

UNIFOR  
ENSINANDO E APRENDENDO

## UNIFOR - Universidade de Fortaleza

### MBA EM CIÊNCIA DE DADOS - INTRODUÇÃO A MACHINE LEARNING

PROF. ERNESON A. OLIVEIRA

JOSÉ VALCEMIR RODRIGUES DA SILVA

Fortaleza-Ceará

## Liga NBA

O Basketball, assim como outros esportes, é um dos esportes que mais bem paga no mundo. Um estudo feito pela SportIntelligence, mostra que, na média, os jogadores mais bem pagos do mundo estão na NBA. A tabela abaixo, mostra a classificação dos times que mais bem pagam no mundo, na qual faz uma comparação com o as ligas do futebol.

Classificação		
Time	Liga	Valor médio em salários (em US\$)
Oklahoma City Thunder	NBA	9.295
Cleveland Cavaliers	NBA	8.945
Golden State Warriors	NBA	8.940
Barcelona	LaLiga	8.576
PSG	Ligue 1	8.414
Charlotte Hornets	NBA	8.379
Portland Trail Blazers	NBA	8.364
LA Clippers	NBA	8.331
Real Madrid	LaLiga	8.092
New Orleans Pelicans	NBA	7.883

Dado o dataset NBA\_players, que encontra-se no link: <https://www.kaggle.com/justinas/nba-players-data>.

jogador mais bem pago do dataset em questão, chama-se Stephen Curry, o salário dele, é justo?

Isso nós iremos descobrir com o estudo feito abaixo.

### Importando bibliotecas

```
In [1]: import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
import seaborn as sns
import plotly.express as px
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import cross_val_predict, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn import svm, linear_model, metrics
from sklearn.metrics import mean_absolute_error
from sklearn import preprocessing
import seaborn as sns

import plotly.offline as py
py.init_notebook_mode(connected=True)

simplefilter('ignore')
#le = preprocessing.LabelEncoder()
```

### Lendo os dados

```
In [2]: df = pd.read_csv('NBA_players.csv')

In [3]: lista_variaveis_dicionario = {}
file = open ('dict.txt', 'r')
lista_variaveis_dicionario = file.readlines()

In [4]: lista_variaveis_dicionario

Out[4]: ['3PT:3-Point Field Goal Percentage',
'3PT:3-Point Field Goals Made-Attempted Per Game',
'AST:Assists Per Game',
'AST:TO:Assist To Turnover Ratio',
'BLK:Blocks Per Game',
'D2:Double Double',
'DQ:Disqualifications',
'DR:Defensive Rebounds Per Game',
'FG:Field Goals Made-Attempted Per Game',
'FG:Field Goal Percentage',
'FGA:Field Goal Attempts',
'FT:Free Throws Made-Attempted Per Game',
'FTS:Free Throw Percentage',
'GP:Games Played',
'GS:Games Started',
'MIN:Minutes Per Game',
'OR:Offensive Rebounds Per Game',
'PF: Fouls Per Game',
'PTS:Points Per Game',
'RAT:Rating',
'REB:Rebounds Per Game',
'SC-EFF:Scoring Efficiency',
'SH-EFF:Shooting Efficiency',
'ST:Steals Per Game',
'ST:TO:Steal To Turnover Ratio',
'TD3:Triple Double',
'TECH:Technical Fouls',
'TO:Turnovers Per Game']
```

### Tratando os dados

```
In [5]: df.columns = df.columns.str.replace(' ', '')

In [6]: df.head()

Out[6]:
```

	TEAM	NAME	EXPERIENCE	URL	POSITION	AGE	HT	WT	COLLEGE	SALARY	...	P
0	Boston Celtics	Aron Baynes	6	<a href="http://www.espn.com/nba/player/_id/2968439">http://www.espn.com/nba/player/_id/2968439</a>	SG	31	208.28	117.65	Washington State	5,193,000	...	0.0
1	Boston Celtics	Justin Bibbe	0	<a href="http://www.espn.com/nba/player/_id/3147500">http://www.espn.com/nba/player/_id/3147500</a>	F	22	195.58	99.55	Virginia Tech	...	...	0.0
2	Boston Celtics	Jabari Bird	1	<a href="http://www.espn.com/nba/player/_id/3064308">http://www.espn.com/nba/player/_id/3064308</a>	SG	24	198.12	89.59	California	1,349,404	...	0.0
3	Boston Celtics	Jaylen Brown	2	<a href="http://www.espn.com/nba/player/_id/3917376">http://www.espn.com/nba/player/_id/3917376</a>	F	21	200.86	99.55	California	5,169,900	...	0.4
4	Boston Celtics	PJ Dozier	1	<a href="http://www.espn.com/nba/player/_id/3923250">http://www.espn.com/nba/player/_id/3923250</a>	PG	21	198.12	92.76	South Carolina	...	...	0.0

5 rows × 30 columns

### Engenharia de features (Transforma uma feature em duas)

```
In [7]: #Transforma uma variável em duas
df['$SALARY'] = df['$SALARY'].str.replace(' ', '').replace('Not signed', np.nan).astype(float)
df['$AGE'] = df['$AGE'].str.replace('-', '0')
df['$AGE'] = df['$AGE'].astype(int).replace(0, df['$AGE'].median())

df['$TMM'] = [i[0] for i in df['$TMM_THA'].str.split('-')]

lista_tha = []
for i in df['$TMM_THA'].str.split('-'):
    try:
        lista_tha.append(i[1])
    except:
        lista_tha.append(0)

df['$THA'] = lista_tha

#FTA: Average Free Throws Attempted
#FTM: Average Free Throws Made
df['$FTM'] = [i[0] for i in df['$FTM_FTA'].str.split('-')]

lista_fta = []
for i in df['$FTM_FTA'].str.split('-'):
    try:
        lista_fta.append(i[1])
    except:
        lista_fta.append(0)

df['$FTA'] = lista_fta

df['$FGM'] = [i[0] for i in df['$FGM_FGA'].str.split('-')]

lista_fga = []
for i in df['$FGM_FGA'].str.split('-'):
    try:
        lista_fga.append(i[1])
    except:
        lista_fga.append(0)

df['$FGA'] = lista_fga

In [8]: df.sort_values('$SALARY', ascending=False)

Out[8]:
```

	TEAM	NAME	EXPERIENCE	URL	POSITION	AGE	HT	WT	COLLEGE	SALAR
103	Golden State Warriors	Stephen Curry	9	<a href="http://www.espn.com/nba/player/_id/3975">http://www.espn.com/nba/player/_id/3975</a>	PG	30	190.50	85.97	Davidson	37457154.0
310	Houston Rockets	Chris Paul	13	<a href="http://www.espn.com/nba/player/_id/2779">http://www.espn.com/nba/player/_id/2779</a>	PG	33	182.88	79.19	Wake Forest	35654150.0
510	Oklahoma City Thunder	Russell Westbrook	10	<a href="http://www.espn.com/nba/player/_id/3468">http://www.espn.com/nba/player/_id/3468</a>	PG	29	190.50	90.50	UCLA	35654150.0
147	Los Angeles Lakers	LeBron James	15	<a href="http://www.espn.com/nba/player/_id/1966">http://www.espn.com/nba/player/_id/1966</a>	SF	33	203.20	113.12	-	35654150.0
306	Houston Rockets	James Harden	9	<a href="http://www.espn.com/nba/player/_id/3992">http://www.espn.com/nba/player/_id/3992</a>	PG	29	195.58	99.55	Arizona State	35650150.0
...	...	...	...	...	...	...	...	...	...	...
538	Utah Jazz	Isiah Cousins	0	<a href="http://www.espn.com/nba/player/_id/2990983">http://www.espn.com/nba/player/_id/2990983</a>	C	24	193.04	86.43	Oklahoma	NaT
540	Utah Jazz	Isaac Haas	0	<a href="http://www.espn.com/nba/player/_id/2990983">http://www.espn.com/nba/player/_id/2990983</a>	PG	22	218.44	131.22	-	NaT
540	Utah Jazz	Trey Lutes	0	<a href="http://www.espn.com/nba/player/_id/2579285">http://www.espn.com/nba/player/_id/2579285</a>	PG	25	187.96	83.71	Kentucky	NaT
541	Utah Jazz	Jarius Lyles	0	<a href="http://www.espn.com/nba/player/_id/3074787">http://www.espn.com/nba/player/_id/3074787</a>	PG	23	187.96	79.19	UMBC	NaT
543	Utah Jazz	Naz Mitrou-Long	1	<a href="http://www.espn.com/nba/player/_id/2990988">http://www.espn.com/nba/player/_id/2990988</a>	SG	25	193.04	98.64	Iowa State	NaT

550 rows × 36 columns

### Verificando as colunas que possuem nulls

```
In [9]: df.isna().sum()

Out[9]: TEAM 0
NAME 0
EXPERIENCE 0
URL 0
POSITION 0
AGE 0
HT 0
WT 0
COLLEGE 0
SALARY 110
PPG_LAST_SEASON 12
APG_LAST_SEASON 12
RPG_LAST_SEASON 12
PER_LAST_SEASON 12
PPG_CAREER 0
APG_CAREER 0
RPG_CAREER 0
GP 0
MPG 0
FGM_FGA 0
FGM 0
TMM_THA 0
THM 0
FTM_FTA 0
FTM 0
FTF 0
APG 0
BLKPG 0
STLPG 0
TOFG 0
PPG 0
TRM 0
THA 0
FTM 0
FTA 0
FGM 0
FGA 0
dtype: int64
```

### Verificando se o dataset contém outliers

Uma abordagem padrão e muitas vezes muito boa é substituir os valores faltantes por média, mediana ou moda. Para valores numéricos você deve ir com média, e se houver alguns outliers, usar a mediana (uma vez que é muito menos sensível a eles).

```
In [10]: ax = sns.boxplot(x=df['$SALARY'])
```

Com o gráfico boxplot acima, é fácil observar que os valores outliers estão entre 2,6, à 3,5, para identificar os quais são esses outliers, iremos usar a abordagem z-score

Formula z-score

$$z = \frac{x - \mu}{\sigma} = \frac{1800 - 1500}{300} = 1.$$

```
In [11]: # Z-score para detectar outlier
def detect_outlier(df):
    outliers = []
    threshold = 3
    mean_1 = np.mean(df)
    std_1 = np.std(df)

    for y in df:
        z_score = (y - mean_1) / std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)

    return outliers

In [12]: detect_outlier(df['$SALARY'])

Out[12]: [37457154.0, 35654150.0, 32088932.0, 35650150.0, 35654150.0, 35654150.0]
```

```
In [13]: df[df['$SALARY'].isin(detect_outlier(df['$SALARY']))]
```

```
Out[13]:
```

	TEAM	NAME	EXPERIENCE	URL	POSITION	AGE	HT	WT	COLLEGE	SALARY
103	Golden State Warriors	Stephen Curry	9	<a href="http://www.espn.com/nba/player/_id/3975">http://www.espn.com/nba/player/_id/3975</a>	PG	30	190.50	85.97	Davidson	37457154.0
147	Los Angeles Lakers	LeBron James	15	<a href="http://www.espn.com/nba/player/_id/1966">http://www.espn.com/nba/player/_id/1966</a>	SF	33	203.20	113.12	-	35654150.0
230	Detroit Pistons	Blake Griffin	8	<a href="http://www.espn.com/nba/player/_id/3989">http://www.espn.com/nba/player/_id/3989</a>	PF	29	208.28	113.12	Oklahoma	32088932.0
306	Houston Rockets	James Harden	9	<a href="http://www.espn.com/nba/player/_id/3992">http://www.espn.com/nba/player/_id/3992</a>	PG	29	195.58	99.55	Arizona State	35650150.0
310	Houston Rockets	Chris Paul	13	<a href="http://www.espn.com/nba/player/_id/2779">http://www.espn.com/nba/player/_id/2779</a>	PG	33	182.88	79.19	Wake Forest	35654150.0
510	Oklahoma City Thunder	Russell Westbrook	10	<a href="http://www.espn.com/nba/player/_id/3468">http://www.espn.com/nba/player/_id/3468</a>	PG	29	190.50	90.50	UCLA	35654150.0

6 rows × 36 columns

Identificamos que a coluna SALARY possui outliers e portanto, faz mais sentido utilizar a mediana para a substituição dos NaNs, pois é menos sensível aos outliers.

```
In [16]: ax = sns.boxplot(x="variable", y="value", data=pd.melt(df, ['PPG_LAST_SEASON', 'APG_LAST_SEASON', 'RPG_LAST_SEASON', 'PER_LAST_SEASON']))
ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
```

```
Out[16]:
```

Como podemos perceber com o gráfico acima, as colunas possuem outliers sim e portanto, iremos substituir pela mediana, como mostra no código abaixo.

```
In [17]: imputer = SimpleImputer(missing_values=np.nan, strategy='median')
df[['PPG_LAST_SEASON', 'APG_LAST_SEASON', 'RPG_LAST_SEASON', 'PER_LAST_SEASON']] = (
    imputer.fit_transform(df[['PPG_LAST_SEASON', 'APG_LAST_SEASON', 'RPG_LAST_SEASON', 'PER_LAST_SEASON']])
)
df['$AGE']
```

```
Out[17]:
```

0	31
1	22
2	24
3	21
4	21
...	...
545	25
546	25
547	27
548	34
549	31

Name: AGE, Length: 550, dtype: int32

### Verifica se o dataset ainda possui algum outlier

```
In [18]: df.isna().sum()

Out[18]: TEAM 0
NAME 0
EXPERIENCE 0
URL 0
POSITION 0
AGE 0
HT 0
WT 0
COLLEGE 0
SALARY 0
PPG_LAST_SEASON 0
APG_LAST_SEASON 0
RPG_LAST_SEASON 0
PER_LAST_SEASON 0
PPG_CAREER 0
APG_CAREER 0
RPG_CAREER 0
GP 0
MPG 0
FGM_FGA 0
FGM 0
TMM_THA 0
THM 0
FTM_FTA 0
FTM 0
FTF 0
APG 0
BLKPG 0
STLPG 0
TOFG 0
PPG 0
TRM 0
THA 0
FTM 0
FTA 0
FGM 0
FGA 0
dtype: int64
```

Pronto, as colunas SALARY, PPG\_LAST\_SEASON, APG\_LAST\_SEASON, RPG\_LAST\_SEASON, e PER\_LAST\_SEASON que possuíam NaNs, foram substituídas pelas suas respectivas medianas, para assim, não serem descartadas do DATASET.

```
In [19]: df['$AGE'].min()

Out[19]: 18
```

### Verificando o percentual de jogadores experientes na temporada.

```
In [20]: fig, ax = plt.subplots(figsize=(10, 10))

experiencia_grupos = df['EXPERIENCE'].value_counts(bins=[0, 3, 5, 8, 10, 15, 25]).values
labels = ('0-3 Anos', '5-8 Anos', '3-5 Anos', '8-10 Anos', '10-15 Anos', 'Maior que 15 Anos')
explodes = (0, 0, 0, 0, 0, 0)
cmpap = plt.get_cmapp("tab20c")
cores = cmap(np.array([1, 3, 10, 13, 15, 18]))

ax = plt.pie(experiencia_grupos, labels=labels, autopct='%1.0f%%', colors=cores, fontsize=10,
    pctdistance=1.5, explode=explode, labeldistance=1.05)
plt.title('Grupo de experiência dos jogadores da NBA na temporada 18-19', fontsize=12);
```

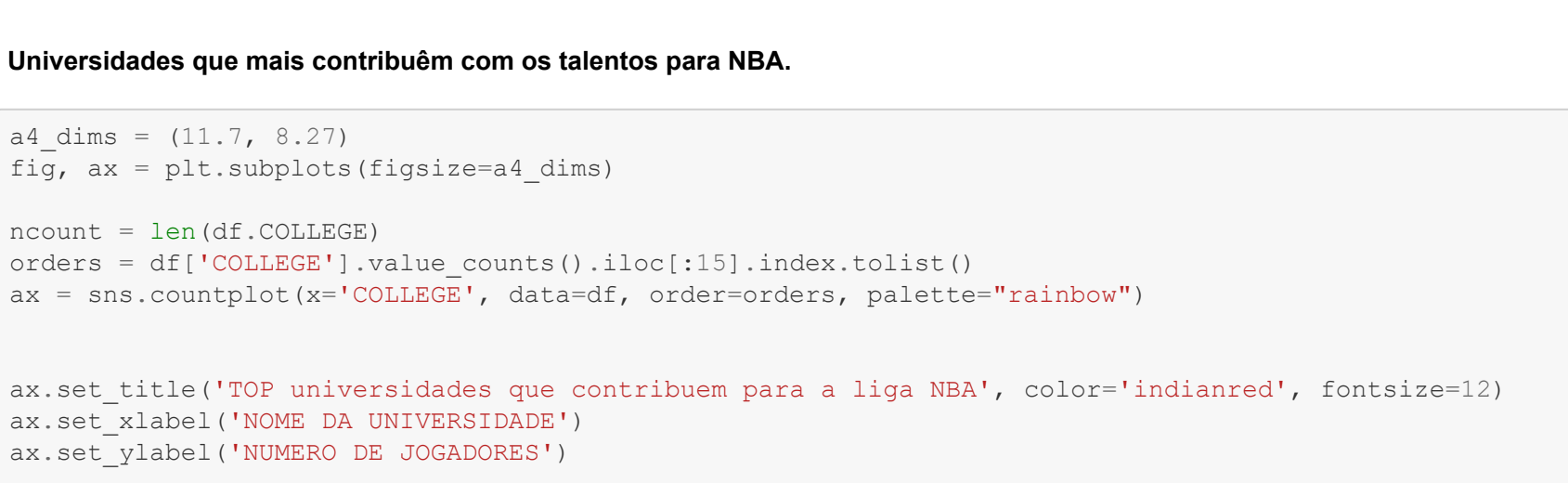


### Universidades que mais contribuíram com os talentos para NBA.

```
In [21]: a4_dims = (11.7, 8.27)
fig, ax = plt.subplots(figsize=a4_dims)

norder = len(df.COLLEGE)
counts = df['COLLEGE'].value_counts().iloc[1:15].index.tolist()
ax.set_title('TOP universidades que contribuem para a liga NBA', color='indianred', fontsize=12)
ax.set_xlabel('NOME DA UNIVERSIDADE')
ax.set_ylabel('NÚMERO DE JOGADORES')
```

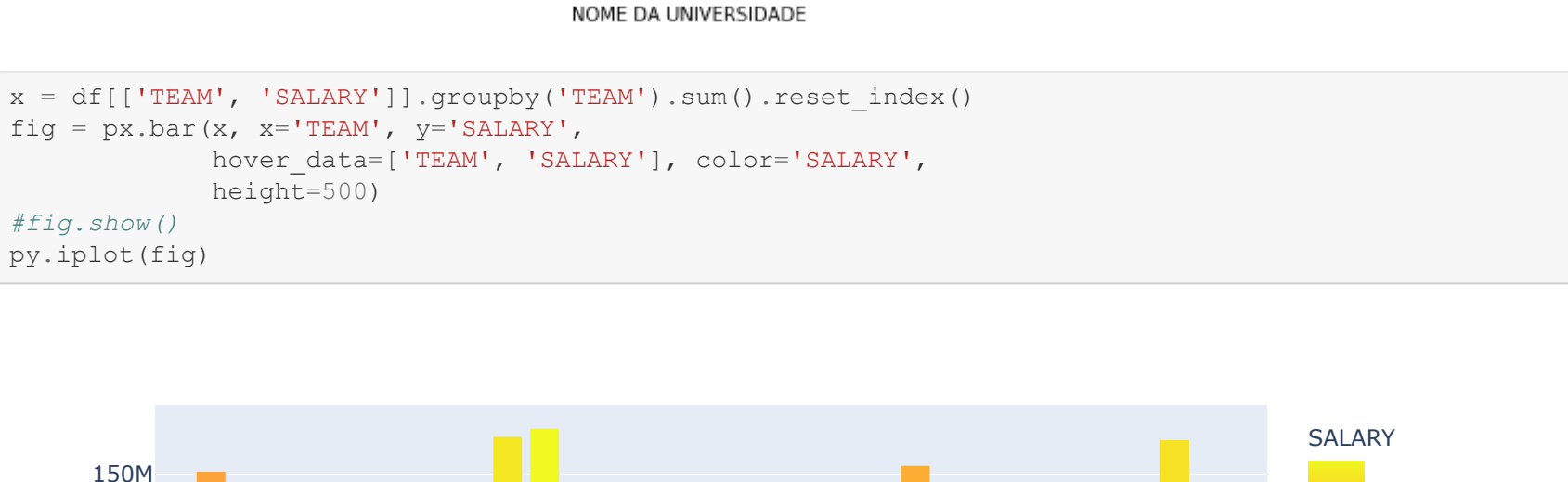
```
for p in ax.patches:
    x, y = p.get_bbox().get_points()[1:, 0]
    y = p.get_bbox().get_points()[1:, 1]
    ax.annotate('%i.1f' % format(100 * y / counts), (x.mean(), y),
        ha='center', va='bottom')
fig.autofmt_xdate()
```



```
In [22]: x = df[['TEAM', '$SALARY']].groupby('TEAM').sum().reset_index()
fig = px.bar(x, x='TEAM', y='$SALARY',
    hover_data=['TEAM', '$SALARY'], color='$SALARY',
    height=500)
fig.show()
py.iplot(fig)
```



```
In [23]: x = df[['POSITION', '$SALARY']].groupby('POSITION').sum().reset_index()
fig = px.bar(x, x='POSITION', y='$SALARY',
    hover_data=['POSITION', '$SALARY'], color='$SALARY',
    labels={'pos': 'position of Canada'}, height=500)
py.iplot(fig)
fig.show()
```



Com os gráficos acima, fica claro que os jogadores mais bem pagos tem as seguintes características, joga em um dos dois times mais caros da liga e joga na posição PG

### Categorizando variáveis nominais.

```
In [24]: le = preprocessing.LabelEncoder()
df['POS_CATEGORICAL'] = le.fit(df['POSITION']).transform(df['POSITION'])
df['TEAM_CATEGORICAL'] = le.fit(df['TEAM']).transform(df['TEAM'])
df['COLLEGE_CATEGORICAL'] = le.fit(df['COLLEGE']).transform(df['COLLEGE'])
```

```
In [25]: x_colunas = ['TEAM', 'NAME', 'URL',
    'POSITION', 'COLLEGE',
    'FGM_FGA', 'TMM_THA', 'FTM_FTA', '$SALARY']
y_colunas = ['PPG_LAST_SEASON', 'APG_LAST_SEASON', 'RPG_LAST_SEASON', 'PER_LAST_SEASON']
x_train = df[df_colunas.difference(x_colunas)].astype(float)
df_y_train = df[y_colunas].astype(int)
```

```
In [26]: df_x_train

Out[26]:
```

	AGE	APG	APG_CAREER	APG_LAST_SEASON	BLKPG	COLLEGE_CATEGORICAL	EXPERIENCE	FGA	FGM	FGP	...	PPG_LAST
0	31.0	0.7	0.7	1.1	0.5	124.0	6.0	4.3	2.2	0.502	...	...
1	22.0	0.0	0.0	0.0	0.0	121.0	0.0	0.0	0.0	0.000	...	...
2	24.0	0.6	0.6	0.6	0.1	16.0	1.0	2.0	1.2	0.577	...	...
3	21.0	1.2	1.2	1.6	0.3	16.0	2.0	8.3	3.8	0.461	...	...
4	21.0	0.0	0.0	0.0	0.0	98.0	1.0	1.0	0.5	0.500	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...
545	25.0	0.3	0.3	0.3	0.0	47.0	2.0	1.5	0.4	0.277	...	...
546	25.0	1.4	1.4	1.4	0.2	6.0	1.0	4.1	1.7	0.423	...	...
547	27.0	7.9	7.9	5.3	0.1	0.0	7.0	9.0	3.4	0.365	...	...
548	34.0	1.5	1.5	0.9	0.4	0.0	12.0	5.2	2.3	0.449	...	...
549	31.0	0.7	3.0	2.4	1.2	6.0	6.0	3.2	1.4	0.437	...	...

550 rows × 30 columns

### Plots a matriz de correlação

```
In [27]: a4_dims = (11.7, 8.27)
fig, ax = plt.subplots(figsize=a4_dims)
corr = df_x_train.corr()
sns.heatmap(corr, cmap = 'Blues',
    xticklabels=corr.columns,
    yticklabels=corr.columns)
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x2d80b340248>
```

Como não sabemos quais variáveis iremos inputar no modelo, utilizando o método Feature Importance (Importância da variável), ela nos traz insights de variáveis que são supostamente irrelevantes para o modelo. Obs: Não é porque esse método mostre que a variável é irrelevante, importante para o modelo, não significa dizer que isso seja verdade.

O exemplo que irei mostrar aqui é o SelectKBest do modulo feature\_selection do sklearn, que utiliza testes estatísticos univariados. Irei considerar apenas as variáveis onde sua importância seja maior que 10% para o modelo.

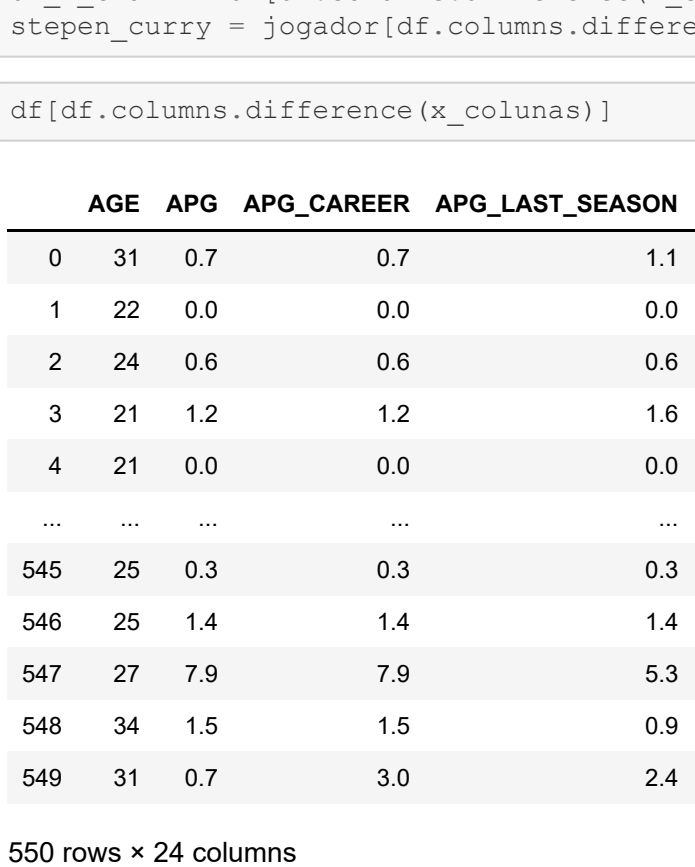


```
[28]: # Seleção de feat
def select_features(X_train, y_train, X_test):
    fs = SelectKBest(score_func=mutual_info_regression, k='all')
    fs.fit(X_train, y_train)
    X_train_fs = fs.transform(X_train)
    X_test_fs = fs.transform(X_test)
    return X_train_fs, X_test_fs, fs

X_train, X_test, y_train, y_test = train_test_split(df_x_train, df_y_train, test_size=0.33, random_state=8)
X_train_fs, X_test_fs, fs = select_features(X_train, y_train, X_test)

# Score para cada feature
for i in range(len(fs.scores_)):
    if (fs.scores_[i] * 100) < 10:
        print(f'Feature {df_x_train.columns[i]}, fs.scores_{i}')
#plot scores
plt.bar([i for i in range(len(fs.scores_))], fs.scores_)
plt.show()
```

Feature 'COLLEGE\_CATEGORICAL', 0.037480181964977711  
Feature 'HT', 0.0710423259804661  
Feature 'POS\_CATEGORICAL', 0.03403164902002853  
Feature 'TEAM\_CATEGORICAL', 0.0  
Feature 'THM', 0.06225311248932819  
Feature 'WT', 0.0690477820801966



Como podemos ver acima, as variáveis COLLEGE\_CATEGORICAL, HT, POS\_CATEGORICAL, TEAM\_CATEGORICAL, THM e WT são as menos importantes para o modelo e portanto, ficaram de fora do modelo, isso porque definimos um limiar de 10%.

```
In [29]: x_colunas = ['TEAM', 'NAME', 'URL',
               'POSITION', 'COLLEGE',
               'FGM_FGA', 'FTM_FTA', 'SALARY',
               'TEAM_CATEGORICAL', 'COLLEGE_CATEGORICAL',
               'HT', 'POS_CATEGORICAL', 'THM', 'WT']
jogador = df.sort_values(by='SALARY', ascending=False).head(1)
df_x_train = df[df.columns.difference(x_colunas)].astype(float)
stepen_curry = jogador[df.columns.difference(x_colunas)].astype(float)
```

```
In [30]: df[df.columns.difference(x_colunas)]
```

```
Out[30]:
```

	AGE	APG	APG_CAREER	APG_LAST_SEASON	BLKPG	EXPERIENCE	FGA	FGM	FGP	FTA	...	PER_LAST_SEASON	PPG	PP
0	31	0.7	0.7	0.7	1.1	0.5	6	4.3	2.2	0.502	1.3	...	12.09	5.4
1	22	0.0	0.0	0.0	0.0	0.0	0	0	0	0.000	0	...	0.00	0.0
2	24	0.6	0.6	0.6	0.6	0.1	1	2.0	1.2	0.577	1.0	...	12.18	3.0
3	21	1.2	1.2	1.2	1.6	0.3	2	8.3	3.8	0.461	2.4	...	13.69	10.4
4	21	0.0	0.0	0.0	0.0	0.0	1	1.0	0.5	0.500	0.0	...	-4.82	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
545	25	0.3	0.3	0.3	0.3	0.0	2	1.5	0.4	0.277	0.1	...	9.22	0.9
546	25	1.4	1.4	1.4	1.4	0.2	1	4.1	1.7	0.423	1.1	...	10.84	5.0
547	27	7.9	7.9	7.9	5.3	0.1	7	9.0	3.4	0.385	3.8	...	15.43	10.8
548	34	1.5	1.5	0.9	0.9	0.4	12	5.2	2.3	0.449	1.1	...	15.86	6.1
549	31	0.7	3.0	3.0	2.4	1.2	6	3.2	1.4	0.437	1.2	...	15.19	3.7

550 rows x 24 columns

```
In [31]: stepen_curry
```

```
Out[31]:
```

	AGE	APG	APG_CAREER	APG_LAST_SEASON	BLKPG	EXPERIENCE	FGA	FGM	FGP	FTA	...	PER_LAST_SEASON	PPG	PP
103	30.0	6.8	6.8	6.8	6.1	0.2	9.0	16.8	8.0	0.477	4.0	...	26.32	23.1

1 rows x 24 columns

**Treinamento e teste do modelo.**

Com o estudo dos dados até aqui, foi possível identificar que os dados do dataset em questão, são bastante dispersos e portanto, precisará ser feito uma normalização.

Sobre os algoritmos de normalização, citarei alguns aqui, são eles:

1. StandardScaler: A ideia por trás StandardScaleré que ele irá transformar seus dados de forma que sua distribuição tenha um valor médio 0 e desvio padrão de 1.
2. MinMaxScaler: O MinMaxScaler transforma recursos dimensionando cada recurso para uma determinada gama. Esse intervalo pode ser definido especificando o parâmetro feature\_range (padrão em (0,1)). Este scaler funciona melhor para casos em que a distribuição não é gaussiana ou o desvio padrão é muito pequeno. No entanto, é sensível a outliers, por isso, se há outliers nos dados, você pode querer considerar outro scaler.
3. Ao contrário dos scalers anteriores, as estatísticas de centralidade e escalação do RobustScaler são baseadas em percentis e, portanto, não são influenciadas por um número de grandes outliers marginais

Para este nosso exemplo, iremos utilizar o RobustScaler, visto que nosso dataset tem bastante outliers

Formula RobustScaler

$$X_{scale} = \frac{x_i - x_{med}}{x_{75} - x_{25}}$$

```
In [32]: #MinMaxScaler
#StandardScaler
#RobustScaler
scalerx = RobustScaler().fit(df_x_train)
scalery = RobustScaler().fit(df_y_train)

X_scaler = scalerx.fit_transform(df_x_train)
Y_scaler = scalery.fit_transform(df_y_train)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X_scaler, Y_scaler, test_size=0.33, random_state=0)
#X_train, X_test, fs = select_features(X_train, Y_train, X_test)
X = X_train
Y = Y_train
```

```
LR = LinearRegression().fit(X, Y)
LRsvm = svm.SVR(kernel='rbf', degree=3).fit(X, Y)
LRSGDRegressor = linear_model.SGDRegressor(loss='squared_loss', penalty='l2', max_iter=1000).fit(X, Y)
```

Estou utilizando 3 modelos, LinerRegression, SVM (KERNEL='rbf') e SGDRegressor, para fazer uma comparação entre eles.

```
In [33]: def resultado_predicao_modelo(modelo):
         dict_modelo = {}
```

```
         y_pred = modelo.predict(scalerx.fit_transform(X_test))
         df_resultado = pd.DataFrame()
         df_resultado['Atual'] = np.squeeze(scalery.inverse_transform(Y_test))
         df_resultado['Predito'] = np.round(scalery.inverse_transform(y_pred).reshape(1, -1))[0], 1)
         return df_resultado.astype(float)
```

```
In [34]: #Mean Absolute Error
#Mean Squared Error
#Root Mean Squared Error
```

```
def plot_resultado_modelo(modelo):
    dict_modelo = {}
```

```
         y_pred = modelo.predict(scalerx.fit_transform(X_test))
         dict_modelo['MODELO'] = str(modelo)[0:-2]
         dict_modelo['SCORE'] = round(modelo.score(X, Y), 2)
         dict_modelo['INTERCEPT'] = round(modelo.intercept_[0], 2)
         dict_modelo['MAE'] = round(metrics.mean_absolute_error(Y_test, y_pred), 2)
         dict_modelo['MSE'] = round(metrics.mean_squared_error(Y_test, y_pred), 2)
         dict_modelo['RMSE'] = round(np.sqrt(metrics.mean_squared_error(Y_test, y_pred)), 2)
```

```
         df_modelo = pd.DataFrame(dict_modelo)
```

```
         ax1 = sns.distplot(Y_test, hist=False, color='r', label='Valor atual')
         plot = sns.distplot(y_pred, hist=False, color='b', label='Valores preditos', ax=ax1)
```

```
         resultado_pred = resultado_predicao_modelo(modelo)
         Y_atual = resultado_pred['Atual']
         Y_pred = resultado_pred['Predito']
```

```
         fig, ax = plt.subplots()
         ax.scatter(Y_atual, Y_pred)
         ax.plot([Y_atual.min(), Y_atual.max()], [Y_atual.min(), Y_atual.max()], 'k--', lw=4)
         ax.set_xlabel('Atual')
         ax.set_ylabel('Predito')
```


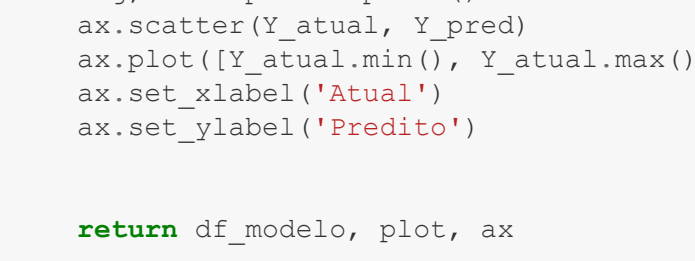
```
         return df_modelo, plot, ax
```

**Modelo 1: LinearRegression**

```
In [35]: plot_resultado_modelo(LR)[0]
```

```
Out[35]:
```

	MODELO	SCORE	INTERCEPT	MAE	MSE	RMSE
0	LinearRegression	0.6	-0.22	0.58	0.66	0.81

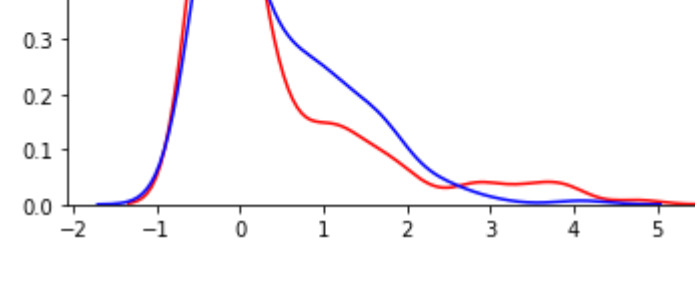


**Modelo 2: SVM Regression (Kernel='rbf')**

```
In [36]: plot_resultado_modelo(LRSvm)[0]
```

```
Out[36]:
```

	MODELO	SCORE	INTERCEPT	MAE	MSE	RMSE
0	SVR	0.73	0.74	0.44	0.46	0.68

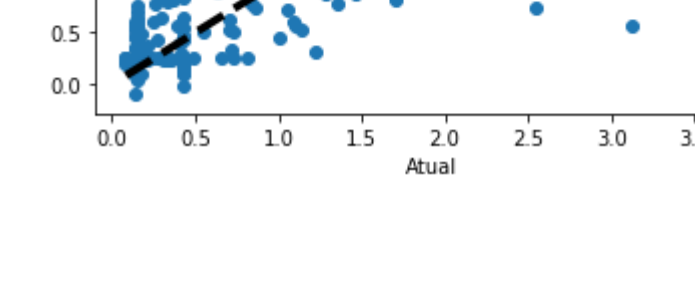


**Modelo 3: SGDRegressor**

```
In [37]: plot_resultado_modelo(LRSGDRegressor)[0]
```

```
Out[37]:
```

	MODELO	SCORE	INTERCEPT	MAE	MSE	RMSE
0	SGDRegressor	0.55	-0.07	0.58	0.64	0.8



Podemos observar que, dentre os 3 modelos, o melhor que se adequou aos dados foi o SVM (Suport Vector Machine) perdendo apenas na 'Intercepção' para o modelo LinearRegression e SGDRegressor. Abaixo, os resultados transcritos:

1. Score: 0.73
2. Erro Médio Absoluto (MAE): 0.44
3. Erro Médio Quadrado (MSE): 0.46
4. Raiz Quadrada do Erro Médio (RMSE): 0.68

```
In [38]: x = scalerx.fit_transform(np.reshape(stepen_curry, (1, -1)))
```

```
         result_lr_modelo = round(scalery.inverse_transform(LR.predict(x_))[0][0], 2)
         result_lr_svm = round(scalery.inverse_transform(LRSvm.predict(x_).reshape(1, -1))[0][0], 2)
         result_lr_sgdregressor = round(scalery.inverse_transform(LRSGDRegressor.predict(x_).reshape(1, -1))[0][0], 2)
```

```
         salario_stepen = jogador['SALARY'].values[0]
```

```
         print(f'!Testando o modelo LinearRegression com dados de teste e predizendo: (result_lr_test_modelo)')
         print(f'!Testando o modelo SVM com dados de teste e predizendo: (result_lr_svm_test)')
         print(f'!Testando o modelo SGDREGRESSOR com dados de teste e predizendo: (result_lr_sgdregressor_test)')
```

```
         print(f'!Salário predito no modelo LinearRegression : { result_lr_modelo } X Salario Stepem Curry: (salario_stepen)')
         print(f'!Salário predito no modelo SVM (rbf): (result_lr_modelo) X Salario Stepem Curry: (salario_stepen)')
         print(f'!Salário predito no modelo SGDRegressor: (result_lr_sgdregressor) X Salario Stepem Curry: (salario_stepen)')
```

```
         Testando o modelo LinearRegression com dados de teste e predizendo: 619321.35
         Testando o modelo SVM com dados de teste e predizendo: 280596.02
         Testando o modelo SGDREGRESSOR com dados de teste e predizendo: 1717603.98
```

```
         Salário predito no modelo LinearRegression : 2824281.49 X Salario Stepem Curry: 37457154.0
         Salário predito no modelo SVM (rbf): 4935036.74 X Salario Stepem Curry: 37457154.0
         Salário predito no modelo SGDRegressor: 3882170.0 X Salario Stepem Curry: 37457154.0
```

Interessante notar que, o modelo LinearRegression e SVM poderiam fazer o Stepem tem um salário de R\$2824281.49 e enquanto que o modelo SGDRegressor, conseguiu aproximar melhor do salário real do jogador.

Pelo fato do modelo ter bastante outliers iremos agora fazer um teste retirando os outliers do dataset e retrainar o modelo. E para isso, iremos utilizar o Z-score do modulo scipy. O código está descrito em algumas páginas acima.

```
In [39]: from scipy import stats
```

```
         x_colunas = ['TEAM', 'NAME', 'URL', 'POSITION', 'COLLEGE',
                     'FGM_FGA', 'FTM_FTA', 'FTM_FTA', 'SALARY',
                     'TEAM_CATEGORICAL', 'COLLEGE_CATEGORICAL',
                     'HT', 'POS_CATEGORICAL', 'THM', 'WT']
```

```
         y_colunas = ['SALARY']
```

```
         x_df = df[df.columns.difference(x_colunas)].astype(float)
         z = np.abs(stats.zscore(x_df))
```

```
In [40]: df[df.columns.difference(x_colunas)].astype(float)
```

```
Out[40]:
```

	AGE	APG	APG_CAREER	APG_LAST_SEASON	BLKPG	EXPERIENCE	FGA	FGM	FGP	FTA	...	PPG	PPG_CAREER	PPG_LAS
0	31.0	0.7	0.7	0.7	1.1	0.5	6.0	4.3	2.2	0.502	1.3	...	5.4	5.4
1	22.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000	0.0	...	0.0	0.0
2	24.0	0.6	0.6	0.6	0.6	0.1	1.0	2.0	1.2	0.577	1.0	...	12.18	3.0
3	21.0	1.2	1.2	1.2	1.6	0.3	2.0	8.3	3.8	0.461	2.4	...	10.4	10.4
4	21.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.5	0.500	0.0	...	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
545	25.0	0.3	0.3	0.3	0.3	0.0	2.0	1.5	0.4	0.277	0.1	...	0.9	0.9
546	25.0	1.4	1.4	1.4	1.4	0.2	1.0	4.1	1.7	0.423	1.1	...	5.0	5.0
547	27.0	7.9	7.9	7.9	5.3	0.1	7.0	9.0	3.4	0.385	3.8	...	10.8	10.8
548	34.0	1.5	1.5	0.9	0.9	0.4	12.0	5.2	2.3	0.449	1.1	...	6.1	6.1
549	31.0	0.7	3.0	3.0	2.4	1.2	6.0	3.2	1.4	0.437	1.2	...	3.7	3.7

550 rows x 25 columns

Filtra todas as linhas que ultrapassam o limiar de 3 desvios padrão dos dados. Todas as linhas que ultrapassarem esse limiar, serão retiradas do 'dataset'.

```
In [41]: x_colunas.append('SALARY')
         new_df = x_df[(z < 3) && !all(axis=1)]
         df_x_train = new_df[new_df.columns.difference(x_colunas)].astype(float)
         df_y_train = new_df[y_colunas].astype(int)
```

```
In [42]: df_x_train
```

```
Out[42]:
```

	AGE	APG	APG_CAREER	APG_LAST_SEASON	BLKPG	EXPERIENCE	FGA	FGM	FGP	FTA	...	PER_LAST_SEASON	PPG	PP
0	31	0.7	0.7	0.7	1.1	0.5	6	4.3	2.2	0.502	1.3	...	12.09	5.4
1	22	0.0	0.0	0.0	0.0	0.0	0	0	0	0.000	0	...	0.00	0.0
2	24	0.6	0.6	0.6	0.6	0.1	1	2.0	1.2	0.577	1.0	...	12.18	3.0
3	21	1.2	1.2	1.2	1.6	0.3	2	8.3	3.8	0.461	2.4	...	13.69	10.4
4	21	0.0	0.0	0.0	0.0	0.0	1	1.0	0.5	0.500	0.0	...	-4.82	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
544	26	1.7	1.7	1.7	1.8	0.1	3	4.1	1.8	0.439	0.7	...	11.91	4.7
545	25	0.3	0.3	0.3	0.3	0.0	2	1.5	0.4	0.277	0.1	...	9.22	0.9
546	25	1.4	1.4	1.4	1.4	0.2	1	4.1	1.7	0.423	1.1	...	10.84	5.0
548	34	1.5	1.5	0.9	0.9	0.4	12	5.2	2.3	0.449	1.1	...	15.86	6.1
549	31	0.7	3.0	3.0	2.4	1.2	6	3.2	1.4	0.437	1.2	...	15.19	3.7

499 rows x 24 columns

```
In [43]: print(f'!Tamanho do Dataset antes {x_df.shape[0]} e depois {new_df.shape[0]} da retirada dos outliers')
         Tamanho do Dataset antes 550 e depois 499 da retirada dos outliers
```

No exemplo anterior, utilizando o normalizador RobustoScaler, neste exemplo, iremos utilizar o MinMaxScaler, visto que nossos dados não possuem mais outliers.

```
In [44]: scalerx = MinMaxScaler().fit(df_x_train)
         scalery = MinMaxScaler().fit(df_y_train)
```

```
         X_scaler = scalerx.fit_transform(df_x_train)
         Y_scaler = scalery.fit_transform(df_y_train)
```

```
         X_train, X_test, Y_train, Y_test = train_test_split(X_scaler, Y_scaler, test_size=0.33, random_state=10)
         #X_train, X_test, fs = select_features(X_train, Y_train, X_test)
         X = X_train
         Y = Y_train
```

```
         LR = LinearRegression().fit(X, Y)
         LRsvm = svm.SVR(kernel='rbf', degree=3).fit(X, Y)
         LRSGDRegressor = linear_model.SGDRegressor(loss='squared_loss', penalty='l2', max_iter=100).fit(X, Y)
```

```
In [45]: plot_resultado_modelo(LR)[0]
```

```
Out[45]:
```

	MODELO	SCORE	INTERCEPT	MAE	MSE	RMSE
0	SVR	0.76	0.18	0.11	0.02	0.15

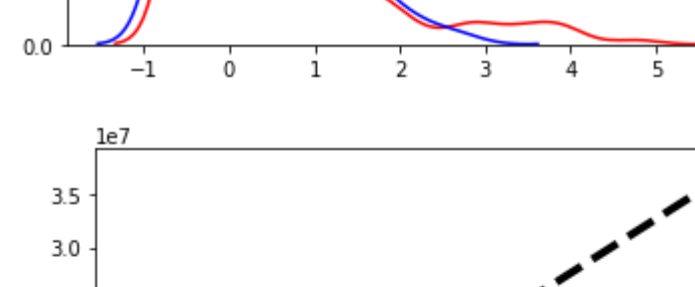


O modelo SVM acima, comparado ao modelo anterior (Modelo 2: SVM Regression (Kernel='rbf')), ele teve uma melhora no score, saindo de 0.73, para 0.76 e uma melhora também no Erro médio Absoluto e Erro quadrático médio.

```
In [46]: plot_resultado_modelo(LRSvm)[0]
```

```
Out[46]:
```

	MODELO	SCORE	INTERCEPT	MAE	MSE	RMSE
0	SVR	0.76	0.18	0.11	0.02	0.15



O modelo SVM acima, comparado ao modelo anterior (Modelo 2: SVM Regression (Kernel='rbf')), ele teve uma melhora no score, saindo de 0.73, para 0.76 e uma melhora também no Erro médio Absoluto e Erro quadrático médio.

```
In [47]: plot_resultado_modelo(LRSGDRegressor)[0]
```

```
Out[47]:
```

	MODELO	SCORE	INTERCEPT	MAE	MSE	RMSE
0	SGDRegressor(max_iter=10)	0.36	0.04	0.14	0.03	0.18



Já o modelo SGDRegressor, com os mesmos parâmetros do modelo anterior, piorou um pouco o score, caindo de 0.55, para 0.36, porém, houve uma melhora de 0.88 para 0.14 no Erro médio Absoluto e 0.64 para 0.03, no Erro médio quadrático.

Retirando os outliers dos dados, os 3 modelos tiveram melhoras consideráveis, apenas o modelo 3 SGDRegressor que continua com um score baixo, mas as outras medidas (INTERCEPT, MAE, MSE, RMSE), melhoraram.

Iremos agora, fazer uma validação cruzada com os 3 modelos utilizando K-Fold.

```
In [48]: # Faz validação cruzada
         def gera_score(modelo, X_train, y_train, X_test, y_test):
             score_train = modelo.score(X_train, y_train)
             score_test = modelo.score(X_test, y_test)
```

```
             y_pred = modelo.predict(X_test)
             mae = mean_absolute_error(y_test, y_pred)
             #print(f'Modelo: {score_train} => Score treinamento:{round(score_train * 100, 2)}%, Score teste :{round(score_test * 100, 2)}%')
             return modelo, \
                     round(score_train * 100, 2), \
```