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

# Upload the original full dataset.

data = pd.read_csv('mlb 1972-2021.csv')

# Check for any missing values. In this case, there are none.

null_count = data.isnull().sum()
null_count
```

```
Tm          0
Year        0
#Bat        0
BatAge      0
G           0
           ..
#a-tA-S     0
1Run        0
Under500    0
SOS         0
W-L%        0
Length: 80, dtype: int64
```

```
# A preview of the dataset.

data.head()
```

```
                  Tm  Year  #Bat  BatAge    G         PA         AB
R/G  \
0     Atlanta Braves  1972    36    27.2  155  38.316129  34.051613
4.05
1  Baltimore Orioles  1972    32    28.7  154  36.857143  32.649351
3.37
2     Boston Red Sox  1972    38    28.9  155  37.883871  33.600000
4.13
3  California Angels  1972    40    29.1  155  36.387097  33.322581
2.93
4       Chicago Cubs  1972    41    30.0  156  38.147436  33.634615
4.39

          H        2B  ...   Fld%  Rtot  Rtot/yr  BPF  PPF  #a-tA-S
1Run  \
0  8.793548  1.200000  ...  0.974   -38       -4  109  109       13
0.536
1  7.487013  1.253247  ...  0.983    78        8  103  100       17
0.448
2  8.316129  1.477419  ...  0.978   -43       -4  106  105       19
```

```
0.579
3  8.058065  1.103226  ...  0.981     8        1   94   95       12
0.567
4  8.628205  1.320513  ...  0.979   -25       -2  110  109       16
0.449

   Under500  SOS   W-L%
0     0.458  0.1  0.455
1     0.448  0.0  0.519
2     0.445  0.0  0.548
3     0.361  0.0  0.484
4     0.551 -0.1  0.548

[5 rows x 80 columns]
```

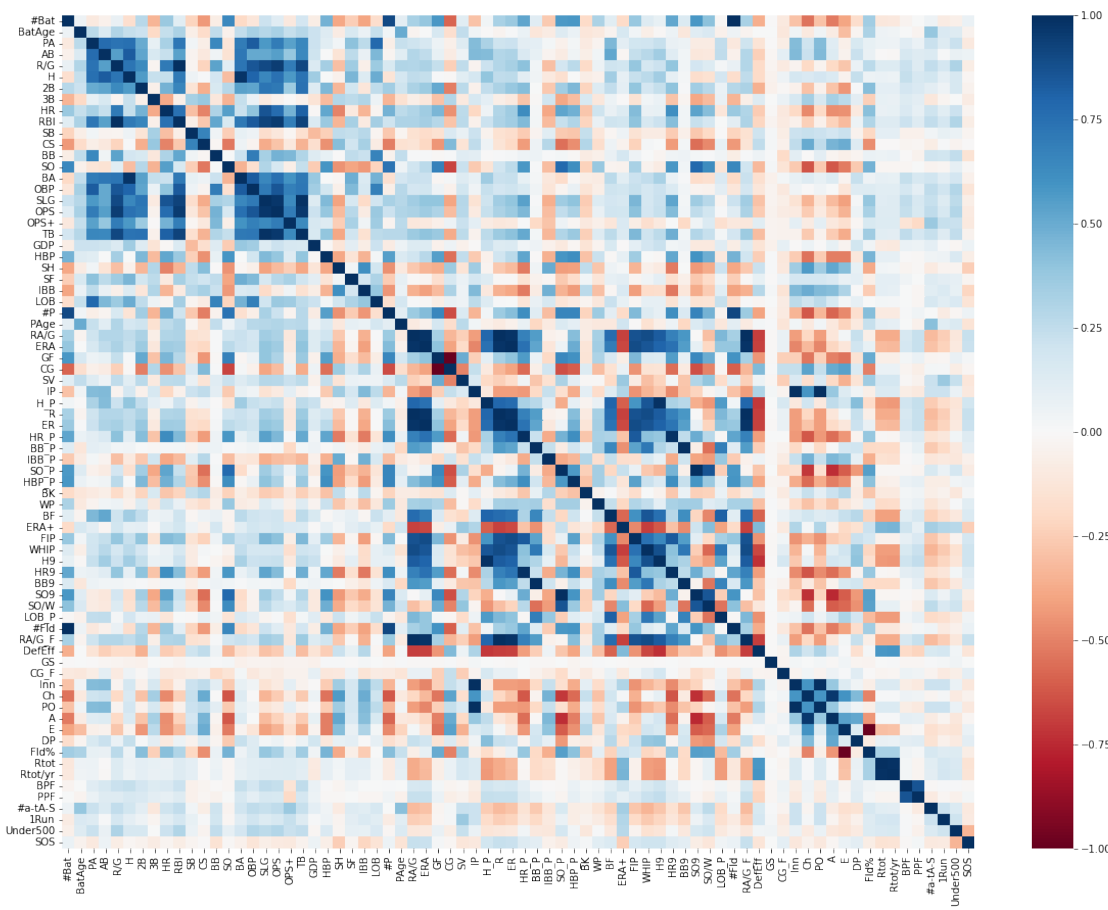
```python
# Team and Year variables will not be selected as part of the final
dataset as they are irrelevant to the research.
# The target variable, W-L% (Win/Loss Percentage) will be temporarily
removed for the upcoming analysis.

mlbdata = data.iloc[:, 2:79]
mlbdata.head()
```

```
   #Bat  BatAge    G         PA         AB   R/G         H
2B  \
0    36    27.2  155  38.316129  34.051613  4.05  8.793548  1.200000

1    32    28.7  154  36.857143  32.649351  3.37  7.487013  1.253247

2    38    28.9  155  37.883871  33.600000  4.13  8.316129  1.477419

3    40    29.1  155  36.387097  33.322581  2.93  8.058065  1.103226

4    41    30.0  156  38.147436  33.634615  4.39  8.628205  1.320513


         3B        HR  ...        DP   Fld%  Rtot  Rtot/yr  BPF  PPF
#a-tA-S  \
0  0.109677  0.929032  ...  0.838710  0.974   -38       -4  109  109
13
1  0.188312  0.649351  ...  0.974026  0.983    78        8  103  100
17
2  0.219355  0.800000  ...  0.909677  0.978   -43       -4  106  105
19
3  0.167742  0.503226  ...  0.870968  0.981     8        1   94   95
12
4  0.256410  0.852564  ...  0.948718  0.979   -25       -2  110  109
16

    1Run  Under500  SOS
```

```
0  0.536     0.458  0.1
1  0.448     0.448  0.0
2  0.579     0.445  0.0
3  0.567     0.361  0.0
4  0.449     0.551 -0.1

[5 rows x 77 columns]
```

```python
# 'G' variable (number of games played per season) will be dropped, as the majority of the feature variables are averaged by this number.
# 'cSho' and 'tSho' variables (Complete game shutout and team shut out, respectively) are also dropped to eliminate bias.
# A shutout results in a win 100% of the time.

mlbdata = mlbdata.drop(['G','cSho','tSho'],axis=1)

corr = mlbdata.corr()
plt.subplots(figsize=(20,15))
sns.heatmap(corr, cmap='RdBu', vmin=-1, vmax=1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb72076b6d0>
```



```python
# All highly correlated features with a correlation coefficient
greater than 0.80 plus or minus will be removed from the dataset.

corr_matrix = corr
upper_tri =
corr_matrix.where(np.triu(np.ones(corr_matrix.shape),k=1).astype(bool)
)
to_drop = [column for column in upper_tri.columns if
any(upper_tri[column] > 0.80) or any(upper_tri[column] < -0.80)]

# The following list contains all of the feature variables with strong
pairwise correlation.

for i in to_drop:
  print(i)
```

```
H
RBI
BA
OBP
SLG
OPS
TB
#P
ERA
CG
R
ER
BF
FIP
WHIP
H9
HR9
BB9
SO9
SO/W
LOB_P
#Fld
RA/G_F
Inn
PO
A
Fld%
Rtot/yr
PPF
```

```python
# The above variables will be dropped from the dataset.

for i in to_drop:
  mlbdata = mlbdata.drop([i],axis=1)
```

```
# The dataset has been reduced to 45 feature variables from the
original count of 79.

mlbdata.head()

   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...      CG_F         Ch         E        DP
Rtot  \
0  0.303226  0.225806  ...  7.380645  38.193548  1.006452  0.838710
-38
1  0.506494  0.266234  ...  7.155844  38.090909  0.649351  0.974026
78
2  0.425806  0.193548  ...  7.580645  37.967742  0.838710  0.909677
-43
3  0.367742  0.238710  ...  7.245161  37.761290  0.735484  0.870968
8
4  0.442308  0.301282  ...  7.410256  39.852564  0.846154  0.948718
-25

   BPF  #a-tA-S   1Run  Under500  SOS
0  109       13  0.536     0.458  0.1
1  103       17  0.448     0.448  0.0
2  106       19  0.579     0.445  0.0
3   94       12  0.567     0.361  0.0
4  110       16  0.449     0.551 -0.1

[5 rows x 45 columns]

# The remaining variables will be normalized through Min-Max
Normalization to rescale the range of features for low variance
analysis.

mlbdatanorm = pd.DataFrame(mlbdata)
mlbdatanormal = (mlbdatanorm - mlbdatanorm.min()) /
(mlbdatanorm.max()-mlbdatanorm.min())

# The variance of the feature variables is analyzed to explore the
need for further dimensionality reduction.
```

```
variance = mlbdatanormal.var()
varianceScore = pd.DataFrame(variance.values)
varianceColumn = pd.DataFrame(variance.index)

varianceDF = pd.concat([varianceColumn,varianceScore], axis=1)
varianceDF.columns = ['Feature','Variance']
pd.set_option('display.max_rows', None)

varianceDF.sort_values(by='Variance', ascending=True)
```

	Feature	Variance
34	GS	0.000534
30	BK	0.006901
23	IP	0.009247
3	AB	0.010397
17	IBB	0.012450
40	BPF	0.012871
2	PA	0.014751
8	SB	0.015078
33	DefEff	0.017971
36	Ch	0.018299
43	Under500	0.018470
24	H_P	0.019347
6	3B	0.019541
14	HBP	0.019562
44	SOS	0.019662
20	RA/G	0.019999
19	PAge	0.021331
32	ERA+	0.021505
39	Rtot	0.021820
18	LOB	0.022587
25	HR_P	0.022849
26	BB_P	0.023101
1	BatAge	0.023211
22	SV	0.023363
9	CS	0.023659
42	1Run	0.023667
35	CG_F	0.024333
38	DP	0.024971
7	HR	0.025006
16	SF	0.025267
0	#Bat	0.025269
31	WP	0.026245
21	GF	0.026356
4	R/G	0.026966
27	IBB_P	0.027405
15	SH	0.027787
37	E	0.027925
12	OPS+	0.028466

```
10        BB  0.028727
13       GDP  0.028911
5         2B  0.029189
41   #a-tA-S  0.031543
29     HBP_P  0.031587
28      SO_P  0.035214
11        SO  0.038384
```

```python
# The 'GS' variable (number of games started on defense) will be
dropped from the dataset due to the extremely low variance.
# The number of games started on defense remains essentially
unchanged.

mlbdata = mlbdata.drop(['GS'], axis=1)

# The target variable is inserted back into the revised dataset.

mlbdata.insert(44,'W-L%',data['W-L%'])

# After dimensionality reduction, the final working dataset consists
of 44 feature variables and 1 target variable.

mlbdata.head()
```

```
   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...         Ch         E        DP  Rtot  BPF
#a-tA-S  \
0  0.303226  0.225806  ...  38.193548  1.006452  0.838710   -38  109
13
1  0.506494  0.266234  ...  38.090909  0.649351  0.974026    78  103
17
2  0.425806  0.193548  ...  37.967742  0.838710  0.909677   -43  106
19
3  0.367742  0.238710  ...  37.761290  0.735484  0.870968     8   94
12
4  0.442308  0.301282  ...  39.852564  0.846154  0.948718   -25  110
16

    1Run  Under500  SOS   W-L%
```

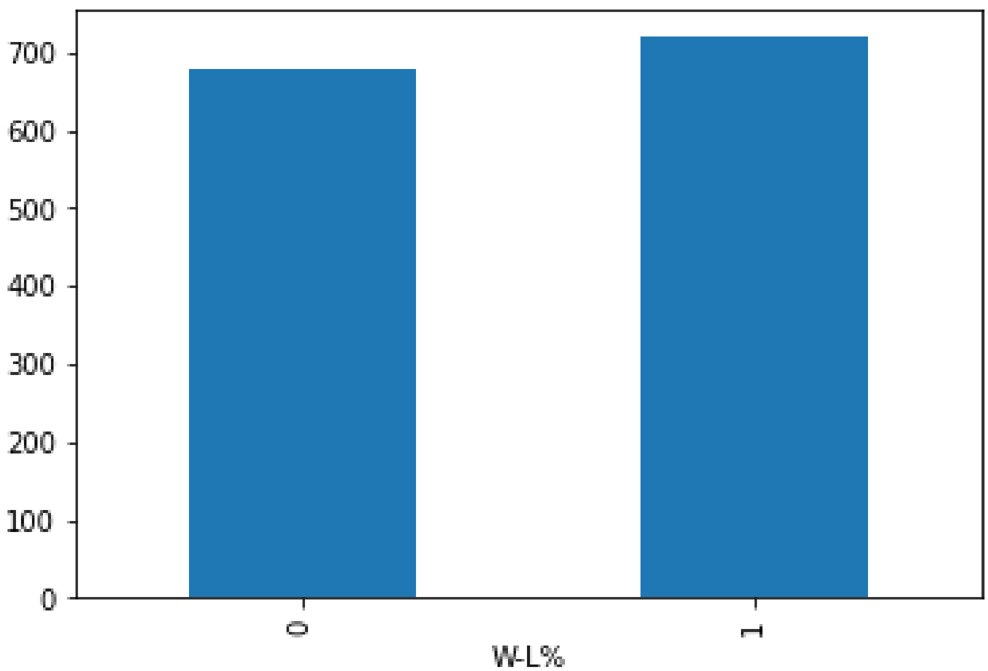
```
0  0.536     0.458  0.1  0.455
1  0.448     0.448  0.0  0.519
2  0.579     0.445  0.0  0.548
3  0.567     0.361  0.0  0.484
4  0.449     0.551 -0.1  0.548

[5 rows x 45 columns]
```

```python
# Save the new dataset to .csv format.

mlbdata.to_csv("mlbdata.csv")
```

```python
# The target variable, win/loss percentage, will be converted to a
binary class for machine learning.
# Teams with a winning percentage below .500 will be represented by 0,
teams over .500 will be represented by 1.
# The target variable is close to evenly balanced, with 48% of the
data being in class 0 and 52% being in class 1.

mlbdata['W-L%'] = ((mlbdata['W-L%'] >= .500).replace({True: 1, False:
0}))
frequency = mlbdata['W-L%'].groupby(mlbdata['W-L%']).count()
print(frequency)
frequency.plot(kind='bar')
```

```
W-L%
0    677
1    719
Name: W-L%, dtype: int64
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb704a9c290>
```



700
600
500
400
300
200
100
0
0
1
W-L%

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

data = pd.read_csv('mlbdata.csv')

mlbdata = data.iloc[:, :44]
mlbdata.head()

   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...      CG_F         Ch         E        DP
Rtot  \
0  0.303226  0.225806  ...  7.380645  38.193548  1.006452  0.838710
-38
1  0.506494  0.266234  ...  7.155844  38.090909  0.649351  0.974026
78
2  0.425806  0.193548  ...  7.580645  37.967742  0.838710  0.909677
-43
3  0.367742  0.238710  ...  7.245161  37.761290  0.735484  0.870968
8
4  0.442308  0.301282  ...  7.410256  39.852564  0.846154  0.948718
-25

   BPF  #a-tA-S   1Run  Under500  SOS
0  109       13  0.536     0.458  0.1
1  103       17  0.448     0.448  0.0
2  106       19  0.579     0.445  0.0
3   94       12  0.567     0.361  0.0
4  110       16  0.449     0.551 -0.1

[5 rows x 44 columns]

# All feature variables are then normalized using Min Max
Normalization to scale the data.

mlbdatanorm = pd.DataFrame(mlbdata)
mlbdatanormal = (mlbdatanorm - mlbdatanorm.min()) /
```

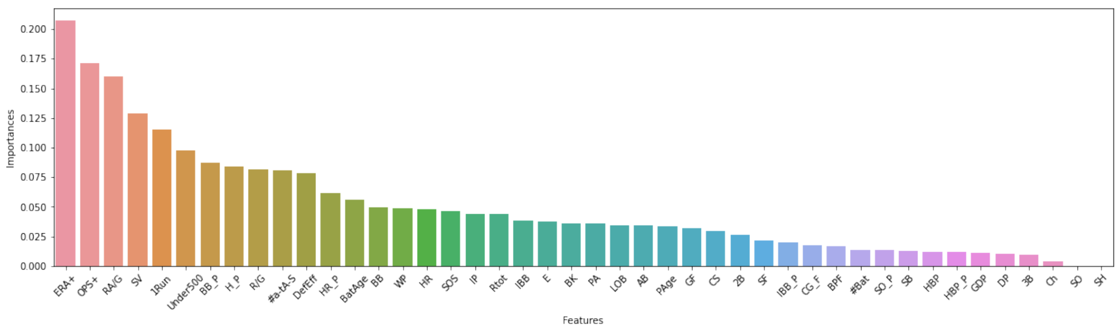
```python
(mlbdatanorm.max()-mlbdatanorm.min())

# Target variable is inserted back into the dataset, and renamed
'WinningRecord'.

mlbdatanormal.insert(44,'WinningRecord',data['W-L%'])

# Feature variable (win/loss percentage) is converted into a binary
class variable represented by values 0 and 1.

mlbdatanormal['WinningRecord'] = ((mlbdatanormal['WinningRecord'] >=
.500).replace({True: 1, False: 0}))

X = mlbdatanormal.iloc[:,0:-1]
y = mlbdatanormal['WinningRecord']

from sklearn.model_selection import train_test_split
from collections import defaultdict
from operator import itemgetter
from sklearn.feature_selection import SelectKBest, mutual_info_classif
mic = SelectKBest(score_func=mutual_info_classif)

# Train/test split and mutual information algorithm is run through 250
seperate iterations and appended to a list.
# The resulting scores are then divided by the total number of
iterations to generate a final dependancy ranking.

dicts = defaultdict(list)
finallist = []
for num in range(250):
  X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
  fit = mic.fit(X_train,y_train)
  dfscores = pd.DataFrame(fit.scores_)
  dfcolumns = pd.DataFrame(X_train.columns)
  featureScores = pd.concat([dfcolumns,dfscores], axis=1)
  featureScores.columns = ['Feature','Score']
  keys = featureScores.index
  values = featureScores.loc[:,'Score']
  for i in keys:
    dicts[i].append(values[i])
for k, v in (dicts.items()):
  total = np.sum(v)
  np.sort(total, axis=None)
  final = (k, total/250)
  finallist.append(final)

finaldf = pd.DataFrame(finallist, columns=['Feature', 'Score'])

allFeatures = featureScores.nlargest(44, 'Score')
allFeatures
```

```
     Feature     Score
32      ERA+  0.207229
12      OPS+  0.171607
20      RA/G  0.160237
22        SV  0.128710
41      1Run  0.114884
42  Under500  0.097407
26      BB_P  0.086893
24       H_P  0.083897
4        R/G  0.081322
40   #a-tA-S  0.080874
33    DefEff  0.078711
25      HR_P  0.061770
1     BatAge  0.056321
10        BB  0.049429
31        WP  0.049138
7         HR  0.047565
43       SOS  0.046270
23        IP  0.044032
38      Rtot  0.043968
17       IBB  0.038224
36         E  0.037465
30        BK  0.036169
2         PA  0.036059
18       LOB  0.034690
3         AB  0.034568
19      PAge  0.033590
21        GF  0.032132
9         CS  0.029734
5         2B  0.026164
16        SF  0.021828
27     IBB_P  0.020062
34      CG_F  0.017499
39       BPF  0.017033
0       #Bat  0.013873
28      SO_P  0.013540
8         SB  0.012527
14       HBP  0.011741
29     HBP_P  0.011673
13       GDP  0.011169
37        DP  0.010056
6         3B  0.009846
35        Ch  0.004105
11        SO  0.000000
15        SH  0.000000
```

```
# A visual representation of variable dependancy ranked from highest
to lowest.

importances = allFeatures['Score']
```

```
final_df2 = pd.DataFrame({'Features': allFeatures['Feature'],
'Importances':importances})
final_df2.set_index('Importances')

final_df2 = final_df2.sort_values('Importances', ascending=False)
plt.figure(figsize=(20,5))
plt.xticks(rotation=45)
sns.barplot(x='Features',y='Importances', data=final_df2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff763d454d0>
```




```
# The features are ordered from highest dependancy to lowest, and a
new dataset is created.
# This dataset will be used to complete the final step in the
dimensionality reduction, where a set number of variables will be
selected for machine learning.

featuresRanked = []
for i in allFeatures['Feature'].head(44):
  featuresRanked.append(i)

featuresRankedMIC = mlbdatanormal[featuresRanked + ['WinningRecord']]

featuresRankedMIC.to_csv("featuresRankedMIC.csv")
```

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

data = pd.read_csv('mlbdata.csv')

mlbdata = data.iloc[:, :44]
mlbdata.head()

   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...      CG_F         Ch         E        DP
Rtot  \
0  0.303226  0.225806  ...  7.380645  38.193548  1.006452  0.838710
-38
1  0.506494  0.266234  ...  7.155844  38.090909  0.649351  0.974026
78
2  0.425806  0.193548  ...  7.580645  37.967742  0.838710  0.909677
-43
3  0.367742  0.238710  ...  7.245161  37.761290  0.735484  0.870968
8
4  0.442308  0.301282  ...  7.410256  39.852564  0.846154  0.948718
-25

   BPF  #a-tA-S   1Run  Under500  SOS
0  109       13  0.536     0.458  0.1
1  103       17  0.448     0.448  0.0
2  106       19  0.579     0.445  0.0
3   94       12  0.567     0.361  0.0
4  110       16  0.449     0.551 -0.1

[5 rows x 44 columns]

# All feature variables are then normalized using Min Max
Normalization to scale the data.

mlbdatanorm = pd.DataFrame(mlbdata)
mlbdatanormal = (mlbdatanorm - mlbdatanorm.min()) /
```

```
(mlbdatanorm.max()-mlbdatanorm.min())

# Target variable is inserted back into the dataset, and renamed
'WinningRecord'.

mlbdatanormal.insert(44,'WinningRecord',data['W-L%'])

# Feature variable (win/loss percentage) is converted into a binary
class variable represented by values 0 and 1.

mlbdatanormal['WinningRecord'] = ((mlbdatanormal['WinningRecord'] >=
.500).replace({True: 1, False: 0}))

mlbdatanormal.head()

    #Bat    BatAge        PA        AB       R/G        2B
3B  \
0  0.175  0.292135  0.594126  0.701810  0.339394  0.151402  0.172537

1  0.075  0.460674  0.356231  0.445843  0.133333  0.191710  0.344682

2  0.225  0.483146  0.523644  0.619373  0.363636  0.361411  0.412642

3  0.275  0.505618  0.279588  0.568733  0.000000  0.078143  0.299651

4  0.300  0.606742  0.566620  0.625692  0.442424  0.242631  0.493763


         HR        SB        CS  ...        Ch         E        DP
Rtot  \
0  0.380797  0.096689  0.246236  ...  0.621987  0.743973  0.384946
0.391473
1  0.213898  0.197865  0.302859  ...  0.609655  0.349282  0.565368
0.841085
2  0.303797  0.157703  0.201055  ...  0.594857  0.558574  0.479570
0.372093
3  0.126699  0.128802  0.264308  ...  0.570052  0.444482  0.427957
0.569767
4  0.335165  0.165917  0.351948  ...  0.821312  0.566802  0.531624
0.441860

        BPF  #a-tA-S      1Run  Under500       SOS  WinningRecord
0  0.547619     0.45  0.559387  0.402703  0.545455              0
1  0.404762     0.65  0.390805  0.389189  0.454545              1
2  0.476190     0.75  0.641762  0.385135  0.454545              1
3  0.190476     0.40  0.618774  0.271622  0.454545              0
4  0.571429     0.60  0.392720  0.528378  0.363636              1

[5 rows x 45 columns]
```

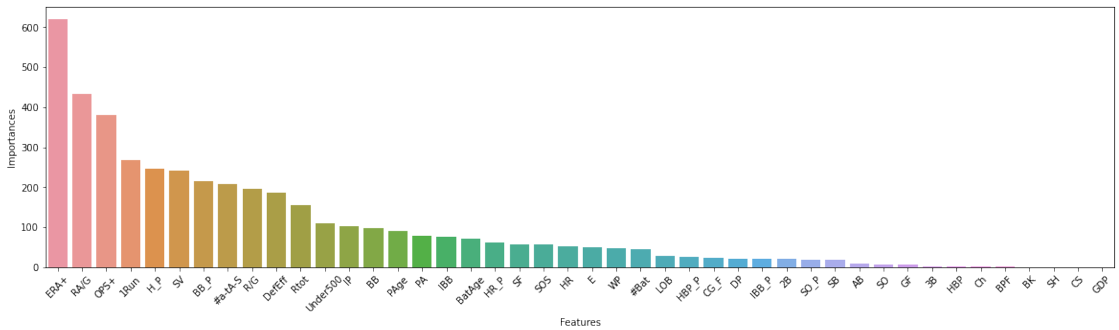
```python
X = mlbdatanormal.iloc[:,0:-1]
y = mlbdatanormal['WinningRecord']

from sklearn.model_selection import train_test_split
from collections import defaultdict
from sklearn.feature_selection import SelectKBest, f_classif
fc = SelectKBest(score_func=f_classif)

# Train/test split and f-statistic algorithm is run through 250
seperate iterations and appended to a list.
# The resulting f values are then divided by the total number of
iterations to identify feature significance.

dicts = defaultdict(list)
finallist = []
for num in range(250):
  X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
  fit = fc.fit(X_train,y_train)
  dfscores = pd.DataFrame(fit.scores_)
  dfcolumns = pd.DataFrame(X_train.columns)
  featureScores = pd.concat([dfcolumns,dfscores], axis=1)
  featureScores.columns = ['Feature','Score']
  keys = featureScores.index
  values = featureScores.loc[:,'Score']
  for i in keys:
    dicts[i].append(values[i])
for k, v in (dicts.items()):
  total = np.sum(v)
  np.sort(total, axis=None)
  final = (k, total/250)
  finallist.append(final)

finaldf = pd.DataFrame(finallist, columns=['Feature', 'Score'])

finaldf.sort_values(by='Score', ascending=False)
allFeatures = featureScores.nlargest(44, 'Score')
allFeatures
```

	Feature	Score
32	ERA+	620.196582
20	RA/G	433.099595
12	OPS+	380.473512
41	1Run	268.377813
24	H_P	245.887168
22	SV	241.582781
26	BB_P	215.484569
40	#a-tA-S	207.575522
4	R/G	195.363866
33	DefEff	186.508176
38	Rtot	156.200932

```
42  Under500  109.528665
23        IP  103.778197
10        BB   98.406377
19      PAge   91.434156
2         PA   78.078066
17       IBB   75.293526
1     BatAge   71.999036
25      HR_P   62.462063
16        SF   58.164168
43       SOS   57.521253
7         HR   52.093041
36         E   50.414073
31        WP   48.502836
0       #Bat   44.117991
18       LOB   29.120736
29     HBP_P   26.784488
34      CG_F   23.087719
37        DP   22.502983
27     IBB_P   22.363738
5         2B   21.458827
28      SO_P   19.369666
8         SB   18.767150
3         AB    8.267317
11        SO    7.156416
21        GF    6.785282
6         3B    2.529453
14       HBP    2.209288
35        Ch    1.332602
39       BPF    1.112041
30        BK    0.751678
15        SH    0.449555
9         CS    0.227785
13       GDP    0.142234
```

```
# A visual representation of variable significance ranked from highest
to lowest.

importances = allFeatures['Score']
final_df2 = pd.DataFrame({'Features': allFeatures['Feature'],
'Importances':importances})
final_df2.set_index('Importances')

final_df2 = final_df2.sort_values('Importances', ascending=False)
plt.figure(figsize=(20,5))
plt.xticks(rotation=45)
sns.barplot(x='Features',y='Importances', data=final_df2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff1acd9a0d0>
```




```
# The features are ordered by significance from highest to lowest, and a new dataset is created.
# This dataset will be used to complete the final step in the dimensionality reduction, where a set number of variables will be selected for machine learning.

featuresRanked = []
for i in allFeatures['Feature'].head(44):
  featuresRanked.append(i)

featuresRankedFC = mlbdatanormal[featuresRanked + ['WinningRecord']]

featuresRankedFC.to_csv("featuresRankedFC.csv")
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statistics
import seaborn as sns
from scipy import stats
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

data = pd.read_csv('mlbdata.csv')

mlbdata = data.iloc[:, :44]
mlbdata.head()

   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...      CG_F         Ch         E        DP
Rtot  \
0  0.303226  0.225806  ...  7.380645  38.193548  1.006452  0.838710
-38
1  0.506494  0.266234  ...  7.155844  38.090909  0.649351  0.974026
78
2  0.425806  0.193548  ...  7.580645  37.967742  0.838710  0.909677
-43
3  0.367742  0.238710  ...  7.245161  37.761290  0.735484  0.870968
8
4  0.442308  0.301282  ...  7.410256  39.852564  0.846154  0.948718
-25

   BPF  #a-tA-S   1Run  Under500  SOS
0  109       13  0.536     0.458  0.1
1  103       17  0.448     0.448  0.0
2  106       19  0.579     0.445  0.0
3   94       12  0.567     0.361  0.0
4  110       16  0.449     0.551 -0.1

[5 rows x 44 columns]

# All feature variables are then normalized using Min Max
Normalization to scale the data.
```

```
mlbdatanorm = pd.DataFrame(mlbdata)
mlbdatanormal = (mlbdatanorm - mlbdatanorm.min()) /
(mlbdatanorm.max()-mlbdatanorm.min())

# Target variable is inserted back into the dataset, and renamed
'WinningRecord'.

mlbdatanormal.insert(44,'WinningRecord',data['W-L%'])

# Feature variable (win/loss percentage) is converted into a binary
class variable represented by values 0 and 1.

mlbdatanormal['WinningRecord'] = ((mlbdatanormal['WinningRecord'] >=
.500).replace({True: 1, False: 0}))

mlbdatanormal.head()
```

```
    #Bat    BatAge        PA        AB      R/G        2B
3B  \
0  0.175  0.292135  0.594126  0.701810  0.339394  0.151402  0.172537

1  0.075  0.460674  0.356231  0.445843  0.133333  0.191710  0.344682

2  0.225  0.483146  0.523644  0.619373  0.363636  0.361411  0.412642

3  0.275  0.505618  0.279588  0.568733  0.000000  0.078143  0.299651

4  0.300  0.606742  0.566620  0.625692  0.442424  0.242631  0.493763


         HR        SB        CS  ...        Ch         E        DP
Rtot  \
0  0.380797  0.096689  0.246236  ...  0.621987  0.743973  0.384946
0.391473
1  0.213898  0.197865  0.302859  ...  0.609655  0.349282  0.565368
0.841085
2  0.303797  0.157703  0.201055  ...  0.594857  0.558574  0.479570
0.372093
3  0.126699  0.128802  0.264308  ...  0.570052  0.444482  0.427957
0.569767
4  0.335165  0.165917  0.351948  ...  0.821312  0.566802  0.531624
0.441860

        BPF  #a-tA-S      1Run  Under500       SOS  WinningRecord
0  0.547619     0.45  0.559387  0.402703  0.545455              0
1  0.404762     0.65  0.390805  0.389189  0.454545              1
2  0.476190     0.75  0.641762  0.385135  0.454545              1
3  0.190476     0.40  0.618774  0.271622  0.454545              0
4  0.571429     0.60  0.392720  0.528378  0.363636              1
```

[5 rows x 45 columns]

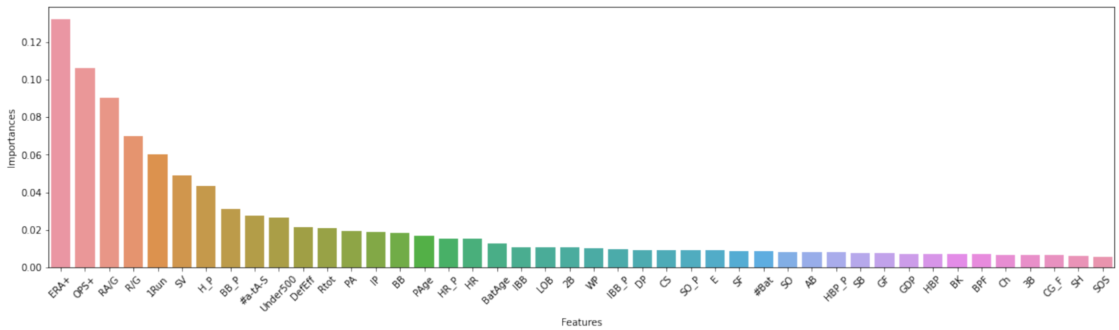
```
X = mlbdatanormal.iloc[:,0:-1]
y = mlbdatanormal['WinningRecord']

rf = RandomForestClassifier()
from collections import defaultdict

# Train/test split and random forest algorithm is run through 250
seperate iterations and appended to a list.
# The resulting scores are then divided by the total number of
iterations to generate a final ranking of feature importance.


dicts = defaultdict(list)
finallist = []
for num in range(250):
  X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
  fit = rf.fit(X_train,y_train)
  dfscores = pd.DataFrame(fit.feature_importances_)
  dfcolumns = pd.DataFrame(X_train.columns)
  featureScores = pd.concat([dfcolumns,dfscores], axis=1)
  featureScores.columns = ['Feature','Score']
  keys = featureScores.index
  values = featureScores.loc[:,'Score']
  for i in keys:
    dicts[i].append(values[i])
for k, v in (dicts.items()):
  total = np.sum(v)
  np.sort(total, axis=None)
  final = (k, total/250)
  finallist.append(final)

finaldf = pd.DataFrame(finallist, columns=['Feature', 'Score'])

finaldf.sort_values(by='Score', ascending=False)
allFeatures = featureScores.nlargest(44, 'Score')
allFeatures
```

	Feature	Score
32	ERA+	0.132205
12	OPS+	0.106154
20	RA/G	0.090351
4	R/G	0.069869
41	1Run	0.060101
22	SV	0.048866
24	H_P	0.043516
26	BB_P	0.031232
40	#a-tA-S	0.027413

```
42  Under500  0.026695
33    DefEff  0.021536
38      Rtot  0.021113
2         PA  0.019559
23        IP  0.018787
10        BB  0.018344
19      PAge  0.016836
25      HR_P  0.015289
7         HR  0.015065
1     BatAge  0.012879
17       IBB  0.010789
18       LOB  0.010694
5         2B  0.010540
31        WP  0.010130
27     IBB_P  0.009412
37        DP  0.009378
9         CS  0.009183
28      SO_P  0.009077
36         E  0.009063
16        SF  0.008760
0       #Bat  0.008393
11        SO  0.008035
3         AB  0.007932
29     HBP_P  0.007898
8         SB  0.007602
21        GF  0.007368
13       GDP  0.007239
14       HBP  0.007181
30        BK  0.007016
39       BPF  0.006989
35        Ch  0.006843
6         3B  0.006674
34      CG_F  0.006672
15        SH  0.005910
43       SOS  0.005412
```

```
# A visual representation of variable importance ranked from highest
to lowest.

importances = allFeatures['Score']
final_df2 = pd.DataFrame({'Features': allFeatures['Feature'],
'Importances':importances})
final_df2.set_index('Importances')

final_df2 = final_df2.sort_values('Importances', ascending=False)
plt.figure(figsize=(20,5))
plt.xticks(rotation=45)
sns.barplot(x='Features',y='Importances', data=final_df2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1488f2a390>
```




```
# The features are ordered by importance from highest to lowest, and a
new dataset is created.
# This dataset will be used to complete the final step in the
dimensionality reduction, where a set number of variables will be
selected for machine learning.

featuresRanked = []
for i in allFeatures['Feature'].head(44):
  featuresRanked.append(i)

featuresRankedRFC = mlbdatanormal[featuresRanked + ['WinningRecord']]

featuresRankedRFC.to_csv("featuresRankedRFC.csv")
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import xticks, yticks
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

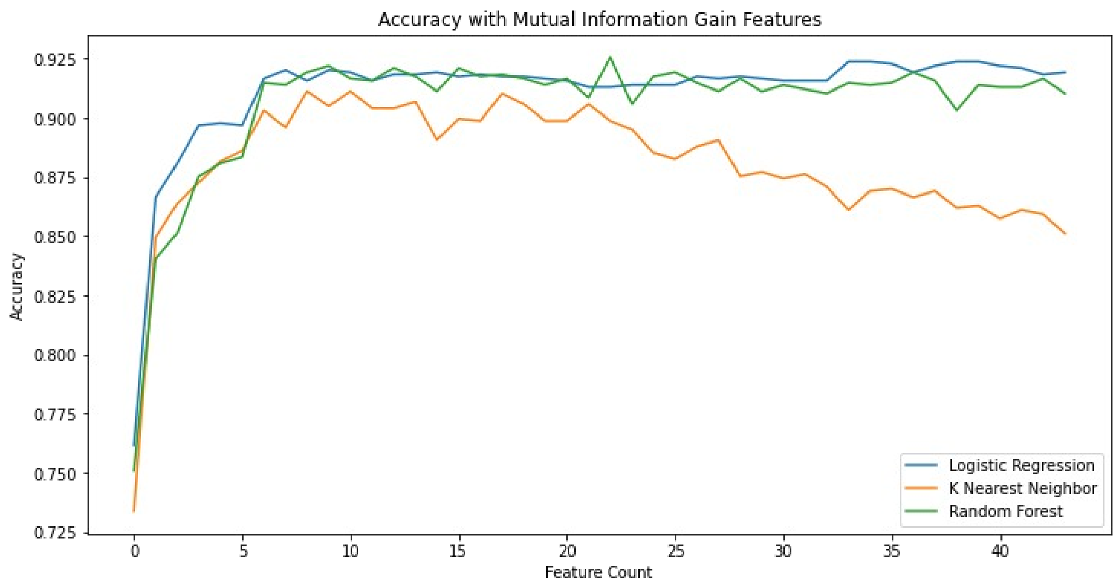
```
featuresMIC = pd.read_csv('featuresRankedMIC.csv')
featuresFC = pd.read_csv('featuresRankedFC.csv')
featuresRFC = pd.read_csv('featuresRankedRFC.csv')
```

Mutual Information Gain Features

```
featuresMIC.head()
```

```
       ERA+  OPS+      RA/G      1Run      BB_P       H_P       R/G
SV  \
0  0.173913  0.44  0.477612  0.559387  0.429180  0.622353  0.339394
0.302094
1  0.637681  0.32  0.000000  0.390805  0.154861  0.181001  0.133333
0.194577
2  0.231884  0.56  0.300995  0.641762  0.429180  0.464920  0.363636
0.265422
3  0.275362  0.36  0.161692  0.618774  0.688073  0.159225  0.000000
0.100396
4  0.594203  0.46  0.208955  0.392720  0.204570  0.482468  0.442424
0.390013

   Under500  #a-tA-S  ...        AB       BPF        SH        SF
3B  \
0  0.402703     0.45  ...  0.701810  0.547619  0.404816  0.358281
0.172537
1  0.389189     0.65  ...  0.445843  0.404762  0.481526  0.398932
0.344682
2  0.385135     0.75  ...  0.619373  0.476190  0.412176  0.538171
0.412642
3  0.271622     0.40  ...  0.568733  0.190476  0.485779  0.124424
0.299651
4  0.528378     0.60  ...  0.625692  0.571429  0.489978  0.443268
0.493763

         DP        SO     HBP_P       GDP  WinningRecord
0  0.384946  0.207712  0.175247  0.625415              0
1  0.565368  0.376993  0.058972  0.443133              1
2  0.479570  0.294791  0.313891  0.414272              1
3  0.427957  0.286874  0.196577  0.562072              0
4  0.531624  0.247071  0.258242  0.564103              1
```

```
[5 rows x 45 columns]

xMIC = featuresMIC.iloc[:,0:-1]
yMIC = featuresMIC['WinningRecord']

# Stratified train test split is used to preserves the same
proportions of examples in each class as observed in the original
dataset.

X_train, X_test, y_train, y_test = train_test_split(xMIC, yMIC,
test_size=.2, random_state=1, stratify=yMIC)

# For-loop is created to evaluate the accuracy score of each learning
algorithm as features are added one by one to the dataset

resultsLR = []
for i in range(1,45):
  score = cross_val_score(LogisticRegression(), X_train.iloc[:, 0:i],
y_train, scoring='accuracy', cv=10)
  resultsLR.append(np.mean(score))

resultsKNN = []
for i in range(1,45):
  score = cross_val_score(KNeighborsClassifier(), X_train.iloc[:,
0:i], y_train, scoring='accuracy', cv=10)
  resultsKNN.append(np.mean(score))

resultsRF = []
for i in range(1,45):
  score = cross_val_score(RandomForestClassifier(), X_train.iloc[:,
0:i], y_train, scoring='accuracy', cv=10)
  resultsRF.append(np.mean(score))

# Visual representation of the performance of all three machine
learning algorithms

plt.figure(figsize=(12,6))
plt.title("Accuracy with Mutual Information Gain Features")
plt.xlabel("Feature Count")
plt.ylabel("Accuracy")
xticks(np.arange(0,45, step=5))
yticks(np.arange(.50, .95, step=0.025))
plt.plot(resultsLR, label = "Logistic Regression")
plt.plot(resultsKNN, label = "K Nearest Neighbor")
plt.plot(resultsRF, label = "Random Forest")
plt.legend()
plt.show()
```

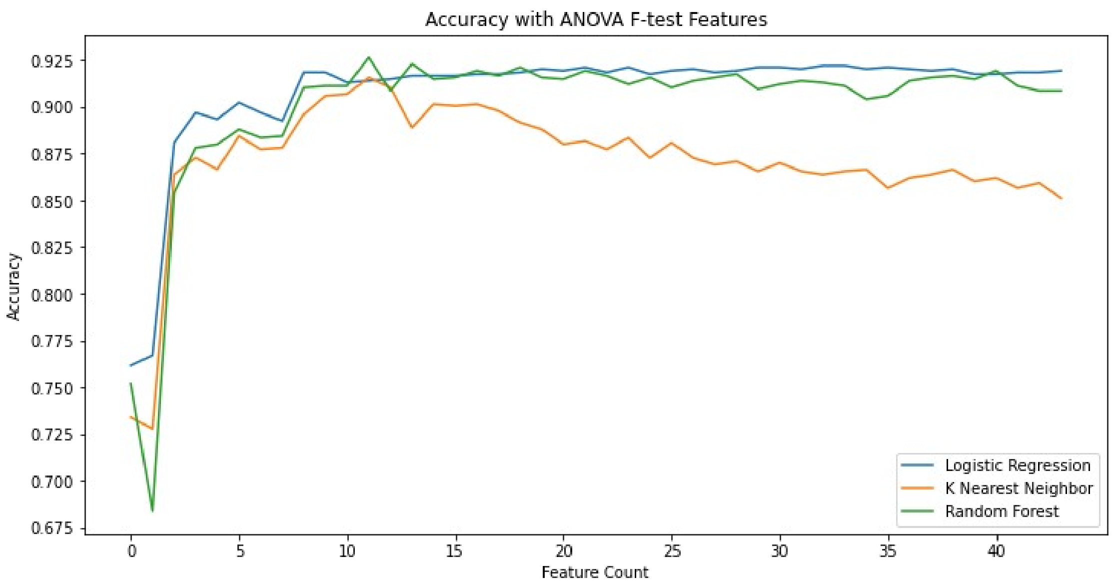


ANOVA F-test Features

```
featuresFC.head()

       ERA+      RA/G  OPS+      1Run       H_P        SV      BB_P
#a-tA-S  \
0  0.173913  0.477612  0.44  0.559387  0.622353  0.302094  0.429180
0.45
1  0.637681  0.000000  0.32  0.390805  0.181001  0.194577  0.154861
0.65
2  0.231884  0.300995  0.56  0.641762  0.464920  0.265422  0.429180
0.75
3  0.275362  0.161692  0.36  0.618774  0.159225  0.100396  0.688073
0.40
4  0.594203  0.208955  0.46  0.392720  0.482468  0.390013  0.204570
0.60

        R/G    DefEff  ...        GF        3B       HBP        Ch
BPF  \
0  0.339394  0.602041  ...  0.555251  0.172537  0.209898  0.621987
0.547619
1  0.133333  1.000000  ...  0.306162  0.344682  0.219712  0.609655
0.404762
2  0.363636  0.581633  ...  0.466301  0.412642  0.225806  0.594857
0.476190
3  0.000000  0.867347  ...  0.366232  0.299651  0.130358  0.570052
0.190476
4  0.442424  0.663265  ...  0.403437  0.493763  0.152792  0.821312
0.571429

         BK        SH        CS       GDP  WinningRecord
```

```
0  0.110017  0.404816  0.246236  0.625415              0
1  0.027683  0.481526  0.302859  0.443133              1
2  0.027504  0.412176  0.201055  0.414272              1
3  0.041256  0.485779  0.264308  0.562072              0
4  0.054656  0.489978  0.351948  0.564103              1

[5 rows x 45 columns]
```

```
# The above steps are repeated for the ranked features from the ANOVA
f-test selection algorithm

xFC = featuresFC.iloc[:,0:-1]
yFC = featuresFC['WinningRecord']

X_train, X_test, y_train, y_test = train_test_split(xFC, yFC,
test_size=.2, random_state=1, stratify=yFC)

resultsLR_FC = []
for i in range(1,45):
  score = cross_val_score(LogisticRegression(), X_train.iloc[:, 0:i],
y_train, scoring='accuracy', cv=10)
  resultsLR_FC.append(np.mean(score))

resultsKNN_FC = []
for i in range(1,45):
  score = cross_val_score(KNeighborsClassifier(), X_train.iloc[:,
0:i], y_train, scoring='accuracy', cv=10)
  resultsKNN_FC.append(np.mean(score))

resultsRF_FC = []
for i in range(1,45):
  score = cross_val_score(RandomForestClassifier(), X_train.iloc[:,
0:i], y_train, scoring='accuracy', cv=10)
  resultsRF_FC.append(np.mean(score))

plt.figure(figsize=(12,6))
plt.title("Accuracy with ANOVA F-test Features")
plt.xlabel("Feature Count")
plt.ylabel("Accuracy")
xticks(np.arange(0,45, step=5))
yticks(np.arange(.50, .95, step=0.025))
plt.plot(resultsLR_FC, label = "Logistic Regression")
plt.plot(resultsKNN_FC, label = "K Nearest Neighbor")
plt.plot(resultsRF_FC, label = "Random Forest")
plt.legend()
plt.show()
```

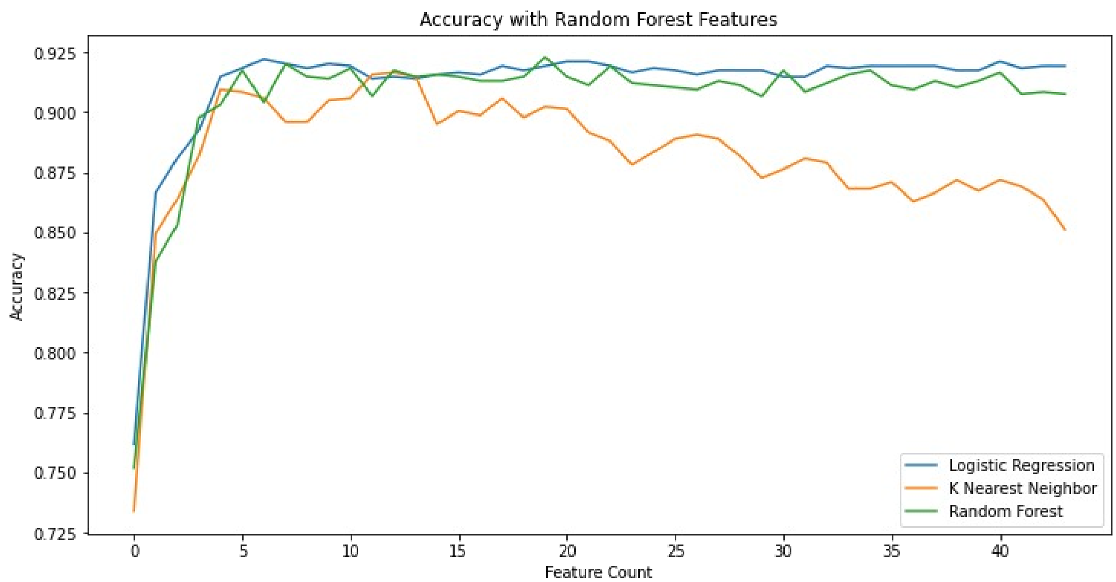


Random Forest Features

```
featuresRFC.head()

       ERA+   OPS+       RA/G        R/G       1Run         SV        H_P  
BB_P  \
0  0.173913   0.44   0.477612   0.339394   0.559387   0.302094   0.622353  
0.429180
1  0.637681   0.32   0.000000   0.133333   0.390805   0.194577   0.181001  
0.154861
2  0.231884   0.56   0.300995   0.363636   0.641762   0.265422   0.464920  
0.429180
3  0.275362   0.36   0.161692   0.000000   0.618774   0.100396   0.159225  
0.688073
4  0.594203   0.46   0.208955   0.442424   0.392720   0.390013   0.482468  
0.204570

   #a-tA-S  Under500  ...        GDP        HBP         BK        BPF  
Ch  \
0     0.45  0.402703  ...   0.625415   0.209898   0.110017   0.547619  
0.621987
1     0.65  0.389189  ...   0.443133   0.219712   0.027683   0.404762  
0.609655
2     0.75  0.385135  ...   0.414272   0.225806   0.027504   0.476190  
0.594857
3     0.40  0.271622  ...   0.562072   0.130358   0.041256   0.190476  
0.570052
4     0.60  0.528378  ...   0.564103   0.152792   0.054656   0.571429  
0.821312

         3B      CG_F          SH        SOS   WinningRecord
```

```
0  0.172537  0.757567  0.404816  0.545455              0
1  0.344682  0.662841  0.481526  0.454545              1
2  0.412642  0.841843  0.412176  0.454545              1
3  0.299651  0.700477  0.485779  0.454545              0
4  0.493763  0.770045  0.489978  0.363636              1

[5 rows x 45 columns]
```

```
# The above steps are repeated for the ranked features from the Random
Forest selection algorithm

xRFC = featuresRFC.iloc[:,0:-1]
yRFC = featuresRFC['WinningRecord']

X_train, X_test, y_train, y_test = train_test_split(xRFC, yRFC,
test_size=.2, random_state=1, stratify=yRFC)

resultsLR_RFC = []
for i in range(1,45):
  score = cross_val_score(LogisticRegression(), X_train.iloc[:, 0:i],
y_train, scoring='accuracy', cv=10)
  resultsLR_RFC.append(np.mean(score))

resultsKNN_RFC = []
for i in range(1,45):
  score = cross_val_score(KNeighborsClassifier(), X_train.iloc[:,
0:i], y_train, scoring='accuracy', cv=10)
  resultsKNN_RFC.append(np.mean(score))

resultsRF_RFC = []
for i in range(1,45):
  score = cross_val_score(RandomForestClassifier(), X_train.iloc[:,
0:i], y_train, scoring='accuracy', cv=10)
  resultsRF_RFC.append(np.mean(score))

plt.figure(figsize=(12,6))
plt.title("Accuracy with Random Forest Features")
plt.xlabel("Feature Count")
plt.ylabel("Accuracy")
xticks(np.arange(0,45, step=5))
yticks(np.arange(.50, .95, step=0.025))
plt.plot(resultsLR_RFC, label = "Logistic Regression")
plt.plot(resultsKNN_RFC, label = "K Nearest Neighbor")
plt.plot(resultsRF_RFC, label = "Random Forest")
plt.legend()
plt.show()
```



Accuracy with Random Forest Features
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
Accuracy
0
5
10
15
20
25
30
35
40
Feature Count
Logistic Regression
K Nearest Neighbor
Random Forest

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_predict, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score,
validation_curve, StratifiedShuffleSplit
from sklearn import metrics

# The three datasets created during the feature selection process are
uploaded.
# Each dataset contains a different ordering of the feature variables
based on their importance.

featuresMIC = pd.read_csv('featuresRankedMIC.csv')
featuresFC = pd.read_csv('featuresRankedFC.csv')
featuresRFC = pd.read_csv('featuresRankedRFC.csv')
```

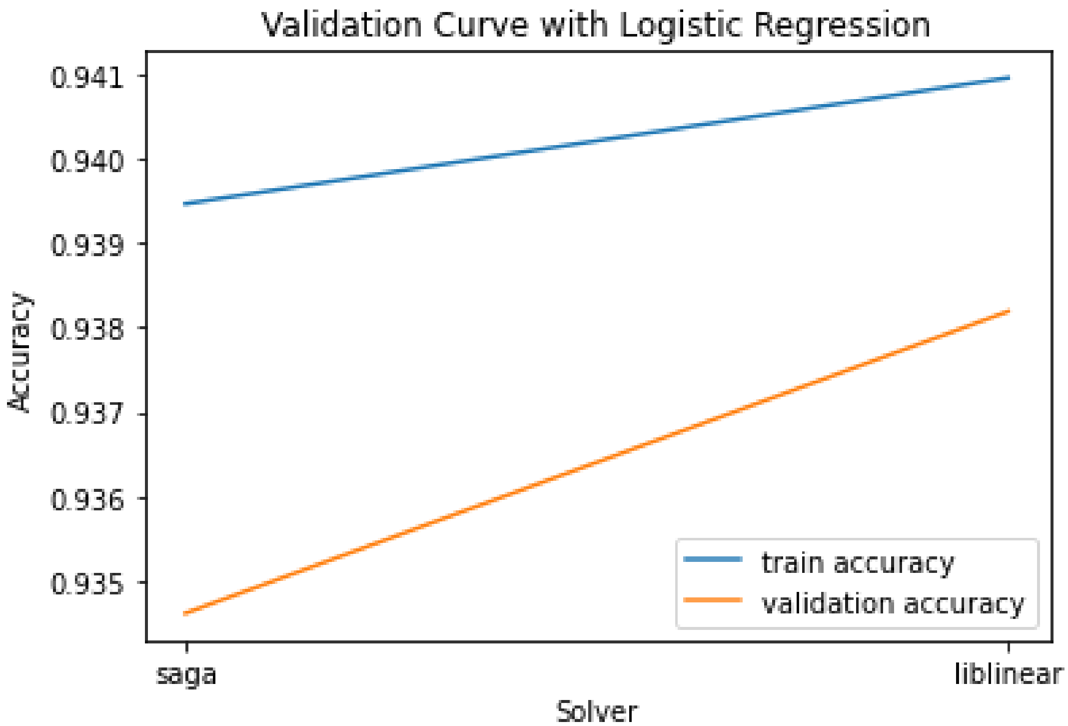
Mutual Information Gain

```
# The top 20 features will be selected from each dataset for machine
learning.

X = featuresMIC.iloc[:,0:20]
y = featuresMIC['WinningRecord']

featuresMIC.iloc[:,0:20].head()

       ERA+  OPS+      RA/G      1Run      BB_P       H_P       R/G
SV  \
0  0.173913  0.44  0.477612  0.559387  0.429180  0.622353  0.339394
0.302094
1  0.637681  0.32  0.000000  0.390805  0.154861  0.181001  0.133333
0.194577
2  0.231884  0.56  0.300995  0.641762  0.429180  0.464920  0.363636
0.265422
3  0.275362  0.36  0.161692  0.618774  0.688073  0.159225  0.000000
0.100396
4  0.594203  0.46  0.208955  0.392720  0.204570  0.482468  0.442424
0.390013

   Under500  #a-tA-S        HR      Rtot       SOS      HR_P        BB
\
0  0.402703     0.45  0.380797  0.391473  0.545455  0.418914  0.465585

1  0.389189     0.65  0.213898  0.841085  0.454545  0.123964  0.410213
```

```
2  0.385135     0.75  0.303797  0.372093  0.454545  0.189573  0.440077

3  0.271622     0.40  0.126699  0.569767  0.454545  0.142855  0.021753

4  0.528378     0.60  0.335165  0.441860  0.363636  0.233241  0.540521


     DefEff       IBB     IBB_P        PA    BatAge
0  0.602041  0.288005  0.495551  0.594126  0.292135
1  1.000000  0.380978  0.471563  0.356231  0.460674
2  0.581633  0.322774  0.441491  0.523644  0.483146
3  0.867347  0.343635  0.477531  0.279588  0.505618
4  0.663265  0.520958  0.501326  0.566620  0.606742
```

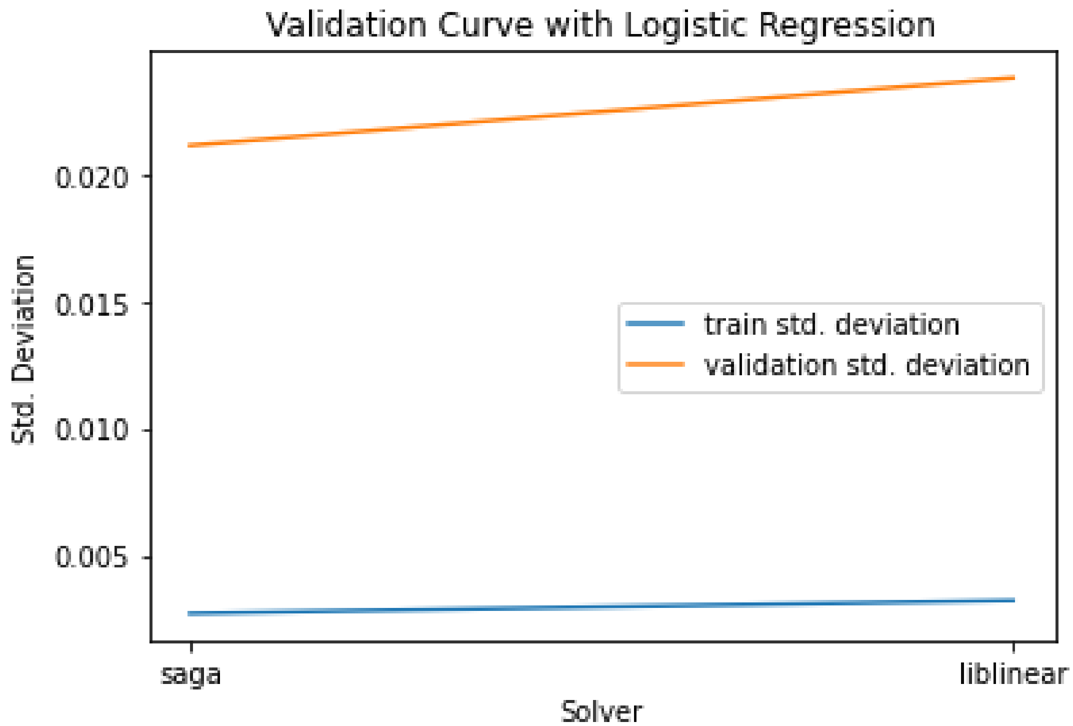
```
# A stratified train-test split is performed on the dataset, with 80%
of the data assigned to training set and 20% assigned to testing set.

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

# Parameters for Logistic Regression algorithm

penalty = ['l1','none', 'l2']
solver = ['saga', 'liblinear', 'sag', 'lbfgs', 'newton-cg']

# 10-fold cross validation is performed on the training set for
parameter tuning.
# The l1 penalty term has shown to provide the highest accuracy
results, and the solver algorithm will be tested for accuracy

train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("Accuracy")
plt.plot(solver, train_scores_mean, label="train accuracy")
plt.plot(solver, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.93461229 0.93819176        nan        nan        nan]
```

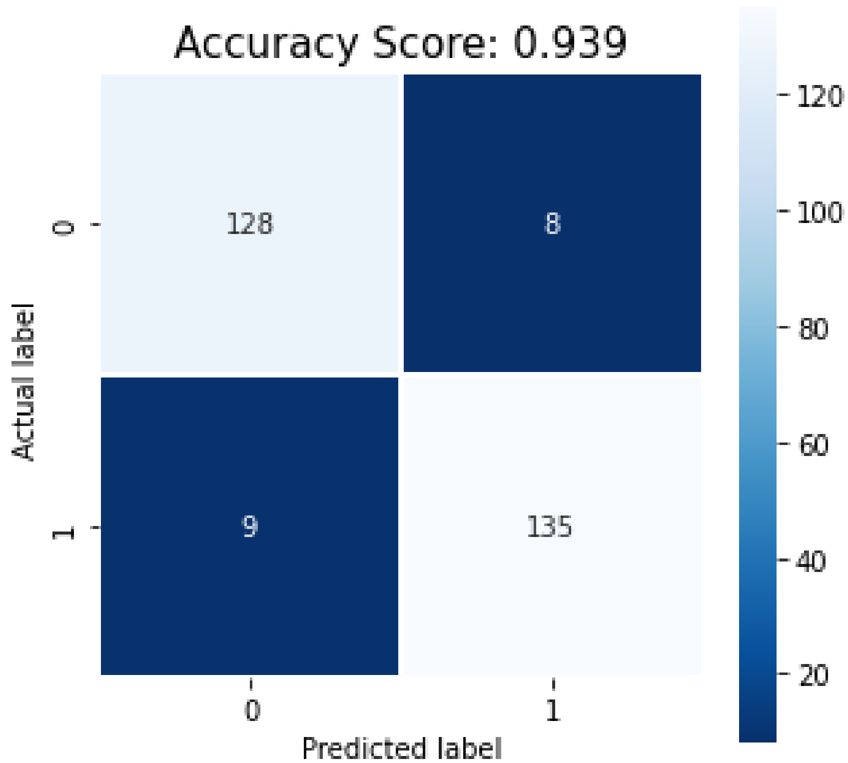



```
# The standard deviation also evaluated for each solver algorithm

train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("Std. Deviation")
plt.plot(solver, train_scores_std, label="train std. deviation")
plt.plot(solver, valid_scores_std, label="validation std. deviation")

plt.legend()
plt.show()
```

```
[0.02111613 0.0237711         nan        nan        nan]
```




```
logreg = LogisticRegression(max_iter=5000, solver='liblinear',
penalty='l1')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  135
FP:  8
TN:  128
FN:  9

              precision    recall  f1-score   support

           0       0.93      0.94      0.94       136
           1       0.94      0.94      0.94       144

    accuracy                           0.94       280
   macro avg       0.94      0.94      0.94       280
weighted avg       0.94      0.94      0.94       280
```

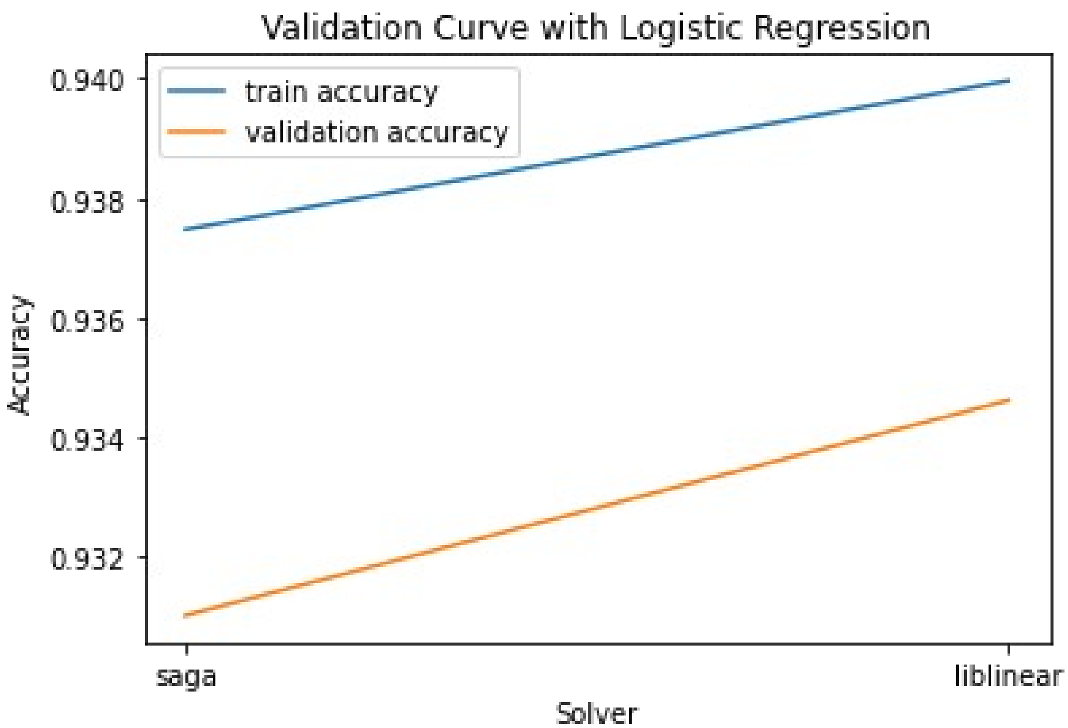
ANOVA F-Test

```
# The above process is now repeated for the top 20 features as
selected by the ANOVA F-test selection algorithm.
```

```
X = featuresFC.iloc[:,0:20]
y = featuresFC['WinningRecord']

featuresFC.iloc[:,0:20].head()

       ERA+      RA/G  OPS+      1Run       H_P        SV      BB_P
#a-tA-S  \
0  0.173913  0.477612  0.44  0.559387  0.622353  0.302094  0.429180
0.45
1  0.637681  0.000000  0.32  0.390805  0.181001  0.194577  0.154861
0.65
2  0.231884  0.300995  0.56  0.641762  0.464920  0.265422  0.429180
0.75
3  0.275362  0.161692  0.36  0.618774  0.159225  0.100396  0.688073
0.40
4  0.594203  0.208955  0.46  0.392720  0.482468  0.390013  0.204570
0.60

        R/G    DefEff      Rtot  Under500        IP        BB
PAge  \
0  0.339394  0.602041  0.391473  0.402703  0.678982  0.465585
0.405941
1  0.133333  1.000000  0.841085  0.389189  0.697641  0.410213
0.465347
2  0.363636  0.581633  0.372093  0.385135  0.710241  0.440077
0.475248
3  0.000000  0.867347  0.569767  0.271622  0.680184  0.021753
0.336634
4  0.442424  0.663265  0.441860  0.528378  0.752545  0.540521
0.455446

         PA       IBB    BatAge      HR_P        SF
0  0.594126  0.288005  0.292135  0.418914  0.358281
1  0.356231  0.380978  0.460674  0.123964  0.398932
2  0.523644  0.322774  0.483146  0.189573  0.538171
3  0.279588  0.343635  0.505618  0.142855  0.124424
4  0.566620  0.520958  0.606742  0.233241  0.443268

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
```

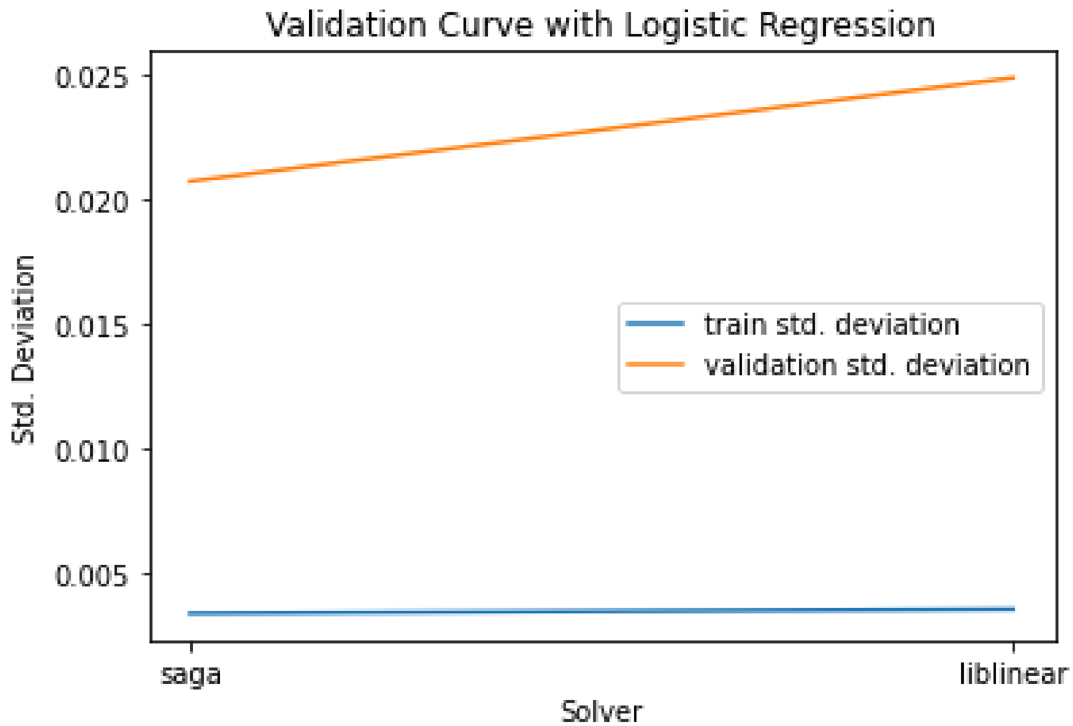
```
plt.ylabel("Accuracy")
plt.plot(solver, train_scores_mean, label="train accuracy")
plt.plot(solver, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.93102477 0.93462033        nan        nan        nan]
```

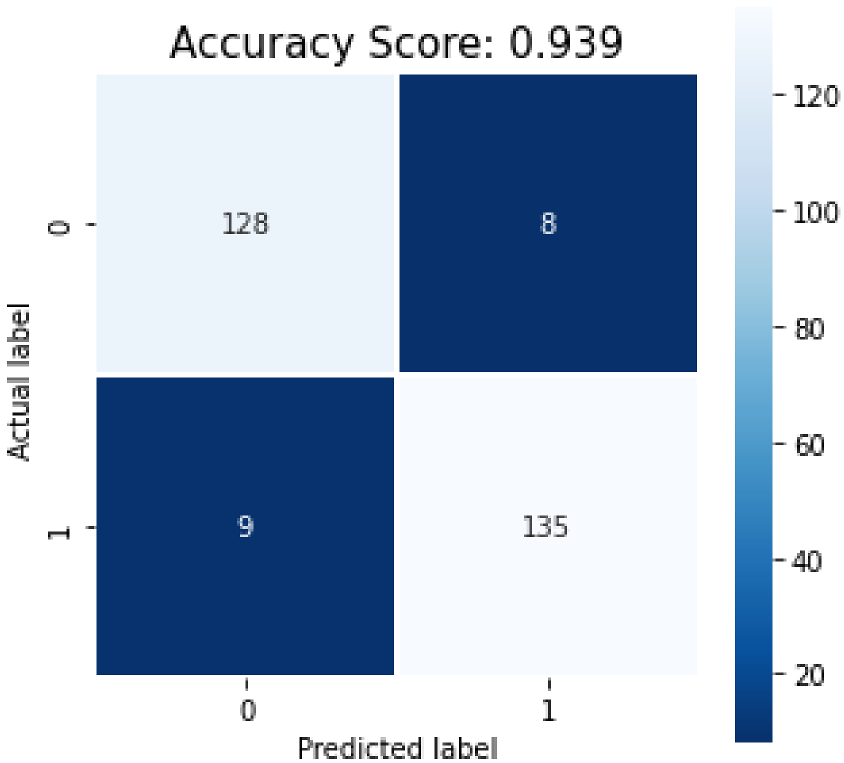



```
train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("Std. Deviation")
plt.plot(solver, train_scores_std, label="train std. deviation")
plt.plot(solver, valid_scores_std, label="validation std. deviation")

plt.legend()
plt.show()
```

```
[0.02074305 0.02490883        nan        nan        nan]
```




```
logreg = LogisticRegression(max_iter=5000, solver='liblinear',
penalty='l1')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  135
FP:  8
TN:  128
FN:  9

              precision    recall  f1-score   support

           0       0.93      0.94      0.94       136
           1       0.94      0.94      0.94       144

    accuracy                           0.94       280
   macro avg       0.94      0.94      0.94       280
weighted avg       0.94      0.94      0.94       280
```

Random Forest

```
# The above process is now repeated for the top 20 features as
selected by the Random Forest selection algorithm.
```

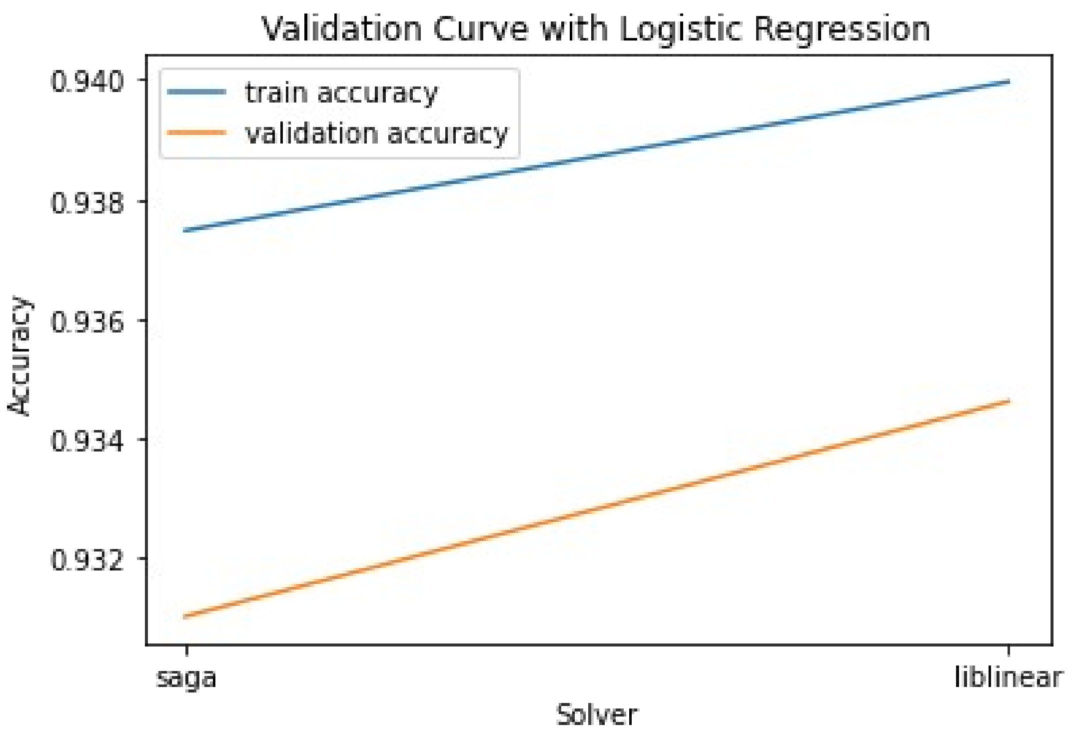
```
X = featuresRFC.iloc[:,0:20]
y = featuresRFC['WinningRecord']

featuresRFC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G       R/G      1Run        SV       H_P
BB_P  \
0  0.173913  0.44  0.477612  0.339394  0.559387  0.302094  0.622353
0.429180
1  0.637681  0.32  0.000000  0.133333  0.390805  0.194577  0.181001
0.154861
2  0.231884  0.56  0.300995  0.363636  0.641762  0.265422  0.464920
0.429180
3  0.275362  0.36  0.161692  0.000000  0.618774  0.100396  0.159225
0.688073
4  0.594203  0.46  0.208955  0.442424  0.392720  0.390013  0.482468
0.204570

   #a-tA-S  Under500    DefEff      Rtot        PA        IP        BB
\
0     0.45  0.402703  0.602041  0.391473  0.594126  0.678982  0.465585

1     0.65  0.389189  1.000000  0.841085  0.356231  0.697641  0.410213

2     0.75  0.385135  0.581633  0.372093  0.523644  0.710241  0.440077

3     0.40  0.271622  0.867347  0.569767  0.279588  0.680184  0.021753

4     0.60  0.528378  0.663265  0.441860  0.566620  0.752545  0.540521


       PAge      HR_P        HR    BatAge       IBB
0  0.405941  0.418914  0.380797  0.292135  0.288005
1  0.465347  0.123964  0.213898  0.460674  0.380978
2  0.475248  0.189573  0.303797  0.483146  0.322774
3  0.336634  0.142855  0.126699  0.505618  0.343635
4  0.455446  0.233241  0.335165  0.606742  0.520958
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Logistic Regression")
```

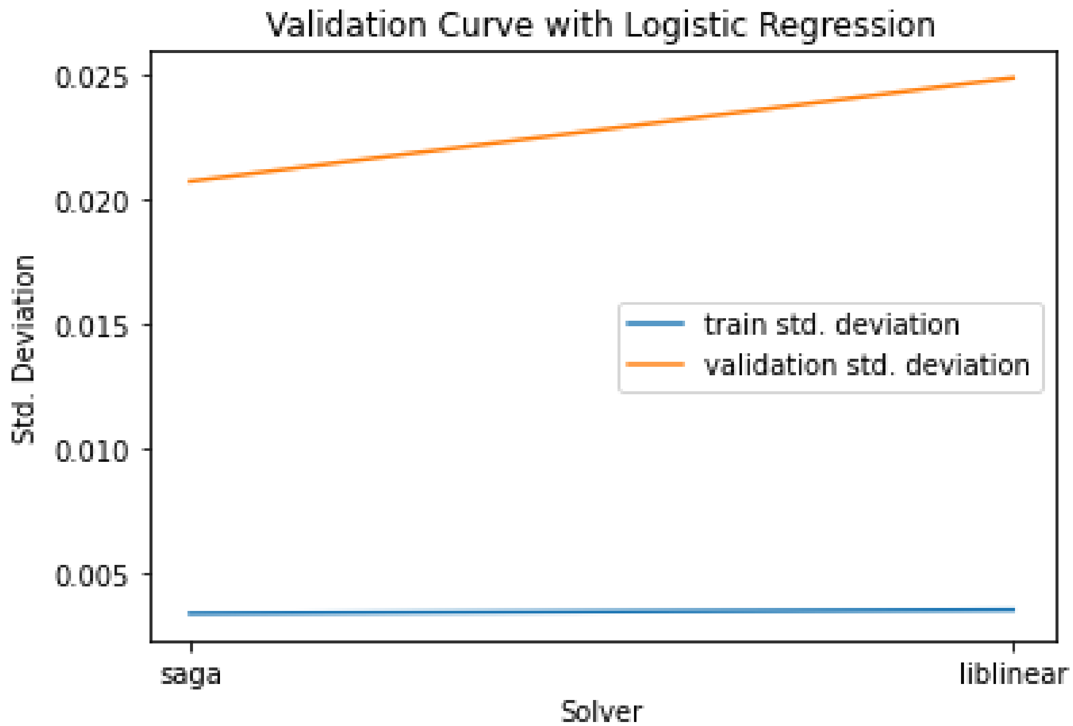
```
plt.xlabel("Solver")
plt.ylabel("Accuracy")
plt.plot(solver, train_scores_mean, label="train accuracy")
plt.plot(solver, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.93102477 0.93462033        nan        nan        nan]
```

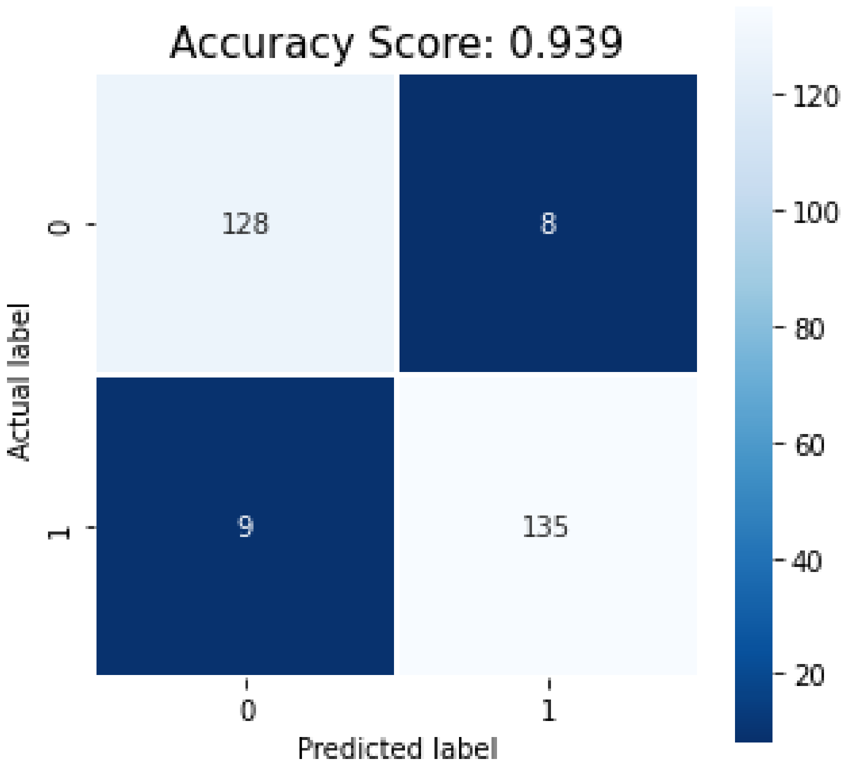



```
train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("Std. Deviation")
plt.plot(solver, train_scores_std, label="train std. deviation")
plt.plot(solver, valid_scores_std, label="validation std. deviation")

plt.legend()
plt.show()
```

```
[0.02074305 0.02490883        nan        nan        nan]
```




```
logreg = LogisticRegression(max_iter=5000, solver='liblinear',
penalty='l1')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  135
FP:  8
TN:  128
FN:  9

              precision    recall  f1-score   support

           0       0.93      0.94      0.94       136
           1       0.94      0.94      0.94       144

    accuracy                           0.94       280
   macro avg       0.94      0.94      0.94       280
weighted avg       0.94      0.94      0.94       280
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, cross_val_score,
validation_curve, StratifiedShuffleSplit
from sklearn import metrics
```

```
# The three datasets created during the feature selection process are
uploaded.
# Each dataset contains a different ordering of the feature variables
based on their importance.

featuresMIC = pd.read_csv('featuresRankedMIC.csv')
featuresFC = pd.read_csv('featuresRankedFC.csv')
featuresRFC = pd.read_csv('featuresRankedRFC.csv')
```

Mutual Information Gain

```
# The top 20 features will be selected from each dataset for machine
learning.

X = featuresMIC.iloc[:,0:20]
y = featuresMIC['WinningRecord']
```

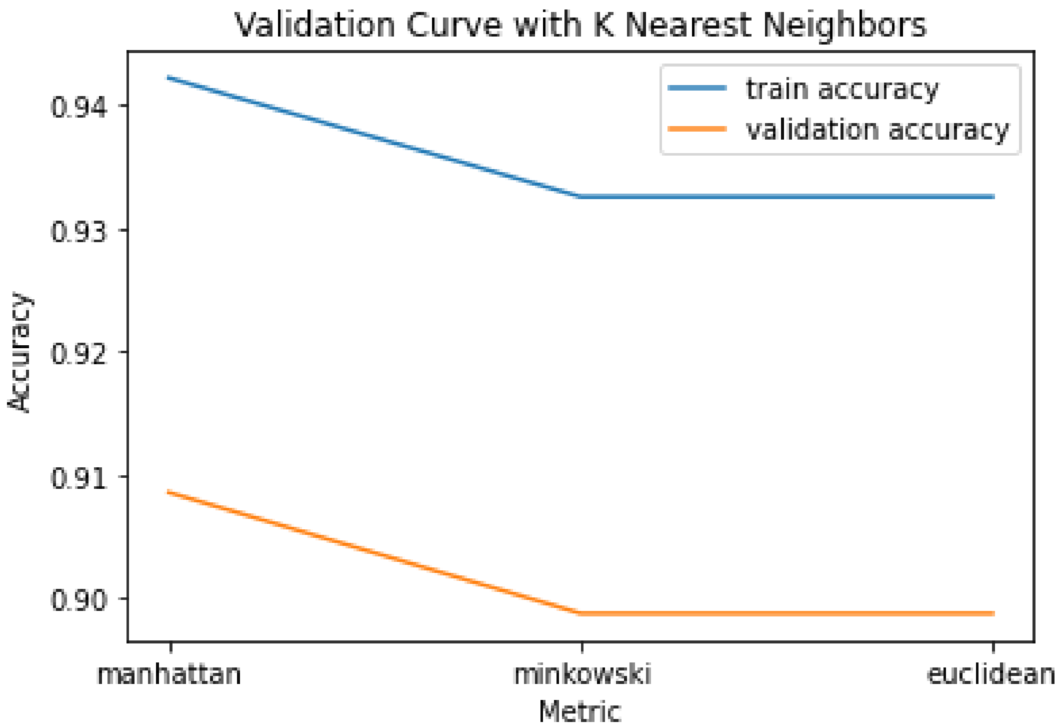
```
featuresMIC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G      1Run      BB_P       H_P       R/G
SV  \
0  0.173913  0.44  0.477612  0.559387  0.429180  0.622353  0.339394
0.302094
1  0.637681  0.32  0.000000  0.390805  0.154861  0.181001  0.133333
0.194577
2  0.231884  0.56  0.300995  0.641762  0.429180  0.464920  0.363636
0.265422
3  0.275362  0.36  0.161692  0.618774  0.688073  0.159225  0.000000
0.100396
4  0.594203  0.46  0.208955  0.392720  0.204570  0.482468  0.442424
0.390013

   Under500  #a-tA-S        HR      Rtot       SOS      HR_P        BB
\
0  0.402703     0.45  0.380797  0.391473  0.545455  0.418914  0.465585

1  0.389189     0.65  0.213898  0.841085  0.454545  0.123964  0.410213
```

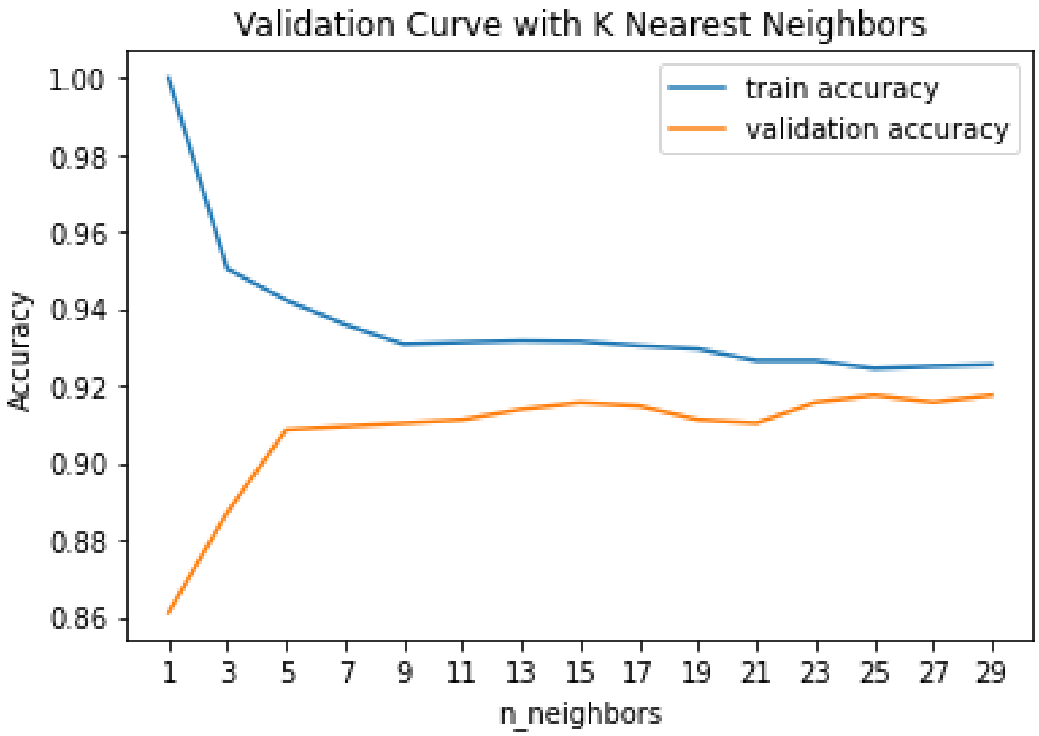
```
2  0.385135     0.75  0.303797  0.372093  0.454545  0.189573  0.440077

3  0.271622     0.40  0.126699  0.569767  0.454545  0.142855  0.021753

4  0.528378     0.60  0.335165  0.441860  0.363636  0.233241  0.540521


     DefEff       IBB     IBB_P        PA    BatAge
0  0.602041  0.288005  0.495551  0.594126  0.292135
1  1.000000  0.380978  0.471563  0.356231  0.460674
2  0.581633  0.322774  0.441491  0.523644  0.483146
3  0.867347  0.343635  0.477531  0.279588  0.505618
4  0.663265  0.520958  0.501326  0.566620  0.606742
```

```
# A stratified train-test split is performed on the dataset, with 80%
of the data assigned to training set and 20% assigned to testing set.

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

# Parameters for KNN algorithm

KNNmetrics = ['manhattan','minkowski','euclidean']
n_neighbors = np.arange(1,30,2)

# 10-fold cross validation is performed on the training set for
parameter tuning.
# The manhattan, minkowski, and euclidean distance metrics are
assessed for accuracy

train_scores, valid_scores = validation_curve(KNeighborsClassifier(),
X_train, y_train, param_name="metric", param_range=KNNmetrics,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("Metric")
plt.ylabel("Accuracy")
plt.plot(KNNmetrics, train_scores_mean, label="train accuracy")
plt.plot(KNNmetrics, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

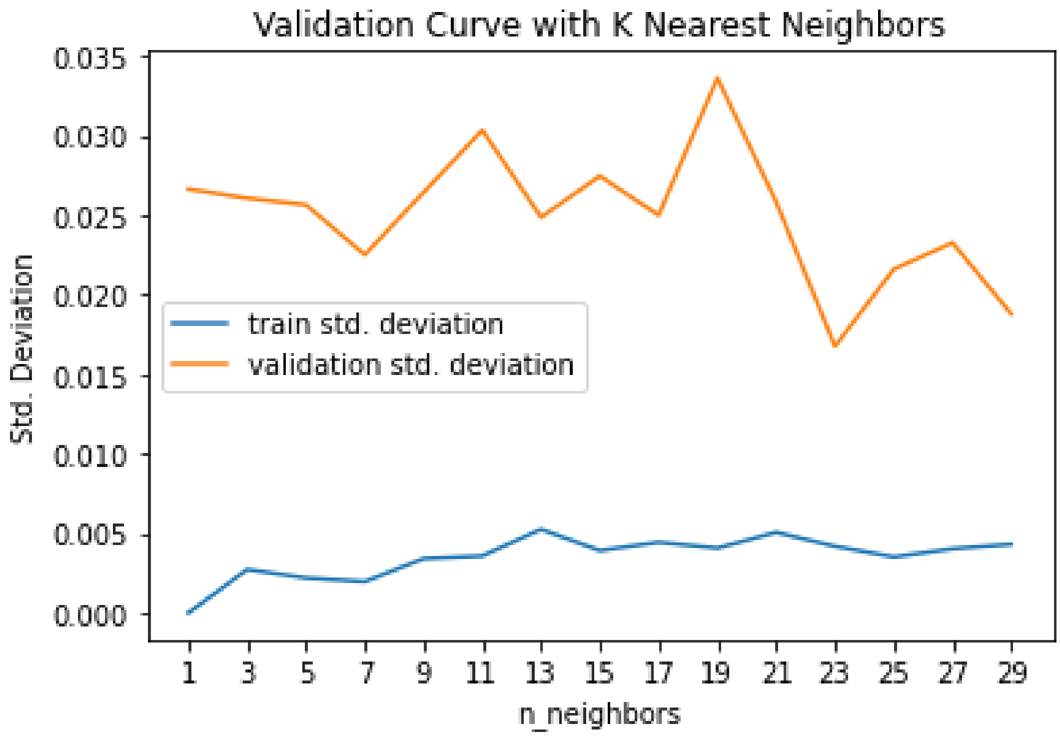
```
[0.90859878 0.89877735 0.89877735]
```



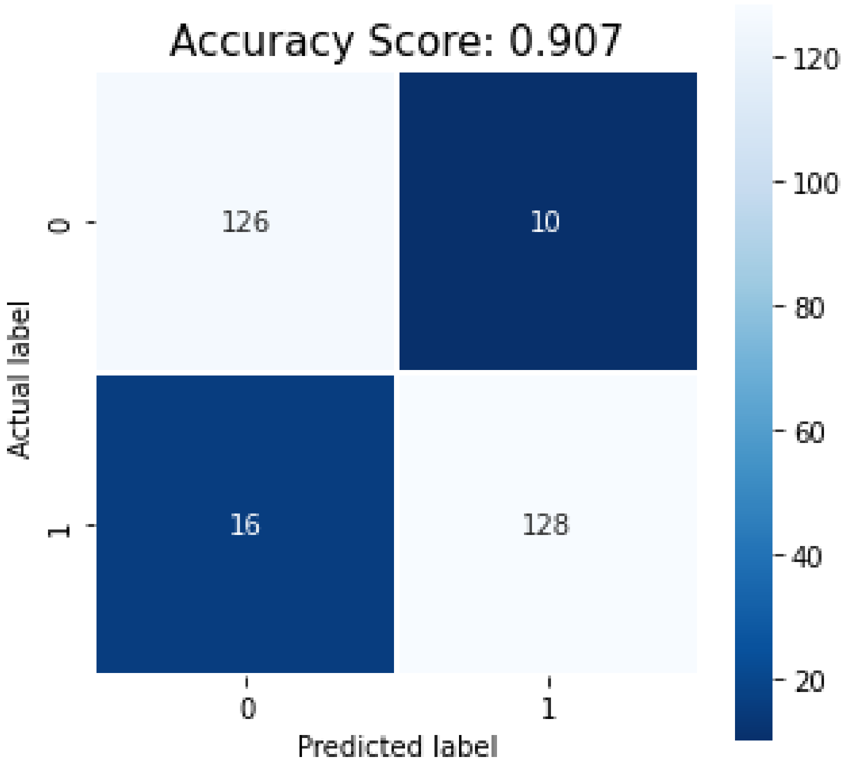

```
# Manhattan distance metric appears to perform best on the training
set and validation set.
# The n_neighbors parameter is tested next for model accuracy.

train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Accuracy")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_mean, label="train accuracy")
plt.plot(n_neighbors, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()

[0.86112452 0.88715412 0.90859878 0.90949968 0.91036036 0.91122909
 0.91393983 0.91570142 0.9148166  0.91123713 0.91036036 0.91574968
 0.91755148 0.91574968 0.91755952]
```




```
# The standard deviation is evaluated for each value of K

train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_std, label="train std. deviation")
plt.plot(n_neighbors, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()

[0.02662817 0.02605362 0.02564029 0.02250552 0.02641716 0.03031819
 0.02486319 0.02744341 0.0249871  0.03360341 0.02581774 0.01674618
 0.02159398 0.02325362 0.01878032]
```




```
KNN = KNeighborsClassifier(n_neighbors=25, metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  128
FP:  10
TN:  126
FN:  16

              precision    recall  f1-score   support

           0       0.89      0.93      0.91       136
           1       0.93      0.89      0.91       144

    accuracy                           0.91       280
   macro avg       0.91      0.91      0.91       280
weighted avg       0.91      0.91      0.91       280
```

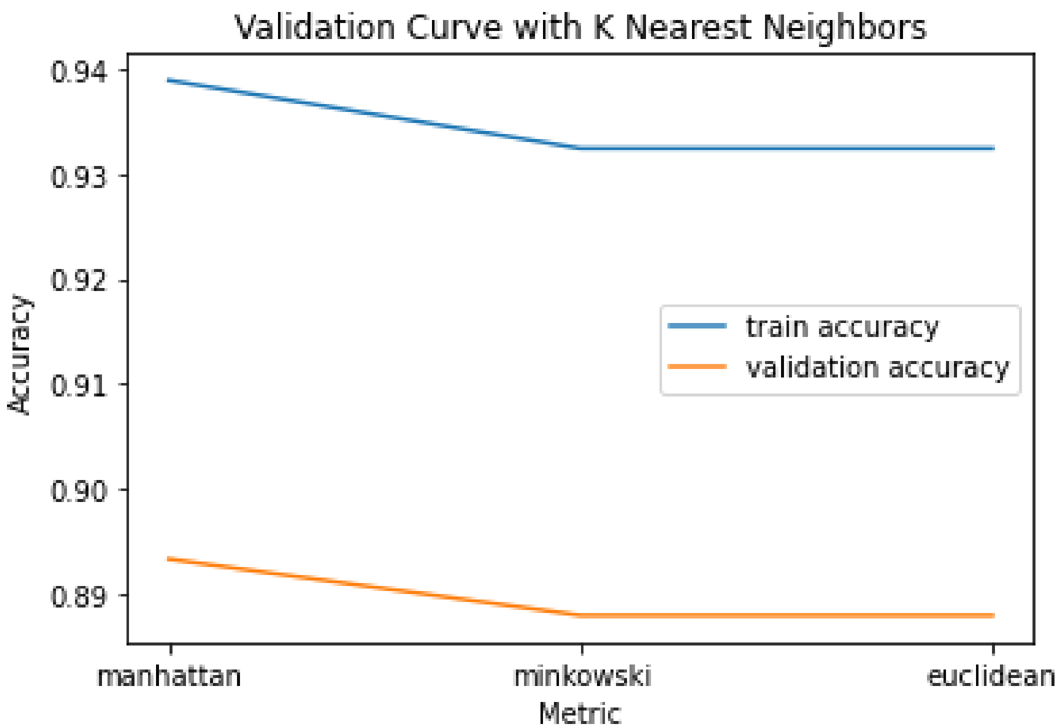
ANOVA F-test

```
# The above process is now repeated for the top 20 features as
selected by the ANOVA F-test selection algorithm.
```

```
X = featuresFC.iloc[:,0:20]
y = featuresFC['WinningRecord']

featuresFC.iloc[:,0:20].head()

       ERA+      RA/G  OPS+      1Run       H_P        SV      BB_P
#a-tA-S  \
0  0.173913  0.477612  0.44  0.559387  0.622353  0.302094  0.429180
0.45
1  0.637681  0.000000  0.32  0.390805  0.181001  0.194577  0.154861
0.65
2  0.231884  0.300995  0.56  0.641762  0.464920  0.265422  0.429180
0.75
3  0.275362  0.161692  0.36  0.618774  0.159225  0.100396  0.688073
0.40
4  0.594203  0.208955  0.46  0.392720  0.482468  0.390013  0.204570
0.60

        R/G    DefEff      Rtot  Under500        IP        BB
PAge  \
0  0.339394  0.602041  0.391473  0.402703  0.678982  0.465585
0.405941
1  0.133333  1.000000  0.841085  0.389189  0.697641  0.410213
0.465347
2  0.363636  0.581633  0.372093  0.385135  0.710241  0.440077
0.475248
3  0.000000  0.867347  0.569767  0.271622  0.680184  0.021753
0.336634
4  0.442424  0.663265  0.441860  0.528378  0.752545  0.540521
0.455446

         PA       IBB    BatAge      HR_P        SF
0  0.594126  0.288005  0.292135  0.418914  0.358281
1  0.356231  0.380978  0.460674  0.123964  0.398932
2  0.523644  0.322774  0.483146  0.189573  0.538171
3  0.279588  0.343635  0.505618  0.142855  0.124424
4  0.566620  0.520958  0.606742  0.233241  0.443268

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores = validation_curve(KNeighborsClassifier(),
X_train, y_train, param_name="metric", param_range=KNNmetrics,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("Metric")
plt.ylabel("Accuracy")
```

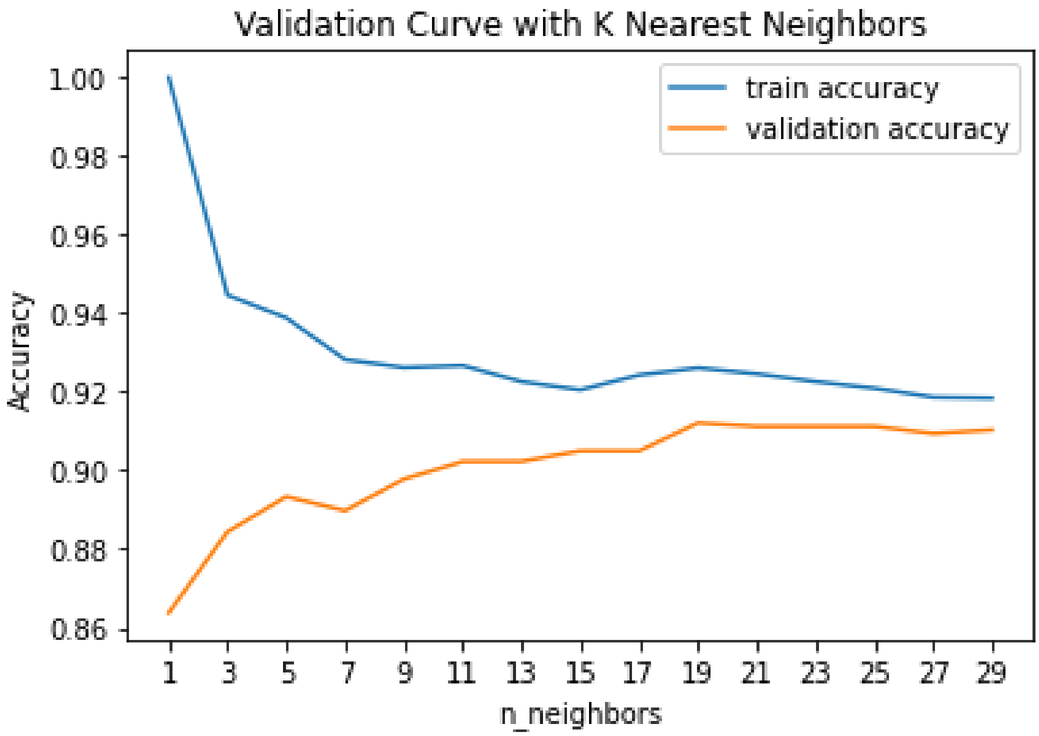
```
plt.plot(KNNmetrics, train_scores_mean, label="train accuracy")
plt.plot(KNNmetrics, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.8933639 0.8880148 0.8880148]
```



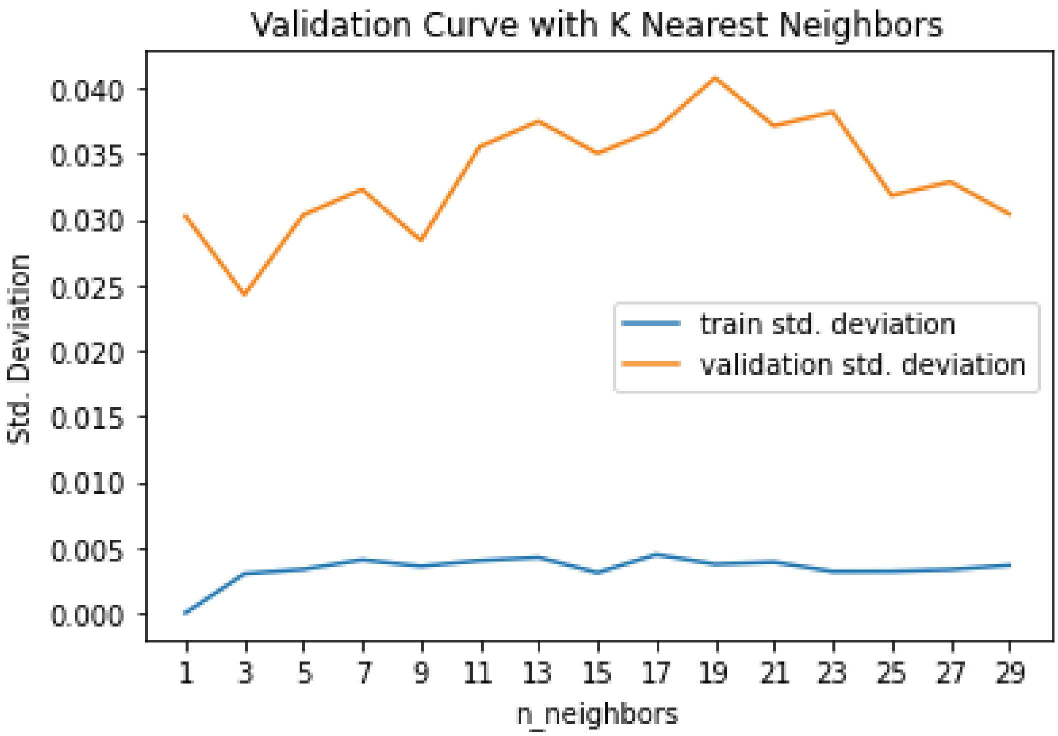

```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Accuracy")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_mean, label="train accuracy")
plt.plot(n_neighbors, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.86379505 0.88441924 0.8933639  0.88980051 0.89785232 0.90230051
 0.90232465 0.90500322 0.90499517 0.91213803 0.91124517 0.91125322
 0.91124517 0.90944337 0.91034427]
```

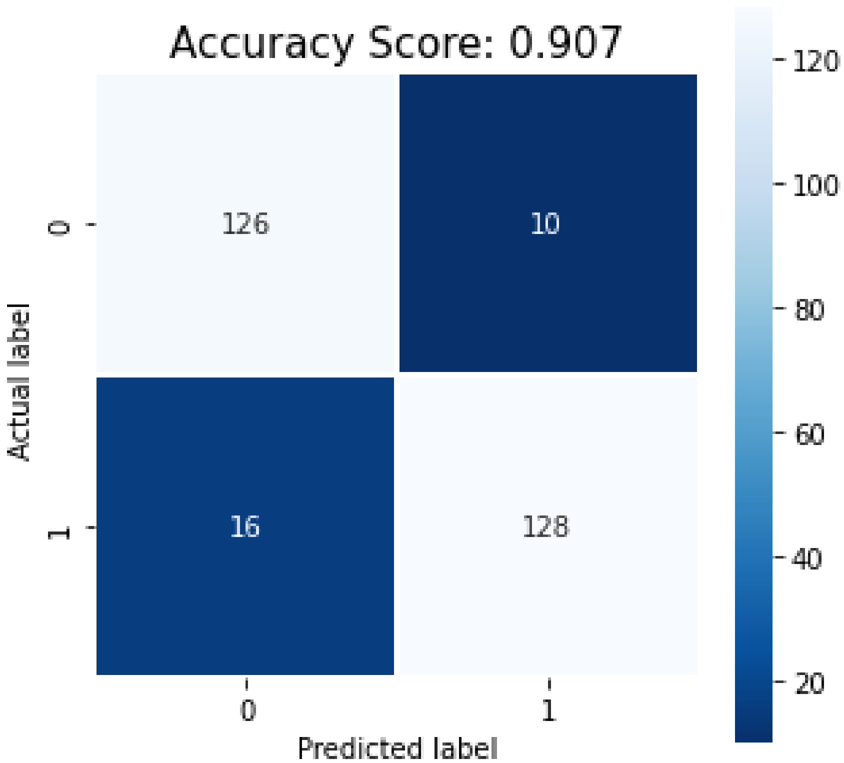



```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_std, label="train std. deviation")
plt.plot(n_neighbors, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.03024712 0.0242586  0.03034205 0.03227753 0.02840147 0.03554162
 0.03748231 0.03503088 0.03686598 0.04077803 0.03713405 0.03818651
 0.03182461 0.03286589 0.03041107]
```




```
KNN = KNeighborsClassifier(n_neighbors=19, metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  128
FP:  10
TN:  126
FN:  16

              precision    recall  f1-score   support

           0       0.89      0.93      0.91       136
           1       0.93      0.89      0.91       144

    accuracy                           0.91       280
   macro avg       0.91      0.91      0.91       280
weighted avg       0.91      0.91      0.91       280
```

Random Forest

```
# The above process is now repeated for the top 20 features as
selected by the Random Forest selection algorithm.
```

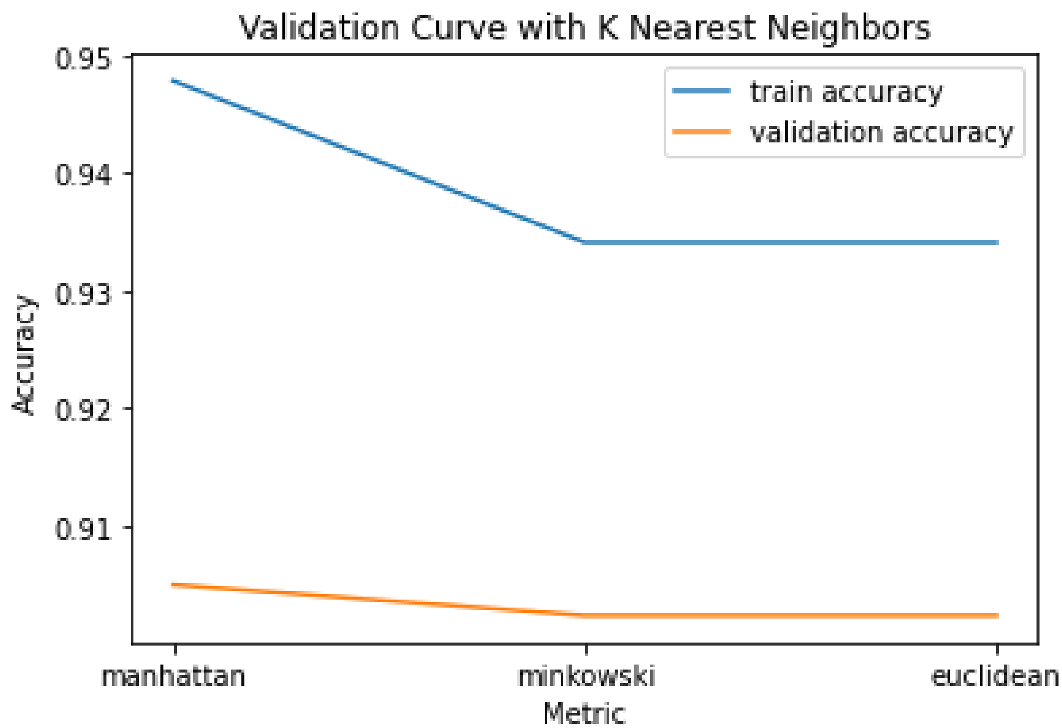
```
X = featuresRFC.iloc[:,0:20]
y = featuresRFC['WinningRecord']

featuresRFC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G       R/G      1Run        SV       H_P
BB_P  \
0  0.173913  0.44  0.477612  0.339394  0.559387  0.302094  0.622353
0.429180
1  0.637681  0.32  0.000000  0.133333  0.390805  0.194577  0.181001
0.154861
2  0.231884  0.56  0.300995  0.363636  0.641762  0.265422  0.464920
0.429180
3  0.275362  0.36  0.161692  0.000000  0.618774  0.100396  0.159225
0.688073
4  0.594203  0.46  0.208955  0.442424  0.392720  0.390013  0.482468
0.204570

   #a-tA-S  Under500    DefEff      Rtot        PA        IP        BB
\
0     0.45  0.402703  0.602041  0.391473  0.594126  0.678982  0.465585

1     0.65  0.389189  1.000000  0.841085  0.356231  0.697641  0.410213

2     0.75  0.385135  0.581633  0.372093  0.523644  0.710241  0.440077

3     0.40  0.271622  0.867347  0.569767  0.279588  0.680184  0.021753

4     0.60  0.528378  0.663265  0.441860  0.566620  0.752545  0.540521


       PAge      HR_P        HR    BatAge       IBB
0  0.405941  0.418914  0.380797  0.292135  0.288005
1  0.465347  0.123964  0.213898  0.460674  0.380978
2  0.475248  0.189573  0.303797  0.483146  0.322774
3  0.336634  0.142855  0.126699  0.505618  0.343635
4  0.455446  0.233241  0.335165  0.606742  0.520958
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores = validation_curve(KNeighborsClassifier(),
X_train, y_train, param_name="metric", param_range=KNNmetrics,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("Metric")
```

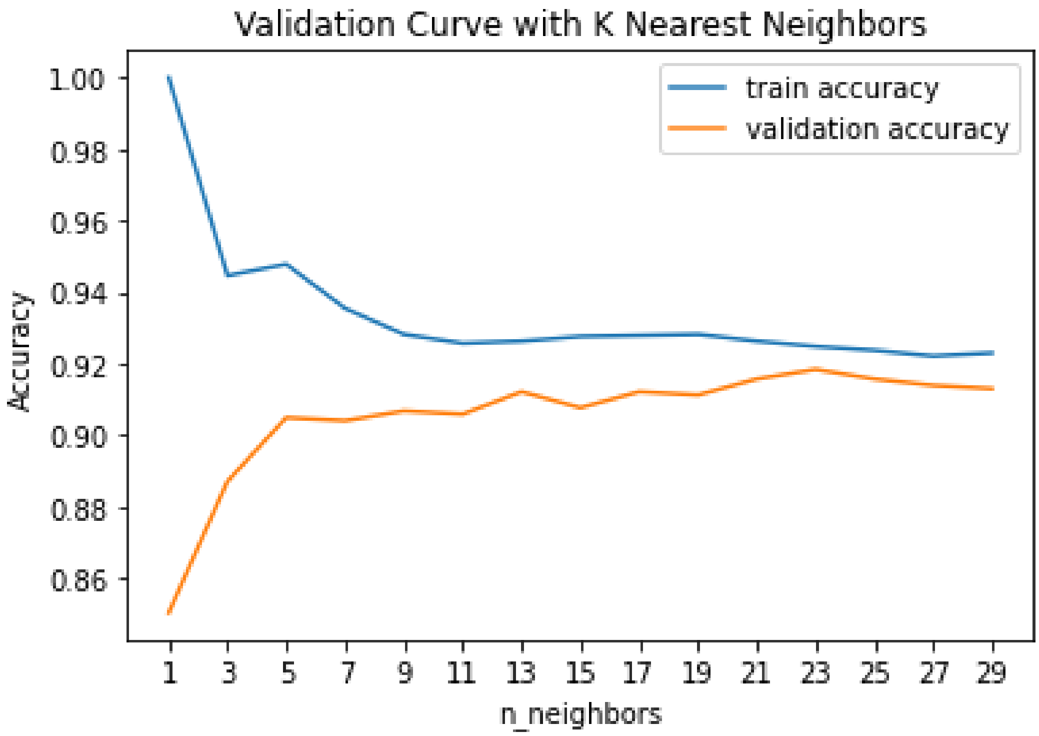
```
plt.ylabel("Accuracy")
plt.plot(KNNmetrics, train_scores_mean, label="train accuracy")
plt.plot(KNNmetrics, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.90497909 0.90237291 0.90237291]
```



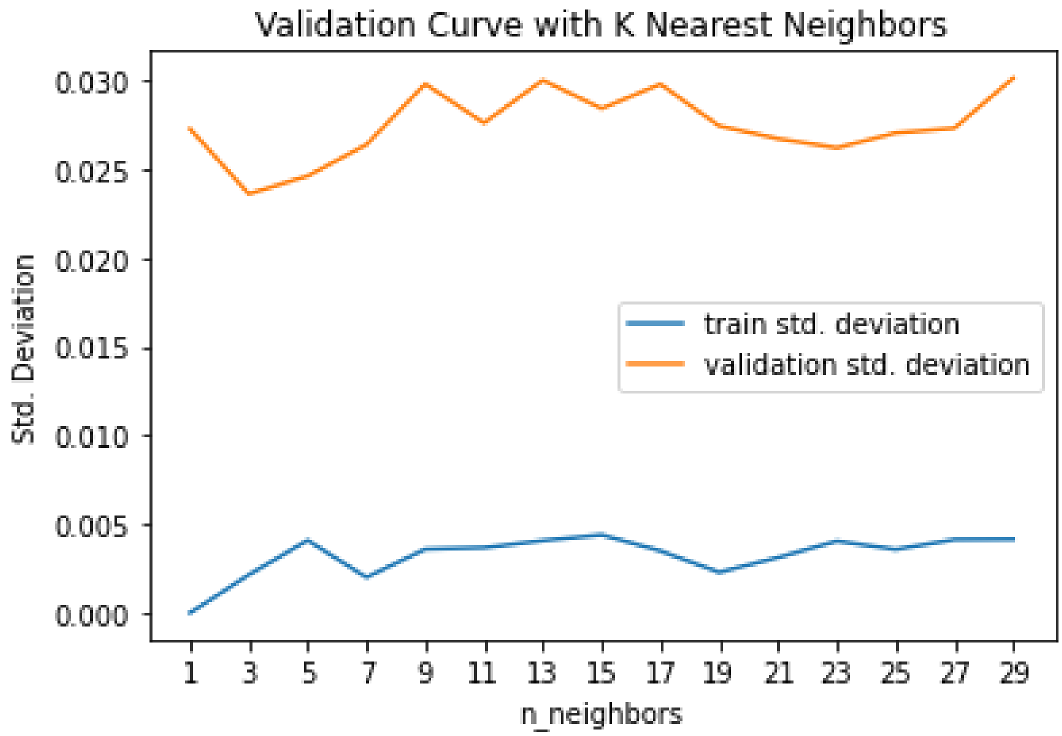

```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Accuracy")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_mean, label="train accuracy")
plt.plot(n_neighbors, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.85036197 0.88712194 0.90497909 0.90409427 0.90679698 0.90590412
 0.91216216 0.90769788 0.91216216 0.91125322 0.91574968 0.91843629
 0.91574163 0.91395592 0.91303893]
```

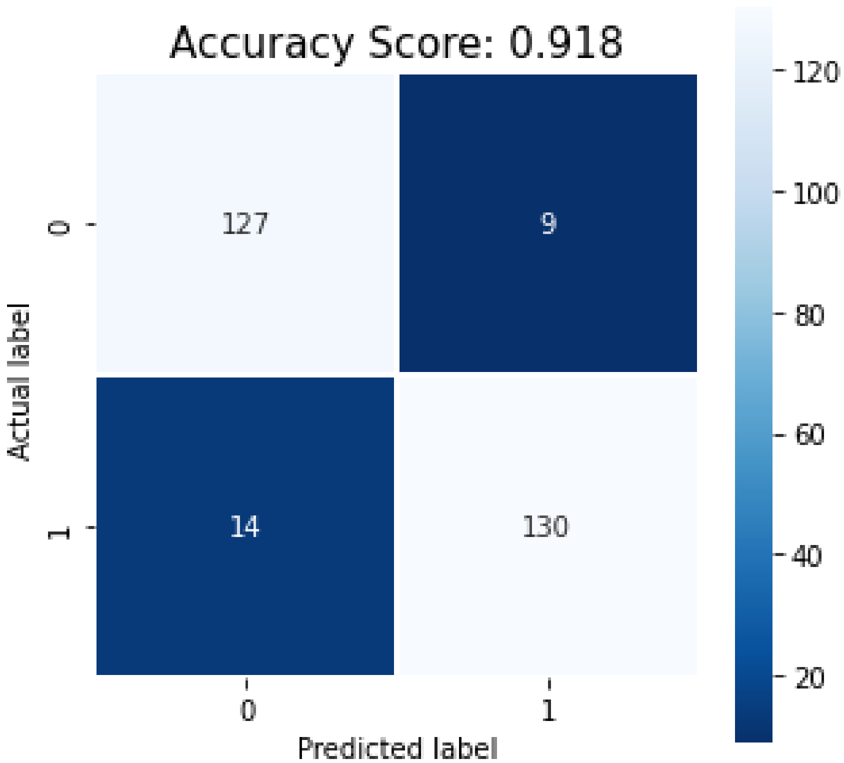



```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_std, label="train std. deviation")
plt.plot(n_neighbors, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.02726307 0.02360006 0.02460746 0.026372   0.02978538 0.02759581
 0.02999964 0.02840803 0.02977636 0.02742374 0.02670632 0.02618986
 0.02702883 0.02730062 0.03012542]
```




```
KNN = KNeighborsClassifier(n_neighbors=23, metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  130
FP:  9
TN:  127
FN:  14

              precision    recall  f1-score   support

           0       0.90      0.93      0.92       136
           1       0.94      0.90      0.92       144

    accuracy                           0.92       280
   macro avg       0.92      0.92      0.92       280
weighted avg       0.92      0.92      0.92       280
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_val_score,
validation_curve, StratifiedShuffleSplit
from sklearn import metrics

# The three datasets created during the feature selection process are
uploaded.
# Each dataset contains a different ordering of the feature variables
based on their importance.

featuresMIC = pd.read_csv('featuresRankedMIC.csv')
featuresFC = pd.read_csv('featuresRankedFC.csv')
featuresRFC = pd.read_csv('featuresRankedRFC.csv')
```

Mutual Information Gain

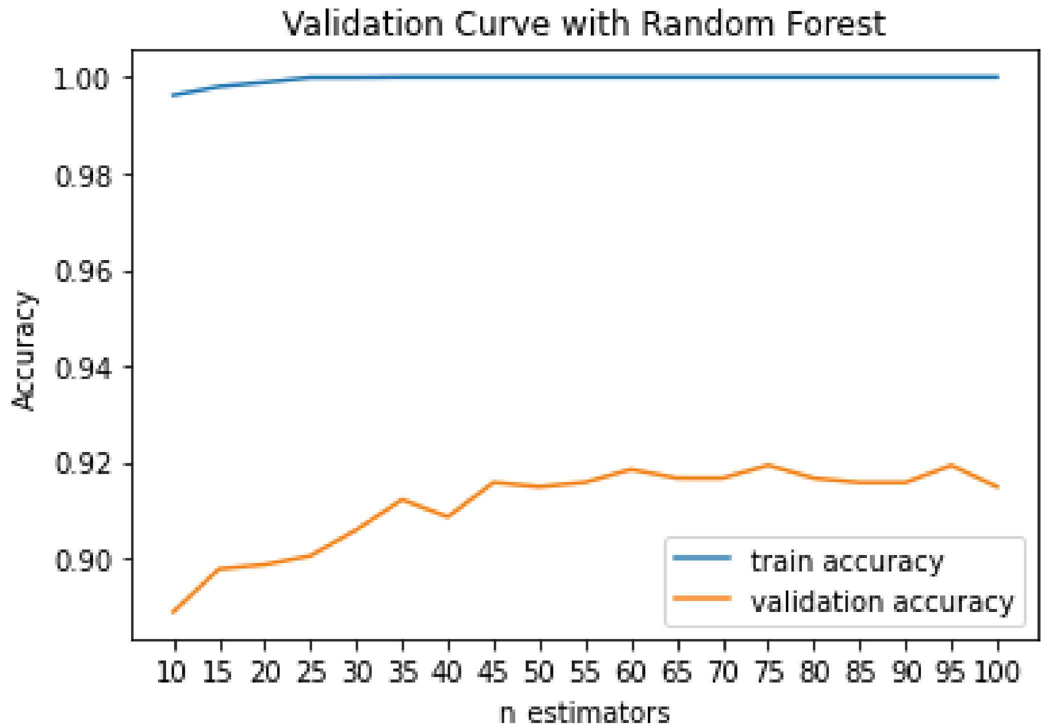
```
# The top 20 features will be selected from each dataset for machine
learning.

X = featuresMIC.iloc[:,0:20]
y = featuresMIC['WinningRecord']

featuresMIC.iloc[:,0:20].head()

       ERA+  OPS+      RA/G      1Run      BB_P       H_P       R/G
SV  \
0  0.173913  0.44  0.477612  0.559387  0.429180  0.622353  0.339394
0.302094
1  0.637681  0.32  0.000000  0.390805  0.154861  0.181001  0.133333
0.194577
2  0.231884  0.56  0.300995  0.641762  0.429180  0.464920  0.363636
0.265422
3  0.275362  0.36  0.161692  0.618774  0.688073  0.159225  0.000000
0.100396
4  0.594203  0.46  0.208955  0.392720  0.204570  0.482468  0.442424
0.390013

   Under500  #a-tA-S        HR      Rtot       SOS      HR_P        BB
\
0  0.402703     0.45  0.380797  0.391473  0.545455  0.418914  0.465585

1  0.389189     0.65  0.213898  0.841085  0.454545  0.123964  0.410213
```

```
2  0.385135     0.75  0.303797  0.372093  0.454545  0.189573  0.440077

3  0.271622     0.40  0.126699  0.569767  0.454545  0.142855  0.021753

4  0.528378     0.60  0.335165  0.441860  0.363636  0.233241  0.540521


     DefEff       IBB     IBB_P        PA    BatAge
0  0.602041  0.288005  0.495551  0.594126  0.292135
1  1.000000  0.380978  0.471563  0.356231  0.460674
2  0.581633  0.322774  0.441491  0.523644  0.483146
3  0.867347  0.343635  0.477531  0.279588  0.505618
4  0.663265  0.520958  0.501326  0.566620  0.606742
```

```
# A stratified train-test split is performed on the dataset, with 80%
of the data assigned to training set and 20% assigned to testing set.

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

# Parameter for Random Forest algorithm

n_estimators = np.arange(10,105,5)

# 10-fold cross validation is performed on the training set for
parameter tuning.
# The n_estimators parameter is assessed for accuracy

train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_mean, label="train accuracy")
plt.plot(n_estimators, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

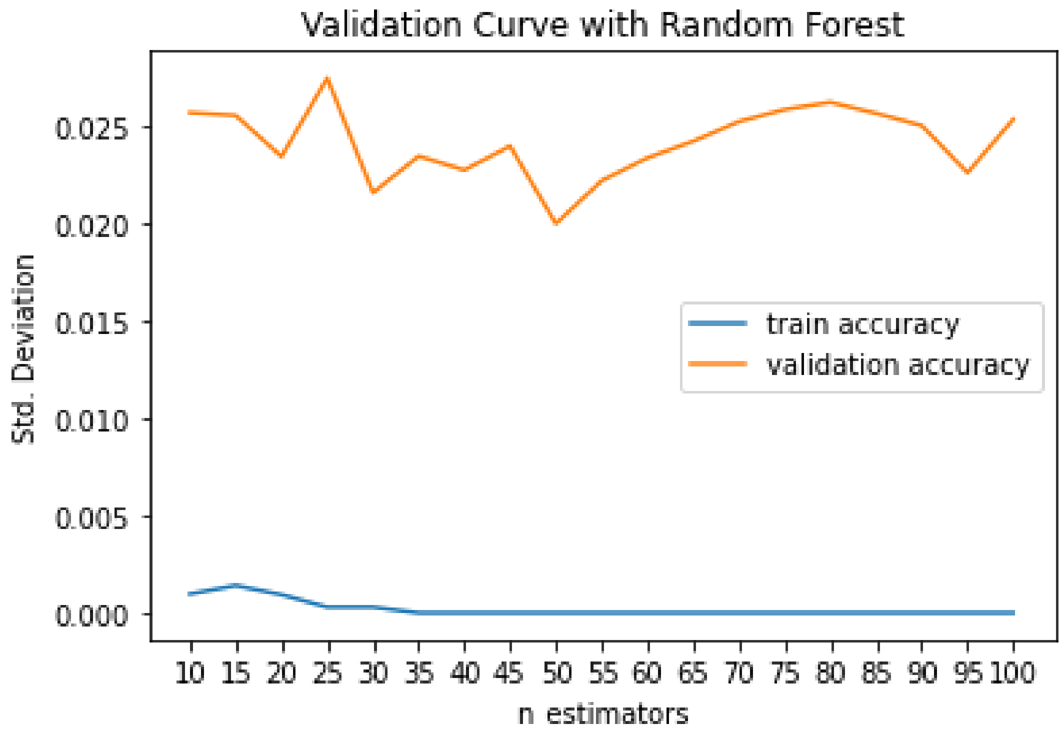
```
[0.88889157 0.89782014 0.89872104 0.90056306 0.90592021 0.91221847
 0.908639   0.9157899  0.91489704 0.91580598 0.91848456 0.91669884
```

```
 0.91668275 0.91936937 0.91666667 0.91576577 0.91576577 0.91935328
 0.91487291]
```

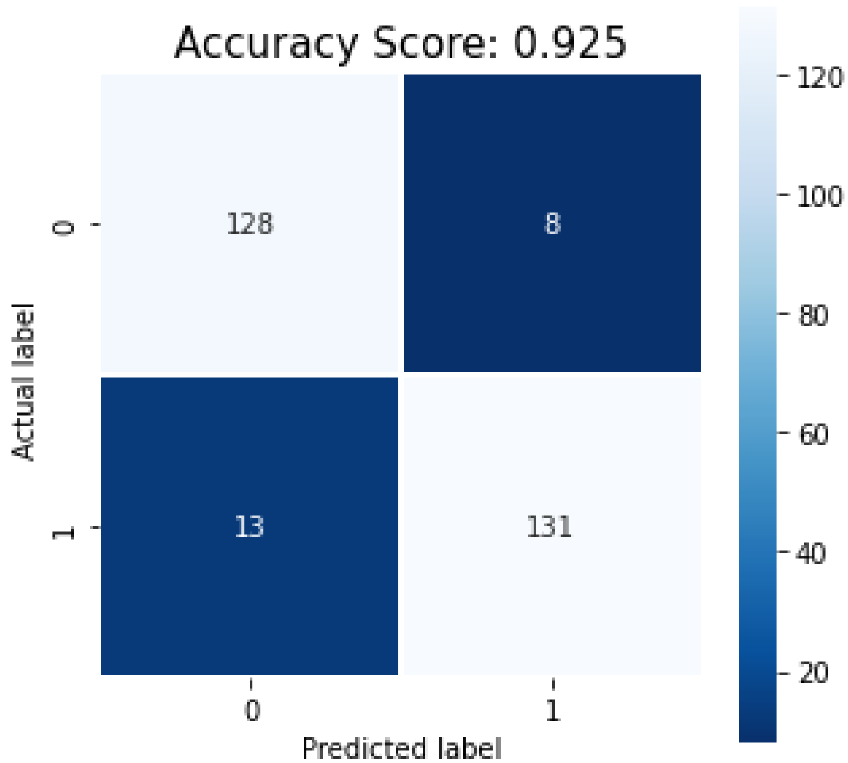



```
# The standard deviation is evaluated for each value of n_estimators

train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_std, label="train accuracy")
plt.plot(n_estimators, valid_scores_std, label="validation accuracy")

plt.legend()
plt.show()
```

```
[0.02568338 0.02554333 0.02342658 0.02746108 0.0215742  0.02344771
 0.02274134 0.02396698 0.01998319 0.02219235 0.02336108 0.02421252
 0.02522505 0.02584637 0.02621498 0.02565164 0.02502237 0.02259687
 0.02534392]
```




```
RF = RandomForestClassifier(random_state=1, n_estimators=75)
RF.fit(X_train, y_train)
y_pred = RF.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  131
FP:  8
TN:  128
FN:  13

              precision    recall  f1-score   support

           0       0.91      0.94      0.92       136
           1       0.94      0.91      0.93       144

    accuracy                           0.93       280
   macro avg       0.93      0.93      0.92       280
weighted avg       0.93      0.93      0.93       280
```

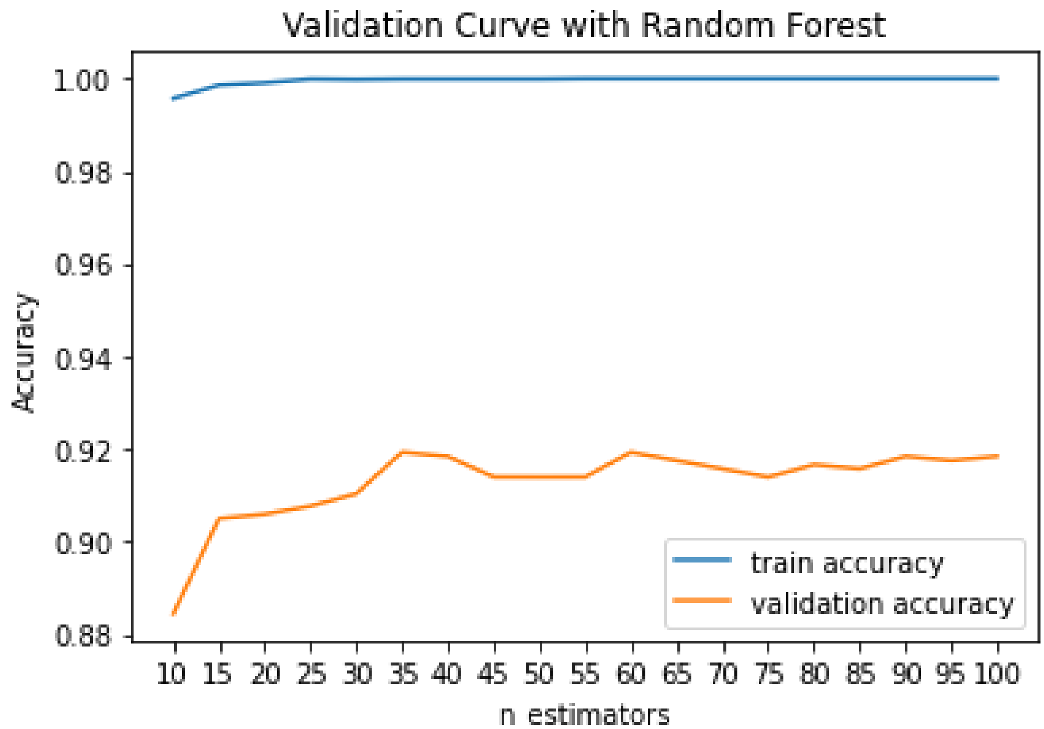
ANOVA F-test

```
# The above process is now repeated for the top 20 features as
selected by the ANOVA F-test selection algorithm.
```

```
X = featuresFC.iloc[:,0:20]
y = featuresFC['WinningRecord']

featuresFC.iloc[:,0:20].head()

       ERA+      RA/G  OPS+      1Run       H_P        SV      BB_P
#a-tA-S  \
0  0.173913  0.477612  0.44  0.559387  0.622353  0.302094  0.429180
0.45
1  0.637681  0.000000  0.32  0.390805  0.181001  0.194577  0.154861
0.65
2  0.231884  0.300995  0.56  0.641762  0.464920  0.265422  0.429180
0.75
3  0.275362  0.161692  0.36  0.618774  0.159225  0.100396  0.688073
0.40
4  0.594203  0.208955  0.46  0.392720  0.482468  0.390013  0.204570
0.60

        R/G    DefEff      Rtot  Under500        IP        BB
PAge  \
0  0.339394  0.602041  0.391473  0.402703  0.678982  0.465585
0.405941
1  0.133333  1.000000  0.841085  0.389189  0.697641  0.410213
0.465347
2  0.363636  0.581633  0.372093  0.385135  0.710241  0.440077
0.475248
3  0.000000  0.867347  0.569767  0.271622  0.680184  0.021753
0.336634
4  0.442424  0.663265  0.441860  0.528378  0.752545  0.540521
0.455446

         PA       IBB    BatAge      HR_P        SF
0  0.594126  0.288005  0.292135  0.418914  0.358281
1  0.356231  0.380978  0.460674  0.123964  0.398932
2  0.523644  0.322774  0.483146  0.189573  0.538171
3  0.279588  0.343635  0.505618  0.142855  0.124424
4  0.566620  0.520958  0.606742  0.233241  0.443268

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
```

```
plt.ylabel("Accuracy")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_mean, label="train accuracy")
plt.plot(n_estimators, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

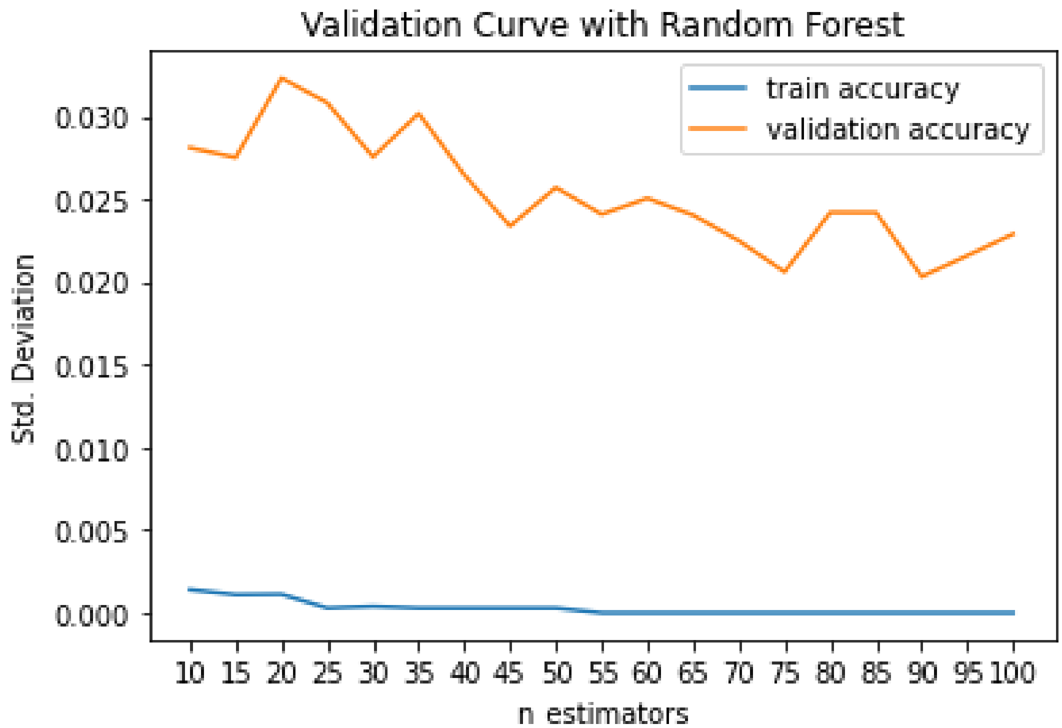
```
[0.8844112  0.90500322 0.90592021 0.90768983 0.91035232 0.91934524
 0.91842021 0.91397201 0.91395592 0.91396396 0.91932111 0.91752735
 0.91573359 0.91394788 0.9166184  0.91574163 0.91842825 0.91753539
 0.91842825]
```




```
train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_std, label="train accuracy")
plt.plot(n_estimators, valid_scores_std, label="validation accuracy")
```

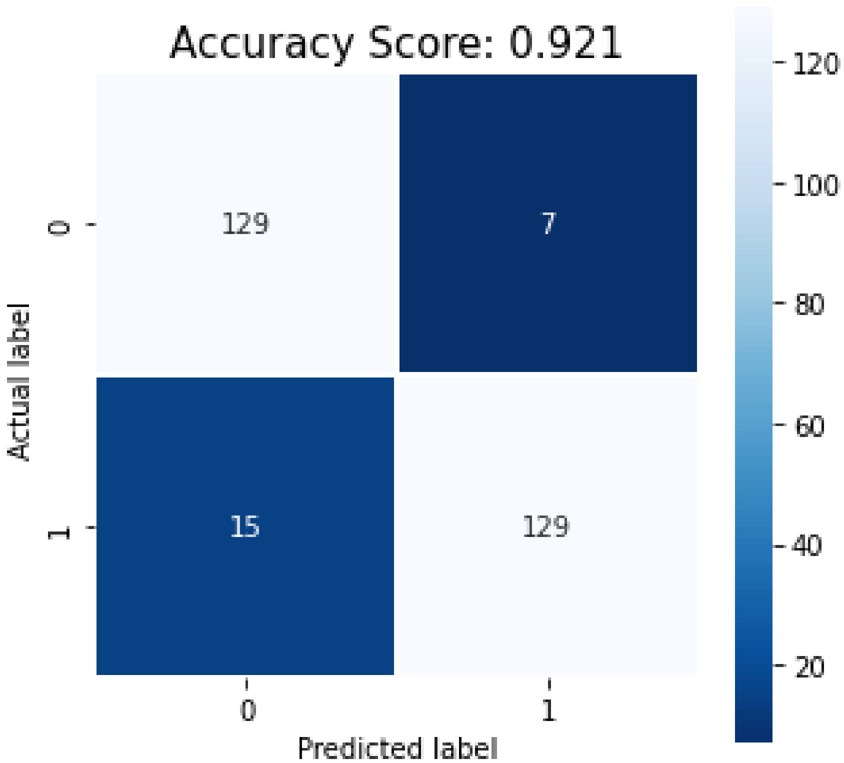
```
plt.legend()
plt.show()
```

```
[0.02811198 0.02752933 0.03232951 0.03083201 0.02756387 0.03018191
 0.02646474 0.02338087 0.02571528 0.02409331 0.02505994 0.02404395
 0.02249738 0.02061353 0.02420231 0.02419362 0.0203324  0.02159936
 0.02291318]
```




```
RF = RandomForestClassifier(random_state=1, n_estimators=35)
RF.fit(X_train, y_train)
y_pred = RF.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  129
FP:  7
TN:  129
FN:  15

              precision    recall  f1-score   support

           0       0.90      0.95      0.92       136
           1       0.95      0.90      0.92       144

    accuracy                           0.92       280
   macro avg       0.92      0.92      0.92       280
weighted avg       0.92      0.92      0.92       280
```

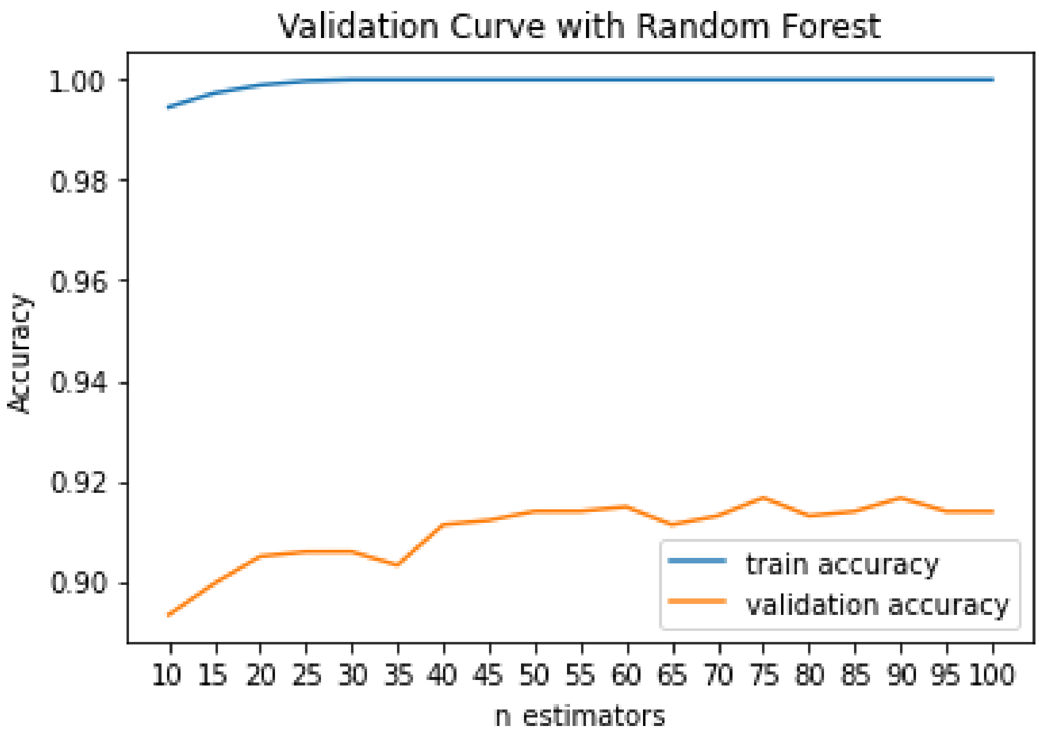
Random Forest

```
# The above process is now repeated for the top 20 features as
selected by the Random Forest selection algorithm.
```

```
X = featuresRFC.iloc[:,0:20]
y = featuresRFC['WinningRecord']

featuresRFC.iloc[:,0:20].head()

       ERA+  OPS+      RA/G       R/G      1Run        SV       H_P
BB_P  \
0  0.173913  0.44  0.477612  0.339394  0.559387  0.302094  0.622353
0.429180
1  0.637681  0.32  0.000000  0.133333  0.390805  0.194577  0.181001
0.154861
2  0.231884  0.56  0.300995  0.363636  0.641762  0.265422  0.464920
0.429180
3  0.275362  0.36  0.161692  0.000000  0.618774  0.100396  0.159225
0.688073
4  0.594203  0.46  0.208955  0.442424  0.392720  0.390013  0.482468
0.204570

   #a-tA-S  Under500    DefEff      Rtot        PA        IP        BB
\
0     0.45  0.402703  0.602041  0.391473  0.594126  0.678982  0.465585

1     0.65  0.389189  1.000000  0.841085  0.356231  0.697641  0.410213

2     0.75  0.385135  0.581633  0.372093  0.523644  0.710241  0.440077

3     0.40  0.271622  0.867347  0.569767  0.279588  0.680184  0.021753

4     0.60  0.528378  0.663265  0.441860  0.566620  0.752545  0.540521


       PAge      HR_P        HR    BatAge       IBB
0  0.405941  0.418914  0.380797  0.292135  0.288005
1  0.465347  0.123964  0.213898  0.460674  0.380978
2  0.475248  0.189573  0.303797  0.483146  0.322774
3  0.336634  0.142855  0.126699  0.505618  0.343635
4  0.455446  0.233241  0.335165  0.606742  0.520958

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='accuracy', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Random Forest")
```

```
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_mean, label="train accuracy")
plt.plot(n_estimators, valid_scores_mean, label="validation accuracy")

plt.legend()
plt.show()
```

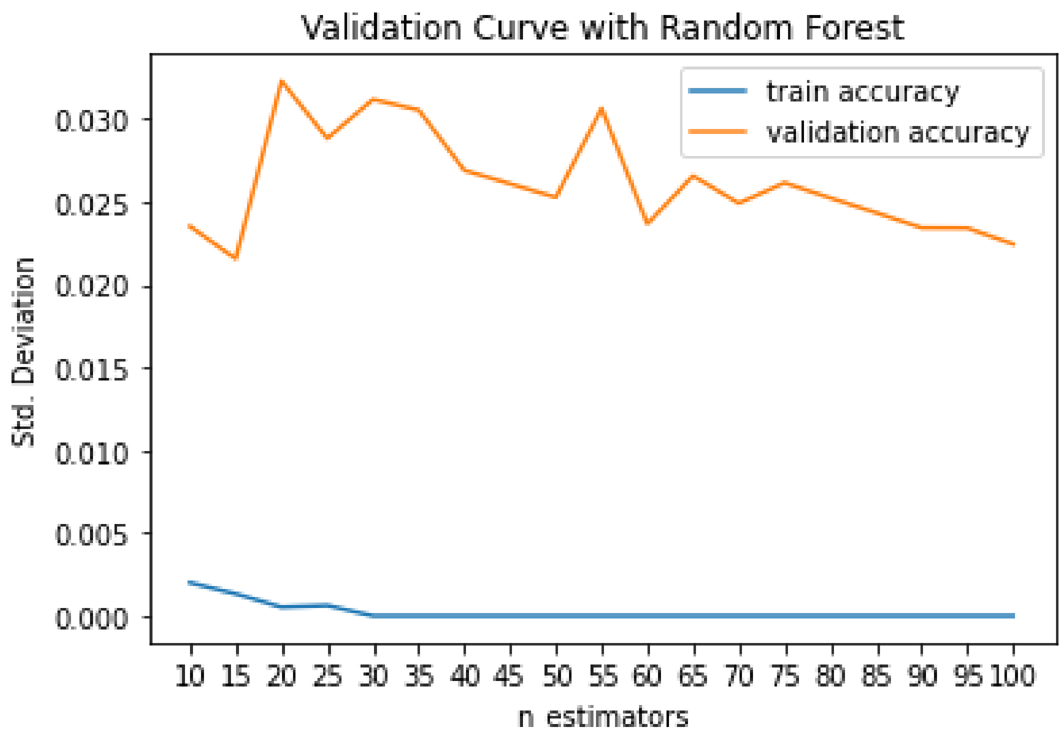
```
[0.89335586 0.89962194 0.90501126 0.90592825 0.90594434 0.90325772
 0.91130148 0.91220238 0.91399614 0.91401223 0.914889   0.91131757
 0.91309524 0.91668275 0.91309524 0.9139881  0.91665058 0.91397201
 0.91395592]
```




```
train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='accuracy', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_std, label="train accuracy")
plt.plot(n_estimators, valid_scores_std, label="validation accuracy")
```

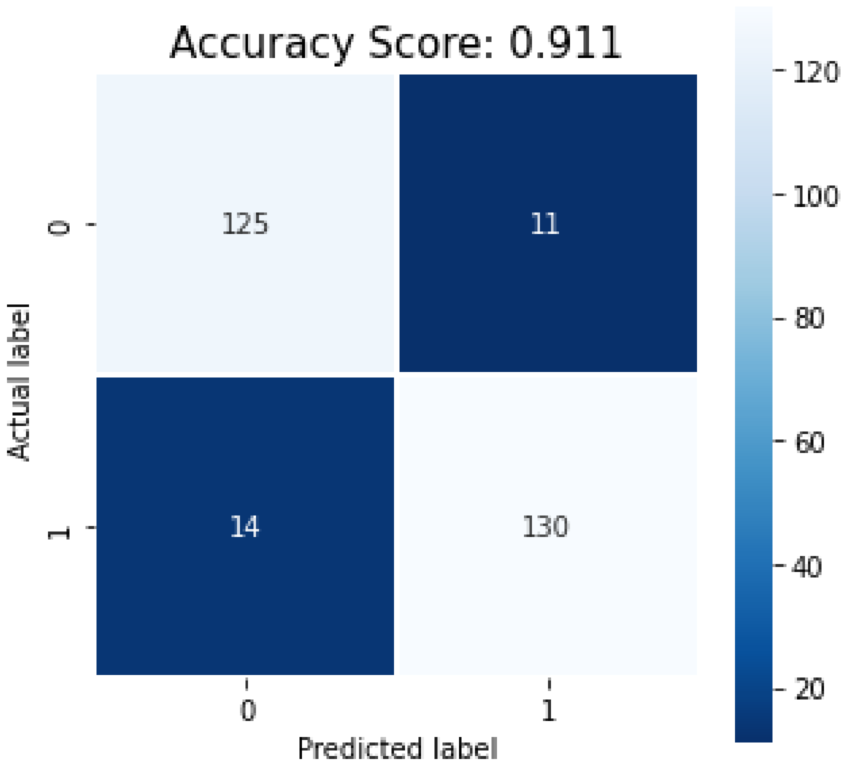
```
plt.legend()
plt.show()
```

```
[0.02352688 0.02156941 0.03230096 0.02883582 0.03118562 0.03055023
 0.02690548 0.02609349 0.02528085 0.03064393 0.02367945 0.02657437
 0.02492058 0.02615597 0.02523844 0.02435028 0.02343349 0.02341787
 0.02245318]
```




```
RF = RandomForestClassifier(random_state=1, n_estimators=90)
RF.fit(X_train, y_train)
y_pred = RF.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
```

```
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score:
{0}'.format(round(accuracy_score(y_test, y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  130
FP:  11
TN:  125
FN:  14

              precision    recall  f1-score   support

           0       0.90      0.92      0.91       136
           1       0.92      0.90      0.91       144

    accuracy                           0.91       280
   macro avg       0.91      0.91      0.91       280
weighted avg       0.91      0.91      0.91       280
```

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

data = pd.read_csv('mlbdata.csv')

mlbdata = data.iloc[:, :44]
mlbdata.head()

   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...      CG_F         Ch         E        DP
Rtot  \
0  0.303226  0.225806  ...  7.380645  38.193548  1.006452  0.838710
-38
1  0.506494  0.266234  ...  7.155844  38.090909  0.649351  0.974026
78
2  0.425806  0.193548  ...  7.580645  37.967742  0.838710  0.909677
-43
3  0.367742  0.238710  ...  7.245161  37.761290  0.735484  0.870968
8
4  0.442308  0.301282  ...  7.410256  39.852564  0.846154  0.948718
-25

   BPF  #a-tA-S   1Run  Under500  SOS
0  109       13  0.536     0.458  0.1
1  103       17  0.448     0.448  0.0
2  106       19  0.579     0.445  0.0
3   94       12  0.567     0.361  0.0
4  110       16  0.449     0.551 -0.1

[5 rows x 44 columns]

mlbdatanorm = pd.DataFrame(mlbdata)
mlbdatanormal = (mlbdatanorm - mlbdatanorm.min()) /
(mlbdatanorm.max()-mlbdatanorm.min())
mlbdatanormal.insert(44,'Playoffs',data['W-L%'])
```

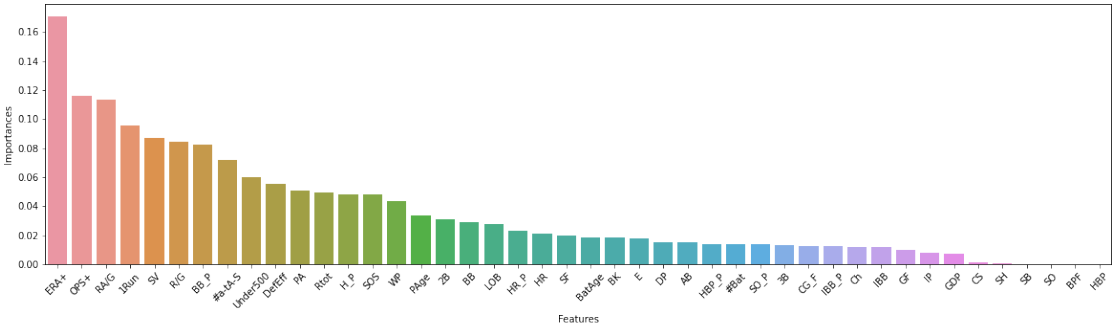
```
mlbdatanormal['Playoffs'] = ((mlbdatanormal['Playoffs'] >=
.550).replace({True: 1, False: 0}))

X = mlbdatanormal.iloc[:,0:-1]
y = mlbdatanormal['Playoffs']

from sklearn.model_selection import train_test_split
from collections import defaultdict
from operator import itemgetter
from sklearn.feature_selection import SelectKBest, mutual_info_classif
mic = SelectKBest(score_func=mutual_info_classif)

dicts = defaultdict(list)
finallist = []
for num in range(250):
  X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
  fit = mic.fit(X_train,y_train)
  dfscores = pd.DataFrame(fit.scores_)
  dfcolumns = pd.DataFrame(X_train.columns)
  featureScores = pd.concat([dfcolumns,dfscores], axis=1)
  featureScores.columns = ['Feature','Score']
  keys = featureScores.index
  values = featureScores.loc[:,'Score']
  for i in keys:
    dicts[i].append(values[i])
for k, v in (dicts.items()):
  total = np.sum(v)
  np.sort(total, axis=None)
  final = (k, total/250)
  finallist.append(final)

finaldf = pd.DataFrame(finallist, columns=['Feature', 'Score'])

allFeatures = featureScores.nlargest(44, 'Score')
allFeatures
```

	Feature	Score
32	ERA+	0.170771
12	OPS+	0.115986
20	RA/G	0.113105
41	1Run	0.095422
22	SV	0.087282
4	R/G	0.084341
26	BB_P	0.082570
40	#a-tA-S	0.071890
42	Under500	0.059844
33	DefEff	0.055560
2	PA	0.051054
38	Rtot	0.049331
24	H_P	0.048038

```
43        SOS  0.047781
31         WP  0.043736
19       PAge  0.033731
5          2B  0.031237
10         BB  0.028754
18        LOB  0.027597
25       HR_P  0.023199
7          HR  0.020799
16         SF  0.019505
1      BatAge  0.018357
30         BK  0.018239
36          E  0.017913
37         DP  0.015339
3          AB  0.014884
29      HBP_P  0.014061
0        #Bat  0.013830
28       SO_P  0.013732
6          3B  0.013407
34       CG_F  0.012620
27      IBB_P  0.012349
35         Ch  0.011923
17        IBB  0.011785
21         GF  0.009966
23         IP  0.007661
13        GDP  0.007316
9          CS  0.001448
15         SH  0.000547
8          SB  0.000000
11         SO  0.000000
39        BPF  0.000000
14        HBP  0.000000
```

```
importances = allFeatures['Score']
final_df2 = pd.DataFrame({'Features': allFeatures['Feature'],
'Importances':importances})
final_df2.set_index('Importances')

final_df2 = final_df2.sort_values('Importances', ascending=False)
plt.figure(figsize=(20,5))
plt.xticks(rotation=45)
sns.barplot(x='Features',y='Importances', data=final_df2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f37e58a9390>
```




```
featuresRanked = []
for i in allFeatures['Feature'].head(44):
  featuresRanked.append(i)

featuresRankedMIC = mlbdatanormal[featuresRanked + ['Playoffs']]

featuresRankedMIC.to_csv("featuresRankedMIC_550.csv")
```

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

data = pd.read_csv('mlbdata.csv')

mlbdata = data.iloc[:, :44]
mlbdata.head()

   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...      CG_F         Ch         E        DP
Rtot  \
0  0.303226  0.225806  ...  7.380645  38.193548  1.006452  0.838710
-38
1  0.506494  0.266234  ...  7.155844  38.090909  0.649351  0.974026
78
2  0.425806  0.193548  ...  7.580645  37.967742  0.838710  0.909677
-43
3  0.367742  0.238710  ...  7.245161  37.761290  0.735484  0.870968
8
4  0.442308  0.301282  ...  7.410256  39.852564  0.846154  0.948718
-25

   BPF  #a-tA-S   1Run  Under500  SOS
0  109       13  0.536     0.458  0.1
1  103       17  0.448     0.448  0.0
2  106       19  0.579     0.445  0.0
3   94       12  0.567     0.361  0.0
4  110       16  0.449     0.551 -0.1

[5 rows x 44 columns]

mlbdatanorm = pd.DataFrame(mlbdata)
mlbdatanormal = (mlbdatanorm - mlbdatanorm.min()) /
(mlbdatanorm.max()-mlbdatanorm.min())
mlbdatanormal.insert(44,'Playoffs',data['W-L%'])
```

```
mlbdatanormal['Playoffs'] = ((mlbdatanormal['Playoffs'] >=
.550).replace({True: 1, False: 0}))

mlbdatanormal.head()

    #Bat    BatAge        PA        AB       R/G        2B
3B  \
0  0.175  0.292135  0.594126  0.701810  0.339394  0.151402  0.172537

1  0.075  0.460674  0.356231  0.445843  0.133333  0.191710  0.344682

2  0.225  0.483146  0.523644  0.619373  0.363636  0.361411  0.412642

3  0.275  0.505618  0.279588  0.568733  0.000000  0.078143  0.299651

4  0.300  0.606742  0.566620  0.625692  0.442424  0.242631  0.493763


         HR        SB        CS  ...        Ch         E        DP
Rtot  \
0  0.380797  0.096689  0.246236  ...  0.621987  0.743973  0.384946
0.391473
1  0.213898  0.197865  0.302859  ...  0.609655  0.349282  0.565368
0.841085
2  0.303797  0.157703  0.201055  ...  0.594857  0.558574  0.479570
0.372093
3  0.126699  0.128802  0.264308  ...  0.570052  0.444482  0.427957
0.569767
4  0.335165  0.165917  0.351948  ...  0.821312  0.566802  0.531624
0.441860

        BPF  #a-tA-S      1Run  Under500       SOS  Playoffs
0  0.547619     0.45  0.559387  0.402703  0.545455         0
1  0.404762     0.65  0.390805  0.389189  0.454545         0
2  0.476190     0.75  0.641762  0.385135  0.454545         0
3  0.190476     0.40  0.618774  0.271622  0.454545         0
4  0.571429     0.60  0.392720  0.528378  0.363636         0

[5 rows x 45 columns]

X = mlbdatanormal.iloc[:,0:-1]
y = mlbdatanormal['Playoffs']

from sklearn.model_selection import train_test_split
from collections import defaultdict
from sklearn.feature_selection import SelectKBest, f_classif
fc = SelectKBest(score_func=f_classif)

dicts = defaultdict(list)
finallist = []
```

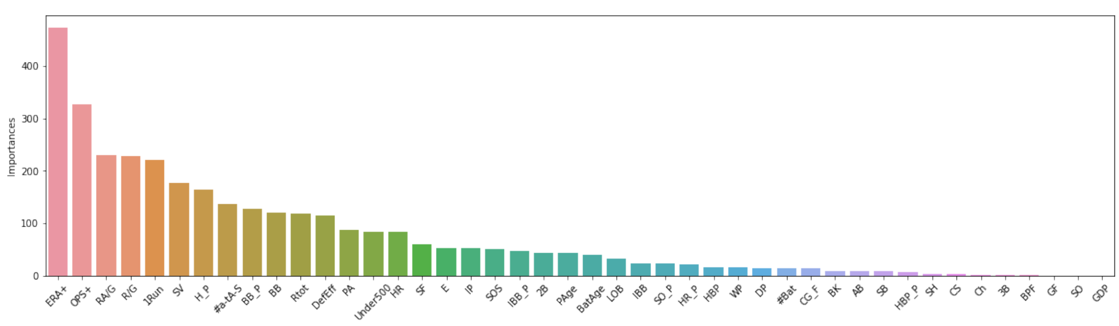
```
for num in range(250):
  X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
  fit = fc.fit(X_train,y_train)
  dfscores = pd.DataFrame(fit.scores_)
  dfcolumns = pd.DataFrame(X_train.columns)
  featureScores = pd.concat([dfcolumns,dfscores], axis=1)
  featureScores.columns = ['Feature','Score']
  keys = featureScores.index
  values = featureScores.loc[:,'Score']
  for i in keys:
    dicts[i].append(values[i])
for k, v in (dicts.items()):
  total = np.sum(v)
  np.sort(total, axis=None)
  final = (k, total/250)
  finallist.append(final)

finaldf = pd.DataFrame(finallist, columns=['Feature', 'Score'])

finaldf.sort_values(by='Score', ascending=False)
allFeatures = featureScores.nlargest(44, 'Score')
allFeatures
```

	Feature	Score
32	ERA+	472.954539
12	OPS+	326.844826
20	RA/G	230.438742
4	R/G	228.183527
41	1Run	220.664916
22	SV	177.572195
24	H_P	163.818450
40	#a-tA-S	136.858087
26	BB_P	128.074300
10	BB	119.678667
38	Rtot	118.772238
33	DefEff	114.019467
2	PA	87.320818
42	Under500	84.304304
7	HR	83.766432
16	SF	60.575249
36	E	52.889044
23	IP	52.203812
43	SOS	50.974763
27	IBB_P	47.879555
5	2B	43.214604
19	PAge	43.039486
1	BatAge	39.266941
18	LOB	33.517892
17	IBB	24.195693
28	SO_P	23.492747

```
25      HR_P   21.689726
14       HBP   16.049431
31        WP   15.724145
37        DP   15.081872
0       #Bat   14.782763
34      CG_F   13.944058
30        BK    9.832544
3         AB    8.270676
8         SB    8.084329
29     HBP_P    7.567585
15        SH    3.066885
9         CS    2.915477
35        Ch    1.318963
6         3B    1.043293
39       BPF    0.898795
21        GF    0.584501
11        SO    0.160604
13       GDP    0.155572
```

```
importances = allFeatures['Score']
final_df2 = pd.DataFrame({'Features': allFeatures['Feature'],
'Importances':importances})
final_df2.set_index('Importances')

final_df2 = final_df2.sort_values('Importances', ascending=False)
plt.figure(figsize=(20,5))
plt.xticks(rotation=45)
sns.barplot(x='Features',y='Importances', data=final_df2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b7399bad0>
```




```
featuresRanked = []
for i in allFeatures['Feature'].head(44):
  featuresRanked.append(i)

featuresRankedFC = mlbdatanormal[featuresRanked + ['Playoffs']]

featuresRankedFC.to_csv("featuresRankedFC_550.csv")
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statistics
import seaborn as sns
from scipy import stats
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

data = pd.read_csv('mlbdata.csv')

mlbdata = data.iloc[:, :44]
mlbdata.head()

   #Bat  BatAge         PA         AB   R/G        2B        3B
HR  \
0    36    27.2  38.316129  34.051613  4.05  1.200000  0.109677
0.929032
1    32    28.7  36.857143  32.649351  3.37  1.253247  0.188312
0.649351
2    38    28.9  37.883871  33.600000  4.13  1.477419  0.219355
0.800000
3    40    29.1  36.387097  33.322581  2.93  1.103226  0.167742
0.503226
4    41    30.0  38.147436  33.634615  4.39  1.320513  0.256410
0.852564

         SB        CS  ...      CG_F         Ch         E        DP
Rtot  \
0  0.303226  0.225806  ...  7.380645  38.193548  1.006452  0.838710
-38
1  0.506494  0.266234  ...  7.155844  38.090909  0.649351  0.974026
78
2  0.425806  0.193548  ...  7.580645  37.967742  0.838710  0.909677
-43
3  0.367742  0.238710  ...  7.245161  37.761290  0.735484  0.870968
8
4  0.442308  0.301282  ...  7.410256  39.852564  0.846154  0.948718
-25

   BPF  #a-tA-S   1Run  Under500  SOS
0  109       13  0.536     0.458  0.1
1  103       17  0.448     0.448  0.0
2  106       19  0.579     0.445  0.0
3   94       12  0.567     0.361  0.0
4  110       16  0.449     0.551 -0.1

[5 rows x 44 columns]

mlbdatanorm = pd.DataFrame(mlbdata)
mlbdatanormal = (mlbdatanorm - mlbdatanorm.min()) /
```

```
(mlbdatanorm.max()-mlbdatanorm.min())
mlbdatanormal.insert(44,'Playoffs',data['W-L%'])

mlbdatanormal['Playoffs'] = ((mlbdatanormal['Playoffs'] >=
.550).replace({True: 1, False: 0}))

mlbdatanormal.head()

    #Bat    BatAge        PA        AB       R/G        2B
3B  \
0  0.175  0.292135  0.594126  0.701810  0.339394  0.151402  0.172537

1  0.075  0.460674  0.356231  0.445843  0.133333  0.191710  0.344682

2  0.225  0.483146  0.523644  0.619373  0.363636  0.361411  0.412642

3  0.275  0.505618  0.279588  0.568733  0.000000  0.078143  0.299651

4  0.300  0.606742  0.566620  0.625692  0.442424  0.242631  0.493763


         HR        SB        CS  ...        Ch         E        DP
Rtot  \
0  0.380797  0.096689  0.246236  ...  0.621987  0.743973  0.384946
0.391473
1  0.213898  0.197865  0.302859  ...  0.609655  0.349282  0.565368
0.841085
2  0.303797  0.157703  0.201055  ...  0.594857  0.558574  0.479570
0.372093
3  0.126699  0.128802  0.264308  ...  0.570052  0.444482  0.427957
0.569767
4  0.335165  0.165917  0.351948  ...  0.821312  0.566802  0.531624
0.441860

        BPF  #a-tA-S      1Run  Under500       SOS  Playoffs
0  0.547619     0.45  0.559387  0.402703  0.545455         0
1  0.404762     0.65  0.390805  0.389189  0.454545         0
2  0.476190     0.75  0.641762  0.385135  0.454545         0
3  0.190476     0.40  0.618774  0.271622  0.454545         0
4  0.571429     0.60  0.392720  0.528378  0.363636         0

[5 rows x 45 columns]

X = mlbdatanormal.iloc[:,0:-1]
y = mlbdatanormal['Playoffs']

rf = RandomForestClassifier()
from collections import defaultdict

dicts = defaultdict(list)
finallist = []
```

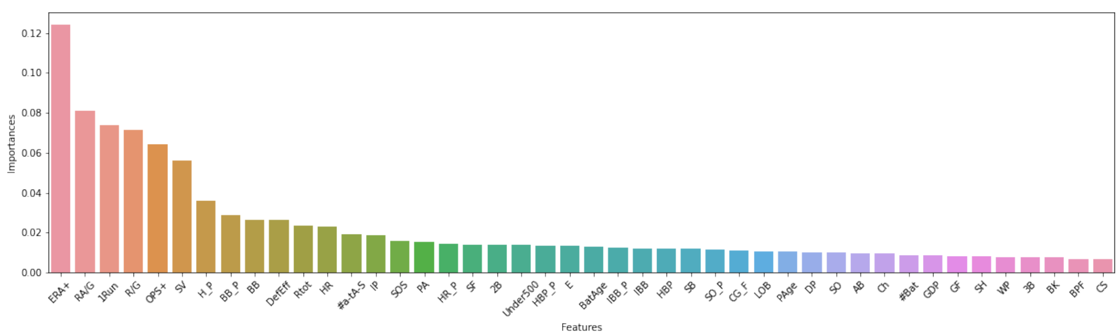
```python
for num in range(250):
  X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
  fit = rf.fit(X_train,y_train)
  dfscores = pd.DataFrame(fit.feature_importances_)
  dfcolumns = pd.DataFrame(X_train.columns)
  featureScores = pd.concat([dfcolumns,dfscores], axis=1)
  featureScores.columns = ['Feature','Score']
  keys = featureScores.index
  values = featureScores.loc[:,'Score']
  for i in keys:
    dicts[i].append(values[i])
for k, v in (dicts.items()):
  total = np.sum(v)
  np.sort(total, axis=None)
  final = (k, total/250)
  finallist.append(final)

finaldf = pd.DataFrame(finallist, columns=['Feature', 'Score'])

finaldf.sort_values(by='Score', ascending=False)
allFeatures = featureScores.nlargest(44, 'Score')
allFeatures
```

	Feature	Score
32	ERA+	0.124151
20	RA/G	0.080989
41	1Run	0.073811
4	R/G	0.071273
12	OPS+	0.064025
22	SV	0.056176
24	H_P	0.035833
26	BB_P	0.028806
10	BB	0.026463
33	DefEff	0.026167
38	Rtot	0.023553
7	HR	0.022916
40	#a-tA-S	0.019344
23	IP	0.018664
43	SOS	0.015872
2	PA	0.015487
25	HR_P	0.014138
16	SF	0.013871
5	2B	0.013828
42	Under500	0.013645
29	HBP_P	0.013556
36	E	0.013545
1	BatAge	0.012920
27	IBB_P	0.012239
17	IBB	0.012147
14	HBP	0.011998

```
8         SB  0.011720
28      SO_P  0.011228
34      CG_F  0.011212
18       LOB  0.010671
19      PAge  0.010303
37        DP  0.010101
11        SO  0.009845
3         AB  0.009600
35        Ch  0.009389
0       #Bat  0.008769
13       GDP  0.008744
21        GF  0.008310
15        SH  0.008296
31        WP  0.007801
6         3B  0.007734
30        BK  0.007458
39       BPF  0.006706
9         CS  0.006695
```

```
importances = allFeatures['Score']
final_df2 = pd.DataFrame({'Features': allFeatures['Feature'],
'Importances':importances})
final_df2.set_index('Importances')

final_df2 = final_df2.sort_values('Importances', ascending=False)
plt.figure(figsize=(20,5))
plt.xticks(rotation=45)
sns.barplot(x='Features',y='Importances', data=final_df2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb414b5d910>
```




```
featuresRanked = []
for i in allFeatures['Feature'].head(44):
  featuresRanked.append(i)

featuresRankedRFC = mlbdatanormal[featuresRanked + ['Playoffs']]

featuresRankedRFC.to_csv("featuresRankedRFC_550.csv")
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import xticks, yticks
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

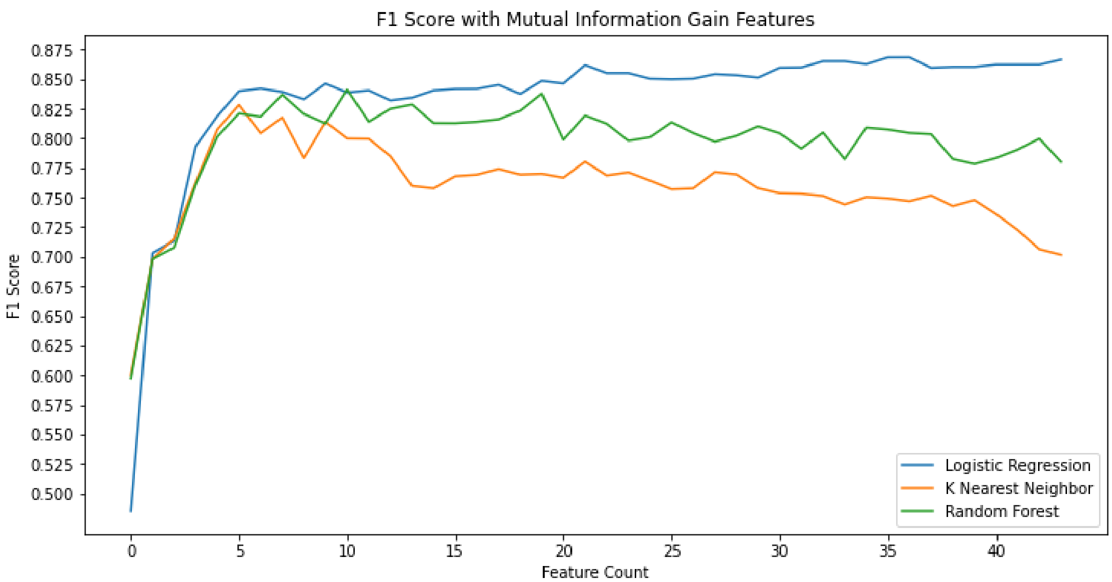
```
featuresMIC = pd.read_csv('featuresRankedMIC_550.csv')
featuresFC = pd.read_csv('featuresRankedFC_550.csv')
featuresRFC = pd.read_csv('featuresRankedRFC_550.csv')
```

Mutual Information Gain Features

```
featuresMIC.head()
```

```
       ERA+  OPS+      RA/G      1Run        SV       R/G      BB_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.559387  0.302094  0.339394  0.429180
0.45
1  0.637681  0.32  0.000000  0.390805  0.194577  0.133333  0.154861
0.65
2  0.231884  0.56  0.300995  0.641762  0.265422  0.363636  0.429180
0.75
3  0.275362  0.36  0.161692  0.618774  0.100396  0.000000  0.688073
0.40
4  0.594203  0.46  0.208955  0.392720  0.390013  0.442424  0.204570
0.60

   Under500    DefEff  ...        GF        IP       GDP        CS
SH  \
0  0.402703  0.602041  ...  0.555251  0.678982  0.625415  0.246236
0.404816
1  0.389189  1.000000  ...  0.306162  0.697641  0.443133  0.302859
0.481526
2  0.385135  0.581633  ...  0.466301  0.710241  0.414272  0.201055
0.412176
3  0.271622  0.867347  ...  0.366232  0.680184  0.562072  0.264308
0.485779
4  0.528378  0.663265  ...  0.403437  0.752545  0.564103  0.351948
0.489978

         SB        SO       BPF       HBP  Playoffs
0  0.096689  0.207712  0.547619  0.209898         0
1  0.197865  0.376993  0.404762  0.219712         0
2  0.157703  0.294791  0.476190  0.225806         0
3  0.128802  0.286874  0.190476  0.130358         0
4  0.165917  0.247071  0.571429  0.152792         0
```

```
[5 rows x 45 columns]

# Stratified train test split is used to preserves the same
proportions of examples in each class as observed in the original
dataset.

xMIC = featuresMIC.iloc[:,0:-1]
yMIC = featuresMIC['Playoffs']

X_train, X_test, y_train, y_test = train_test_split(xMIC, yMIC,
test_size=.2, random_state=1, stratify=yMIC)

# For-loop is created to evaluate the F1 score of each learning
algorithm as features are added one by one to the dataset

resultsLR = []
for i in range(1,45):
  score = cross_val_score(LogisticRegression(), X_train.iloc[:, 0:i],
y_train, scoring='f1', cv=10)
  resultsLR.append(np.mean(score))

resultsKNN = []
for i in range(1,45):
  score = cross_val_score(KNeighborsClassifier(), X_train.iloc[:,
0:i], y_train, scoring='f1', cv=10)
  resultsKNN.append(np.mean(score))

resultsRF = []
for i in range(1,45):
  score = cross_val_score(RandomForestClassifier(), X_train.iloc[:,
0:i], y_train, scoring='f1', cv=10)
  resultsRF.append(np.mean(score))

# Visual representation of the performance of all three machine
learning algorithms

plt.figure(figsize=(12,6))
plt.title("F1 Score with Mutual Information Gain Features")
plt.xlabel("Feature Count")
plt.ylabel("F1 Score")
xticks(np.arange(0,45, step=5))
yticks(np.arange(.50, .95, step=0.025))
plt.plot(resultsLR, label = "Logistic Regression")
plt.plot(resultsKNN, label = "K Nearest Neighbor")
plt.plot(resultsRF, label = "Random Forest")
plt.legend()
plt.show()
```



ANOVA F-test Features

```
featuresFC.head()

       ERA+  OPS+      RA/G       R/G      1Run        SV       H_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.339394  0.559387  0.302094  0.622353
0.45
1  0.637681  0.32  0.000000  0.133333  0.390805  0.194577  0.181001
0.65
2  0.231884  0.56  0.300995  0.363636  0.641762  0.265422  0.464920
0.75
3  0.275362  0.36  0.161692  0.000000  0.618774  0.100396  0.159225
0.40
4  0.594203  0.46  0.208955  0.442424  0.392720  0.390013  0.482468
0.60

       BB_P        BB  ...     HBP_P        SH        CS        Ch
3B  \
0  0.429180  0.465585  ...  0.175247  0.404816  0.246236  0.621987
0.172537
1  0.154861  0.410213  ...  0.058972  0.481526  0.302859  0.609655
0.344682
2  0.429180  0.440077  ...  0.313891  0.412176  0.201055  0.594857
0.412642
3  0.688073  0.021753  ...  0.196577  0.485779  0.264308  0.570052
0.299651
4  0.204570  0.540521  ...  0.258242  0.489978  0.351948  0.821312
0.493763

        BPF        GF        SO       GDP  Playoffs
```

```
0  0.547619  0.555251  0.207712  0.625415         0
1  0.404762  0.306162  0.376993  0.443133         0
2  0.476190  0.466301  0.294791  0.414272         0
3  0.190476  0.366232  0.286874  0.562072         0
4  0.571429  0.403437  0.247071  0.564103         0

[5 rows x 45 columns]
```

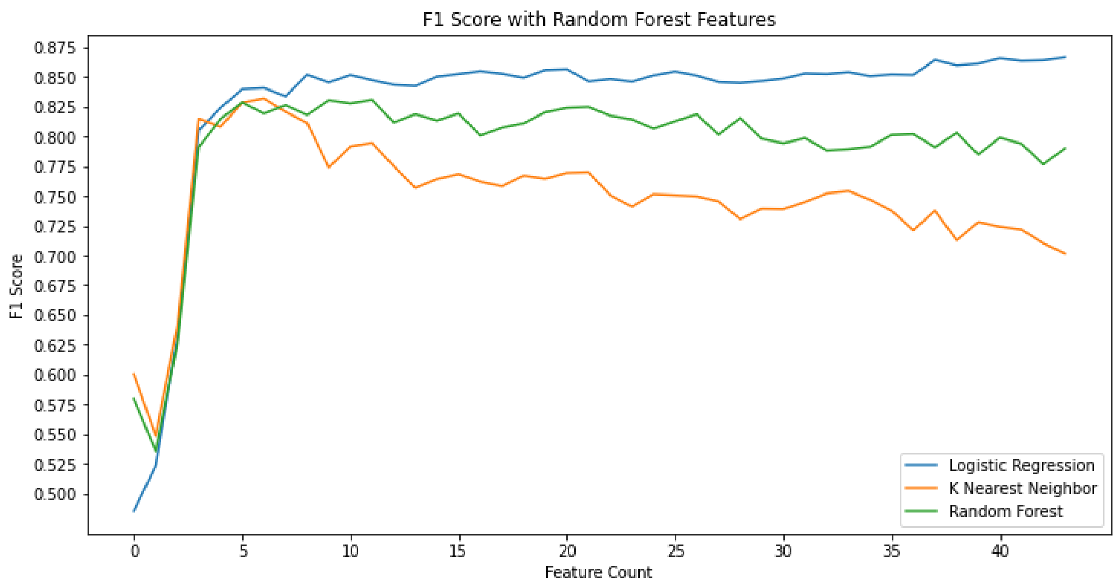
```
# The above steps are repeated for the ranked features from the ANOVA
f-test selection algorithm

xFC = featuresFC.iloc[:,0:-1]
yFC = featuresFC['Playoffs']

X_train, X_test, y_train, y_test = train_test_split(xFC, yFC,
test_size=.2, random_state=1, stratify=yFC)

resultsLR_FC = []
for i in range(1,45):
  score = cross_val_score(LogisticRegression(), X_train.iloc[:, 0:i],
y_train, scoring='f1', cv=10)
  resultsLR_FC.append(np.mean(score))

resultsKNN_FC = []
for i in range(1,45):
  score = cross_val_score(KNeighborsClassifier(), X_train.iloc[:,
0:i], y_train, scoring='f1', cv=10)
  resultsKNN_FC.append(np.mean(score))

resultsRF_FC = []
for i in range(1,45):
  score = cross_val_score(RandomForestClassifier(), X_train.iloc[:,
0:i], y_train, scoring='f1', cv=10)
  resultsRF_FC.append(np.mean(score))

plt.figure(figsize=(12,6))
plt.title("F1 Score with ANOVA F-test Features")
plt.xlabel("Feature Count")
plt.ylabel("F1 Score")
xticks(np.arange(0,45, step=5))
yticks(np.arange(.50, .95, step=0.025))
plt.plot(resultsLR_FC, label = "Logistic Regression")
plt.plot(resultsKNN_FC, label = "K Nearest Neighbor")
plt.plot(resultsRF_FC, label = "Random Forest")
plt.legend()
plt.show()
```

Random Forest Features

```
featuresRFC.head()

        ERA+      RA/G      1Run       R/G  OPS+        SV       H_P
BB_P  \
0  0.173913  0.477612  0.559387  0.339394  0.44  0.302094  0.622353
0.429180
1  0.637681  0.000000  0.390805  0.133333  0.32  0.194577  0.181001
0.154861
2  0.231884  0.300995  0.641762  0.363636  0.56  0.265422  0.464920
0.429180
3  0.275362  0.161692  0.618774  0.000000  0.36  0.100396  0.159225
0.688073
4  0.594203  0.208955  0.392720  0.442424  0.46  0.390013  0.482468
0.204570

          BB    DefEff  ...   #Bat       GDP        GF        SH
WP  \
0  0.465585  0.602041  ...  0.175  0.625415  0.555251  0.404816
0.687796
1  0.410213  1.000000  ...  0.075  0.443133  0.306162  0.481526
0.262041
2  0.440077  0.581633  ...  0.225  0.414272  0.466301  0.412176
0.221357
3  0.021753  0.867347  ...  0.275  0.562072  0.366232  0.485779
0.435667
4  0.540521  0.663265  ...  0.300  0.564103  0.403437  0.489978
0.218771

          3B        BK       BPF        CS  Playoffs
```

```
0  0.172537  0.110017  0.547619  0.246236         0
1  0.344682  0.027683  0.404762  0.302859         0
2  0.412642  0.027504  0.476190  0.201055         0
3  0.299651  0.041256  0.190476  0.264308         0
4  0.493763  0.054656  0.571429  0.351948         0

[5 rows x 45 columns]
```

```python
# The above steps are repeated for the ranked features from the Random
Forest selection algorithm

xRFC = featuresRFC.iloc[:,0:-1]
yRFC = featuresRFC['Playoffs']

X_train, X_test, y_train, y_test = train_test_split(xRFC, yRFC,
test_size=.2, random_state=1, stratify=yRFC)

resultsLR_RFC = []
for i in range(1,45):
  score = cross_val_score(LogisticRegression(), X_train.iloc[:, 0:i],
y_train, scoring='f1', cv=10)
  resultsLR_RFC.append(np.mean(score))

resultsKNN_RFC = []
for i in range(1,45):
  score = cross_val_score(KNeighborsClassifier(), X_train.iloc[:,
0:i], y_train, scoring='f1', cv=10)
  resultsKNN_RFC.append(np.mean(score))

resultsRF_RFC = []
for i in range(1,45):
  score = cross_val_score(RandomForestClassifier(), X_train.iloc[:,
0:i], y_train, scoring='f1', cv=10)
  resultsRF_RFC.append(np.mean(score))

plt.figure(figsize=(12,6))
plt.title("F1 Score with Random Forest Features")
plt.xlabel("Feature Count")
plt.ylabel("F1 Score")
xticks(np.arange(0,45, step=5))
yticks(np.arange(.50, .95, step=0.025))
plt.plot(resultsLR_RFC, label = "Logistic Regression")
plt.plot(resultsKNN_RFC, label = "K Nearest Neighbor")
plt.plot(resultsRF_RFC, label = "Random Forest")
plt.legend()
plt.show()
```



F1 Score with Random Forest Features
F1 Score
Feature Count
Logistic Regression
K Nearest Neighbor
Random Forest
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0
5
10
15
20
25
30
35
40

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_predict, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report,
f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score,
validation_curve, StratifiedShuffleSplit
from sklearn import metrics

featuresMIC = pd.read_csv('featuresRankedMIC_550.csv')
featuresFC = pd.read_csv('featuresRankedFC_550.csv')
featuresRFC = pd.read_csv('featuresRankedRFC_550.csv')
```

Mutual Information Gain

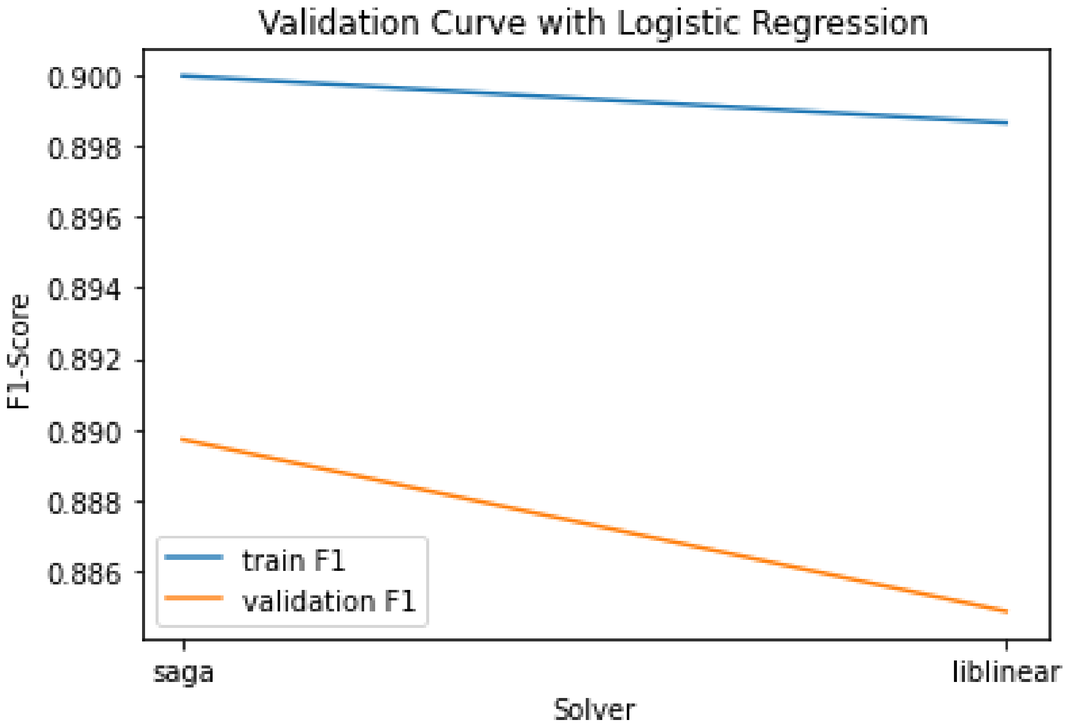
```
X = featuresMIC.iloc[:,0:20]
y = featuresMIC['Playoffs']

featuresMIC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G      1Run        SV       R/G      BB_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.559387  0.302094  0.339394  0.429180
0.45
1  0.637681  0.32  0.000000  0.390805  0.194577  0.133333  0.154861
0.65
2  0.231884  0.56  0.300995  0.641762  0.265422  0.363636  0.429180
0.75
3  0.275362  0.36  0.161692  0.618774  0.100396  0.000000  0.688073
0.40
4  0.594203  0.46  0.208955  0.392720  0.390013  0.442424  0.204570
0.60

   Under500    DefEff        PA      Rtot       H_P       SOS
WP  \
0  0.402703  0.602041  0.594126  0.391473  0.622353  0.545455
0.687796
1  0.389189  1.000000  0.356231  0.841085  0.181001  0.454545
0.262041
2  0.385135  0.581633  0.523644  0.372093  0.464920  0.454545
0.221357
3  0.271622  0.867347  0.279588  0.569767  0.159225  0.454545
0.435667
4  0.528378  0.663265  0.566620  0.441860  0.482468  0.363636
0.218771
```

```
       PAge        2B        BB       LOB      HR_P
0  0.405941  0.151402  0.465585  0.695640  0.418914
1  0.465347  0.191710  0.410213  0.400425  0.123964
2  0.475248  0.361411  0.440077  0.555105  0.189573
3  0.336634  0.078143  0.021753  0.387544  0.142855
4  0.455446  0.242631  0.540521  0.517147  0.233241
```

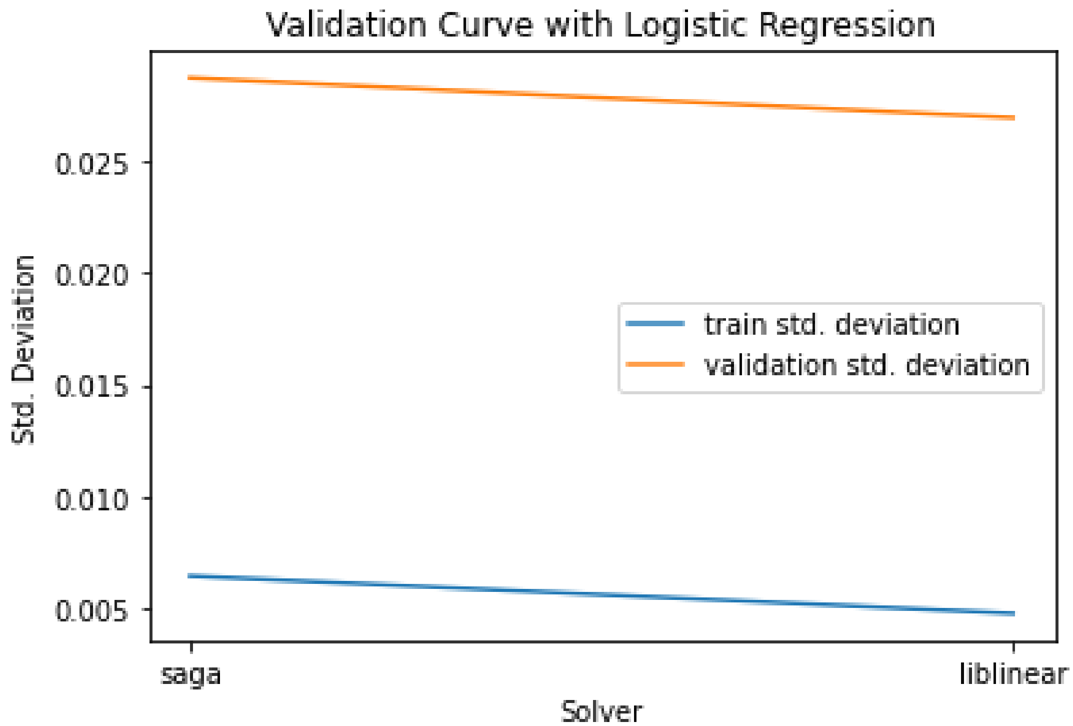
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

penalty = ['none', 'l2', 'l1']
solver = ['saga', 'liblinear', 'sag', 'lbfgs', 'newton-cg']

train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("F1-Score")
plt.plot(solver, train_scores_mean, label="train F1")
plt.plot(solver, valid_scores_mean, label="validation F1")

plt.legend()
plt.show()
```

```
[0.88971155 0.88485759        nan        nan        nan]
```

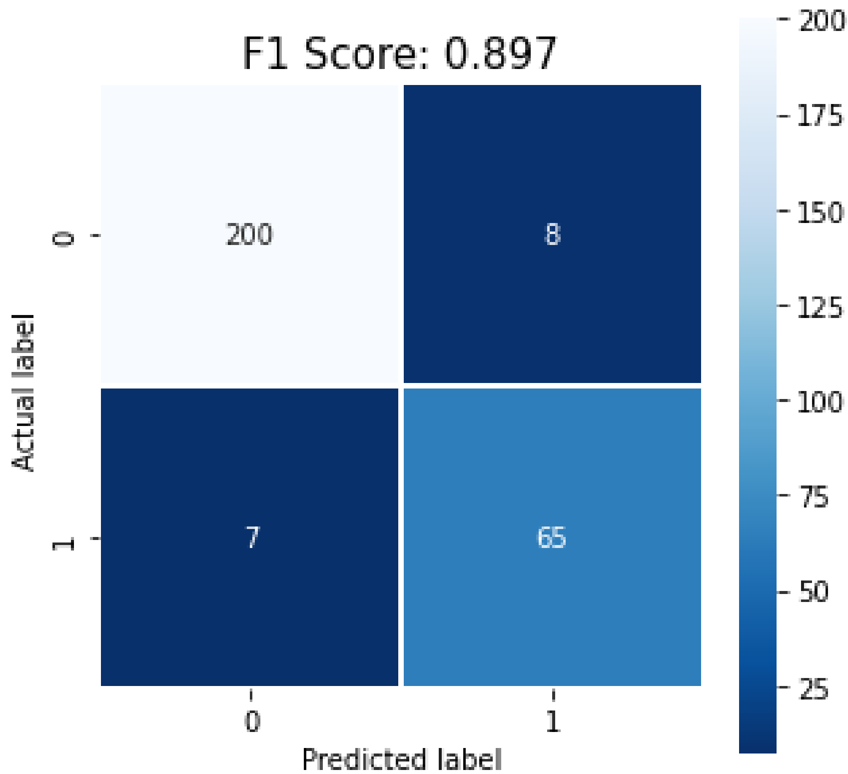



```
train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("Std. Deviation")
plt.plot(solver, train_scores_std, label="train std. deviation")
plt.plot(solver, valid_scores_std, label="validation std. deviation")

plt.legend()
plt.show()
```

```
[0.02874262 0.02695945        nan        nan        nan]
```




```
logreg = LogisticRegression(max_iter=5000, solver='saga',
penalty='l1')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1 Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  65
FP:  8
TN:  200
FN:  7

              precision    recall  f1-score   support

           0       0.97      0.96      0.96       208
           1       0.89      0.90      0.90        72

    accuracy                           0.95       280
   macro avg       0.93      0.93      0.93       280
weighted avg       0.95      0.95      0.95       280
```

ANOVA F-Test

```
X = featuresFC.iloc[:,0:20]
y = featuresFC['Playoffs']
```

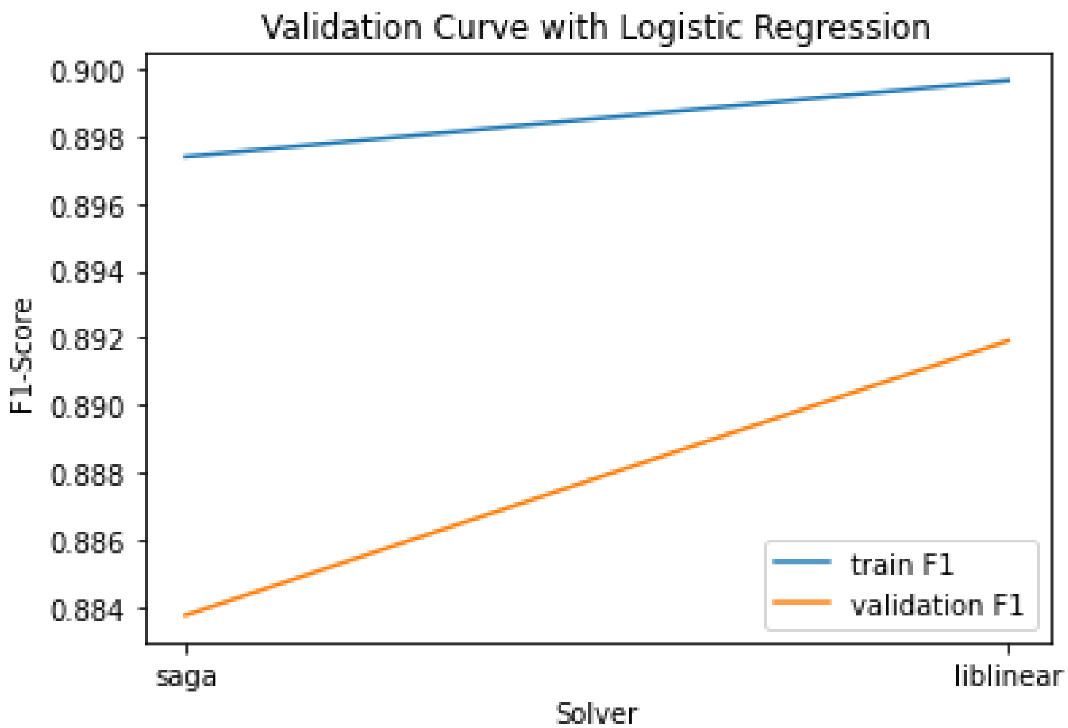
```
featuresFC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G       R/G      1Run        SV       H_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.339394  0.559387  0.302094  0.622353
0.45
1  0.637681  0.32  0.000000  0.133333  0.390805  0.194577  0.181001
0.65
2  0.231884  0.56  0.300995  0.363636  0.641762  0.265422  0.464920
0.75
3  0.275362  0.36  0.161692  0.000000  0.618774  0.100396  0.159225
0.40
4  0.594203  0.46  0.208955  0.442424  0.392720  0.390013  0.482468
0.60

       BB_P        BB      Rtot    DefEff        PA  Under500
HR  \
0  0.429180  0.465585  0.391473  0.602041  0.594126  0.402703
0.380797
1  0.154861  0.410213  0.841085  1.000000  0.356231  0.389189
0.213898
2  0.429180  0.440077  0.372093  0.581633  0.523644  0.385135
0.303797
3  0.688073  0.021753  0.569767  0.867347  0.279588  0.271622
0.126699
4  0.204570  0.540521  0.441860  0.663265  0.566620  0.528378
0.335165

         SF         E        IP       SOS     IBB_P
0  0.358281  0.743973  0.678982  0.545455  0.495551
1  0.398932  0.349282  0.697641  0.454545  0.471563
2  0.538171  0.558574  0.710241  0.454545  0.441491
3  0.124424  0.444482  0.680184  0.454545  0.477531
4  0.443268  0.566802  0.752545  0.363636  0.501326
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("F1-Score")
plt.plot(solver, train_scores_mean, label="train F1")
plt.plot(solver, valid_scores_mean, label="validation F1")
```

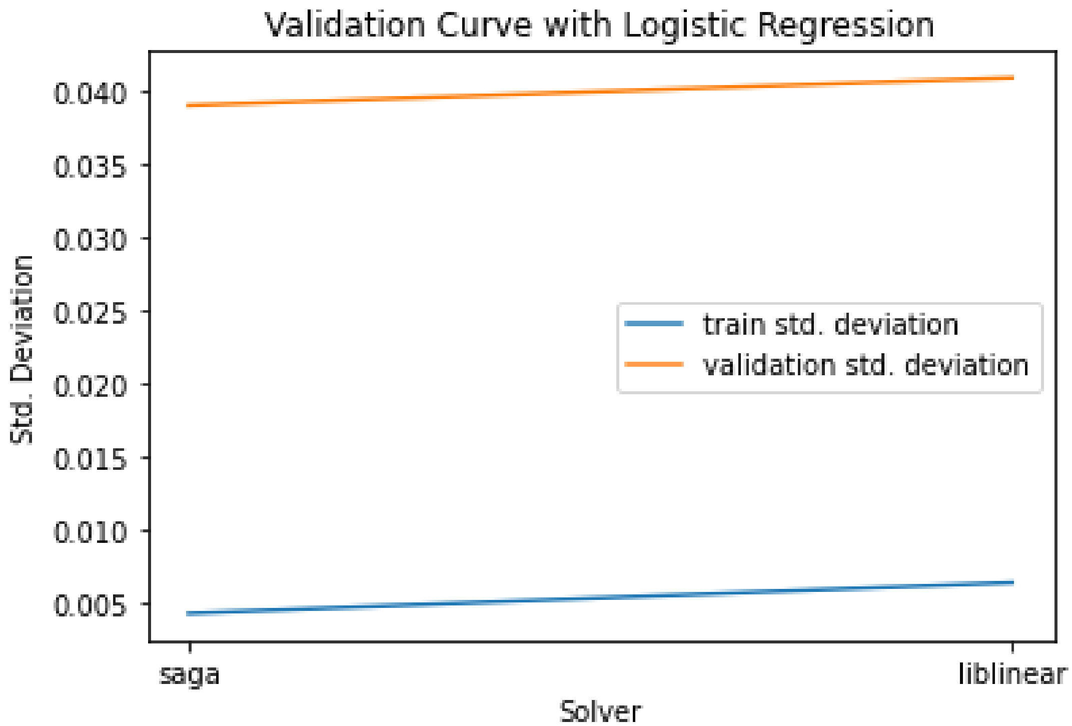
```
plt.legend()
plt.show()
```

```
[0.88376246 0.89191886        nan        nan        nan]
```

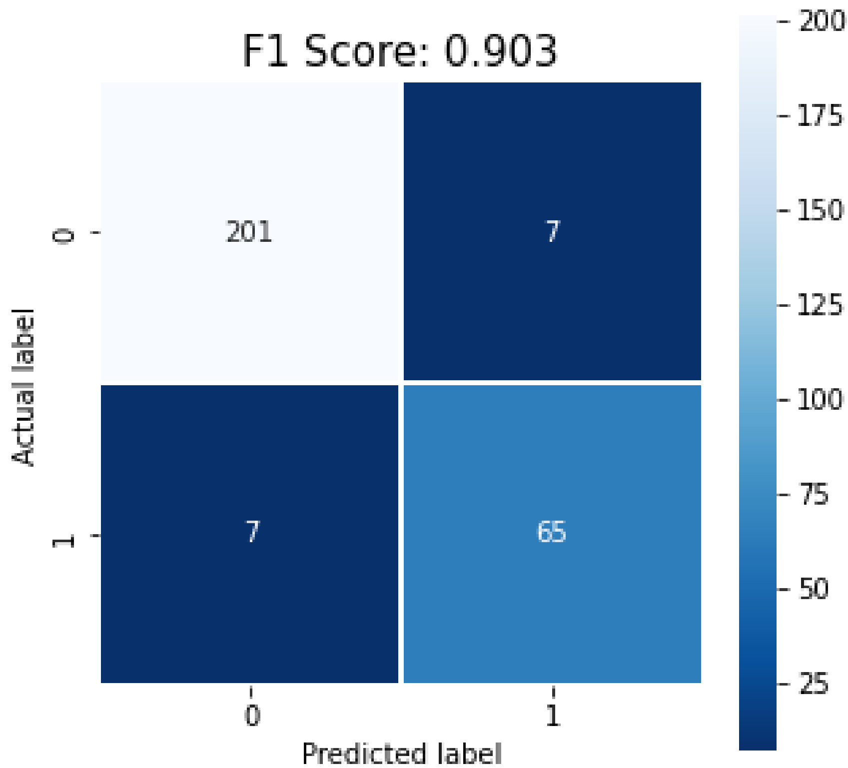



```
train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("Std. Deviation")
plt.plot(solver, train_scores_std, label="train std. deviation")
plt.plot(solver, valid_scores_std, label="validation std. deviation")

plt.legend()
plt.show()
```

```
[0.03908391 0.04097291        nan        nan        nan]
```




```
logreg = LogisticRegression(max_iter=5000, solver='saga',
penalty='l1')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1 Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  65
FP:  7
TN:  201
FN:  7

              precision    recall  f1-score   support

           0       0.97      0.97      0.97       208
           1       0.90      0.90      0.90        72

    accuracy                           0.95       280
   macro avg       0.93      0.93      0.93       280
weighted avg       0.95      0.95      0.95       280
```

Random Forest

```
X = featuresRFC.iloc[:,0:20]
y = featuresRFC['Playoffs']
```

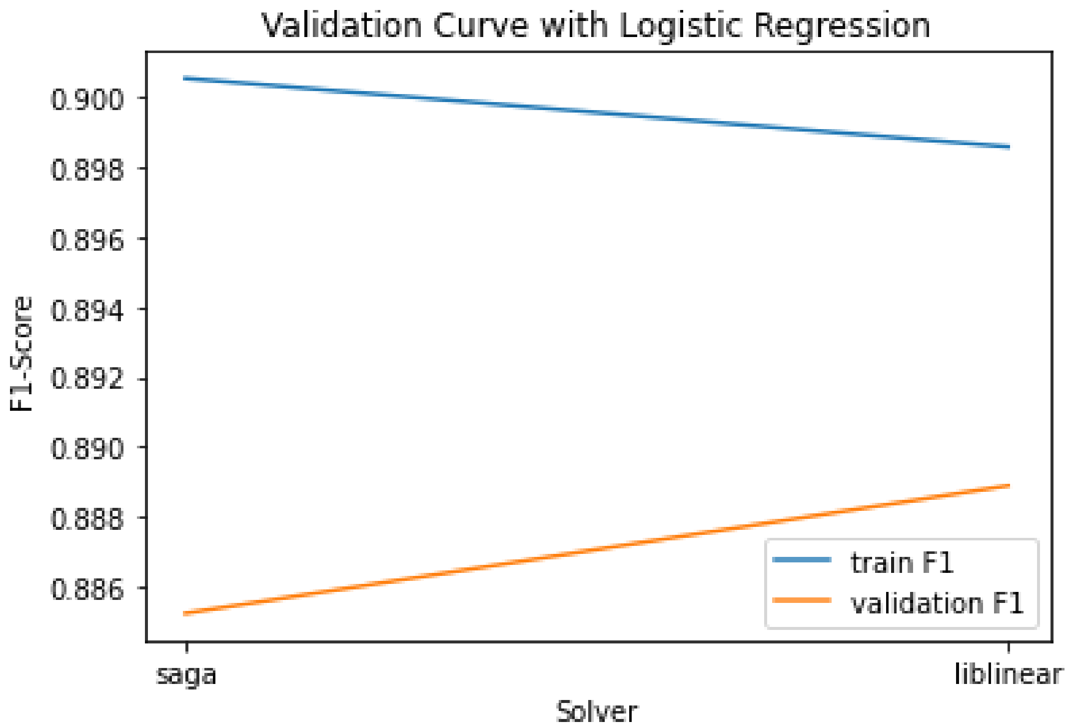
```
featuresRFC.iloc[:,0:20].head()
```

```
       ERA+      RA/G      1Run       R/G  OPS+        SV       H_P
BB_P  \
0  0.173913  0.477612  0.559387  0.339394  0.44  0.302094  0.622353
0.429180
1  0.637681  0.000000  0.390805  0.133333  0.32  0.194577  0.181001
0.154861
2  0.231884  0.300995  0.641762  0.363636  0.56  0.265422  0.464920
0.429180
3  0.275362  0.161692  0.618774  0.000000  0.36  0.100396  0.159225
0.688073
4  0.594203  0.208955  0.392720  0.442424  0.46  0.390013  0.482468
0.204570

         BB    DefEff      Rtot        HR  #a-tA-S        IP       SOS
\
0  0.465585  0.602041  0.391473  0.380797     0.45  0.678982  0.545455

1  0.410213  1.000000  0.841085  0.213898     0.65  0.697641  0.454545

2  0.440077  0.581633  0.372093  0.303797     0.75  0.710241  0.454545

3  0.021753  0.867347  0.569767  0.126699     0.40  0.680184  0.454545

4  0.540521  0.663265  0.441860  0.335165     0.60  0.752545  0.363636


         PA      HR_P        SF        2B  Under500
0  0.594126  0.418914  0.358281  0.151402  0.402703
1  0.356231  0.123964  0.398932  0.191710  0.389189
2  0.523644  0.189573  0.538171  0.361411  0.385135
3  0.279588  0.142855  0.124424  0.078143  0.271622
4  0.566620  0.233241  0.443268  0.242631  0.528378
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("F1-Score")
plt.plot(solver, train_scores_mean, label="train F1")
plt.plot(solver, valid_scores_mean, label="validation F1")
```

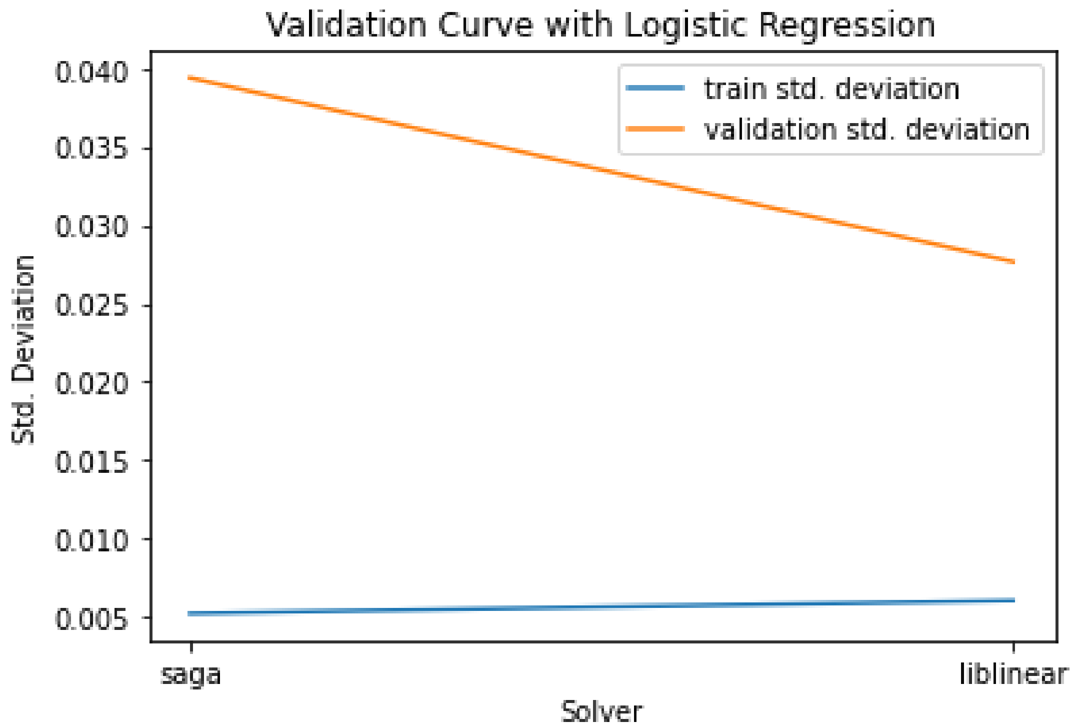
```
plt.legend()
plt.show()
```

```
[0.88523787 0.88888916        nan        nan        nan]
```

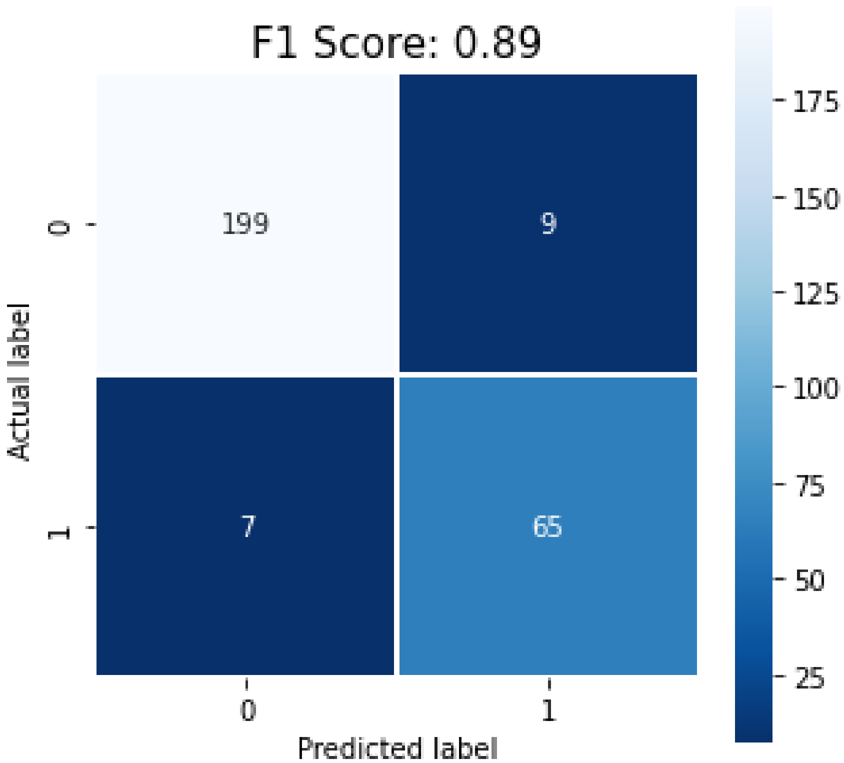



```
train_scores, valid_scores =
validation_curve(LogisticRegression(max_iter=5000, penalty='l1'),
X_train, y_train, param_name="solver", param_range=solver,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("Solver")
plt.ylabel("Std. Deviation")
plt.plot(solver, train_scores_std, label="train std. deviation")
plt.plot(solver, valid_scores_std, label="validation std. deviation")

plt.legend()
plt.show()
```

```
[0.03938935 0.02765877        nan        nan        nan]
```




```
logreg = LogisticRegression(max_iter=5000, solver='liblinear',
penalty='l1')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1 Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  65
FP:  9
TN:  199
FN:  7

              precision    recall  f1-score   support

           0       0.97      0.96      0.96       208
           1       0.88      0.90      0.89        72

    accuracy                           0.94       280
   macro avg       0.92      0.93      0.93       280
weighted avg       0.94      0.94      0.94       280
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score, f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, cross_val_score,
validation_curve, StratifiedShuffleSplit
from sklearn import metrics

featuresMIC = pd.read_csv('featuresRankedMIC_550.csv')
featuresFC = pd.read_csv('featuresRankedFC_550.csv')
featuresRFC = pd.read_csv('featuresRankedRFC_550.csv')
```

Mutual Information Gain

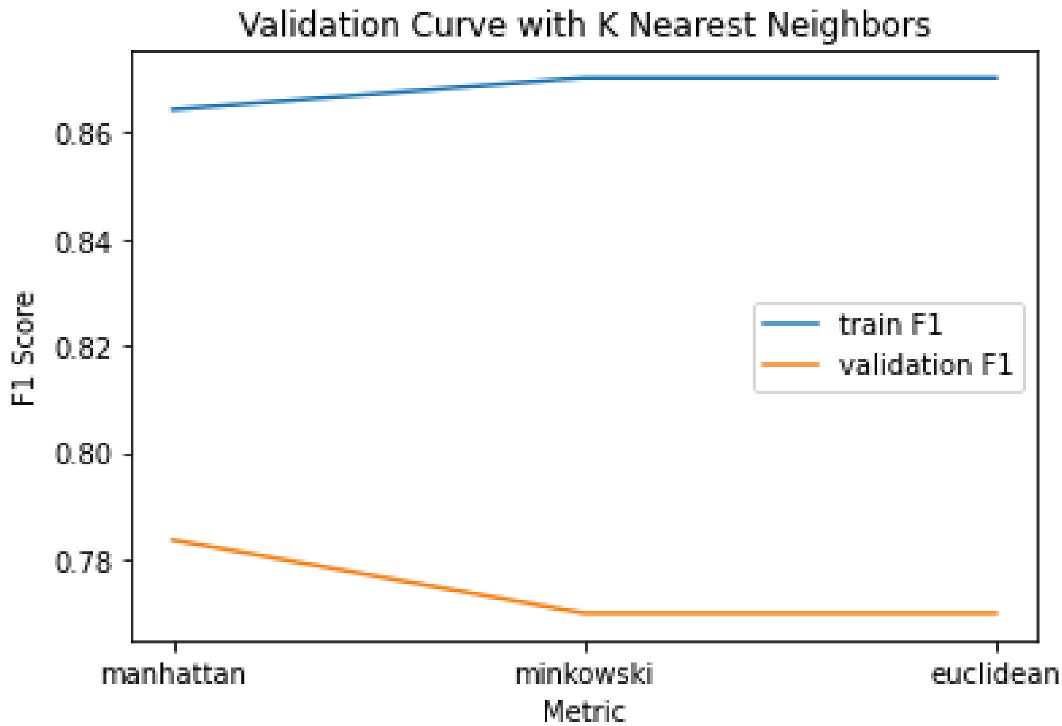
```
X = featuresMIC.iloc[:,0:20]
y = featuresMIC['Playoffs']

featuresMIC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G      1Run        SV       R/G      BB_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.559387  0.302094  0.339394  0.429180
0.45
1  0.637681  0.32  0.000000  0.390805  0.194577  0.133333  0.154861
0.65
2  0.231884  0.56  0.300995  0.641762  0.265422  0.363636  0.429180
0.75
3  0.275362  0.36  0.161692  0.618774  0.100396  0.000000  0.688073
0.40
4  0.594203  0.46  0.208955  0.392720  0.390013  0.442424  0.204570
0.60

   Under500    DefEff        PA      Rtot       H_P       SOS
WP  \
0  0.402703  0.602041  0.594126  0.391473  0.622353  0.545455
0.687796
1  0.389189  1.000000  0.356231  0.841085  0.181001  0.454545
0.262041
2  0.385135  0.581633  0.523644  0.372093  0.464920  0.454545
0.221357
3  0.271622  0.867347  0.279588  0.569767  0.159225  0.454545
0.435667
4  0.528378  0.663265  0.566620  0.441860  0.482468  0.363636
0.218771
```

```
       PAge        2B        BB       LOB      HR_P
0  0.405941  0.151402  0.465585  0.695640  0.418914
1  0.465347  0.191710  0.410213  0.400425  0.123964
2  0.475248  0.361411  0.440077  0.555105  0.189573
3  0.336634  0.078143  0.021753  0.387544  0.142855
4  0.455446  0.242631  0.540521  0.517147  0.233241
```

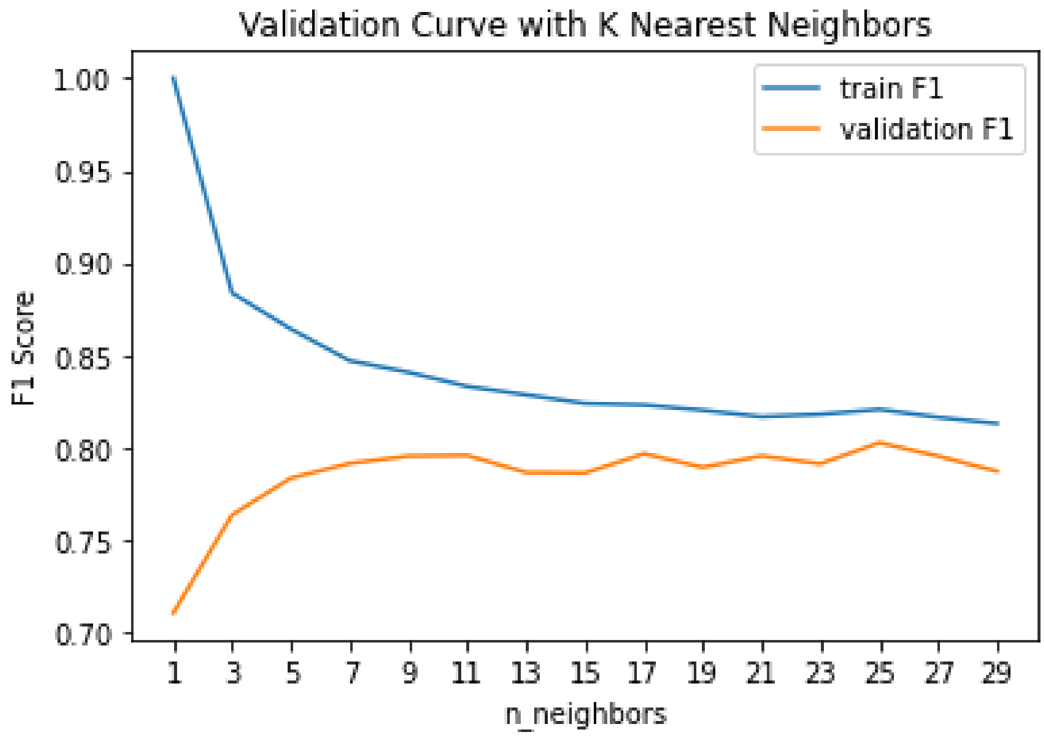
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

KNNmetrics = ['manhattan','minkowski','euclidean']
n_neighbors = np.arange(1,30,2)

train_scores, valid_scores = validation_curve(KNeighborsClassifier(),
X_train, y_train, param_name="metric", param_range=KNNmetrics,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("Metric")
plt.ylabel("F1 Score")
plt.plot(KNNmetrics, train_scores_mean, label="train F1")
plt.plot(KNNmetrics, valid_scores_mean, label="validation F1")

plt.legend()
plt.show()
```

```
[0.78367449 0.76993909 0.76993909]
```



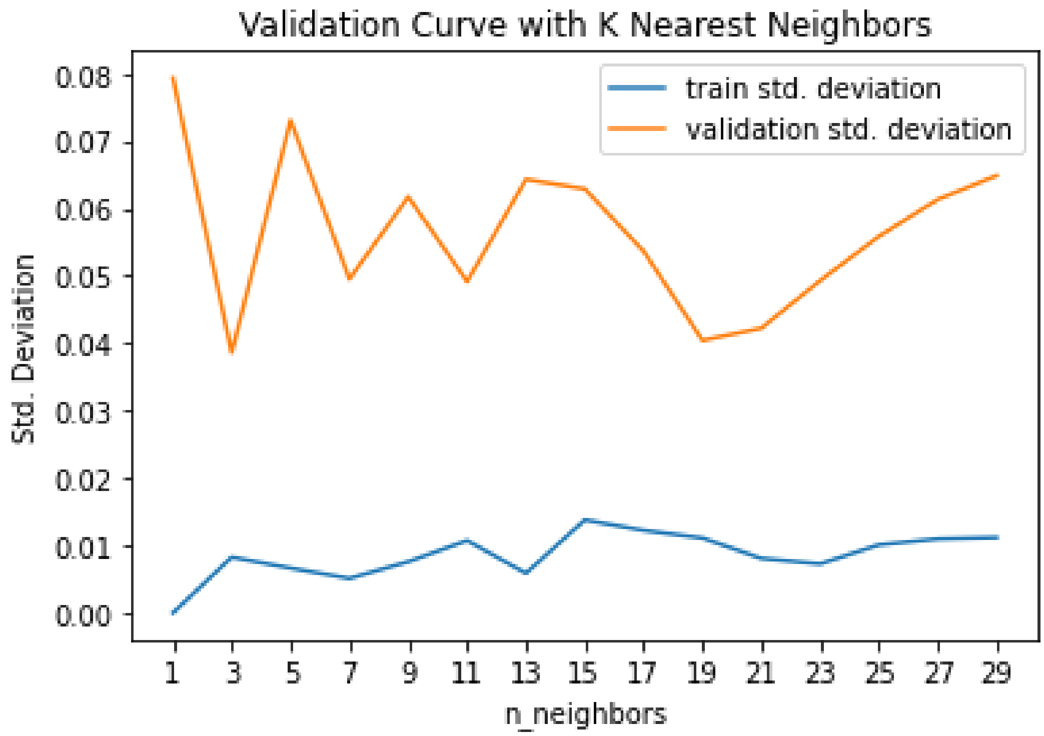

```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("F1 Score")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_mean, label="train F1")
plt.plot(n_neighbors, valid_scores_mean, label="validation F1")

plt.legend()
plt.show()
```

```
[0.71074244 0.76376035 0.78367449 0.79164172 0.7955126  0.79584306
 0.78670741 0.78646968 0.79677739 0.78956032 0.79561571 0.79143729
 0.80276471 0.7955169  0.78737018]
```

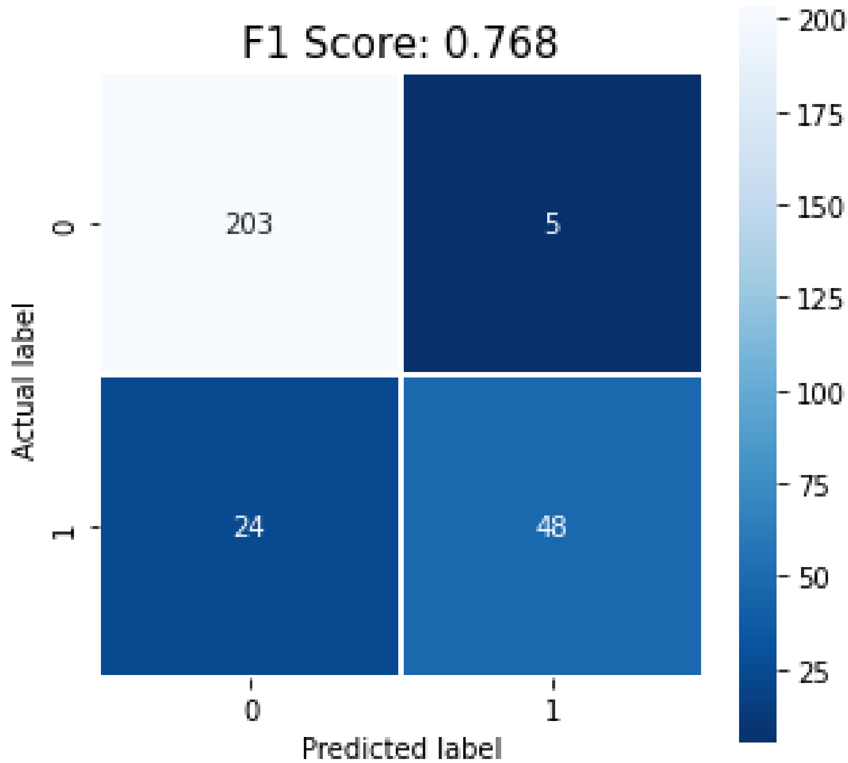



```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_std, label="train std. deviation")
plt.plot(n_neighbors, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.07937231 0.03864267 0.07309271 0.04954025 0.06173995 0.04917792
 0.06431924 0.0629297  0.05366677 0.04045119 0.04225328 0.04931703
 0.05589523 0.06136702 0.06488849]
```




```
KNN = KNeighborsClassifier(n_neighbors=17, metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1 Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  48
FP:  5
TN:  203
FN:  24

              precision    recall  f1-score   support

           0       0.89      0.98      0.93       208
           1       0.91      0.67      0.77        72

    accuracy                           0.90       280
   macro avg       0.90      0.82      0.85       280
weighted avg       0.90      0.90      0.89       280
```

ANOVA F-test

```
X = featuresFC.iloc[:,0:20]
y = featuresFC['Playoffs']
```

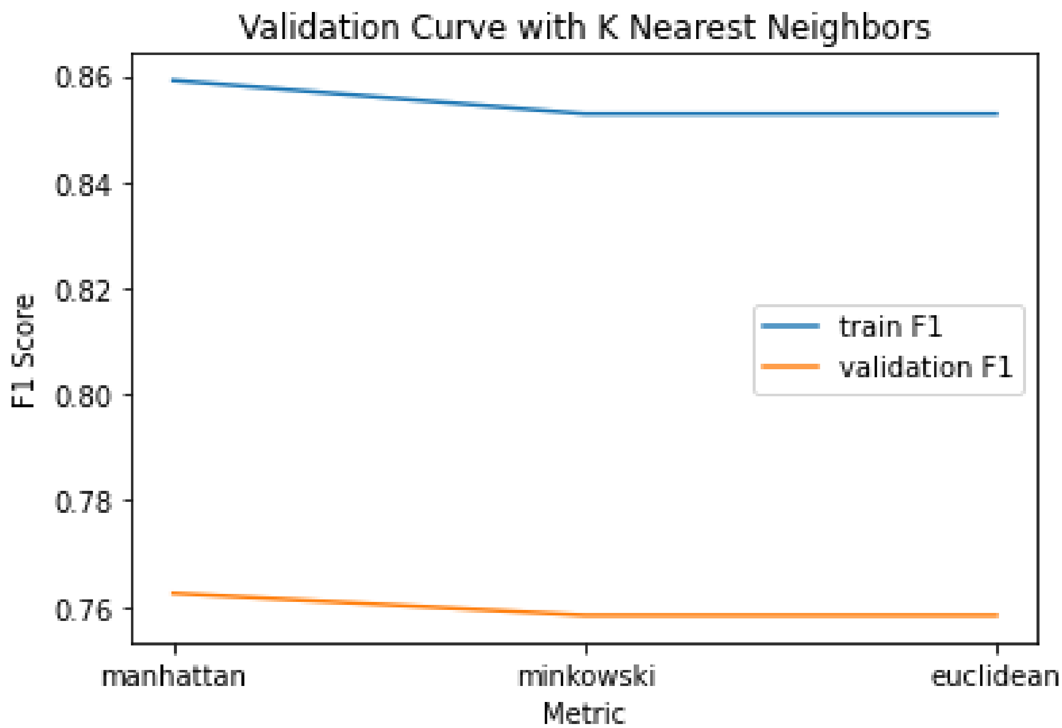
```
featuresFC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G       R/G      1Run        SV       H_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.339394  0.559387  0.302094  0.622353
0.45
1  0.637681  0.32  0.000000  0.133333  0.390805  0.194577  0.181001
0.65
2  0.231884  0.56  0.300995  0.363636  0.641762  0.265422  0.464920
0.75
3  0.275362  0.36  0.161692  0.000000  0.618774  0.100396  0.159225
0.40
4  0.594203  0.46  0.208955  0.442424  0.392720  0.390013  0.482468
0.60

       BB_P        BB      Rtot    DefEff        PA  Under500
HR  \
0  0.429180  0.465585  0.391473  0.602041  0.594126  0.402703
0.380797
1  0.154861  0.410213  0.841085  1.000000  0.356231  0.389189
0.213898
2  0.429180  0.440077  0.372093  0.581633  0.523644  0.385135
0.303797
3  0.688073  0.021753  0.569767  0.867347  0.279588  0.271622
0.126699
4  0.204570  0.540521  0.441860  0.663265  0.566620  0.528378
0.335165

         SF         E        IP       SOS     IBB_P
0  0.358281  0.743973  0.678982  0.545455  0.495551
1  0.398932  0.349282  0.697641  0.454545  0.471563
2  0.538171  0.558574  0.710241  0.454545  0.441491
3  0.124424  0.444482  0.680184  0.454545  0.477531
4  0.443268  0.566802  0.752545  0.363636  0.501326
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores = validation_curve(KNeighborsClassifier(),
X_train, y_train, param_name="metric", param_range=KNNmetrics,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("Metric")
plt.ylabel("F1 Score")
plt.plot(KNNmetrics, train_scores_mean, label="train F1")
plt.plot(KNNmetrics, valid_scores_mean, label="validation F1")
```

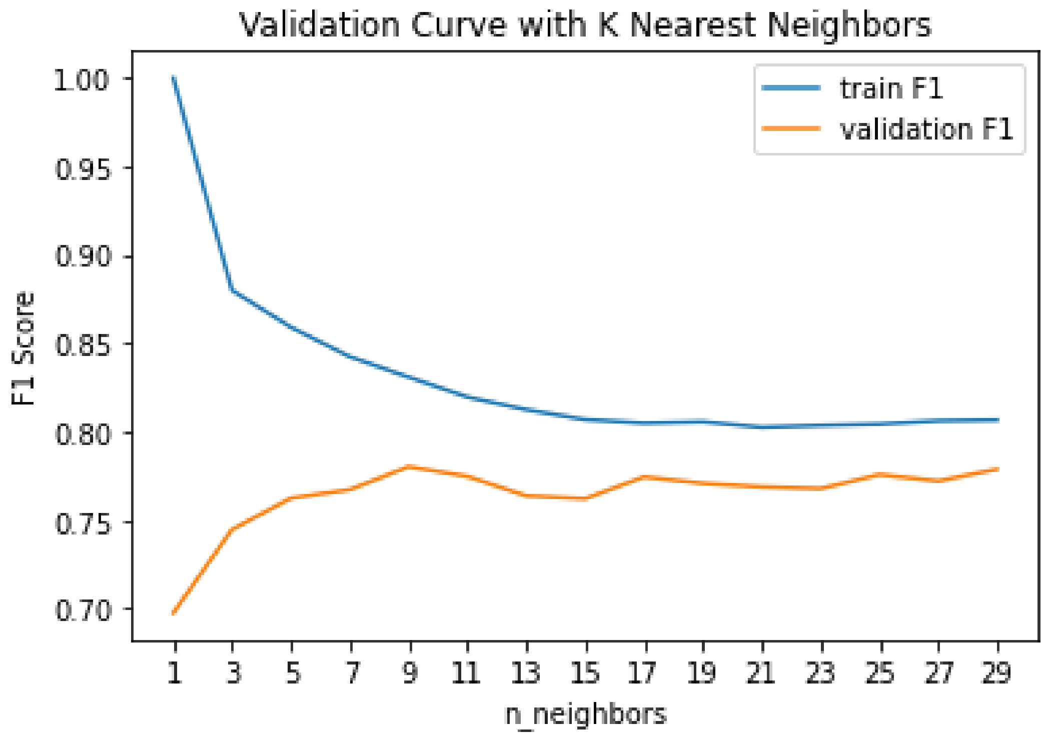
```
plt.legend()
plt.show()
```

```
[0.76251307 0.75835737 0.75835737]
```



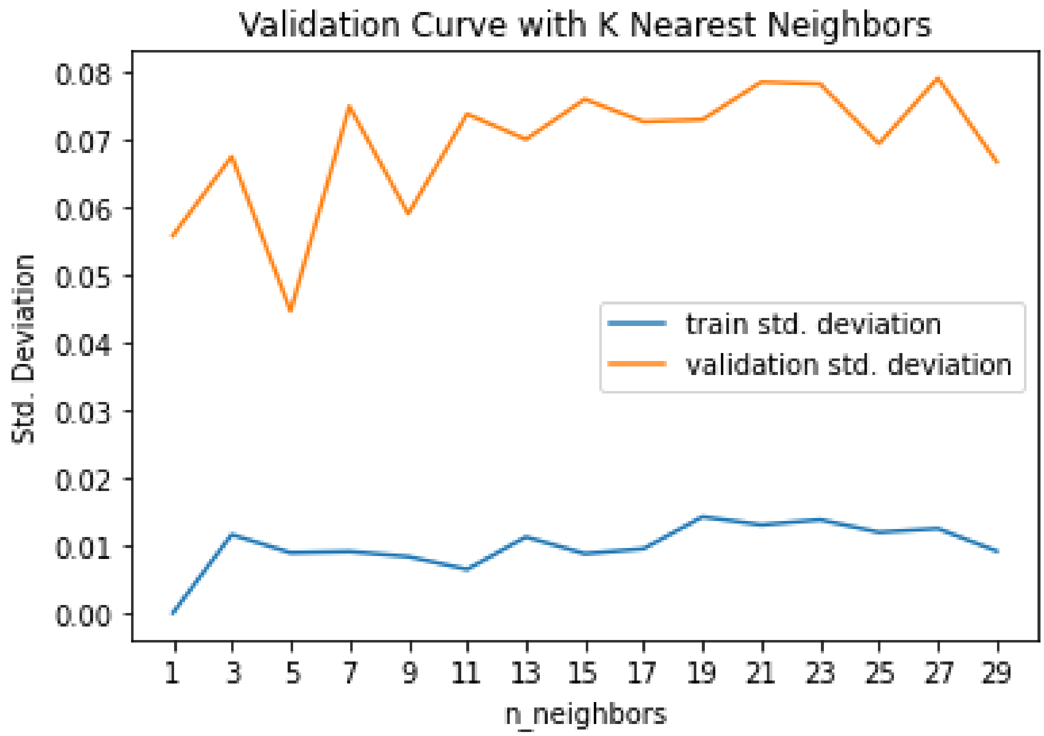

```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("F1 Score")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_mean, label="train F1")
plt.plot(n_neighbors, valid_scores_mean, label="validation F1")

plt.legend()
plt.show()
```

```
[0.69766351 0.74486567 0.76251307 0.7673495  0.78034566 0.77488069
 0.76384426 0.76222866 0.77458248 0.77094247 0.76893188 0.76792548
 0.77586455 0.77227976 0.77888503]
```

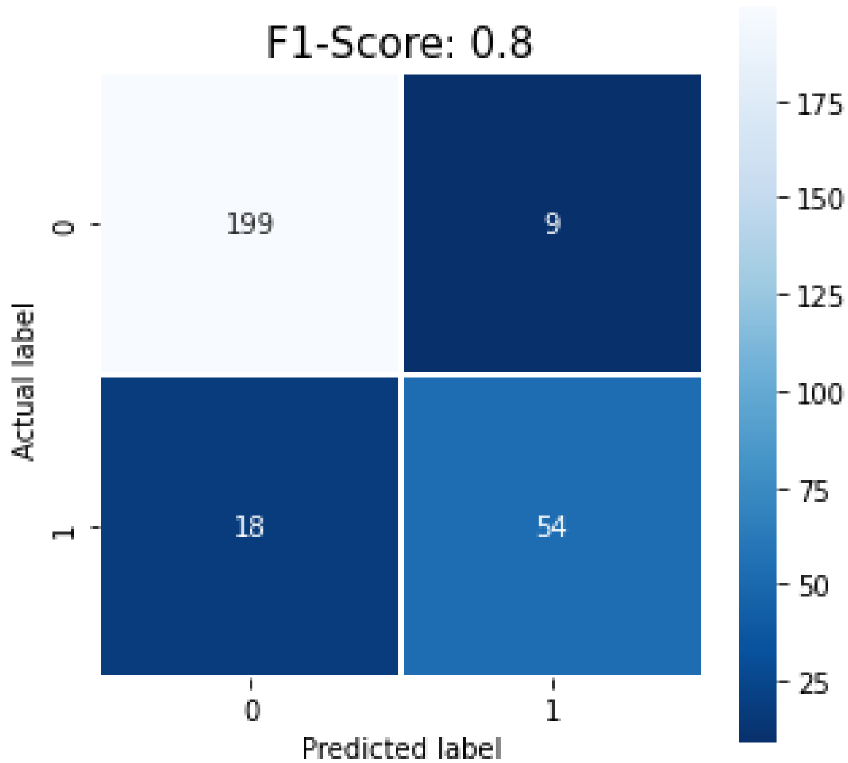



```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_std, label="train std. deviation")
plt.plot(n_neighbors, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.05576638 0.06740893 0.04458933 0.07482727 0.05896097 0.07369611
 0.06995413 0.07587622 0.07256825 0.07287657 0.07842958 0.07813558
 0.06931441 0.07901671 0.06663155]
```




```
KNN = KNeighborsClassifier(n_neighbors=17, metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1-Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  54
FP:  9
TN:  199
FN:  18

              precision    recall  f1-score   support

           0       0.92      0.96      0.94       208
           1       0.86      0.75      0.80        72

    accuracy                           0.90       280
   macro avg       0.89      0.85      0.87       280
weighted avg       0.90      0.90      0.90       280
```

Random Forest

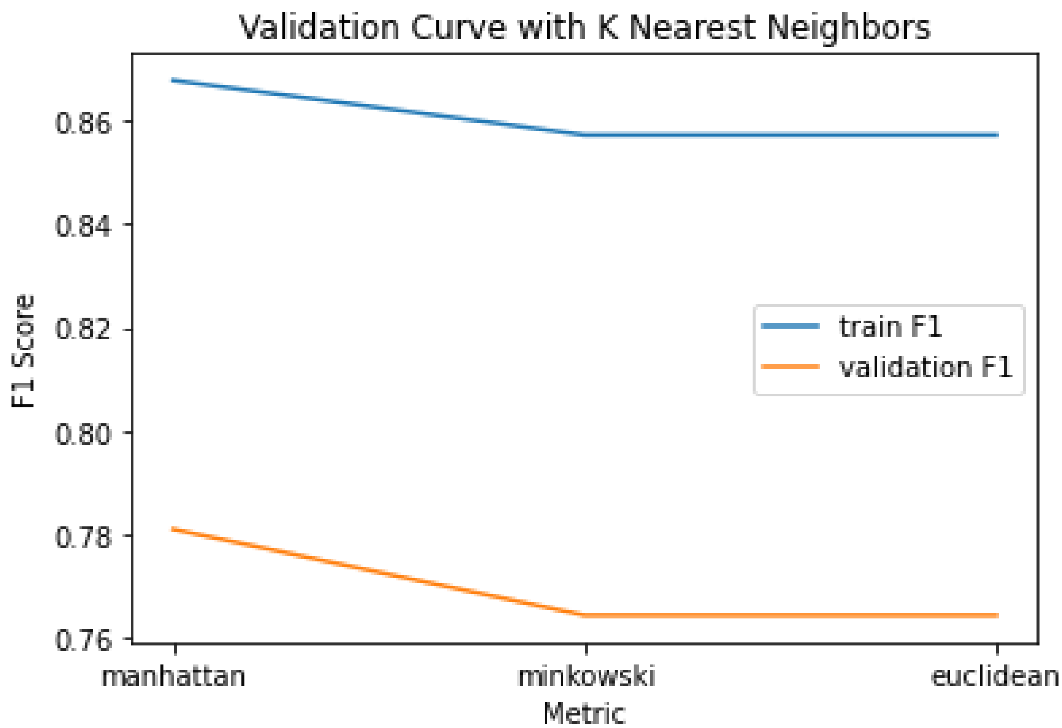
```
X = featuresRFC.iloc[:,0:20]
y = featuresRFC['Playoffs']

featuresRFC.iloc[:,0:20].head()
```

```
       ERA+      RA/G      1Run       R/G  OPS+        SV       H_P
BB_P  \
0  0.173913  0.477612  0.559387  0.339394  0.44  0.302094  0.622353
0.429180
1  0.637681  0.000000  0.390805  0.133333  0.32  0.194577  0.181001
0.154861
2  0.231884  0.300995  0.641762  0.363636  0.56  0.265422  0.464920
0.429180
3  0.275362  0.161692  0.618774  0.000000  0.36  0.100396  0.159225
0.688073
4  0.594203  0.208955  0.392720  0.442424  0.46  0.390013  0.482468
0.204570

         BB    DefEff      Rtot        HR  #a-tA-S        IP       SOS
\
0  0.465585  0.602041  0.391473  0.380797     0.45  0.678982  0.545455

1  0.410213  1.000000  0.841085  0.213898     0.65  0.697641  0.454545

2  0.440077  0.581633  0.372093  0.303797     0.75  0.710241  0.454545

3  0.021753  0.867347  0.569767  0.126699     0.40  0.680184  0.454545

4  0.540521  0.663265  0.441860  0.335165     0.60  0.752545  0.363636


         PA      HR_P        SF        2B  Under500
0  0.594126  0.418914  0.358281  0.151402  0.402703
1  0.356231  0.123964  0.398932  0.191710  0.389189
2  0.523644  0.189573  0.538171  0.361411  0.385135
3  0.279588  0.142855  0.124424  0.078143  0.271622
4  0.566620  0.233241  0.443268  0.242631  0.528378
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores = validation_curve(KNeighborsClassifier(),
X_train, y_train, param_name="metric", param_range=KNNmetrics,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("Metric")
plt.ylabel("F1 Score")
plt.plot(KNNmetrics, train_scores_mean, label="train F1")
plt.plot(KNNmetrics, valid_scores_mean, label="validation F1")
```

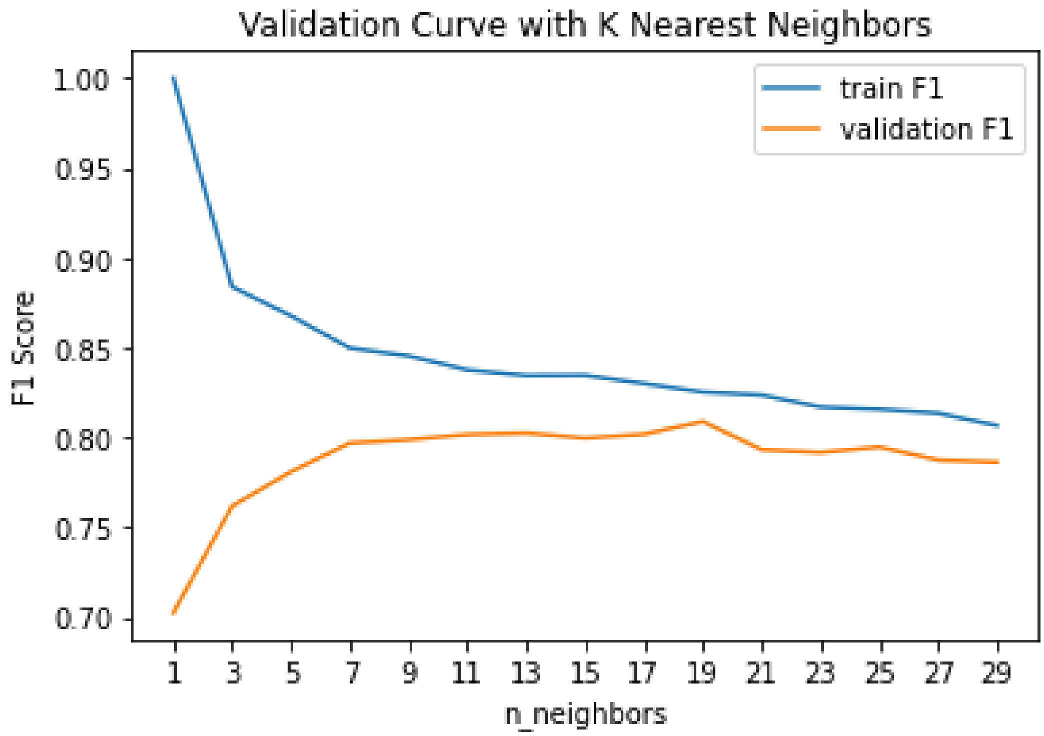
```
plt.legend()
plt.show()
```

```
[0.78104987 0.76440496 0.76440496]
```



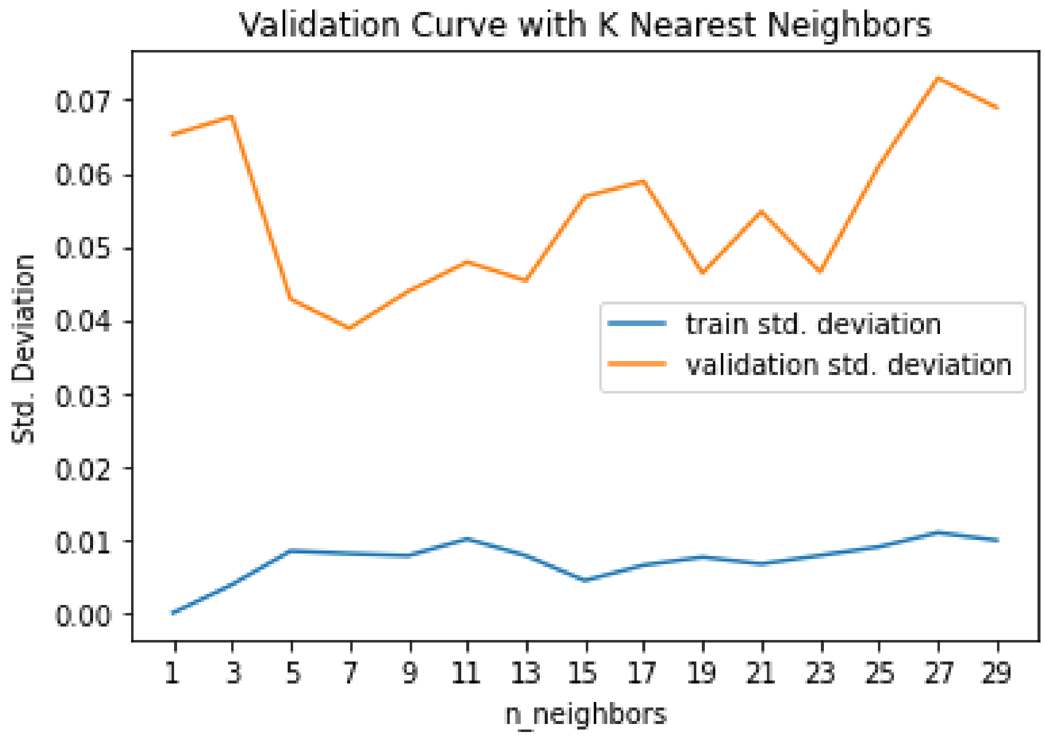

```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("F1 Score")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_mean, label="train F1")
plt.plot(n_neighbors, valid_scores_mean, label="validation F1")

plt.legend()
plt.show()
```

```
[0.70247957 0.76189353 0.78104987 0.79696148 0.79882727 0.80171604
 0.80249457 0.79970061 0.80186779 0.80898437 0.79323209 0.79174391
 0.7946962  0.78748753 0.78633519]
```

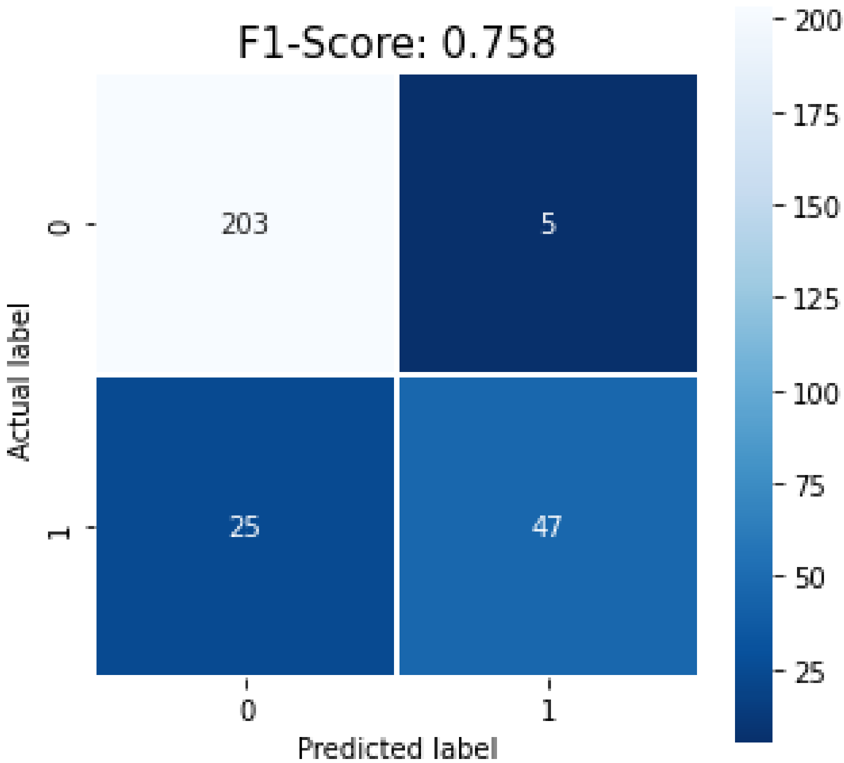



```
train_scores, valid_scores =
validation_curve(KNeighborsClassifier(metric='manhattan'), X_train,
y_train, param_name="n_neighbors", param_range=n_neighbors,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with K Nearest Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(1,30,step=2))
plt.plot(n_neighbors, train_scores_std, label="train std. deviation")
plt.plot(n_neighbors, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.06528712 0.06773952 0.04283011 0.03886243 0.04388834 0.04788952
 0.04536385 0.05683875 0.05890677 0.04638513 0.05476298 0.04655408
 0.06103022 0.07300808 0.06896755]
```




```
KNN = KNeighborsClassifier(n_neighbors=19, metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
group_names = ['TN', 'FP', 'FN', 'TP']
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1-Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  47
FP:  5
TN:  203
FN:  25

              precision    recall  f1-score   support

           0       0.89      0.98      0.93       208
           1       0.90      0.65      0.76        72

    accuracy                           0.89       280
   macro avg       0.90      0.81      0.84       280
weighted avg       0.89      0.89      0.89       280
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report,
f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_val_score,
validation_curve, StratifiedShuffleSplit
from sklearn import metrics
```

```
featuresMIC = pd.read_csv('featuresRankedMIC_550.csv')
featuresFC = pd.read_csv('featuresRankedFC_550.csv')
featuresRFC = pd.read_csv('featuresRankedRFC_550.csv')
```

Mutual Information Gain

```
X = featuresMIC.iloc[:,0:20]
y = featuresMIC['Playoffs']
```

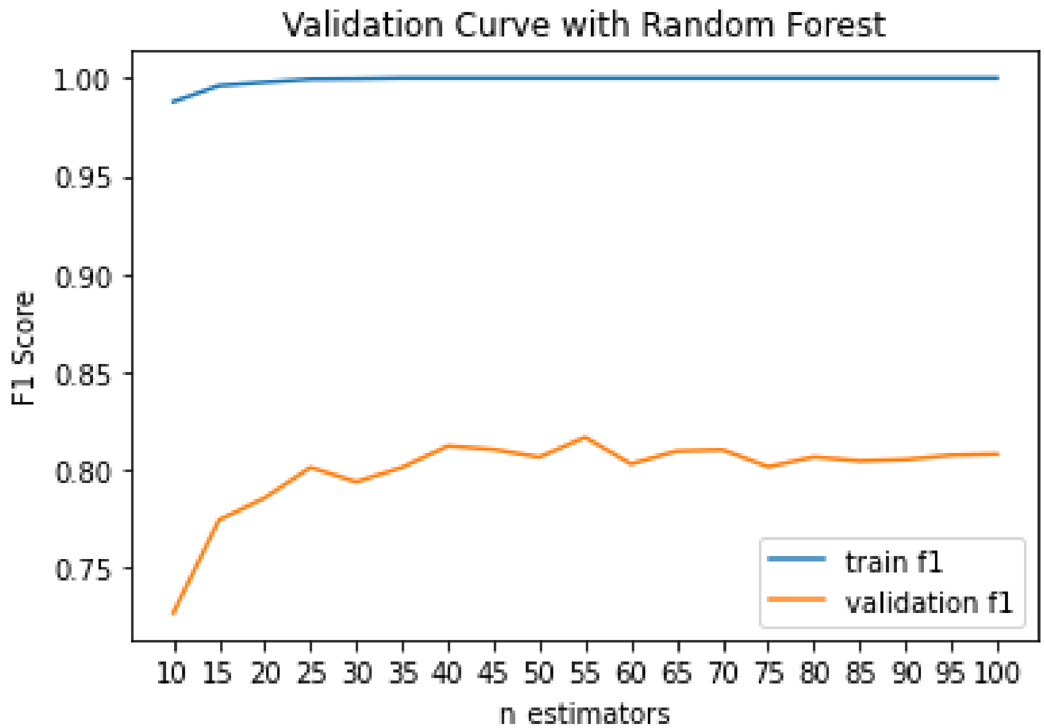
```
featuresMIC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G      1Run        SV       R/G      BB_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.559387  0.302094  0.339394  0.429180
0.45
1  0.637681  0.32  0.000000  0.390805  0.194577  0.133333  0.154861
0.65
2  0.231884  0.56  0.300995  0.641762  0.265422  0.363636  0.429180
0.75
3  0.275362  0.36  0.161692  0.618774  0.100396  0.000000  0.688073
0.40
4  0.594203  0.46  0.208955  0.392720  0.390013  0.442424  0.204570
0.60

   Under500    DefEff        PA      Rtot       H_P       SOS
WP  \
0  0.402703  0.602041  0.594126  0.391473  0.622353  0.545455
0.687796
1  0.389189  1.000000  0.356231  0.841085  0.181001  0.454545
0.262041
2  0.385135  0.581633  0.523644  0.372093  0.464920  0.454545
0.221357
3  0.271622  0.867347  0.279588  0.569767  0.159225  0.454545
0.435667
4  0.528378  0.663265  0.566620  0.441860  0.482468  0.363636
0.218771
```

```
       PAge        2B        BB       LOB      HR_P
0  0.405941  0.151402  0.465585  0.695640  0.418914
1  0.465347  0.191710  0.410213  0.400425  0.123964
2  0.475248  0.361411  0.440077  0.555105  0.189573
3  0.336634  0.078143  0.021753  0.387544  0.142855
4  0.455446  0.242631  0.540521  0.517147  0.233241
```

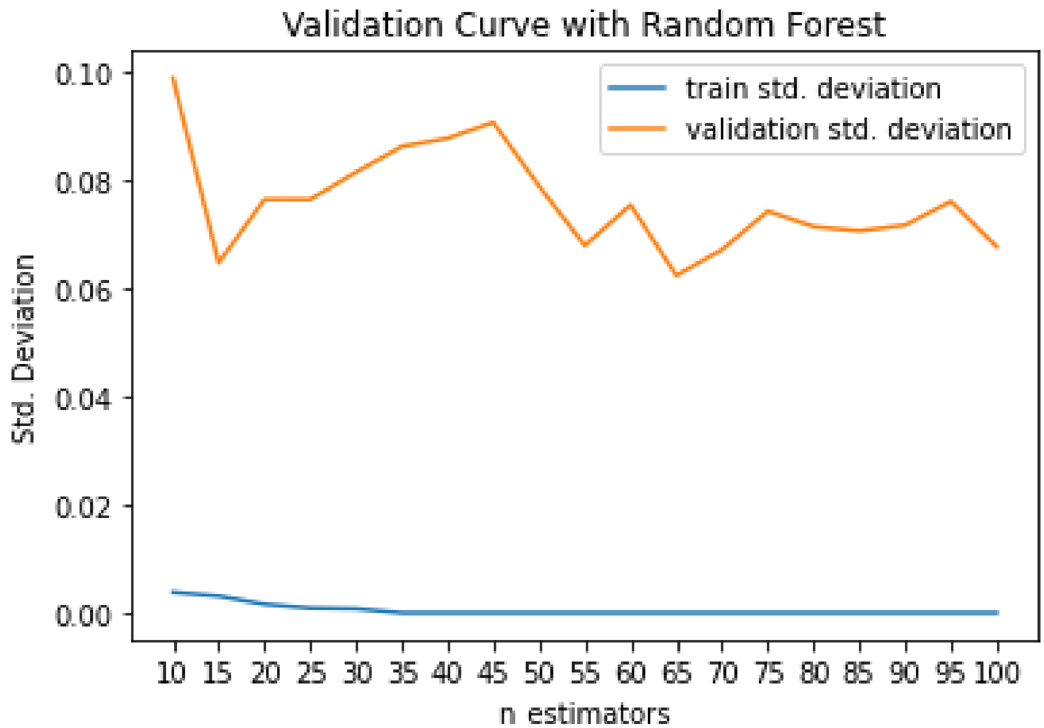
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

n_estimators = np.arange(10,105,5)

train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("F1 Score")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_mean, label="train f1")
plt.plot(n_estimators, valid_scores_mean, label="validation f1")

plt.legend()
plt.show()
```

```
[0.72709696 0.77451802 0.78577794 0.80138221 0.79396098 0.80135448
 0.81224439 0.81048299 0.80654699 0.81671196 0.80313006 0.80964252
 0.81030621 0.80171985 0.80666674 0.80462619 0.80534236 0.80754204
 0.80807864]
```

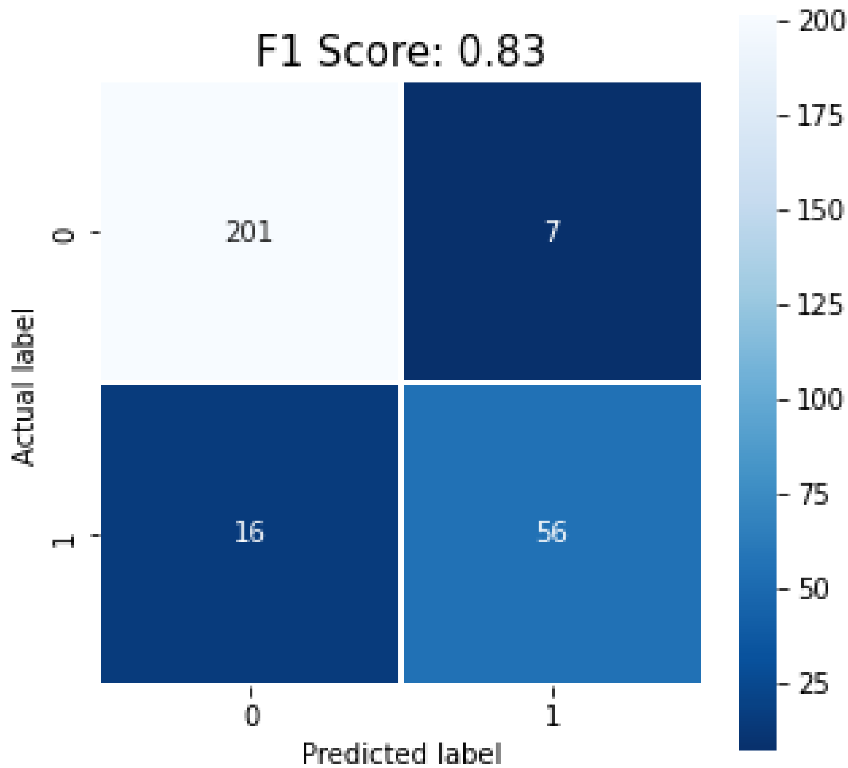



```
train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_std, label="train std. deviation")
plt.plot(n_estimators, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.09893717 0.06475081 0.0764921  0.07648663 0.08151771 0.08627151
 0.08778752 0.09073355 0.07895281 0.06795139 0.07545484 0.06241328
 0.06721315 0.07429631 0.07146142 0.07066385 0.07171814 0.07613502
 0.06771855]
```




```
RF = RandomForestClassifier(random_state=1, n_estimators=100)
RF.fit(X_train, y_train)
y_pred = RF.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1 Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  56
FP:  7
TN:  201
FN:  16

              precision    recall  f1-score   support

           0       0.93      0.97      0.95       208
           1       0.89      0.78      0.83        72

    accuracy                           0.92       280
   macro avg       0.91      0.87      0.89       280
weighted avg       0.92      0.92      0.92       280
```

ANOVA F-test

```
X = featuresFC.iloc[:,0:20]
y = featuresFC['Playoffs']
```

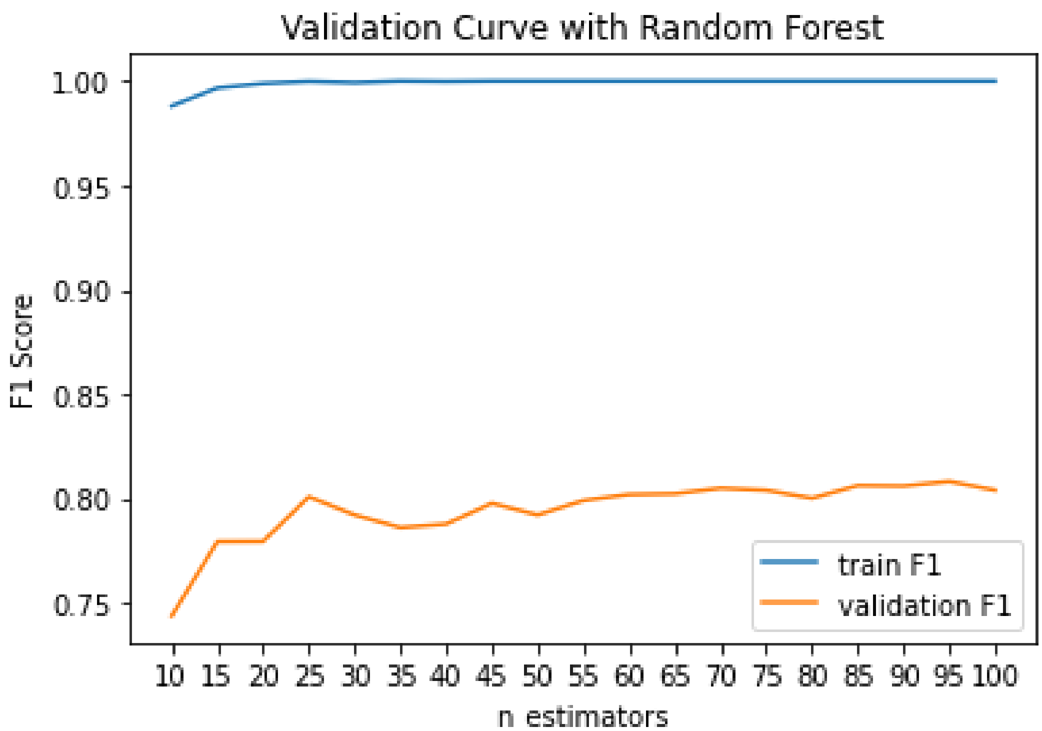
```
featuresFC.iloc[:,0:20].head()
```

```
       ERA+  OPS+      RA/G       R/G      1Run        SV       H_P
#a-tA-S  \
0  0.173913  0.44  0.477612  0.339394  0.559387  0.302094  0.622353
0.45
1  0.637681  0.32  0.000000  0.133333  0.390805  0.194577  0.181001
0.65
2  0.231884  0.56  0.300995  0.363636  0.641762  0.265422  0.464920
0.75
3  0.275362  0.36  0.161692  0.000000  0.618774  0.100396  0.159225
0.40
4  0.594203  0.46  0.208955  0.442424  0.392720  0.390013  0.482468
0.60

       BB_P        BB      Rtot    DefEff        PA  Under500
HR  \
0  0.429180  0.465585  0.391473  0.602041  0.594126  0.402703
0.380797
1  0.154861  0.410213  0.841085  1.000000  0.356231  0.389189
0.213898
2  0.429180  0.440077  0.372093  0.581633  0.523644  0.385135
0.303797
3  0.688073  0.021753  0.569767  0.867347  0.279588  0.271622
0.126699
4  0.204570  0.540521  0.441860  0.663265  0.566620  0.528378
0.335165

         SF         E        IP       SOS     IBB_P
0  0.358281  0.743973  0.678982  0.545455  0.495551
1  0.398932  0.349282  0.697641  0.454545  0.471563
2  0.538171  0.558574  0.710241  0.454545  0.441491
3  0.124424  0.444482  0.680184  0.454545  0.477531
4  0.443268  0.566802  0.752545  0.363636  0.501326
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("F1 Score")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_mean, label="train F1")
plt.plot(n_estimators, valid_scores_mean, label="validation F1")
```

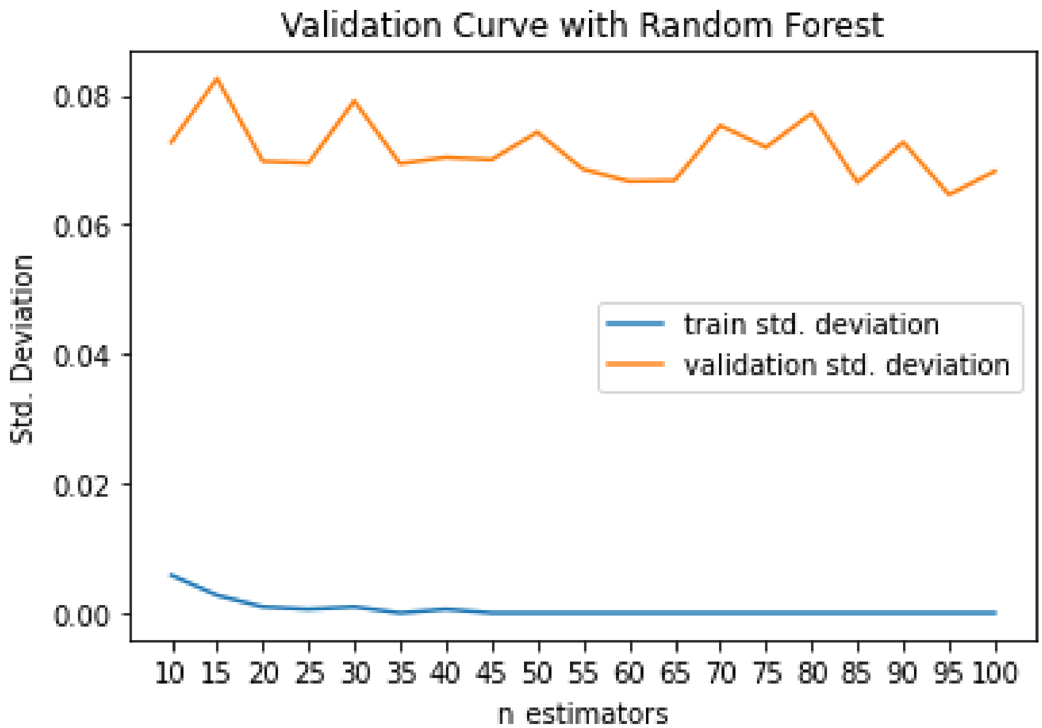
```
plt.legend()
plt.show()
```

```
[0.74389622 0.7795246  0.77958606 0.80092258 0.79224219 0.78623199
 0.78787656 0.79781815 0.79226751 0.79914508 0.80206373 0.80240211
 0.80504349 0.8040459  0.80025461 0.80632611 0.80609527 0.80830148
 0.80408189]
```

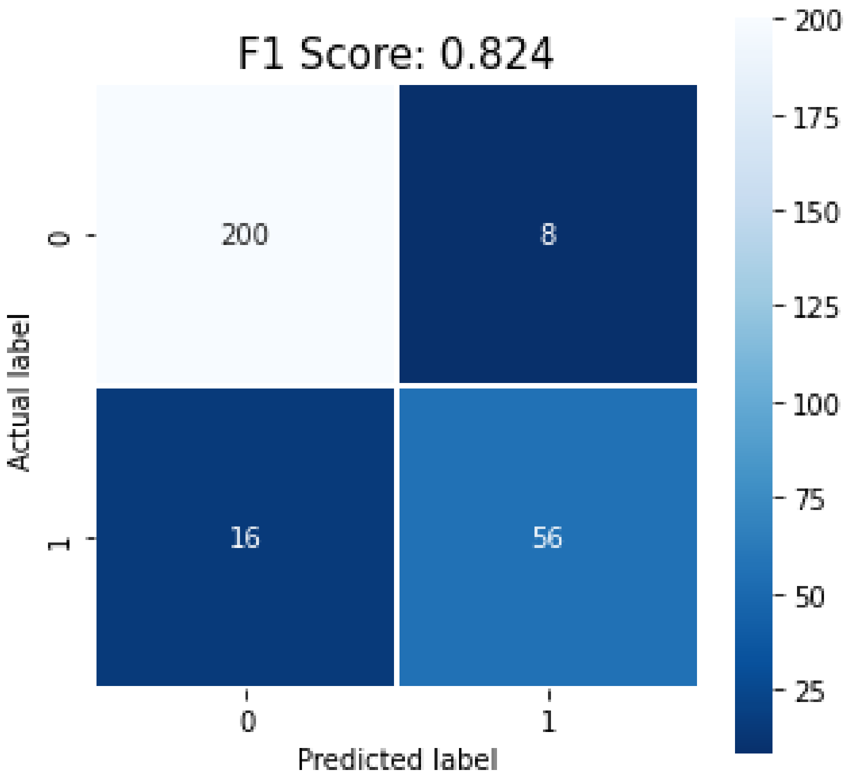



```
train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_std, label="train std. deviation")
plt.plot(n_estimators, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.0727835  0.08262158 0.06983619 0.06956043 0.0791565  0.06946607
 0.07043088 0.0700737  0.07430854 0.06859519 0.06673675 0.066847
 0.07534981 0.07197964 0.07725169 0.06655199 0.07277599 0.06466768
 0.06827151]
```




```
RF = RandomForestClassifier(random_state=1, n_estimators=95)
RF.fit(X_train, y_train)
y_pred = RF.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1 Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  56
FP:  8
TN:  200
FN:  16

              precision    recall  f1-score   support

           0       0.93      0.96      0.94       208
           1       0.88      0.78      0.82        72

    accuracy                           0.91       280
   macro avg       0.90      0.87      0.88       280
weighted avg       0.91      0.91      0.91       280
```

Random Forest

```
X = featuresRFC.iloc[:,0:20]
y = featuresRFC['Playoffs']
```

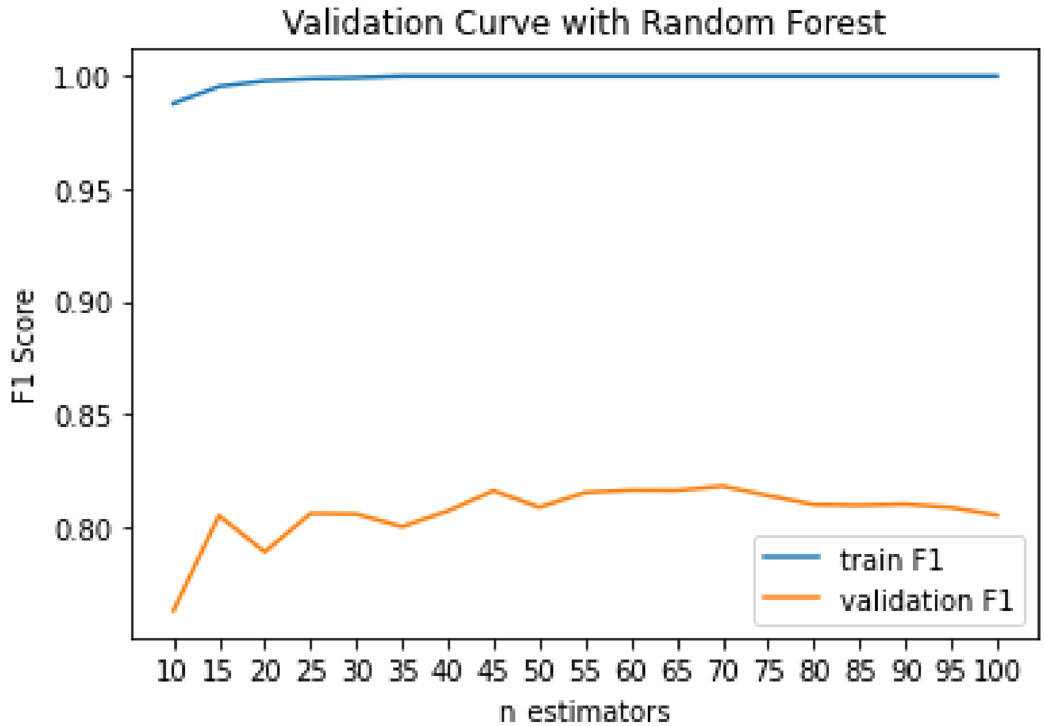
```
featuresRFC.iloc[:,0:20].head()
```

```
       ERA+      RA/G      1Run       R/G  OPS+        SV       H_P
BB_P  \
0  0.173913  0.477612  0.559387  0.339394  0.44  0.302094  0.622353
0.429180
1  0.637681  0.000000  0.390805  0.133333  0.32  0.194577  0.181001
0.154861
2  0.231884  0.300995  0.641762  0.363636  0.56  0.265422  0.464920
0.429180
3  0.275362  0.161692  0.618774  0.000000  0.36  0.100396  0.159225
0.688073
4  0.594203  0.208955  0.392720  0.442424  0.46  0.390013  0.482468
0.204570

         BB    DefEff      Rtot        HR  #a-tA-S        IP       SOS
\
0  0.465585  0.602041  0.391473  0.380797     0.45  0.678982  0.545455

1  0.410213  1.000000  0.841085  0.213898     0.65  0.697641  0.454545

2  0.440077  0.581633  0.372093  0.303797     0.75  0.710241  0.454545

3  0.021753  0.867347  0.569767  0.126699     0.40  0.680184  0.454545

4  0.540521  0.663265  0.441860  0.335165     0.60  0.752545  0.363636


         PA      HR_P        SF        2B  Under500
0  0.594126  0.418914  0.358281  0.151402  0.402703
1  0.356231  0.123964  0.398932  0.191710  0.389189
2  0.523644  0.189573  0.538171  0.361411  0.385135
3  0.279588  0.142855  0.124424  0.078143  0.271622
4  0.566620  0.233241  0.443268  0.242631  0.528378
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='f1', cv=10)
train_scores_mean = np.mean(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)

print(valid_scores_mean)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("F1 Score")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_mean, label="train F1")
plt.plot(n_estimators, valid_scores_mean, label="validation F1")
```

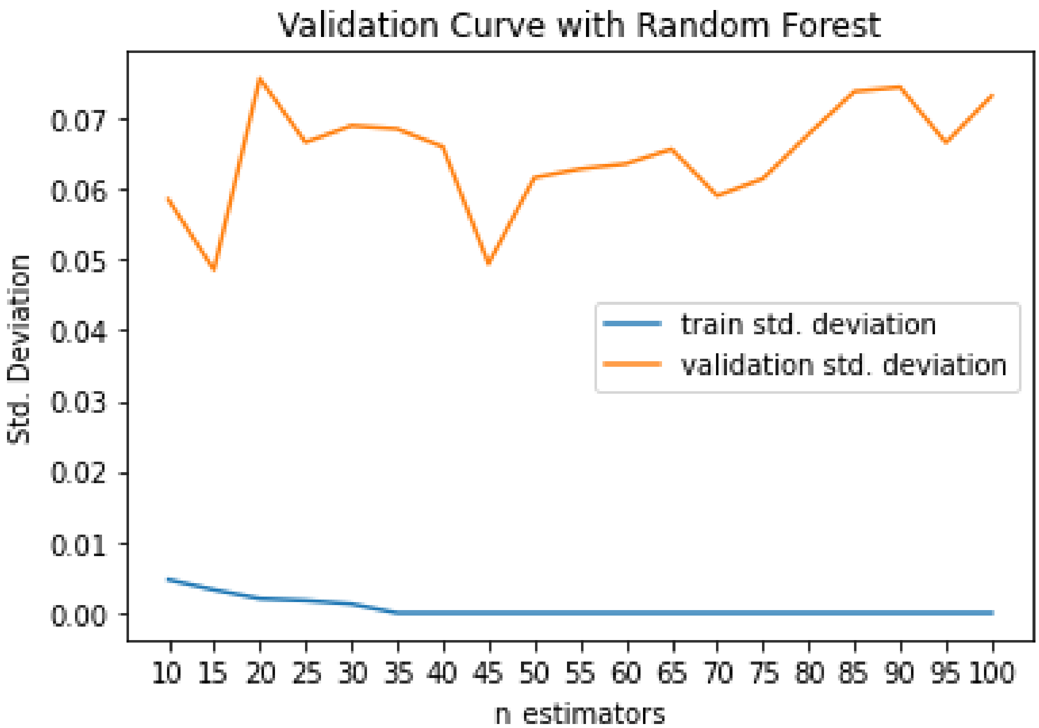
```
plt.legend()
plt.show()
```

```
[0.76250923 0.8048455  0.78867351 0.80576499 0.80555535 0.7999216
 0.80688917 0.8158639  0.80853202 0.81508851 0.81608986 0.81599237
 0.81792287 0.81370436 0.80971346 0.80940928 0.80992921 0.80846188
 0.80499108]
```

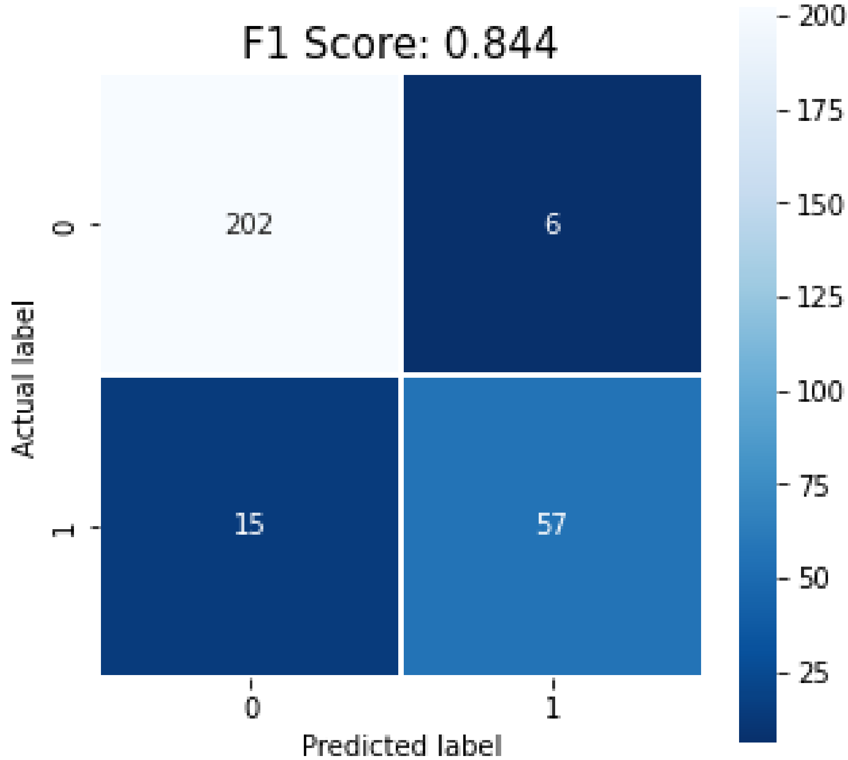



```
train_scores, valid_scores =
validation_curve(RandomForestClassifier(random_state=1), X_train,
y_train, param_name="n_estimators", param_range=n_estimators,
scoring='f1', cv=10)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

print(valid_scores_std)
plt.title("Validation Curve with Random Forest")
plt.xlabel("n_estimators")
plt.ylabel("Std. Deviation")
plt.xticks(np.arange(10,105,step=5))
plt.plot(n_estimators, train_scores_std, label="train std. deviation")
plt.plot(n_estimators, valid_scores_std, label="validation std.
deviation")

plt.legend()
plt.show()
```

```
[0.05851683 0.04855409 0.0756404  0.0666062  0.0689507  0.06851683
 0.06596398 0.04942261 0.06159713 0.06283382 0.06356071 0.0656398
 0.05905561 0.06146309 0.06778588 0.07381071 0.07440108 0.06654187
 0.07317817]
```




```
RF = RandomForestClassifier(random_state=1, n_estimators=70)
RF.fit(X_train, y_train)
y_pred = RF.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True,
cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'F1 Score: {0}'.format(round(f1_score(y_test,
y_pred),3))
plt.title(all_sample_title, size = 15);
```




```
tn, fp, fn, tp=cm.ravel()
print ("TP: ", tp,"\n" "FP: ",fp,"\n" "TN: ", tn,"\n" "FN: ",fn)
print()
print(classification_report(y_test, y_pred))
```

```
TP:  57
FP:  6
TN:  202
FN:  15

              precision    recall  f1-score   support

           0       0.93      0.97      0.95       208
           1       0.90      0.79      0.84        72

    accuracy                           0.93       280
   macro avg       0.92      0.88      0.90       280
weighted avg       0.92      0.93      0.92       280
```