

UNIVERSIDADE DE FORTALEZA

MBA em Ciência de Dados

Prof. PhD Ernerson A. Oliveira

Discente:Sabrina Rodrigues de Sousa

Matrícula:2129475

Trabalho de Introdução ao Aprendizado de Máquina

1. Introdução

Para praticar o que foi visto em sala, este trabalho visa desenvolver um modelo de Machine Learning capaz de fazer previsões. O dataset utilizado nesse projeto são de jogadores da NBA, cujas informações contidas são referentes a seus rendimentos em quadra. Assim, o desafio é chegar num modelo capaz de prever o salário de um jogador Stepen Curry, com base em certas métricas, e posteriormente poder verificar se o salário é justo em relação aos demais.

2. Dados e ferramentas

Os dados usados são dos jogadores da NBA(National Basketball Association) ESPN antes do início da temporada 2018-2019 e foram disponibilizados pela plataforma Kaggle¹ f. Os arquivos, que estão no formato csv,contemplam as métricas dos jogadores.

Para realizar a análise dos dados serão utilizados os módulos pandas, numpy , matplotlib e sklearn Além de leitura de um arquivo em csv dos dados.

2.1 Sobre os arquivos utilizados

- NBA_DF: -(NBA_Players.csv): Dataframe com vários dados de desempenho dos jogadores.

É importante antes de iniciarmos o projeto, explanar o que cada coluna representa, esse essa análise inicial é de extrema importância para o entendimento e escolhas de metodologias de higienização e aplicação do machine learn. Segue o significado de algumas das colunas:

EXPERIENCE:Experiência

AGE: Idade

HT: Altura

WT: Peso

SALARY: Salário

PPG_LAST_SEASON: Pontos por jogo (última temporada)

APG_LAST_SEASON: Assistências por jogo (última temporada)

RPG_LAST_SEASON: Rebotes por jogo (última temporada)

PER_LAST_SEASON: Classificação de eficiência do jogador

PPG_CAREER: Pontos por jogo na carreira

APG_CAREER: Rebotes por jogo na carreira

RGP_CAREER: Classificação de eficiência do jogador na carreira

GP: Jogos jogados

MPG: Minutos jogados por jogo

FGP: Arremessos convertidos

FTP: Porcentagem de lances livres

STLPG = Roubadas de bola por jogo

TOPG = Bola desperdiçada por jogo

3 Metodologia de análise

Os primeiros passos, após a importação dos módulos e carga dos dados, será analisar todo o DataFrame e fazer uma higienização nos valores que não têm significância para as análises. Será verificado, os dados nulos, valores em branco e/ou zerados, os tipos de dados e se há informações repetidas.

Em seguida, será analisada as correlações existentes do "SALARIO" (variável alvo) com os demais atributos através de gráficos e números apresentados a partir do módulo SEABORN.

Posteriormente as 5 melhores correlações vão ser selecionadas e seus dados serão divididos em conjuntos de treino e teste. Estes serão aplicados em dois modelos: o de regressão linear e o de árvore de decisão com regressão, a fim de verificar qual modelo que mais se adequa ao objetivo desejado no projeto. Os dois modelos foram selecionados por se caracterizarem pela necessidade de prever um único dado ou valor. A escolha do modelo será baseada através do melhor score entre os dados de teste e treino de cada um deles.

Por fim, com o modelo de previsão selecionado, irá aplicar as métricas do jogador Stephen Curry para verificar a previsão do seu salário e compará-lo ao real, podendo responder a nossa pergunta principal.

Os processos serão detalhados durante a execução do notebook.

4 Resultados e Discussões

4.1 Importação de dados e módulos

4.1.1 Importando Módulos

```
In [1]: import pandas as pd
import seaborn as sb
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import numpy as np
import matplotlib.pyplot as plt

# import the regressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
%matplotlib inline
```

4.1.2 Importando os dados

Os dados em formato de csv são importados em um data frame

```
In [2]: NBA_df=pd.read_csv(r'NBA_Players.csv')
```

4.2 identificação dos dados

Nesta etapa, analisou os dados extraídos da base, quantidade de linha, suas colunas e seu tipo, com o objetivo de definir quais dados seriam explorados no trabalho.

4.2.1 Analise do arquivo

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

Out[3]:

	TEAM	NAME	EXPERIENCE	URL	POSITION	AGE	HT
0	Boston Celtics	Aron Baynes	6	http://www.espn.com/nba/player/_id/2968439	SF	31	208.28
1	Boston Celtics	Justin Bibbs	0	http://www.espn.com/nba/player/_id/3147500	G	22	195.58
2	Boston Celtics	Jabari Bird	1	http://www.espn.com/nba/player/_id/3064308	SG	24	198.12
3	Boston Celtics	Jaylen Brown	2	http://www.espn.com/nba/player/_id/3917376	F	21	200.66
4	Boston Celtics	PJ Dozier	1	http://www.espn.com/nba/player/_id/3923250	PG	21	198.12

In [4]: `NBA_df.info()`

#	Column	Non-Null Count	Dtype
0	TEAM	550 non-null	object
1	NAME	550 non-null	object
2	EXPERIENCE	550 non-null	int64
3	URL	550 non-null	object
4	POSITION	550 non-null	object
5	AGE	550 non-null	object
6	HT	550 non-null	float64
7	WT	550 non-null	float64
8	COLLEGE	550 non-null	object
9	SALARY	550 non-null	object
10	PPG_LAST_SEASON	538 non-null	float64
11	APG_LAST_SEASON	538 non-null	float64
12	RPG_LAST_SEASON	538 non-null	float64
13	PER_LAST_SEASON	538 non-null	float64
14	PPG_CAREER	550 non-null	float64
15	APG_CAREER	550 non-null	float64
16	RPG_CAREER	550 non-null	float64
17	GP	550 non-null	int64

In [5]: NBA_df.describe()

Out[5]:

	EXPERIENCE	HT	WT	PPG_LAST_SEASON	APG_LAST_SEASON	RPG_LAST_SEASON
count	550.000000	550.000000	550.000000	538.000000	538.000000	538.000000
mean	4.018182	200.512218	98.371255	7.316171	2.667472	2.809521
std	4.144876	8.592139	10.883866	6.625138	2.809521	2.809521
min	0.000000	175.260000	76.920000	0.000000	0.000000	0.000000
25%	1.000000	193.675000	90.500000	1.000000	0.200000	0.200000
50%	3.000000	200.660000	97.290000	6.150000	1.900000	1.900000
75%	6.000000	208.280000	106.330000	11.700000	4.200000	4.200000
max	20.000000	220.980000	131.220000	30.400000	16.000000	16.000000

In [6]: NBA_df.columns

Out[6]: Index(['TEAM', 'NAME', 'EXPERIENCE', 'URL', 'POSITION', 'AGE', 'HT',
'WT', 'COLLEGE', 'SALARY', 'PPG_LAST_SEASON', 'APG_LAST_SEASON',
'RPG_LAST_SEASON', 'PER_LAST_SEASON', 'PPG_CAREER', 'APG_CAREER',
'RPG_CAREER', 'GP', 'MPG', 'FGM_FGA', 'FGP', 'THM_THA', 'THP',
'FTM_FTA', 'FTP', 'APG', 'BLKPG', 'STLPG', 'TOPG', 'PPG'],
dtype='object')

```
In [7]: NBA_df.isnull().sum()
```

```
Out[7]: TEAM          0  
NAME          0  
EXPERIENCE    0  
URL           0  
POSITION       0  
AGE            0  
HT             0  
WT             0  
COLLEGE        0  
SALARY         0  
PPG_LAST_SEASON 12  
APG_LAST_SEASON 12  
RPG_LAST_SEASON 12  
PER_LAST_SEASON 12  
PPG_CAREER     0  
APG_CAREER     0  
RGP_CAREER     0  
GP              0  
MPG             0  
FGM_FGA         0  
FGP             0  
THM_THA         0  
THP             0  
FTM_FTA         0  
FTP             0  
APG             0  
BLKPG           0  
STLPG           0  
TOPG            0  
PPG             0  
dtype: int64
```

4.2.2 Definindo dados para estudo

Como é possível verificar nas análises acima, há dados com tipo objeto e dados tipo float. Para nosso modelo, precisamos alinhar o dado para somente um formato, que será o float. Dentre os objetos, somente a coluna AGE e SALARY será usada para o estudo, pois apresentam o valor número. TEAM, NAME, URL e POSITION serão desconsiderados.

Outros dados a serem observados são as colunas de FGM_FGA, THM_THA, FTM_FTA. Eles demonstram uma relação tentativa e acerto, por exemplo FGM_FGA são numero de lances livres convertidos em relação as tentativas. Essa informação é dada de forma isolada com FGP. Dessa forma, irá ser retirado as "colunas relação" e mantidas as FGP, THP e FTP.

4.2.2.1 Manuseando e selecionando colunas do dataframe

In [8]: *#Removendo espaços da descrição das colunas*

```
df = NBA_df
df.columns= NBA_df.columns.str.replace(' ', '')
df.columns
```

Out[8]: Index(['TEAM', 'NAME', 'EXPERIENCE', 'URL', 'POSITION', 'AGE', 'HT', 'WT',
 'COLLEGE', 'SALARY', 'PPG_LAST_SEASON', 'APG_LAST_SEASON',
 'RPG_LAST_SEASON', 'PER_LAST_SEASON', 'PPG_CAREER', 'APG_CAREER',
 'RGP_CAREER', 'GP', 'MPG', 'FGM_FGA', 'FGP', 'THM_THA', 'THP',
 'FTM_FTA', 'FTP', 'APG', 'BLKPG', 'STLPG', 'TOPG', 'PPG'],
 dtype='object')

Como iremos retirar os dados como nome do dataframe, vamos separar os dados do jogador Curry para posteriormente comparar com a previsão.

In [9]: *#informações de Stephen Curry para prever seu salário*

```
stephen = NBA_df[NBA_df.NAME == 'Stephen Curry']
stephen = stephen.loc[:, :]
stephen
```

Out[9]:

	TEAM	NAME	EXPERIENCE	URL	POSITION	AGE	HT
103	Golden State Warriors	Stephen Curry	9	http://www.espn.com/nba/player/_/id/3975	PG	30	190.5

Nesta etapa vamos excluir as colunas que entrarão para o modelo

In [10]: *#Remover dados que não serão utilizados*

```
df.drop(columns=["THM_THA", "FTM_FTA", "FGM_FGA", "URL", "TEAM", "PPG", "COLLEGE"]
df.columns
```

Out[10]: Index(['EXPERIENCE', 'AGE', 'HT', 'WT', 'SALARY', 'PPG_LAST_SEASON',
 'APG_LAST_SEASON', 'RPG_LAST_SEASON', 'PER_LAST_SEASON', 'PPG_CAREER',
 'APG_CAREER', 'RGP_CAREER', 'GP', 'MPG', 'FGP', 'THP', 'FTP', 'APG',
 'BLKPG', 'STLPG', 'TOPG'],
 dtype='object')

4.3 Higienização do DataFrame

Para uma boa aplicação do modelo, temos que garantir que dentre os dados que antes estão em formato objeto contém alguma informação que não seja número.

```
In [11]: idade = df['AGE'].unique()
salario = df['SALARY'].unique()
print(idade, '\n', '\n', salario)

['31' '22' '24' '21' '28' '32' '26' '23' '29' '20' '25' '33' '27' '19' '-'
 '30' '34' '18' '36' '37' '35' '40' '38' '41']

[5,193,600 'Not signed' 1,349,464 5,169,960 31,214,295
 28,928,709 20,099,189 5,375,000 1,378,242 3,050,390
 11,660,716 6,700,800 838,464 2,667,600 2,034,120 15,400,000
 18,500,000 4,449,000 1,656,092 9,530,000 13,764,045 1,512,601
 8,000,000 2,470,357 1,618,320 1,702,800 1,632,240 1,942,422
 7,019,698 5,000,000 4,544,000 1,795,015 17,325,000 6,500,000
 18,622,514 3,739,920 1,619,260 12,253,780 4,294,480 4,155,720
 5,697,054 1,485,440 7,119,650 8,575,916 12,800,562 10,464,092
 25,467,250 8,339,880 1,740,000 1,600,520 12,250,000 2,526,840
 1,703,649 6,434,520 10,000,000 21,666,667 23,114,067
 31,200,000 8,333,333 1,826,300 9,367,200 1,569,360 1,544,951
 16,539,326 8,653,847 2,536,898 5,337,000 37,457,154
 30,000,000 1,644,240 17,469,565 16,000,000 8,307,692
 18,988,725 5,027,028 12,000,000 21,587,579 3,375,360
 13,565,218 6,000,000 14,800,000 6,134,520 7,000,000 4,320,500
 3,046,200 6,300,000 1,349,383 7,461,960 3,500,000 1,000,000
 1,000,000 1,000,000 1,000,000 1,000,000 1,000,000 1,000,000]
```

A coluna AGE e SALARY apresentam dados não numéricos, estes serão substituídos por vazios e posteriormente suas linhas vazias serão retiradas.

In [12]:

```
df= df.replace('-',np.Nan)
df= df.replace('Not signed',np.Nan)
idade = df['AGE'].unique()
salario = df['SALARY'].unique()
print(idade, '\n', salario)
```

```
['31' '22' '24' '21' '28' '32' '26' '23' '29' '20' '25' '33' '27' '19' nan
 '30' '34' '18' '36' '37' '35' '40' '38' '41']

[5,193,600' nan '1,349,464' '5,169,960' '31,214,295' '28,928,709'
 '20,099,189' '5,375,000' '1,378,242' '3,050,390' '11,660,716' '6,700,800'
 '838,464' '2,667,600' '2,034,120' '15,400,000' '18,500,000' '4,449,000'
 '1,656,092' '9,530,000' '13,764,045' '1,512,601' '8,000,000' '2,470,357'
 '1,618,320' '1,702,800' '1,632,240' '1,942,422' '7,019,698' '5,000,000'
 '4,544,000' '1,795,015' '17,325,000' '6,500,000' '18,622,514' '3,739,920'
 '1,619,260' '12,253,780' '4,294,480' '4,155,720' '5,697,054' '1,485,440'
 '7,119,650' '8,575,916' '12,800,562' '10,464,092' '25,467,250'
 '8,339,880' '1,740,000' '1,600,520' '12,250,000' '2,526,840' '1,703,649'
 '6,434,520' '10,000,000' '21,666,667' '23,114,067' '31,200,000'
 '8,333,333' '1,826,300' '9,367,200' '1,569,360' '1,544,951' '16,539,326'
 '8,653,847' '2,536,898' '5,337,000' '37,457,154' '30,000,000' '1,644,240'
 '17,469,565' '16,000,000' '8,307,692' '18,988,725' '5,027,028'
 '12,000,000' '21,587,579' '3,375,360' '13,565,218' '6,000,000'
 '14,800,000' '6,134,520' '7,000,000' '4,320,500' '3,046,200' '6,300,000'
 '1,349,383' '7,461,960' '3,500,000' '1,000,000' '1,655,160' '5,757,120'
 '35,654,150' '1,689,840' '1,487,694' '9,000,000' '1,762,080' '20,421,546'
 '15,000,000' '7,464,912' '8,165,160' '4,661,280' '3,314,365' '3,552,960'
 '13,585,000' '3,258,539' '6,041,520' '949,000' '1,238,464' '11,750,000'
 '7,305,600' '4,696,875' '3,000,000' '5,470,920' '2,207,040' '3,844,760'
 '2,807,880' '8,739,500' '5,460,000' '11,692,308' '11,011,234'
 '11,286,516' '4,441,200' '4,221,000' '8,740,980' '4,384,616' '1,990,520'
 '19,500,000' '14,357,750' '4,536,120' '20,000,000' '3,263,294'
 '2,494,346' '2,280,600' '12,500,000' '2,760,095' '19,000,000' '3,472,887'
 '7,560,000' '24,119,025' '2,272,391' '2,775,000' '4,068,600' '14,720,000'
 '1,952,760' '2,500,000' '25,434,263' '1,857,480' '32,088,932'
 '17,043,478' '3,940,402' '3,275,280' '10,002,681' '4,075,000'
 '10,500,000' '12,400,000' '1,911,960' '7,945,000' '2,407,560' '7,333,333'
 '21,000,000' '2,659,800' '3,410,284' '13,964,045' '24,157,303'
 '1,641,000' '9,607,500' '2,481,000' '11,327,466' '3,382,000' '2,799,720'
 '13,000,000' '10,607,143' '2,534,280' '3,710,850' '24,107,258'
 '1,230,000' '6,560,640' '22,897,200' '9,631,250' '3,819,960' '15,293,104'
 '3,206,160' '1,621,415' '13,500,375' '35,650,150' '3,651,480'
 '14,631,250' '7,969,537' '8,641,000' '30,521,115' '7,666,667' '5,915,040'
 '5,285,394' '12,252,928' '25,976,111' '2,205,000' '8,808,685' '1,567,707'
 '22,347,015' '6,153,846' '2,487,000' '27,739,975' '3,125,000'
 '16,800,000' '10,087,200' '11,571,429' '2,947,320' '2,357,160'
 '1,667,160' '2,516,048' '18,089,887' '1,634,640' '2,299,080' '7,200,000'
 '2,250,960' '4,350,000' '13,766,421' '1,620,480' '5,356,440' '24,000,000'
 '17,000,000' '3,206,640' '988,464' '3,627,842' '7,488,372' '3,447,480'
 '14,087,500' '13,528,090' '2,955,840' '18,109,175' '6,270,000'
 '14,651,700' '19,245,370' '12,537,527' '11,550,000' '25,434,262'
 '3,448,926' '7,250,000' '4,865,040' '1,050,000' '21,590,909' '2,639,314'
 '4,969,080' '2,416,222' '12,750,000' '2,749,080' '15,944,154' '3,454,500'
 '8,600,000' '3,208,630' '26,011,913' '12,650,000' '3,129,187' '5,450,000'
 '19,169,800' '11,830,358' '1,773,840' '2,000,000' '16,517,857'
 '2,166,360' '24,605,181' '1,874,640' '3,364,249' '29,230,769' '3,499,800'
 '12,917,808' '2,894,160' '20,445,779' '15,170,787' '14,000,000'
```

```
'2,444,053' '2,160,746' '4,750,000' '7,839,435' '5,455,236' '2,118,840'
'30,560,700' '1,757,429' '5,451,600' '15,500,000' '6,957,105' '3,628,920'
'2,795,000' '10,837,079' '10,595,506' '27,977,689' '25,759,766'
'11,111,111' '1,760,520' '17,868,853' '2,074,320' '1,679,520'
'11,536,515' '7,305,825' '9,600,000' '16,900,000' '23,241,573'
'13,045,455' '3,111,480' '2,150,000' '14,975,000' '5,250,000' '3,360,000']
```

In [13]: #Definindo todos os dados em formato Float
df['SALARY']= df['SALARY'].str.replace(',', '')
df = df.astype(float)
df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   EXPERIENCE       550 non-null    float64
 1   AGE              548 non-null    float64
 2   HT               550 non-null    float64
 3   WT               550 non-null    float64
 4   SALARY           440 non-null    float64
 5   PPG_LAST_SEASON 538 non-null    float64
 6   APG_LAST_SEASON 538 non-null    float64
 7   RPG_LAST_SEASON 538 non-null    float64
 8   PER_LAST_SEASON 538 non-null    float64
 9   PPG_CAREER       550 non-null    float64
 10  APG_CAREER      550 non-null    float64
 11  RGP_CAREER      550 non-null    float64
 12  GP               550 non-null    float64
 13  MPG              550 non-null    float64
 14  FGP              550 non-null    float64
 15  THP              550 non-null    float64
 16  FTP              550 non-null    float64
 17  APG              550 non-null    float64
 18  BLKPG            550 non-null    float64
 19  STLPG            550 non-null    float64
 20  TOPG             550 non-null    float64
dtypes: float64(21)
memory usage: 90.4 KB
```

In [14]: #Excluindo Linhas que contém dados nulos, que foram identificados nas colunas #(PPG_LAST_SEASON,RPG_LAST_SEASON,RPG_LAST_SEASON, PER_LAST_SEASON)

```
df.dropna(inplace = True)
df.isnull().sum()
```

Out[14]:

EXPERIENCE	0
AGE	0
HT	0
WT	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
RGP_CAREER	0
GP	0
MPG	0
FGP	0
THP	0
FTP	0
APG	0
BLKPG	0
STLPG	0
TOPG	0

dtype: int64

In [15]: #Constatando que os dados estão no formato correto

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 436 entries, 0 to 549
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   EXPERIENCE      436 non-null    float64
 1   AGE              436 non-null    float64
 2   HT               436 non-null    float64
 3   WT               436 non-null    float64
 4   SALARY           436 non-null    float64
 5   PPG_LAST_SEASON 436 non-null    float64
 6   APG_LAST_SEASON 436 non-null    float64
 7   RPG_LAST_SEASON 436 non-null    float64
 8   PER_LAST_SEASON 436 non-null    float64
 9   PPG_CAREER      436 non-null    float64
 10  APG_CAREER     436 non-null    float64
 11  RGP_CAREER     436 non-null    float64
 12  GP               436 non-null    float64
 13  MPG              436 non-null    float64
 14  FGP              436 non-null    float64
 15  THP              436 non-null    float64
 16  FTP              436 non-null    float64
 17  APG              436 non-null    float64
 18  BLKPG            436 non-null    float64
 19  STLPG            436 non-null    float64
 20  TOPG             436 non-null    float64
dtypes: float64(21)
memory usage: 74.9 KB
```

In [16]: df.min()

Out[16]:

EXPERIENCE	0.00
AGE	18.00
HT	175.26
WT	76.92
SALARY	838464.00
PPG_LAST_SEASON	0.00
APG_LAST_SEASON	0.00
RPG_LAST_SEASON	0.00
PER_LAST_SEASON	-3.19
PPG_CAREER	0.00
APG_CAREER	0.00
RPG_CAREER	0.00
GP	0.00
MPG	0.00
FGP	0.00
THP	0.00
FTP	0.00
APG	0.00
BLKPG	0.00
STLPG	0.00
TOPG	0.00

dtype: float64

In [17]: df.max()

Out[17]:

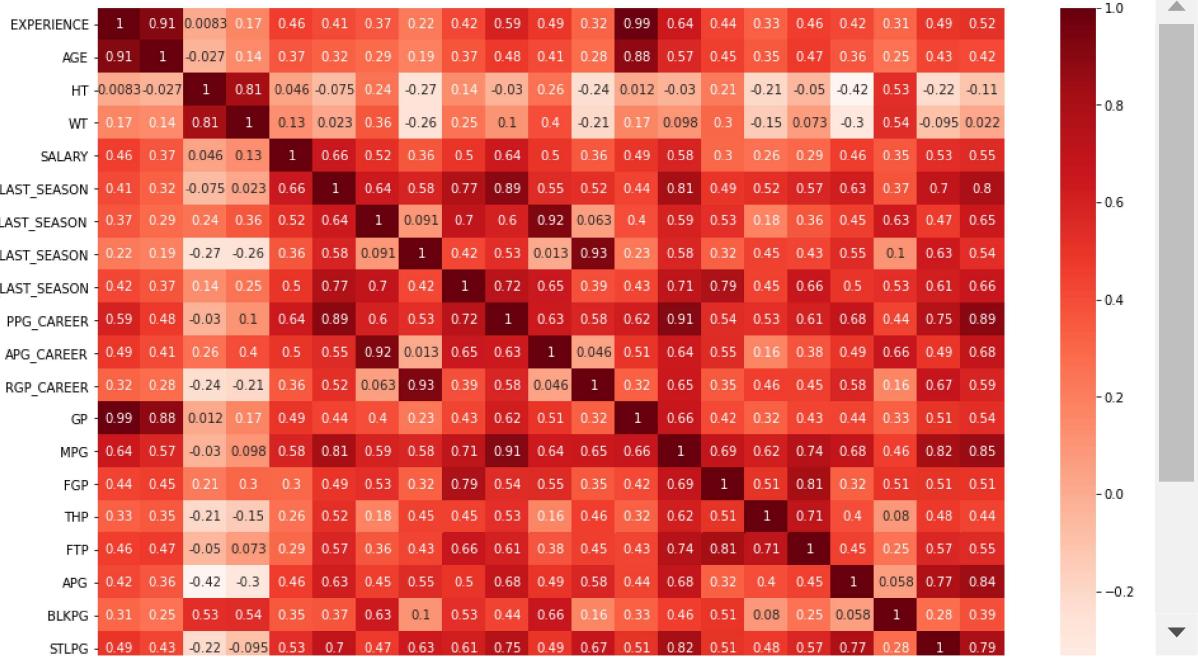
WT	1.312200e+02
SALARY	3.745715e+07
PPG_LAST_SEASON	3.040000e+01
APG_LAST_SEASON	1.600000e+01
RPG_LAST_SEASON	1.230000e+01
PER_LAST_SEASON	2.987000e+01
PPG_CAREER	2.720000e+01
APG_CAREER	1.340000e+01
RPG_CAREER	1.170000e+01
GP	1.471000e+03
MPG	3.880000e+01
FGP	7.310000e-01
THP	1.000000e+00
FTP	1.000000e+00
APG	9.800000e+00
BLKPG	2.400000e+00
STLPG	2.300000e+00
TOPG	4.000000e+00

dtype: float64

Feita a limpeza e ficaremos com 436 linhas e 21 colunas

4.4 Analisando a correlação dos dados

```
In [18]: plt.figure(figsize=(16,10))
correlacao = df.corr()
sb.heatmap(correlacao, annot=True, cmap=plt.cm.Reds )
plt.show()
```



Pode-se observar que as colunas não demonstram uma alta correlação com o salário, mas entre as colunas que contém mais correlação se destacam: Pontos por jogo (última temporada), Pontos por jogo (carreira), Minutos jogados por jogo, Roubadas de bola por jogo e Bolaz desperdiçadas por jogo.

Para o caso, vamos utilizar as 5 maiores correlações com o salário.

4.5 Dividindo os dados em treino e teste

Para fazermos as análises no futuro, precisaremos dividir nossa base em variáveis dependentes e independentes (X e y) com os seguintes comandos.

```
In [19]: y = df['SALARY']
X = df[['PPG_LAST_SEASON', 'PPG_CAREER', 'MPG', 'TOPG', 'STLPG']]
```

Agora, precisamos fazer algo importante para o resultado final de qualquer processo de aprendizagem de máquina: ter dados para treinar nossa máquina (para que a máquina possa aprender sobre nossos dados) e ter dados para testar (para verificar se o modelo está adequado a base). Para este projeto, que contém poucos dados, utilizaremos o formato 80% para treino e 20% para testes. Assim:

```
In [20]: semente = 42
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
```

Nesse momento poderia ser utilizado da normalização dos dados, mas como o dataset não tem dados tão discrepantes entre eles, optou-se por não aplica-lo.

4.6 Aplicando os modelos

No projeto, vamos testar dois modelos para verificar qual deles se adequa mais aos dados e nos auxilie a responder a questão principal. O primeiro será o de regressão linear, e o segundo será o de árvore de decisão. Para definição da escolha, usaremos a função .score para avaliar a precisão de cada um.

4.6.1 Regressão Linear

```
In [21]: modelo = linear_model.LinearRegression()
```

```
In [22]: modelo.fit(X_treino,y_treino)
```

```
Out[22]: LinearRegression()
```

```
In [23]: previsao = modelo.predict(X_teste)
```

```
In [24]: #Calculando R2  
r2_score(y_teste,previsao)
```

```
Out[24]: 0.5062617294839166
```

```
In [25]: # Calculando score do treino  
modelo.score(X_treino, y_treino)
```

```
Out[25]: 0.41894311960105834
```

```
In [26]: #Calculando score do teste  
modelo.score(X_teste, y_teste)
```

```
Out[26]: 0.5062617294839166
```

Podemos ver que apesar de apresentar um score mediando, o resultado com os dados de treino e teste são parecidos. Vamos para o outro modelo.

4.6.3 Árvore de decisão

```
In [27]: modelo2 = DecisionTreeRegressor(random_state = semente )  
  
modelo2.fit(X_treino, y_treino)
```

```
Out[27]: DecisionTreeRegressor(random_state=42)
```

```
In [28]: previsao2= modelo2.predict(X_teste)
```

```
In [29]: r2_score(y_teste,previsao2)
```

```
Out[29]: 0.15908657831731554
```

```
In [30]: modelo2.score(X_treino, y_treino)
```

```
Out[30]: 0.9934265304991785
```

```
In [31]: modelo2.score(X_teste, y_teste)
```

```
Out[31]: 0.15908657831731554
```

Como pode-se observar o score da árvore de decisão é absurdamente alta no conjunto de treino, enquanto no de teste ele está bem baixo. Quando isso acontece, mostra que está acontecendo um problema de Overfitting — quando o modelo “adivinha” muito bem os dados que foram usados para treiná-lo, mas ele não consegue se sair muito bem com dados que nunca viu.

Assim, o melhor modelo entre os dois é o de regressão linear.

4.7 Respondendo a pergunta principal

Com o modelo escolhido, utilizaremos para verificar qual seria o valor previsão do salário do Stephen Curry e comparar com o real salário que ele recebeu na temporada.

```
In [32]: #Salário Real
stephen['SALARY']= stephen['SALARY'].str.replace(',', '')
salario = stephen.SALARY.astype(float)
salario
```

```
Out[32]: 103    37457154.0
Name: SALARY, dtype: float64
```

Agora vamos separar as features do Stephen Curry e posteriormente prever seu salário

```
In [33]: # Separando features de Stephen Curry para prever seu salário
stephen
stephen_x = stephen[['PPG_LAST_SEASON','PPG_CAREER','MPG','TOPG','STLPG']]
stephen_x
```

```
Out[33]:
```

	PPG_LAST_SEASON	PPG_CAREER	MPG	TOPG	STLPG
103	26.4	23.1	34.4	3.2	1.8

In [34]: # Prevendo salário

```
previsao = round(modelo.predict(stephen_x)[0],0)
previsao
```

Out[34]: 22431843.0

In [35]: # Erro de \$ 15.025.311

```
ERRO = previsao - salario
ERRO
```

Out[35]: 103 -15025311.0
Name: SALARY, dtype: float64

5 Conclusão

O modelo ainda não apresenta números precisos em suas previsões. Contudo, com esta análise é possível deduzir que Stephen Curry recebe um valor acima do que seria ideal. Por fim, é preciso continuar o trabalho no modelo, aplicando um treinamento maior, analisando outras correlações, como a posição e o time que joga, por exemplo, até atingir um erro menor. Sendo possível ter informações mais precisas

A partir desses dados e do presente trabalho, foi possível observar que mesmo com a aplicação de conceitos básicos de machine learning para previsão é possível retirar inúmeras informações e aprendizados, que vão muito além das técnicas da linguagem de programação, que é o lado do estudo estatístico e analítico dos dados.

Além disso, foi necessário de um amplo estudo sobre o tema estudando, através de pesquisas em sites, como também conversas com pessoas que se interessam pelo assunto. A fim de obter mais embasamento para os caminhos a serem seguidos no projeto.

A elaboração do trabalho no geral foi prazerosa, pelo fato de conseguir-se ver claramente a evolução teórica e prática obtida durante a própria execução da tarefa.

Referências

In [36]:

```
#https://www.kaggle.com/edgarhuichen/espn-nba-players-data (dados NBA)
#https://medium.com/horadecodar/como-tratar-dados-nulos-no-dataset-4f0470b22d38 (como tratar dados nulos)
#https://estatsite.com.br/2020/07/25/dividindo-o-dataset-em-treino-e-teste-no-pyton (dividindo dataset)
#https://tatianaesc.medium.com/implementando-um-modelo-de-classificac%C3%A7%C3%A3o-em-machine-learning-3a2a2a2a (implementando modelo)
#https://www.linkedin.com/pulse/pr%C3%A9-processamento-de-dados-intui%C3%A7%C3%A3o-e-estruturação-de-dados-17a3c3a3 (pré-processamento)
```

In []:

