NBA Salary Exploratory Data Analysis

Brian Fernando Aarush Maddela Nixon Tan
Sharona Yang

Wednesday 17th December, 2025

In this notebook, we'll do some EDA on the dataset containing NBA salary and stats data.

0.1 Importing the data

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

data_path_2024 = os.getcwd()+'/data/final_2024_player.csv'
data_path_2025 = os.getcwd()+'/data/final_2025_player.csv'

df_2024 = pd.read_csv(data_path_2024)
df_2025 = pd.read_csv(data_path_2025)

df = pd.concat([df_2024, df_2025])

df.head()
```

	Rk_x	Player	Age	Team	Pos	G	GS	MP_x	FG	FGA	...	SecondTeam	ThirdTeam	DefTeam1	DefTeam2	Salary	Guaranteed	2023-24_contract_year	2024-25_contract_year	Next_Year_Salary	Next_Year_Guaranteed
0	363.0	A.J. Green	24.0	MIL	SG	56.0	0.0	11.0	1.5	3.5	...	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
1	476.0	AJ Griffin	20.0	ATL	SF	20.0	0.0	8.6	0.9	3.1	...	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2	109.0	Aaron Gordon	28.0	DEN	PF	73.0	73.0	31.5	5.5	9.8	...	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	278.0	Aaron Holiday	27.0	HOU	PG	78.0	1.0	16.3	2.4	5.3	...	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
4	136.0	Aaron Nesmith	24.0	IND	SF	72.0	47.0	27.7	4.4	8.8	...	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

5 rows × 67 columns

0.2 Null Values

```
null = df.isnull().sum()
```

```
null_dict = dict(zip(null.index, null.values))
```

```
[key for key, value in null_dict.items() if value != 0]
```

```
['3P%', 'FT%', 'Awards', '2023 -24_contract_year', '2024 -25_contract_year']
```

In this analysis, we won't be using any of these columns, so we can safely drop them.

```
df = df.drop(['3P%', 'FT%', 'Awards', '2023 -24_contract_year', '2024 -25_contract_year'], a
```

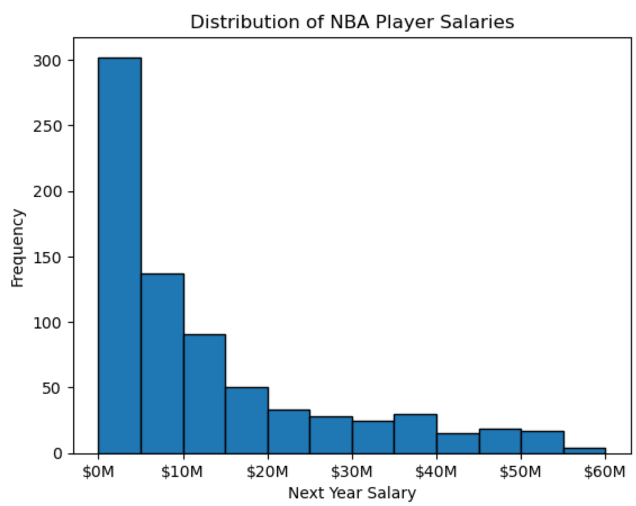
```
df.columns
```

```
Index(['Rk_x', 'Player', 'Age', 'Team', 'Pos', 'G', 'GS', 'MP_x', 'FG', 'FGA',
       'FG%', '3P', '3PA', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'ORB',
       'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS', 'PER', 'TS%',
       '3PAr', 'FTr', 'ORB%', 'DRB%', 'TRB%', 'AST%', 'STL%', 'BLK%', 'TOV%',
       'USG%', 'OWS', 'DWS', 'WS', 'WS/48', 'OBPM', 'DBPM', 'BPM', 'VORP',
       'Season', 'Experience', 'NumOfAwards', 'All -Star', 'AwardWinner',
       'FirstTeam', 'SecondTeam', 'ThirdTeam', 'DefTeam1', 'DefTeam2',
       'Salary', 'Guaranteed', 'Next_Year_Salary', 'Next_Year_Guaranteed'],
      dtype='object')
```

1 Analysis

1.1 NBA Salary Distribution

```
bins = np.arange(0, df['Next_Year_Salary'].max()+5000000, 5000000)

plt.hist(df['Next_Year_Salary'], bins = bins, edgecolor = 'black')
ax = plt.gca()

ax.xaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)

plt.xlabel('Next Year Salary')
plt.ylabel('Frequency')
plt.title('Distribution of NBA Player Salaries')

plt.savefig("outputs/salary_distribution.png")
```



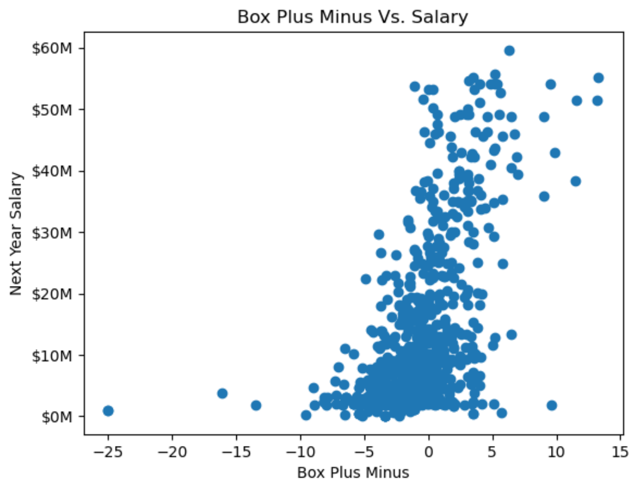
The distribution of NBA salaries is clearly right skewed. Most players earn below 5 million dollars. This makes sense, because there are very few “superstars” who are deserving of a salary above 30 million, and majority of NBA players are role players or depth pieces that aren’t expected to make much relative to others in their careers.

1.2 Salary and Stats (BPM, VORP)

```
plt.scatter(df['BPM'], df['Next_Year_Salary'])
ax = plt.gca()
```

```
ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)
plt.xlabel("Box Plus Minus")
plt.ylabel("Next Year Salary")
plt.title("Box Plus Minus Vs. Salary")
```

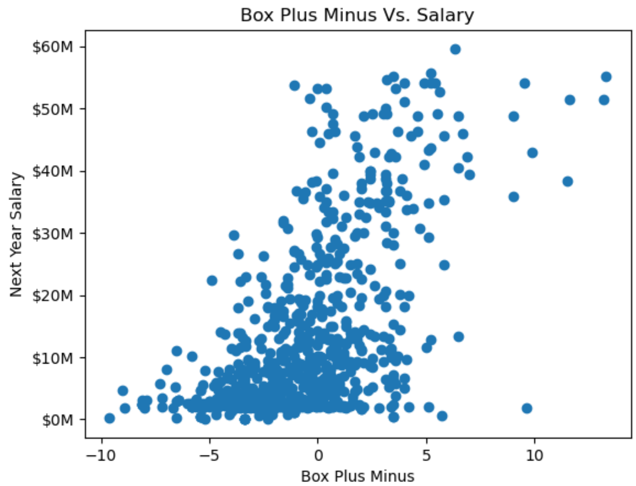
```
Text(0.5, 1.0, 'Box Plus Minus Vs. Salary')
```



Box plus minus is a popular basketball advanced statistic that estimates a player's contribution to the team's performance, both offensively and defensively. A higher box plus minus score means that a player is contributing more towards their team's success. It seems like there is a general positive trend in that players with higher BPM scores can expect to see higher salaries. However, there are a couple outliers, so we'll subset the dataset to only include players with BPM scores higher than -10 and replot this relationship.

```
subset = df[df['BPM'] > -10]

plt.scatter(subset['BPM'], subset['Next_Year_Salary'])
ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)
plt.xlabel("Box Plus Minus")
plt.ylabel("Next Year Salary")
plt.title("Box Plus Minus Vs. Salary")
```

```
Text(0.5, 1.0, 'Box Plus Minus Vs. Salary')
```




```
print(f"Correlation coefficient: {df['BPM'].corr(df['Next_Year_Salary'])}")

print(f"Subsetted Correlation coefficient: {subset['BPM'].corr(subset['Next_Year_Salary'])}'
```

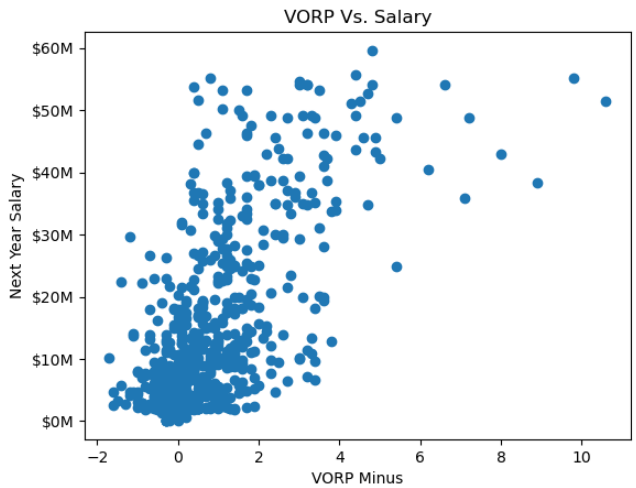
```
Correlation coefficient: 0.5565013579110638
Subsetted Correlation coefficient: 0.5871708693441092
```

After removing the outliers, the scatterplot gives a better view of the relationship between BPM and NBA salaries. The correlation coefficient confirms that there is a moderately strong positive association between the two.

Let's do the same with VORP now

```
plt.scatter(df['VORP'], df['Next_Year_Salary'])
ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)
plt.xlabel("VORP Minus")
plt.ylabel("Next Year Salary")
plt.title("VORP Vs. Salary")
```

```
Text(0.5, 1.0, 'VORP Vs. Salary')
```



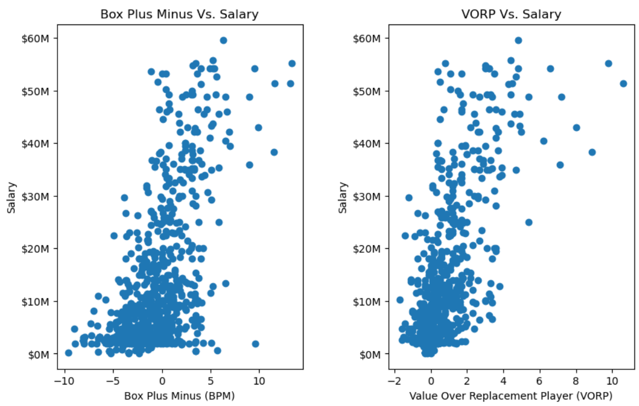

```
print(f"Correlation coefficient: {df['VORP'].corr(df['Next_Year_Salary'])}")
```

```
Correlation coefficient: 0.6744527184663939
```

VORP is another popular advanced stat that estimates a player's overall contribution to a team compared to an average NBA player. It shows the same positive association with NBA salaries as BPM did; however, the correlation coefficient does show a stronger linear relationship than with BPM.

I'll combine both plots into one figure

```
fig, axs = plt.subplots(nrows = 1, ncols = 2, figsize = (10,6))

plt.subplots_adjust(wspace=0.35)

axs[0].scatter(subset['BPM'], subset['Next_Year_Salary'])
axs[1].scatter(df['VORP'], df['Next_Year_Salary'])

axs[0].set_xlabel("Box Plus Minus (BPM)")
axs[0].set_ylabel("Salary")
axs[0].set_title("Box Plus Minus Vs. Salary")

axs[1].set_xlabel("Value Over Replacement Player (VORP)")
axs[1].set_ylabel("Salary")
axs[1].set_title("VORP Vs. Salary")

formatter = plt.FuncFormatter(lambda y, _: f"${y/1e6:.0f}M")

for ax in axs:
    ax.yaxis.set_major_formatter(formatter)

plt.savefig("outputs/bpm_vorp_vs_salary.png")
```

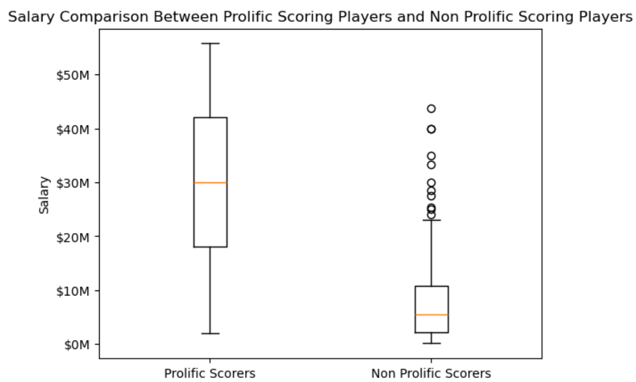


1.3 Offensive Metrics vs. Defensive Metrics and Their Impacts on Salaries

Let's take a look at how offensive and defensive metrics relate to NBA salaries

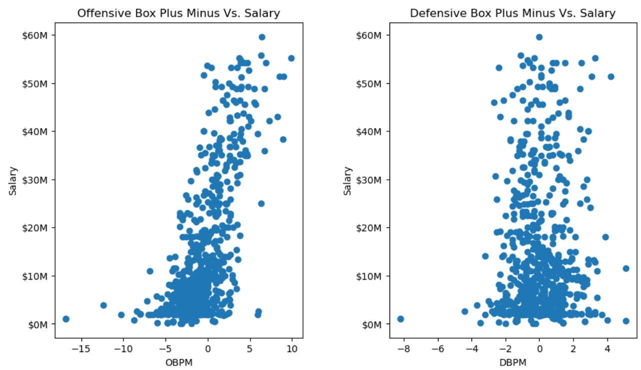
```
good_scoring = df[(df['FGA'] >= 10) & (df['FG%'] >= 0.45)]
bad_scoring = df[~(df['FGA'] >= 10) & (df['FG%'] >= 0.45)]

plt.boxplot([good_scoring['Next_Year_Salary'], bad_scoring['Next_Year_Salary']], labels = ['

ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)

plt.ylabel('Salary')
plt.title('Salary Comparison Between Prolific Scoring Players and Non Prolific Scoring Playe

plt.savefig("outputs/scoring_volume_salary_comparison.png")
```

```
/tmp/ipykernel_1497/3721848652.py:4: MatplotlibDeprecationWarning: The 'labels' parameter of
  plt.boxplot([good_scoring['Next_Year_Salary'], bad_scoring['Next_Year_Salary']], labels =
```



This plot shows that there is quite a big difference in salaries between efficient scorers in the league and those who aren't. Moreover, the boxplot for prolific scorers is about normally distributed, while the boxplot for nonprolific scorers is definitely right skewed. This shows that most players who aren't efficiently scoring for their teams earn significantly less than those that do. This is probably a big factor that goes into determining a player's salary, which makes sense, as basketball is primarily an offensively minded sport.

```
fig, axs = plt.subplots(nrows = 1, ncols = 2, figsize=(11,6))

plt.subplots_adjust(wspace=0.35)

axs[0].scatter(df['OBPM'], df['Next_Year_Salary'])
axs[1].scatter(df['DBPM'], df['Next_Year_Salary'])

axs[0].set_xlabel("OBPM")
axs[0].set_ylabel("Salary")
axs[0].set_title("Offensive Box Plus Minus Vs. Salary")

axs[1].set_xlabel("DBPM")
axs[1].set_ylabel("Salary")
axs[1].set_title("Defensive Box Plus Minus Vs. Salary")

formatter = plt.FuncFormatter(lambda y, _: f"${y/1e6:.0f}M")

for ax in axs:
    ax.yaxis.set_major_formatter(formatter)

plt.savefig("outputs/obpm_dbpm_salary_comp.png")
```



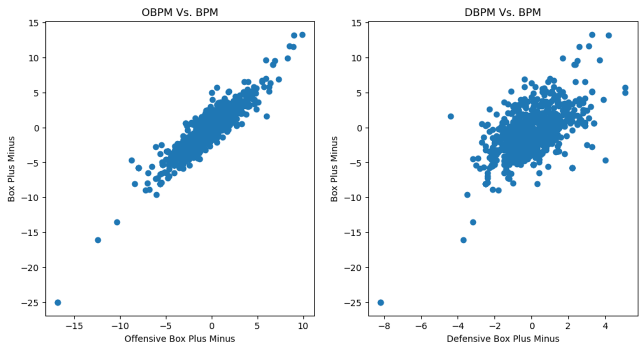

```
print(f"OBPM Correlation coefficient: {df['OBPM'].corr(df['Next_Year_Salary'])}")
print(f"DBPM Correlation coefficient: {df['DBPM'].corr(df['Next_Year_Salary'])}")
```

```
OBPM Correlation coefficient: 0.6401944026484849
DBPM Correlation coefficient: 0.05995507005883653
```

It's difficult to group players into defensively good and bad categories using defensive stats, as they aren't as accurate as offensive stats in evaluating a player's effectiveness. Instead, we used defensive box plus minus, which is essentially the same as box plus minus, but it evaluates a player's defensive efficiency/potential. We compare this to its counterpart, offensive box plus minus. The scatterplots and correlation coefficients both show that there isn't much of a relationship between defensive BPM and salaries, whereas, there is a moderately strong positive correlation between offensive BPM and salaries.

The last two plots show that offensive potential is more valued than defensive potential in evaluating a player's worth to a team through their salary. There is a clear salary difference between good offensive players and more average players, while there is not much of a relationship between defensive stats and NBA salaries.

```
fig, axs = plt.subplots(nrows = 1, ncols = 2, figsize = (12, 6))

axs[0].scatter(df['OBPM'], df['BPM'])
axs[1].scatter(df['DBPM'], df['BPM'])

axs[0].set_xlabel("Offensive Box Plus Minus")
axs[0].set_ylabel("Box Plus Minus")

axs[1].set_xlabel("Defensive Box Plus Minus")
axs[1].set_ylabel("Box Plus Minus")

axs[0].set_title("OBPM Vs. BPM")
axs[1].set_title("DBPM Vs. BPM")

plt.savefig("outputs/obpm_dbpm_vs_bpm.png")
```

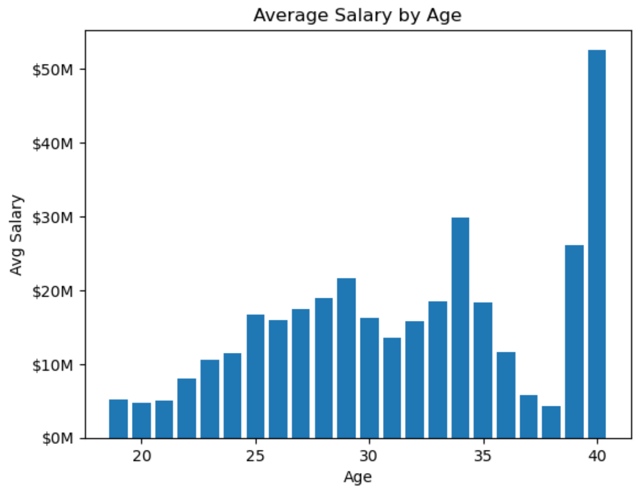


Overall BPM is the sum of Offensive BPM and Defensive BPM. Our earlier analysis shows that there is a higher correlation between offensive stats and salaries than for defensive stats. We also showed a general positive association between BPM and salaries. Out of curiosity, I wanted to see the relationship between OBPM/DBPM and general BPM. There is clearly a stronger positive relationship between OBPM and BPM than for DBPM, which aligns with our earlier analysis between stats and salaries.

From a salary standpoint, it seems like an offensive superstar is worth more than a defensive superstar.

1.4 Salary In Relation to Experience and Awards

```
age_salary = df.groupby('Age')['Next_Year_Salary'].mean()

ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)

plt.bar(age_salary.index, age_salary.values)
plt.xlabel("Age")
plt.ylabel("Avg Salary")
plt.title("Average Salary by Age")
plt.show()
```

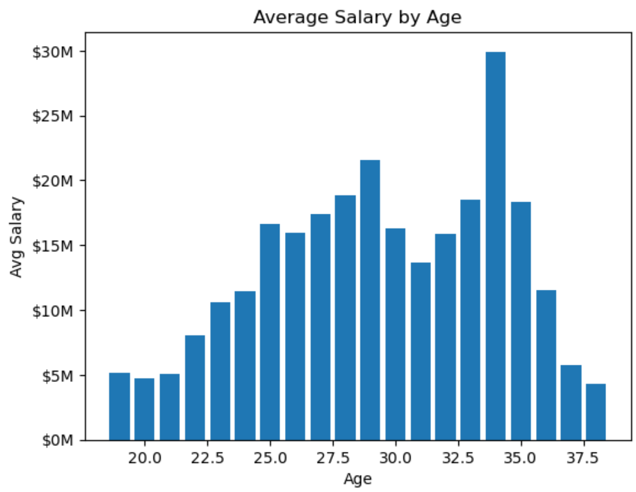



```
df[df['Age'] == 40]
```

	Rk	Player	Age	Team	Pos	G	GS	MP	FG_x	FGA	...	Awards	First	Second	Third	DPOY	Def	Salary	Guaranteed	Next_Year_Salary	Next_Year_Salary_Guaranteed
201	19.0	LeBron James	40.0	LAL	SF	70.0	70.0	34.9	9.3	18.1	...	0	0	1	0	0	0	4872[illegible]	[illegible]	[illegible]	[illegible]7153

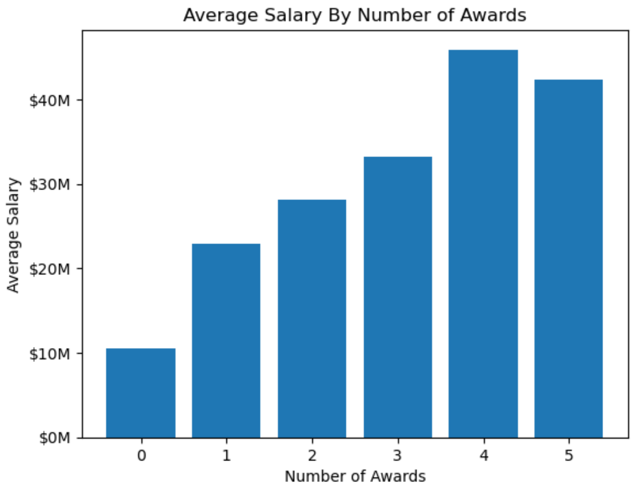
1 rows × 62 columns

```
subset = df[df['Age'] < 39]
age_salary = subset.groupby('Age')['Next_Year_Salary'].mean()

ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)

plt.bar(age_salary.index, age_salary.values)
plt.xlabel("Age")
plt.ylabel("Avg Salary")
plt.title("Average Salary by Age")

plt.savefig("outputs/avg_salary_by_age.png")
```



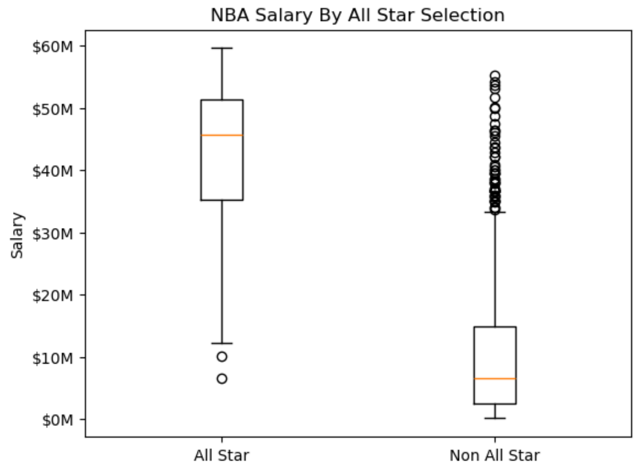
The 40 years old column stands out, but after inspecting the dataframe, it makes sense, as there is only one 40 year old player, which is Lebron James. Arguably the greatest player of all time, he has a salary of about $48 million. Other than that though, the graph looks to be bimodal, with peaks around 29 years and 34 years. This makes sense, as players in their late 20s are typically around their peaks, which would coincide with higher salaries. The second peak is around older players, but the only players to remain in the leage in that age range are typically older superstars whose careers are defined by longevity. It then falls off, as very old players in their late 30s aren't as efficient as they used to be, which coincides with smaller salaries.

```
awards_salary = df.groupby('NumOfAwards')['Next_Year_Salary'].mean()

plt.bar(awards_salary.index, awards_salary.values)

ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)

plt.xlabel('Number of Awards')
plt.ylabel("Average Salary")

plt.title("Average Salary By Number of Awards")

plt.savefig("outputs/avg_salary_by_awards.png")
```




```
all_star = df[df['All -Star'] == 1]
non_all_star = df[df['All -Star'] == 0]

plt.boxplot([all_star['Next_Year_Salary'], non_all_star['Next_Year_Salary']], labels = ["All
ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)

plt.ylabel("Salary")
plt.title("NBA Salary By All Star Selection")

plt.savefig("outputs/all_star_salary_comp.png")
```

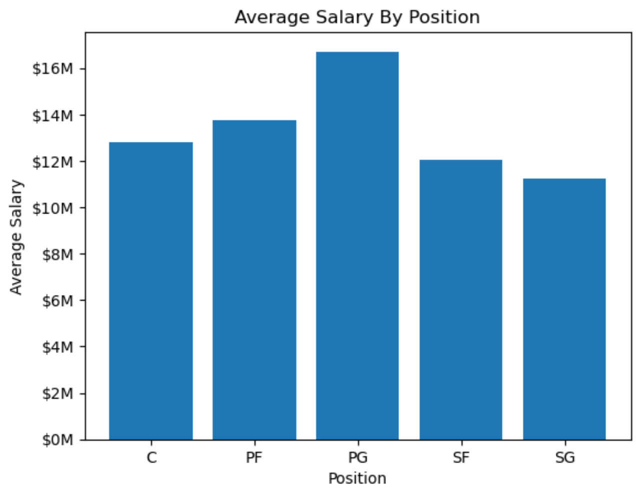
```
/tmp/ipykernel_1497/133333447.py:4: MatplotlibDeprecationWarning: The 'labels' parameter of
  plt.boxplot([all_star['Next_Year_Salary'], non_all_star['Next_Year_Salary']], labels = ["A
```



There are many awards in the NBA most notably: Most Valuable Player (MVP), Defensive Player of the Year (DPOY), and All NBA team selections. As expected, players with more awards tend to have higher salaries, as those with more awards are naturally better players. Another important accolade is selection for the All Star team. The side by side boxplot shows that All Star Players do earn much more than those who aren't. However, there are a lot of non All Star players as outliers who make about the same as All Star players. This is because All Star nominations are extremely selective, as there is only a certain amount of players each year who can be selected. As a result, there are many extremely talented players who are left off of the All Star roster.

1.5 Salary By Position

```
position_salary = df.groupby('Pos')['Next_Year_Salary'].mean()

plt.bar(position_salary.index, position_salary.values)

ax = plt.gca()

ax.yaxis.set_major_formatter(
    plt.FuncFormatter(lambda x, _: f"${x/1e6:.0f}M")
)

plt.xlabel("Position")
plt.ylabel("Average Salary")

plt.title("Average Salary By Position")

plt.savefig("outputs/salary_by_pos.png")
```



It seems like point guards on average are paid more than their peers. This makes sense, because usually the point guards are the players managing a team's offense. Oftentimes, team offenses are largely structured around the play of their pointguards. Due to their importance to teams, it does make sense that they are paid more on average than other positions. All other positions are on average paid around the same at around $12 million.

2 Multicollinearity and VIF Analysis

NBA Stats data is highly correlated, as most advanced stats are functions of other stats. Any offensive percentage based statistic is simply the quotient of successful attempts and total attempts. For instance, FG% = FG/FGA (Field Goals Made / Field Goals Attempted). It's important to conduct a VIF analysis and remove highly correlated variables, as they can cause instability in our later regression models.

2.1 Current VIF Scores

```
vif_columns = ['Age', 'G', 'GS', 'MP_x', 'FG', 'FGA',
        'FG%', '3P', '3PA', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA',
         'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS', 'PER', 'TS%', '3PAr',
        'STL%', 'BLK%', 'TOV%', 'USG%', 'OWS', 'DWS', 'WS', 'WS/48', 'OBPM',
        'DBPM', 'BPM', 'VORP', 'Experience', 'NumOfAwards',
        'All -Star', 'AwardWinner', 'FirstTeam', 'SecondTeam', 'ThirdTeam',
        'DefTeam1', 'DefTeam2']

non_numeric = ['Player', 'Team', 'Pos']

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
```

```
df_vif = df[vif_columns].copy()
df_vif = df_vif.fillna(df_vif.median())

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_vif)

vif_df = pd.DataFrame()
vif_df['feature'] = df_vif.columns
vif_df['VIF'] = [variance_inflation_factor(X_scaled, i)
                 for i in range(X_scaled.shape[1])]

vif_df.sort_values('VIF', ascending = False)
```

	feature	VIF
5	FGA	11314.221656
23	PTS	9776.988257
4	FG	6132.743387
10	2PA	5944.950433
42	BPM	5438.085530
40	OBPM	3779.923247
38	WS	3567.776088
8	3PA	2562.969471
17	TRB	2562.891993
9	2P	2021.851050
36	OWS	1787.044350
16	DRB	1476.600673
30	TRB%	1404.880118
41	DBPM	895.620263
7	3P	738.359541
29	DRB%	627.270609
13	FT	535.581498
37	DWS	500.505318
6	FG%	415.821142
12	eFG%	303.521027
28	ORB%	278.749742
15	ORB	270.179573
24	PER	225.317081
14	FTA	141.136785
26	3PAr	101.853601
39	WS/48	101.298727
25	TS%	79.766231
3	MP_x	75.411293
35	USG%	54.571440
43	VORP	46.427325
18	AST	44.591487
21	TOV	30.921930
31	AST%	25.900951
33	BLK%	14.185062
20	BLK	14.082939
19	STL	11.970258
32	STL%	11.292303
45	NumOfAwards	9.918648
44	Experience	9.062526
0	Age	8.625368
34	TOV%	7.526962
11	2P%	7.413079
27	FTr	6.024722
1	G	5.453902
2	GS	4.688047
22	PF	4.434805
48	FirstTeam	3.847879
46	All-Star	3.490438
49	SecondTeam	2.462567
51	DefTeam1	1.943521
52	DefTeam2	1.653303
50	ThirdTeam	1.638871
47	AwardWinner	1.386805

There are a lot of highly collinear variables, as shown by the high VIF scores. Let's drop all of the counting stats and keep percentage based stats. We'll also drop most of the advanced stats but keep BPM, as most advanced stats are factored into BPM.

2.2 Dropping Collinear Columns

```
cols2drop = ['G', 'GS', 'FG', 'FGA', '3P', '3PA', '2P', '2PA', 'FT', 'FTA', 'ORB', 'DRB', 'T

df_vif.drop(cols2drop, axis=1, inplace=True)

df_vif.columns

Index(['Age', 'MP_x', 'PF', 'TS%', 'TRB%', 'AST%', 'STL%', 'BLK%', 'TOV%',
       'USG%', 'BPM', 'NumOfAwards', 'All -Star', 'AwardWinner', 'FirstTeam',
       'SecondTeam', 'ThirdTeam', 'DefTeam1', 'DefTeam2'],
      dtype='object')

df_vif = df_vif.fillna(df_vif.median())

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_vif)

vif_df = pd.DataFrame()
vif_df['feature'] = df_vif.columns
vif_df['VIF'] = [variance_inflation_factor(X_scaled, i)
                 for i in range(X_scaled.shape[1])]

vif_df.sort_values('VIF', ascending = False)
```

	feature	VIF
11	NumOfAwards	9.163318
10	BPM	9.101377
5	AST%	4.024821
1	MP_x	3.598090
3	TS%	3.450686
14	FirstTeam	3.128119
12	All-Star	3.072324
8	TOV%	2.705655
2	PF	2.578249
15	SecondTeam	2.320510
4	TRB%	2.306702
7	BLK%	2.070009
9	USG%	2.003561
17	DefTeam1	1.746462
6	STL%	1.717763
18	DefTeam2	1.557843
16	ThirdTeam	1.513444
13	AwardWinner	1.327046
0	Age	1.118065

Much better, we can keep all of these columns, as the VIF values are all less than 10.

```
cols2keep = list(vif_df.feature)
```

```
cols2keep
```

```
['Age',
 'MP_x',
 'PF',
 'TS%',
 'TRB%',
 'AST%',
 'STL%',
 'BLK%',
 'TOV%',
 'USG%',
 'BPM',
 'NumOfAwards',
 'All -Star',
 'AwardWinner',
 'FirstTeam',
 'SecondTeam',
 'ThirdTeam',
 'DefTeam1',
```

```
 'DefTeam2']

final_df = df[cols2keep+['Pos', 'Next_Year_Salary']]

final_df.columns

Index(['Age', 'MP_x', 'PF', 'TS%', 'TRB%', 'AST%', 'STL%', 'BLK%', 'TOV%',
       'USG%', 'BPM', 'NumOfAwards', 'All -Star', 'AwardWinner', 'FirstTeam',
       'SecondTeam', 'ThirdTeam', 'DefTeam1', 'DefTeam2', 'Pos',
       'Next_Year_Salary'],
      dtype='object')

display(final_df)
```

	Age	MP_x	PF	TS%	TRB%	AST%	STL%	BLK%	TOV%	USG%	...	NumOfAwards	All-Star	AwardWinner	FirstTeam	SecondTeam	ThirdTeam	DefTeam1	DefTeam2	Pos	Next_Year_Salary
0	24.0	11.0	0.9	0.615	7.7	6.4	0.7	0.6	5.5	15.2	...	0	0	0	0	0	0	0	0	SG	2120693
1	20.0	8.6	0.3	0.385	2.7	3.7	0.3	1.1	11.3	17.3	...	0	0	0	0	0	0	0	0	SF	250000
2	28.0	31.5	1.9	0.601	11.4	15.2	1.2	1.7	11.2	17.8	...	0	0	0	0	0	0	0	0	PF	22841455
3	27.0	16.3	1.6	0.578	3.2	15.4	1.6	0.4	10.6	16.7	...	0	0	0	0	0	0	0	0	PG	4668000
4	24.0	27.7	3.3	0.637	1.8	6.6	1.5	2.0	8.4	16.0	...	0	0	0	0	0	0	0	0	SF	11000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
307	27.0	15.3	2.1	0.605	15.6	15.6	1.4	2.6	15.1	17.1	...	0	0	0	0	0	0	0	0	PF	18080496
308	22.0	21.5	2.8	0.624	20.5	6.1	1.2	5.5	15.3	16.4	...	1	0	0	0	0	0	0	0	C	6045000
309	29.0	35.2	1.6	0.639	9.5	18.3	1.1	0.4	13.4	25.3	...	0	0	0	0	0	0	0	0	SG	47499660
310	24.0	10.7	1.1	0.578	4.1	4.6	2.0	5.6	8.0	12.2	...	0	0	0	0	0	0	0	0	PF	8177778
311	24.0	28.6	2.7	0.601	13.6	36.6	2.1	3.0	12.8	34.7	...	0	0	0	0	0	0	0	0	PF	39446090

751 rows × 21 columns

```
final_df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 751 entries, 0 to 311
Data columns (total 21 columns):
 #   Column            Non -Null Count  Dtype
 - - -  - - - - - - -          - - - - - - - - - - - - - -  - - - - -
 0   Age               751 non -null    float64
 1   MP_x              751 non -null    float64
 2   PF                751 non -null    float64
 3   TS%               751 non -null    float64
 4   TRB%              751 non -null    float64
 5   AST%              751 non -null    float64
 6   STL%              751 non -null    float64
 7   BLK%              751 non -null    float64
```

```
 8   TOV%              751 non -null    float64
 9   USG%              751 non -null    float64
 10  BPM               751 non -null    float64
 11  NumOfAwards       751 non -null    int64
 12  All -Star          751 non -null    int64
 13  AwardWinner       751 non -null    int64
 14  FirstTeam         751 non -null    int64
 15  SecondTeam        751 non -null    int64
 16  ThirdTeam         751 non -null    int64
 17  DefTeam1          751 non -null    int64
 18  DefTeam2          751 non -null    int64
 19  Pos               751 non -null    object
 20  Next_Year_Salary  751 non -null    int64
dtypes: float64(11), int64(9), object(1)
memory usage: 129.1+ KB
```

Above, we printed out the type of each column/feature. We see that all columns are of type int or float except the ‘Pos’ column which is of type of object currently. We want to transform this column from type object to bool using one-hot encoding and then cast the bool values into integer values.

```
oh_df = pd.get_dummies(final_df, columns=['Pos'])
oh_df['Pos_C'] = oh_df['Pos_C'].astype(int)
oh_df['Pos_PF'] = oh_df['Pos_PF'].astype(int)
oh_df['Pos_PG'] = oh_df['Pos_PG'].astype(int)
oh_df['Pos_SF'] = oh_df['Pos_SF'].astype(int)
oh_df['Pos_SG'] = oh_df['Pos_SG'].astype(int)

display(oh_df)
```

	Age	MP	PF_x	TS%	TRB%	AST%	STL%	BLK%	TOV%	USG%	...	SecondTeam	ThirdTeam	DefTeam1	DefTeam2	Next_Year_Salary	Pos_C	Pos_PF	Pos_PG	Pos_SF	Pos_SG
0	24.0	11.0	0.9	0.615	7.7	6.4	0.7	0.6	5.5	15.2	...	0	0	0	0	2120693	0	0	0	0	1
1	20.0	8.6	0.3	0.382	2.7	3.7	0.3	1.1	11.3	17.3	...	0	0	0	0	2500000	0	0	0	1	0
2	28.0	31.5	1.9	0.607	11.4	15.2	1.2	1.7	11.2	17.8	...	0	0	0	0	22841455	[illegible]	[illegible]	0	0	0
3	27.0	16.3	1.6	0.578	8.2	15.4	1.6	0.4	10.6	16.7	...	0	0	0	0	4668000	0	0	1	0	0
4	24.0	27.7	3.3	0.637	1.8	6.6	1.5	2.0	8.4	16.0	...	0	0	0	0	11000000	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
307	27.0	15.3	2.1	0.605	5.6	15.6	1.4	2.6	15.1	17.1	...	0	0	0	0	18080496	[illegible]	[illegible]	0	0	0
308	22.0	21.5	2.8	0.624	20.5	6.1	1.2	5.5	15.3	16.4	...	0	0	0	0	6045000	[illegible]	[illegible]	0	0	0
309	29.0	35.2	1.6	0.639	9.5	18.3	1.1	0.4	13.4	25.3	...	0	0	0	0	47499660	0	0	0	0	1
310	24.0	10.7	1.1	0.578	4.1	4.6	2.0	5.6	8.0	12.2	...	0	0	0	0	8177778	0	1	0	0	0
311	24.0	28.6	2.7	0.600	13.6	36.6	2.1	3.0	12.8	34.7	...	0	0	0	0	39446090	[illegible]	[illegible]	0	0	0

751 rows × 25 columns

```
oh_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 751 entries, 0 to 311
Data columns (total 25 columns):
 #   Column            Non -Null Count  Dtype
 - - -  - - - - - - -          - - - - - - - - - - - - - -  - - - - -
 0   Age               751 non -null    float64
 1   MP_x              751 non -null    float64
 2   PF                751 non -null    float64
 3   TS%               751 non -null    float64
 4   TRB%              751 non -null    float64
 5   AST%              751 non -null    float64
 6   STL%              751 non -null    float64
 7   BLK%              751 non -null    float64
 8   TOV%              751 non -null    float64
 9   USG%              751 non -null    float64
 10  BPM               751 non -null    float64
 11  NumOfAwards       751 non -null    int64
 12  All -Star          751 non -null    int64
 13  AwardWinner       751 non -null    int64
 14  FirstTeam         751 non -null    int64
 15  SecondTeam        751 non -null    int64
 16  ThirdTeam         751 non -null    int64
 17  DefTeam1          751 non -null    int64
 18  DefTeam2          751 non -null    int64
 19  Next_Year_Salary  751 non -null    int64
 20  Pos_C             751 non -null    int64
 21  Pos_PF            751 non -null    int64
 22  Pos_PG            751 non -null    int64
 23  Pos_SF            751 non -null    int64
 24  Pos_SG            751 non -null    int64
dtypes: float64(11), int64(14)
memory usage: 152.5 KB
```

After one-hot encoding, we see that the original ‘Pos’ column was dropped and replaced by 5 new columns, each corresponding to a ‘Pos’ value and we are now ready to fit models on this processed dataset.

```
oh_df.to_csv("data/oh_df.csv", index=False)
```