Predicting an NBA Player's Position

Project Goal

> *The NBA is considered the most "positionless" major sport. The game has evolved to the point where 6'8" players could play any position, from guard to center. Can we use machine learning to predict an NBA player's position based on their in-game stats?*

Data

> NBA Player's Stats since 1950

Libraries Used

> - Pandas
> - Numpy
> - Sklearn
> - Seaborn
> - Geopandas

Machine Learning Algorithms Used

> - Decision Trees
> - Random Forest
> - K-Nearest Neighbors
> - Support Vector Machine

1. Reading/Cleaning Data

In [120…

```
import pandas as pd
pd.set_option('display.max_columns', 500)

df = pd.read_csv("Seasons_Stats.csv")

df.head()
```

Out[120…

	Unnamed: 0	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%
0	0	1950.0	Curly Armstrong	G-F	31.0	FTW	63.0	NaN	NaN	NaN	0.368	NaN	0.467	NaN
1	1	1950.0	Cliff Barker	SG	29.0	INO	49.0	NaN	NaN	NaN	0.435	NaN	0.387	NaN
2	2	1950.0	Leo Barnhorst	SF	25.0	CHS	67.0	NaN	NaN	NaN	0.394	NaN	0.259	NaN
3	3	1950.0	Ed Bartels	F	24.0	TOT	15.0	NaN	NaN	NaN	0.312	NaN	0.395	NaN

	Unnamed: 0	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	I
4	4	1950.0	Ed Bartels	F	24.0	DNN	13.0	NaN	NaN	NaN	0.308	NaN	0.378	NaN	

In [121…

```
df.columns
```

Out[121…

```
Index(['Unnamed: 0', 'Year', 'Player', 'Pos', 'Age', 'Tm', 'G', 'GS', 'MP',
       'PER', 'TS%', '3PAr', 'FTr', 'ORB%', 'DRB%', 'TRB%', 'AST%', 'STL%',
       'BLK%', 'TOV%', 'USG%', 'blanl', 'OWS', 'DWS', 'WS', 'WS/48', 'blank2',
       'OBPM', 'DBPM', 'BPM', 'VORP', 'FG', 'FGA', 'FG%', '3P', '3PA', '3P%',
       '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'FT%', 'ORB', 'DRB', 'TRB',
       'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS'],
      dtype='object')
```

Finding columns with null values

In [122…

```
df.drop(columns = ['Unnamed: 0'], inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24691 entries, 0 to 24690
Data columns (total 52 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Year    24624 non-null  float64
 1   Player  24624 non-null  object 
 2   Pos     24624 non-null  object 
 3   Age     24616 non-null  float64
 4   Tm      24624 non-null  object 
 5   G       24624 non-null  float64
 6   GS      18233 non-null  float64
 7   MP      24138 non-null  float64
 8   PER     24101 non-null  float64
 9   TS%     24538 non-null  float64
 10  3PAr    18839 non-null  float64
 11  FTr     24525 non-null  float64
 12  ORB%    20792 non-null  float64
 13  DRB%    20792 non-null  float64
 14  TRB%    21571 non-null  float64
 15  AST%    22555 non-null  float64
 16  STL%    20792 non-null  float64
 17  BLK%    20792 non-null  float64
 18  TOV%    19582 non-null  float64
 19  USG%    19640 non-null  float64
 20  blanl   0 non-null      float64
 21  OWS     24585 non-null  float64
 22  DWS     24585 non-null  float64
 23  WS      24585 non-null  float64
 24  WS/48   24101 non-null  float64
 25  blank2  0 non-null      float64
 26  OBPM    20797 non-null  float64
 27  DBPM    20797 non-null  float64
 28  BPM     20797 non-null  float64
 29  VORP    20797 non-null  float64
 30  FG      24624 non-null  float64
 31  FGA     24624 non-null  float64
 32  FG%     24525 non-null  float64
 33  3P      18927 non-null  float64
 34  3PA     18927 non-null  float64
 35  3P%     15416 non-null  float64
 36  2P      24624 non-null  float64
```

```
 37  2PA     24624 non-null  float64
 38  2P%     24496 non-null  float64
 39  eFG%    24525 non-null  float64
 40  FT      24624 non-null  float64
 41  FTA     24624 non-null  float64
 42  FT%     23766 non-null  float64
 43  ORB     20797 non-null  float64
 44  DRB     20797 non-null  float64
 45  TRB     24312 non-null  float64
 46  AST     24624 non-null  float64
 47  STL     20797 non-null  float64
 48  BLK     20797 non-null  float64
 49  TOV     19645 non-null  float64
 50  PF      24624 non-null  float64
 51  PTS     24624 non-null  float64
dtypes: float64(49), object(3)
memory usage: 9.5+ MB
```

Finding duplicated rows

In [123…

```
df.duplicated().sum()
```

Out[123…

```
66
```

In [124…

```
df['MPG'] = df['MP'] / df['G']
```

In [125…

```
df.isna().sum()
```

Out[125…

```
Year          67
Player        67
Pos           67
Age           75
Tm            67
G             67
GS          6458
MP           553
PER          590
TS%          153
3PAr        5852
FTr          166
ORB%        3899
DRB%        3899
TRB%        3120
AST%        2136
STL%        3899
BLK%        3899
TOV%        5109
USG%        5051
blanl      24691
OWS          106
DWS          106
WS           106
WS/48        590
blank2     24691
OBPM        3894
DBPM        3894
BPM         3894
VORP        3894
FG            67
FGA           67
FG%          166
3P          5764
3PA         5764
```

```
3P%       9275
2P          67
2PA         67
2P%        195
eFG%       166
FT          67
FTA         67
FT%        925
ORB       3894
DRB       3894
TRB        379
AST         67
STL       3894
BLK       3894
TOV       5046
PF          67
PTS         67
MPG        553
dtype: int64
```

In [126…

```
df[df['FT%'].isnull()]
```

Out[126…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%
79	1950.0	Normie Glick	F	22.0	MNL	1.0	NaN	NaN	NaN	1.000	NaN	0.0	NaN	NaN
132	1950.0	Lee Knorek	C	28.0	BLB	1.0	NaN	NaN	NaN	0.000	NaN	0.0	NaN	NaN
175	1950.0	Murray Mitchell	C	26.0	AND	2.0	NaN	NaN	NaN	0.333	NaN	0.0	NaN	NaN
187	1950.0	Jim Nolan	C	22.0	PHW	5.0	NaN	NaN	NaN	0.190	NaN	0.0	NaN	NaN
312	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
24568	2017.0	Damjan Rudez	SF	30.0	ORL	45.0	0.0	314.0	6.3	0.466	0.727	0.0	1.7	7.1
24620	2017.0	Mike Tobey	C	22.0	CHO	2.0	0.0	25.0	-0.1	0.250	0.000	0.0	8.7	4.4
24622	2017.0	Axel Toupane	SF	24.0	TOT	4.0	0.0	47.0	6.2	0.611	0.444	0.0	0.0	2.3
24623	2017.0	Axel Toupane	SF	24.0	MIL	2.0	0.0	6.0	-9.9	0.000	1.000	0.0	0.0	0.0
24624	2017.0	Axel Toupane	SF	24.0	NOP	2.0	0.0	41.0	8.6	0.688	0.375	0.0	0.0	2.6

925 rows × 53 columns

In [127…

```
df[df['FT%'].isnull()].mean()['G']
```

Out[127…

```
4.301864801864802
```

In [128…
```
df[df['FT%'].isnull()].mean()['MP']
```
Out[128… 26.24970691676436

In [129…
```
df[df['FT%'].isnull()].mean()['MPG']
```
Out[129… 5.798066690777163

In [130…
```
all_years = df.copy()
df = df[df['Year'] >= 2010]
df = df[(df['G'] >= 20) & (df['MPG'] >= 15)]
df.shape
```
Out[130… (2845, 53)

In [131…
```
df.fillna(0, inplace=True)
```

Finding the different positions in the dataset

> Some positions have multiple values, such as "PF-SF". To fix this, I filtered the dataset by the specific values, and changed the positions to make it uniform.

In [132…
```
df['Pos'].value_counts()
```
Out[132…
```
SG       601
PG       590
PF       568
SF       566
C        481
PF-SF      7
SG-PG      7
SF-PF      6
PG-SG      6
SG-SF      4
SF-SG      3
PF-C       3
C-PF       2
SG-PF      1
Name: Pos, dtype: int64
```

In [133…
```
df[df['Pos'] == 'PF-SF']
```
Out[133…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%
20216	2010.0	Jared Jeffries	PF-SF	28.0	TOT	70.0	37.0	1794.0	9.8	0.507	0.226	0.382	8.1	10.3
20401	2010.0	James Singleton	PF-SF	28.0	TOT	57.0	3.0	977.0	11.4	0.452	0.146	0.244	12.6	20.0
20725	2011.0	Danilo Gallinari	PF-SF	22.0	TOT	62.0	60.0	2104.0	15.7	0.597	0.458	0.611	3.2	13.4
20753	2011.0	Jeff Green	PF-SF	24.0	TOT	75.0	51.0	2427.0	12.9	0.538	0.257	0.298	3.7	13.5
22048	2013.0	Marcus Morris	PF-SF	23.0	TOT	77.0	23.0	1524.0	11.3	0.516	0.443	0.222	6.0	14.6

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%
22625	2014.0	Luc Mbah	PF-SF	27.0	TOT	64.0	7.0	1003.0	8.3	0.503	0.088	0.347	6.0	10.1
23439	2015.0	Lance Thomas	PF-SF	26.0	TOT	62.0	37.0	1490.0	8.0	0.456	0.050	0.224	4.8	9.9

In [134…

```
df[df['Pos'] == 'PG-SG']
```

Out[134…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%
20164	2010.0	Eddie House	PG-SG	31.0	TOT	68.0	0.0	1217.0	10.5	0.495	0.499	0.114	1.3	9.3
20705	2011.0	Raymond Felton	PG-SG	26.0	TOT	75.0	54.0	2737.0	16.6	0.524	0.327	0.240	2.0	9.2
22020	2013.0	Eric Maynor	PG-SG	25.0	TOT	64.0	0.0	963.0	9.3	0.472	0.342	0.221	1.1	4.4
22080	2013.0	Jeremy Pargo	PG-SG	26.0	TOT	39.0	11.0	655.0	10.4	0.478	0.296	0.248	1.3	7.5
23193	2015.0	Brandon Knight	PG-SG	23.0	TOT	63.0	61.0	2035.0	17.1	0.543	0.361	0.251	1.6	12.0
23356	2015.0	Austin Rivers	PG-SG	22.0	TOT	76.0	5.0	1563.0	10.3	0.481	0.264	0.254	2.0	8.9

In [135…

```
df[df['Pos'] == 'C-PF']
```

Out[135…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%
20111	2010.0	Drew Gooden	C-PF	28.0	TOT	70.0	33.0	1755.0	16.9	0.547	0.013	0.359	13.7	21.9
21802	2013.0	Ed Davis	C-PF	23.0	TOT	81.0	28.0	1631.0	17.8	0.561	0.000	0.346	10.9	22.8

In [136…

```
df[df['Pos'] == 'SG-SF']
```

Out[136…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%
20459	2010.0	Henry Walker	SG-SF	22.0	TOT	35.0	13.0	768.0	14.6	0.649	0.500	0.211	2.2	11.0
20956	2011.0	Mickael Pietrus	SG-SF	28.0	TOT	57.0	4.0	1107.0	9.8	0.526	0.647	0.156	1.6	11.5
21850	2013.0	Francisco Garcia	SG-SF	31.0	TOT	58.0	20.0	1029.0	11.0	0.519	0.597	0.070	0.9	9.4
22733	2014.0	John Salmons	SG-SF	34.0	TOT	78.0	8.0	1726.0	7.8	0.462	0.395	0.126	1.3	9.6

In [137…

```
df[df['Pos'] == 'SG-PF']
```

Out[137…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%	T
20204	2010.0	Stephen Jackson	SG-PF	31.0	TOT	81.0	81.0	3129.0	15.6	0.518	0.277	0.306	3.0	12.2	

In [138…

```
df[df['Pos'] == 'PF-C']
```

Out[138…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%
20015	2010.0	Marcus Camby	PF-C	35.0	TOT	74.0	74.0	2314.0	17.9	0.501	0.014	0.244	12.7	31.9
20435	2010.0	Tyrus Thomas	PF-C	23.0	TOT	54.0	3.0	1220.0	16.8	0.511	0.007	0.417	7.8	23.6
23682	2016.0	Channing Frye	PF-C	32.0	TOT	70.0	32.0	1200.0	12.9	0.586	0.677	0.101	3.2	18.6

In [139…

```
def change_pos(pos):
    if pos == 'PF-SF':
        return 'SF'
    if pos == 'SG-PG':
        return 'PG'
    if pos == 'PG-SG':
        return 'PG'
    if pos == 'SF-PF':
        return 'SF'
    if pos == 'C-PF':
        return 'PF'
    if pos == 'SG-SF':
        return 'SG'
    if pos == 'SF-SG':
        return 'SF'
    if pos == 'PF-C':
        return 'PF'
    if pos == 'SG-PF':
        return 'SF'
    if pos == 'C-SF':
        return 'C'
    else:
        return pos

df['Pos'] = df['Pos'].apply(lambda x : change_pos(x))
df['Pos'].value_counts()
```

Out[139…

```
SG    605
PG    603
SF    583
PF    573
C     481
Name: Pos, dtype: int64
```

2. Merging dataset of players heights/weights

In [140…
```
info = pd.read_csv("Players.csv")
info = info[['Player', 'height', 'weight']]
info.head()
```

Out[140…

	Player	height	weight
0	Curly Armstrong	180.0	77.0
1	Cliff Barker	188.0	83.0
2	Leo Barnhorst	193.0	86.0
3	Ed Bartels	196.0	88.0
4	Ralph Beard	178.0	79.0

In [141…
```
df = df.merge(info, on = 'Player', how = 'left')
df.head()
```

Out[141…

	Year	Player	Pos	Age	Tm	G	GS	MP	PER	TS%	3PAr	FTr	ORB%	DRB%	TRB%
0	2010.0	Arron Afflalo	SG	24.0	DEN	82.0	75.0	2221.0	10.9	0.576	0.426	0.168	3.1	9.9	6.
1	2010.0	LaMarcus Aldridge	PF	24.0	POR	78.0	78.0	2922.0	18.2	0.535	0.014	0.260	8.1	18.6	13.
2	2010.0	Ray Allen	SG	34.0	BOS	80.0	80.0	2819.0	15.2	0.601	0.410	0.260	2.0	8.8	5.
3	2010.0	Tony Allen	SG	28.0	BOS	54.0	8.0	889.0	14.2	0.540	0.020	0.470	7.4	12.5	10.
4	2010.0	Rafer Alston	PG	33.0	TOT	52.0	38.0	1421.0	8.2	0.443	0.377	0.182	1.0	9.7	5.

In [142…
```
df.columns
```

Out[142…
```
Index(['Year', 'Player', 'Pos', 'Age', 'Tm', 'G', 'GS', 'MP', 'PER', 'TS%',
       '3PAr', 'FTr', 'ORB%', 'DRB%', 'TRB%', 'AST%', 'STL%', 'BLK%', 'TOV%',
       'USG%', 'blanl', 'OWS', 'DWS', 'WS', 'WS/48', 'blank2', 'OBPM', 'DBPM',
       'BPM', 'VORP', 'FG', 'FGA', 'FG%', '3P', '3PA', '3P%', '2P', '2PA',
       '2P%', 'eFG%', 'FT', 'FTA', 'FT%', 'ORB', 'DRB', 'TRB', 'AST', 'STL',
       'BLK', 'TOV', 'PF', 'PTS', 'MPG', 'height', 'weight'],
      dtype='object')
```

In [143…
```
df.fillna(0, inplace=True)
df.isna().sum()
```

Out[143…
```
Year      0
Player    0
Pos       0
Age       0
Tm        0
G         0
GS        0
MP        0
PER       0
TS%       0
3PAr      0
FTr       0
```

```
ORB%      0
DRB%      0
TRB%      0
AST%      0
STL%      0
BLK%      0
TOV%      0
USG%      0
blanl     0
OWS       0
DWS       0
WS        0
WS/48     0
blank2    0
OBPM      0
DBPM      0
BPM       0
VORP      0
FG        0
FGA       0
FG%       0
3P        0
3PA       0
3P%       0
2P        0
2PA       0
2P%       0
eFG%      0
FT        0
FTA       0
FT%       0
ORB       0
DRB       0
TRB       0
AST       0
STL       0
BLK       0
TOV       0
PF        0
PTS       0
MPG       0
height    0
weight    0
dtype: int64
```

3. Grouping by Position and finding averages by position

In [144…

```
by_pos = df.groupby('Pos').mean()
by_pos.head()
```

Out[144…

	Year	Age	G	GS	MP	PER	TS%	3PAr	FTr
Pos									
C	2013.569647	26.802495	61.904366	40.632017	1541.937630	16.598337	0.555112	0.037717	0.367597
PF	2013.568935	26.745201	61.783595	33.980803	1567.617801	15.379930	0.537162	0.167290	0.286969
PG	2013.570481	26.817579	60.407960	34.220564	1616.782753	14.680929	0.524381	0.330415	0.251323
SF	2013.578045	27.090909	62.049743	35.713551	1639.550600	13.015609	0.532441	0.360063	0.246926

	Year	Age	G	GS	MP	PER	TS%	3PAr	FTr
Pos									
SG	2013.528926	27.031405	61.480992	32.114050	1638.849587	13.405289	0.534098	0.377689	0.235301

In [145...

```
year_pos = all_years.groupby(['Pos', 'Year'], as_index=False).mean()
year_pos = year_pos[year_pos['Year'] >= 1980]
year_pos.head()
```

Out[145...

	Pos	Year	Age	G	GS	MP	PER	TS%	3PAr	FTr
30	C	1980.0	26.830769	56.969231	41.000000	1460.076923	14.024615	0.520292	0.003277	0.360692
31	C	1981.0	26.875000	60.015625	27.333333	1458.921875	12.854687	0.511016	0.003156	0.369203
32	C	1982.0	27.106061	58.469697	29.484848	1372.075758	12.683333	0.532909	0.004212	0.409970
33	C	1983.0	27.045455	52.511364	23.795455	1170.795455	12.153409	0.502420	0.004489	0.355636
34	C	1984.0	27.478261	63.014493	30.985507	1452.072464	12.649275	0.537420	0.003232	0.376986

In [146...

```
#only using percentages
df.columns
per = df[['Year', 'Player', 'Pos', 'Age', 'TS%', 'ORB%', 'DRB%', 'TRB%', 'AST%', 'STL%'
```

In [147...

```
non_per = df[['Year', 'Player', 'Pos', 'Age', 'PER', '3PAr', 'FTr', 'OWS', 'DWS', 'WS',
```

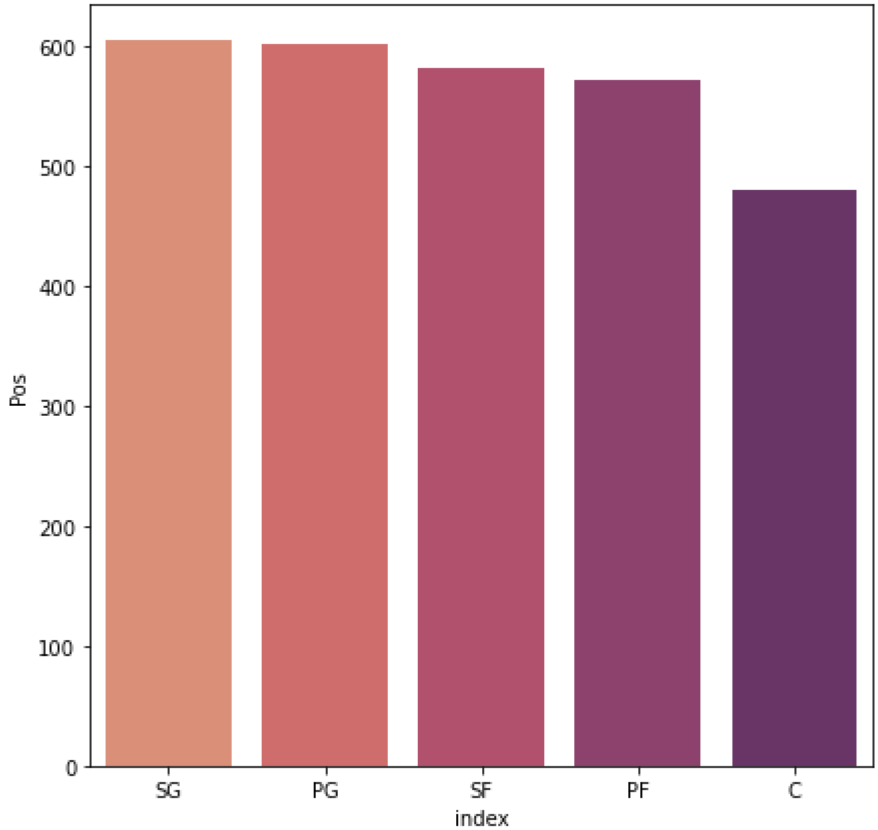
In [148...

```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.gridspec as gridspec
import numpy as np
```

4. Exploratory Data Analysis

In [149...

```
counts = df['Pos'].value_counts().reset_index()
plt.figure(figsize = (7,7))
sns.barplot(x = 'index', y = 'Pos', data = counts, palette = "flare")
plt.show()
```



> There are significantly less centers in the dataset. This is because shooting guards and point guards make up "Guards", small forwards and power forwards make up "Forwards", but only centers are included in "Centers"

In [150…

```
c = year_pos[year_pos['Pos'] == 'C']
pf = year_pos[year_pos['Pos'] == 'PF']
sf = year_pos[year_pos['Pos'] == 'SF']
sg = year_pos[year_pos['Pos'] == 'SG']
pg = year_pos[year_pos['Pos'] == 'PG']
```

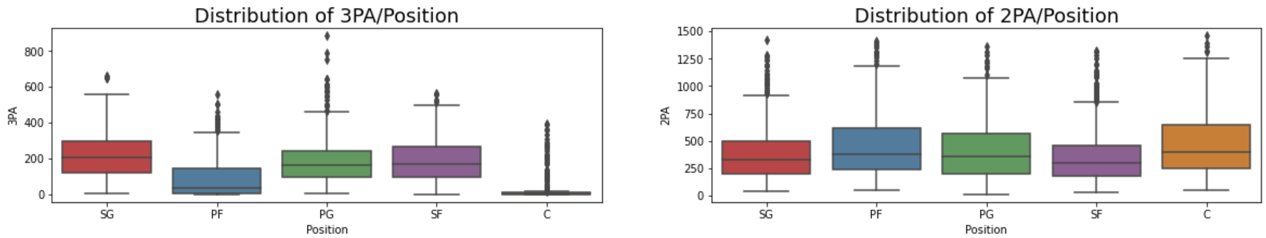
In [151…

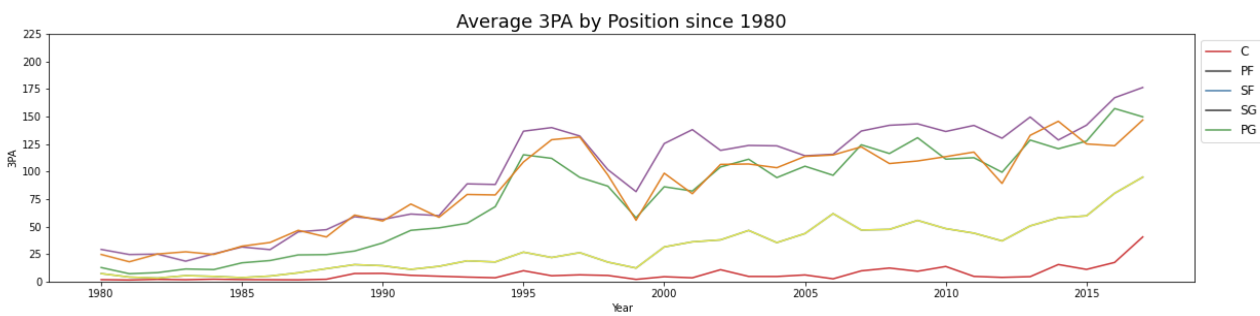
```
sns.set_palette("Set1", 8, .75)

gs = gridspec.GridSpec(3,2)
plt.figure(figsize = (20,10))

ax1 = plt.subplot(gs[0, 0]) # row 0, col 0
sns.boxplot(x = 'Pos', y = '3PA', data = df)
ax1.set_title("Distribution of 3PA/Position", {'fontsize' : 18.0})
ax1.set_xlabel("Position")
ax1.set_ylabel("3PA")

ax2 = plt.subplot(gs[0, 1]) # row 0, col 0
sns.boxplot(x = 'Pos', y = '2PA', data = df)
ax2.set_title("Distribution of 2PA/Position",{'fontsize' : 18.0} )
ax2.set_xlabel("Position")
ax2.set_ylabel("2PA")

plt.figure(figsize = (20,15))
ax3 = plt.subplot(gs[1,:]) # row 0, col 0
ax3.set_title("Average 3PA by Position since 1980", {'fontsize' : 18.0})
ax3.set_ylim(0,225)
```

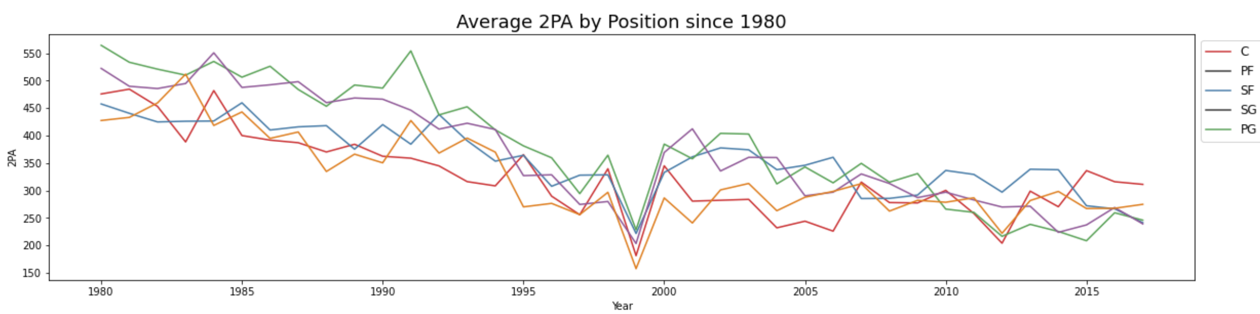
```
sns.lineplot(x = 'Year', y = '3PA', data = c, style = 'Pos')
sns.lineplot(x = 'Year', y = '3PA', data = pf, style = 'Pos')
sns.lineplot(x = 'Year', y = '3PA', data = sf, style = 'Pos')
sns.lineplot(x = 'Year', y = '3PA', data = sg, style = 'Pos')
sns.lineplot(x = 'Year', y = '3PA', data = pg, style = 'Pos')
sns.lineplot(x = 'Year', y = '3PA', data = pf, style = 'Pos')
plt.legend(bbox_to_anchor=(1, 1),labels=['C','PF','SF','SG','PG'], loc='upper left', pr

plt.figure(figsize = (20,15))
ax4 = plt.subplot(gs[2,:])
sns.lineplot(x = 'Year', y = '2PA', data = c, style = 'Pos', label='C')
sns.lineplot(x = 'Year', y = '2PA', data = pf, style = 'Pos', label='PF')
sns.lineplot(x = 'Year', y = '2PA', data = sf, style = 'Pos', label='SF')
sns.lineplot(x = 'Year', y = '2PA', data = sg, style = 'Pos', label='SG')
sns.lineplot(x = 'Year', y = '2PA', data = pg, style = 'Pos', label='PG')
ax4.set_title("Average 2PA by Position since 1980", {'fontsize' : 18.0})

plt.legend(bbox_to_anchor=(1, 1),labels=['C','PF','SF','SG','PG'] ,loc='upper left', pr
```

Out[151…

```
<matplotlib.legend.Legend at 0x2ea2d340>
```







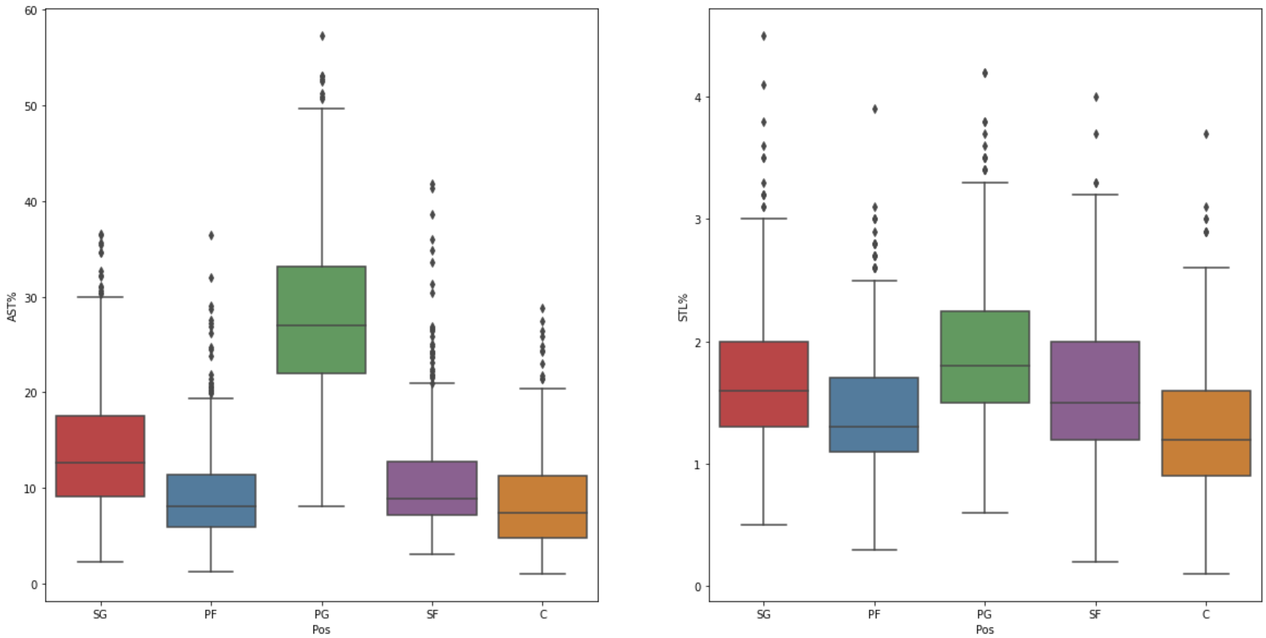
- Guards shoot more threes than forwards or centers
- Centers and power fowards shoot more two pointers than guards or small forwards
- The average 3PA by position has increased significantly since 1980 for every position
- Conversely, the average 2PA by position has decreased since 1980 for every position

In [152…

```
gs = gridspec.GridSpec(1,2)

plt.figure(figsize = (20,10))
ax1 = plt.subplot(gs[0,0])
sns.boxplot(x = 'Pos', y = 'AST%', data = df)


ax2 = plt.subplot(gs[0,1])
sns.boxplot(x = 'Pos', y = 'STL%', data = df)
```

Out[152…

```
<AxesSubplot:xlabel='Pos', ylabel='STL%'>
```



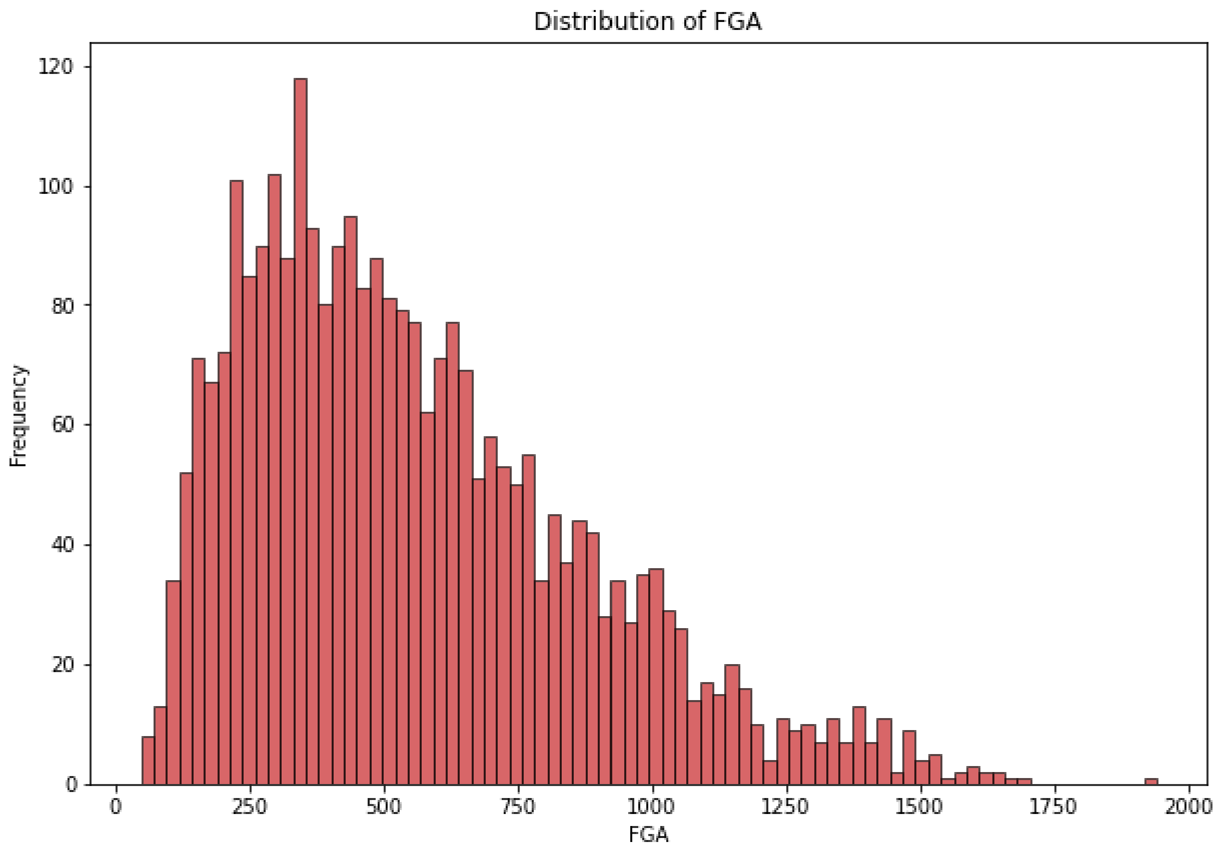
> - Point guards have a much higher AST% and STL% than any other position
> - Guards in general have a higher AST% and STL% than any position because they spend the most time with the ball in their hands, and defending other players that have the ball in their hands

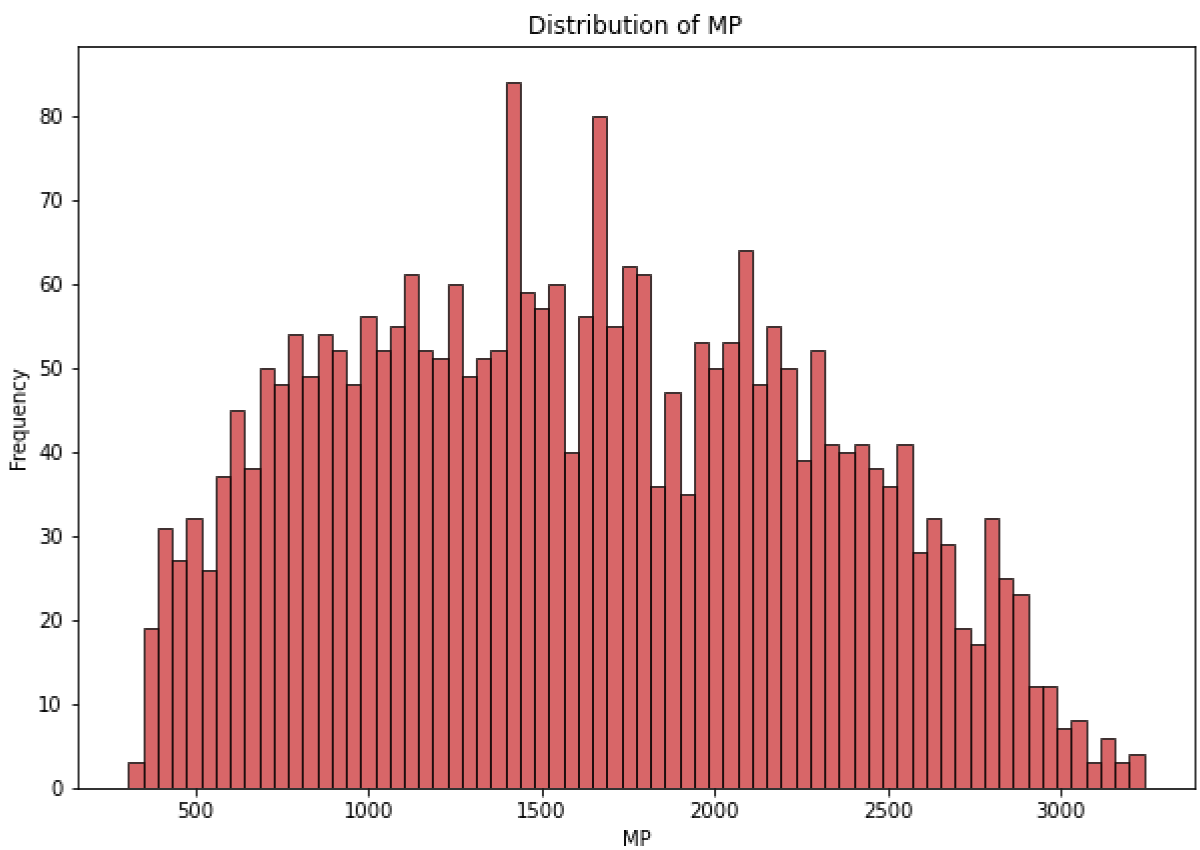
In [153…

```
gs = gridspec.GridSpec(nrows = 2,ncols = 1)

plt.figure(figsize = (10,15))
ax1 = plt.subplot(gs[0,0])
sns.histplot(x='FGA',bins=80, data = df, kde=False, ax = ax1)
ax1.set_ylabel("Frequency")
ax1.set_title("Distribution of FGA")

plt.figure(figsize = (10,15))
ax2 = plt.subplot(gs[1,0])
sns.histplot(x='MP', bins = 70, data = df, kde=False, ax = ax2)
ax2.set_ylabel("Frequency")
ax2.set_title("Distribution of MP")
```

Out[153…

```
Text(0.5, 1.0, 'Distribution of MP')
```

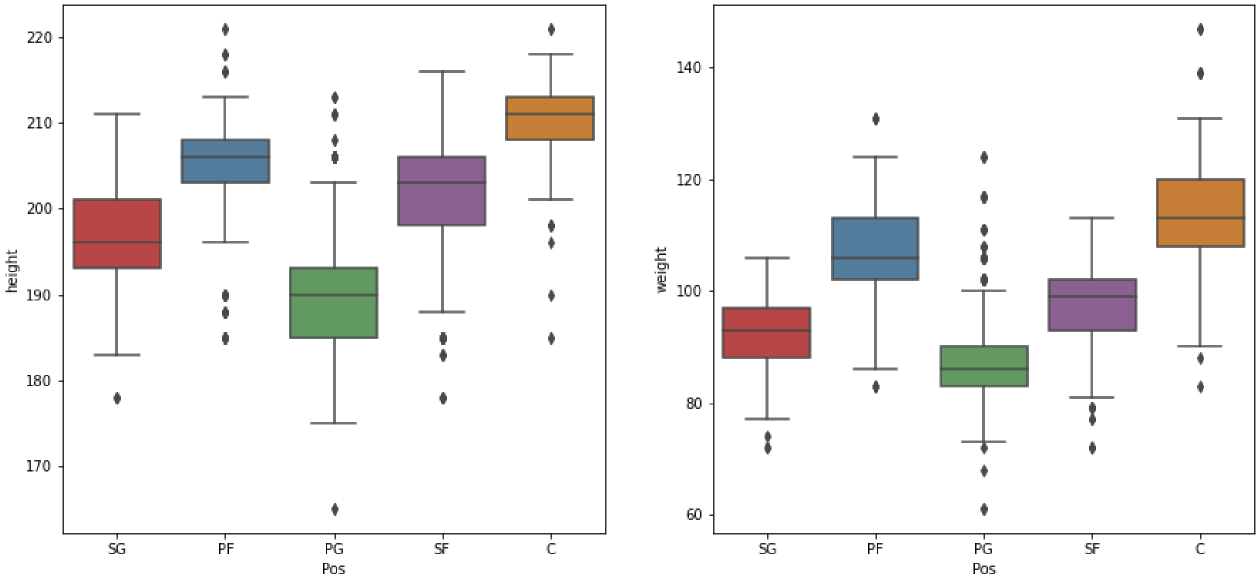




> - FGA (field goal attempts) follows a unimodal distribution, with most players shooting around 400-500 times per season
> - MP (minutes played) follows a bimodal distribution, but it is almost normally distributed

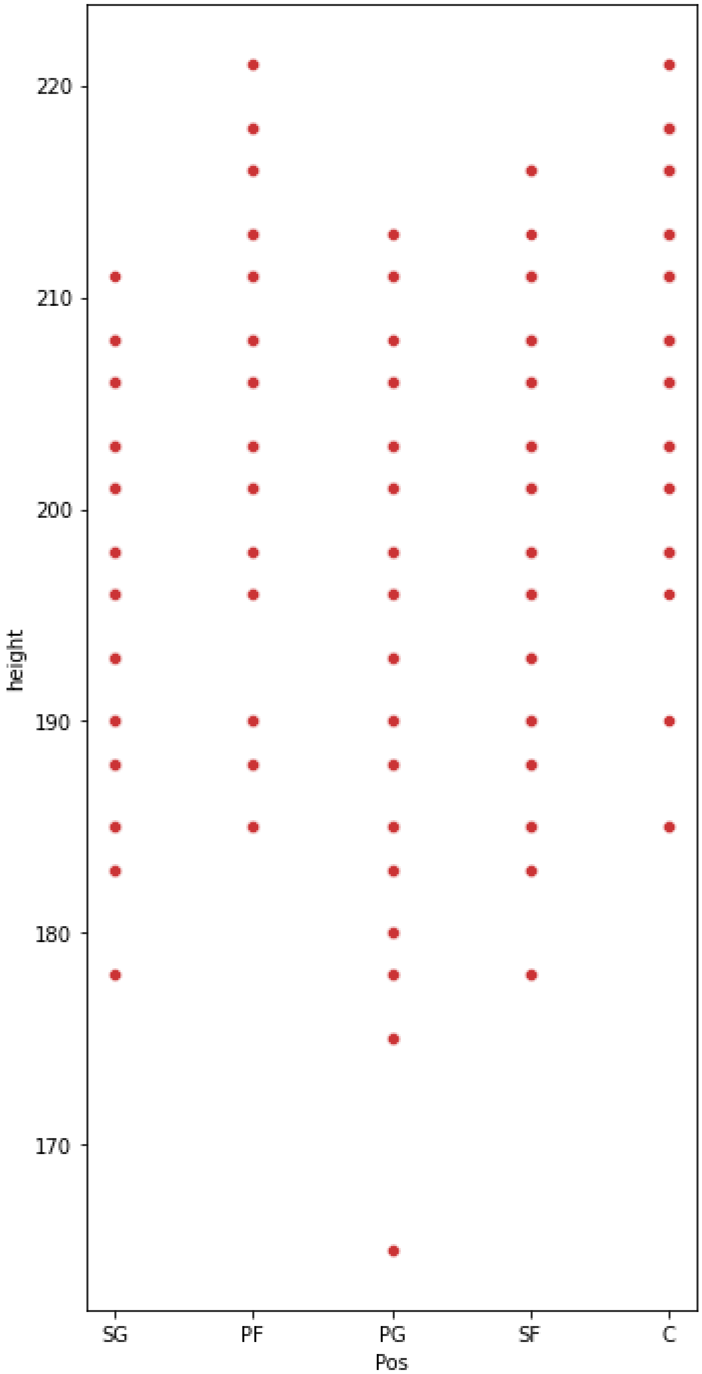
In [154…

```
gs = gridspec.GridSpec(2,2)

plt.figure(figsize = (15,15))
ax1 = plt.subplot(gs[0,0])
sns.boxplot(x='Pos', y='height', data = df)

ax2 = plt.subplot(gs[0,1])
sns.boxplot(x = 'Pos', y='weight', data = df)



plt.show()
```



In [155…

```
gs = gridspec.GridSpec(1,2)

plt.figure(figsize = (12,12))
ax1 = plt.subplot(gs[0,0])
ax1 = sns.scatterplot(x='Pos', y='height', data = df)
```

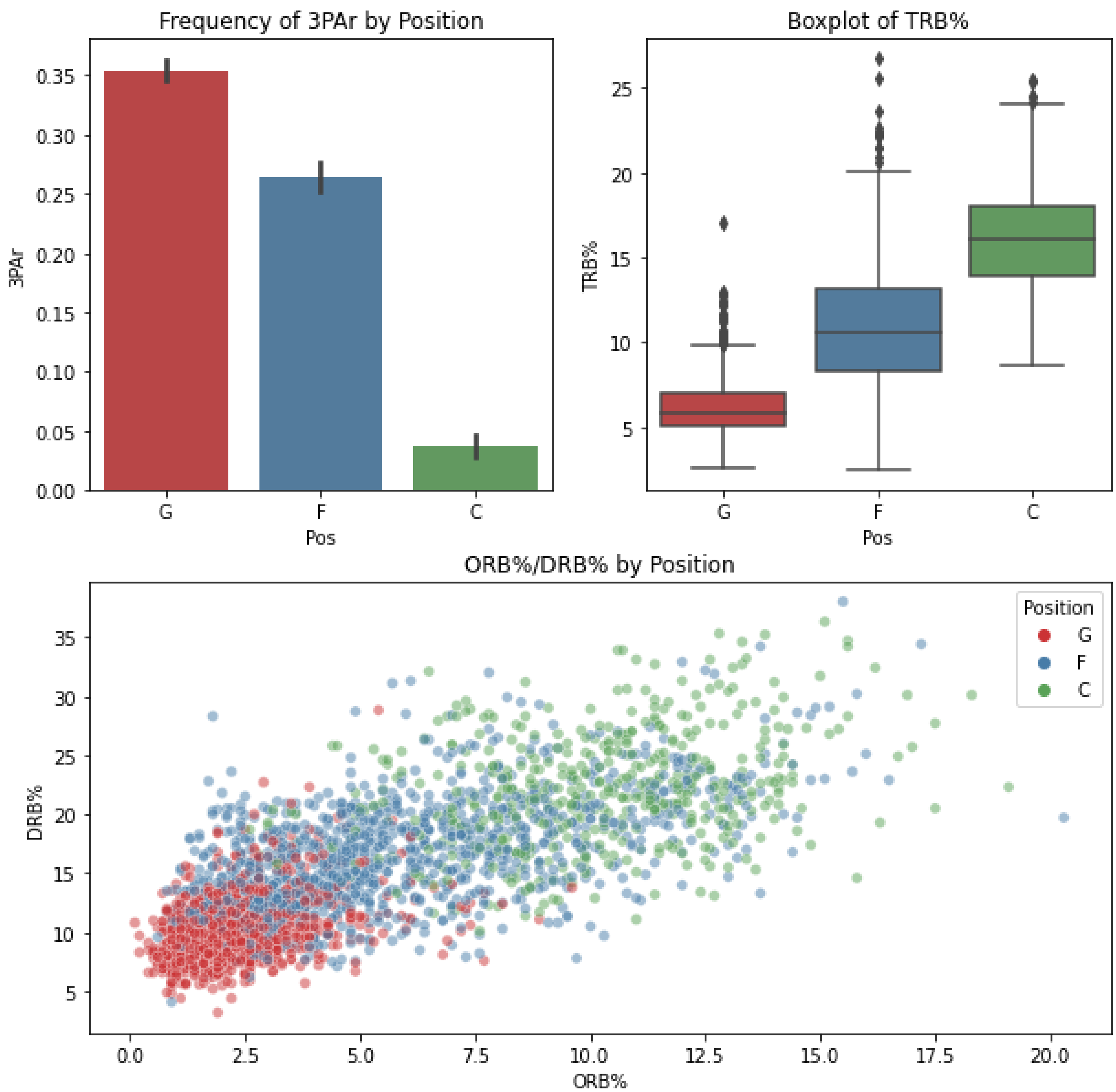


> - As expected, centers and forwards have higher heights and weights than guards, but there are some outliers.
> - The outliers are from the many players that can play the position of guard, but are over 6'6", like Ben Simmons, or LeBron James.

In [156…

```
def temp(pos):
    if (pos == "PG") | (pos == 'SG'):
        return 'G'
    if (pos == 'SF') | (pos == 'PF'):
        return 'F'
    else:
        return 'C'

df['Pos'] = df['Pos'].apply(lambda x : temp(x))
```

```
gs = gridspec.GridSpec(2,2)
plt.figure(figsize = (10,10))
ax1 = plt.subplot(gs[0,0])
ax1.set_title("Frequency of 3PAr by Position")
ax1 = sns.barplot(x='Pos', y= '3PAr', data=df)

ax2 = plt.subplot(gs[0,1])
ax2 = sns.boxplot(x = 'Pos', y = 'TRB%', data = df)
ax2.set_title("Boxplot of TRB%")

ax3 = plt.subplot(gs[1,:])
ax3 = sns.scatterplot(x = 'ORB%', y='DRB%', data = df, hue = 'Pos', alpha = .5)
ax3.set_title("ORB%/DRB% by Position")
plt.legend(title = "Position")
plt.show()
```



- Guards have a high relative 3PAr (3 point per field goal attempt) value
- Centers do not shoot many three pointers relative to total field goal attempts
- Centers and Forwards have a much higher TRB% than guards

> - There are some outliers for both Guards and Forwards for TRB%. Again, this is because there are some guards that are taller than average, and can rebound more effectively because of their height

5. Preprocessing for Machine Learning

In [157…

```
def to_encoded(pos):
    if pos == 'G':
        return 0
    if pos == 'F':
        return 1
    if pos == 'C':
        return 2


df['y'] = df['Pos'].apply(lambda x : to_encoded(x))
df['y'].value_counts()
```

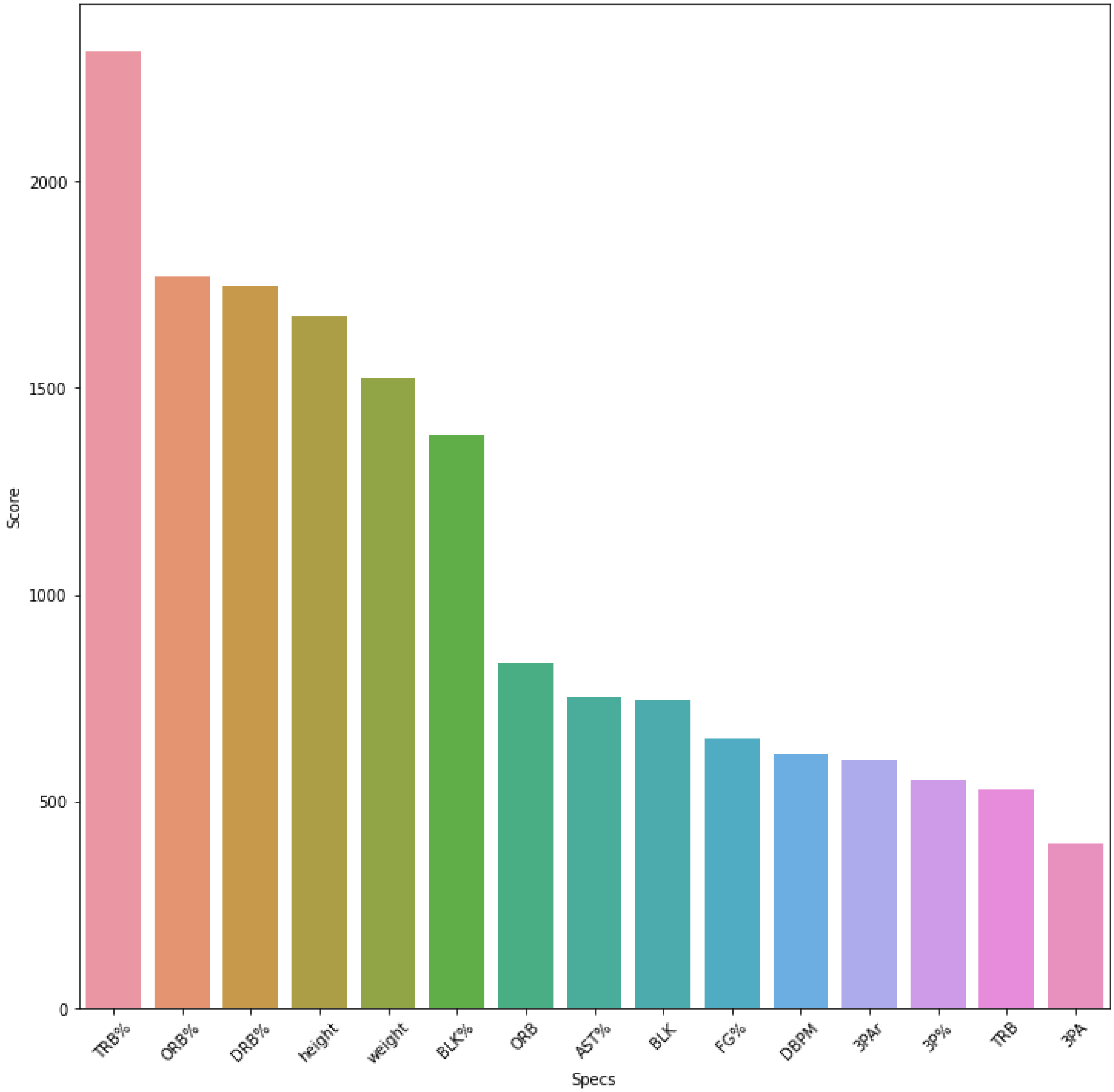
Out[157…

```
0    1208
1    1156
2     481
Name: y, dtype: int64
```

> Encoding Position to integers

In [158…

```
#Kbest features
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
temp_x = df.drop(columns = ['Year', 'Player', 'Pos', 'Tm', 'G', 'GS', 'MP','blanl', 'bl
temp_y = df['y']

kbest = SelectKBest(score_func = f_classif, k = 15)
fit = kbest.fit(temp_x, temp_y)
dfscores = pd.DataFrame(fit.scores_)
cols = pd.DataFrame(temp_x.columns)

scores = pd.concat([cols, dfscores], axis=1)
scores.columns = ['Specs', 'Score']

scores = scores.nlargest(15, 'Score')


plt.figure(figsize = (12,12))
sns.barplot(x = 'Specs', y = 'Score', data = scores)
plt.xticks(rotation = 45)
plt.show()
```



- The most important features make sense; players with a higher rebounding percentage are more likely to be a center.
- Similarly, height and weight makes sense because taller/heavier players are more likely to be a center or forward than a guard
- AST%, 3P%, and 3PA are important because it differentiates guards from centers or forwards

In [159…

```
from sklearn.metrics import confusion_matrix
def conf_mat(y_true, y_pred):
    labels = {"G" : 0,
              "F" : 1,
              "C" : 2}

    mat = confusion_matrix(y_true, y_pred)
    plot = sns.heatmap(mat, annot=True, fmt = "d", linewidths = 1, cmap = "Blues", xtic
    return plot
```

In [160…

```
from sklearn.model_selection import train_test_split

X = df[['TRB%', 'ORB%', 'height', 'weight', 'FG%', 'AST%', 'BLK%', 'ORB', 'BLK', 'DBPM'
y = df['y']

X_train, X_test, y_train, y_test = train_test_split(X,y)
X_train.head()
X_train.columns
```

Out[160…

```
Index(['TRB%', 'ORB%', 'height', 'weight', 'FG%', 'AST%', 'BLK%', 'ORB', 'BLK',
       'DBPM', 'AST', 'TRB', '3PA'],
      dtype='object')
```

Decision Tree Classifier with PCA

In [161…

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier

pipe = Pipeline([
    ('std', StandardScaler()) ,
    ('PCA', PCA()) ,
    ('CLF', DecisionTreeClassifier(class_weight = 'balanced'))
])
print("PCA Parameters:", pipe['PCA'].get_params())
print()
print("KMeans paramters:", pipe['CLF'].get_params())
```

```
PCA Parameters: {'copy': True, 'iterated_power': 'auto', 'n_components': None, 'random_s
tate': None, 'svd_solver': 'auto', 'tol': 0.0, 'whiten': False}

KMeans paramters: {'ccp_alpha': 0.0, 'class_weight': 'balanced', 'criterion': 'gini', 'm
ax_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease':
0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_wei
ght_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': None, 'splitter': 'bes
t'}
```

In [162…

```
cv = StratifiedKFold(shuffle=True)

param_grid = {"PCA__n_components" : [5, 7, 10, 12, 15] ,
             "PCA__whiten" : [True, False] ,
             "CLF__max_depth" : [3, 5, 10, 15] ,
             "CLF__criterion" : ['gini', 'entropy'] ,
             "CLF__max_features" : [5, 7, 9, 12, 15]}

grid_search = GridSearchCV(pipe,
                          param_grid,
                          verbose = 0,
                          scoring = 'accuracy',
                          cv = cv,
                          n_jobs = -1)

grid_search.fit(X_train, y_train)
base_score = grid_search.score(X_test, y_test)
print('Baseline Score:', base_score)
```

```
Baseline Score: 0.8089887640449438
```

In [163…

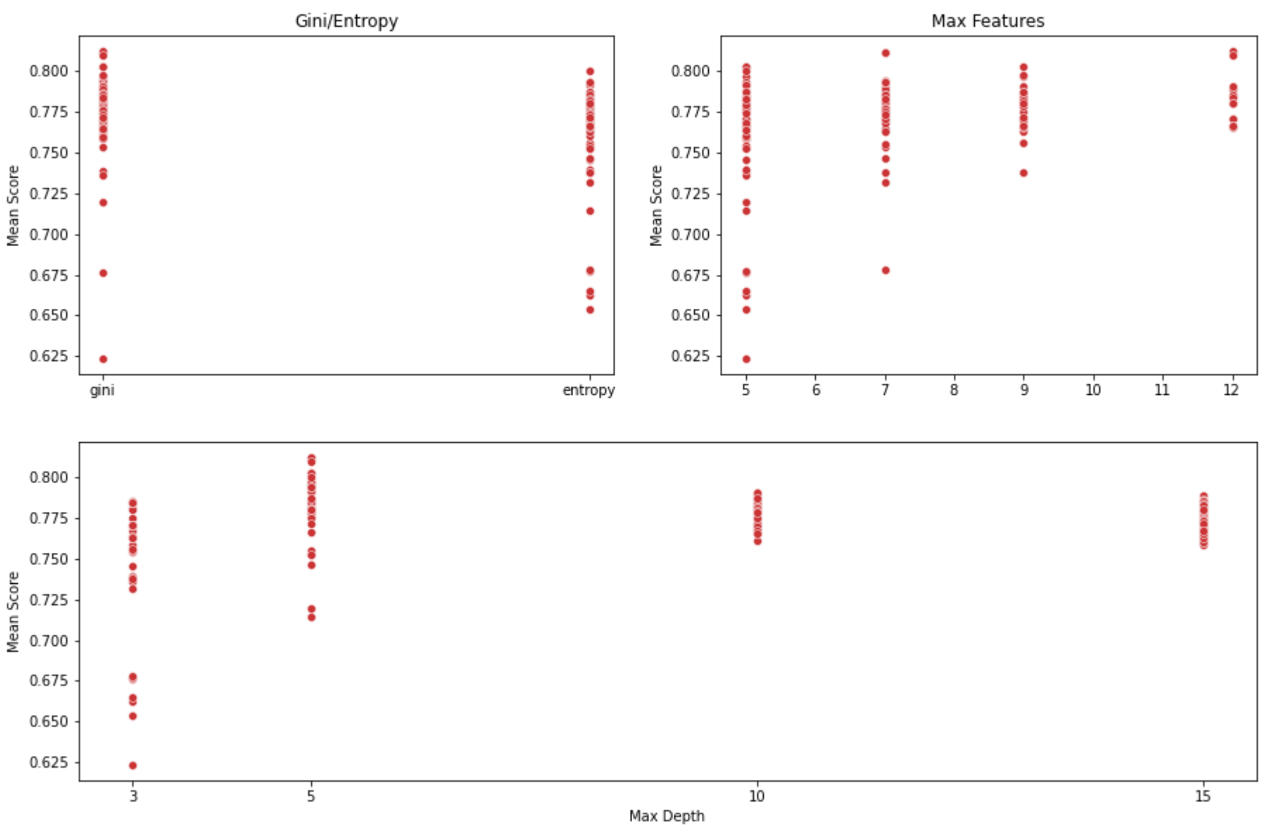
```
grid_search.best_estimator_
```

Out[163…

```
Pipeline(steps=[('std', StandardScaler()),
                ('PCA', PCA(n_components=12, whiten=True)),
                ('CLF',
                 DecisionTreeClassifier(class_weight='balanced', max_depth=5,
                                        max_features=12))])
```

In [164…

```
gs = gridspec.GridSpec(3,2)
data = grid_search.cv_results_

plt.subplots(figsize = (15,15))
ax1 = plt.subplot(gs[0,0])
ax1 = sns.scatterplot(x = 'param_CLF__criterion', y = 'mean_test_score', data = data)
ax1.set_ylabel("Mean Score")
ax1.set_title("Gini/Entropy")
ax1.set_xlabel('')

ax2 = plt.subplot(gs[0,1], sharey=ax1)
ax2 = sns.scatterplot(x = "param_CLF__max_features", y = "mean_test_score", data = data
ax2.set_ylabel('Mean Score')
ax2.set_title("Max Features")
ax2.set_xlabel('')

ax3 = plt.subplot(gs[1,:])
ax3 = sns.scatterplot(x = "param_CLF__max_depth", y = "mean_test_score", data = data)
ax3.set_xticks([3,5,10,15])
ax3.set_ylabel("Mean Score")
ax3.set_xlabel('Max Depth')


plt.show()
```

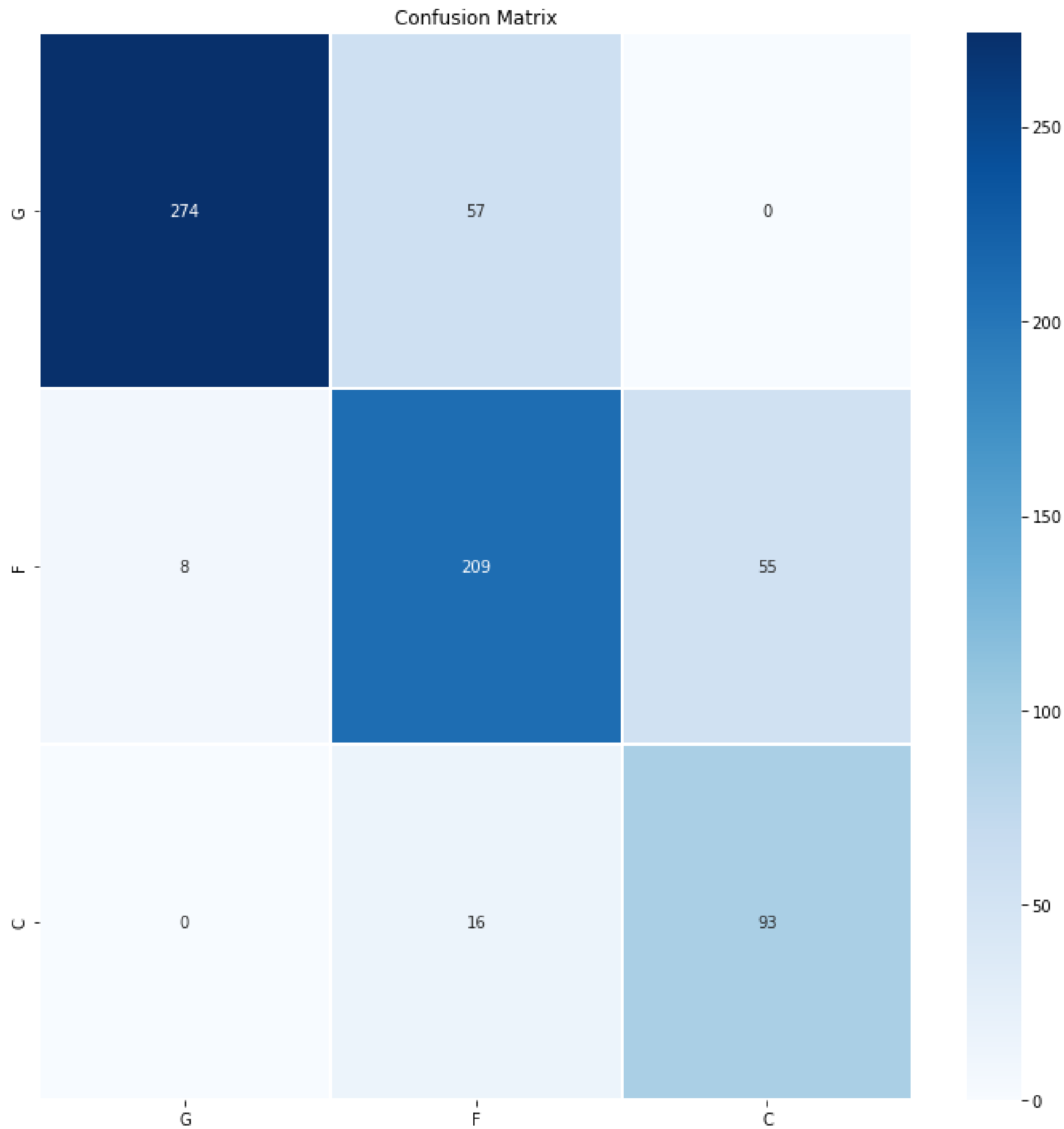


- Using gridsearch, we can see that the highest scoring trials use gini, max features

> of 12, and max depth of 5

In [165…

```
plt.figure(figsize = (10,10))

ax4 = conf_mat(y_test, grid_search.predict(X_test))
ax4.set_title("Confusion Matrix")

plt.tight_layout()
```



> The confusion matrix shows the algorithm had trouble discerning centers from forwards, and guards from forwards. Since the algorithms are using height and weight as a feature, it is most likely getting confused when it sees guards that are taller than average, and incorrectly classifies them as a forward.

Random Forest Classifier

In [167…

```
from sklearn.ensemble import RandomForestClassifier

pipe2 = Pipeline([
    ('std', StandardScaler()),
    ("rf" , RandomForestClassifier(n_jobs = -1, class_weight = 'balanced'))
])


param_grid2 = {'rf__n_estimators' : [200, 250],
              'rf__max_depth' : [10, 12, 15, 17 ,20],
              'rf__min_samples_split' : [5, 7, 10]}

rf_gs = GridSearchCV(pipe2,
                     param_grid2,
                     cv = cv,
                     scoring = 'accuracy'
)

rf_gs.fit(X_train, y_train)
rf_score = rf_gs.score(X_test, y_test)
print("Baseline Score:", rf_score)
```

```
Baseline Score: 0.8946629213483146
```

In [168…

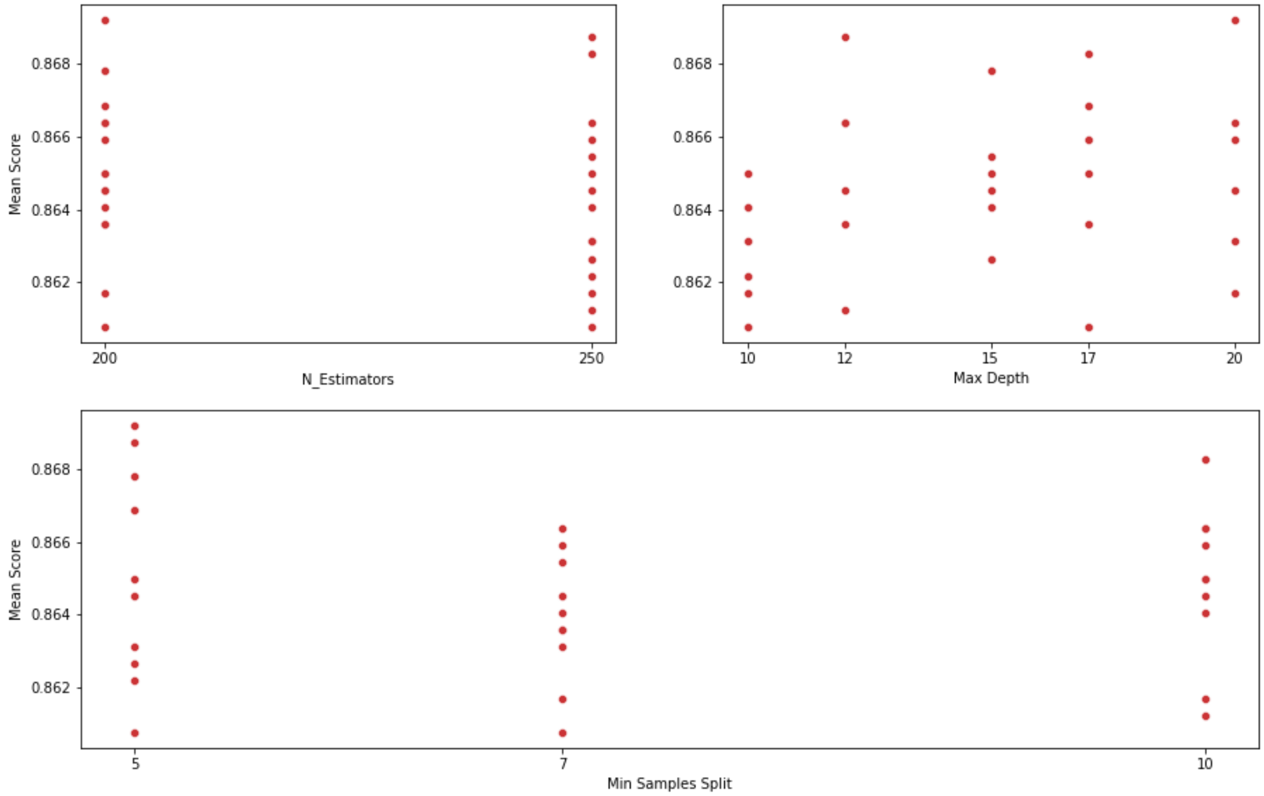
```
rf_gs.best_estimator_
```

Out[168…

```
Pipeline(steps=[('std', StandardScaler()),
                ('rf',
                 RandomForestClassifier(class_weight='balanced', max_depth=20,
                                        min_samples_split=5, n_estimators=200,
                                        n_jobs=-1))])
```

In [169…

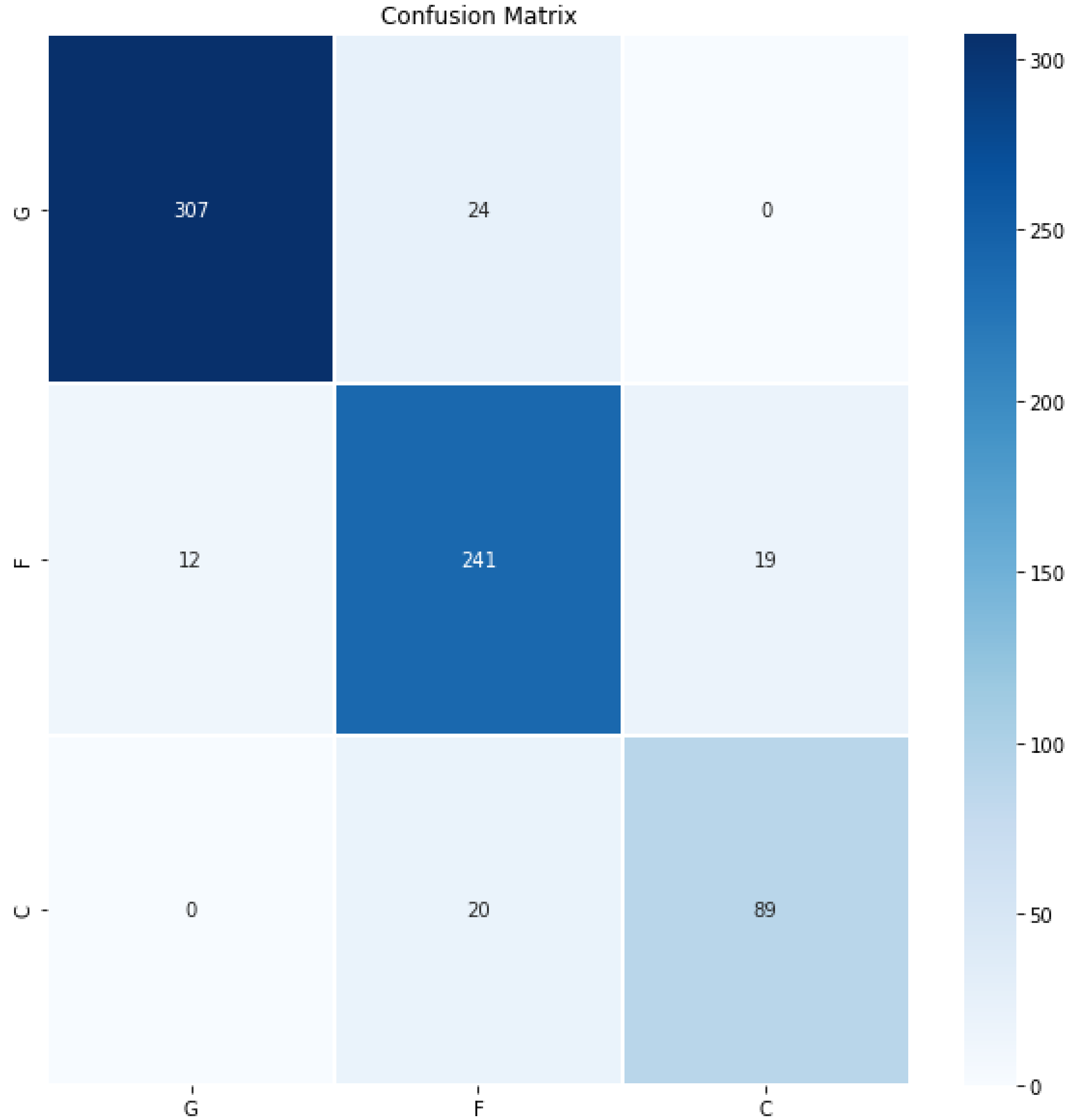
```
gs = gridspec.GridSpec(3,2)
data = rf_gs.cv_results_

plt.figure(figsize = (15,15))
ax1 = plt.subplot(gs[0,0])
sns.scatterplot(x = 'param_rf__n_estimators', y = 'mean_test_score', data = data)
ax1.set_xticks([200, 250])
ax1.set_ylabel("Mean Score")
ax1.set_xlabel("N_Estimators")

ax2 = plt.subplot(gs[0,1])
ax2 = sns.scatterplot(x = "param_rf__max_depth", y = "mean_test_score", data = data)
ax2.set_xticks([10,12,15,17, 20])
ax2.set_ylabel('')
ax2.set_xlabel('Max Depth')

ax3 = plt.subplot(gs[1,:])
ax3 = sns.scatterplot(x = "param_rf__min_samples_split", y = "mean_test_score", data =
ax3.set_xticks([5,7,10])
ax3.set_ylabel("Mean Score")
ax3.set_xlabel("Min Samples Split")



plt.show()
```



> - Using gridsearch, we can see that the highest scoring trials use n_estimators of 200, max depth of 20, and min samples split of 5.

In [170…

```
plt.figure(figsize = (10,10))

ax4 = conf_mat(y_test, rf_gs.predict(X_test))
ax4.set_title("Confusion Matrix")
plt.show()
```



> Random Forest seems to be the most effective at differentiating between centers and forwards.

K-Nearest Neighbors

In [171…

```
from sklearn.neighbors import KNeighborsClassifier

pipe3 = Pipeline([
    ('std', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

param_grid3 = {"knn__n_neighbors" : np.arange(3, 26, 3),
               "knn__p" : [1, 2]
}

knn_gs = GridSearchCV(pipe3,
                     param_grid3,
                     cv=cv,
                     scoring='accuracy')
```

```
knn_gs.fit(X_train, y_train)
knn_score = knn_gs.score(X_test, y_test)
print("Baseline Score:", knn_score)
```

```
Baseline Score: 0.8707865168539326
```

In [172…

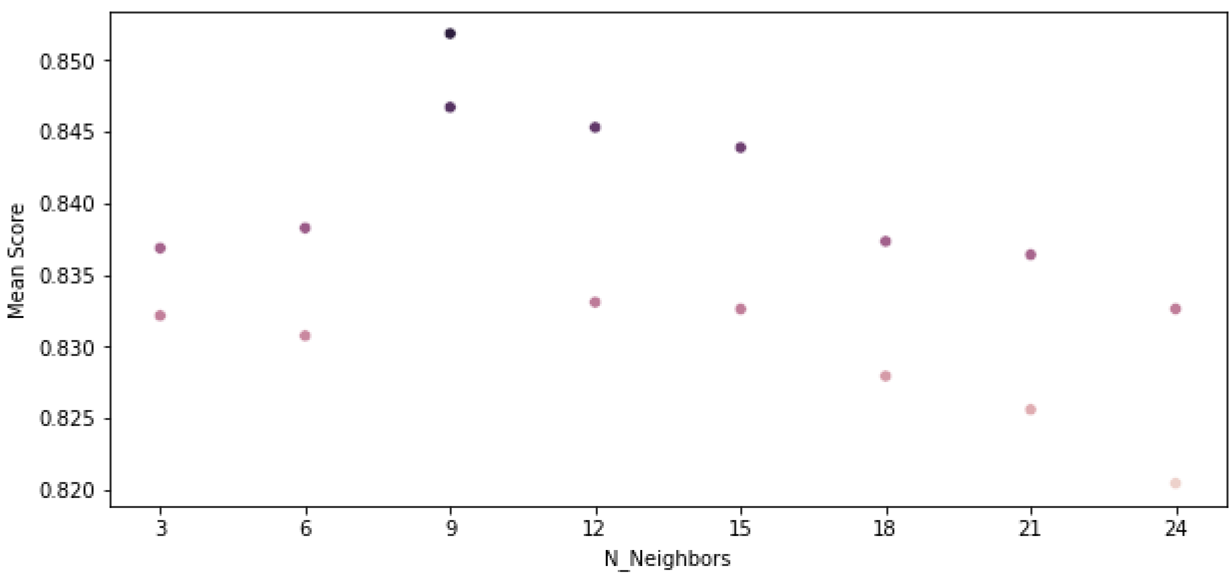
```
knn_gs.best_estimator_
```

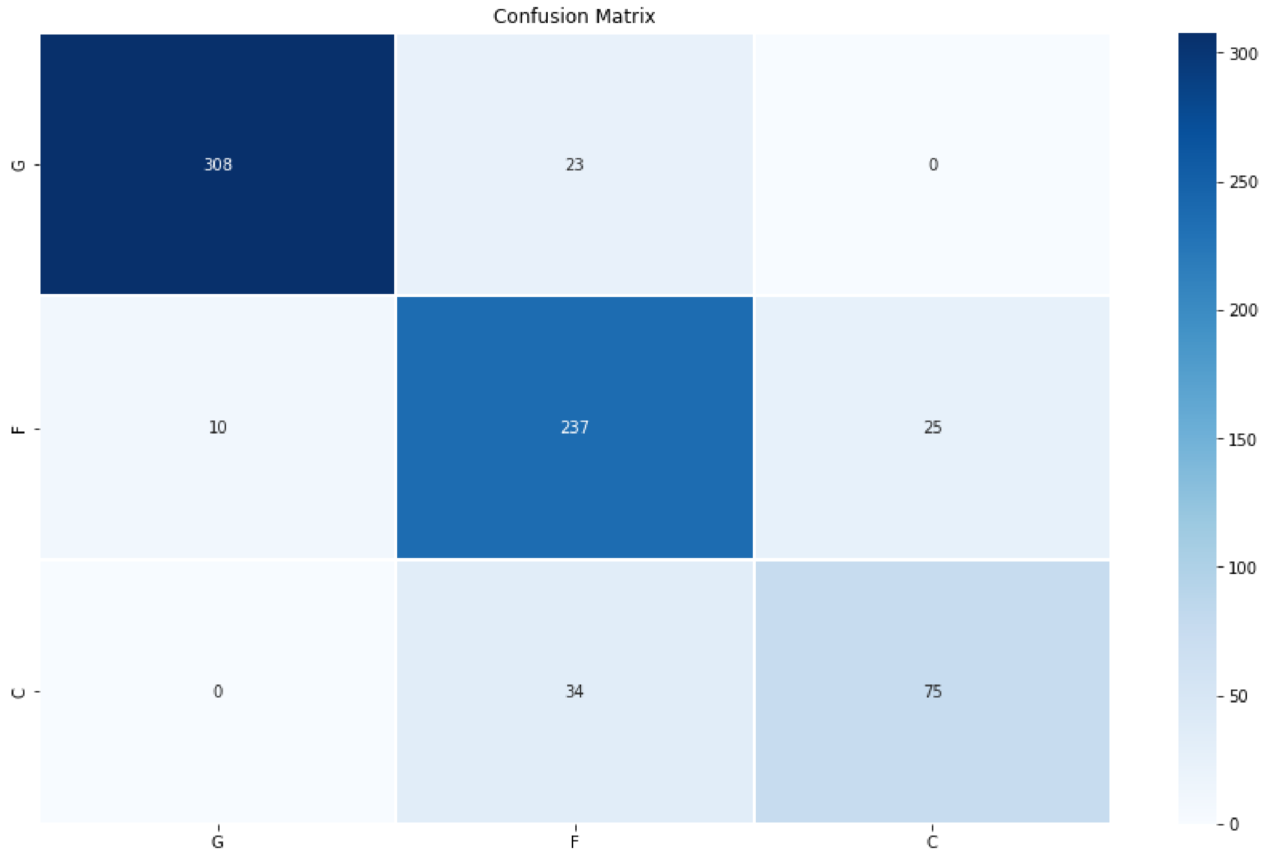
Out[172…

```
Pipeline(steps=[('std', StandardScaler()),
                ('knn', KNeighborsClassifier(n_neighbors=9, p=1))])
```

In [173…

```
gs = gridspec.GridSpec(2,1)
data = knn_gs.cv_results_

plt.figure(figsize = (10,10))
ax1 = plt.subplot(gs[0,0])
sns.scatterplot(x = 'param_knn__n_neighbors', y = 'mean_test_score', data = data, hue="
ax1.set_xticks(np.arange(3, 26, 3))
ax1.set_ylabel('Mean Score')
ax1.set_xlabel('N_Neighbors')
plt.legend([],[], frameon=False)

plt.figure(figsize = (15,20))
ax2 = plt.subplot(gs[1,0])
ax2.set_title("Confusion Matrix")
ax2 = conf_mat(y_test, knn_gs.predict(X_test))
```





In [174...

```
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
pipe4 = Pipeline([
    ('min_max_scale', MinMaxScaler()),
    ('svm', SVC(class_weight = 'balanced', gamma = 'scale', probability=True))

])

param_grid4 = {
    'svm__C':np.arange(3, 15, 1),
    'svm__kernel' : ['rbf', 'poly']
}

svc_gs = GridSearchCV(pipe4,
                     param_grid4,
                     scoring='accuracy',
                     cv=cv,
                     n_jobs=-1)

svc_gs.fit(X_train, y_train)
svc_score = svc_gs.score(X_test, y_test)
print("Baseline Score:", svc_score)
```

```
Baseline Score: 0.8595505617977528
```

In [175...

```
svc_gs.best_estimator_
```

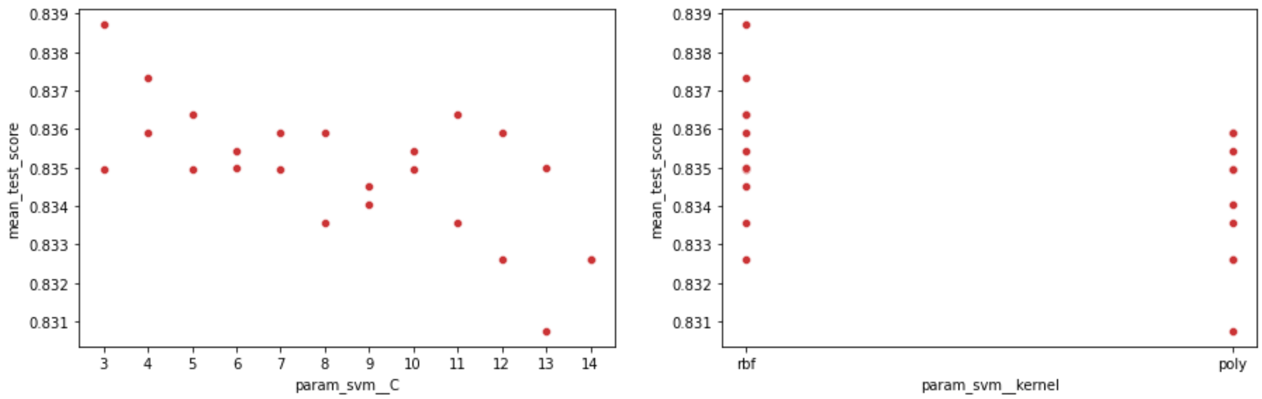
Out[175...

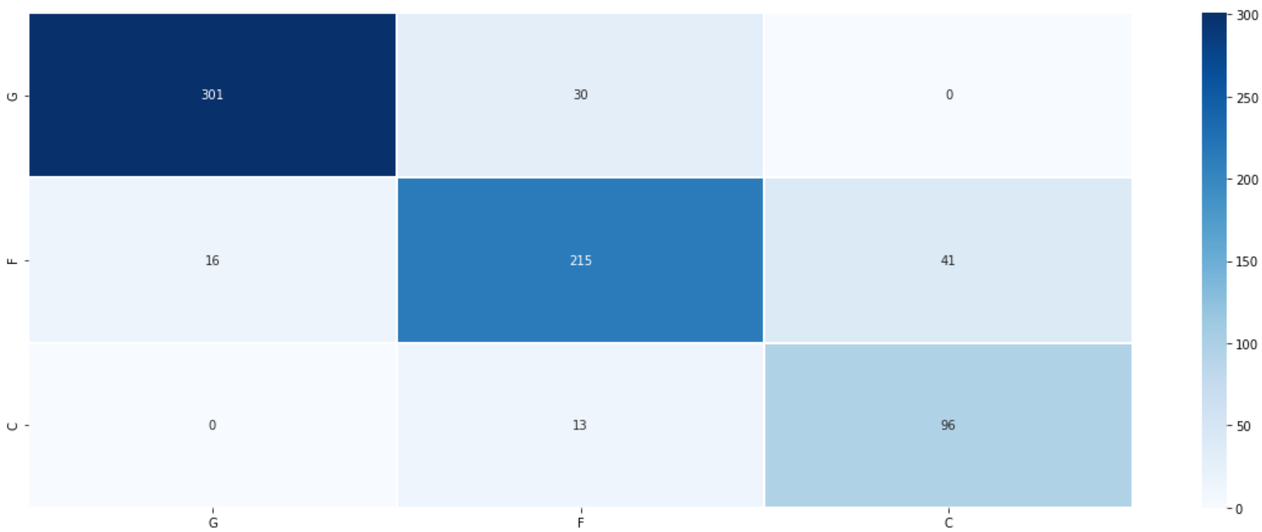
```
Pipeline(steps=[('min_max_scale', MinMaxScaler()),
                ('svm', SVC(C=3, class_weight='balanced', probability=True))])
```

In [176...

```
gs = gridspec.GridSpec(3,2)
data = svc_gs.cv_results_
```

```
plt.figure(figsize = (15,15))
ax1 = plt.subplot(gs[0,0])
sns.scatterplot(x = 'param_svm__C', y = 'mean_test_score', data = data)
ax1.set_xticks(np.arange(3,15,1))

ax2 = plt.subplot(gs[0,1])
ax2 = sns.scatterplot(x = "param_svm__kernel", y = "mean_test_score", data = data)


plt.figure(figsize = (20,25))
ax3 = plt.subplot(gs[1,:])
ax3 = conf_mat(y_test, svc_gs.predict(X_test))
```





In [177…

```
## model comparison ##
```

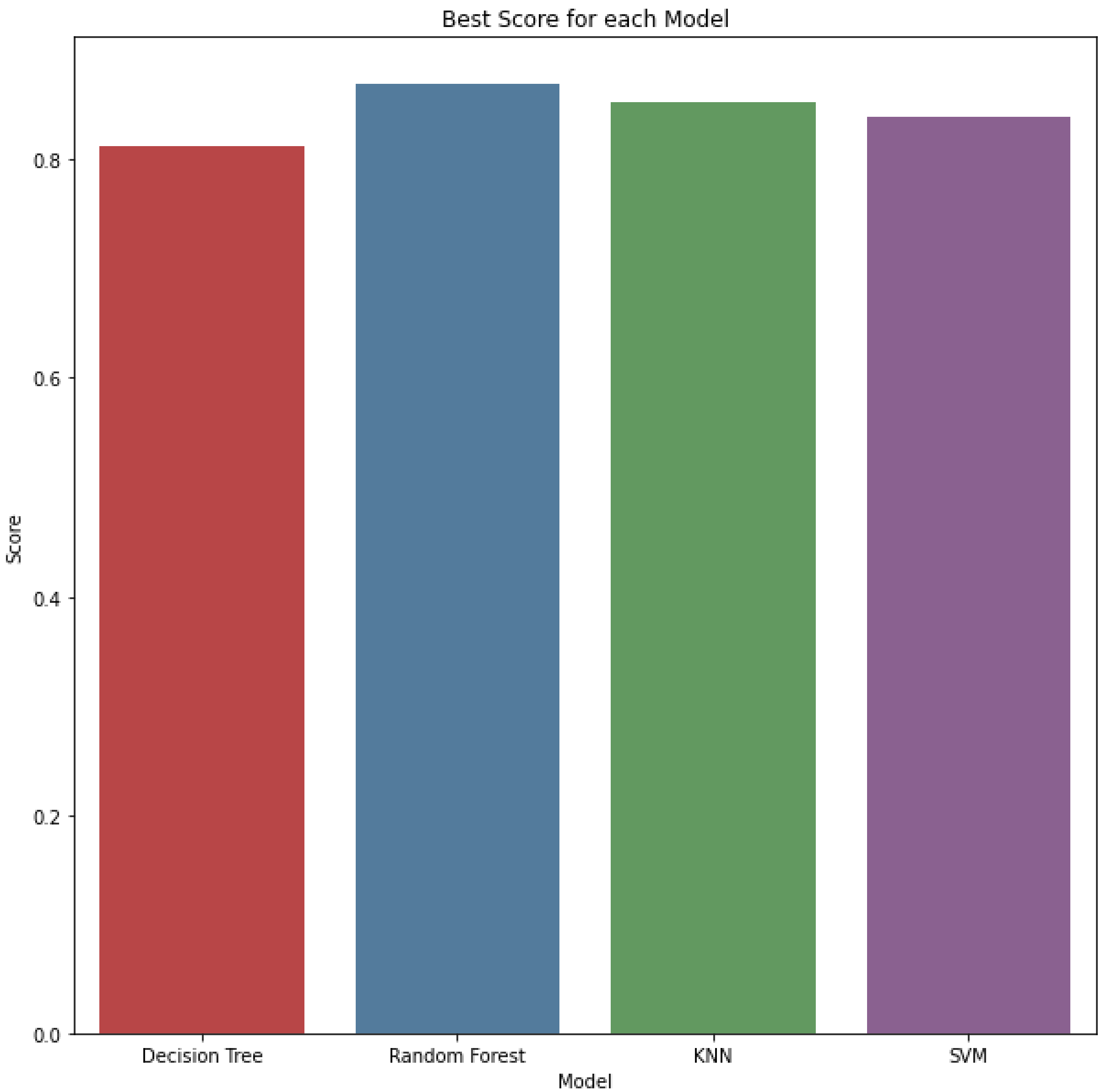
In [178…

```
models = [('Decision Tree', grid_search),
          ("Random Forest", rf_gs),
          ('KNN', knn_gs),
          ('SVM', svc_gs)]
def get_scores(models, y_test):
    temp = []
    for name, gs in models:
        fit = gs.cv_results_['mean_fit_time']
        fit = fit[~np.isnan(fit)]

        test_score = gs.cv_results_['mean_test_score']
        test_score = test_score[~np.isnan(test_score)]

        temp.append((name, gs.best_score_, test_score.mean(), fit.mean()))
```

```
    return temp


model_scores = get_scores(models, y_test)
scores = pd.DataFrame(model_scores, columns = ['Model', 'Best Score', 'Mean Test Score'
```

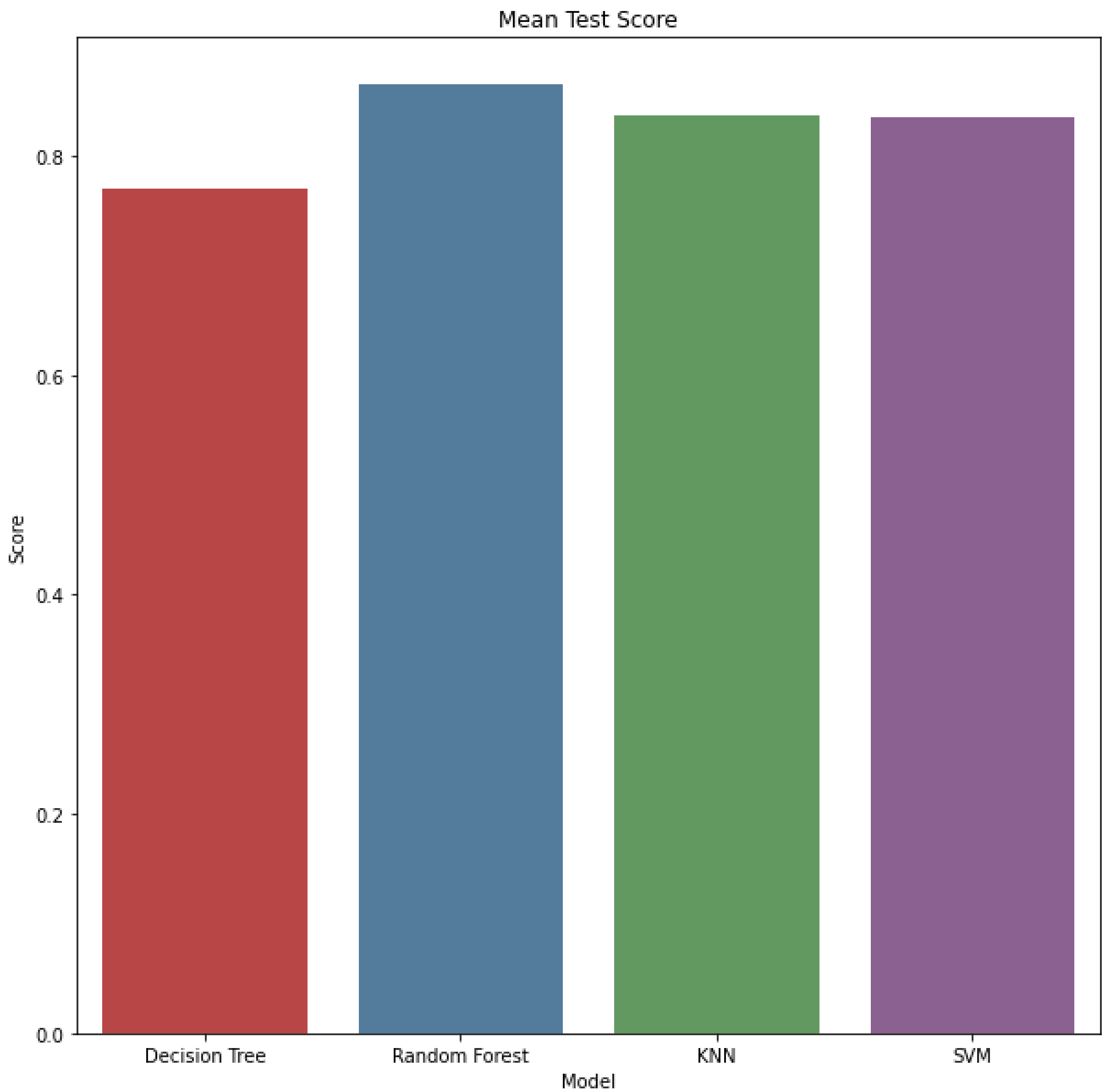
In [187…

```
fig = plt.figure(figsize = (10,10))
sns.barplot(x = 'Model', y = 'Best Score', data = scores)
plt.ylabel('Score')
plt.title('Best Score for each Model')
plt.show()
```



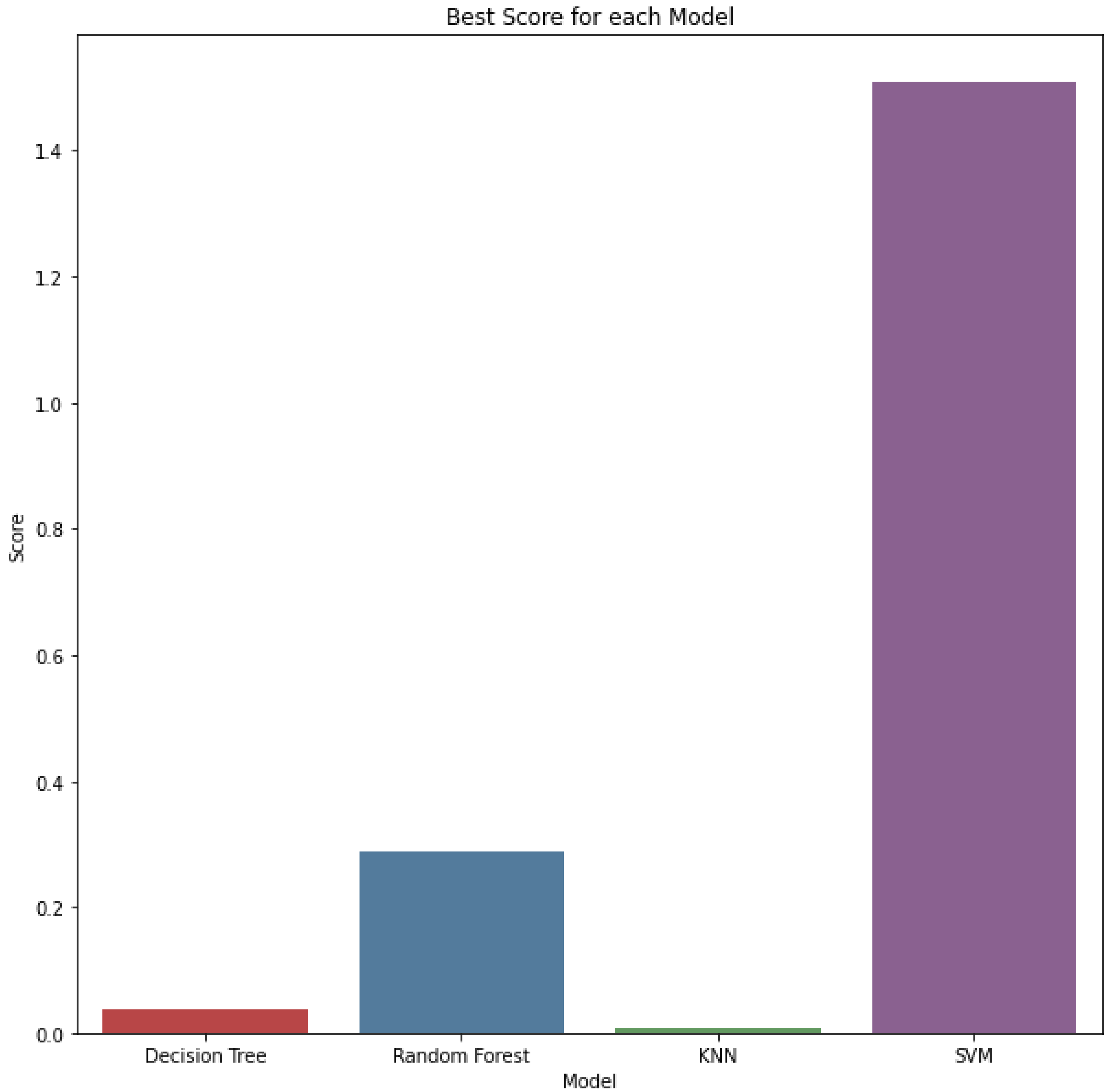
In [183…

```
fig = plt.figure(figsize = (10,10))
sns.barplot(x = 'Model', y = 'Mean Test Score', data = scores)
plt.ylabel('Score')
plt.title('Mean Test Score')
plt.show()
```



In [188…

```
fig = plt.figure(figsize = (10,10))
sns.barplot(x = 'Model', y = 'Mean Fit Time', data = scores)
plt.ylabel('Score')
plt.title('Best Score for each Model')
plt.show()
```


Best Score for each Model
Score
Model
Decision Tree
Random Forest
KNN
SVM
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4