In [284…
```
import matplotlib.pyplot as plt

import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
import statsmodels.formula.api as smf
import seaborn as sns
```

In [285…
```
#Import Fantasy Point Stats for Players. This includes Players, Fantasy Points, 3 game r
#5 Game Rolling Average, 7 Game Rolling Average, and Season Average
df = pd.read_excel(r"C:\Users\djbro\OneDrive\Desktop\DFS\NBA\Updated_Fantasy_GameLogs\Fa
```

In [286…
```
#DataFrame for decision tree
dataset = df[['Player','Last3','Last5','Last7','Avg','FP']]
dataset = dataset.dropna()
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9051 entries, 0 to 9050
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Player  9051 non-null   object 
 1   Last3   9051 non-null   float64
 2   Last5   9051 non-null   float64
 3   Last7   9051 non-null   float64
 4   Avg     9051 non-null   float64
 5   FP      9051 non-null   float64
dtypes: float64(5), object(1)
memory usage: 424.4+ KB
```

In [287…
```
#Read in Draft Kings Players and Salaries for the day's competition
df_players = pd.read_csv(r"C:\Users\djbro\Downloads\DKSalaries (10).csv")
df_players = df_players['Name'].values.tolist()
```

In [288…
```
#remove those players from lineup tonight from the dataset that we train and test on
dataset = dataset[~dataset['Player'].isin(df_players)]
```

In [289…
```
#Name features and labels. Assign features and Labels values
featureNames = ['Last3', 'Last5','Last7','Avg']
labelName = ['FP']

dfFeatures = dataset[['Last3', 'Last5','Last7','Avg']]
dfFeatures.info()

dfLabels = dataset[['FP']]
dfLabels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7600 entries, 0 to 9026
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Last3   7600 non-null   float64
 1   Last5   7600 non-null   float64
 2   Last7   7600 non-null   float64
 3   Avg     7600 non-null   float64
dtypes: float64(4)
memory usage: 296.9 KB
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 7600 entries, 0 to 9026
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   FP      7600 non-null   float64
dtypes: float64(1)
memory usage: 118.8 KB
```

In [290…
```
#Turn labels and features into arrays in order to enter into decision tree
labels = np.array(dfLabels)
features = np.array(dfFeatures)
```

In [291…
```
#Split data into training and testing
train, test, trainLabels, testLabels = train_test_split(features, labels, test_size=(0.1
```

In [292…
```
testLabels.shape
```

Out[292]:
```
(760, 1)
```

In [293…
```
#initalize and assign Decsion Tree Regressor
#tree = DecisionTreeRegressor(random_state=0)
tree = RandomForestRegressor(n_estimators=100)
```

In [294…
```
#Fit to training data. Print size of decision tree if applicable
tree.fit(train, trainLabels)
#print(f'Decision tree has {tree.tree_.node_count} nodes with maximum depth {tree.tree_.
```

```
C:\Users\djbro\AppData\Local\Temp\ipykernel_28148\74403802.py:2: DataConversionWarning:
A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples,), for example using ravel().
  tree.fit(train, trainLabels)
```

Out[294]:
```
RandomForestRegressor()
```

In [295…
```
#Using model to predict both train and test
train_predictions = tree.predict(train)
predictions = tree.predict(test)
```

In [296…
```
test
```

Out[296]:
```
array([[20.41666667, 22.        , 20.07142857, 16.40789474],
       [17.16666667, 16.7       , 17.        , 18.34090909],
       [12.91666667, 13.35      , 13.42857143, 13.08928571],
       ...,
       [15.58333333, 10.2       ,  7.64285714,  6.975     ],
       [11.16666667, 10.65      , 10.64285714, 13.04166667],
       [12.41666667, 12.3       , 11.57142857, 11.43181818]])
```

In [297…
```
#Create pandas dataframe
df1 = pd.DataFrame(test, columns=['Last3', 'Last5','Last7','Avg'])
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 760 entries, 0 to 759
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Last3   760 non-null    float64
 1   Last5   760 non-null    float64
 2   Last7   760 non-null    float64
 3   Avg     760 non-null    float64
dtypes: float64(4)
memory usage: 23.9 KB
```

In [298…
```
#See how our model did. The testLabels are the actual outcomes of fantasy points and the
#what our model came up with for the test data after training on the training data.
#We can see our mean error and and std error
df1['actual'] = testLabels
df1['predicted'] = predictions
df1['error'] = abs(df1['actual'] - df1['predicted'])
df1.describe()
```

Out[298]:

	Last3	Last5	Last7	Avg	actual	predicted	error
count	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000
mean	20.438651	20.299808	20.181801	19.695675	21.160855	20.444890	5.690964
std	13.272383	12.915697	12.743828	12.472147	15.069892	13.260634	4.802264
min	0.000000	0.000000	0.000000	0.000000	-1.000000	-0.203333	0.005373
25%	10.145833	10.387500	10.566071	10.540179	9.750000	10.504955	1.963147
50%	18.416667	18.275000	18.125000	17.523504	18.875000	18.838750	4.520625
75%	28.583333	27.950000	27.571429	27.000000	30.062500	28.475938	8.399250
max	75.500000	71.600000	65.375000	65.375000	81.750000	75.540000	26.785000

In [299…
```
#Now we want our model to predict for players outside of the train and test data
#In our case this will be for players who we will place on our lineups for DFS
```

In [300…
```
#Read in Draft Kings Players and Salaries
df2 = pd.read_csv(r"C:\Users\djbro\Downloads\DKSalaries (10).csv")
df2.head()
```

Out[300]:

	Position	Name + ID	Name	ID	Roster Position	Salary	Game Info	TeamAbbrev	AvgPointsPerGame
0	SG	Luke Kennard (25912249)	Luke Kennard	25912249	SG/G/UTIL	3800	PHX@LAC 12/15/2022 10:30PM ET	LAC	14.96
1	SG/SF	Max Strus (25912176)	Max Strus	25912176	SG/SF/F/G/UTIL	4800	MIA@HOU 12/15/2022 08:00PM ET	MIA	23.19
2	SF	Caleb Martin (25912137)	Caleb Martin	25912137	SF/F/UTIL	5300	MIA@HOU 12/15/2022 08:00PM ET	MIA	23.21
3	PF/C	Jabari Smith Jr. (25912119)	Jabari Smith Jr.	25912119	PF/C/F/UTIL	5500	MIA@HOU 12/15/2022 08:00PM ET	HOU	25.52
4	C	Ivica Zubac (25912135)	Ivica Zubac	25912135	C/UTIL	5300	PHX@LAC 12/15/2022 10:30PM ET	LAC	28.63

In [301…
```
#Rename columns to enable merge in next step
df2.rename(columns = {'Name':'Player'}, inplace = True)
df2.sort_values(by=['AvgPointsPerGame'],ascending=False).head()
```

Out[301]:

	Position	Name + ID	Player	ID	Roster Position	Salary	Game Info	TeamAbbrev	AvgPoint
64	PF/C	Giannis Antetokounmpo (25912006)	Giannis Antetokounmpo	25912006	PF/C/F/UTIL	12200	MIL@MEM 12/15/2022 08:00PM ET	MIL	

63	PG	Ja Morant (25912010)	Ja Morant	25912010	PG/G/UTIL	10400	MIL@MEM 12/15/2022 08:00PM ET	MEM
62	PG/SG	Devin Booker (25912016)	Devin Booker	25912016	PG/SG/G/UTIL	9300	PHX@LAC 12/15/2022 10:30PM ET	PHX
61	PF	Zion Williamson (25912013)	Zion Williamson	25912013	PF/F/UTIL	9800	NOP@UTA 12/15/2022 09:00PM ET	NOP
6	PF	Jimmy Butler (25912025)	Jimmy Butler	25912025	PF/F/UTIL	8700	MIA@HOU 12/15/2022 08:00PM ET	MIA

In [302…

```
#Merge Draft kings Salaries with the dataset we read in that has all fantasy calculation
#This will give us a dataframe with players we can choose for tonight's DFS competition
df3 = df.merge(df2, how='inner', on='Player')
df3.head()
```

Out[302]:

	Unnamed: 0	Player	Date	FP	Last3	Last5	Last7	Avg	Position	Name + ID	ID
0	242	Alperen Sengun	2022-10-19	20.75	20.750000	20.750000	20.750000	20.750000	C	Alperen Sengun (25912095)	25912095
1	243	Alperen Sengun	2022-10-21	46.50	33.625000	33.625000	33.625000	33.625000	C	Alperen Sengun (25912095)	25912095
2	244	Alperen Sengun	2022-10-22	26.25	31.166667	31.166667	31.166667	31.166667	C	Alperen Sengun (25912095)	25912095
3	245	Alperen Sengun	2022-10-28	36.25	36.333333	32.437500	32.437500	32.437500	C	Alperen Sengun (25912095)	25912095
4	246	Alperen Sengun	2022-10-30	31.25	31.250000	32.200000	32.200000	32.200000	C	Alperen Sengun (25912095)	25912095

In [303…

```
#Some data cleaning to ensure we are ready to apply to the ML model
df4 = df3.sort_values(by='Date',ascending = False)
df4 = df4.drop_duplicates(subset=['Player']).reset_index()
df4 = df4.dropna()
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63 entries, 0 to 62
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   index             63 non-null     int64
 1   Unnamed: 0        63 non-null     int64
 2   Player            63 non-null     object
 3   Date              63 non-null     datetime64[ns]
 4   FP                63 non-null     float64
 5   Last3             63 non-null     float64
 6   Last5             63 non-null     float64
 7   Last7             63 non-null     float64
 8   Avg               63 non-null     float64
 9   Position          63 non-null     object
```

```
 10  Name + ID          63 non-null     object
 11  ID                 63 non-null     int64
 12  Roster Position    63 non-null     object
 13  Salary             63 non-null     int64
 14  Game Info          63 non-null     object
 15  TeamAbbrev         63 non-null     object
 16  AvgPointsPerGame   63 non-null     float64
dtypes: datetime64[ns](1), float64(6), int64(4), object(6)
memory usage: 8.5+ KB
```

In [304…
```
#Preparing for ML model
df4 = df4[['Player','Last3','Last5','Last7','Avg','FP']]
df4.sort_values(by=['FP'],ascending=False)
```

Out[304]:

	Player	Last3	Last5	Last7	Avg	FP
11	Giannis Antetokounmpo	48.500000	52.150	52.321429	54.295455	54.50
44	Jarred Vanderbilt	34.750000	30.100	27.392857	24.500000	49.00
58	Tyus Jones	30.583333	31.800	27.071429	21.785714	46.25
61	Ja Morant	55.000000	53.800	52.178571	49.336957	46.00
1	Tyler Herro	36.083333	38.950	40.750000	34.797619	45.25
...	...	...	...	...	...	...
47	Herbert Jones	22.500000	19.750	22.000000	20.916667	9.00
31	Trey Murphy III	13.000000	20.600	23.678571	22.910000	6.00
62	Orlando Robinson	15.875000	15.875	15.875000	15.875000	6.00
20	Devonte' Graham	6.750000	9.350	9.535714	12.000000	5.50
37	Jordan Nwora	3.583333	9.100	8.678571	12.641304	1.25

63 rows × 6 columns

In [305…
```
#Labels for our ML model
df5 = df4[['Last3', 'Last5','Last7','Avg']]
```

In [306…
```
#Turn into array for ML model
test1 = np.array(df5)
```

In [307…
```
#Run out regression tree model and get prediction
game_predictions = tree.predict(test1)
print(game_predictions)
```

```
[23.895       34.7375       8.1375      15.2025      10.285       20.95229167
 14.5475      23.115       27.0925      10.92604167 23.325       51.415
 19.0983631   30.9525      33.445       11.48        39.          21.6525
 47.37        33.2875       7.5675      24.200625    16.09125     11.0975
 17.89285034 39.2125      16.01        14.06        29.9125      35.8075
 16.67416667 12.9975       6.76        16.965       28.5875      29.8975
 19.97416667  2.22        36.755       25.3075      35.1175      28.975
 24.015       28.2475      39.3675      20.6225      17.9605      22.73108333
 20.5175      12.735       21.80854167 26.2275      21.8125      43.89725
 10.4933631   25.64541667 40.172       27.64        26.645       26.0625
 18.15        54.2525      17.56129167]
```

In [308…
```
#Turn Label back into dataframe so we can marge back to player names
df_table = pd.DataFrame(test1, columns=['Last3', 'Last5','Last7','Avg'])
df_table.head()
```

Out[308]:

	Last3	Last5	Last7	Avg
0	25.166667	26.15	23.035714	16.966667
1	36.083333	38.95	40.750000	34.797619
2	9.500000	17.35	20.500000	27.250000
3	14.166667	17.60	19.428571	13.447368
4	13.000000	15.30	14.750000	22.601852

In [309…

```
#Label our predictions
df_table['predictions'] = game_predictions
df_table.head()
```

Out[309]:

	Last3	Last5	Last7	Avg	predictions
0	25.166667	26.15	23.035714	16.966667	23.8950
1	36.083333	38.95	40.750000	34.797619	34.7375
2	9.500000	17.35	20.500000	27.250000	8.1375
3	14.166667	17.60	19.428571	13.447368	15.2025
4	13.000000	15.30	14.750000	22.601852	10.2850

In [310…

```
#Get an idea of the accuracy of our model on new data
df_final = df_table.merge(df4,on=['Last3', 'Last5','Last7','Avg'])
df_final['Error'] = df_final['FP'] - df_final['predictions']
df_final.describe()
```

Out[310]:

	Last3	Last5	Last7	Avg	predictions	FP	Error
count	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000
mean	23.947751	24.158333	23.947751	23.956989	24.104194	24.107143	0.002949
std	10.716416	9.685208	9.748915	9.684526	11.104764	12.462420	6.435564
min	3.583333	9.100000	8.678571	11.421053	2.220000	1.250000	-13.731083
25%	16.270833	17.475000	16.428571	15.825431	16.382708	14.250000	-3.855000
50%	22.416667	24.400000	23.535714	22.601852	23.115000	23.000000	-0.395417
75%	30.333333	28.450000	27.232143	30.391492	29.905000	31.750000	4.966667
max	55.000000	53.800000	52.321429	54.295455	54.252500	54.500000	19.605000

In [311…

```
#Merge onto the list of tonight's players so we have our predictions and see how we perf
df_final = df_final.merge(df4,how='left', on=['Last3', 'Last5','Last7','Avg','FP'])
df_final = df_final.drop_duplicates(subset='Player_x')
df_final = df_final[['Player_x','predictions','Last3','Last5','Last7','Avg','FP','Error'

df_final.sort_values(by='FP',ascending=False).head(10)
```

Out[311]:

	Player	predictions	Last3	Last5	Last7	Avg	FP	Error
11	Giannis Antetokounmpo	51.41500	48.500000	52.15	52.321429	54.295455	54.50	3.08500
44	Jarred Vanderbilt	39.36750	34.750000	30.10	27.392857	24.500000	49.00	9.63250
58	Tyus Jones	26.64500	30.583333	31.80	27.071429	21.785714	46.25	19.60500
61	Ja Morant	54.25250	55.000000	53.80	52.178571	49.336957	46.00	-8.25250

1	Tyler Herro	34.73750	36.083333	38.95	40.750000	34.797619	45.25	10.51250
18	Zion Williamson	47.37000	47.333333	46.20	49.321429	40.750000	44.75	-2.62000
13	Bobby Portis	30.95250	33.833333	29.65	31.250000	31.596154	41.75	10.79750
56	Jaren Jackson Jr.	40.17200	36.750000	35.65	34.571429	35.125000	41.25	1.07800
53	Jimmy Butler	43.89725	42.666667	42.75	40.178571	41.723684	39.25	-4.64725
38	Jalen Green	36.75500	35.250000	32.40	32.321429	33.105769	39.25	2.49500

In [312…
```
#See how our model performed on the new data. The mean error is around 0 and our standar
df_final.describe()
```

Out[312]:

	predictions	Last3	Last5	Last7	Avg	FP	Error
count	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000
mean	24.104194	23.947751	24.158333	23.947751	23.956989	24.107143	0.002949
std	11.104764	10.716416	9.685208	9.748915	9.684526	12.462420	6.435564
min	2.220000	3.583333	9.100000	8.678571	11.421053	1.250000	-13.731083
25%	16.382708	16.270833	17.475000	16.428571	15.825431	14.250000	-3.855000
50%	23.115000	22.416667	24.400000	23.535714	22.601852	23.000000	-0.395417
75%	29.905000	30.333333	28.450000	27.232143	30.391492	31.750000	4.966667
max	54.252500	55.000000	53.800000	52.321429	54.295455	54.500000	19.605000

In [313…
```
#Now the goal is to get an idea how much better our predictive model is compared to just
#Avg Fantasy Points per game. And also how much better it is compared to Projections giv
#DraftKings and a site called FantasyFuel. We will also try averaging my projections, Dr
#Fantasy Fuels. We can see if our ensemble model makes any improvement
```

In [314…
```
#Create dataframe of PLayer and average fantasy points per game that were provided by Dr
# is the DraftKings predictions for these players, not the rolling average
df_DKProj = df2[['Player','AvgPointsPerGame']]
df_DKProj.head()
```

Out[314]:

	Player	AvgPointsPerGame
0	Luke Kennard	14.96
1	Max Strus	23.19
2	Caleb Martin	23.21
3	Jabari Smith Jr.	25.52
4	Ivica Zubac	28.63

In [315…
```
#Merge DraftKings predictions with our predictions
df_final = df_final.merge(df_DKProj,how='left', on=['Player'])
df_final.head()
```

Out[315]:

	Player	predictions	Last3	Last5	Last7	Avg	FP	Error	AvgPointsPerGame
0	Nicolas Batum	23.8950	25.166667	26.15	23.035714	16.966667	29.25	5.3550	16.39
1	Tyler Herro	34.7375	36.083333	38.95	40.750000	34.797619	45.25	10.5125	37.71
2	Ivica Zubac	8.1375	9.500000	17.35	20.500000	27.250000	10.00	1.8625	28.63

3	Robert Covington	15.2025	14.166667	17.60	19.428571	13.447368	12.25	-2.9525	14.33
4	Max Strus	10.2850	13.000000	15.30	14.750000	22.601852	15.50	5.2150	23.19

In [316…
```
#Import Fantasy Projection from a site called Fantasy Fuel
FF_Projections = pd.read_csv(r"C:\Users\djbro\Downloads\DFF_NBA_cheatsheet_2022-12-15.cs
FF_Projections['Player'] = FF_Projections['first_name'] + ' ' + FF_Projections['last_nam
FF_Projections = FF_Projections[['Player','ppg_projection']]
FF_Projections.head()
```

Out[316]:

	Player	ppg_projection
0	Giannis Antetokounmpo	57.5
1	Ja Morant	48.6
2	Zion Williamson	46.5
3	Jimmy Butler	44.0
4	Devin Booker	43.6

In [317…
```
#Merge Fantasy Fuel predictions with our predictions and DraftKings prediction dataframe
df_final = df_final.merge(FF_Projections,how='left', on=['Player'])
df_final.head(5)
```

Out[317]:

	Player	predictions	Last3	Last5	Last7	Avg	FP	Error	AvgPointsPerGame	ppg_project
0	Nicolas Batum	23.8950	25.166667	26.15	23.035714	16.966667	29.25	5.3550	16.39	2
1	Tyler Herro	34.7375	36.083333	38.95	40.750000	34.797619	45.25	10.5125	37.71	4
2	Ivica Zubac	8.1375	9.500000	17.35	20.500000	27.250000	10.00	1.8625	28.63	2
3	Robert Covington	15.2025	14.166667	17.60	19.428571	13.447368	12.25	-2.9525	14.33	1
4	Max Strus	10.2850	13.000000	15.30	14.750000	22.601852	15.50	5.2150	23.19	2

In [318…
```
df_final['AvgError'] = df_final['FP'] - df_final['Avg']
df_final['FFerror'] = df_final['FP'] - df_final['ppg_projection']
df_final['AvgPointsPerGameError'] = df_final['FP'] - df_final['AvgPointsPerGame']
df_final['3ModelAvg'] = (df_final['predictions']+df_final['ppg_projection']+df_final['Av
df_final['3ModelAvgError'] =  df_final['FP'] - df_final['3ModelAvg']
df_final = df_final[['FP','predictions','Error','Avg','AvgError','ppg_projection','FFerr
df_final.describe()
```

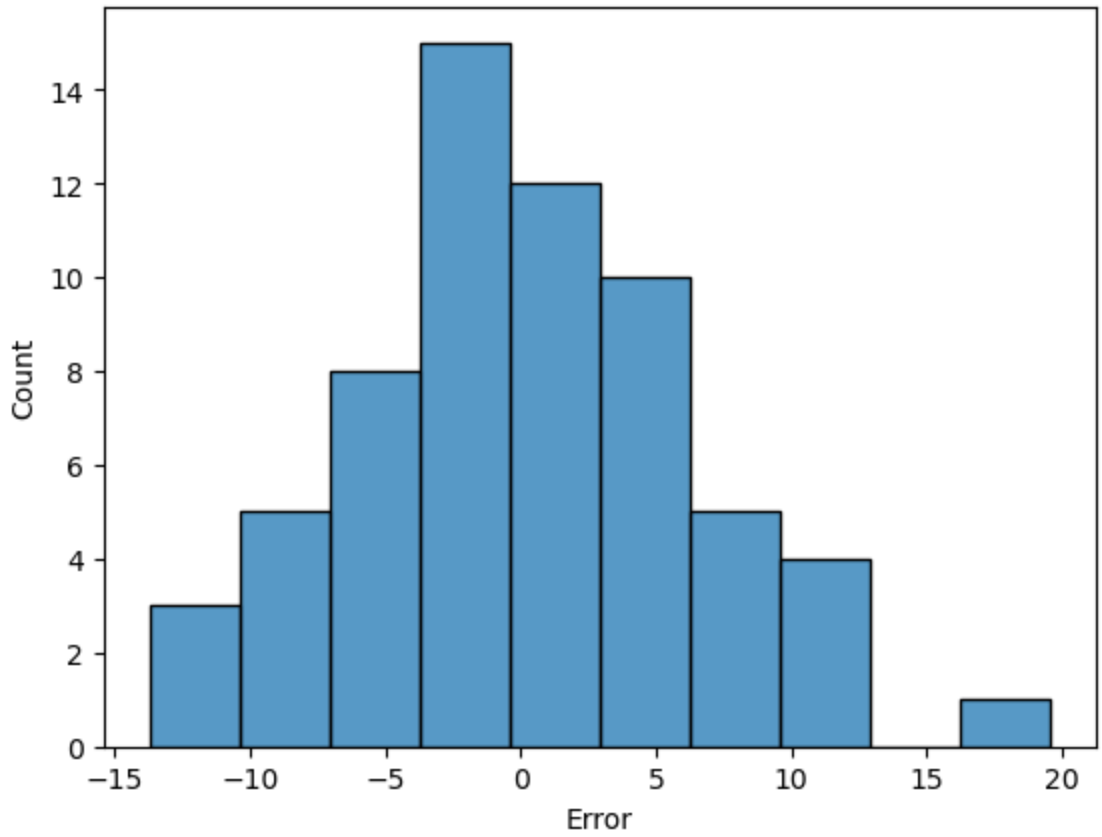
Out[318]:

	FP	predictions	Error	Avg	AvgError	ppg_projection	FFerror	AvgPointsPerGame
count	63.000000	63.000000	63.000000	63.000000	63.000000	62.000000	62.000000	63.000000
mean	24.107143	24.104194	0.002949	23.956989	0.150154	26.162903	-1.687097	24.748254
std	12.462420	11.104764	6.435564	9.684526	9.312586	10.144646	9.544864	10.110135
min	1.250000	2.220000	-13.731083	11.421053	-17.250000	6.800000	-20.050000	12.030000
25%	14.250000	16.382708	-3.855000	15.825431	-4.998043	19.650000	-6.962500	16.345000
50%	23.000000	23.115000	-0.395417	22.601852	-0.500000	23.950000	-1.825000	23.190000
75%	31.750000	29.905000	4.966667	30.391492	4.033333	31.450000	2.687500	31.060000

max	54.500000	54.252500	19.605000	54.295455	24.500000	57.500000	25.250000	56.730000

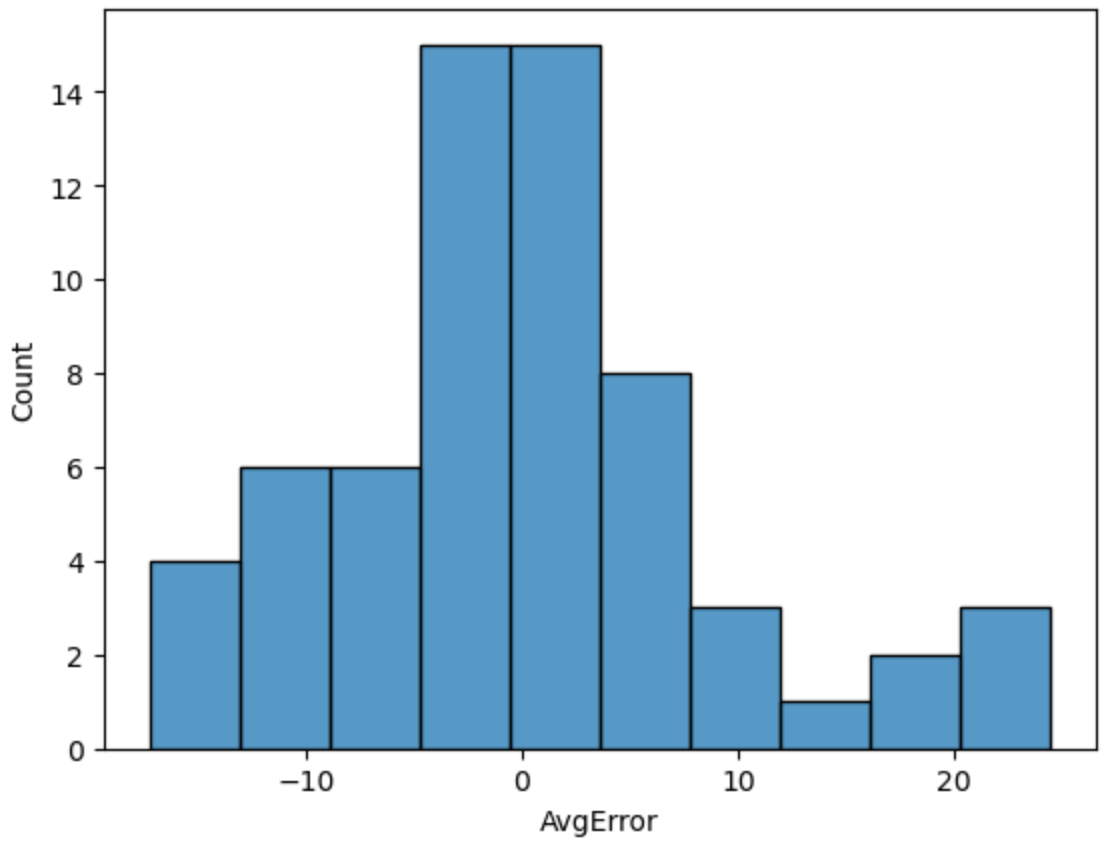
In [319…
```
sns.histplot(data=df_final,x='Error',bins=10)
```

Out[319]:
```
<AxesSubplot:xlabel='Error', ylabel='Count'>
```



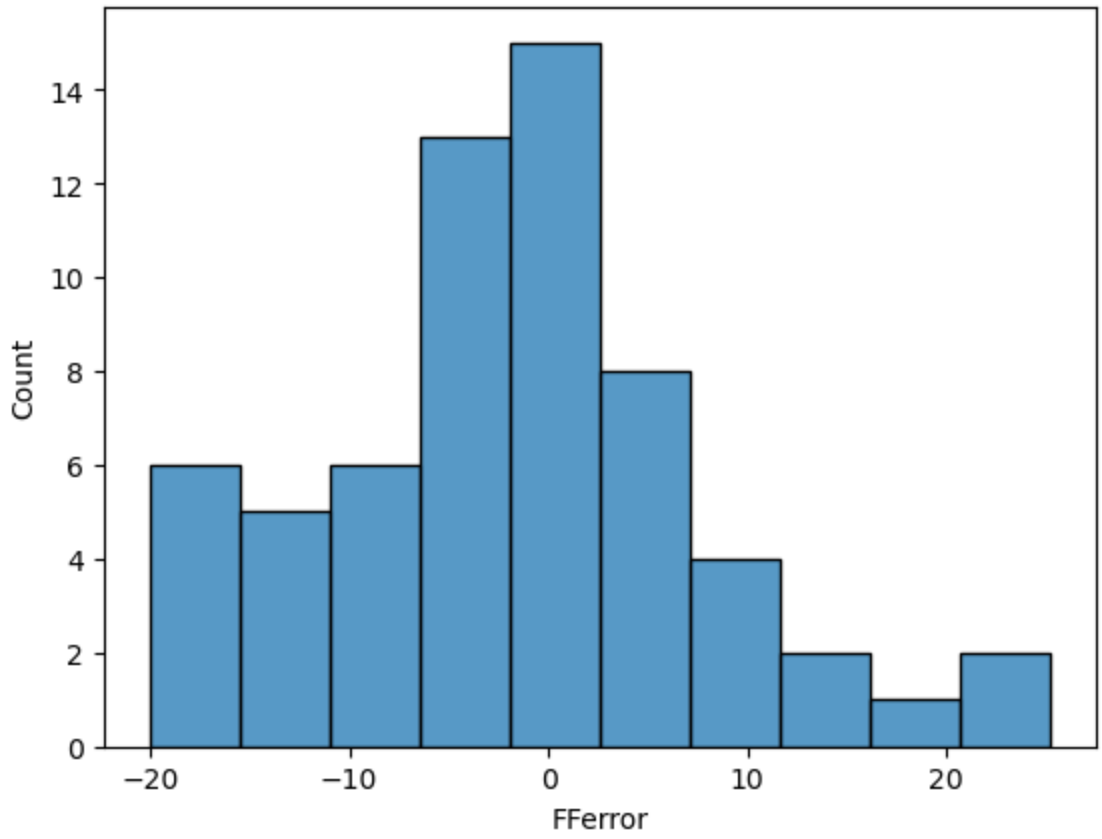
In [320…
```
sns.histplot(data=df_final,x='AvgError',bins=10)
```

Out[320]:
```
<AxesSubplot:xlabel='AvgError', ylabel='Count'>
```



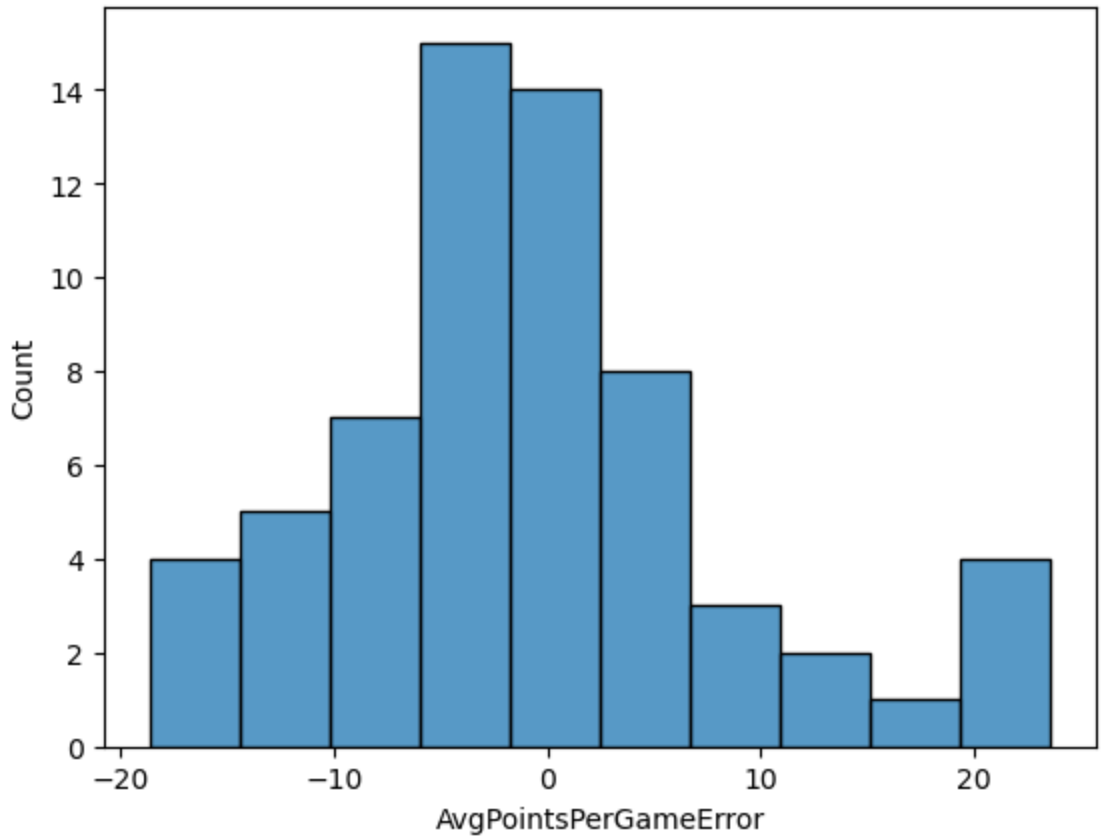
In [321…
```
sns.histplot(data=df_final,x='FFerror',bins=10)
```

Out[321]:
```
<AxesSubplot:xlabel='FFerror', ylabel='Count'>
```



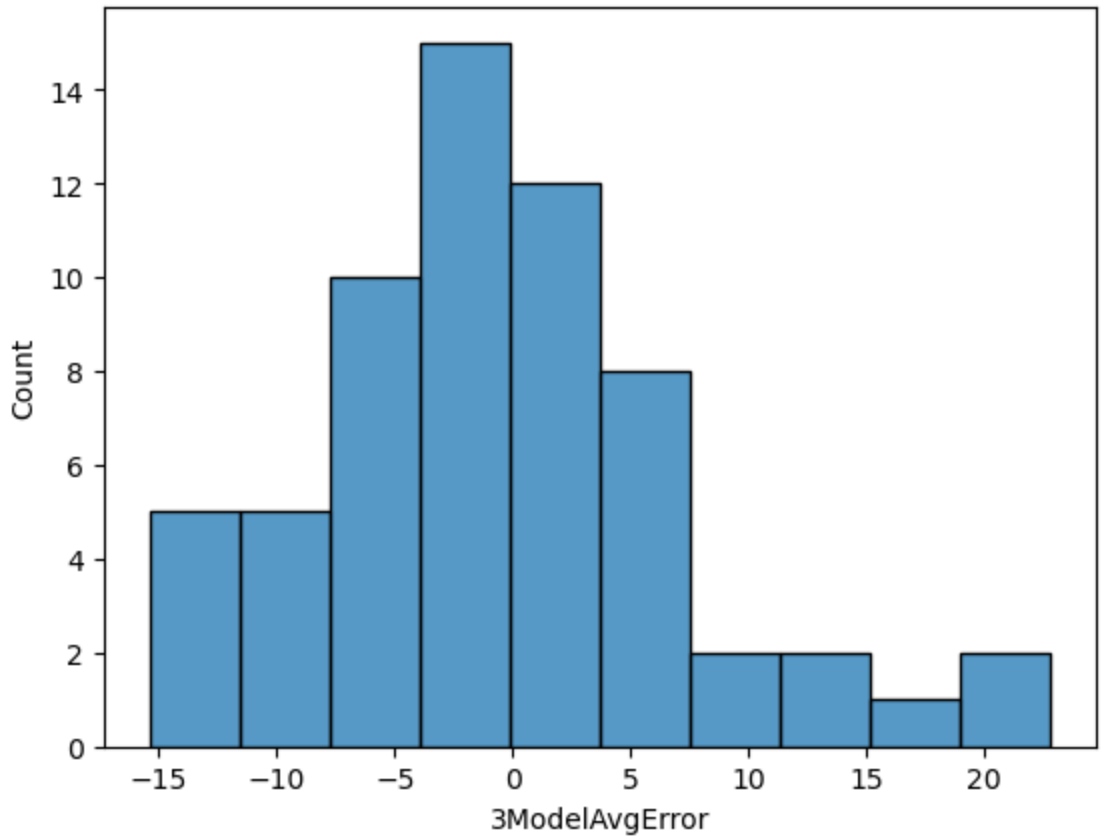
In [322…
```
sns.histplot(data=df_final,x='AvgPointsPerGameError',bins=10)
```

Out[322]:
```
<AxesSubplot:xlabel='AvgPointsPerGameError', ylabel='Count'>
```



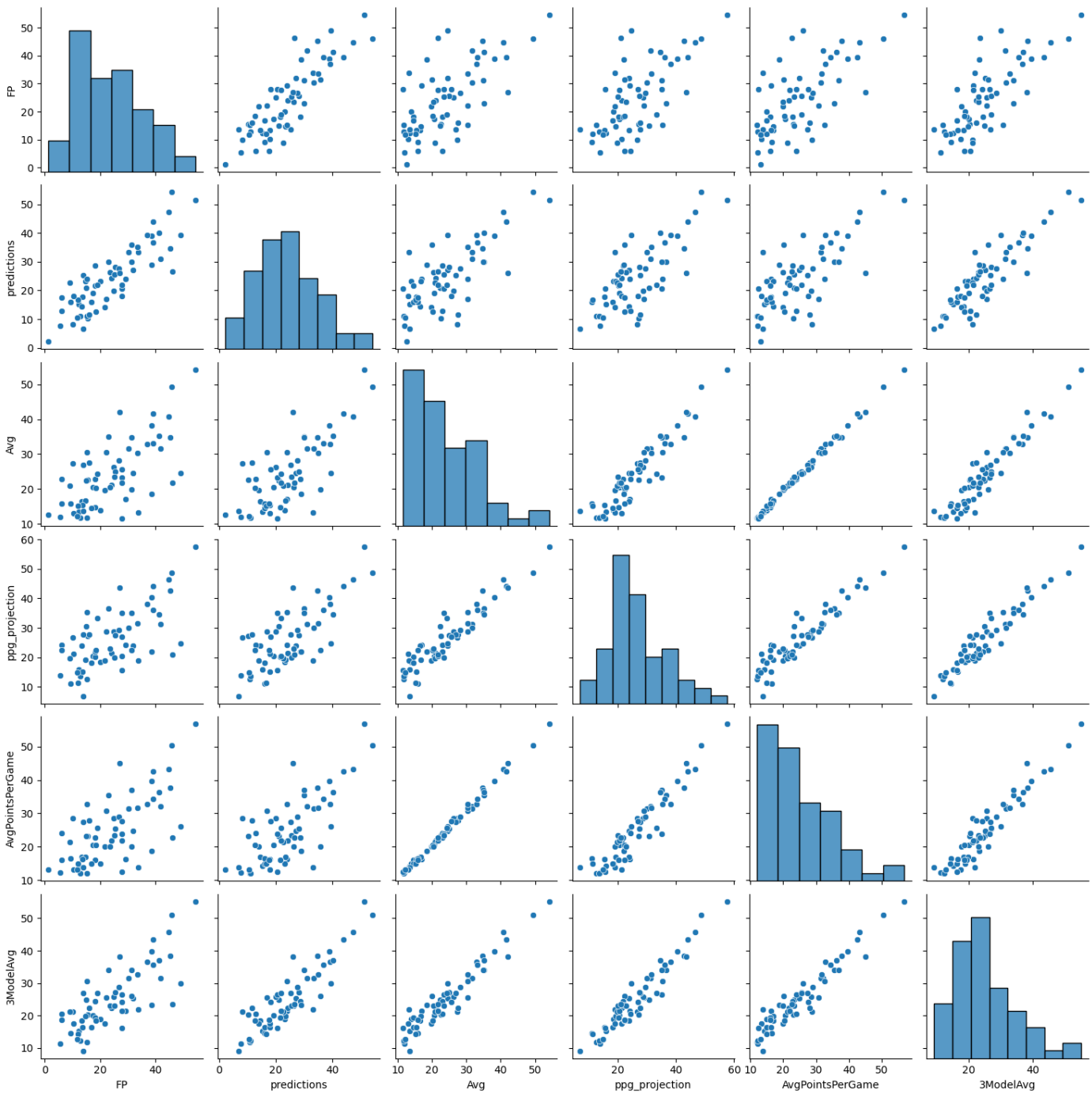
In [323…
```
sns.histplot(data=df_final,x='3ModelAvgError',bins=10)
```

Out[323]: <AxesSubplot:xlabel='3ModelAvgError', ylabel='Count'>



In [324…
```
# UNIVARIATE AND BIVARIATE Visualization via seaborn.
import seaborn as sns
df_final = df_final[['FP','predictions','Avg','ppg_projection','AvgPointsPerGame','3Mode
sns.pairplot(df_final)
```

Out[324]: <seaborn.axisgrid.PairGrid at 0x205fd199dc0>



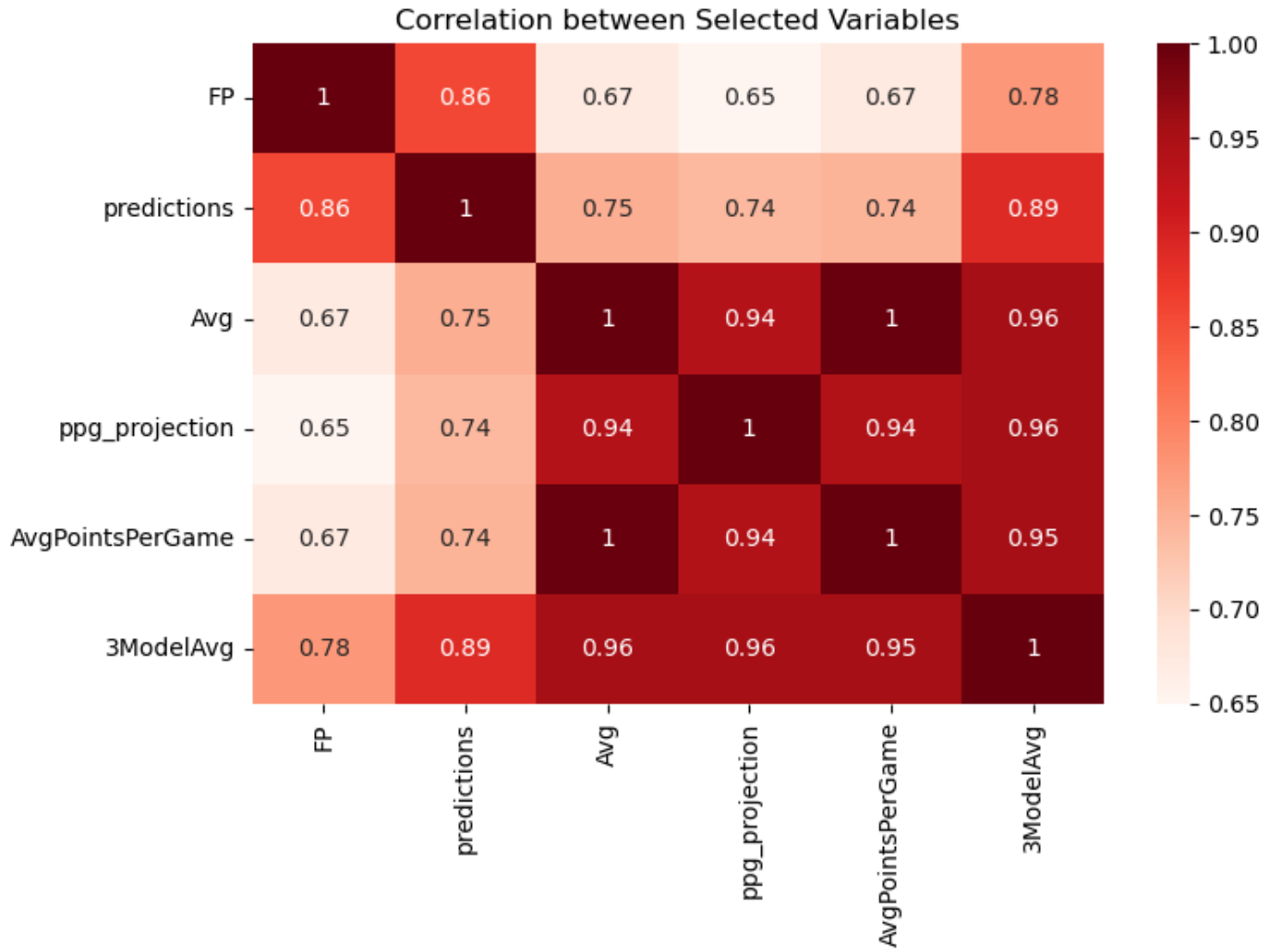
In [325…
```
#Create a correlation matrix to show relationship between select variables
corr_matrix = df_final.corr()
corr_matrix
```

Out[325]:

	FP	predictions	Avg	ppg_projection	AvgPointsPerGame	3ModelAvg
FP	1.000000	0.857025	0.672691	0.649597	0.671628	0.775454
predictions	0.857025	1.000000	0.751805	0.739346	0.744094	0.890153
Avg	0.672691	0.751805	1.000000	0.939444	0.997839	0.955584
ppg_projection	0.649597	0.739346	0.939444	1.000000	0.943536	0.955296
AvgPointsPerGame	0.671628	0.744094	0.997839	0.943536	1.000000	0.954890
3ModelAvg	0.775454	0.890153	0.955584	0.955296	0.954890	1.000000

In [326…
```
#Create a heatmap to visualize correlation
plt.figure(figsize=[8,5])
sns.heatmap(corr_matrix,annot=True,cmap='Reds')
```

```
plt.title("Correlation between Selected Variables")
plt.show()
```



In [327…

```
#Using cross validation to ensure there is not overfitting, and then checking for RMSE a
#of features with Fantasy Points Per Game. The R squared of 0.67 means 67% of variabilit
#and PPG
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import LinearRegression
from math import sqrt
from statsmodels.formula.api import ols

#Create features and target dfs
df_final = df_final.dropna()
X = df_final.drop(columns='FP')
y = df_final.FP

cv = KFold(n_splits=10, random_state=0,shuffle=True)
classifier_pipeline = make_pipeline(StandardScaler(), KNeighborsRegressor(n_neighbors=10
vals = [0.1,0.2,0.3,0.4,0.5,0.6,0.7]
for val in vals:
    features = abs(df_final.corr()["FP"][abs(df_final.corr()["FP"])>val].drop('FP')).ind

    X = df_final.drop(columns='FP')
    X=X[features]

    print(features)
```

```
    y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
    print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),2)))
    print("R_squared: " + str(round(r2_score(y,y_pred),2)))
```

```
['predictions', 'Avg', 'ppg_projection', 'AvgPointsPerGame', '3ModelAvg']
RMSE: 8.0
R_squared: 0.56
['predictions', 'Avg', 'ppg_projection', 'AvgPointsPerGame', '3ModelAvg']
RMSE: 8.0
R_squared: 0.56
['predictions', 'Avg', 'ppg_projection', 'AvgPointsPerGame', '3ModelAvg']
RMSE: 8.0
R_squared: 0.56
['predictions', 'Avg', 'ppg_projection', 'AvgPointsPerGame', '3ModelAvg']
RMSE: 8.0
R_squared: 0.56
['predictions', 'Avg', 'ppg_projection', 'AvgPointsPerGame', '3ModelAvg']
RMSE: 8.0
R_squared: 0.56
['predictions', 'Avg', 'ppg_projection', 'AvgPointsPerGame', '3ModelAvg']
RMSE: 8.0
R_squared: 0.56
['predictions', '3ModelAvg']
RMSE: 7.34
R_squared: 0.63
```

In [328…

```
#Linear Regression Model with FPPG and PPG and ASTPG
FP_vs_features = ols('FP ~ predictions',data=df_final)
FP_vs_features = FP_vs_features.fit()
print(FP_vs_features.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     FP   R-squared:                       0.719
Model:                            OLS   Adj. R-squared:                  0.715
Method:                 Least Squares   F-statistic:                     153.8
Date:                Fri, 16 Dec 2022   Prob (F-statistic):           3.33e-18
Time:                        12:39:59   Log-Likelihood:                -203.23
No. Observations:                  62   AIC:                             410.5
Df Residuals:                      60   BIC:                             414.7
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept       1.0902      2.060      0.529      0.599      -3.029       5.210
predictions     0.9562      0.077     12.402      0.000       0.802       1.110
==============================================================================
Omnibus:                        1.543   Durbin-Watson:                   2.059
Prob(Omnibus):                  0.462   Jarque-Bera (JB):                0.966
Skew:                           0.287   Prob(JB):                        0.617
Kurtosis:                       3.211   Cond. No.                         66.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```