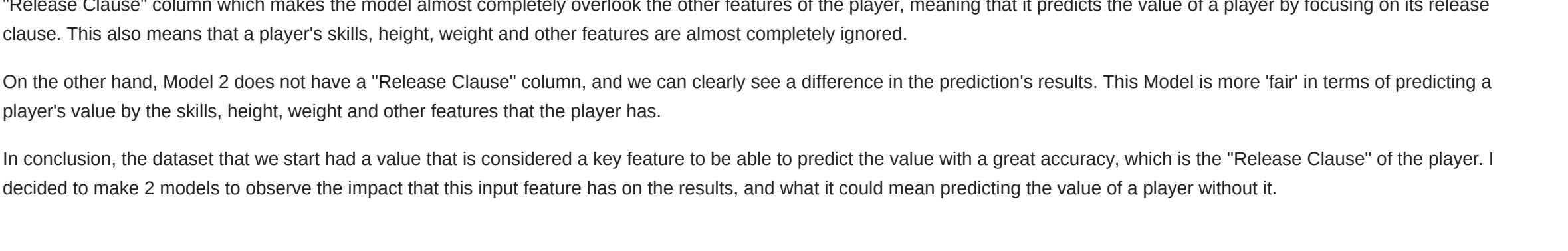
In [30]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(df2, val, test_size=0.33, random_state=45)
reg2 = linear_model.LinearRegression().fit(X_train2, y_train2)
preds2 = reg2.predict(X_test2)

print('R2 score: ' + str(r2_score(preds2, y_test2)))
```

```
R2 score: 0.8373748631704755
```

A graphic representation of some of the data, comparing the prediction with the test value.

In [31]:
```
fig2 = plt.figure(figsize=(50,10))
fig2 = plt.plot(preds2[0:100], 'r')
fig2 = plt.plot(y_test2.values[0:100], 'b')
```



Conclusions:

After Comparing the 2 models, we can see that Model 1 has a better R2 score, but that does not mean it is better. The reason Model 1 has a better R2 score is because we keep the "Release Clause" column which makes the model almost completely overlook the other features of the player, meaning that it predicts the value of a player by focusing on its release clause. This also means that a player's skills, height, weight and other features are almost completely ignored.

On the other hand, Model 2 does not have a "Release Clause" column, and we can clearly see a difference in the prediction's results. This Model is more 'fair' in terms of predicting a player's value by the skills, height, weight and other features that the player has.

In conclusion, the dataset that we start had a value that is considered a key feature to be able to predict the value with a great accuracy, which is the "Release Clause" of the player. I decided to make 2 models to observe the impact that this input feature has on the results, and what it could mean predicting the value of a player without it.