nba_salary

March 25, 2020

0.1 Introdução a Aprendizado de Máquina

0.1.1 Professor Enerson Oliveira

0.1.2 João Manoel Lins - 1924520-X

0.1.3 joaomanoellins@gmail.com

O objetivo é encontrar um modelo, que de acordo com as médias do desempenho dos jogadores em quadra, idade, altura, peso e entre outros, consiga prever seu salário.

Em memória de Kobe Bryant

#KB24

[1]:
```
import os

import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

pd.options.display.max_columns = None
sns.set()

%matplotlib inline
```

[2]:
```
dataset_path = os.path.join('.', 'datasets', 'NBA_Players.csv')
```

1 Carregando o dataset

[3]:
```
nba = pd.read_csv(dataset_path)
```

2 Analisando os dados

[4]:
```
nba.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 30 columns):
TEAM                550 non-null object
 NAME               550 non-null object
 EXPERIENCE         550 non-null int64
 URL                550 non-null object
 POSITION           550 non-null object
 AGE                550 non-null object
 HT                 550 non-null float64
 WT                 550 non-null float64
 COLLEGE            550 non-null object
 SALARY             550 non-null object
 PPG_LAST_SEASON    538 non-null float64
 APG_LAST_SEASON    538 non-null float64
 RPG_LAST_SEASON    538 non-null float64
 PER_LAST_SEASON    538 non-null float64
 PPG_CAREER         550 non-null float64
 APG_CAREER         550 non-null float64
 RGP_CAREER         550 non-null float64
 GP                 550 non-null int64
 MPG                550 non-null float64
 FGM_FGA            550 non-null object
 FGP                550 non-null float64
 THM_THA            550 non-null object
 THP                550 non-null float64
 FTM_FTA            550 non-null object
 FTP                550 non-null float64
 APG                550 non-null float64
 BLKPG              550 non-null float64
 STLPG              550 non-null float64
 TOPG               550 non-null float64
 PPG                550 non-null float64
dtypes: float64(18), int64(2), object(10)
memory usage: 129.0+ KB
```

Legenda:

PPG_LAST_SEASON - Points Per Game - Last Season

APG_LAST_SEASON - Assists Per Game - Last Season

RPG_LAST_SEASON - Rebounds Per Game - Last Season

PER_LAST_SEASON - Player Efficiency Rating is the overall rating of a player's per-minute statistical production. The league average is 15.00 every season.

PPG_CARRER - Points Per Game - Carrer

APG_CARRER - Assists Per Game - Carrer

RPG_CARRER - Rebounds Per Game - Carrer

GP - Games Played

MPG - Minutes Per Game

FGM_FGA - Field Goals Made-Attempted Per Game

FGP - Field Goal Percentage

THM_THA - ???

THP - ???

FTM_FTA - Free Throws Made-Attempted Per Game

FTP - Free Throws Percentage

APG - Assists Per Game

BLKPG - Blocks Per Game

STLPG - Steals Per Game

TOPG - Turnover Per Game

PPG - Points Per Game

[5]:
```
nba.describe()
```

[5]:

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

	APG_LAST_SEASON	RPG_LAST_SEASON	PER_LAST_SEASON	PPG_CAREER
count	538.000000	538.000000	538.000000	550.000000
mean	2.667472	1.727881	10.870725	6.944545
std	2.809521	1.861049	9.382982	5.912676
min	0.000000	0.000000	-28.480000	0.000000
25%	0.200000	0.000000	3.745000	1.500000
50%	1.900000	1.100000	12.105000	6.400000
75%	4.200000	2.875000	15.877500	10.500000
max	16.000000	12.300000	133.950000	27.200000

	APG_CAREER	RGP_CAREER	GP	MPG	FGP
count	550.000000	550.000000	550.000000	550.000000	550.000000
mean	2.622182	1.708182	249.561818	16.548000	0.350011
std	2.665679	1.732489	285.841975	11.496338	0.202934
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.300000	0.125000	6.000000	4.725000	0.327750

```
50%       1.800000     1.200000   141.500000   18.250000    0.430500
75%       4.100000     2.800000   409.500000   26.100000    0.469000
max      13.400000    11.700000  1471.000000   38.800000    1.000000

              THP         FTP         APG       BLKPG       STLPG        TOPG  \
count  550.000000  550.000000  550.000000  550.000000  550.000000  550.000000
mean     0.230956    0.560867    1.493455    0.336909    0.541091    0.943273
std      0.171717    0.331210    1.658239    0.409453    0.454148    0.823085
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.423000    0.125000    0.000000    0.100000    0.200000
50%      0.310000    0.723000    1.100000    0.200000    0.500000    0.900000
75%      0.358750    0.796500    2.100000    0.500000    0.800000    1.400000
max      1.000000    1.000000    9.800000    2.400000    2.300000    4.000000

              PPG
count  550.000000
mean     6.944545
std      5.912676
min      0.000000
25%      1.500000
50%      6.400000
75%     10.500000
max     27.200000
```

[6]:
```
nba.head(5)
```

[6]:
```
              TEAM          NAME   EXPERIENCE  \
0  Boston Celtics   Aron Baynes            6
1  Boston Celtics  Justin Bibbs            0
2  Boston Celtics   Jabari Bird            1
3  Boston Celtics  Jaylen Brown            2
4  Boston Celtics     PJ Dozier            1

                                          URL  POSITION  AGE      HT      WT  \
0  http://www.espn.com/nba/player/_/id/2968439        SF   31  208.28  117.65
1  http://www.espn.com/nba/player/_/id/3147500         G   22  195.58   99.55
2  http://www.espn.com/nba/player/_/id/3064308        SG   24  198.12   89.59
3  http://www.espn.com/nba/player/_/id/3917376         F   21  200.66   99.55
4  http://www.espn.com/nba/player/_/id/3923250        PG   21  198.12   92.76

            COLLEGE      SALARY   PPG_LAST_SEASON   APG_LAST_SEASON  \
0  Washington State   5,193,600               6.0               1.1
1     Virginia Tech  Not signed               0.0               0.0
2        California   1,349,464               3.0               0.6
3        California   5,169,960              14.5               1.6
4    South Carolina  Not signed               1.0               0.0
```

```
    RPG_LAST_SEASON   PER_LAST_SEASON   PPG_CAREER   APG_CAREER   RGP_CAREER  \
0               5.4             12.09          5.4          0.7          4.4
1               0.0              0.00          0.0          0.0          0.0
2               1.5             12.18          3.0          0.6          1.5
3               4.9             13.69         10.4          1.2          3.8
4               0.5             -4.82          1.0          0.0          0.5

    GP   MPG  FGM_FGA    FGP  THM_THA    THP  FTM_FTA    FTP   APG   BLKPG  \
0  376  15.0  2.2-4.3  0.502  0.0-0.1  0.143  1.0-1.3  0.802   0.7     0.5
1    0   0.0        0  0.000        0  0.000        0  0.000   0.0     0.0
2   13   8.8  1.2-2.0  0.577  0.2-0.5  0.429  0.5-1.0  0.462   0.6     0.1
3  148  23.6  3.8-8.3  0.461  1.1-3.0  0.379  1.6-2.4  0.658   1.2     0.3
4    2   1.5  0.5-1.0  0.500  0.0-0.0  0.000  0.0-0.0  0.000   0.0     0.0

    STLPG   TOPG   PPG
0     0.2    0.8   5.4
1     0.0    0.0   0.0
2     0.2    0.6   3.0
3     0.7    1.3  10.4
4     0.0    0.5   1.0
```

[7]:
```
[col for col in nba.columns.array]
```

[7]:
```
['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']
```

As colunas possuem um problema. Há um espaço antes de cada nome.

[8]:
```
# Corrigindo. Tirando os espaços no nome das colunas
# Aproveitei para colocar as colunas em minúsculo
nba.columns = [c.strip().lower() for c in nba.columns]

[col for col in nba.columns.array]
```

[8]:
```
['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']
```

2.1 Correções nas colunas

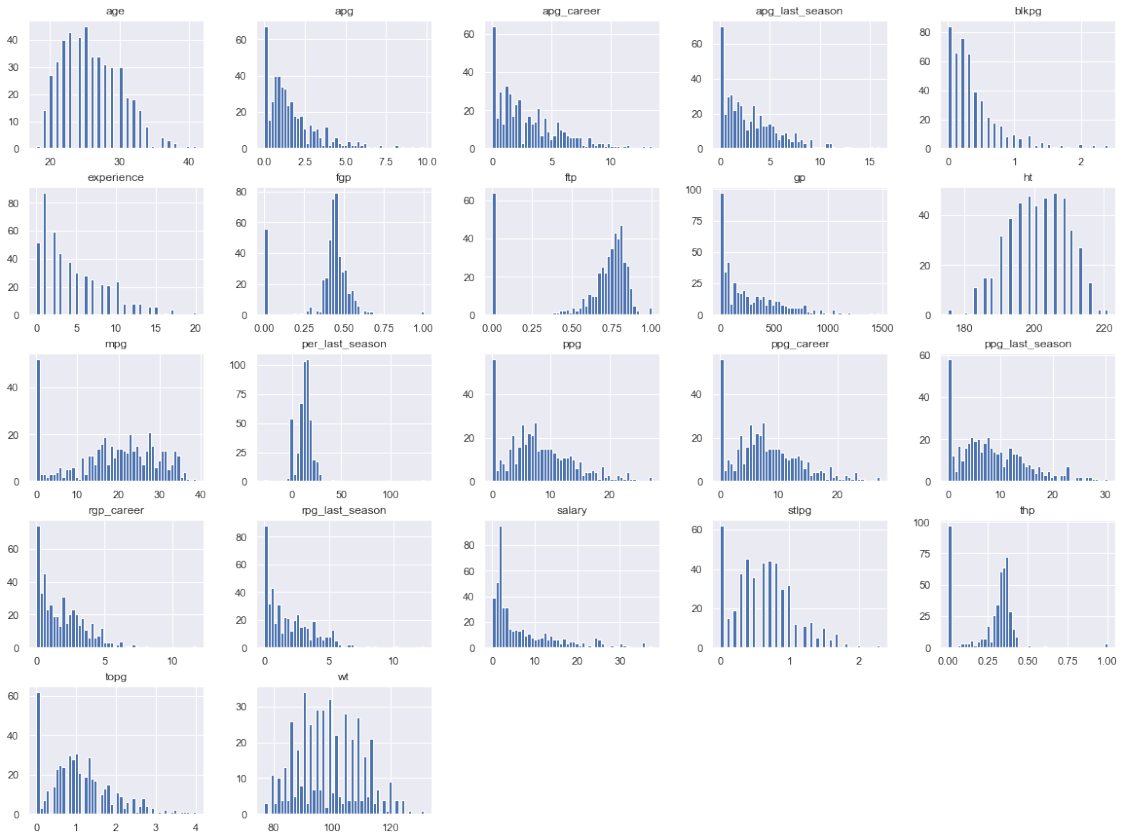
[9]:
```
def is_teams_ok():
    # Há 30 times na NBA
    assert len(nba['team'].value_counts().index) == 30

def is_position_ok():
    pos_len = len(nba['position'].value_counts().index)

    # Geralmente se usa no basquete 5 posições.
    # Mas pode ter até 7 se considerarmos as posições G e F
    # G é um armador, mas não considerado um PG ou SG
    # F é um ala, mas não considerado um SF ou PF
    assert pos_len >= 5 and pos_len <= 7

# Remover jogadores que não tiveram salário e nem jogaram uma partida
def remove_players_no_games_salary(df):
    idxs = df[(df['gp'] == 0) & (df['salary'] == 0)].index
    df.drop(idxs, axis=0, inplace=True)

# Se tiver erro vai parar o algoritmo
# Precisa de uma intervenção manual
is_teams_ok()

# Se tiver erro vai parar o algoritmo
# Precisa de uma intervenção manual
is_position_ok()

# "age" -> tem um bug do -
# Vamos trocar - por 0
nba['age'] = nba['age'].str.replace('-', '0').astype(int)

# "college" -> quem tiver "-" é porque pode ser estrangeiro,
# veio direto do ensino médio (LeBron James), não chegou a cursar faculdade
# ou por falta informação na coleta
nba['college'] = nba['college'].str.replace('-', 'ND')

# Temos 2 problemas:
# 1) Not Signed não é um float
# 2) Salário com ","
nba['salary'] = nba['salary'].str.replace('Not signed', '0')
nba['salary'] = nba['salary'].str.replace(',', '').astype(float)
nba['salary'] = nba['salary'] / 1000000 # Colocar em milhões

print(f'Quantidade de jogadores antes da limpeza: {nba.shape[0]}')
remove_players_no_games_salary(nba)
```

```
# Vamos filtrar os jogadores das posições G e F. Pois são poucos
nba = nba.loc[(nba['position'] != 'G') & (nba['position'] != 'F')]
nba_bkp = nba.copy()

print(f'Quantidade de jogadores depois da limpeza: {nba.shape[0]}')
```

```
Quantidade de jogadores antes da limpeza: 550
Quantidade de jogadores depois da limpeza: 473
```

[10]:
```
nba.hist(bins=50, figsize=(20,15))
```

[10]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1185e7050>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x111bd3590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a65ba50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a694dd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a6caa90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x11a709e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a73ead0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a77c310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a77ce50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a7bb810>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x11a825b50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a863ed0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a89ab50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a8d8ed0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a90db90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x11a94df10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a981bd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a9c1f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11a9f4c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11aa35f90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x11aa69c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11aaa8fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11aaddc90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11ab1c4d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11ab51cd0>]],
      dtype=object)
```



Podemos notar que há um grupo considerável de jogadores com 0 de média em vários atributos.

3 Criar um conjunto de teste

Esse conjuto vai ser separado e não seria mexido. Vamos separar os jogadores em grupos de acordo com a sua quantidade de temporadas na liga.

Separamos em 4 categorias:

0 a 5 anos de exp

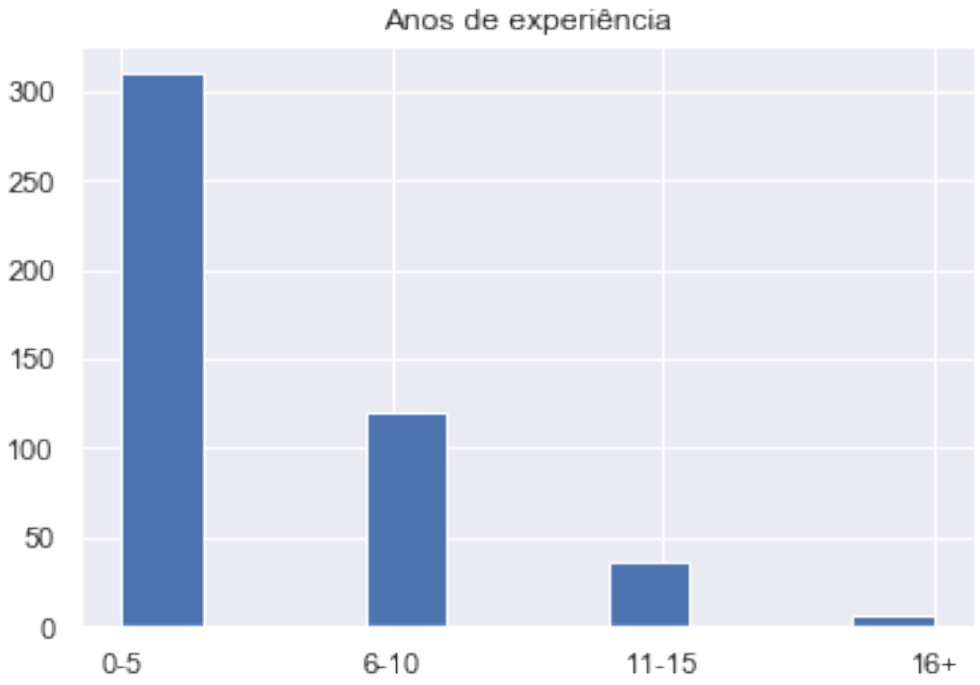
6 a 10 anos de exp

11 a 15 anos de exp

16 ou mais anos de exp

[11]:
```
nba['exp_cat'] = pd.cut(nba['experience'] / 5,
                        bins=[-np.inf, 1, 2, 3, np.Inf],
                        labels=['0-5', '6-10', '11-15', '16+'])

plt.title('Anos de experiência')
nba['exp_cat'].sort_values().hist()
```

[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1185e7790>



Agora que temos uma separação por grupo, vamos fazer o split dos conjuntos levando.

4 Visualizar e ganhar insights

4.1 Visualizar gráficos

[12]:
```
def top_college_chart(df, total=15):
    order = df['college'].value_counts(ascending=False).iloc[:total].index.
 ↪tolist()
    fig, _ = plt.subplots(figsize=((10, 5)))

    g = sns.countplot('college', data=df, order=order)
    g.set_xlabel('Universidade')
    g.set_ylabel('Quantidade')

    for p in g.patches:
        points = p.get_bbox().get_points()
        x = sum(points[:, 0]) / 2
        y = points[1,1]
        label = '{:.1f}%'.format((y / len(df)) * 100)

        g.annotate(label, (x, y), ha='center', va='bottom')
```

```
    g.set_title(f'Top {total} Universidades')
    fig.autofmt_xdate(rotation=45)
```

[13]:
```
def exp_pie_chart(df):
    labels = [c + ' anos' for c in df['exp_cat'].cat.categories.tolist()]
    values = df['exp_cat'].value_counts().tolist()
    explode = (0, 0, 0.1, 0.2)

    fig, ax = plt.subplots(figsize=(5, 5))
    _,_, autotexts = ax.pie(values, labels=labels,
           autopct='%1.1f%%', startangle=90,
            explode=explode,
           labeldistance=1.05, pctdistance=.55)
    ax.axis('equal')

    plt.title('Divisão por Quantidade de Temporadas')

    for autotext in autotexts:
        autotext.set_color('white')
```

[14]:
```
def pos_pie_chart(df):
    counts = df['position'].value_counts()
    labels = counts.keys().tolist()
    values = counts.tolist()

    fig, ax = plt.subplots(figsize=(6, 6))
    _,_,autotexts = ax.pie(values, labels=labels,
           autopct='%1.1f%%', startangle=0,
           labeldistance=1.05, pctdistance=.75)

    plt.title('Divisão por Posição')

    for autotext in autotexts:
        autotext.set_color('white')
```

[15]:
```
def salary_sum_chart(df):
    team_salary = df.groupby('team')['salary'].sum().
 ↪sort_values(ascending=False)
    labels = team_salary.index.tolist()
    values = team_salary.values
    salary_mean = np.mean(values)
    colors = ['#4C9900' if v > salary_mean else '#CC0000' for v in values]

    fig, _ = plt.subplots(figsize=(20, 8))
    ax = sns.barplot(x=labels, y=values, palette=colors)
```

```
    ax.set_title('Salário Total por Time', size=20)
    ax.set_xlabel('Time', size=14)
    ax.set_ylabel('Salário (Milhões)', size=14)

    ax.axhline(salary_mean, color='blue', linewidth=.8, label='Média de Salário␣
↪por Ano')
    ax.legend()

    for p in ax.patches:
        points = p.get_bbox().get_points()
        x = sum(points[:, 0]) / 2
        y = points[1, 1]
        label = '${:.1f} M'.format(y)

        ax.annotate(label, (x, y), ha='center', va='bottom')

    fig.autofmt_xdate(rotation=45)
```

[16]:
```
def stats_grouped_by_position_chart(df, attr, attr_name, kind, kind_name):
    g = sns.relplot(attr, y=kind,
                col='position', col_wrap=3,
                hue='exp_cat', size='exp_cat',
                legend='full',
                data=df)

    plt.subplots_adjust(top=0.90)

    g.fig.suptitle(f'Média de {attr_name} por {kind_name} - Agrupado por␣
↪Posição', size=22)
    g.set_axis_labels(attr_name, kind_name)
```

[17]:
```
def hightest_salary_per_position_by_team_heatmap(df):
    # ids dos jogadores com maior salario por posição por time
    nba_hghst_salary = df.groupby(['team', 'position'])['salary'].idxmax().
↪tolist()
    data = df.loc[nba_hghst_salary]
    nba_salary_table = pd.pivot_table(data, values='salary', index='team',␣
↪columns=['position'])

    fig, ax = plt.subplots(figsize=(12, 12))
    ax.set_title('Maior Salário por Posição por Time (Milhão)')

    sns.heatmap(nba_salary_table, cmap="seismic", linewidth=.7,
                robust=True, annot=True, annot_kws={'size': 13})

    ax.set_xlabel('Posição')
    ax.set_ylabel('Time')
```

[18]:
```
def mean_salary_per_position_by_team_heatmap(df):
    nba_mean_table = pd.pivot_table(df, values='salary',
                                    index='team', columns=['position'],
                                    aggfunc=np.mean)

    fig, ax = plt.subplots(figsize=(12, 12))
    ax.set_title('Média de Salário por Posição Por Time (Milhão)')

    sns.heatmap(nba_mean_table, cmap="seismic", linewidth=.7,
                robust=True, annot=True, annot_kws={'size': 13})

    ax.set_xlabel('Posição')
    ax.set_ylabel('Time')
```

[19]:
```
def top20_salaries(df):
    idx = df['salary'].sort_values(ascending=False)[:20].index
    return nba.loc[idx, ['team', 'name', 'salary', 'position', 'age', 'mpg',␣
 ↪'ppg', 'apg', 'stlpg', 'topg']]
```

Vamos visualizar:

Média de pontos e minutos por jogo

Média de assistências e minutos por jogo

Média de tocos e minutos por jogo

Média de roubadas e minutos por jogo

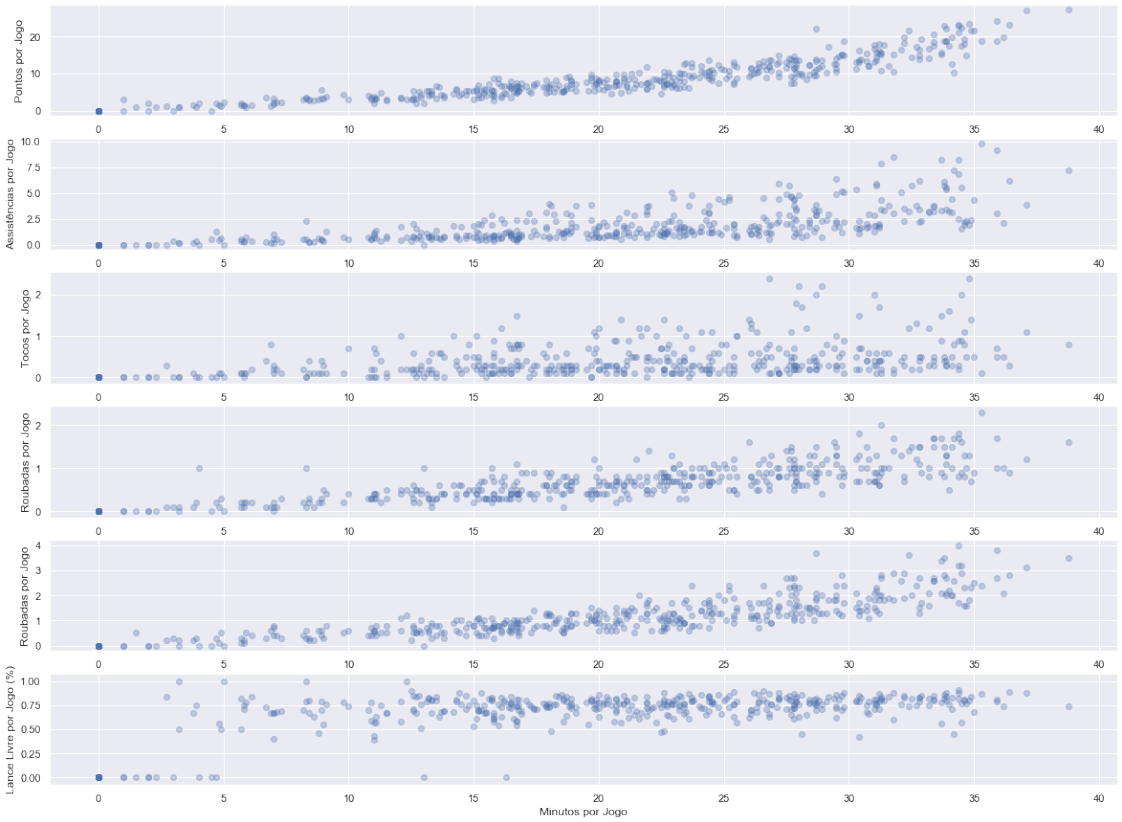
Média de turnover e minutos por jogo

Média de lance-livre e minutos por jogo

Média de field-goal e minutos por jogo

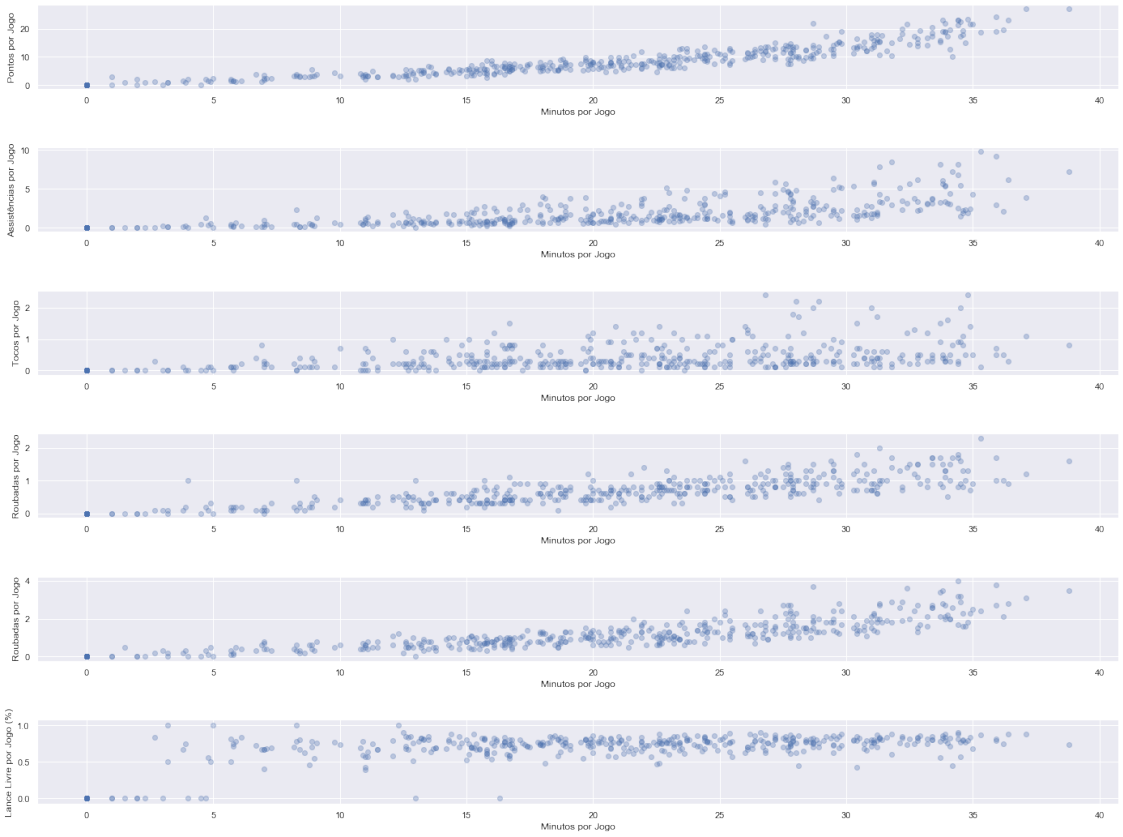
[20]:
```
keys = ['ppg', 'apg', 'blkpg', 'stlpg', 'topg', 'ftp', 'fgp']
labels = [
    'Pontos por Jogo',
    'Assistências por Jogo',
    'Tocos por Jogo',
    'Roubadas por Jogo',
    'Roubadas por Jogo',
    'Lance Livre por Jogo (%)',
    'Field-Goal por Jogo (%)',
]

fig, axs = plt.subplots(6, figsize=(20,15))
fig.set_colorbar = True

for idx, ax in enumerate(axs):
    key = keys[idx]
```

```
    label = labels[idx]

    ax.set_xlabel('Minutos por Jogo')
    ax.set_ylabel(label)

    ax.scatter(nba['mpg'], nba[key], alpha=0.3)
```



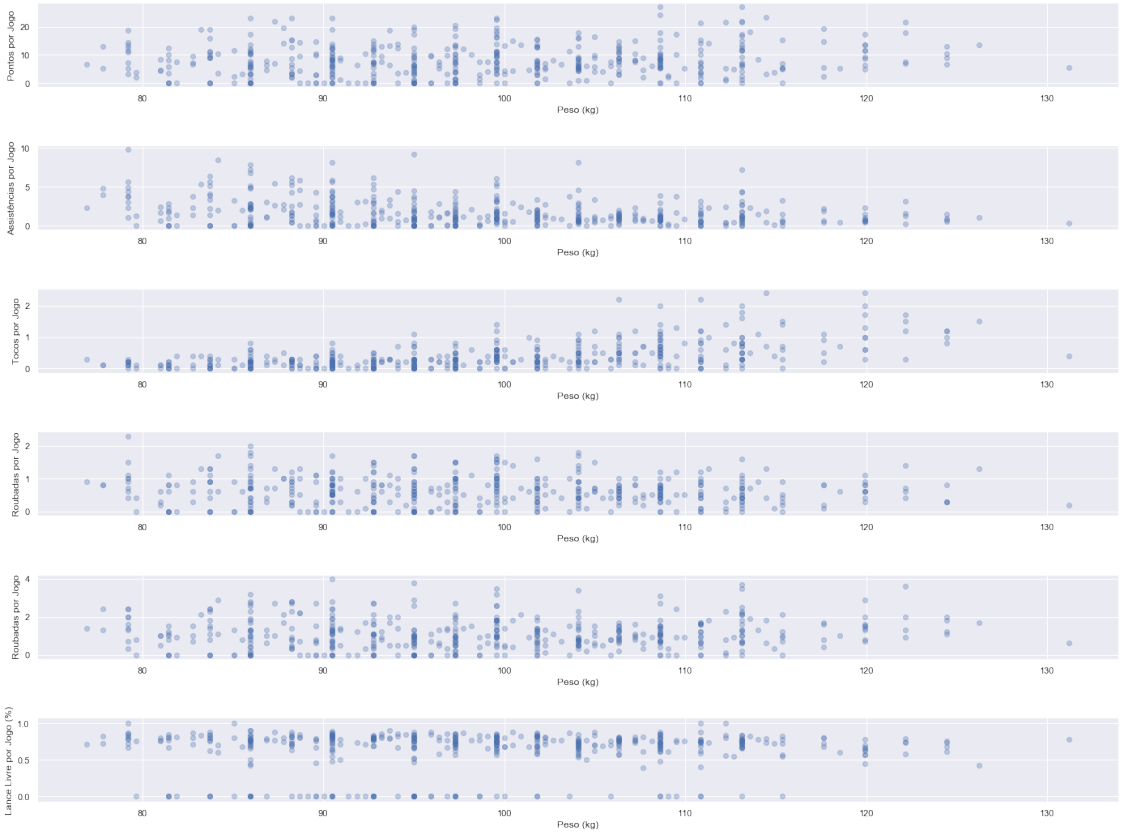
[21]:
```
# Analise de minutos em relação aos stats
keys = ['ppg', 'apg', 'blkpg', 'stlpg', 'topg', 'ftp', 'fgp']
labels = [
    'Pontos por Jogo',
    'Assistências por Jogo',
    'Tocos por Jogo',
    'Roubadas por Jogo',
    'Roubadas por Jogo',
    'Lance Livre por Jogo (%)',
    'Field-Goal por Jogo (%)',
]

fig, axs = plt.subplots(6, figsize=(20,15))
```

```
for idx, ax in enumerate(axs):
    key = keys[idx]
    label = labels[idx]

    ax.set_xlabel('Minutos por Jogo')
    ax.set_ylabel(label)

    ax.scatter(nba['mpg'], nba[key], alpha=0.3)

fig.tight_layout(pad=3.0)
```



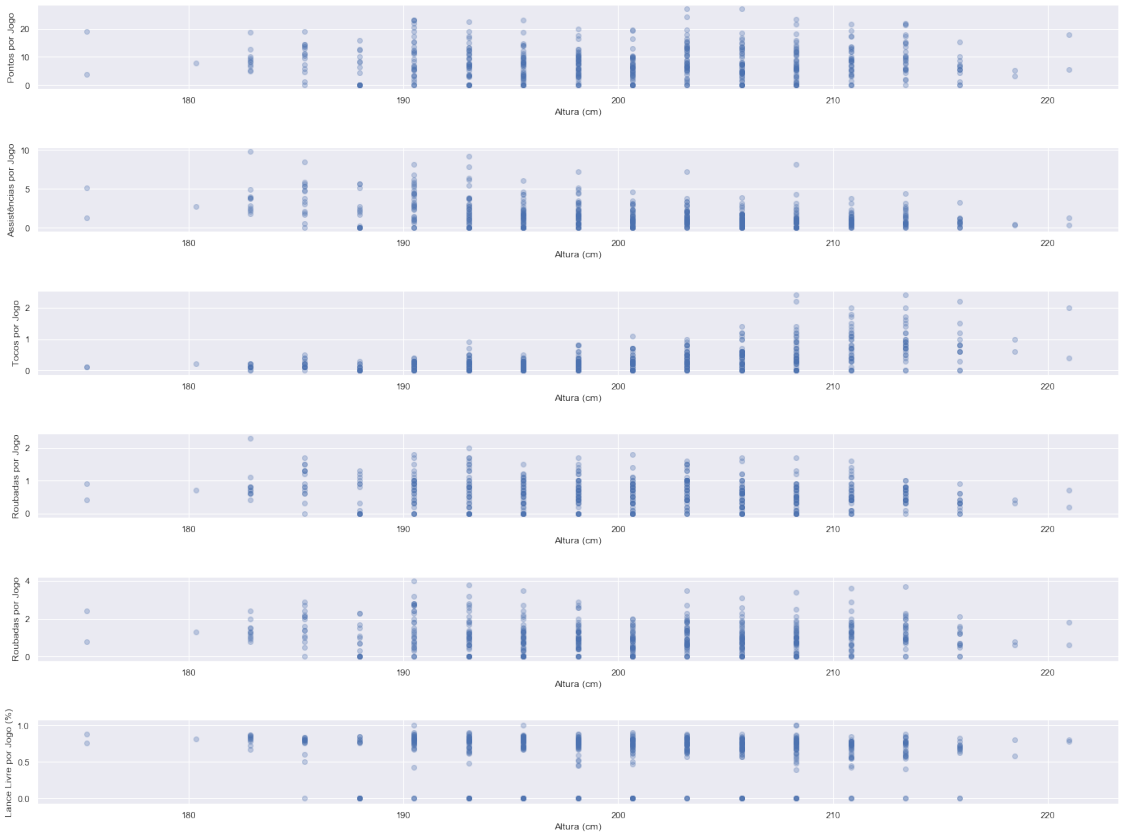
[22]:
```
# Analise de peso em relação aos stats
keys = ['ppg', 'apg', 'blkpg', 'stlpg', 'topg', 'ftp', 'fgp']
labels = [
    'Pontos por Jogo',
    'Assistências por Jogo',
    'Tocos por Jogo',
    'Roubadas por Jogo',
    'Roubadas por Jogo',
    'Lance Livre por Jogo (%)',
    'Field-Goal por Jogo (%)',
```

```
]

fig, axs = plt.subplots(6, figsize=(20,15))

for idx, ax in enumerate(axs):
    key = keys[idx]
    label = labels[idx]

    ax.set_xlabel('Peso (kg)')
    ax.set_ylabel(label)

    ax.scatter(nba['wt'], nba[key], alpha=0.3)

fig.tight_layout(pad=3.0)
```

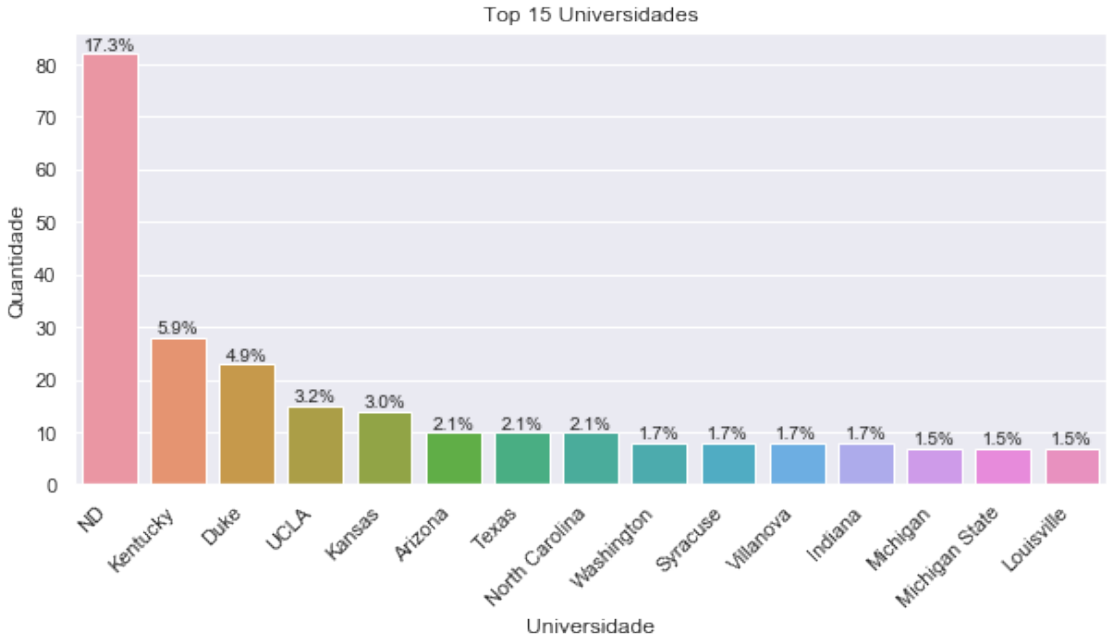


[23]:
```
# Analise de altura em relação aos stats
keys = ['ppg', 'apg', 'blkpg', 'stlpg', 'topg', 'ftp', 'fgp']
labels = [
    'Pontos por Jogo',
    'Assistências por Jogo',
    'Tocos por Jogo',
```

```
    'Roubadas por Jogo',
    'Roubadas por Jogo',
    'Lance Livre por Jogo (%)',
    'Field-Goal por Jogo (%)',
]

fig, axs = plt.subplots(6, figsize=(20,15))

for idx, ax in enumerate(axs):
    key = keys[idx]
    label = labels[idx]

    ax.set_xlabel('Altura (cm)')
    ax.set_ylabel(label)

    ax.scatter(nba['ht'], nba[key], alpha=0.3)

fig.tight_layout(pad=3.0)
```

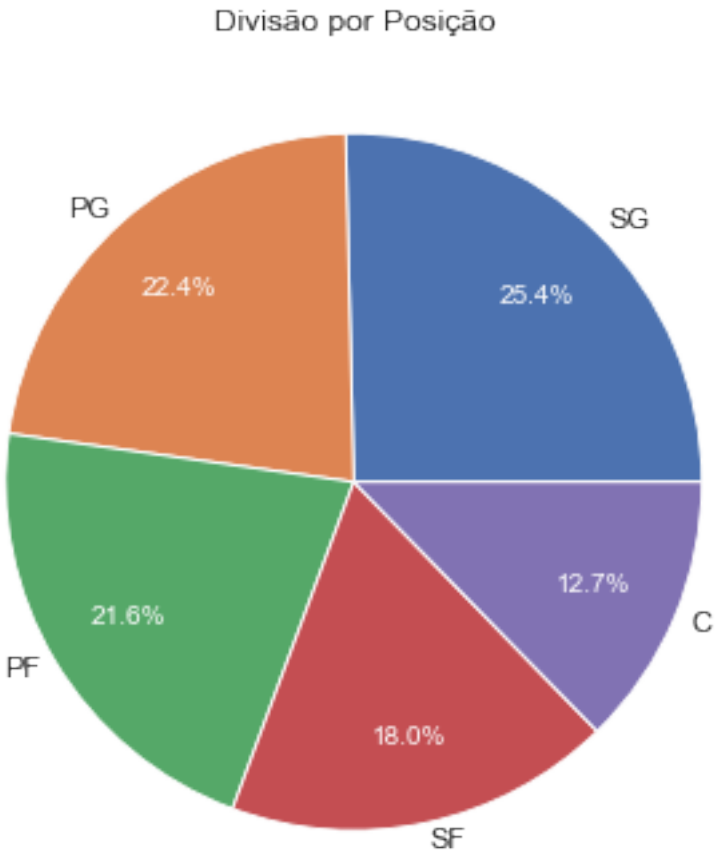


[24]:
```
top_college_chart(nba)
```

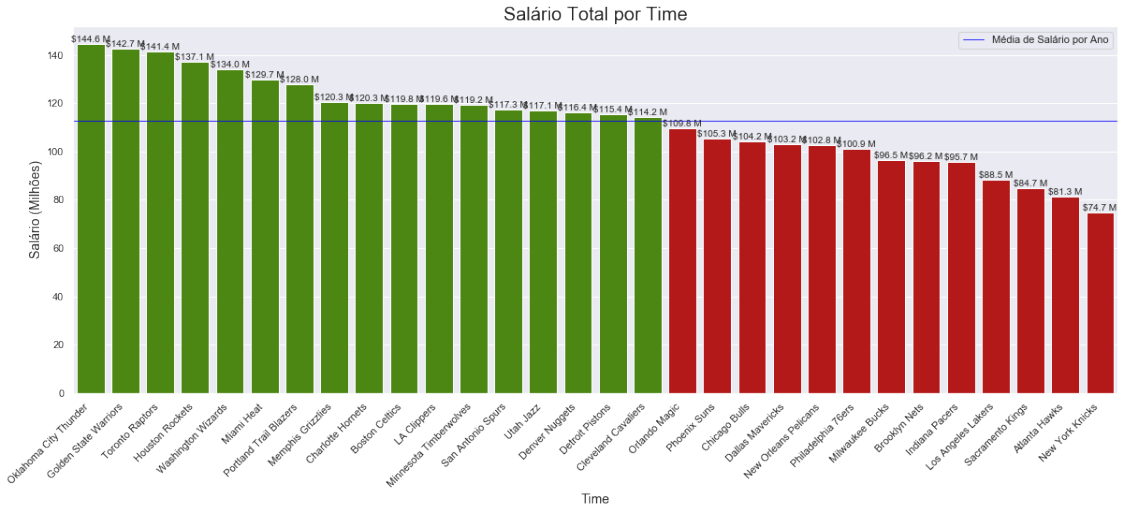


[25]:
```
exp_pie_chart(nba)
```

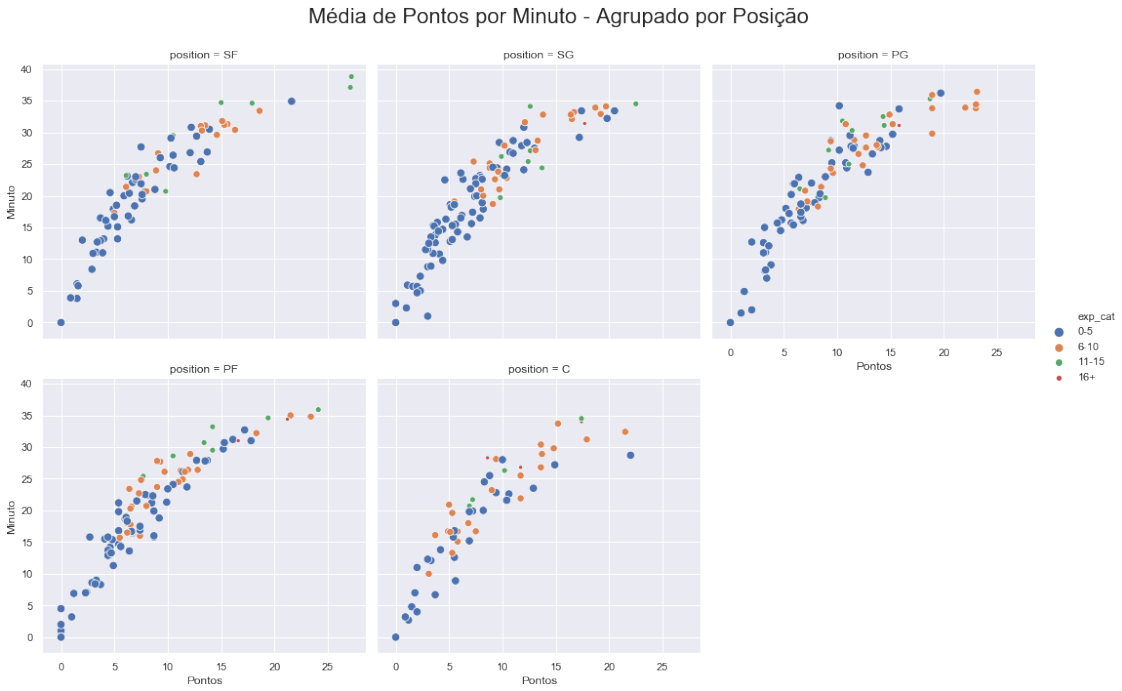
[26]:
```
pos_pie_chart(nba)
```

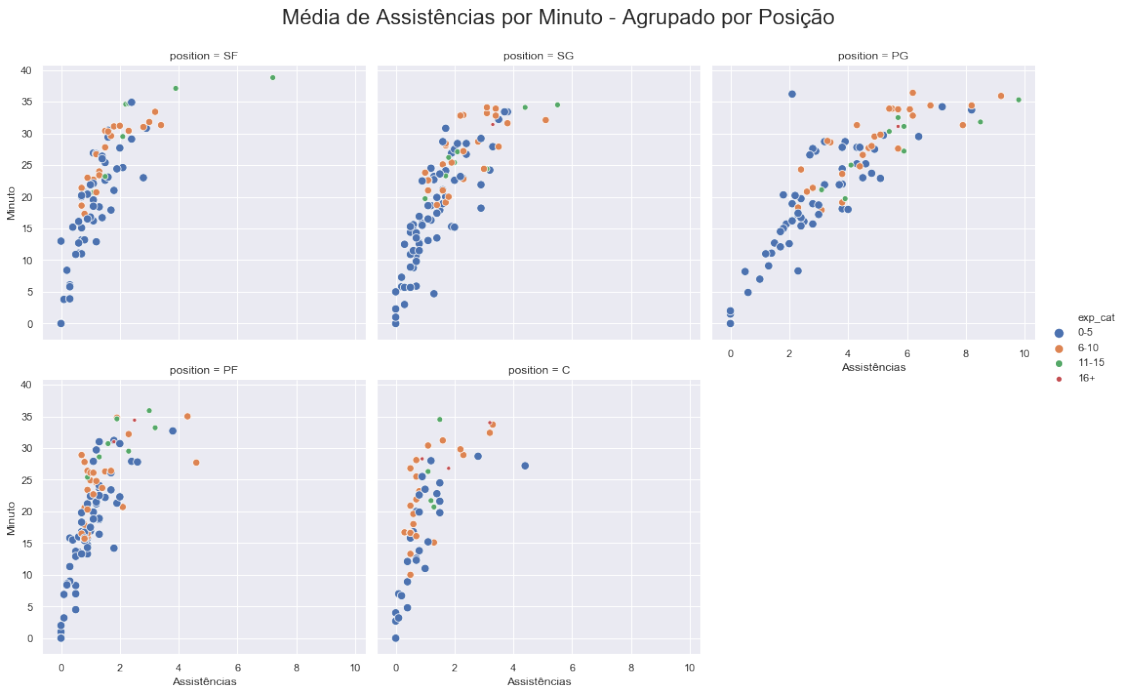


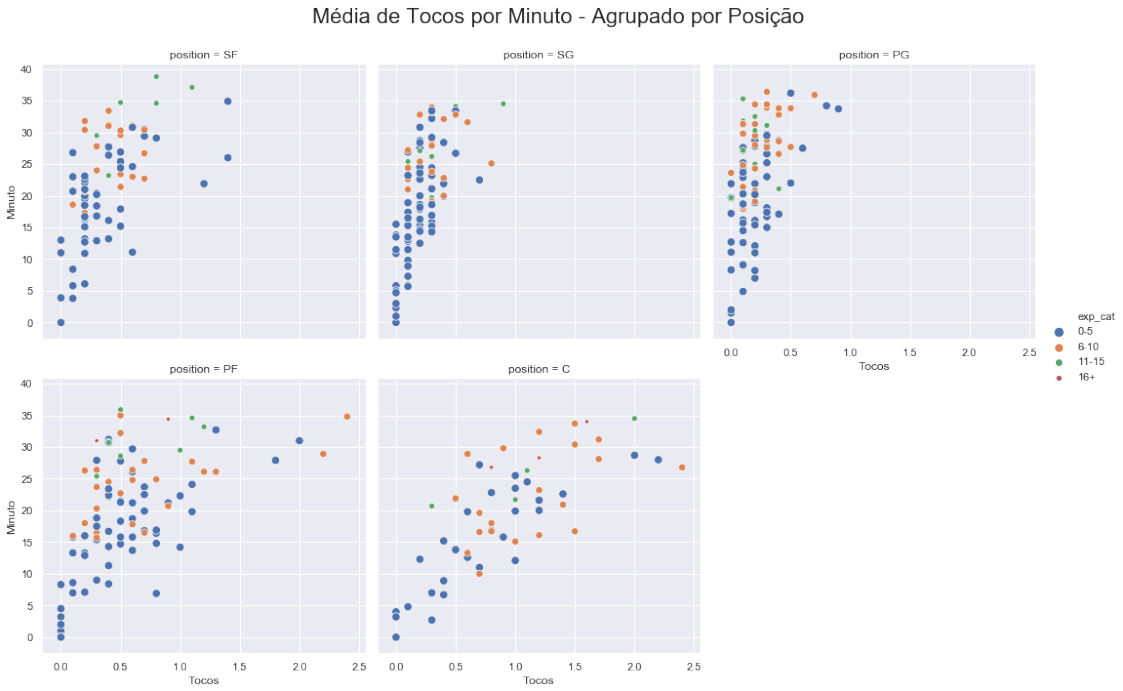
[27]:
```
salary_sum_chart(nba)
```



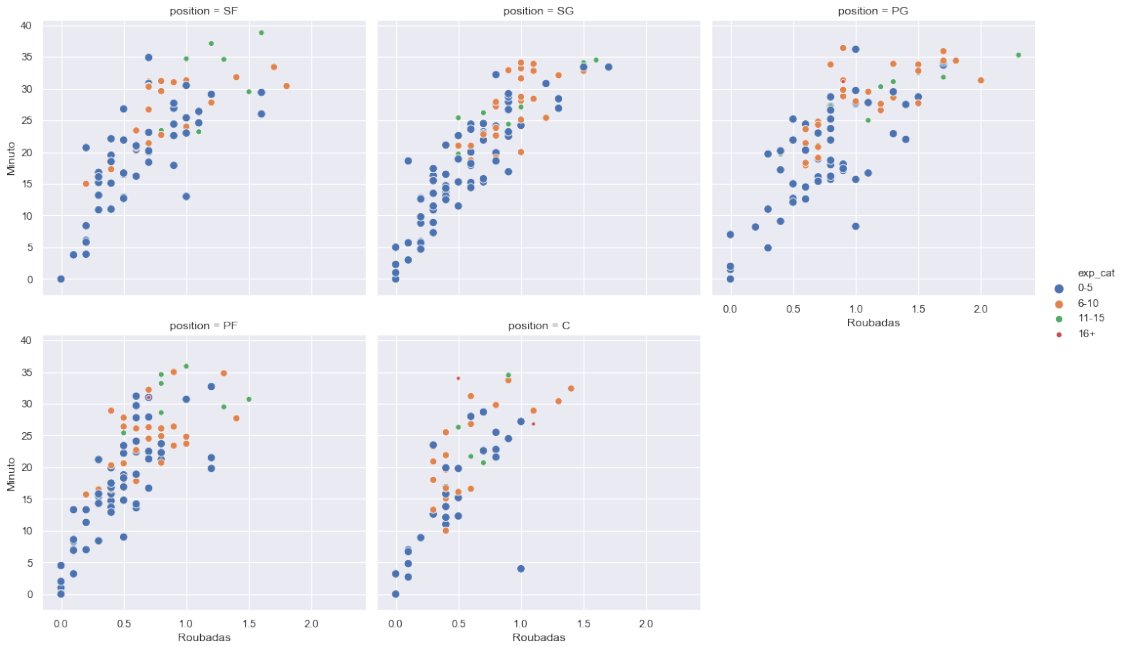
[28]:
```
kinds = ['mpg', 'salary']
kinds_name = ['Minuto', 'Salário']

attrs = ['ppg', 'apg', 'blkpg', 'stlpg', 'topg']
attrs_name = ['Pontos', 'Assistências', 'Tocos', 'Roubadas', 'Turnovers']

for kind, kName in zip(kinds, kinds_name):
    for attr, aName in zip(attrs, attrs_name):
        stats_grouped_by_position_chart(nba, attr, aName, kind, kName)
```



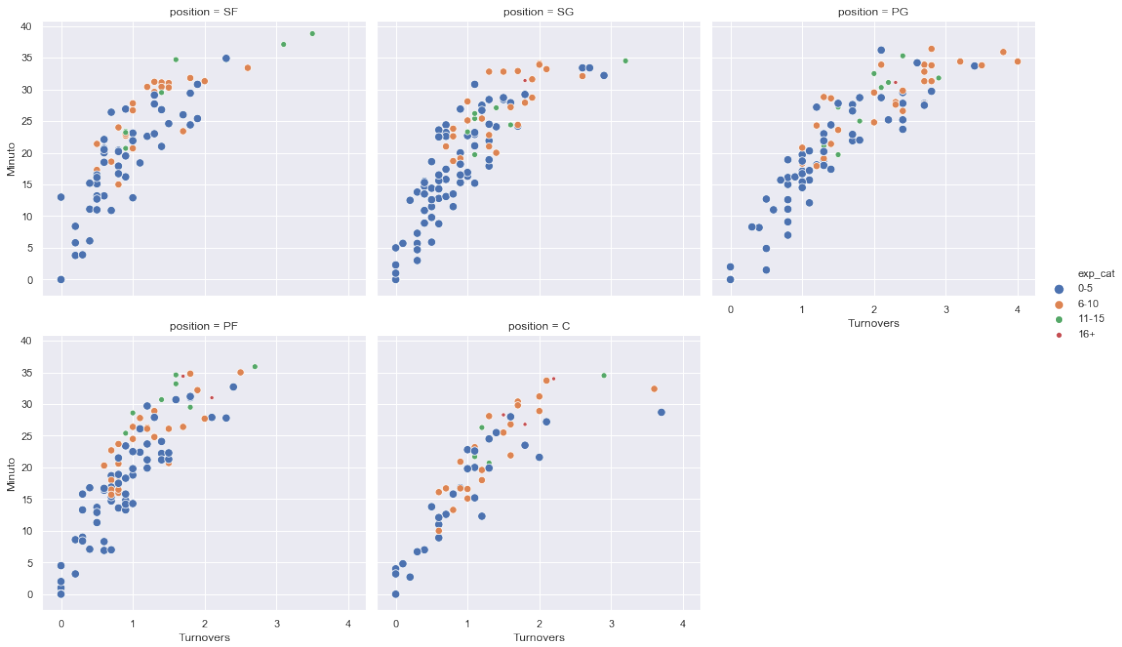

Média de Assistências por Minuto - Agrupado por Posição
position = SF
position = SG
position = PG
position = PF
position = C
Minuto
Assistências
exp_cat
0-5
6-10
11-15
16+


Média de Tocos por Minuto - Agrupado por Posição
position = SF
position = SG
position = PG
position = PF
position = C
Minuto
Tocos
exp_cat
0-5
6-10
11-15
16+

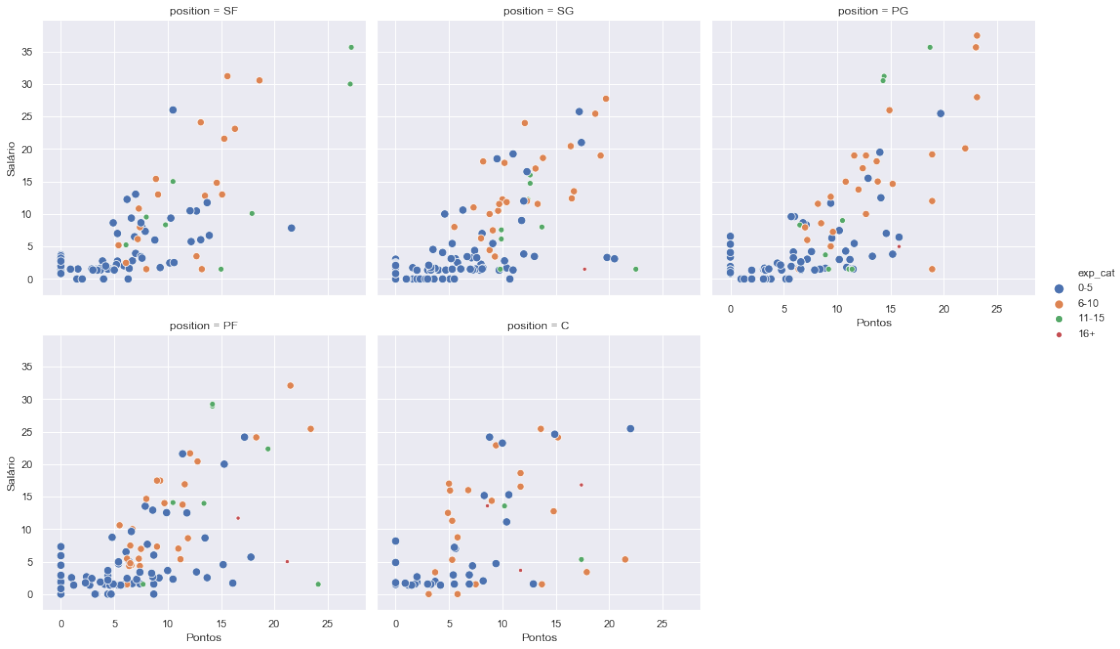
Média de Roubadas por Minuto - Agrupado por Posição



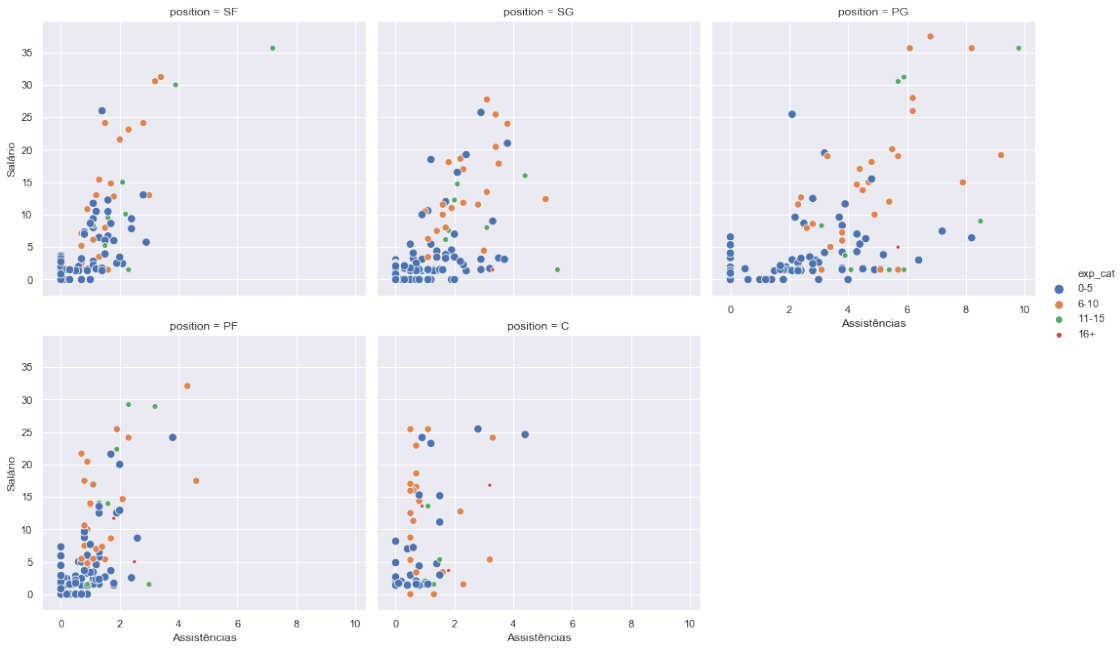
Média de Turnovers por Minuto - Agrupado por Posição

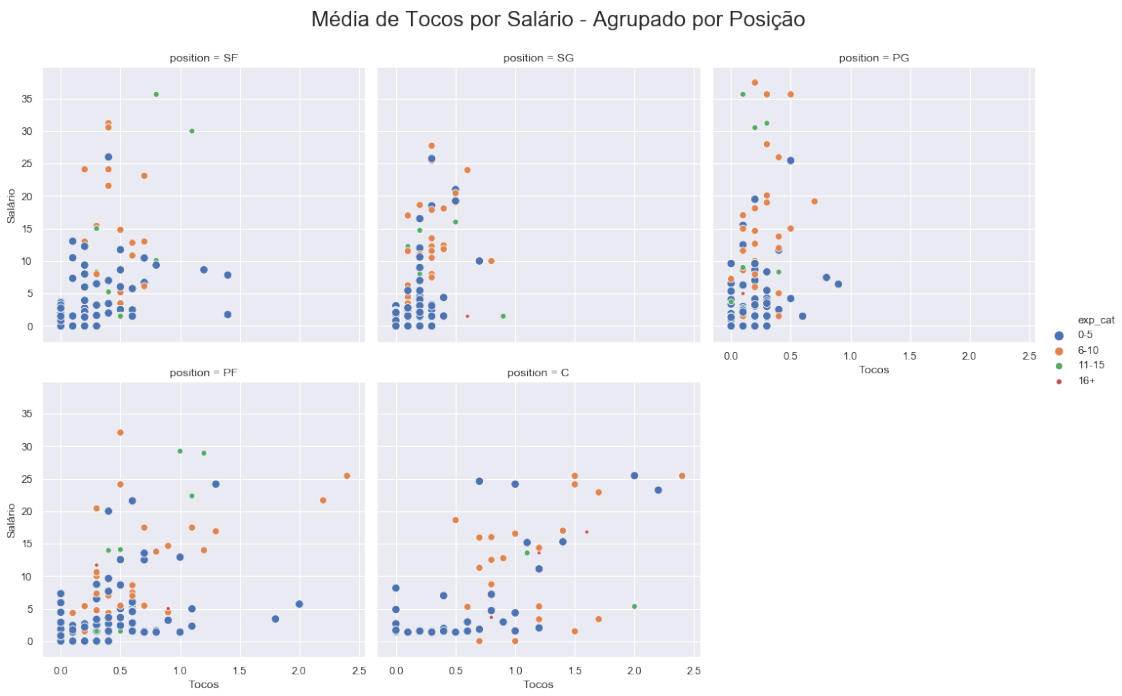


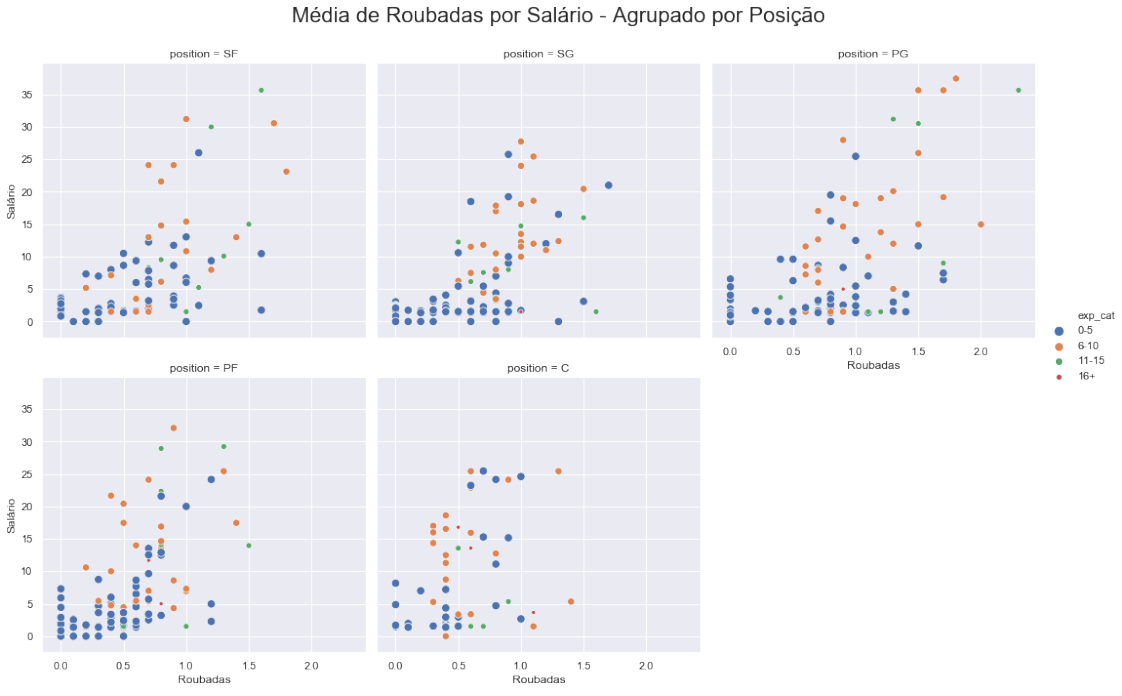
Média de Pontos por Salário - Agrupado por Posição

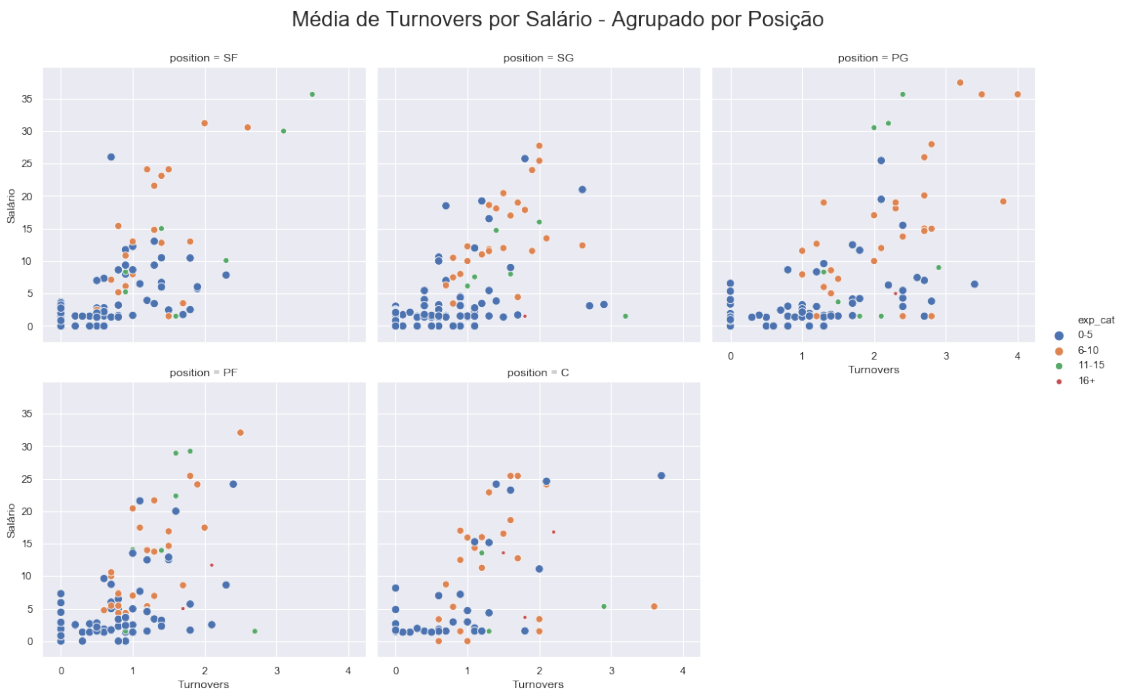


Média de Assistências por Salário - Agrupado por Posição

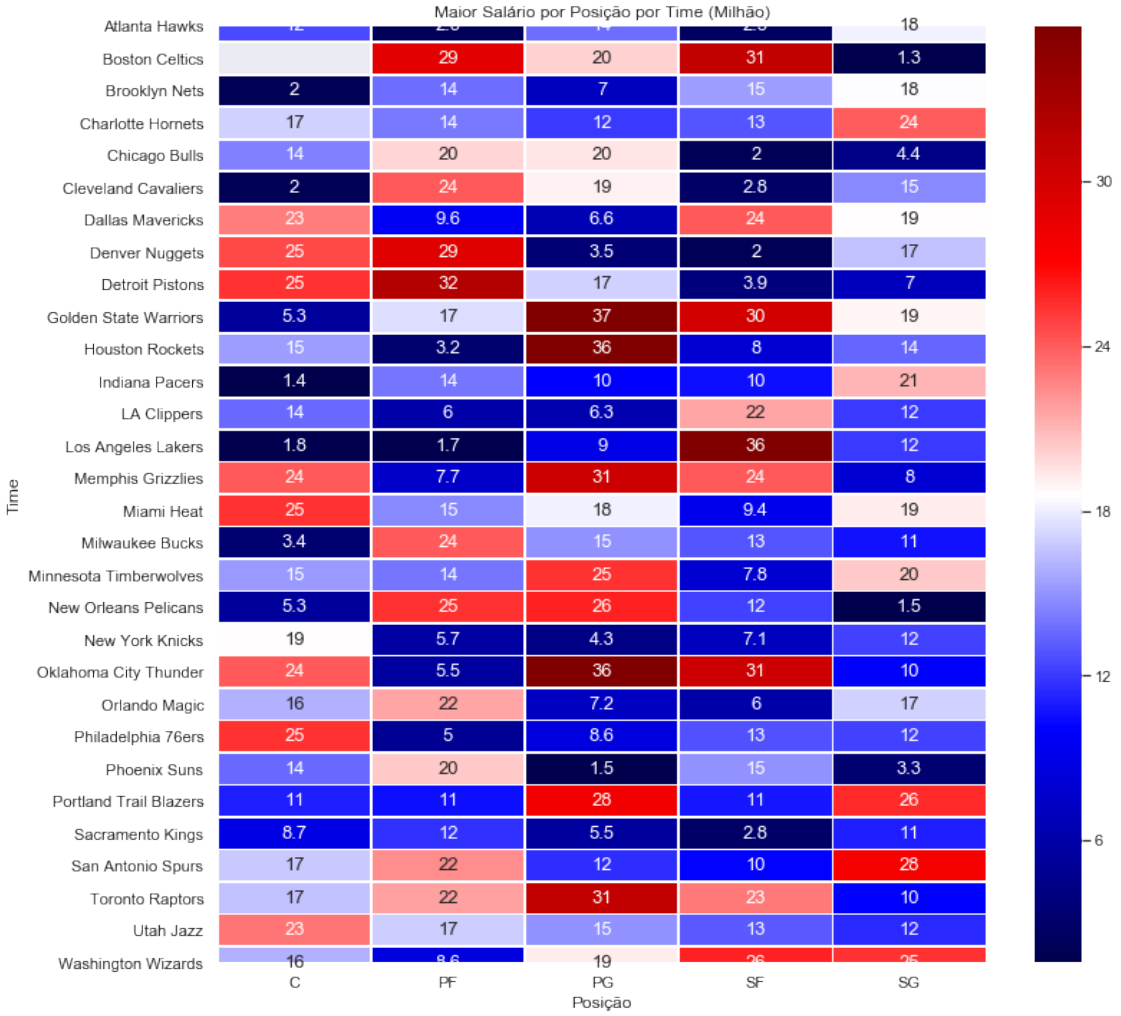



Média de Tocos por Salário - Agrupado por Posição
position = SF
position = SG
position = PG
position = PF
position = C
Salário
Tocos
exp_cat
0-5
6-10
11-15
16+

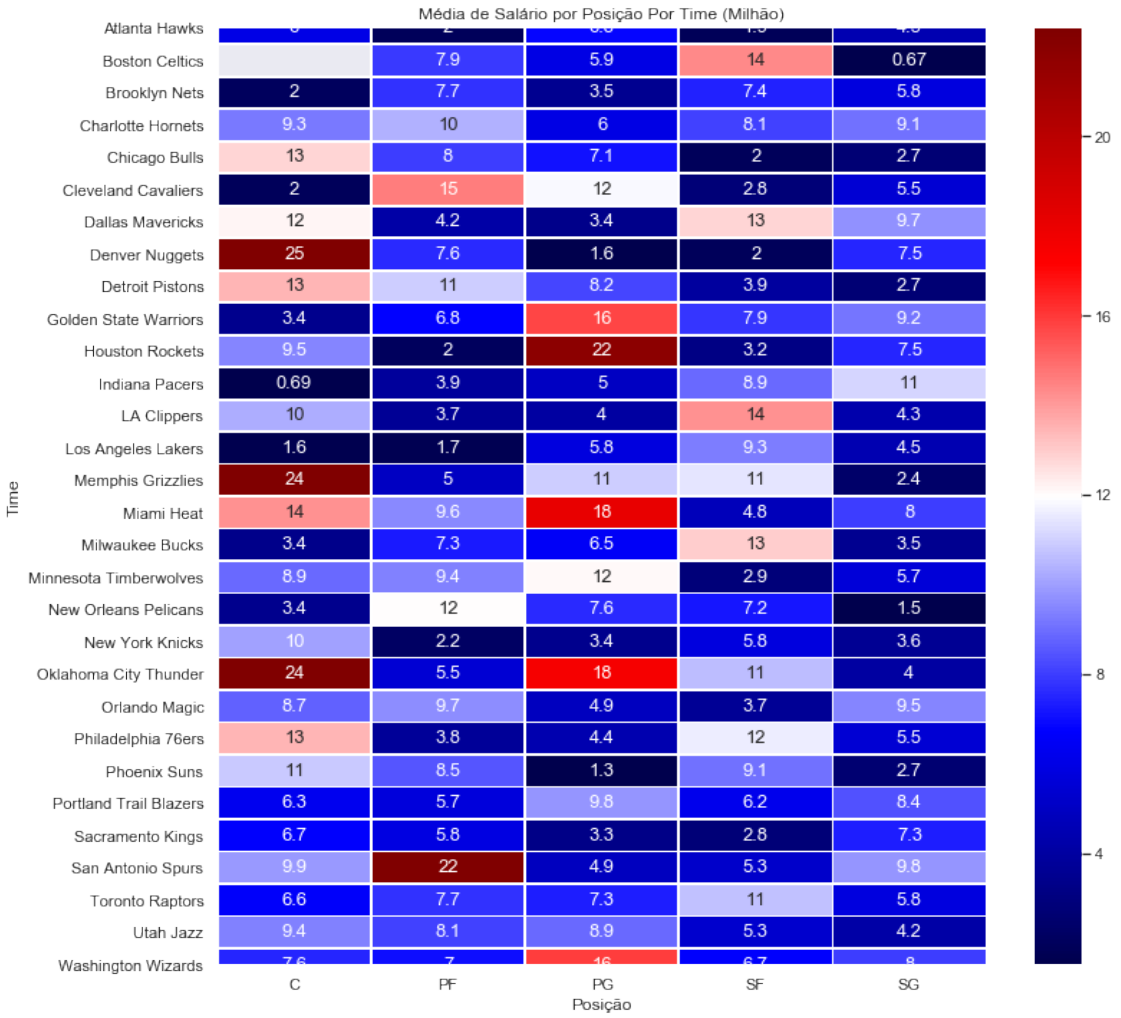

Média de Roubadas por Salário - Agrupado por Posição
position = SF
position = SG
position = PG
position = PF
position = C
Salário
Roubadas
exp_cat
0-5
6-10
11-15
16+



[29]:
```
hightest_salary_per_position_by_team_heatmap(nba)
```



[30]:
```
mean_salary_per_position_by_team_heatmap(nba)
```



[31]:
```
top20_salaries(nba)
```

[31]:

	team	name	salary	position	age	mpg	\
103	Golden State Warriors	Stephen Curry	37.457154	PG	30	34.4	
510	Oklahoma City Thunder	Russell Westbrook	35.654150	PG	29	34.4	
147	Los Angeles Lakers	LeBron James	35.654150	SF	33	38.8	
310	Houston Rockets	Chris Paul	35.654150	PG	33	35.3	
306	Houston Rockets	James Harden	35.650150	PG	29	33.8	
230	Detroit Pistons	Blake Griffin	32.088932	PF	29	35.0	
6	Boston Celtics	Gordon Hayward	31.214295	SF	28	31.3	
89	Toronto Raptors	Kyle Lowry	31.200000	PG	32	31.1	
502	Oklahoma City Thunder	Paul George	30.560700	SF	28	33.4	
320	Memphis Grizzlies	Mike Conley	30.521115	PG	31	32.5	
105	Golden State Warriors	Kevin Durant	30.000000	SF	30	37.1	
467	Denver Nuggets	Paul Millsap	29.230769	PF	33	29.5	
7	Boston Celtics	Al Horford	28.928709	PF	32	33.2	

```
518  Portland Trail Blazers     Damian Lillard  27.977689       PG   28  36.4
357       San Antonio Spurs      DeMar DeRozan  27.739975       SG   29  34.1
452      Washington Wizards    Otto Porter Jr.  26.011913       SF   25  26.4
341    New Orleans Pelicans       Jrue Holiday  25.976111       PG   28  32.8
519  Portland Trail Blazers        CJ McCollum  25.759766       SG   27  29.2
67       Philadelphia 76ers        Joel Embiid  25.467250        C   24  28.7
493  Minnesota Timberwolves     Andrew Wiggins  25.467250       PG   23  36.2

      ppg  apg  stlpg  topg
103  23.1  6.8    1.8   3.2
510  23.0  8.2    1.7   4.0
147  27.2  7.2    1.6   3.5
310  18.7  9.8    2.3   2.4
306  23.0  6.1    1.5   3.5
230  21.5  4.3    0.9   2.5
6    15.6  3.4    1.0   2.0
89   14.4  5.9    1.3   2.2
502  18.6  3.2    1.7   2.6
320  14.3  5.7    1.5   2.0
105  27.1  3.9    1.2   3.1
467  14.2  2.3    1.3   1.8
7    14.2  3.2    0.8   1.6
518  23.1  6.2    0.9   2.8
357  19.7  3.1    1.0   2.0
452  10.5  1.4    1.1   0.7
341  14.9  6.2    1.5   2.7
519  17.2  2.9    0.9   1.8
67   22.0  2.8    0.7   3.7
493  19.7  2.1    1.0   2.1
```

4.2 Analisar correlação

Vamos analisar os valores de cada feature. Se em alguma feature possui valores estranhos.

[32]:
```
corr_matrix = nba.corr()

corr_matrix['salary'].sort_values(ascending=False)
```

[32]:
```
salary             1.000000
ppg_last_season    0.671378
ppg                0.657985
ppg_career         0.657985
mpg                0.607698
topg               0.575782
stlpg              0.552072
apg_last_season    0.539316
gp                 0.527409
```

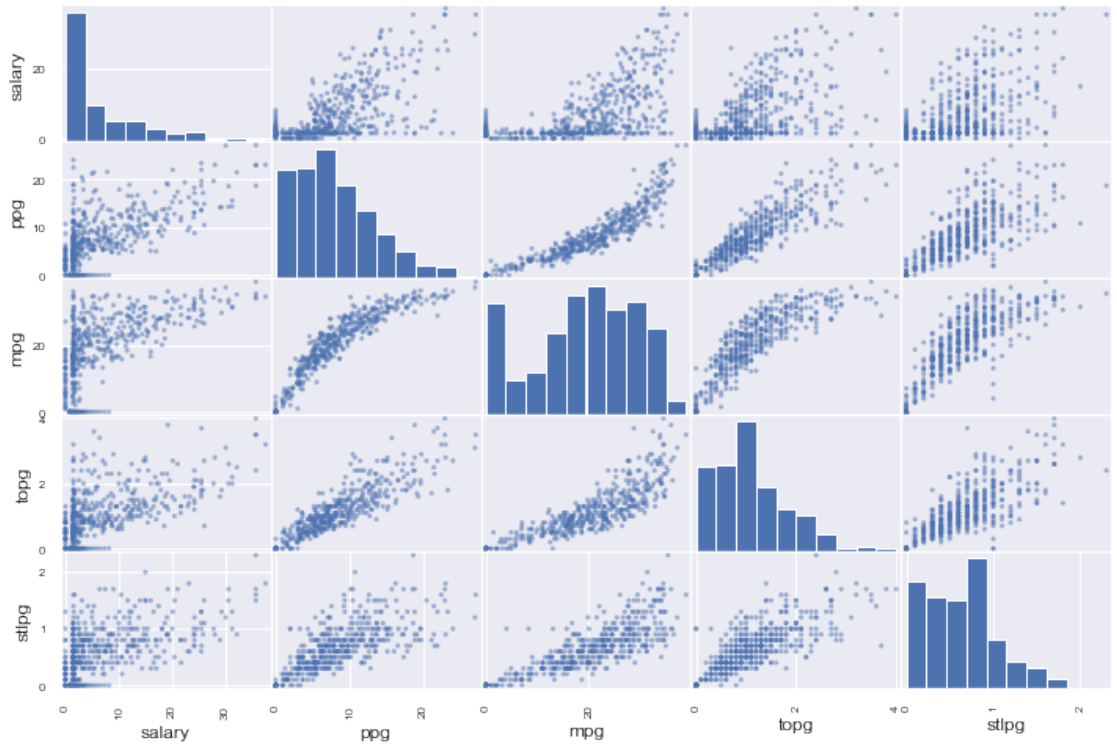
```
apg_career          0.519632
experience          0.486838
apg                 0.481251
blkpg               0.378015
rgp_career          0.372038
rpg_last_season     0.368273
age                 0.357326
per_last_season     0.351810
ftp                 0.290302
fgp                 0.264267
thp                 0.237894
wt                  0.158201
ht                  0.070537
Name: salary, dtype: float64
```

[33]:
```
nba.plot(kind='scatter', figsize=(15, 7),
         x='mpg', y='ppg',
         c='salary', cmap=plt.get_cmap('jet'),
         colorbar=True, alpha=0.3)
```

[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x122c02cd0>
```

[34]:
```
from pandas.plotting import scatter_matrix

attrs = ['salary', 'ppg', 'mpg', 'topg', 'stlpg']
scatter_matrix(nba[attrs], figsize=(12, 8))
```

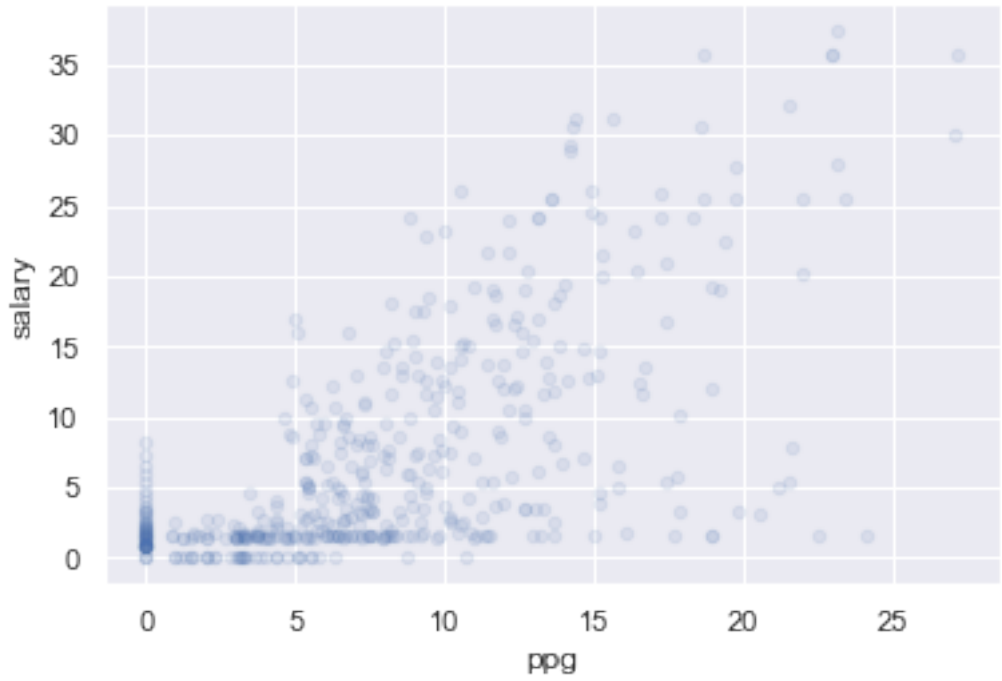
```
[34]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1184cf950>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x118366bd0>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11d262550>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x118075910>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11d195350>],
             [<matplotlib.axes._subplots.AxesSubplot object at 0x11d1d2a50>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11805b410>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11812f990>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11de67250>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x1054ad410>],
             [<matplotlib.axes._subplots.AxesSubplot object at 0x1200ae250>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x121145690>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x122bd0a10>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11c7191d0>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11c680a50>],
             [<matplotlib.axes._subplots.AxesSubplot object at 0x118088dd0>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x122c2da90>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x1181e2b50>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11de4fad0>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x1225bce50>],
             [<matplotlib.axes._subplots.AxesSubplot object at 0x122194b10>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x11e06be90>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x118018b50>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x118040ed0>,
              <matplotlib.axes._subplots.AxesSubplot object at 0x1221ddb90>]],
            dtype=object)
```



[35]:
```
nba.plot(kind='scatter', x='ppg', y='salary', alpha=0.1)
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```

[35]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11deb6190>
```



5 Preparar dados para o algoritmo de ML

[36]:
```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train_index, test_index in split.split(nba, nba['exp_cat']):
    # https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
 ↪html#reindexing
    strat_train_set = nba.loc[nba.index.intersection(train_index)]
    strat_test_set = nba.loc[nba.index.intersection(test_index)]

print(f'Tamanho do conjunto de treino: {strat_train_set.shape[0]}')
print(f'Tamanho do conjunto de teste: {strat_test_set.shape[0]}')

print('')

print(strat_test_set['exp_cat'].value_counts() / len(strat_test_set))

# Removendo a categoria extra para ser possível stratificar nosso dataset
for set_ in (strat_train_set, strat_test_set):
    set_.drop('exp_cat', axis=1, inplace=True)
```

```
Tamanho do conjunto de treino: 323
```

```
Tamanho do conjunto de teste: 83

0-5      0.626506
6-10     0.265060
11-15    0.084337
16+      0.024096
Name: exp_cat, dtype: float64
```

Vamos separar os conjuntos de treino e testes em strats

5.1 Limpar dados

[37]:
```
# Vamos pegar os dados novamente sem nenhuma alteração
nba = strat_train_set.drop('salary', axis=1)
nba_labels = strat_train_set['salary'].copy()

# Separar em dataframes com categorias númericas e categóricas
# Removendo colunas não numéricas
# Queremos somente as colunas numéricas
nba_num = nba.drop(['team', 'name', 'url', 'position', 'college', 'fgm_fga',␣
 ↪'thm_tha', 'ftm_fta'], axis=1)

# colunas categóricas que serão usadas
nba_cat = nba[['team', 'position']]
```

[38]:
```
from sklearn.impute import SimpleImputer

# Vai adicionar 0 nos campos que vazios
fill_imputer = SimpleImputer(strategy='constant', fill_value=0)

# Vai substituir o valor zero pela mediana
median_imputer = SimpleImputer(strategy='median', missing_values=0)

X = fill_imputer.fit_transform(nba_num)
X = median_imputer.fit_transform(X)

nba_tr = pd.DataFrame(X, columns=nba_num.columns, index=nba_num.index)
nba_tr.head(5)
```

[38]:

	experience	age	ht	wt	ppg_last_season	apg_last_season	\
0	6.0	31.0	208.28	117.65	6.0	1.1	
4	1.0	21.0	198.12	92.76	1.0	3.0	
5	2.0	24.0	195.58	102.26	1.4	0.2	
6	8.0	28.0	203.20	101.81	2.0	3.0	
7	11.0	32.0	208.28	110.86	12.9	7.4	

	rpg_last_season	per_last_season	ppg_career	apg_career	rgp_career	\
0	5.4	12.09	5.4	0.7	4.4	

```
4              0.5           -4.82          1.0          3.0          0.5
5              0.4            6.88          1.6          0.2          0.5
6              1.0            7.43         15.6          3.4          4.2
7              1.1           17.63         14.2          8.6          1.2
```

```
     gp   mpg    fgp    thp    ftp  apg  blkpg  stlpg  topg   ppg
0  376.0  15.0  0.502  0.143  0.802  0.7    0.5    0.2   0.8   5.4
4    2.0   1.5  0.500  0.342  0.756  1.4    0.3    0.7   0.5   1.0
5   47.0   5.8  0.418  0.294  0.710  0.2    0.3    0.1   0.1   1.6
6  517.0  31.3  0.444  0.368  0.820  3.4    0.4    1.0   2.0  15.6
7  718.0  33.2  0.525  0.370  0.750  3.2    1.2    0.8   1.6  14.2
```

5.2 Cuidando dos campos de texto/categorias

[39]:
```
# vamos converter categorias de texto para números
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
nba_cat_encoded = cat_encoder.fit_transform(nba_cat)
```

[40]:
```
cat_encoder.categories_
```

[40]:
```
[array(['Atlanta Hawks', 'Boston Celtics', 'Brooklyn Nets',
        'Charlotte Hornets', 'Chicago Bulls', 'Cleveland Cavaliers',
        'Dallas Mavericks', 'Denver Nuggets', 'Detroit Pistons',
        'Golden State Warriors', 'Houston Rockets', 'Indiana Pacers',
        'LA Clippers', 'Los Angeles Lakers', 'Memphis Grizzlies',
        'Miami Heat', 'Milwaukee Bucks', 'New Orleans Pelicans',
        'New York Knicks', 'Orlando Magic', 'Philadelphia 76ers',
        'Phoenix Suns', 'Sacramento Kings', 'San Antonio Spurs',
        'Toronto Raptors', 'Washington Wizards'], dtype=object),
 array(['C', 'PF', 'PG', 'SF', 'SG'], dtype=object)]
```

Vamos criar um pipeline para facilitar os passos necessários que precisam ser executados

[41]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

num_attrs = list(nba_num)
cat_attrs = list(nba_cat)

# pipeline dos atributos numericos
num_pipeline = Pipeline([
    ('fill_imputer', SimpleImputer(strategy='constant', fill_value=0)),
    ('median_imputer', SimpleImputer(strategy='median', missing_values=0)),
    ('std_scaler', StandardScaler())
```

```
])

cat_pipeline = Pipeline([
    ('cat_encoder', OneHotEncoder())
])

full_pipeline = ColumnTransformer([
    ('num_pipeline', num_pipeline, num_attrs),
    ('cat_pipeline', cat_pipeline, cat_attrs)
])

nba_prepared = full_pipeline.fit_transform(nba)
```

[42]:
```
from sklearn.metrics import mean_squared_error

data_test = nba.iloc[:5]
data_test_labels = nba_labels[:5]

data_test_prepared = full_pipeline.transform(data_test)
```

[43]:
```
# EXEMPLO DE UNDERFITTING?
from sklearn.linear_model import LinearRegression

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

nba_predictions = lin_reg.predict(data_test_prepared)

print(f'Predictions: {nba_predictions}')
print(f'Labels: {list(data_test_labels)}')

lin_mse = mean_squared_error(data_test_labels, nba_predictions)
lin_rsme = np.sqrt(lin_mse)

print(f'RSME: {lin_rsme}')
```

```
Predictions: [ 4.56359863  0.28820801 -0.18530273 14.26464844 18.78198242]
Labels: [5.1936, 0.0, 0.0, 31.214295, 28.928709]
RSME: 8.84037195514633
```

[44]:
```
# EXEMPLO DE OVERFITTING?
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(nba_prepared, nba_labels)

nba_predictions_2 = tree_reg.predict(data_test_prepared)
```

```
print(f'Predictions: {nba_predictions_2}')
print(f'Labels: {list(data_test_labels)}')

tree_mse = mean_squared_error(data_test_labels, nba_predictions_2)
tree_rsme = np.sqrt(tree_mse)

print(f'RSME: {tree_rsme}')
```

```
Predictions: [ 5.1936    0.        0.       31.214295 28.928709]
Labels: [5.1936, 0.0, 0.0, 31.214295, 28.928709]
RSME: 0.0
```

[45]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()
forest_reg.fit(nba_prepared, nba_labels)

nba_predictions_3 = forest_reg.predict(data_test_prepared)

print(f'Predictions: {nba_predictions_3}')
print(f'Labels: {list(data_test_labels)}')

forest_mse = mean_squared_error(data_test_labels, nba_predictions_3)
forest_rmse = np.sqrt(forest_mse)

print(f'RSME: {forest_rmse}')
```

```
Predictions: [ 5.1936     0.         0.6356204 17.1175634 26.529695 ]
Labels: [5.1936, 0.0, 0.0, 31.214295, 28.928709]
RSME: 6.4012049376274565
```

```
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

5.3 Usando cross-validation

A ideia é analisar qual o modelo apresenta melhor expectativa para ser utilizado como nosso modelo final.

[46]:
```
def display_scores(scores):
    print(f'Scores: {scores}')
    print(f'Mean: {scores.mean()}')
    print(f'Standard Deviation: {scores.std()}')
```

[47]:
```
from sklearn.model_selection import cross_val_score

# Cross Validation usando a árvore de decisão
tree_scores = cross_val_score(tree_reg,
                          nba_prepared, nba_labels,
                          scoring='neg_mean_squared_error',
                          cv=10)

tree_rmse_scores = np.sqrt(-tree_scores)

print('Árvore de decisão:')
display_scores(tree_rmse_scores)
print('-' * 70 + '\n')

# Cross Validation usando regressão linear
lin_reg_scores = cross_val_score(lin_reg,
                                 nba_prepared, nba_labels,
                                 scoring='neg_mean_squared_error',
                                 cv=10)

lin_reg_rmse_scores = np.sqrt(-lin_reg_scores)
print('Regressão linear:')
display_scores(lin_reg_rmse_scores)
print('-' * 70 + '\n')

# Cross Validation usando a floresta aleatórias
forest_scores = cross_val_score(forest_reg,
                                nba_prepared, nba_labels,
                                scoring='neg_mean_squared_error',
                                cv=10)

forest_rmse_scores = np.sqrt(-forest_scores)
print('Florestas Aleatórias:')
display_scores(forest_rmse_scores)
print('-' * 70 + '\n')
```

```
Árvore de decisão:
Scores: [3.57507994 7.27237962 5.06371998 3.49907171 7.06561995 6.52071773
 9.13963013 7.51800389 6.94867555 6.18828015]
Mean: 6.279117865875032
Standard Deviation: 1.6838979395011766
----------------------------------------------------------------------

Regressão linear:
Scores: [5.56125769e+11 5.13004777e+11 1.00732539e+11 1.04232410e+12
 5.89073236e+00 1.42493961e+12 1.05446980e+12 4.14619824e+10
 7.98213196e+11 3.12927039e+12]
```

```
Mean: 866054215875.0836
Standard Deviation: 879891658905.8341
------------------------------------------------------------------------

Florestas Aleatórias:
Scores: [5.12675887 5.0965817  3.92850941 4.32315549 5.51383977 4.60178764
 7.20677244 4.45849796 5.07328264 6.52554737]
Mean: 5.185473329371152
Standard Deviation: 0.9594759727725465
------------------------------------------------------------------------
```

5.4 Sintonia fina do modelo

5.4.1 GridSearch

[48]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

params_grid = [
    { 'n_estimators': [3, 10, 30], 'max_features': [4, 5, 6, 7, 8]},
    { 'n_estimators': [3, 5, 10], 'max_features': [4, 5, 6], 'bootstrap':␣
 ↪[False]}
]

forest_reg = RandomForestRegressor()

# Round 1 - 5 * 3 = 15
# Round 2 - 3 * 3 = 9
# Total = 24 * 5(quantidade de cv) = 120 combinações
grid_search = GridSearchCV(forest_reg, params_grid, cv=10)

grid_search.fit(nba_prepared, nba_labels)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_search.py:814:
DeprecationWarning: The default of the `iid` parameter will change from True to
False in version 0.22 and will be removed in 0.24. This will change numeric
results when test-set sizes are unequal.
  DeprecationWarning)
```

[48]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
```

```
                                       min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators='warn', n_jobs=None,
                                       oob_score=False, random_state=None,
                                       verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'max_features': [4, 5, 6, 7, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [4, 5, 6],
                          'n_estimators': [3, 5, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

5.5 Testando Árvores Aleatórias

5.5.1 Usando os parâmetros encontrados pelo GridSearch

[49]:
```
data_test = nba.iloc[:5]
data_test_labels = nba_labels[:5]

data_test_prepared = full_pipeline.transform(data_test)

final_model = grid_search.best_estimator_
nba_predictions_f = final_model.predict(data_test_prepared)

print(f'Predictions: {nba_predictions_f}')
print(f'Labels: {list(data_test_labels)}')

final_model_mse = mean_squared_error(data_test_labels, nba_predictions_f)
final_model_rmse = np.sqrt(final_model_mse)

print(f'RSME: {final_model_rmse}')
```

```
Predictions: [ 6.56796383  0.13590027  0.26372877 25.20826007 26.96491707]
Labels: [5.1936, 0.0, 0.0, 31.214295, 28.928709]
RSME: 2.8950244236989673
```

5.5.2 RandomizedSearch

[50]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
    'n_estimators': randint(low=1, high=200),
    'max_features': randint(low=1, high=10)
}
```

```
forest_reg = RandomForestRegressor()

rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=15, cv=10,␣
↪scoring='neg_mean_squared_error',
                                random_state=42)

rnd_search.fit(nba_prepared, nba_labels)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_search.py:814:
DeprecationWarning: The default of the `iid` parameter will change from True to
False in version 0.22 and will be removed in 0.24. This will change numeric
results when test-set sizes are unequal.
  DeprecationWarning)
```

[50]:
```
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators='warn',
                                                   n_jobs=None, oob_score=False,
                                                   random_st…
                                                   warm_start=False),
                   iid='warn', n_iter=15, n_jobs=None,
                   param_distributions={'max_features':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x11eb28d90>,
                                        'n_estimators':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x11eb28b90>},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

5.6 Testando Árvores Aleatórias

5.6.1 Usando os parâmetros encontrados pelo RandomizedSearch

[51]:
```
data_test = nba.iloc[:5]
data_test_label = nba_labels.iloc[:5]

data_test_prepared = full_pipeline.transform(data_test)
```

```
final_model = rnd_search.best_estimator_
final_model.fit(nba_prepared, nba_labels)
nba_predcitions_f_2 = final_model.predict(data_test_prepared)

print(f'Predictions: {nba_predcitions_f_2}')
print(f'Labels: {list(data_test_labels)}')

final_model_mse = mean_squared_error(data_test_labels, nba_predcitions_f_2)
final_model_rmse = np.sqrt(final_model_mse)

print(f'RSME: {final_model_rmse}')
```

```
Predictions: [ 5.61378296  0.32067446  0.44243049 24.51912922 25.06424902]
Labels: [5.1936, 0.0, 0.0, 31.214295, 28.928709]
RSME: 3.4708623235413416
```

[52]:
```
# Importância das funcionalidades
features_importances = grid_search.best_estimator_.feature_importances_

cat_encoder = full_pipeline.named_transformers_['cat_pipeline'].
 ↪named_steps['cat_encoder']
cat_encoder_categories = cat_encoder.categories_
cat_enconder_attrs = list(cat_encoder_categories[0]) +␣
 ↪list(cat_encoder_categories[1])

attrs = list(num_attrs) + cat_enconder_attrs

for importance, attr in sorted(zip(features_importances, attrs), reverse=True):
    print(f'{attr} - {importance * 100: .2f}%')
```

```
ppg_last_season -  12.36%
ppg -  11.07%
ppg_career -  7.98%
experience -  7.07%
gp -  6.40%
per_last_season -  5.42%
mpg -  5.15%
topg -  4.70%
apg -  4.30%
age -  4.07%
stlpg -  4.01%
apg_career -  3.90%
apg_last_season -  3.52%
rpg_last_season -  2.50%
fgp -  2.30%
blkpg -  2.27%
```

```
thp -  2.10%
wt -  1.79%
ftp -  1.73%
rgp_career -  1.68%
ht -  1.58%
PG -  0.39%
Washington Wizards -  0.37%
Memphis Grizzlies -  0.35%
Chicago Bulls -  0.33%
Houston Rockets -  0.27%
SF -  0.24%
Phoenix Suns -  0.20%
SG -  0.19%
PF -  0.18%
C -  0.17%
Detroit Pistons -  0.16%
Indiana Pacers -  0.14%
Boston Celtics -  0.14%
Toronto Raptors -  0.13%
Orlando Magic -  0.11%
Miami Heat -  0.08%
Charlotte Hornets -  0.06%
New Orleans Pelicans -  0.06%
Philadelphia 76ers -  0.06%
LA Clippers -  0.05%
New York Knicks -  0.05%
Milwaukee Bucks -  0.05%
Cleveland Cavaliers -  0.05%
Brooklyn Nets -  0.05%
Dallas Mavericks -  0.05%
Denver Nuggets -  0.04%
Atlanta Hawks -  0.04%
San Antonio Spurs -  0.04%
Sacramento Kings -  0.02%
Los Angeles Lakers -  0.02%
Golden State Warriors -  0.01%
```

Podemos notar que a posição do jogador e/ou o time que ele joga tem muita pouca importância para determinar seu salário.

Como a resposta para o trabalho é se o salário da temporada é justo, eu não vou considerar as estatísticas da temporada passada e nem sua média durante a carreira.

Depois será feito um teste sem os jogadores que receberam algum salário, mas não fizeram nenhum ponto, assistência e etc. Pode ser um jogador que recebeu o salário, se machucou e ficou fora da temporada, como pode ser dados inexatos.

[53]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.compose import ColumnTransformer

num_attrs = ['experience', 'age', 'gp', 'fgp', 'thp', 'ftp', 'apg', 'blkpg',␣
 ↪'stlpg', 'topg', 'ppg']

def get_num_column_pipeline():
    # pipeline dos atributos numericos
    num_pipeline = Pipeline([
        ('fill_imputer', SimpleImputer(strategy='constant', fill_value=0)),
        ('median_imputer', SimpleImputer(strategy='median', missing_values=0)),
        ('std_scaler', StandardScaler())
    ])

    full_pipeline = ColumnTransformer([
        ('num_pipeline', num_pipeline, num_attrs)
    ])

    return full_pipeline
```

5.6.2 RandomizedSearch

[54]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

nba_prepared = get_num_column_pipeline().fit_transform(nba)

param_distribs = {
    'n_estimators': randint(low=1, high=200),
    'max_features': randint(low=1, high=10)
}

forest_reg = RandomForestRegressor()

rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=15, cv=10,␣
 ↪scoring='neg_mean_squared_error',
                                random_state=42)

rnd_search.fit(nba_prepared, nba_labels)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_search.py:814:
DeprecationWarning: The default of the `iid` parameter will change from True to
False in version 0.22 and will be removed in 0.24. This will change numeric
results when test-set sizes are unequal.
  DeprecationWarning)
```

[54]:
```
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators='warn',
                                                   n_jobs=None, oob_score=False,
                                                   random_st…
                                                   warm_start=False),
                   iid='warn', n_iter=15, n_jobs=None,
                   param_distributions={'max_features':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x11eb3ed50>,
                                        'n_estimators':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x11eb09f90>},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

5.7 Testando Árvores Aleatórias

5.7.1 Utilzando o melhor modelo encontrado pelo RandomizedSearch

[55]:
```
def test_data_test():
    print(f'NBA Shape: {nba.shape}')
    full_pipeline = get_num_column_pipeline()
    nba_prepared = full_pipeline.fit_transform(nba)

    data_test = nba.iloc[:5]
    data_test_labels = nba_labels.iloc[:5]

    data_test_prepared = full_pipeline.transform(data_test)

    final_model = rnd_search.best_estimator_
    final_model.fit(nba_prepared, nba_labels)

    nba_predictions = final_model.predict(data_test_prepared)

    print(f'Predictions: {nba_predictions}')
    print(f'Labels: {list(data_test_labels)}')

    final_model_mse = mean_squared_error(data_test_labels, nba_predictions)
```

```
    final_model_rmse = np.sqrt(final_model_mse)

    print(f'RSME: {final_model_rmse}')

# Testando antes de remover os jogadores que não contribuíram
test_data_test()
```

```
NBA Shape: (323, 29)
Predictions: [ 6.22226754  0.51693788  0.36693528 27.49760038 24.0461203 ]
Labels: [5.1936, 0.0, 0.0, 31.214295, 28.928709]
RSME: 2.7969095273838445
```

Melhorou o nosso erro, mas ainda vamos testar removendo os jogadores que não tiveram nenhum estatística durante o ano (gp/ppg/apg).

[56]:
```
# Filtro para remover
drop_index = nba[
    (nba['ppg'] == 0)
    & (nba['gp'] == 0)
    & (nba['apg'] == 0)
].index

# Removendo os jogadores
nba.drop(drop_index, axis=0, inplace=True)

# Removendo os salários
nba_labels.drop(drop_index, axis=0, inplace=True)

# Testando novamente
test_data_test()
```

```
NBA Shape: (291, 29)
Predictions: [ 5.64223255  0.47153481  0.48183596 26.36100652 25.9946519 ]
Labels: [5.1936, 0.0, 0.0, 31.214295, 28.928709]
RSME: 2.5619868185085934
```

Remover 79 jogadores que não possui estatísticas não melhorou tanto o nosso modelo. Tem momentos que o erro é menor e outras que são maiores.

5.8 Teste final

Vamos utilizar o conjunto de teste para avaliar

[57]:
```
final_model = rnd_search.best_estimator_

# Um jogador estrela
curry_idx = nba_bkp[nba_bkp['name'] == 'Stephen Curry'].index
curry = nba_bkp.loc[curry_idx]
curry_feature = curry.drop('salary', axis=1)
```

```
curry_label = curry['salary'].copy().iloc[0]

# Um jogador sem ser estrela
abaynes = nba_bkp.loc[[0]]
abaynes_feature = abaynes.drop('salary', axis=1)
abaynes_label = abaynes['salary'].copy().iloc[0]

X_test = strat_test_set.drop('salary', axis=1)
y_test = strat_test_set['salary'].copy()

pipeline = get_num_column_pipeline()
pipeline.fit(nba)

X_test_prepared = pipeline.transform(X_test)

predicted = final_model.predict(X_test_prepared)

final_model_mse = mean_squared_error(y_test, predicted)
final_model_rmse = np.sqrt(final_model_mse)

print(f'RMSE do modelo final: {final_model_rmse: .2f}M\n')

pipeline.fit(nba)
tst_features_prepared = pipeline.transform(pd.concat([curry_feature,␣
 ↪abaynes_feature]))

tst_predicted = final_model.predict(tst_features_prepared)

curry_predicted = tst_predicted[0]
abaynes_predicted = tst_predicted[1]

print(f'Salário do Stephen Curry deveria ser: {curry_predicted:.1f}M\n')
print(f'Salário do Stephen Curry é: {curry_label:.1f}M\n')
print(f'Diferença: {curry_label - curry_predicted: .1f}M\n')
print(f'Erro de {(100 - (curry_predicted * 100) / curry_label): .1f}%')

print('')
print('*' * 40)
print('')

print(f'Salário do Aron Baynes deveria ser: {abaynes_predicted:.1f}M\n')
print(f'Salário do Aron Baynes é: {abaynes_label:.1f}M\n')
print(f'Diferença: {abaynes_label - abaynes_predicted: .1f}M\n')
print(f'Erro de {(100 - (abaynes_predicted * 100) / abaynes_label): .1f}%')
```

```
RMSE do modelo final:  5.67M
```

```
Salário do Stephen Curry deveria ser: 33.1M

Salário do Stephen Curry é: 37.5M

Diferença:  4.3M

Erro de  11.5%

**************************************

Salário do Aron Baynes deveria ser: 5.6M

Salário do Aron Baynes é: 5.2M

Diferença: -0.4M

Erro de -8.6%
```

6 Conclusão

É sabido que vários fatores externos determinam o salário de um jogador, principalmente se ele for uma estrela. Um jogador de grande repercussão traz patrocinadores, mais telespectadores, maior público, vende mais camisas e entre outros...

Isso pode fazer com que um algoritmo de aprendizado de máquina, que só analisa seu desempenho em quadra não consiga prever se seu salário é compatível ou não.

O algoritmo utilizado nesse trabalho chegou bem perto do salário recebido pelo Stephen Curry. Eu diria que seu salário é justo, levando em consideração apenas suas estatística dentro de quadra.