UNIVERSIDADE DE FORTALEZA

MBA em Ciência de Dados

Introdução ao Aprendizado de Máquina - Professor Erneson Oliveira.

Nome: Thiago Barbosa Vieira
E-mail: thiagovieirahc@gmail.com
Matrícula: 1925332

1. Introdução

Este trabalho tem como objetivo desenvolver um sistema de aprendizagem de máquina capaz de fazer previsões de salários dos jogadores da NBA, a partir de um conjunto de dados disponibilizado. Bem como, após todas as análises e predições, verificar se o valor do salário real do jogador Stephen Curry é justo.

2. Metodologia

Primeiramente, será realizada uma higienização dos dados no DataFrame para remover valores que não possuem relevância para a análise. Após isso, serão realizadas análises de algumas características como média de minutos na quadra, pontos por jogo na ultima temporada, eficiência, número de rotatividade por jogo, etc. Em seguida, será feita uma correlação para análisar as melhores características. Por último, serão criados alguns modelos como Regressão Linear, Floresta aleatória, Árvore de Decisão, para que seja verificada a melhor predição.

3. Obtenção dos Dados para Análise

In [718]:

```
# Importação das bibliotecas Pandas e Pyplot
import pandas as pd
import matplotlib.pyplot as plt
```

In [719]:

```
#Leitura e criação do DataFrame que será utilizado ao longo deste trabalho.
df = pd.read_csv('NBA_Players.csv',sep=',',encoding='utf-8', skipinitialspace=True)
#retirar espaço em branco no inicio do nome das colunas.
df.columns = [c.strip().upper().replace(' ', '') for c in df.columns]
```

In [720]:

```
df.head(10)
```

Out[720]:

	TEAM	NAME	EXPERIENCE	URL	POSITION	AGE
0	Boston Celtics	Aron Baynes	6	http://www.espn.com/nba/player/_/id/2968439	SF	31
1	Boston Celtics	Justin Bibbs	0	http://www.espn.com/nba/player/_/id/3147500	G	22
2	Boston Celtics	Jabari Bird	1	http://www.espn.com/nba/player/_/id/3064308	SG	24
3	Boston Celtics	Jaylen Brown	2	http://www.espn.com/nba/player/_/id/3917376	F	21
4	Boston Celtics	PJ Dozier	1	http://www.espn.com/nba/player/_/id/3923250	PG	21
5	Boston Celtics	Marcus Georges-Hunt	2	http://www.espn.com/nba/player/_/id/2982261	SG	24
6	Boston Celtics	Gordon Hayward	8	http://www.espn.com/nba/player/_/id/4249	SF	28
7	Boston Celtics	Al Horford	11	http://www.espn.com/nba/player/_/id/3213	PF	32
8	Boston Celtics	Kyrie Irving	7	http://www.espn.com/nba/player/_/id/6442	PG	26
9	Boston Celtics	Nick King	0	http://www.espn.com/nba/player/_/id/3057182	F	23

10 rows × 30 columns

In [721]:

```
#Removendo colunas que não serão úteis na análise:
df = df.drop(columns=["URL", "COLLEGE"])
```

In [722]:

```
#Formatação do campo Salário
df["SALARY"] = df["SALARY"].replace("Not signed",0)
#df["SALARY"].str.replace(',','.')
df["SALARY"] = df["SALARY"].str.replace(',','').astype(float)
df["SALARY"] = df["SALARY"].fillna(0)
df['SALARY'] = df['SALARY']/1000000
```

In [723]:

```
df["PPG_LAST_SEASON"] = df["PPG_LAST_SEASON"].fillna(0)
df["PER_LAST_SEASON"] = df["PER_LAST_SEASON"].fillna(0)
df["APG_LAST_SEASON"] = df["APG_LAST_SEASON"].fillna(0)
df["RPG_LAST_SEASON"] = df["RPG_LAST_SEASON"].fillna(0)
```

In [724]:

```
#df.head(100)
# deletar linhas que possuem salário zerado
df.drop(df[df["SALARY"] == 0.000].index, inplace=True)
# deletar linhas que possuem idade = '-'
df.drop(df[df["AGE"] == '-'].index, inplace=True)
# Deletar linhas com eficiência menor que 0
df.drop(df[df["PER_LAST_SEASON"] <= 0].index, inplace=True)
```

In [725]:

```
#Análise geral dos dados do Stephen Curry

df[df.NAME == "Stephen Curry"]
```

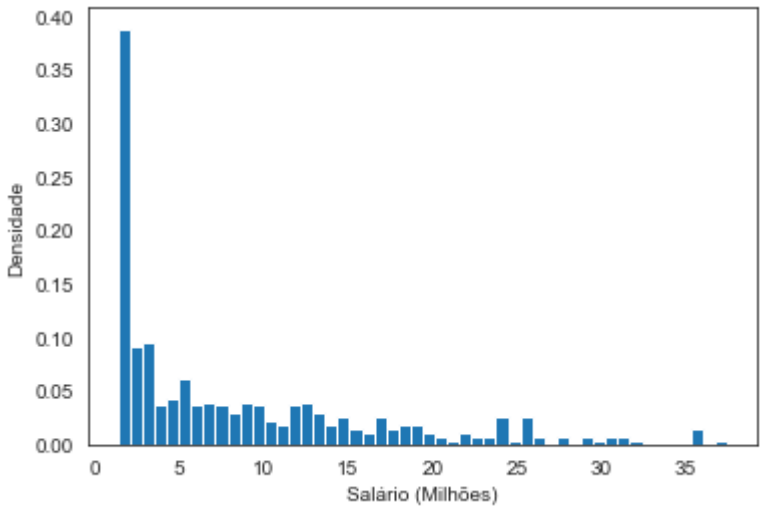
Out[725]:

	TEAM	NAME	EXPERIENCE	POSITION	AGE	HT	WT	SALARY	PPG_LAST_S
103	Golden State Warriors	Stephen Curry	9	PG	30	190.5	85.97	37.457154	

1 rows × 28 columns

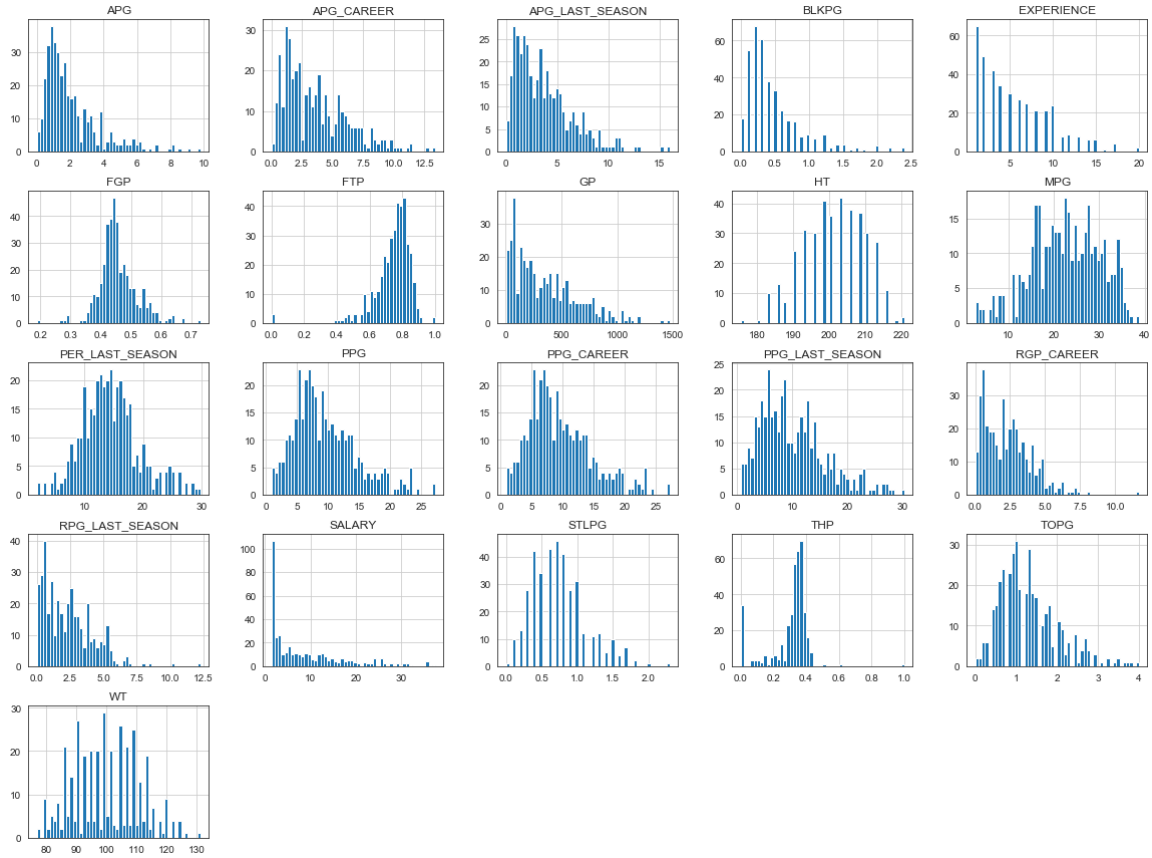
In [726]:

```
# Histograma de Salários
plt.hist(df['SALARY'], density=True, bins=50)
plt.xlabel('Salário (Milhões)')
plt.ylabel('Densidade')
plt.show()
```



In [727]:

```
hist = df.hist(bins=50, figsize=(20,15))
```



In [729]:

```
# Minutos por jogo na quadra.
df.sort_values(by="MPG", ascending=False, inplace=True)
df[["NAME","MPG","SALARY"]].head(20)
```

Out[729]:

	NAME	MPG	SALARY
147	LeBron James	38.8	35.654150
105	Kevin Durant	37.1	30.000000
518	Damian Lillard	36.4	27.977689
493	Andrew Wiggins	36.2	25.467250
295	Carmelo Anthony	35.9	1.512601
457	John Wall	35.9	19.169800
310	Chris Paul	35.3	35.654150
230	Blake Griffin	35.0	32.088932
492	Karl-Anthony Towns	34.9	7.839435
338	Anthony Davis	34.8	25.434263
479	Luol Deng	34.7	1.512601
352	LaMarcus Aldridge	34.6	22.347015
361	Rudy Gay	34.6	10.087200
418	Dwyane Wade	34.5	1.512601
446	Dwight Howard	34.5	5.337000
290	Dirk Nowitzki	34.4	5.000000
510	Russell Westbrook	34.4	35.654150
103	Stephen Curry	34.4	37.457154
138	Lonzo Ball	34.2	7.461960
109	Andre Iguodala	34.1	16.000000

In [730]:

```
# Pontos por jogo na ultima temporada
#df.describe()
df.sort_values(by="PPG_LAST_SEASON", ascending=False, inplace=True)
df[["NAME","PPG_LAST_SEASON", "SALARY"]].head(10)
```

Out[730]:

	NAME	PPG_LAST_SEASON	SALARY
306	James Harden	30.4	35.650150
338	Anthony Davis	28.1	25.434263
147	LeBron James	27.5	35.654150
518	Damian Lillard	26.9	27.977689
259	Giannis Antetokounmpo	26.9	24.157303
105	Kevin Durant	26.4	30.000000
103	Stephen Curry	26.4	37.457154
510	Russell Westbrook	25.4	35.654150
102	DeMarcus Cousins	25.2	5.337000
161	Devin Booker	24.9	3.314365

In [731]:

```
# Eficiência por jogo na última temporada
df.sort_values(by="PER_LAST_SEASON", ascending=False, inplace=True)
df[["NAME", "PER_LAST_SEASON", "SALARY"]].head(10)
```

Out[731]:

	NAME	PER_LAST_SEASON	SALARY
306	James Harden	29.87	35.650150
338	Anthony Davis	28.98	25.434263
147	LeBron James	28.65	35.654150
103	Stephen Curry	28.32	37.457154
259	Giannis Antetokounmpo	27.37	24.157303
468	Monte Morris	27.14	1.349383
317	MarShon Brooks	26.40	1.656092
105	Kevin Durant	26.05	30.000000
88	Kawhi Leonard	26.03	23.114067
129	Boban Marjanovic	25.98	7.000000

In [732]:

```
# Idade dos jogadores
df.sort_values(by="AGE", ascending=False, inplace=True)
df[["NAME","AGE", "SALARY"]].head(50)
```

Out[732]:

	NAME	AGE	SALARY
374	Vince Carter	41	1.512601
290	Dirk Nowitzki	40	5.000000
407	Udonis Haslem	38	1.512601
360	Pau Gasol	38	16.800000
189	Zach Randolph	37	11.692308
213	Kyle Korver	37	7.560000
225	Jose Calderon	37	1.512601
308	Nene	36	3.651480
418	Dwyane Wade	36	1.512601
164	Tyson Chandler	36	13.585000
400	Tony Parker	36	5.000000
209	Channing Frye	35	1.512601
284	Devin Harris	35	1.512601
109	Andre Iguodala	34	16.000000
548	Thabo Sefolosha	34	5.250000
238	Zaza Pachulia	34	1.512601
125	Marcin Gortat	34	13.565218
278	J.J. Barea	34	3.710850
500	Raymond Felton	34	1.512601
295	Carmelo Anthony	34	1.512601
75	JJ Redick	34	12.250000
352	LaMarcus Aldridge	33	22.347015
220	JR Smith	33	14.720000
54	Courtney Lee	33	12.253780
314	PJ Tucker	33	7.969537
491	Anthony Tolliver	33	4.750000
479	Luol Deng	33	1.512601
322	Marc Gasol	33	24.119025
467	Paul Millsap	33	29.230769
157	Trevor Ariza	33	15.000000
113	Shaun Livingston	33	8.307692
26	Jared Dudley	33	9.530000
310	Chris Paul	33	35.654150
147	LeBron James	33	35.654150
481	Taj Gibson	33	14.000000
193	Omer Asik	32	11.286516
333	Garrett Temple	32	8.000000

	NAME	AGE	SALARY
361	Rudy Gay	32	10.087200
210	George Hill	32	19.000000
456	Jason Smith	32	5.450000
289	Salah Mejri	32	1.512601
89	Kyle Lowry	32	31.200000
402	Marvin Williams	32	14.087500
7	Al Horford	32	28.928709
305	Gerald Green	32	1.512601
353	Marco Belinelli	32	6.153846
437	Timofey Mozgov	32	16.000000
21	DeMarre Carroll	32	15.400000
405	Goran Dragic	32	18.109175
446	Dwight Howard	32	5.337000

4. Conjunto de treinamento e conjunto de testes.

4.1 Amostragem aleatória

In [733]:

```
from sklearn.model_selection import train_test_split

test_size=0.2
random_state=42

train_set, test_set=train_test_split(df,
                                     test_size=test_size,
                                     random_state=random_state)
```

In [734]:

```
train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 305 entries, 248 to 235
Data columns (total 28 columns):
TEAM               305 non-null object
NAME               305 non-null object
EXPERIENCE         305 non-null int64
POSITION           305 non-null object
AGE                305 non-null object
HT                 305 non-null float64
WT                 305 non-null float64
SALARY             305 non-null float64
PPG_LAST_SEASON    305 non-null float64
APG_LAST_SEASON    305 non-null float64
RPG_LAST_SEASON    305 non-null float64
PER_LAST_SEASON    305 non-null float64
PPG_CAREER         305 non-null float64
APG_CAREER         305 non-null float64
RGP_CAREER         305 non-null float64
GP                 305 non-null int64
MPG                305 non-null float64
FGM_FGA            305 non-null object
FGP                305 non-null float64
THM_THA            305 non-null object
THP                305 non-null float64
FTM_FTA            305 non-null object
FTP                305 non-null float64
APG                305 non-null float64
BLKPG              305 non-null float64
STLPG              305 non-null float64
TOPG               305 non-null float64
PPG                305 non-null float64
dtypes: float64(19), int64(2), object(7)
memory usage: 69.1+ KB
```

In [735]:

```
test_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 77 entries, 371 to 285
Data columns (total 28 columns):
TEAM               77 non-null object
NAME               77 non-null object
EXPERIENCE         77 non-null int64
POSITION           77 non-null object
AGE                77 non-null object
HT                 77 non-null float64
WT                 77 non-null float64
SALARY             77 non-null float64
PPG_LAST_SEASON    77 non-null float64
APG_LAST_SEASON    77 non-null float64
RPG_LAST_SEASON    77 non-null float64
PER_LAST_SEASON    77 non-null float64
PPG_CAREER         77 non-null float64
APG_CAREER         77 non-null float64
RGP_CAREER         77 non-null float64
GP                 77 non-null int64
MPG                77 non-null float64
FGM_FGA            77 non-null object
FGP                77 non-null float64
THM_THA            77 non-null object
THP                77 non-null float64
FTM_FTA            77 non-null object
FTP                77 non-null float64
APG                77 non-null float64
BLKPG              77 non-null float64
STLPG              77 non-null float64
TOPG               77 non-null float64
PPG                77 non-null float64
dtypes: float64(19), int64(2), object(7)
memory usage: 17.4+ KB
```

4.2 Amostragem estratificada

In [736]:

```
import numpy as np

df["EFICIENCIA_CAT"]= np.ceil(df["PER_LAST_SEASON"]/2) # Limitando o numero de categori
as
df["EFICIENCIA_CAT"].where(df["EFICIENCIA_CAT"]<8,8.0, inplace=True)
```

In [737]:

```
from sklearn.model_selection import train_test_split

test_size=0.2 # tamanho do conjunto de teste: 20%
random_state=42 # Semente aleatória
strat_train_set, strat_test_set = train_test_split(df,
                                                   test_size=test_size,
                                                   random_state=random_state,
                                                   stratify=df["EFICIENCIA_CAT"])
```

In [738]:

```
S_overall = df['EFICIENCIA_CAT'].value_counts()/len(df)
S_overall.rename('EFICIENCIA_CAT_OVERALL', inplace=True)
```

Out[738]:

```
8.0    0.502618
7.0    0.188482
6.0    0.123037
5.0    0.104712
4.0    0.047120
3.0    0.023560
2.0    0.005236
1.0    0.005236
Name: EFICIENCIA_CAT_OVERALL, dtype: float64
```

In [739]:

```
# amostras aleatórias com a categoria da eficiência
from sklearn.model_selection import train_test_split

test_size=0.2
random_state=42

train_set_with_efi_cat, test_set_with_efi_cat=train_test_split(df,
                                    test_size=test_size,
                                    random_state=random_state)

S_random = test_set_with_efi_cat['EFICIENCIA_CAT'].value_counts()/len(test_set_with_efi
_cat)
S_random.rename('EFICIENCIA_CAT_RANDOM',inplace=True)
```

Out[739]:

```
8.0    0.389610
7.0    0.298701
5.0    0.155844
6.0    0.116883
4.0    0.038961
Name: EFICIENCIA_CAT_RANDOM, dtype: float64
```

In [740]:

```
# Proporções da categoria de eficiencia no conjunto de testes gerados usando amostragem
stratificada
S_stratified=strat_test_set["EFICIENCIA_CAT"].value_counts()/len(strat_test_set)
S_stratified.rename('EFICIENCIA_CAT_STRATIFIED', inplace=True)
```

Out[740]:

```
8.0    0.506494
7.0    0.194805
6.0    0.116883
5.0    0.103896
4.0    0.051948
3.0    0.025974
Name: EFICIENCIA_CAT_STRATIFIED, dtype: float64
```

In [741]:

```
# Comparação entre as proporções
pd.concat([S_overall, S_random, S_stratified], axis=1).sort_index()
```

Out[741]:

	EFICIENCIA_CAT_OVERALL	EFICIENCIA_CAT_RANDOM	EFICIENCIA_CAT_STRATIFIED
1.0	0.005236	NaN	NaN
2.0	0.005236	NaN	NaN
3.0	0.023560	NaN	0.025974
4.0	0.047120	0.038961	0.051948
5.0	0.104712	0.155844	0.103896
6.0	0.123037	0.116883	0.116883
7.0	0.188482	0.298701	0.194805
8.0	0.502618	0.389610	0.506494

In [742]:

```
# Removendo o atributo EFICIENCIA_CAT
strat_train_set = strat_train_set.drop('EFICIENCIA_CAT', axis=1)
strat_test_set = strat_test_set.drop('EFICIENCIA_CAT', axis=1)
```

5. Visualização dos dados

5.1 Explorando o conjunto de treinamentos

In [743]:

```
nba=strat_train_set.copy()
print(nba.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 305 entries, 401 to 508
Data columns (total 28 columns):
TEAM               305 non-null object
NAME               305 non-null object
EXPERIENCE         305 non-null int64
POSITION           305 non-null object
AGE                305 non-null object
HT                 305 non-null float64
WT                 305 non-null float64
SALARY             305 non-null float64
PPG_LAST_SEASON    305 non-null float64
APG_LAST_SEASON    305 non-null float64
RPG_LAST_SEASON    305 non-null float64
PER_LAST_SEASON    305 non-null float64
PPG_CAREER         305 non-null float64
APG_CAREER         305 non-null float64
RGP_CAREER         305 non-null float64
GP                 305 non-null int64
MPG                305 non-null float64
FGM_FGA            305 non-null object
FGP                305 non-null float64
THM_THA            305 non-null object
THP                305 non-null float64
FTM_FTA            305 non-null object
FTP                305 non-null float64
APG                305 non-null float64
BLKPG              305 non-null float64
STLPG              305 non-null float64
TOPG               305 non-null float64
PPG                305 non-null float64
dtypes: float64(19), int64(2), object(7)
memory usage: 69.1+ KB
None
```

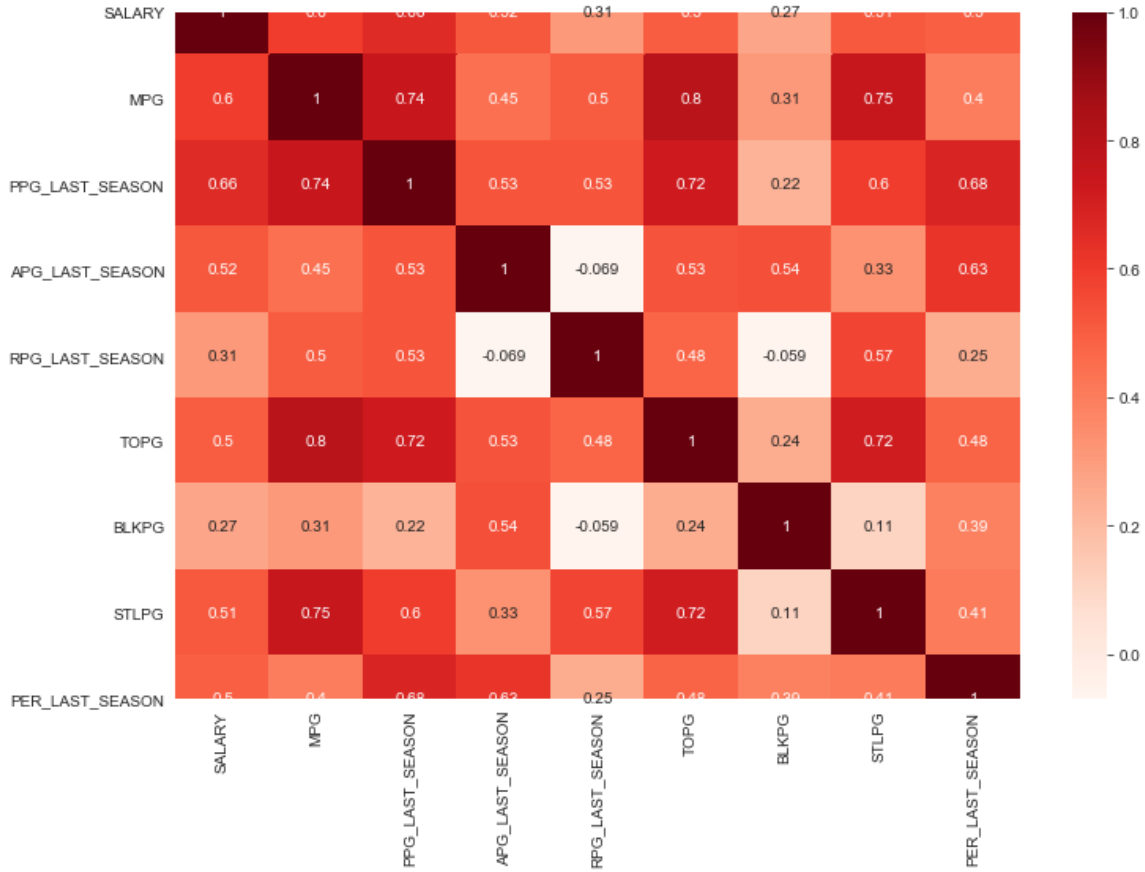
5.1.1 Buscando correlações

In [744]:

```
# Selecionando as características que julguei mais importantes para correlacionar.
import seaborn as sns

#sns.set_style("white")
plt.figure(figsize=(12,8))
heatmap_per = nba[["SALARY", "MPG", "PPG_LAST_SEASON", "APG_LAST_SEASON", "RPG_LAST_SEASON", "TOPG", "BLKPG", "STLPG", "AGE", "PER_LAST_SEASON"]]
df_heat = heatmap_per.corr()
sns.heatmap(df_heat, annot=True, cmap=plt.cm.Reds)
plt.show()

#Analisando as características é possível perceber que o campo MPG ("Minutos por jogo em quadra") e
# ("TOPG: Rotatividade por jogo") e PPG_LAST_SEASON ("Pontos por jogo na ultima temporada")
# são os atributos que mais se cocrelacionam
```



5.1.1.2 Correlação entre pontos por jogo e salário

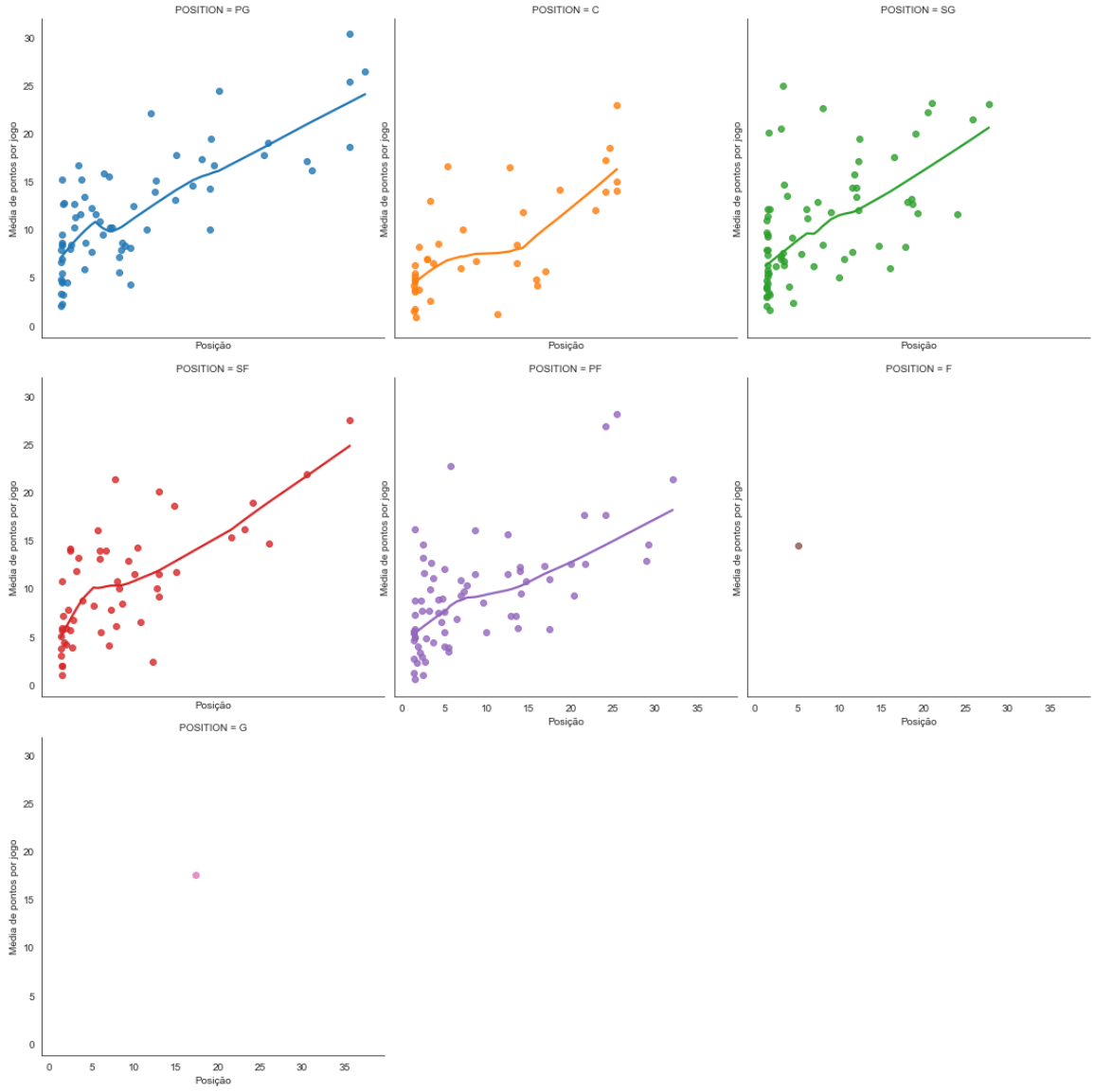
In [745]:

```
sns.lmplot(x="SALARY", y="PPG_LAST_SEASON", hue="POSITION", col="POSITION", col_wrap=3,
data=nba, lowess=True).set(xlabel='Posição', ylabel='Média de pontos por jogo')
```

```
C:\Users\Thiago\Anaconda3\lib\site-packages\statsmodels\nonparametric\smoo
thers_lowess.py:165: RuntimeWarning: invalid value encountered in true_div
ide
  res = _lowess(y, x, frac=frac, it=it, delta=delta)
```

Out[745]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf863bd88>
```

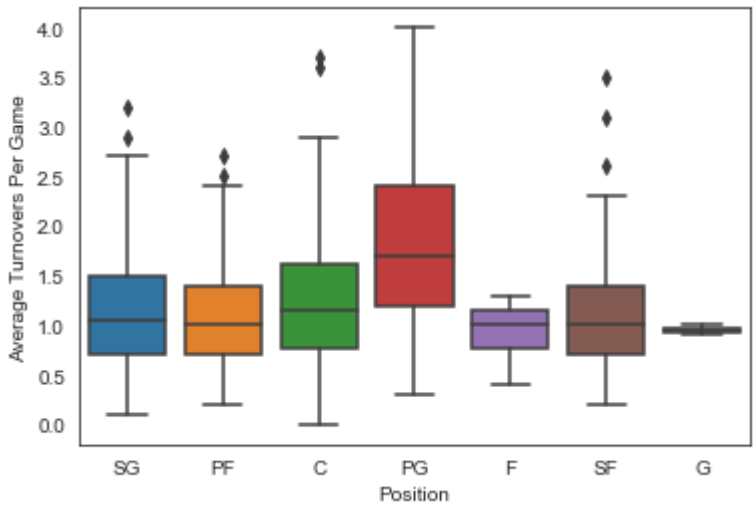


In [746]:

```
#Qual posição é mais propensa a rotatividade?
sns.boxplot(x="POSITION", y="TOPG", data=df).set(xlabel="Position", ylabel="Average Tur
novers Per Game")
```

Out[746]:

```
[Text(0, 0.5, 'Average Turnovers Per Game'), Text(0.5, 0, 'Position')]
```

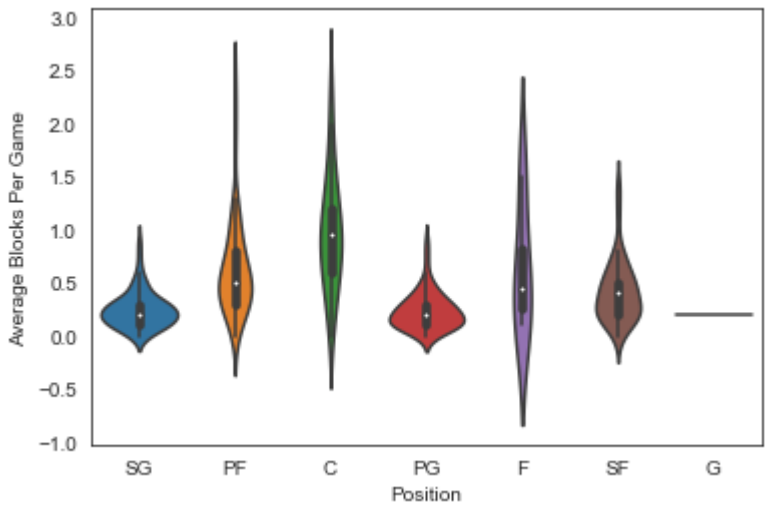


In [747]:

```
# Qual posição tem os maiores bloqueios?
sns.violinplot(x="POSITION", y="BLKPG", data=df).set(xlabel="Position", ylabel="Average
Blocks Per Game")
```

Out[747]:

```
[Text(0, 0.5, 'Average Blocks Per Game'), Text(0.5, 0, 'Position')]
```



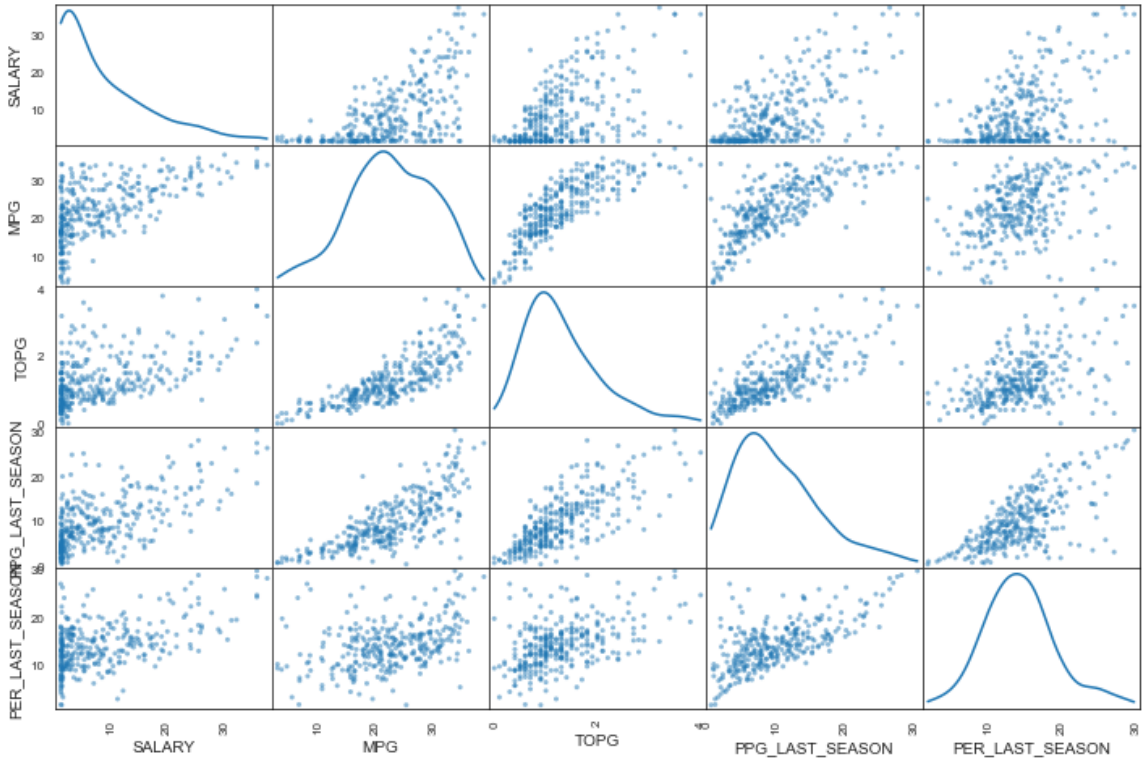
In [696]:

```
from pandas.plotting import scatter_matrix

atributes = ["SALARY", "MPG", "TOPG", "PPG_LAST_SEASON", "PER_LAST_SEASON"]
scatter_matrix(heatmap_per[atributes], figsize=(12,8), diagonal='kde')
```

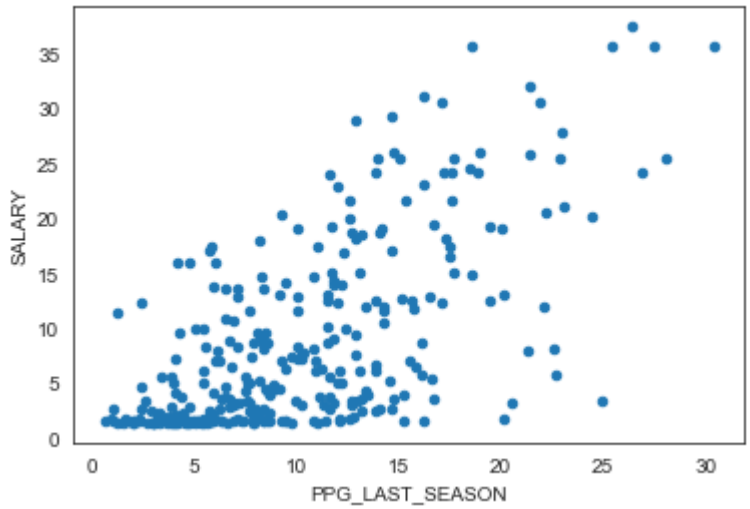
Out[696]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF884C54
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF8BD340
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF8C0AC0
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF8C43A0
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9CACB0
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9CE5C0
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9D1DCC
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9D55DC
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9D619C
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9D99B8
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9E0514
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9E3E20
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9E7734
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9EAE44
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9EE654
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9F24E8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9F5770
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9F9080
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CF9FC890
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CFA001A4
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000001CFA03AB4
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CFA075C0
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CFA0AED4
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CFA0E5E4
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000001CFA11FF4
8>]],
      dtype=object)
```



In [748]:

```
g=heatmap_per.plot(kind='scatter',x='PPG_LAST_SEASON', y="SALARY")
```



In [749]:

```
corr_matrix=nba.corr()
corr_matrix['SALARY'].sort_values(ascending=False)
```

Out[749]:

```
SALARY              1.000000
PPG_LAST_SEASON     0.658278
PPG                 0.606026
PPG_CAREER          0.606026
MPG                 0.597137
APG_LAST_SEASON     0.517157
STLPG               0.514620
TOPG                0.498969
PER_LAST_SEASON     0.498125
APG_CAREER          0.468525
GP                  0.433706
APG                 0.409380
EXPERIENCE          0.389446
RPG_LAST_SEASON     0.310001
BLKPG               0.273259
RGP_CAREER          0.266624
FGP                 0.173544
FTP                 0.155398
THP                 0.134140
WT                  0.080882
HT                  0.012826
Name: SALARY, dtype: float64
```

6. Preparar modelos de AM

6.1 Limpeza dos dados

In [750]:

```
nba=strat_train_set.drop('SALARY', axis=1) # X
nba_labels=strat_train_set['SALARY'].copy() # y
```

6.1.1 Tratando os valores faltantes com o Scikit-Learn

In [751]:

```
nba=strat_train_set.drop('SALARY', axis=1) #X
nba_num  = nba.drop(['TEAM','NAME','POSITION','AGE','FGM_FGA','THM_THA','FTM_FTA'], axi
s=1)

from sklearn.impute import SimpleImputer # Estimador

imputer = SimpleImputer(strategy="mean") # Objeto da classe Imputer
# Imputer é um estimador
imputer.fit(nba_num)

print(imputer.statistics_) # Média de cada coluna do nba_num
print(nba_num.mean().values)

#Imputer é também um "Transformador": Parametros estimados + dado de entrada -> dado tr
ansformado
X=imputer.transform(nba_num)

# dado de entrada -> parâmetros estimados; parametros estimados + dado de entrada -> da
do transformado
#X=imputer.fit_transform(nba_num)

nba_num_transformed=pd.DataFrame(X,columns=nba_num.columns) # de nDarray para DataFrame

nba_num_transformed.info()
```

```
[5.59016393e+00 2.00884852e+02 9.93933115e+01 1.01524590e+01
 3.70885246e+00 2.33016393e+00 1.43281639e+01 9.68918033e+00
 3.67475410e+00 2.32229508e+00 3.57675410e+02 2.28216393e+01
 4.54806557e-01 3.06718033e-01 7.40616393e-01 2.12000000e+00
 4.61639344e-01 7.50819672e-01 1.32196721e+00 9.68918033e+00]
[5.59016393e+00 2.00884852e+02 9.93933115e+01 1.01524590e+01
 3.70885246e+00 2.33016393e+00 1.43281639e+01 9.68918033e+00
 3.67475410e+00 2.32229508e+00 3.57675410e+02 2.28216393e+01
 4.54806557e-01 3.06718033e-01 7.40616393e-01 2.12000000e+00
 4.61639344e-01 7.50819672e-01 1.32196721e+00 9.68918033e+00]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 305 entries, 0 to 304
Data columns (total 20 columns):
EXPERIENCE         305 non-null float64
HT                 305 non-null float64
WT                 305 non-null float64
PPG_LAST_SEASON    305 non-null float64
APG_LAST_SEASON    305 non-null float64
RPG_LAST_SEASON    305 non-null float64
PER_LAST_SEASON    305 non-null float64
PPG_CAREER         305 non-null float64
APG_CAREER         305 non-null float64
RGP_CAREER         305 non-null float64
GP                 305 non-null float64
MPG                305 non-null float64
FGP                305 non-null float64
THP                305 non-null float64
FTP                305 non-null float64
APG                305 non-null float64
BLKPG              305 non-null float64
STLPG              305 non-null float64
TOPG               305 non-null float64
PPG                305 non-null float64
dtypes: float64(20)
memory usage: 47.8 KB
```

6.2 Manipulando texto e atributos categóricos

In [752]:

```
nba = strat_train_set.drop('SALARY', axis=1) #X
nba_cat = nba[['TEAM','POSITION']] # Apenas atributos categoricos
nba_cat.head(10)
```

Out[752]:

	TEAM	POSITION
401	Charlotte Hornets	PG
444	Washington Wizards	C
187	Sacramento Kings	PG
351	New Orleans Pelicans	SG
539	Utah Jazz	SF
431	Orlando Magic	PF
361	San Antonio Spurs	SF
279	Dallas Mavericks	SF
13	Boston Celtics	PG
229	Detroit Pistons	SG

6.2.2 OneHotEncoder

In [753]:

```
from sklearn.preprocessing import OneHotEncoder # Transformador

nba_cat_reshaped=nba_cat.to_numpy().reshape(-1,1)
print(nba_cat_reshaped)

encoder=OneHotEncoder()
nba_cat_1hot = encoder.fit_transform(nba_cat_reshaped) # A Saída é um scipy sparse matr
ix

print(nba_cat_1hot.toarray())
```

```
[['Charlotte Hornets']
 ['PG']
 ['Washington Wizards']
 ['C']
 ['Sacramento Kings']
 ['PG']
 ['New Orleans Pelicans']
 ['SG']
 ['Utah Jazz']
 ['SF']
 ['Orlando Magic']
 ['PF']
 ['San Antonio Spurs']
 ['SF']
 ['Dallas Mavericks']
 ['SF']
 ['Boston Celtics']
 ['PG']
 ['Detroit Pistons']
 ['SG']
 ['Miami Heat']
 ['SG']
 ['Boston Celtics']
 ['F']
 ['Detroit Pistons']
 ['SF']
 ['LA Clippers']
 ['SF']
 ['Dallas Mavericks']
 ['SG']
 ['Miami Heat']
 ['PG']
 ['Minnesota Timberwolves']
 ['PG']
 ['Houston Rockets']
 ['SG']
 ['LA Clippers']
 ['SG']
 ['Utah Jazz']
 ['PF']
 ['Chicago Bulls']
 ['PF']
 ['New York Knicks']
 ['SG']
 ['Houston Rockets']
 ['SF']
 ['Los Angeles Lakers']
 ['PG']
 ['Memphis Grizzlies']
 ['C']
 ['Atlanta Hawks']
 ['SG']
 ['Houston Rockets']
 ['PG']
 ['Indiana Pacers']
 ['PF']
 ['Portland Trail Blazers']
 ['PG']
 ['New Orleans Pelicans']
 ['SG']
 ['Golden State Warriors']
```

```
['SG']
['Chicago Bulls']
['PG']
['Sacramento Kings']
['PF']
['Charlotte Hornets']
['PF']
['Los Angeles Lakers']
['C']
['Orlando Magic']
['PF']
['LA Clippers']
['PF']
['Atlanta Hawks']
['C']
['Detroit Pistons']
['PF']
['Toronto Raptors']
['SG']
['Sacramento Kings']
['SG']
['Denver Nuggets']
['SG']
['Atlanta Hawks']
['SG']
['New Orleans Pelicans']
['SF']
['Philadelphia 76ers']
['PG']
['Phoenix Suns']
['PG']
['Minnesota Timberwolves']
['SF']
['Memphis Grizzlies']
['SG']
['Denver Nuggets']
['PF']
['Toronto Raptors']
['SF']
['Sacramento Kings']
['SF']
['Milwaukee Bucks']
['PF']
['Utah Jazz']
['PG']
['Philadelphia 76ers']
['SG']
['Memphis Grizzlies']
['PG']
['Milwaukee Bucks']
['C']
['Memphis Grizzlies']
['SG']
['Philadelphia 76ers']
['PF']
['Brooklyn Nets']
['PG']
['Golden State Warriors']
['C']
['Boston Celtics']
['PG']
```

```
['Utah Jazz']
['C']
['Detroit Pistons']
['SG']
['Chicago Bulls']
['SG']
['Miami Heat']
['PF']
['Houston Rockets']
['PF']
['Chicago Bulls']
['C']
['Utah Jazz']
['PG']
['Denver Nuggets']
['PG']
['Utah Jazz']
['SF']
['Memphis Grizzlies']
['SG']
['Los Angeles Lakers']
['SG']
['Milwaukee Bucks']
['SG']
['Phoenix Suns']
['SF']
['Golden State Warriors']
['PG']
['Indiana Pacers']
['PF']
['Denver Nuggets']
['PF']
['Cleveland Cavaliers']
['SF']
['Brooklyn Nets']
['SF']
['Golden State Warriors']
['PF']
['Cleveland Cavaliers']
['PG']
['San Antonio Spurs']
['SG']
['Orlando Magic']
['C']
['Dallas Mavericks']
['PG']
['Atlanta Hawks']
['SG']
['Miami Heat']
['SG']
['LA Clippers']
['SG']
['Milwaukee Bucks']
['SG']
['Phoenix Suns']
['PF']
['Atlanta Hawks']
['SF']
['Orlando Magic']
['PF']
['Indiana Pacers']
```

```
['PF']
['Oklahoma City Thunder']
['SF']
['Oklahoma City Thunder']
['SF']
['Charlotte Hornets']
['SF']
['Toronto Raptors']
['PG']
['San Antonio Spurs']
['PG']
['Portland Trail Blazers']
['SF']
['Brooklyn Nets']
['SF']
['Portland Trail Blazers']
['PF']
['Dallas Mavericks']
['PF']
['LA Clippers']
['C']
['Utah Jazz']
['SF']
['Philadelphia 76ers']
['PG']
['Oklahoma City Thunder']
['SF']
['New Orleans Pelicans']
['PG']
['Oklahoma City Thunder']
['SG']
['Denver Nuggets']
['PG']
['New York Knicks']
['G']
['Sacramento Kings']
['SG']
['Philadelphia 76ers']
['PF']
['Minnesota Timberwolves']
['PF']
['Washington Wizards']
['SF']
['Portland Trail Blazers']
['SG']
['Toronto Raptors']
['SF']
['Miami Heat']
['SG']
['Orlando Magic']
['C']
['Milwaukee Bucks']
['PF']
['Phoenix Suns']
['SG']
['LA Clippers']
['SF']
['Atlanta Hawks']
['SF']
['Washington Wizards']
['C']
```

```
['Utah Jazz']
['SG']
['Detroit Pistons']
['PG']
['San Antonio Spurs']
['C']
['Phoenix Suns']
['C']
['Boston Celtics']
['PF']
['Denver Nuggets']
['PF']
['LA Clippers']
['PF']
['Denver Nuggets']
['SF']
['Cleveland Cavaliers']
['PF']
['San Antonio Spurs']
['SF']
['New Orleans Pelicans']
['SF']
['Boston Celtics']
['PF']
['Memphis Grizzlies']
['PF']
['Milwaukee Bucks']
['SG']
['Minnesota Timberwolves']
['SG']
['Minnesota Timberwolves']
['PG']
['Sacramento Kings']
['C']
['Indiana Pacers']
['PF']
['Golden State Warriors']
['PG']
['LA Clippers']
['PG']
['Denver Nuggets']
['SG']
['New Orleans Pelicans']
['PF']
['Miami Heat']
['SG']
['Washington Wizards']
['PG']
['New York Knicks']
['PG']
['Boston Celtics']
['SG']
['Memphis Grizzlies']
['SG']
['Brooklyn Nets']
['PF']
['Washington Wizards']
['PF']
['Toronto Raptors']
['PF']
['Golden State Warriors']
```

```
['SG']
['Portland Trail Blazers']
['SG']
['Washington Wizards']
['SG']
['Toronto Raptors']
['PF']
['Detroit Pistons']
['C']
['Brooklyn Nets']
['PG']
['Portland Trail Blazers']
['PF']
['New York Knicks']
['PG']
['Charlotte Hornets']
['SG']
['Washington Wizards']
['SF']
['Memphis Grizzlies']
['PG']
['LA Clippers']
['SG']
['Portland Trail Blazers']
['PF']
['Indiana Pacers']
['SF']
['Utah Jazz']
['SG']
['Brooklyn Nets']
['SG']
['Los Angeles Lakers']
['SF']
['Washington Wizards']
['SF']
['Milwaukee Bucks']
['PF']
['Phoenix Suns']
['SG']
['Charlotte Hornets']
['SG']
['Detroit Pistons']
['SG']
['Toronto Raptors']
['PG']
['LA Clippers']
['PG']
['Toronto Raptors']
['SF']
['Philadelphia 76ers']
['SF']
['Dallas Mavericks']
['C']
['Dallas Mavericks']
['PG']
['Indiana Pacers']
['SG']
['Milwaukee Bucks']
['PG']
['Detroit Pistons']
['PF']
```

```
['LA Clippers']
['PG']
['Brooklyn Nets']
['C']
['San Antonio Spurs']
['SG']
['Detroit Pistons']
['PG']
['Golden State Warriors']
['PG']
['Orlando Magic']
['SF']
['Charlotte Hornets']
['C']
['Los Angeles Lakers']
['C']
['Detroit Pistons']
['C']
['Indiana Pacers']
['PG']
['Boston Celtics']
['PF']
['Golden State Warriors']
['PF']
['Detroit Pistons']
['PG']
['Denver Nuggets']
['C']
['Miami Heat']
['C']
['Los Angeles Lakers']
['SG']
['Phoenix Suns']
['PG']
['Minnesota Timberwolves']
['PG']
['Utah Jazz']
['PF']
['Charlotte Hornets']
['PG']
['Houston Rockets']
['PF']
['Atlanta Hawks']
['SG']
['Dallas Mavericks']
['SF']
['Cleveland Cavaliers']
['SG']
['Chicago Bulls']
['C']
['Charlotte Hornets']
['SG']
['Washington Wizards']
['SG']
['LA Clippers']
['C']
['Boston Celtics']
['SF']
['Chicago Bulls']
['PF']
['Dallas Mavericks']
```

```
['PG']
['Philadelphia 76ers']
['SG']
['Milwaukee Bucks']
['SG']
['Indiana Pacers']
['SG']
['Oklahoma City Thunder']
['PG']
['San Antonio Spurs']
['SF']
['New York Knicks']
['SF']
['New Orleans Pelicans']
['PF']
['LA Clippers']
['SF']
['Los Angeles Lakers']
['SF']
['Cleveland Cavaliers']
['SG']
['Sacramento Kings']
['PG']
['Dallas Mavericks']
['PF']
['Los Angeles Lakers']
['PG']
['Houston Rockets']
['C']
['Chicago Bulls']
['PG']
['Cleveland Cavaliers']
['C']
['Utah Jazz']
['PG']
['Miami Heat']
['C']
['Detroit Pistons']
['PF']
['Toronto Raptors']
['PG']
['Oklahoma City Thunder']
['C']
['Milwaukee Bucks']
['SF']
['San Antonio Spurs']
['SG']
['Oklahoma City Thunder']
['PF']
['Philadelphia 76ers']
['C']
['Sacramento Kings']
['PG']
['New York Knicks']
['PF']
['Washington Wizards']
['C']
['Dallas Mavericks']
['C']
['Houston Rockets']
['PG']
```

```
['San Antonio Spurs']
['PG']
['Philadelphia 76ers']
['C']
['Indiana Pacers']
['SF']
['Phoenix Suns']
['PF']
['New York Knicks']
['SG']
['Orlando Magic']
['PG']
['New Orleans Pelicans']
['PF']
['Atlanta Hawks']
['C']
['Cleveland Cavaliers']
['PF']
['Philadelphia 76ers']
['PG']
['Denver Nuggets']
['PF']
['New Orleans Pelicans']
['PG']
['Miami Heat']
['PF']
['New York Knicks']
['PG']
['Houston Rockets']
['SF']
['Orlando Magic']
['C']
['Cleveland Cavaliers']
['PF']
['Sacramento Kings']
['SG']
['Minnesota Timberwolves']
['PF']
['Miami Heat']
['PF']
['Houston Rockets']
['PF']
['New York Knicks']
['C']
['Orlando Magic']
['SF']
['Charlotte Hornets']
['PF']
['Cleveland Cavaliers']
['SG']
['Denver Nuggets']
['SG']
['New Orleans Pelicans']
['PF']
['New Orleans Pelicans']
['C']
['Miami Heat']
['SF']
['Golden State Warriors']
['PF']
['New York Knicks']
```

```
['PF']
['Portland Trail Blazers']
['SG']
['Memphis Grizzlies']
['SG']
['Denver Nuggets']
['PG']
['Brooklyn Nets']
['PF']
['Memphis Grizzlies']
['PF']
['New York Knicks']
['SG']
['Charlotte Hornets']
['C']
['Milwaukee Bucks']
['PG']
['Miami Heat']
['SG']
['Boston Celtics']
['PF']
['Washington Wizards']
['PG']
['Milwaukee Bucks']
['PF']
['Utah Jazz']
['C']
['Minnesota Timberwolves']
['SF']
['Dallas Mavericks']
['PF']
['Washington Wizards']
['PF']
['Sacramento Kings']
['PF']
['Charlotte Hornets']
['PF']
['Chicago Bulls']
['PG']
['Houston Rockets']
['PG']
['Cleveland Cavaliers']
['PG']
['Brooklyn Nets']
['SG']
['Toronto Raptors']
['PG']
['Phoenix Suns']
['SF']
['Orlando Magic']
['PG']
['Oklahoma City Thunder']
['SF']
['LA Clippers']
['SG']
['Memphis Grizzlies']
['SF']
['Los Angeles Lakers']
['SF']
['Oklahoma City Thunder']
['SG']]
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

6.3 Escalonamento das características

6.3.1 Normalização

In [754]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
nba_num_normalized=scaler.fit_transform(nba_num) #Normlizado

nba_num_normalized
```

Out[754]:

```
array([[0.31578947, 0.22222222, 0.0782241 , ..., 0.56521739, 0.525     ,
        0.68441065],
       [0.        , 0.77777778, 0.63482606, ..., 0.04347826, 0.025     ,
        0.02281369],
       [0.        , 0.11111111, 0.13030944, ..., 0.30434783, 0.325     ,
        0.2661597 ],
       ...,
       [0.42105263, 0.66666667, 0.43474918, ..., 0.26086957, 0.25      ,
        0.26996198],
       [0.47368421, 0.66666667, 0.52162214, ..., 0.26086957, 0.425     ,
        0.4486692 ],
       [0.21052632, 0.55555556, 0.30424755, ..., 0.39130435, 0.15      ,
        0.14068441]])
```

6.3.2 Padronização

In [755]:

```
from sklearn.preprocessing import StandardScaler # Bom para outliers, ruim para redes n
eurais

scaler=StandardScaler()
nba_num_standardized=scaler.fit_transform(nba_num)

nba_num_standardized
```

Out[755]:

```
array([[ 0.340599  , -1.78246822, -1.48638384, ...,  1.39275141,
         1.07285695,  1.83061476],
       [-1.10892698,  1.14511825,  1.18174019, ..., -1.65051436,
        -1.68501385, -1.62756789],
       [-1.10892698, -2.36798552, -1.23670787, ..., -0.12888147,
        -0.03029137, -0.35559267],
       ...,
       [ 0.82377433,  0.55960096,  0.22265279, ..., -0.38248695,
        -0.44397199, -0.33571805],
       [ 1.06536199,  0.55960096,  0.63908651, ..., -0.38248695,
         0.52128279,  0.59838876],
       [-0.14257633, -0.02591634, -0.40291911, ...,  0.37832949,
        -0.99554615, -1.01145489]])
```

6.4 Pipelines de Transformação

In [756]:

```
nba=strat_train_set.drop('SALARY', axis=1) #X
#nba_num=nba.drop(['TEAM','NAME','POSITION','AGE','FGM_FGA','THM_THA','FTM_FTA'], axis=
1)
nba_num=nba[['PPG_LAST_SEASON','PER_LAST_SEASON', 'TOPG','MPG']]
#nba_num
nba_cat=nba[['TEAM','POSITION']]

nba_num
```

Out[756]:

	PPG_LAST_SEASON	PER_LAST_SEASON	TOPG	MPG
401	22.1	20.62	2.1	33.9
444	1.5	9.89	0.1	4.8
187	7.9	13.02	1.3	18.9
351	6.3	15.18	1.0	16.9
539	11.5	14.42	1.3	23.0
...	...	...	...	...
503	8.4	16.19	1.0	21.9
135	3.9	8.22	0.7	15.8
319	5.7	15.56	1.0	20.7
139	13.2	17.89	1.7	23.4
508	5.0	10.90	0.6	22.5

305 rows × 4 columns

6.4.1 Pipeline para atributos numéricos

In [757]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

num_pipeline=Pipeline([('imputer', SimpleImputer(strategy="median")),
                       ('scaler', StandardScaler())])

nba_num_transformed=num_pipeline.fit_transform(nba_num)

nba_num_transformed
```

Out[757]:

```
array([[ 2.03628394,  1.27900858,  1.07285695,  1.49332   ],
       [-1.47468532, -0.90219289, -1.68501385, -2.429247  ],
       [-0.38389876, -0.26592443, -0.03029137, -0.52862176],
       ...,
       [-0.75885664,  0.25040845, -0.44397199, -0.28598875],
       [ 0.51940887,  0.72405238,  0.52128279,  0.07796077],
       [-0.87816142, -0.69687942, -0.99554615, -0.04335573]])
```

6.4.2 Pipeline para atributos categóricos

In [758]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder # Transformador

nba_cat_reshaped=nba_cat.values.reshape(-1,1)

cat_pipeline = Pipeline([('imputer', SimpleImputer(strategy="constant", fill_value="mis
sing")),
                         ('cat_encoder', OneHotEncoder(sparse=False))])

nba_cat_transformed=cat_pipeline.fit_transform(nba_cat_reshaped);

nba_cat_transformed
```

Out[758]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

6.4.3 Pipeline para atributos numéricos e categóricos

In [759]:

```
from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

num_attribs=["MPG", "PPG_LAST_SEASON", "PER_LAST_SEASON", "TOPG"]

cat_attribs=['TEAM','POSITION']

num_pipeline=Pipeline([('imputer', SimpleImputer(strategy="mean")),
                       ('scaler', StandardScaler())])

cat_pipeline=Pipeline([('imputer', SimpleImputer(strategy="constant",fill_value="missin
g")),
                       ('encoder', OneHotEncoder(sparse=False))])

full_pipeline=ColumnTransformer(transformers=[('num_pipeline', num_pipeline, num_attrib
s),
                                              ('cat_pipeline', cat_pipeline, cat_attrib
s)])

nba_prepared=full_pipeline.fit_transform(nba)
print(nba_prepared.shape)
nba_prepared
```

```
(305, 41)
```

Out[759]:

```
array([[ 1.49332   ,  2.03628394,  1.27900858, ...,  1.        ,
         0.        ,  0.        ],
       [-2.429247  , -1.47468532, -0.90219289, ...,  0.        ,
         0.        ,  0.        ],
       [-0.52862176, -0.38389876, -0.26592443, ...,  1.        ,
         0.        ,  0.        ],
       ...,
       [-0.28598875, -0.75885664,  0.25040845, ...,  0.        ,
         1.        ,  0.        ],
       [ 0.07796077,  0.51940887,  0.72405238, ...,  0.        ,
         1.        ,  0.        ],
       [-0.04335573, -0.87816142, -0.69687942, ...,  0.        ,
         0.        ,  1.        ]])
```

7. Selecionar e treinar o modelo de AM

7.1 Treinando e avaliando o conjunto de treinamento

7.1.1 Modelo de Regressão Linear

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \cdot \mathbf{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad \text{e} \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Aqui:

- n é o número de características de cada instância;
- $\boldsymbol{\theta}$ é o vetor de parâmetros de característica;
- $\mathbf{x}$ é o vetor de características de instância;
- $\hat{y}$ é o vetor de rótulos da i-ésima instância;
- $h_{\boldsymbol{\theta}}(\mathbf{x})$ é a função predição (Hipótese).

Método dos mínimos quadrados:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

In [760]:

```
from sklearn.linear_model import LinearRegression

lin_reg=LinearRegression()
lin_reg.fit(nba_prepared,nba_labels)

print(lin_reg.get_params())
```

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize': Fals
e}
```

In [761]:

```
from sklearn.metrics import mean_squared_error

nba_predicted = lin_reg.predict(nba_prepared)
lin_reg_rmse=np.sqrt(mean_squared_error(nba_labels, nba_predicted))

print(lin_reg_rmse) #underfitting
```

```
5.722186186423075
```

7.1.2 Árvore de Decisão

In [762]:

```
from sklearn.tree import DecisionTreeRegressor

dec_tree_reg=DecisionTreeRegressor()

dec_tree_reg.fit(nba_prepared, nba_labels)
dec_tree_rmse=np.sqrt(mean_squared_error(nba_labels, nba_predicted))

print(dec_tree_reg.get_params())
print(dec_tree_rmse)
```

```
{'ccp_alpha': 0.0, 'criterion': 'mse', 'max_depth': None, 'max_features':
None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_
split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_f
raction_leaf': 0.0, 'presort': 'deprecated', 'random_state': None, 'splitt
er': 'best'}
5.722186186423075
```

7.1.3 Floresta Aleatória

In [763]:

```
from sklearn.ensemble import RandomForestRegressor

rand_forest_reg=RandomForestRegressor()

rand_forest_reg.fit(nba_prepared, nba_labels)
rand_forest_reg_rmse=np.sqrt(mean_squared_error(nba_labels, nba_predicted))

print(rand_forest_reg_rmse) # Overfitting?
print(rand_forest_reg.get_params())
```

```
5.722186186423075
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'mse', 'max_depth': Non
e, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'm
in_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_lea
f': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimat
ors': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verb
ose': 0, 'warm_start': False}
```

7.2 Validação Cruzada

7.2.1 Conjunto de Validação

In [764]:

```
from sklearn.model_selection import train_test_split

test_size=0.2 # tamanho do conjunto de teste: 20%
random_state=42
strat_train_set0, strat_test_set0=train_test_split(df,
                                                   test_size=test_size,
                                                   random_state=random_state,
                                                   stratify=df['EFICIENCIA_CAT'])

valid_size=0.25 # tamanho do conjunto de validação: 25% de 80% -> 20% de 100%
random_state=24
strat_train_set0, strat_valid_set0=train_test_split(strat_train_set0,
                                                   test_size=valid_size,
                                                   random_state=random_state,
                                                   stratify=strat_train_set0['EFICIENC
IA_CAT'])

strat_train_set0=strat_train_set0.drop('EFICIENCIA_CAT', axis=1)
strat_valid_set0=strat_valid_set0.drop('EFICIENCIA_CAT', axis=1)
strat_test_set0=strat_test_set0.drop('EFICIENCIA_CAT',axis=1)

print(strat_train_set0)
print(strat_valid_set0)
print(strat_test_set0)
```

```
                      TEAM              NAME  EXPERIENCE POSITION AGE  \
384           Atlanta Hawks    Alex Poythress           2       SF  25
372           Atlanta Hawks     Kent Bazemore           6       SG  29
254          Indiana Pacers  Domantas Sabonis           2       PF  22
444      Washington Wizards     Thomas Bryant           1        C  21
258          Indiana Pacers    Thaddeus Young          11       PF  30
..                      ...               ...         ...      ...  ..
519  Portland Trail Blazers       CJ McCollum           5       SG  27
214     Cleveland Cavaliers        Kevin Love          10       PF  30
418              Miami Heat       Dwyane Wade          15       SG  36
292        Dallas Mavericks  Dennis Smith Jr.           1       PG  20
61          New York Knicks       Noah Vonleh           4       PF  23

          HT      WT     SALARY  PPG_LAST_SEASON  APG_LAST_SEASON  ...    F
GP  \
384  205.74  106.33   1.544951              1.0              0.1  ...  0.4
50
372  195.58   90.95  18.089887             12.9              3.5  ...  0.4
22
254  210.82  108.60   2.659800             11.6              7.7  ...  0.4
66
444  210.82  112.22   1.378242              1.5              1.1  ...  0.3
81
258  203.20  100.00  13.964045             11.8              6.3  ...  0.4
99
..      ...     ...        ...              ...              ...  ...
...
519  190.50   85.97  25.759766             21.4              3.4  ...  0.4
54
214  208.28  113.57  24.119025             17.6              9.3  ...  0.4
44
418  193.04   99.55   1.512601             11.4              3.4  ...  0.4
83
292  190.50   88.24   3.819960             15.2              5.2  ...  0.3
95
61   205.74  113.12   1.512601              4.9              5.8  ...  0.4
44

     THM_THA    THP  FTM_FTA    FTP  APG  BLKPG STLPG  TOPG   PPG
384  0.3-1.0  0.333  0.3-0.3  0.800  0.2    0.1   0.2   0.2   2.9
372  1.0-2.7  0.360  1.3-1.9  0.718  1.8    0.4   1.0   1.4   8.2
254  0.4-1.3  0.327  1.4-1.9  0.729  1.5    0.4   0.5   1.4   8.6
444  0.1-0.7  0.100  0.3-0.6  0.556  0.4    0.1   0.1   0.1   1.5
258  0.4-1.3  0.326  1.4-2.1  0.675  1.6    0.4   1.5   1.4  13.4
..       ...    ...      ...    ...  ...    ...   ...   ...   ...
519  1.9-4.7  0.408  2.2-2.6  0.842  2.9    0.3   0.9   1.8  17.2
214  1.6-4.4  0.370  4.6-5.6  0.824  2.3    0.5   0.7   1.9  18.3
418  0.5-1.6  0.287  5.6-7.4  0.767  5.5    0.9   1.6   3.2  22.5
292  1.5-4.9  0.313  1.9-2.8  0.694  5.2    0.3   1.0   2.8  15.2
61   0.2-0.6  0.297  0.7-1.0  0.664  0.4    0.3   0.3   0.7   4.1

[228 rows x 28 columns]
                      TEAM              NAME  EXPERIENCE POSITION AGE  \
196           Chicago Bulls         Kris Dunn           2       PG  24
67       Philadelphia 76ers       Joel Embiid           2        C  24
42           New York Knicks         Ron Baker           2       SG  25
22             Brooklyn Nets      Allen Crabbe           5       SG  26
47           New York Knicks  Tim Hardaway Jr.           5        G  26
..                      ...               ...         ...      ...  ..
489  Minnesota Timberwolves       Jeff Teague           9       PG  30
32             Brooklyn Nets      Caris LeVert           2       SG  24
```

```
78       Philadelphia 76ers       Ben Simmons           1       PG  22
413              Miami Heat   Rodney McGruder           2       SG  27
119            LA Clippers  Patrick Beverley           6       PG  30

         HT      WT     SALARY  PPG_LAST_SEASON  APG_LAST_SEASON  ...    F
GP  \
196  193.04   92.76   4.221000             13.4              6.0  ...  0.4
13
67   213.36  113.12  25.467250             22.9             11.0  ...  0.4
78
42   193.04   99.55   4.544000              2.4              1.6  ...  0.3
70
22   198.12   95.93  18.500000             13.2              1.6  ...  0.4
39
47   198.12   92.76  17.325000             17.5              2.7  ...  0.4
25
..      ...     ...        ...              ...              ...  ...
...
489  187.96   88.24  19.000000             14.2              7.0  ...  0.4
47
32   200.66   92.31   1.702800             12.1              4.2  ...  0.4
40
78   208.28  104.07   6.434520             15.8              8.2  ...  0.5
45
413  193.04   90.50   1.544951              5.1              0.9  ...  0.4
24
119  185.42   83.71   5.027028             12.2              2.9  ...  0.4
13

     THM_THA    THP  FTM_FTA    FTP  APG  BLKPG STLPG  TOPG   PPG
196  0.5-1.6  0.309  0.9-1.3  0.694  3.8    0.5   1.4   1.8   7.6
67   1.1-3.3  0.327  5.9-7.6  0.774  2.8    2.0   0.7   3.7  22.0
42   0.4-1.5  0.286  0.6-0.9  0.696  1.9    0.2   0.7   0.9   3.5
22   1.6-4.0  0.397  1.1-1.3  0.849  1.2    0.3   0.6   0.7   9.5
47   1.7-5.0  0.344  1.7-2.2  0.806  1.7    0.2   0.6   1.0  12.1
..       ...    ...      ...    ...  ...    ...   ...   ...   ...
489  0.9-2.5  0.357  2.8-3.3  0.844  5.7    0.3   1.2   2.3  12.7
32   1.1-3.4  0.336  1.6-2.2  0.714  3.2    0.2   1.0   1.7  10.4
78   0.0-0.1  0.000  2.4-4.2  0.560  8.2    0.9   1.7   3.4  15.8
413  0.9-2.7  0.345  0.5-0.8  0.608  1.5    0.2   0.6   0.6   6.1
119  1.7-4.4  0.376  1.0-1.3  0.767  3.4    0.4   1.3   1.4   9.4

[77 rows x 28 columns]
                     TEAM             NAME  EXPERIENCE POSITION AGE
HT  \
537              Utah Jazz      Rudy Gobert           5        C  26  215.
90
198          Chicago Bulls   Justin Holiday           5       SG  29  198.
12
167           Phoenix Suns   Richaun Holmes           3       PF  24  208.
28
53         New York Knicks      Luke Kornet           1       PF  23  215.
90
545              Utah Jazz    Georges Niang           2       SF  25  203.
20
..                     ...              ...         ...      ...  ..
...
261        Milwaukee Bucks  Malcolm Brogdon           2       PG  25  195.
58
383          Atlanta Hawks    Miles Plumlee           6        C  30  210.
82
```

```
35           Brooklyn Nets   Shabazz Napier           4       PG  27  185.
42
500  Oklahoma City Thunder   Raymond Felton          13       PG  34  185.
42
11           Boston Celtics    Marcus Morris           7       PF  29  205.
74

         WT     SALARY  PPG_LAST_SEASON  APG_LAST_SEASON  ...    FGP  THM_
THA  \
537  110.86  23.241573             13.5             10.7  ...  0.614  0.0-
0.0
198   81.90   4.384616             12.2              2.1  ...  0.392  1.2-
3.5
167  106.33   1.600520              6.5              4.4  ...  0.547  0.3-
1.0
53   113.12   1.619260              6.7              3.2  ...  0.392  1.4-
4.0
545  104.07   1.512601              1.0              0.3  ...  0.277  0.0-
0.5
..      ...        ...              ...              ...  ...    ...
...
261  103.62   1.544951             13.0              3.2  ...  0.469  1.1-
2.9
383  112.67  12.500000              4.3              4.1  ...  0.537  0.0-
0.0
35    81.45   1.942422              8.7              2.0  ...  0.395  0.8-
2.3
500   92.76   1.512601              6.9              2.5  ...  0.412  1.0-
2.9
11   106.33   5.375000             13.6              5.4  ...  0.428  1.3-
3.6

       THP  FTM_FTA    FTP  APG  BLKPG STLPG  TOPG   PPG
537  0.000  2.7-4.3  0.629  1.2    2.2   0.6   1.6  10.0
198  0.350  1.0-1.2  0.811  1.4    0.4   0.8   0.9   7.4
167  0.257  1.0-1.5  0.687  1.0    0.8   0.5   0.7   7.4
53   0.354  0.4-0.6  0.727  1.3    0.8   0.3   0.6   6.7
545  0.063  0.1-0.1  0.750  0.3    0.0   0.2   0.3   0.9
..     ...      ...    ...  ...    ...   ...   ...   ...
261  0.395  1.5-1.7  0.872  3.8    0.2   1.0   1.5  11.3
383  0.000  0.6-1.0  0.544  0.5    0.8   0.4   0.9   4.9
35   0.363  1.0-1.2  0.802  1.9    0.1   0.8   1.1   5.7
500  0.329  1.8-2.3  0.789  5.4    0.2   1.2   2.1  11.4
11   0.357  1.6-2.1  0.731  1.5    0.2   0.7   1.2  11.2

[77 rows x 28 columns]
```

In [765]:

```
nba=strat_train_set.drop('SALARY', axis=1) # X
nba_labels=strat_train_set['SALARY'].copy() # y

nba_prepared=full_pipeline.transform(nba)
```

In [798]:

```
# Método criado para avaliar e mostrar a predição do modelo
def RegEvaluation(model, player, predict=True):
    #player_data = player_data.iloc[0:0]
    #sc_data

    #player_data = nba.iloc[0:0]
    #df_player

    player_data=nba[nba.NAME == player]
    #player_data

    player_data_prepared=full_pipeline.transform(player_data)
    player_data_predicted= round(model.predict(player_data_prepared)[0],6)
    df_player = df.loc[df.NAME==player]
    df_player['SALARIO_PREVISTO'] = player_data_predicted

#df_stephen = df.loc[df.NAME == 'Stephen Curry']
    if predict:
        return df_player[['NAME', 'SALARY', 'SALARIO_PREVISTO']]
```

7.2.2.1 Regressão Linear

In [767]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

lin_reg=LinearRegression()
scores=cross_val_score(lin_reg,
                       nba_prepared,
                       nba_labels,
                       scoring='neg_mean_squared_error', # cross_val_score expects a ut
ility function
                       cv=4)

lin_reg_rmse_score=np.sqrt(-scores)

print(lin_reg_rmse_score)
print(lin_reg_rmse_score.mean())
print(lin_reg_rmse_score.std())
##############################################
## Predição:
lin_reg.fit(nba_prepared, nba_labels)
#full_pipeline.transform(player_data)
RegEvaluation(lin_reg, 'Stephen Curry')
```

```
[1.37090750e+11 8.94337157e+11 6.37584419e+00 6.57184225e+00]
257856976737.6353
371709544680.20135

C:\Users\Thiago\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[767]:

	NAME	SALARY	SALARIO_PREVISTO
103	Stephen Curry	37.457154	25.464844

7.2.2.2 Árvore de Decisão

In [768]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score

dec_tree_reg=DecisionTreeRegressor()

scores=cross_val_score(dec_tree_reg,
                       nba_prepared,
                       nba_labels,
                       scoring='neg_mean_squared_error',
                       cv=4)

dec_tree_reg_rmse_score=np.sqrt(-scores)

print(dec_tree_reg_rmse_score)
print(dec_tree_reg_rmse_score.mean())
print(dec_tree_reg_rmse_score.std())

#########################
## Predição:
dec_tree_reg.fit(nba_prepared, nba_labels)
RegEvaluation(dec_tree_reg, 'Stephen Curry')
```

```
[6.63285863 8.17311036 8.37852431 7.78784313]
7.743084108790161
0.6751501353379792

C:\Users\Thiago\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[768]:

	NAME	SALARY	SALARIO_PREVISTO
103	Stephen Curry	37.457154	37.457154

7.2.2.3 Floresta Aleatória

In [769]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score

rand_forest_reg=RandomForestRegressor()

scores=cross_val_score(rand_forest_reg,
                       nba_prepared,
                       nba_labels,
                       scoring='neg_mean_squared_error',
                       cv=4)

rand_forest_reg_rmse_score=np.sqrt(-scores)

print(rand_forest_reg_rmse_score)
print(rand_forest_reg_rmse_score.mean())
print(rand_forest_reg_rmse_score.std()) # Bem melhor, mas ainda overfittando.

########################
## Predição:
rand_forest_reg.fit(nba_prepared,nba_labels)
RegEvaluation(rand_forest_reg, 'Stephen Curry')
```

```
[6.10254896 6.49826555 5.58744136 6.17003873]
6.089573651034548
0.3262701684963087

C:\Users\Thiago\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[769]:

	NAME	SALARY	SALARIO_PREVISTO
103	Stephen Curry	37.457154	33.171267

8. Ajuste do modelo

8.1 Grid Search

In [770]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

rand_forest_reg=RandomForestRegressor()

param_grid=[{'n_estimators':[3,10,30], 'max_features':[2,4,6,8]}, #3x4=12
            {'bootstrap':[False], 'n_estimators':[3,10], 'max_features':[2,3,4]}] # 1x2
x3=6

grid_search=GridSearchCV(rand_forest_reg,
                         param_grid,
                         scoring='neg_mean_squared_error',
                         refit=True,
                         cv=5,
                         return_train_score=True) # 12+6=18

grid_search.fit(nba_prepared, nba_labels) # 18*5=90 rounds de treino.

print(grid_search.best_params_)

############## Predição:
RegEvaluation(grid_search, 'Stephen Curry')
```

```
{'max_features': 8, 'n_estimators': 30}

C:\Users\Thiago\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[770]:

	NAME	SALARY	SALARIO_PREVISTO
103	Stephen Curry	37.457154	31.842928

In [771]:

```
rand_forest_reg=grid_search.best_estimator_ # melhor modelo treinado em todo o conjunto.

#print(rand_forest_reg.get_params())
rand_forest_reg.fit(nba_prepared, nba_labels)
RegEvaluation(rand_forest_reg, 'Stephen Curry')
```

```
C:\Users\Thiago\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[771]:

	NAME	SALARY	SALARIO_PREVISTO
103	Stephen Curry	37.457154	32.710702

In [772]:

```
d=grid_search.cv_results_

for mean_train_score, mean_test_score, params in zip(d['mean_train_score'], d['mean_tes
t_score'],d['params']):
    print(np.sqrt(-mean_train_score), np.sqrt(-mean_test_score), params)
```

```
3.698578677256702 7.743906023598103 {'max_features': 2, 'n_estimators': 3}
2.966646468018949 6.979407491493342 {'max_features': 2, 'n_estimators': 1
0}
2.550081889208129 6.528543790935954 {'max_features': 2, 'n_estimators': 3
0}
3.5986670682946387 7.295956508293055 {'max_features': 4, 'n_estimators':
3}
2.727464141831927 6.728324229940121 {'max_features': 4, 'n_estimators': 1
0}
2.4742108095502635 6.40596043319244 {'max_features': 4, 'n_estimators': 3
0}
3.5269428558489277 7.44668702933789 {'max_features': 6, 'n_estimators': 3}
2.7718120358177316 6.777053444041929 {'max_features': 6, 'n_estimators': 1
0}
2.4345175318231322 6.430832255342448 {'max_features': 6, 'n_estimators': 3
0}
3.6463785341470953 7.5371374364379164 {'max_features': 8, 'n_estimators':
3}
2.7853437447962803 6.4986533143240335 {'max_features': 8, 'n_estimators':
10}
2.4369536491321897 6.270791588702372 {'max_features': 8, 'n_estimators': 3
0}
9.543305649517192e-09 7.461379493462362 {'bootstrap': False, 'max_feature
s': 2, 'n_estimators': 3}
1.6405466901551136e-07 6.966147044774636 {'bootstrap': False, 'max_feature
s': 2, 'n_estimators': 10}
7.730666583738068e-07 7.769642141520165 {'bootstrap': False, 'max_feature
s': 3, 'n_estimators': 3}
4.0173972760727997e-07 6.832299760125102 {'bootstrap': False, 'max_feature
s': 3, 'n_estimators': 10}
5.467656181278783e-07 7.143858379778542 {'bootstrap': False, 'max_feature
s': 4, 'n_estimators': 3}
2.3227315745184477e-07 6.708908140642286 {'bootstrap': False, 'max_feature
s': 4, 'n_estimators': 10}
```

8.2 Randomized Search

In [773]:

```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

rand_forest_reg=RandomForestRegressor()

param_grid=[{'n_estimators': np.arange(100), 'max_features': np.arange(1,10)},
            {'bootstrap':[False], 'n_estimators': np.arange(50), 'max_features': np.ara
nge(1,10)}] # 1x50x9=450

randomized_search=RandomizedSearchCV(rand_forest_reg,
                                     param_grid,
                                     n_iter=10, # Numero de iterações
                                     scoring='neg_mean_squared_error',
                                     refit=True,
                                     cv=10,
                                     random_state=42,
                                     return_train_score=True) #900+450=1350

randomized_search.fit(nba_prepared, nba_labels) #1350*10=13500 rodadas de treinamento.

print(randomized_search.best_params_)
RegEvaluation(randomized_search, 'Stephen Curry')
```

```
{'n_estimators': 66, 'max_features': 5}

C:\Users\Thiago\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[773]:

	NAME	SALARY	SALARIO_PREVISTO
103	Stephen Curry	37.457154	33.473709

In [779]:

```
rand_forest_reg=randomized_search.best_estimator_ # melhor modelo treinado em todo o co
njunto de treinamento

#print(rand_forest_reg.get_params())
rand_forest_reg.fit(nba_prepared, nba_labels)
df_predict = RegEvaluation(randomized_search, 'Stephen Curry')
df_predict
```

```
C:\Users\Thiago\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[779]:

	NAME	SALARY	SALARIO_PREVISTO
103	Stephen Curry	37.457154	31.990118

In [775]:

```
d=randomized_search.cv_results_

for mean_train_score, mean_test_score, params in zip(d['mean_train_score'],d['mean_test
_score'], d['params']):
    print(np.sqrt(-mean_train_score), np.sqrt(-mean_test_score), params)
```

```
4.121519440739094e-07 6.4536623390772885 {'n_estimators': 26, 'max_feature
s': 5, 'bootstrap': False}
2.3680797796284114 6.241351009624532 {'n_estimators': 60, 'max_features':
9}
3.5144361563631266e-07 6.295403583709484 {'n_estimators': 44, 'max_feature
s': 8, 'bootstrap': False}
2.422171173928614e-07 6.380736709400406 {'n_estimators': 30, 'max_feature
s': 5, 'bootstrap': False}
2.1780977958488306e-07 6.4300794022093415 {'n_estimators': 45, 'max_featur
es': 4, 'bootstrap': False}
1.9397949910375987e-07 6.272882363213412 {'n_estimators': 44, 'max_feature
s': 3, 'bootstrap': False}
2.586619436716584 6.506025482708783 {'n_estimators': 21, 'max_features':
2}
2.3664453556713965 6.140592292811336 {'n_estimators': 66, 'max_features':
5}
3.436579280654067e-07 6.464522977071286 {'n_estimators': 38, 'max_feature
s': 7, 'bootstrap': False}
2.5042745453342334 6.4416413329084605 {'n_estimators': 30, 'max_features':
4}
```

9. Salário do Stephen Curry é justo?

In [796]:

```
# Buscando salário real do Stephen Curry
salario_real = df[df.NAME == 'Stephen Curry'].SALARY
print(salario_real)
```

```
103    37.457154
Name: SALARY, dtype: float64
```

In [797]:

```
# Salário previsto pelo modelo
print(df_predict.SALARIO_PREVISTO)
```

```
103    31.990118
Name: SALARIO_PREVISTO, dtype: float64
```

In [790]:

```
# Diferença de 5.467036
dif = salario_real - df_predict.SALARIO_PREVISTO
dif
```

Out[790]:

```
103    5.467036
dtype: float64
```

In [791]:

```
# Diferença de 15% na previsão
difPercent = round(dif/salario_real * 100,00)
difPercent
```

Out[791]:

```
103    15.0
dtype: float64
```

10. Conclusão

Foi selecionado o modelo de **Floresta Aleatória** com o melhor estimador da busca randomizada.

Resultado da análise do SAM: É possível deduzir que o salário de Stephen Curry está acima do que seria o valor justo, de acordo com o que foi previsto no modelo.