Final Project Submission
Student name: Qilun Chen, Evan Serrano
Student pace: full time
Scheduled project review date/time: April/1/2022
Instructor name: Praveen Gowtham, Joe Comeaux
Blog post URL:https://github.com/nkbuddy/dsc-phase-3-project-NBA

Table of Contents

STEP 1: Define the Problem

Analysis of what sorts of external feature were likely to make a shot in NBA. In particular, we ask you to apply the tools of machine learning to predict which external feature make the shot.

Binary classification problem

Step 2: Gather the Data

The dataset is also given to us at kaggle https://www.kaggle.com/datasets/dansbecker/nba-shot-logs

Step 3: Prepare Data for Consumption

3.1 Import Libraries

In [111…

```
import sys
import pandas as pd
import matplotlib
import numpy as np
import scipy as sp
import IPython
import sklearn
import random
import time
import warnings
import datetime
warnings.filterwarnings('ignore')
from subprocess import check_output
```

3.11 Load Data Modelling Libraries

In [112…
```
from sklearn import svm, tree, linear_model, neighbors, naive_bayes, ensemble
from xgboost import XGBClassifier
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.datasets import load_boston
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import seaborn as sns
from scipy import stats as stats
from pandas.plotting import scatter_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_
%matplotlib inline
mpl.style.use('ggplot')
sns.set_style('white')
```

3.2 Meet and Greet Data

1)The FGM variable is outcome or dependent variable. It is a binary nominal datatype of 1 for make and 0 for missed. All other variables are potential or independent variables. Its important to note, more predictor variables do not make a better model, but the right variables.
2)The GameID, match, win, Final_margin, shot_number, and PTS are assumed to be random unique identifiers, that have no impact on the outcome veribale. Thus, they will be excluded from analysis.
4)The Name and shot_number variable are nominal datatype. It could be used in feature engineering to derive the who the best defender is, the hot-hand hypothesis, etc. Since these variables already exist, we'll make use of it to see if player makes a difference.
5)The Location and PTS_type variables are a nominal datatype. They will be converted to dummy variables for mathematical calculations.
6)The game_clock, shot_clock, dribbles, touch_time, shot_distance, and closet_defender_distance variable are continuous quantitative datatypes.

In [113…
```
df = pd.read_csv('shot_logs.csv')
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128069 entries, 0 to 128068
Data columns (total 21 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   GAME_ID                     128069 non-null  int64  
 1   MATCHUP                     128069 non-null  object 
 2   LOCATION                    128069 non-null  object 
 3   W                           128069 non-null  object 
 4   FINAL_MARGIN                128069 non-null  int64  
 5   SHOT_NUMBER                 128069 non-null  int64  
 6   PERIOD                      128069 non-null  int64  
 7   GAME_CLOCK                  128069 non-null  object 
 8   SHOT_CLOCK                  122502 non-null  float64
 9   DRIBBLES                    128069 non-null  int64  
 10  TOUCH_TIME                  128069 non-null  float64
 11  SHOT_DIST                   128069 non-null  float64
 12  PTS_TYPE                    128069 non-null  int64  
 13  SHOT_RESULT                 128069 non-null  object 
 14  CLOSEST_DEFENDER            128069 non-null  object 
 15  CLOSEST_DEFENDER_PLAYER_ID  128069 non-null  int64  
 16  CLOSE_DEF_DIST              128069 non-null  float64
 17  FGM                         128069 non-null  int64  
 18  PTS                         128069 non-null  int64  
 19  player_name                 128069 non-null  object 
 20  player_id                   128069 non-null  int64  
dtypes: float64(4), int64(10), object(7)
memory usage: 20.5+ MB
```

Out[113…

	GAME_ID	MATCHUP	LOCATION	W	FINAL_MARGIN	SHOT_NUMBER	PERIOD	GAME_CLOC
0	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	1	1	1:(
1	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	2	1	0:
2	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	3	1	0:(
3	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	4	2	11:
4	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	5	2	10:

5 rows × 21 columns

3.21 The 4 C's of Data Cleaning: Correcting, Completing, Creating, and Converting

In [114…

```
print('columns with null values:\n', df.isnull().sum())
```

```
columns with null values:
 GAME_ID                          0
MATCHUP                         0
LOCATION                        0
W                               0
FINAL_MARGIN                    0
SHOT_NUMBER                     0
PERIOD                          0
GAME_CLOCK                      0
SHOT_CLOCK                   5567
DRIBBLES                        0
TOUCH_TIME                      0
SHOT_DIST                       0
PTS_TYPE                        0
SHOT_RESULT                     0
CLOSEST_DEFENDER                0
CLOSEST_DEFENDER_PLAYER_ID      0
CLOSE_DEF_DIST                  0
FGM                             0
PTS                             0
player_name                     0
player_id                       0
dtype: int64
```

In [115…

```
df.describe(include = 'all')
```

Out[115...

	GAME_ID	MATCHUP	LOCATION	W	FINAL_MARGIN	SHOT_NUMBER	
count	1.280690e+05	128069	128069	128069	128069.000000	128069.000000	128069
unique	NaN	1808	2	2	NaN	NaN	
top	NaN	FEB 07, 2015 - DAL vs. POR	A	W	NaN	NaN	
freq	NaN	105	64135	64595	NaN	NaN	
mean	2.140045e+07	NaN	NaN	NaN	0.208723	6.506899	2
std	2.578773e+02	NaN	NaN	NaN	13.233267	4.713260	'
min	2.140000e+07	NaN	NaN	NaN	-53.000000	1.000000	1
25%	2.140023e+07	NaN	NaN	NaN	-8.000000	3.000000	1
50%	2.140045e+07	NaN	NaN	NaN	1.000000	5.000000	2
75%	2.140067e+07	NaN	NaN	NaN	9.000000	9.000000	3
max	2.140091e+07	NaN	NaN	NaN	53.000000	38.000000	7

11 rows × 21 columns

3.22 Clean Data

In [116...

```
df['SHOT_CLOCK'].fillna(0, inplace = True)
```

In [117...

```
df['TOUCH_TIME'] = df['TOUCH_TIME'].clip(lower=0)
```

In [118...

```
import datetime
df['TIME_ELAPSED_SECS'] = pd.to_datetime(df['GAME_CLOCK'], format='%M:%S')
df['TIME_ELAPSED_SECS'] = df['TIME_ELAPSED_SECS'].dt.hour * 3600 + df['TIME_E
df['TIME_ELAPSED_SECONDS'] = (720 * df['PERIOD']) - df['TIME_ELAPSED_SECS']
```

In [119...

```
df['LOCATION'] = df['LOCATION'].astype('category')
df['LOCATION'] = df['LOCATION'].cat.codes
```

In [120...

```
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = df[col].astype('category')
```

In [121...

```
drop_column = ['GAME_ID','MATCHUP', 'W', 'FINAL_MARGIN', 'SHOT_NUMBER','CLOSE
df = df.drop(drop_column, axis=1)
print(df.isnull().sum())
```

```
LOCATION                0
SHOT_CLOCK              0
DRIBBLES                0
TOUCH_TIME              0
SHOT_DIST               0
PTS_TYPE                0
CLOSE_DEF_DIST          0
FGM                     0
TIME_ELAPSED_SECONDS    0
dtype: int64
```

In [122…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128069 entries, 0 to 128068
Data columns (total 9 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   LOCATION              128069 non-null  int8   
 1   SHOT_CLOCK            128069 non-null  float64
 2   DRIBBLES              128069 non-null  int64  
 3   TOUCH_TIME            128069 non-null  float64
 4   SHOT_DIST             128069 non-null  float64
 5   PTS_TYPE              128069 non-null  int64  
 6   CLOSE_DEF_DIST        128069 non-null  float64
 7   FGM                   128069 non-null  int64  
 8   TIME_ELAPSED_SECONDS  128069 non-null  int64  
dtypes: float64(4), int64(4), int8(1)
memory usage: 7.9 MB
```

3.23 Convert Formats

In [123…

```
df.loc[df['TOUCH_TIME']>1]
```

Out[123…

	LOCATION	SHOT_CLOCK	DRIBBLES	TOUCH_TIME	SHOT_DIST	PTS_TYPE	CLOSE_D
0	0	10.8	2	1.9	7.7	2	
2	0	0.0	3	2.7	10.1	2	
3	0	10.3	2	1.9	17.2	2	
4	0	10.9	2	2.7	3.7	2	
5	0	9.1	2	4.4	18.4	2	
...	...	...	...	...	...	...	
128064	0	18.3	5	6.2	8.7	2	
128065	0	19.8	4	5.2	0.6	2	
128066	0	23.0	2	4.2	16.9	2	
128067	0	9.1	4	4.5	18.3	2	
128068	0	0.0	5	4.7	5.1	2	

79833 rows × 9 columns

In [124…

```
df[['IsCatchAndShot','IsLayupOrDunk','Is3point','IsOpen']] = 1
df["IsCatchAndShot"].loc[df["TOUCH_TIME"]>1] = 0
df['IsLayupOrDunk'].loc[df['SHOT_DIST']>3] = 0
df['Is3point'].loc[df['PTS_TYPE']==2] = 0
df['IsOpen'].loc[df['CLOSE_DEF_DIST']<3] = 0
df.head()
```

Out[124…

	LOCATION	SHOT_CLOCK	DRIBBLES	TOUCH_TIME	SHOT_DIST	PTS_TYPE	CLOSE_DEF_DIS
0	0	10.8	2	1.9	7.7	2	
1	0	3.4	0	0.8	28.2	3	
2	0	0.0	3	2.7	10.1	2	
3	0	10.3	2	1.9	17.2	2	
4	0	10.9	2	2.7	3.7	2	

3.24 Da-Double Check Cleaned Data

In [125…

```
print('columns with null values: \n', df.isnull().sum())
print("-"*10)
print (df.info())
print("-"*10)

df.describe(include = 'all')
```

```
columns with null values:
 LOCATION               0
SHOT_CLOCK             0
DRIBBLES               0
TOUCH_TIME             0
SHOT_DIST              0
PTS_TYPE               0
CLOSE_DEF_DIST         0
FGM                    0
TIME_ELAPSED_SECONDS    0
IsCatchAndShot         0
IsLayupOrDunk          0
Is3point               0
IsOpen                 0
dtype: int64
----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128069 entries, 0 to 128068
Data columns (total 13 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   LOCATION              128069 non-null  int8
 1   SHOT_CLOCK            128069 non-null  float64
 2   DRIBBLES              128069 non-null  int64
 3   TOUCH_TIME            128069 non-null  float64
 4   SHOT_DIST             128069 non-null  float64
 5   PTS_TYPE              128069 non-null  int64
 6   CLOSE_DEF_DIST        128069 non-null  float64
 7   FGM                   128069 non-null  int64
 8   TIME_ELAPSED_SECONDS  128069 non-null  int64
 9   IsCatchAndShot        128069 non-null  int64
 10  IsLayupOrDunk         128069 non-null  int64
 11  Is3point              128069 non-null  int64
 12  IsOpen                128069 non-null  int64
dtypes: float64(4), int64(8), int8(1)
memory usage: 11.8 MB
None
----------
```

Out[125…

	LOCATION	SHOT_CLOCK	DRIBBLES	TOUCH_TIME	SHOT_DIST	PT
count	128069.000000	128069.000000	128069.000000	128069.000000	128069.000000	128069.
mean	0.499215	11.912012	2.023355	2.771957	13.571504	2.
std	0.500001	6.182215	3.477760	2.986698	8.888964	C
min	0.000000	0.000000	0.000000	0.000000	0.000000	2.
25%	0.000000	7.500000	0.000000	0.900000	4.700000	2.
50%	0.000000	12.000000	1.000000	1.600000	13.700000	2.
75%	1.000000	16.400000	2.000000	3.700000	22.500000	3.
max	1.000000	24.000000	32.000000	24.900000	47.200000	3.

3.25 Split Training and Testing Data

In [126…

```
drop_column2 = ['SHOT_CLOCK', 'DRIBBLES', 'TOUCH_TIME', 'SHOT_DIST', 'PTS_TYP
X = df.drop(columns='FGM',axis =1)
y = df['FGM']

X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42)
print("DataFrame Shape: {}".format(df.shape))
print("Train Shape: {}".format(X_train.shape))
print("Test Shape: {}".format(X_test.shape))
```

```
DataFrame Shape: (128069, 13)
Train Shape: (96051, 12)
Test Shape: (32018, 12)
```

Step 4: Perform Exploratory Analysis with Statistics

In [127…

```
for x in X:
    print('shot Correlation by:', x)
    print(df[[x, 'FGM']].groupby(x, as_index=False).mean())
    print('-'*10, '\n')
```

```
shot Correlation by: LOCATION
   LOCATION       FGM
0         0  0.448117
1         1  0.456174
----------

shot Correlation by: SHOT_CLOCK
     SHOT_CLOCK       FGM
0           0.0  0.361382
1           0.1  0.313433
2           0.2  0.187500
3           0.3  0.183099
4           0.4  0.227273
..          ...       ...
236        23.6  0.503401
237        23.7  0.468750
238        23.8  0.410714
239        23.9  0.351648
240        24.0  0.600467

[241 rows x 2 columns]
----------

shot Correlation by: DRIBBLES
    DRIBBLES       FGM
0          0  0.471809
1          1  0.454068
2          2  0.424459
3          3  0.425802
```

```
4          4  0.431429
5          5  0.429153
6          6  0.414703
7          7  0.405113
8          8  0.426752
9          9  0.435463
10        10  0.443838
11        11  0.431862
12        12  0.410651
13        13  0.429752
14        14  0.426559
15        15  0.434316
16        16  0.383178
17        17  0.386179
18        18  0.427230
19        19  0.380117
20        20  0.371429
21        21  0.369863
22        22  0.385417
23        23  0.400000
24        24  0.315789
25        25  0.418605
26        26  0.458333
27        27  0.428571
28        28  0.076923
29        29  0.250000
30        30  0.333333
31        31  0.000000
32        32  0.000000
----------

shot Correlation by: TOUCH_TIME
     TOUCH_TIME       FGM
0           0.0  0.461584
1           0.1  0.603624
2           0.2  0.631638
3           0.3  0.649645
4           0.4  0.633920
..          ...       ...
236        24.0  0.000000
237        24.1  0.000000
238        24.4  0.000000
239        24.5  1.000000
240        24.9  1.000000

[241 rows x 2 columns]
----------

shot Correlation by: SHOT_DIST
     SHOT_DIST       FGM
0          0.0  0.500000
1          0.1  0.559322
2          0.2  0.640000
3          0.3  0.676829
4          0.4  0.645631
..         ...       ...
443       46.2  0.000000
```

```
444       46.3  0.000000
445       46.7  0.000000
446       46.9  0.000000
447       47.2  1.000000

[448 rows x 2 columns]
----------

shot Correlation by: PTS_TYPE
   PTS_TYPE       FGM
0         2  0.488357
1         3  0.351516
----------

shot Correlation by: CLOSE_DEF_DIST
     CLOSE_DEF_DIST       FGM
0               0.0  0.431818
1               0.1  0.439863
2               0.2  0.441296
3               0.3  0.495327
4               0.4  0.457327
..              ...       ...
294            49.5  1.000000
295            52.0  1.000000
296            52.6  1.000000
297            52.9  1.000000
298            53.2  1.000000

[299 rows x 2 columns]
----------

shot Correlation by: TIME_ELAPSED_SECONDS
      TIME_ELAPSED_SECONDS  FGM
0                        3  0.0
1                        4  1.0
2                        5  1.0
3                        6  0.0
4                        7  0.0
...                    ...  ...
3310                  5018  0.0
3311                  5023  0.0
3312                  5030  0.0
3313                  5035  0.0
3314                  5037  0.0

[3315 rows x 2 columns]
----------

shot Correlation by: IsCatchAndShot
   IsCatchAndShot       FGM
0               0  0.434069
1               1  0.482047
----------

shot Correlation by: IsLayupOrDunk
   IsLayupOrDunk       FGM
0              0  0.421029
```

```
1               1  0.643192
----------

shot Correlation by: Is3point
   Is3point       FGM
0          0  0.488357
1          1  0.351516
----------

shot Correlation by: IsOpen
   IsOpen       FGM
0        0  0.461673
1        1  0.446417
----------
```

In [128…

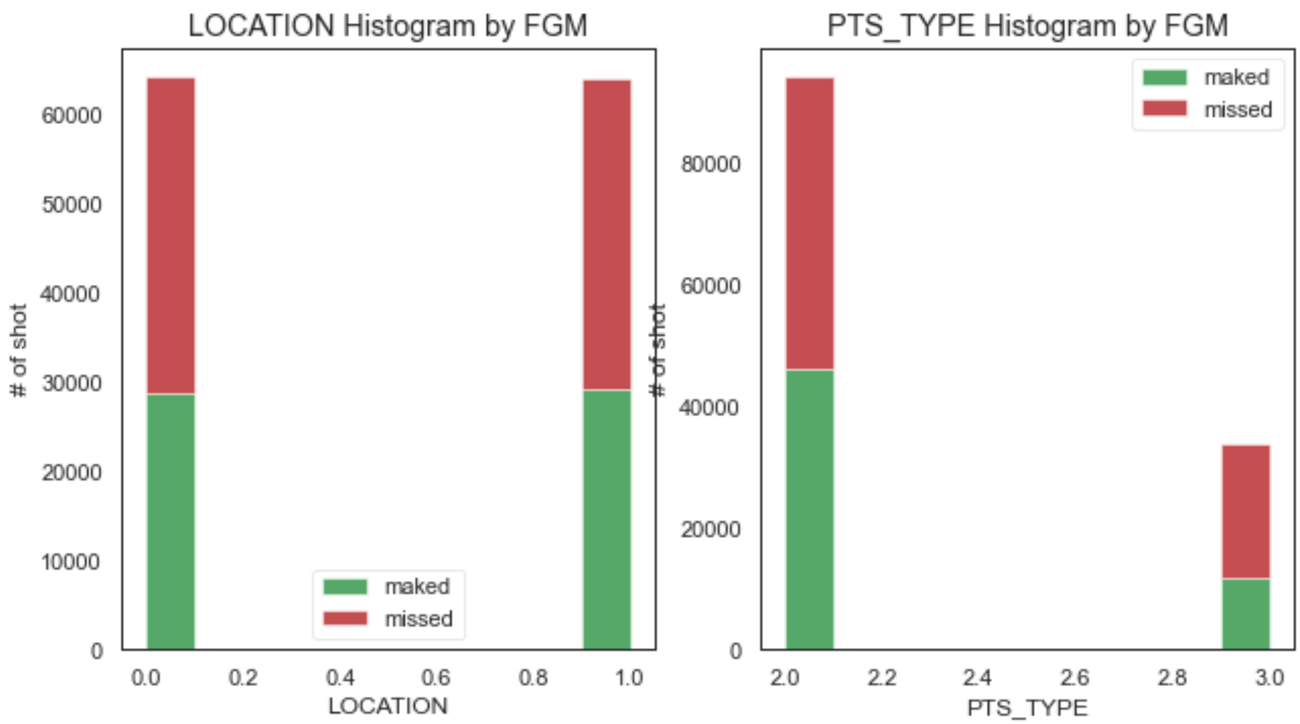
```
plt.figure(figsize=[16,12])

plt.subplot(234)
plt.hist(x = [df[df['FGM']==1]['LOCATION'], df[df['FGM']==0]['LOCATION']],
         stacked=True, color = ['g','r'],label = ['maked','missed'])
plt.title('LOCATION Histogram by FGM')
plt.xlabel('LOCATION')
plt.ylabel('# of shot')
plt.legend()


plt.subplot(235)
plt.hist(x = [df[df['FGM']==1]['PTS_TYPE'], df[df['FGM']==0]['PTS_TYPE']],
         stacked=True, color = ['g','r'],label = ['maked','missed'])
plt.title('PTS_TYPE Histogram by FGM')
plt.xlabel('PTS_TYPE')
plt.ylabel('# of shot')
plt.legend()
```

Out[128…

```
<matplotlib.legend.Legend at 0x7f9301a275b0>
```



In [129…

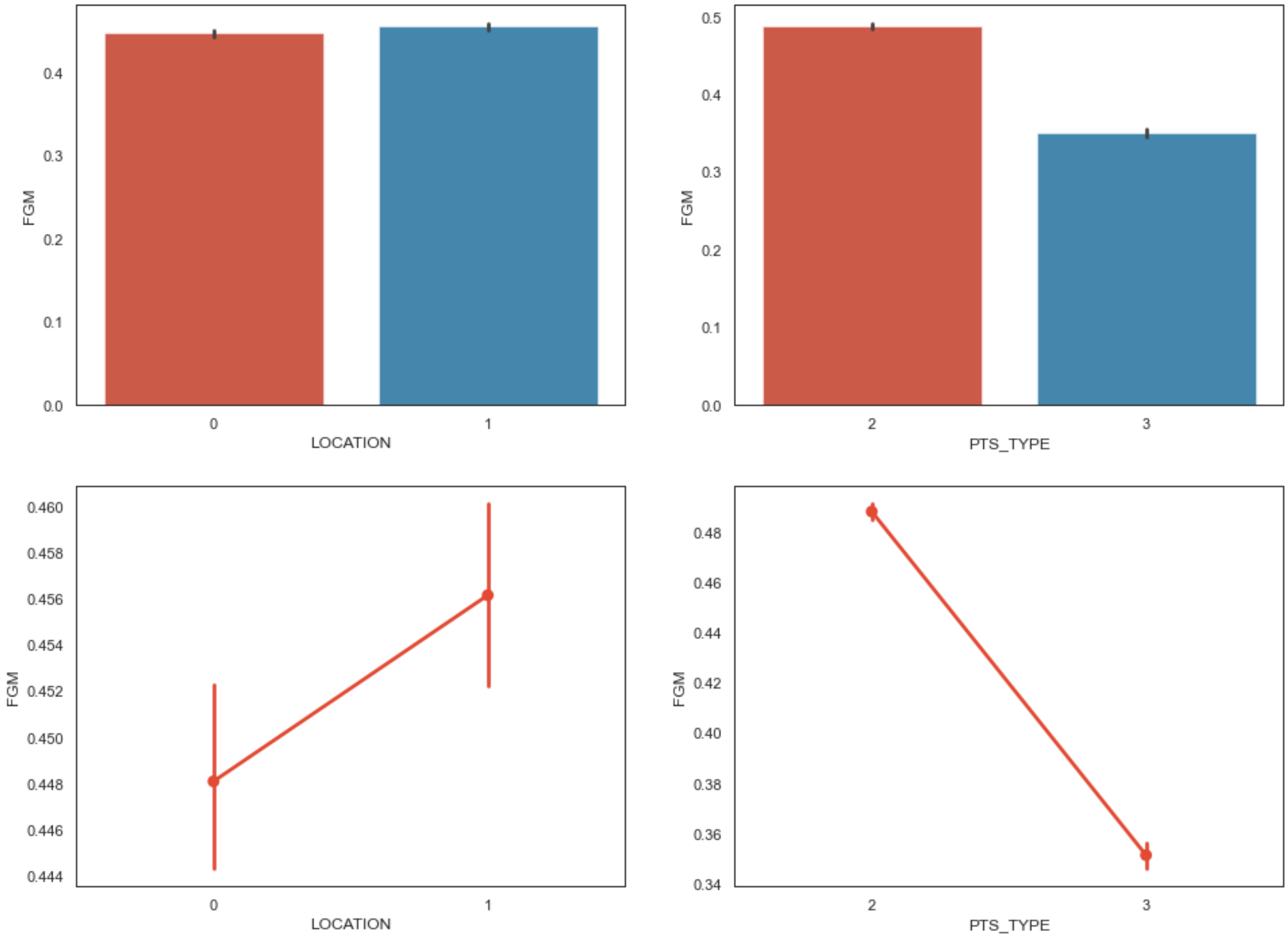
```
fig, saxis = plt.subplots(2, 2,figsize=(16,12))

sns.barplot(x = 'LOCATION', y = 'FGM', data=df, ax = saxis[0,0])
sns.barplot(x = 'PTS_TYPE', y = 'FGM', data=df, ax = saxis[0,1])

sns.pointplot(x = 'LOCATION', y = 'FGM',  data=df, ax = saxis[1,0])
sns.pointplot(x = 'PTS_TYPE', y = 'FGM', data=df, ax = saxis[1,1])
```

Out[129…

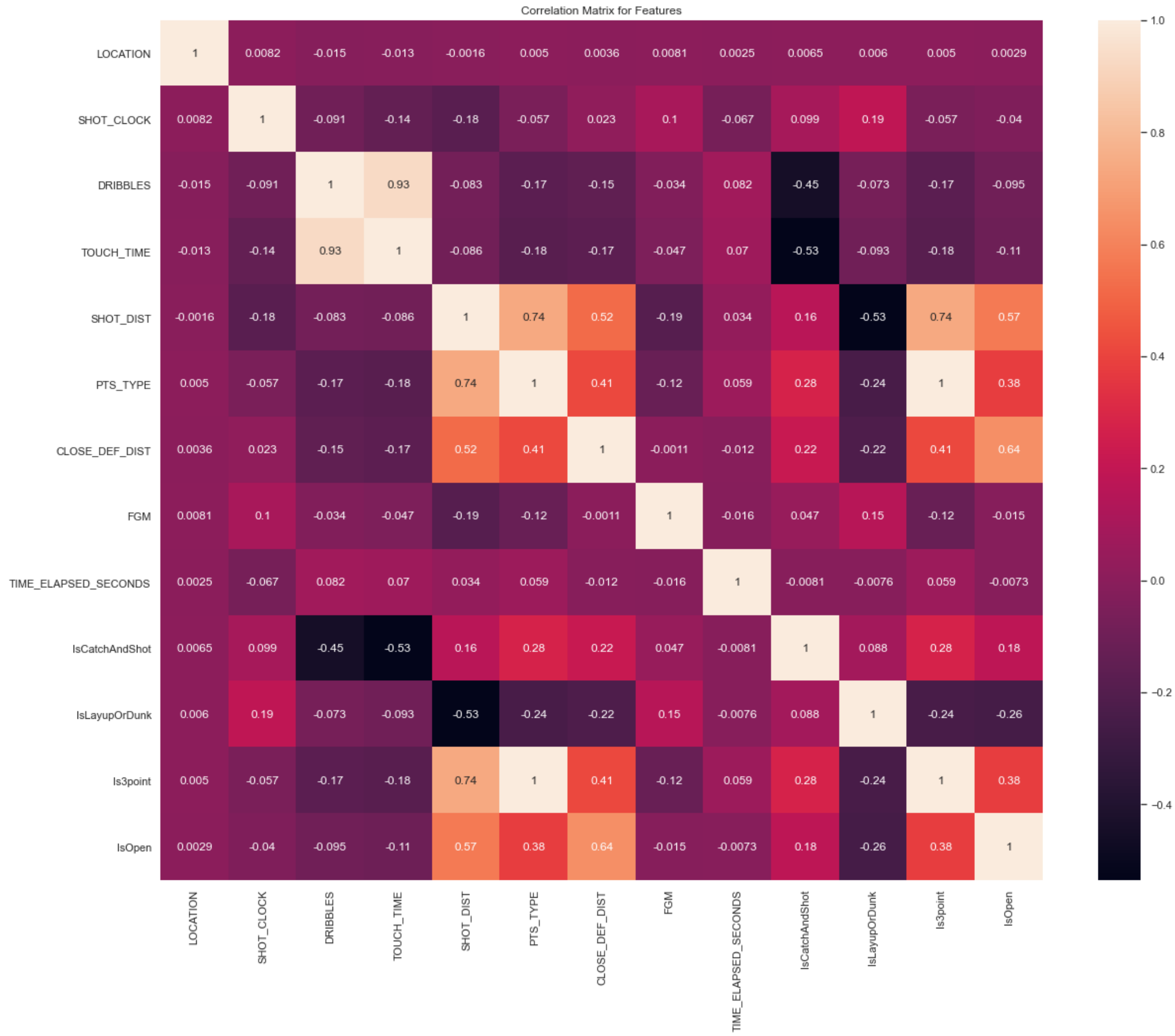
```
<AxesSubplot:xlabel='PTS_TYPE', ylabel='FGM'>
```



In [159…

```
plt.figure(figsize = (20, 16))
sns.set(style="white")
sns.heatmap(df.corr(), annot = True)


plt.title('Correlation Matrix for Features')
plt.show()
```



Step 5: Model Data

Stacking

In [131…

```
from sklearn.feature_selection import VarianceThreshold

selection = VarianceThreshold(threshold=(0.1))
X = selection.fit_transform(X)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, test_size=0.2, random_state=42
)
```

K nearest neighbors

In [132…

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(3) # Define classifier
knn.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)

# Training set performance
knn_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accura
knn_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
knn_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculat

# Test set performance
knn_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
knn_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
knn_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F

print('Model performance for Training set')
print('- Accuracy: %s' % knn_train_accuracy)
print('- MCC: %s' % knn_train_mcc)
print('- F1 score: %s' % knn_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % knn_test_accuracy)
print('- MCC: %s' % knn_test_mcc)
print('- F1 score: %s' % knn_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.7690693475184227
- MCC: 0.532316405778505
- F1 score: 0.768386546917778
----------------------------------
Model performance for Test set
- Accuracy: 0.5484891075193253
- MCC: 0.08389749917798749
- F1 score: 0.5469391930269014
```

Decision tree

In [133…

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=5) # Define classifier
dt.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

# Training set performance
dt_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accurac
dt_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
dt_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate

# Test set performance
dt_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
dt_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
dt_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1

print('Model performance for Training set')
print('- Accuracy: %s' % dt_train_accuracy)
print('- MCC: %s' % dt_train_mcc)
print('- F1 score: %s' % dt_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % dt_test_accuracy)
print('- MCC: %s' % dt_test_mcc)
print('- F1 score: %s' % dt_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.6197745351617784
- MCC: 0.22533235955841302
- F1 score: 0.587626073367328
----------------------------------
Model performance for Test set
- Accuracy: 0.6153275552432264
- MCC: 0.21516725289276312
- F1 score: 0.5814663776353239
```

Random forest

In [162…
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=10) # Define classifier
rf.fit(X_train, y_train) # Train model

# Make predictions
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)

# Training set performance
rf_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accurac
rf_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
rf_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculate

# Test set performance
rf_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
rf_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
rf_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F1

print('Model performance for Training set')
print('- Accuracy: %s' % rf_train_accuracy)
print('- MCC: %s' % rf_train_mcc)
print('- F1 score: %s' % rf_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % rf_test_accuracy)
print('- MCC: %s' % rf_test_mcc)
print('- F1 score: %s' % rf_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.9797569664730857
- MCC: 0.9594924615656203
- F1 score: 0.9797214808932537
----------------------------------
Model performance for Test set
- Accuracy: 0.5854220348247052
- MCC: 0.146339338798333
- F1 score: 0.57120903740748
```

In [163…
```
print(metrics.classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.60      0.75      0.66     14033
           1       0.56      0.39      0.46     11581

    accuracy                           0.59     25614
   macro avg       0.58      0.57      0.56     25614
weighted avg       0.58      0.59      0.57     25614
```

Neural network

In [135…

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(alpha=1, max_iter=1000)
mlp.fit(X_train, y_train)

# Make predictions
y_train_pred = mlp.predict(X_train)
y_test_pred = mlp.predict(X_test)

# Training set performance
mlp_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accura
mlp_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
mlp_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculat

# Test set performance
mlp_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
mlp_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
mlp_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F

print('Model performance for Training set')
print('- Accuracy: %s' % mlp_train_accuracy)
print('- MCC: %s' % mlp_train_mcc)
print('- F1 score: %s' % mlp_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % mlp_test_accuracy)
print('- MCC: %s' % mlp_test_mcc)
print('- F1 score: %s' % mlp_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.5763603533258503
- MCC: 0.1372987482920126
- F1 score: 0.4775888901914582
----------------------------------
Model performance for Test set
- Accuracy: 0.5804247677051613
- MCC: 0.15291348042835778
- F1 score: 0.48271408154893586
```

XGBClassifier

In [164…

```
from xgboost import XGBClassifier

XGB = XGBClassifier()
XGB.fit(X_train, y_train)

# Make predictions
y_train_pred = XGB.predict(X_train)
y_test_pred = XGB.predict(X_test)

# Training set performance
XGB_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accura
XGB_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
XGB_train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculat

# Test set performance
XGB_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
XGB_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
XGB_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F

print('Model performance for Training set')
print('- Accuracy: %s' % XGB_train_accuracy)
print('- MCC: %s' % XGB_train_mcc)
print('- F1 score: %s' % XGB_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % XGB_test_accuracy)
print('- MCC: %s' % XGB_test_mcc)
print('- F1 score: %s' % XGB_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.6641257137279781
- MCC: 0.3211206776470334
- F1 score: 0.6462222023438322
----------------------------------
Model performance for Test set
- Accuracy: 0.6113843991567112
- MCC: 0.20239205373303035
- F1 score: 0.5886018007918407
```

In [165…

```
print(metrics.classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.61      0.82      0.70     14033
           1       0.62      0.36      0.46     11581

    accuracy                           0.61     25614
   macro avg       0.61      0.59      0.58     25614
weighted avg       0.61      0.61      0.59     25614
```

Build Stacked model

In [137…

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

estimator_list = [
    ('knn',knn),
    ('dt',dt),
    ('rf',rf),
    ('mlp',mlp)]

# Build stack model
stack_model = StackingClassifier(
    estimators=estimator_list, final_estimator=LogisticRegression()
)

# Train stacked model
stack_model.fit(X_train, y_train)

# Make predictions
y_train_pred = stack_model.predict(X_train)
y_test_pred = stack_model.predict(X_test)

# Training set model performance
stack_model_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculat
stack_model_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate
stack_model_train_f1 = f1_score(y_train, y_train_pred, average='weighted') #

# Test set model performance
stack_model_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate A
stack_model_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
stack_model_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Cal

print('Model performance for Training set')
print('- Accuracy: %s' % stack_model_train_accuracy)
print('- MCC: %s' % stack_model_train_mcc)
print('- F1 score: %s' % stack_model_train_f1)
print('----------------------------------')
print('Model performance for Test set')
print('- Accuracy: %s' % stack_model_test_accuracy)
print('- MCC: %s' % stack_model_test_mcc)
print('- F1 score: %s' % stack_model_test_f1)
```

```
Model performance for Training set
- Accuracy: 0.649143526426236
- MCC: 0.2926133581198161
- F1 score: 0.6234565559970068
----------------------------------
Model performance for Test set
- Accuracy: 0.6152885140938549
- MCC: 0.21404350832940483
- F1 score: 0.5837509723153406
```

In [138…

```
acc_train_list = {'knn':knn_train_accuracy,
'dt': dt_train_accuracy,
'rf': rf_train_accuracy,
'mlp': mlp_train_accuracy,
'stack': stack_model_train_accuracy,
'XGBoost': XGB_train_accuracy}

mcc_train_list = {'knn':knn_train_mcc,
'dt': dt_train_mcc,
'rf': rf_train_mcc,
'mlp': mlp_train_mcc,
'stack': stack_model_train_mcc,
'XGBoost': XGB_train_mcc}

f1_train_list = {'knn':knn_train_f1,
'dt': dt_train_f1,
'rf': rf_train_f1,
'mlp': mlp_train_f1,
'stack': stack_model_train_f1,
'XGBoost': XGB_train_f1}
```

In [139…

```
acc_df = pd.DataFrame.from_dict(acc_train_list, orient='index', columns=['Acc
mcc_df = pd.DataFrame.from_dict(mcc_train_list, orient='index', columns=['MCC
f1_df = pd.DataFrame.from_dict(f1_train_list, orient='index', columns=['F1'])
result_df = pd.concat([acc_df, mcc_df, f1_df], axis=1)
result_df.sort_values(by = ['Accuracy'], ascending = False, inplace = True)
result_df
```

Out[139…

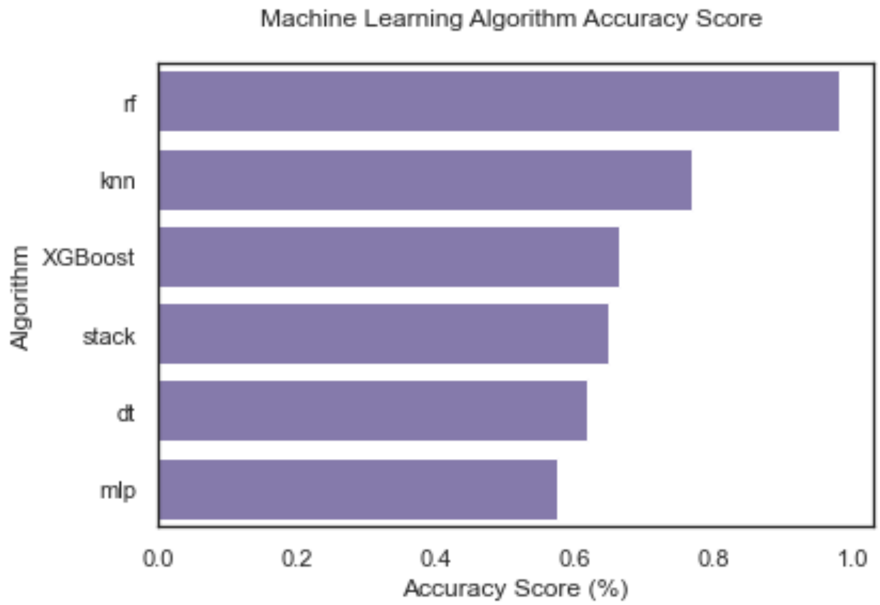
	Accuracy	MCC	F1
rf	0.980528	0.960993	0.980497
knn	0.769069	0.532316	0.768387
XGBoost	0.664126	0.321121	0.646222
stack	0.649144	0.292613	0.623457
dt	0.619775	0.225332	0.587626
mlp	0.576360	0.137299	0.477589

In [140…

```
sns.barplot(x=result_df['Accuracy'],y=result_df.index, data = df, color = 'm'

plt.title('Machine Learning Algorithm Accuracy Score \n')
plt.xlabel('Accuracy Score (%)')
plt.ylabel('Algorithm')
```

Out[140… Text(0, 0.5, 'Algorithm')



In [141…

```
MLA = [
    #Ensemble Methods
    #ensemble.AdaBoostClassifier(),
    ensemble.BaggingClassifier(),
    #ensemble.ExtraTreesClassifier(),
    #ensemble.GradientBoostingClassifier(),
    ensemble.RandomForestClassifier(),

    #Gaussian Processes
    #gaussian_process.GaussianProcessClassifier(),

    #GLM
    #linear_model.LogisticRegressionCV(),
    linear_model.PassiveAggressiveClassifier(),
    linear_model.RidgeClassifierCV(),
    linear_model.SGDClassifier(),
    linear_model.Perceptron(),

    #Navies Bayes
    naive_bayes.BernoulliNB(),
    naive_bayes.GaussianNB(),

    #Nearest Neighbor
    neighbors.KNeighborsClassifier(),

    #SVM
    #svm.SVC(probability=True),
    #svm.NuSVC(probability=True),
    svm.LinearSVC(),

    #Trees
    tree.DecisionTreeClassifier(),
```

```
    tree.ExtraTreeClassifier(),

    #Discriminant Analysis
    discriminant_analysis.LinearDiscriminantAnalysis(),
    discriminant_analysis.QuadraticDiscriminantAnalysis(),


    #xgboost: http://xgboost.readthedocs.io/en/latest/model.html
    XGBClassifier()
    ]
MLA_columns = ['MLA Name', 'MLA Parameters','MLA Train Accuracy', 'MLA Test A
MLA_compare = pd.DataFrame(columns = MLA_columns)

row_index = 0
for alg in MLA:
    alg.fit(X_train, y_train)
    y_train_pred = alg.predict(X_train)
    y_test_pred = alg.predict(X_test)

    MLA_name = alg.__class__.__name__
    # Training set performance
    train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate Accura
    train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
    train_f1 = f1_score(y_train, y_train_pred, average='weighted') # Calculat

    # Test set performance
    test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
    test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
    test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate F

    MLA_compare.loc[row_index, 'MLA Name'] = MLA_name
    MLA_compare.loc[row_index, 'MLA Parameters'] = str(alg.get_params())
    MLA_compare.loc[row_index, 'MLA Train Accuracy'] = train_accuracy
    MLA_compare.loc[row_index, 'MLA Test Accuracy'] = test_accuracy
    MLA_compare.loc[row_index, 'MLA Train mcc'] = train_mcc
    MLA_compare.loc[row_index, 'MLA Test mcc'] = test_mcc
    MLA_compare.loc[row_index, 'MLA Train f1'] = train_f1
    MLA_compare.loc[row_index, 'MLA Test f1'] = test_f1
    row_index+=1

MLA_compare.sort_values(by = ['MLA Test Accuracy'], ascending = False, inplac
MLA_compare
```

Out[141…

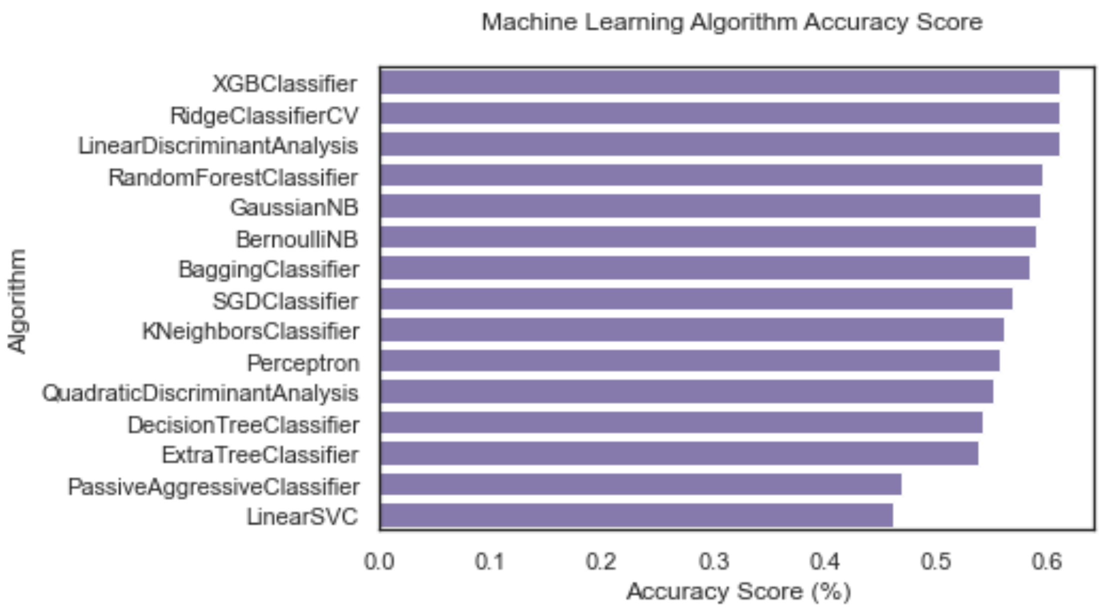
	MLA Name	MLA Parameters	MLA Train Accuracy	MLA Test Accuracy	MLA Train mcc	MLA Test mcc
14	XGBClassifier	{'objective': 'binary:logistic', 'base_score':...	0.664126	0.611384	0.321121	0.202392
3	RidgeClassifierCV	{'alphas': array([0.1, 1. , 10.]), 'class_w...	0.613586	0.61115	0.207103	0.201646

12	LinearDiscriminantAnalysis	{'n_components': None, 'priors': None, 'shrink...	0.613538	0.610838	0.207008	0.200972
1	RandomForestClassifier	{'bootstrap': True, 'ccp_alpha': 0.0, 'class_w...	0.99999	0.595846	0.99998	0.169125
7	GaussianNB	{'priors': None, 'var_smoothing': 1e-09}	0.595705	0.594401	0.176866	0.172908
6	BernoulliNB	{'alpha': 1.0, 'binarize': 0.0, 'class_prior':...	0.592504	0.590185	0.159068	0.153589
0	BaggingClassifier	{'base_estimator': None, 'bootstrap': True, 'b...	0.980177	0.583665	0.960321	0.142838
4	SGDClassifier	{'alpha': 0.0001, 'average': False, 'class_wei...	0.569245	0.569298	0.157157	0.15577
8	KNeighborsClassifier	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	0.713045	0.561099	0.417609	0.107308
5	Perceptron	{'alpha': 0.0001, 'class_weight': None, 'early...	0.555229	0.556688	0.143554	0.146932
13	QuadraticDiscriminantAnalysis	{'priors': None, 'reg_param': 0.0, 'store_cova...	0.549822	0.552081	0.0315564	0.0415203
10	DecisionTreeClassifier	{'ccp_alpha': 0.0, 'class_weight': None, 'crit...	0.99999	0.541696	0.99998	0.0753268
11	ExtraTreeClassifier	{'ccp_alpha': 0.0, 'class_weight': None, 'crit...	0.99999	0.538026	0.99998	0.0670502
2	PassiveAggressiveClassifier	{'C': 1.0, 'average': False, 'class_weight': N...	0.469953	0.468884	0.0140295	0.0108873
9	LinearSVC	{'C': 1.0, 'class_weight': None, 'dual': True,...	0.45945	0.460725	0.0246903	0.030818

In [142…
```
sns.barplot(x='MLA Test Accuracy', y = 'MLA Name', data = MLA_compare, color

#prettify using pyplot: https://matplotlib.org/api/pyplot_api.html
plt.title('Machine Learning Algorithm Accuracy Score \n')
plt.xlabel('Accuracy Score (%)')
plt.ylabel('Algorithm')
```

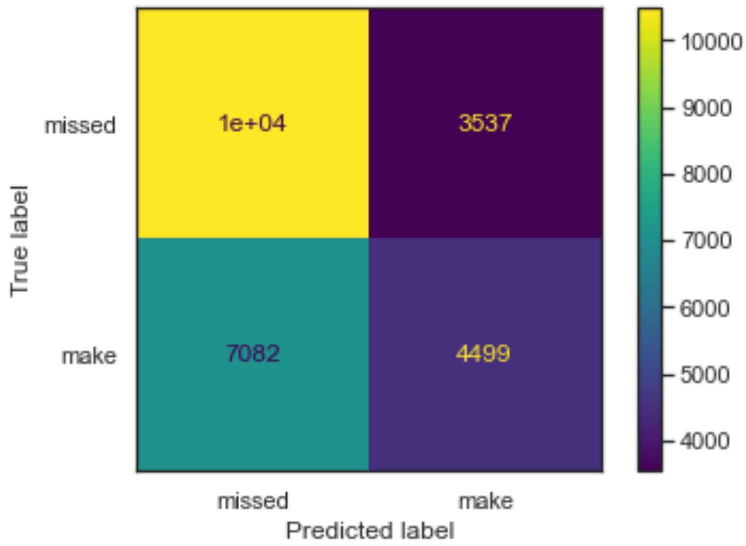
Out[142…
```
Text(0, 0.5, 'Algorithm')
```



5.1 Evaluate Model Performance

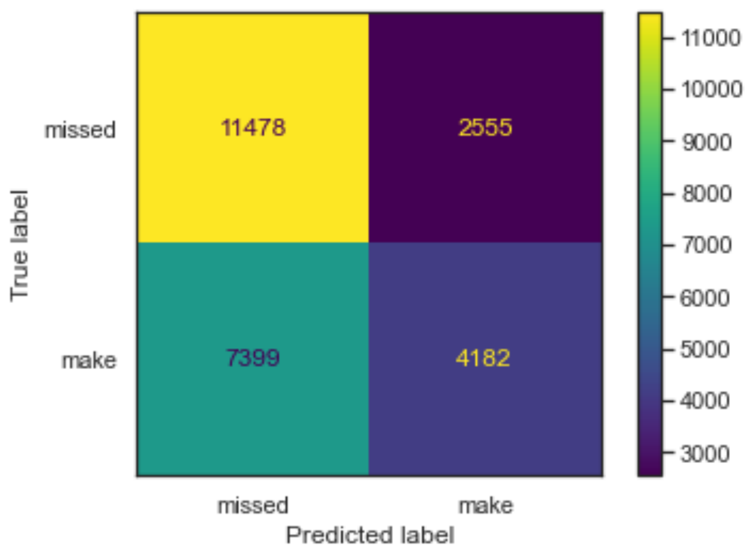
In [222…
```
label =['missed',"make"]
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf, X_test, y_test, display_labels=label)
```

Out[222... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f93196a1400>



In [223...
```
plot_confusion_matrix(XGB, X_test, y_test, display_labels=label)
```

Out[223... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f9301627e80>



5.11 Model Performance with Cross-Validation (CV)

In [145...
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(rf, X_train, y_train, cv=3)
print("Random Forest cross-validation score{:.2f}". format(cv_scores.mean()))
```

```
Random Forest cross-validation score 0.5848421304002649
```

In [146...
```
cv_scores = cross_val_score(XGB, X_train, y_train,cv=3)
print("XGBoost cross-validation score{:.2f}". format(cv_scores.mean()))
```

```
XGBoost cross-validation score 0.6125323548195283
```

5.12 Tune Model with Hyper-Parameters

In [168…

```
#base model
base_results = model_selection.cross_validate(rf, X_train, y_train, cv=3, ret

print('BEFORE RF Parameters: ', rf.get_params())
print("BEFORE RF Training w/bin score mean: {:.2f}". format(base_results['tra
print("BEFORE RF Test w/bin score mean: {:.2f}". format(base_results['test_sc
print("BEFORE RF Test w/bin score 3*std: +/- {:.2f}". format(base_results['te
#print("BEFORE DT Test w/bin set score min: {:.2f}". format(base_results['tes
print('-'*10)


#tune hyper-parameters: http://scikit-learn.org/stable/modules/generated/skle
param_grid = {'criterion': ['gini', 'entropy'],  #scoring methodology; two su
              #'splitter': ['best', 'random'], #splitting methodology; two su
              'max_depth': [2,4,6,8,10,None], #max depth tree can grow; defau
              #'min_samples_split': [2,5,10,.03,.05], #minimum subset size BE
              #'min_samples_leaf': [1,5,10,.03,.05], #minimum subset size AFT
              #'max_features': [None, 'auto'], #max features to consider when
              'random_state': [0] #seed or control random number generator: h
             }

#print(list(model_selection.ParameterGrid(param_grid)))

#choose best model with grid_search: #http://scikit-learn.org/stable/modules/
#http://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_searc
tune_model = model_selection.GridSearchCV(RandomForestClassifier(), param_gri
tune_model.fit(X_train, y_train)

#print(tune_model.cv_results_.keys())
#print(tune_model.cv_results_['params'])
print('AFTER RF Parameters: ', tune_model.best_params_)
#print(tune_model.cv_results_['mean_train_score'])
print("AFTER RF Training w/bin score mean: {:.2f}". format(tune_model.cv_resu
#print(tune_model.cv_results_['mean_test_score'])
print("AFTER RF Test w/bin score mean: {:.2f}". format(tune_model.cv_results_
print("AFTER RF Test w/bin score 3*std: +/- {:.2f}". format(tune_model.cv_res
print('-'*10)


#duplicates gridsearchcv
tune_results = model_selection.cross_validate(tune_model, X_train, y_train, c

print('AFTER DT Parameters: ', tune_model.best_params_)
print("AFTER DT Training w/bin set score mean: {:.2f}". format(tune_results['
print("AFTER DT Test w/bin set score mean: {:.2f}". format(tune_results['test
print("AFTER DT Test w/bin set score min: {:.2f}". format(tune_results['test_
print('-'*10)
```

```
BEFORE RF Parameters:  {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': N
one, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf
_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurit
y_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fra
ction_leaf': 0.0, 'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'ran
dom_state': None, 'verbose': 0, 'warm_start': False}
BEFORE RF Training w/bin score mean: 97.96
BEFORE RF Test w/bin score mean: 58.48
BEFORE RF Test w/bin score 3*std: +/- 0.74
----------
AFTER RF Parameters:  {'criterion': 'entropy', 'max_depth': 10, 'random_state'
: 0}
AFTER RF Training w/bin score mean: 69.28
AFTER RF Test w/bin score mean: 64.08
AFTER RF Test w/bin score 3*std: +/- 0.62
----------
AFTER DT Parameters:  {'criterion': 'entropy', 'max_depth': 10, 'random_state'
: 0}
AFTER DT Training w/bin set score mean: 67.22
AFTER DT Test w/bin set score mean: 64.07
AFTER DT Test w/bin set score min: 63.80
----------
```

In [166…

```
#base model
XGB_base_results = model_selection.cross_validate(XGB, X_train, y_train, cv=3

print('BEFORE XGBoost Parameters: ', XGB.get_params())
print("BEFORE XGBoost Training w/bin score mean: {:.2f}". format(XGB_base_res
print("BEFORE XGBoost Test w/bin score mean: {:.2f}". format(XGB_base_results
print("BEFORE XGBoost Test w/bin score 3*std: +/- {:.2f}". format(XGB_base_re
#print("BEFORE DT Test w/bin set score min: {:.2f}". format(base_results['tes
print('-'*10)


#tune hyper-parameters: http://scikit-learn.org/stable/modules/generated/skle
XGB_param_grid = {#'criterion': ['gini', 'entropy'],  #scoring methodology; t
              #'splitter': ['best', 'random'], #splitting methodology; two su
              'max_depth': [2,4,6,8,10,None], #max depth tree can grow; defau
              #'min_samples_split': [2,5,10,.03,.05], #minimum subset size BE
              #'min_samples_leaf': [1,5,10,.03,.05], #minimum subset size AFT
              #'max_features': [None, 'auto'], #max features to consider when
              'random_state': [0] #seed or control random number generator: h
             }

#print(list(model_selection.ParameterGrid(param_grid)))

#choose best model with grid_search: #http://scikit-learn.org/stable/modules/
#http://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_searc
tune_model = model_selection.GridSearchCV(XGBClassifier(), param_grid=XGB_par
tune_model.fit(X_train, y_train)

#print(tune_model.cv_results_.keys())
#print(tune_model.cv_results_['params'])
print('AFTER XGBoost Parameters: ', tune_model.best_params_)
#print(tune_model.cv_results_['mean_train_score'])
print("AFTER XGBoost Training w/bin score mean: {:.2f}". format(tune_model.cv
#print(tune_model.cv_results_['mean_test_score'])
print("AFTER XGBoost Test w/bin score mean: {:.2f}". format(tune_model.cv_res
print("AFTER XGBoost Test w/bin score 3*std: +/- {:.2f}". format(tune_model.c
print('-'*10)


#duplicates gridsearchcv
#tune_results = model_selection.cross_validate(tune_model, data1[data1_x_bin]

#print('AFTER XGBoost Parameters: ', tune_model.best_params_)
#print("AFTER XGBoost Training w/bin set score mean: {:.2f}". format(tune_res
#print("AFTER XGBoost Test w/bin set score mean: {:.2f}". format(tune_results
#print("AFTER XGBoost Test w/bin set score min: {:.2f}". format(tune_results[
#print('-'*10)
```

```
BEFORE XGBoost Parameters:  {'objective': 'binary:logistic', 'base_score': 0.5
, 'booster': 'gbtree', 'colsample_bylevel': 1, 'colsample_bynode': 1, 'colsamp
le_bytree': 1, 'gamma': 0, 'gpu_id': -1, 'importance_type': 'gain', 'interacti
on_constraints': '', 'learning_rate': 0.300000012, 'max_delta_step': 0, 'max_d
epth': 6, 'min_child_weight': 1, 'missing': nan, 'monotone_constraints': '()',
'n_estimators': 100, 'n_jobs': 0, 'num_parallel_tree': 1, 'random_state': 0, '
reg_alpha': 0, 'reg_lambda': 1, 'scale_pos_weight': 1, 'subsample': 1, 'tree_m
ethod': 'exact', 'validate_parameters': 1, 'verbosity': None}
BEFORE XGBoost Training w/bin score mean: 67.90
BEFORE XGBoost Test w/bin score mean: 61.25
BEFORE XGBoost Test w/bin score 3*std: +/- 0.57
----------
AFTER XGBoost Parameters:  {'max_depth': 2, 'random_state': 0}
AFTER XGBoost Training w/bin score mean: 65.19
AFTER XGBoost Test w/bin score mean: 64.07
AFTER XGBoost Test w/bin score 3*std: +/- 0.71
----------
```

5.13 Tune Model with Feature Selection

In [153…

```
x = df.drop(columns='FGM')
print('BEFORE RF RFE Training Shape Old: ', x.shape)
print('BEFORE RF RFE Training Columns Old: ', x.columns.values)
print("BEFORE RF RFE Training w/bin score mean: {:.2f}". format(base_results[
print("BEFORE RF RFE Test w/bin score mean: {:.2f}". format(base_results['tes
print("BEFORE RF RFE Test w/bin score 3*std: +/- {:.2f}". format(base_results
print('-'*10)



RF_rfe = feature_selection.RFECV(rf, step = 1, scoring = 'accuracy', cv = 3)
RF_rfe.fit(X_train, y_train)

#transform x&y to reduced features and fit new model
#alternative: can use pipeline to reduce fit and transform steps: http://scik
X_rfe = x.columns.values[RF_rfe.get_support()]
rfe_results = model_selection.cross_validate(rf, x[X_rfe], y, cv  = 3,return_

#print(rf_rfe.grid_scores_)
print('AFTER RF RFE Training Shape New: ', x[X_rfe].shape)
print('AFTER RF RFE Training Columns New: ', X_rfe)
print("AFTER RF RFE Training w/bin score mean: {:.2f}". format(rfe_results['t
print("AFTER RF RFE Test w/bin score mean: {:.2f}". format(rfe_results['test_
print("AFTER RF RFE Test w/bin score 3*std: +/- {:.2f}". format(rfe_results['
print('-'*10)


#tune rfe model
rfe_tune_model = model_selection.GridSearchCV(RandomForestClassifier(), param
rfe_tune_model.fit(x[X_rfe], y)

#print(rfe_tune_model.cv_results_.keys())
#print(rfe_tune_model.cv_results_['params'])
print('AFTER RF RFE Tuned Parameters: ', rfe_tune_model.best_params_)
#print(rfe_tune_model.cv_results_['mean_train_score'])
print("AFTER RF RFE Tuned Training w/bin score mean: {:.2f}". format(rfe_tune
#print(rfe_tune_model.cv_results_['mean_test_score'])
print("AFTER RF RFE Tuned Test w/bin score mean: {:.2f}". format(rfe_tune_mod
print("AFTER RF RFE Tuned Test w/bin score 3*std: +/- {:.2f}". format(rfe_tun
print('-'*10)
```

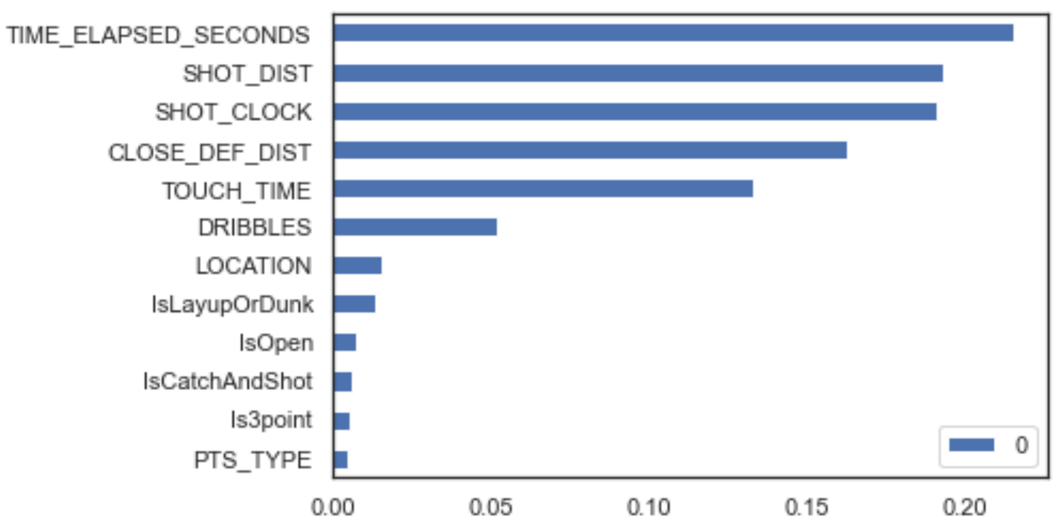
```
BEFORE RF RFE Training Shape Old:  (128069, 12)
BEFORE RF RFE Training Columns Old:  ['LOCATION' 'SHOT_CLOCK' 'DRIBBLES' 'TOUC
H_TIME' 'SHOT_DIST' 'PTS_TYPE'
 'CLOSE_DEF_DIST' 'TIME_ELAPSED_SECONDS' 'IsCatchAndShot' 'IsLayupOrDunk'
 'Is3point' 'IsOpen']
BEFORE RF RFE Training w/bin score mean: 98.02
BEFORE RF RFE Test w/bin score mean: 58.34
BEFORE RF RFE Test w/bin score 3*std: +/- 0.67
----------
AFTER RF RFE Training Shape New:  (128069, 12)
AFTER RF RFE Training Columns New:  ['LOCATION' 'SHOT_CLOCK' 'DRIBBLES' 'TOUCH
_TIME' 'SHOT_DIST' 'PTS_TYPE'
 'CLOSE_DEF_DIST' 'TIME_ELAPSED_SECONDS' 'IsCatchAndShot' 'IsLayupOrDunk'
 'Is3point' 'IsOpen']
AFTER RF RFE Training w/bin score mean: 98.03
AFTER RF RFE Test w/bin score mean: 58.16
AFTER RF RFE Test w/bin score 3*std: +/- 0.65
----------
AFTER RF RFE Tuned Parameters:  {'criterion': 'entropy', 'max_depth': 10, 'ran
dom_state': 0}
AFTER RF RFE Tuned Training w/bin score mean: 62.55
AFTER RF RFE Tuned Test w/bin score mean: 62.44
AFTER RF RFE Tuned Test w/bin score 3*std: +/- 0.95
----------
```

In [216…

```python
sort_rf_feature_importances = pd.DataFrame(rf.feature_importances_, x.columns
sort_rf_feature_importances.sort_values(0,ascending = True, axis=0,inplace=Tr
sort_rf_feature_importances.plot(kind='barh')
```

Out[216…

```
<AxesSubplot:>
```



In [167…

```
x = df.drop(columns='FGM')
print('BEFORE XGBoost RFE Training Shape Old: ', x.shape)
print('BEFORE XGBoost RFE Training Columns Old: ', x.columns.values)
print("BEFORE XGBoost RFE Training w/bin score mean: {:.2f}". format(XGB_base_
print("BEFORE XGBoost RFE Test w/bin score mean: {:.2f}". format(XGB_base_res
print("BEFORE XGBoost RFE Test w/bin score 3*std: +/- {:.2f}". format(XGB_bas
print('-'*10)



XGB_rfe = feature_selection.RFECV(XGB, step = 1, scoring = 'accuracy', cv = 3
XGB_rfe.fit(X_train, y_train)

#transform x&y to reduced features and fit new model
#alternative: can use pipeline to reduce fit and transform steps: http://scik
X_rfe = x.columns.values[XGB_rfe.get_support()]
rfe_results = model_selection.cross_validate(XGB, x[X_rfe], y, cv  = 3,return_

#print(XGB_rfe.grid_scores_)
print('AFTER XGBoost RFE Training Shape New: ', x[X_rfe].shape)
print('AFTER XGBoost RFE Training Columns New: ', X_rfe)
print("AFTER XGBoost RFE Training w/bin score mean: {:.2f}". format(rfe_resul
print("AFTER XGBoost RFE Test w/bin score mean: {:.2f}". format(rfe_results['
print("AFTER XGBoost RFE Test w/bin score 3*std: +/- {:.2f}". format(rfe_resu
print('-'*10)


#tune rfe model
rfe_tune_model = model_selection.GridSearchCV(XGBClassifier(), param_grid=XGB_
rfe_tune_model.fit(x[X_rfe], y)

#print(rfe_tune_model.cv_results_.keys())
#print(rfe_tune_model.cv_results_['params'])
print('AFTER XGBoost RFE Tuned Parameters: ', rfe_tune_model.best_params_)
#print(rfe_tune_model.cv_results_['mean_train_score'])
print("AFTER XGBoost RFE Tuned Training w/bin score mean: {:.2f}". format(rfe_
#print(rfe_tune_model.cv_results_['mean_test_score'])
print("AFTER XGBoost RFE Tuned Test w/bin score mean: {:.2f}". format(rfe_tun
print("AFTER XGBoost RFE Tuned Test w/bin score 3*std: +/- {:.2f}". format(rf
print('-'*10)
```

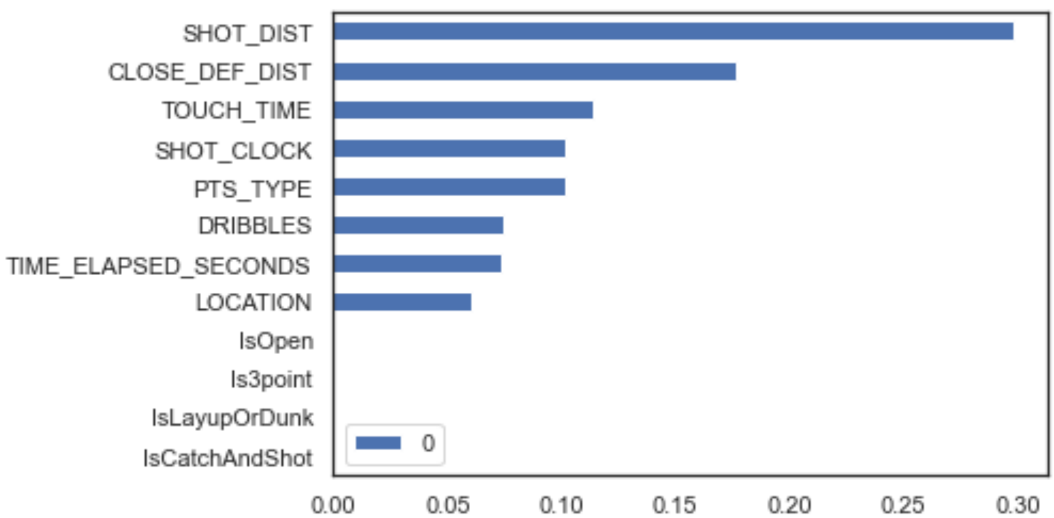
```
BEFORE XGBoost RFE Training Shape Old:  (128069, 12)
BEFORE XGBoost RFE Training Columns Old:  ['LOCATION' 'SHOT_CLOCK' 'DRIBBLES'
'TOUCH_TIME' 'SHOT_DIST' 'PTS_TYPE'
 'CLOSE_DEF_DIST' 'TIME_ELAPSED_SECONDS' 'IsCatchAndShot' 'IsLayupOrDunk'
 'Is3point' 'IsOpen']
BEFORE XGBoost RFE Training w/bin score mean: 67.90
BEFORE XGBoost RFE Test w/bin score mean: 61.25
BEFORE XGBoost RFE Test w/bin score 3*std: +/- 0.57
----------
AFTER XGBoost RFE Training Shape New:  (128069, 6)
AFTER XGBoost RFE Training Columns New:  ['SHOT_CLOCK' 'DRIBBLES' 'TOUCH_TIME'
'SHOT_DIST' 'PTS_TYPE'
 'CLOSE_DEF_DIST']
AFTER XGBoost RFE Training w/bin score mean: 65.87
AFTER XGBoost RFE Test w/bin score mean: 61.51
AFTER XGBoost RFE Test w/bin score 3*std: +/- 0.88
----------
AFTER XGBoost RFE Tuned Parameters:  {'max_depth': 2, 'random_state': 0}
AFTER XGBoost RFE Tuned Training w/bin score mean: 64.84
AFTER XGBoost RFE Tuned Test w/bin score mean: 64.04
AFTER XGBoost RFE Tuned Test w/bin score 3*std: +/- 0.74
----------
```

In [217…

```
sort_XGB_feature_importances = pd.DataFrame(XGB.feature_importances_, x.colum
sort_XGB_feature_importances.sort_values(0,ascending = True, axis=0,inplace=T
sort_XGB_feature_importances.plot(kind='barh')
```

Out[217…

```
<AxesSubplot:>
```



Step 6: Validate and Implement

Step 7: Optimize and Strategize

Conclusion

Iteration one of the Data Science Framework, seems to converge on 0.77990 submission accuracy. Using the same dataset and different implementation of a random forest (adaboost,decision tree, gradient boost, xgboost, etc.) with tuning does not exceed the 0.77990 submission accuracy. Interesting for this dataset, the simple random forest algorithm had the best default submission score and with tuning achieved the same best accuracy score.

While no general conclusions can be made from testing a handful of algorithms on a single dataset, there are several observations on the mentioned dataset.

The train dataset has a different distribution than the test/validation dataset and population. This created wide margins between the cross validation (CV) accuracy score . Given the same dataset, random forest based algorithms, seemed to converge on the same accuracy score after proper tuning. Despite tuning, no machine learning algorithm, exceeded the homemade algorithm. The author will theorize, that for small datasets, a manmade algorithm is the bar to beat. With that in mind, for iteration two, I would spend more time on preprocessing and feature engineering. In order to better align the CV score and improve the overall accuracy.

```
In [ ]:
```