2020 MLB Postseason Participants Prediction

Introduction

2020 MLB season has already been hard to predict given it's short nature. This research aims to build a perfect model using team batting stats of the past eight years and then use the model to predict which teams' batting stats on 08/13/2020 is worthy of getting into postseason on a traditional 10-team postseason format.

Methods

Model was built using the combination of 16 team regular season stats:PA, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB from 2012-2019 and whether that corresponding team went into postseason or not.

Using SQL Server and Python

logistic regression
k-nearest neighbors
support vector machine
decision tree
random forest
gradient boosting
XGBoost

Step 1: Import data

import regular season stats from MLB teams who got into postseason during 2012-2019

items include Tm, PA, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB

total rows are 8(years)*10(teams each year)=80

In [1]:

```
import pandas as pd
import pyodbc

sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                            SERVER=ALLENHO\MSSQLSERVER002;
                            DATABASE=Playoffbound;
                            Trusted_Connection=yes''')
query = '''
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['19B$']
where Tm in ('WSN','LAD','MIL','ATL','STL','HOU','NYY','MIN','TBR','OAK')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['18B$']
where Tm in ('BOS','LAD','MIL','ATL','CHC','HOU','NYY','CLE','COL','OAK')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['17B$']
where Tm in ('BOS','LAD','COL','WSN','CHC','HOU','NYY','CLE','ARI','MIN')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['16B$']
where Tm in ('TOR','CLE','BOS','BAL','TEX','NYM','CHC','LAD','WSN','SFG')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['15B$']
where Tm in ('TOR','KCR','HOU','NYY','TEX','NYM','CHC','LAD','STL','PIT')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['14B$']
where Tm in ('BAL','KCR','OAK','LAA','DET','WSN','STL','LAD','PIT','SFG')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['13B$']
```

```
where Tm in ('BOS','TBR','OAK','CLE','DET','ATL','STL','LAD','PIT','CIN')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['12B$']
where Tm in ('TEX','BAL','OAK','NYY','DET','ATL','STL','SFG','WSN','CIN')
'''
df = pd.read_sql(query, sql_conn)

#stored as df_post
df_post = df
```

import regular season stats from MLB teams who DIDN'T get into postseason during 2012-2019
items are the same as above
total rows are 8(years)*20(teams each year)=160

In [2]:

```
sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                             SERVER=ALLENHO\MSSQLSERVER002;
                             DATABASE=Playoffbound;
                             Trusted_Connection=yes''')
query = '''
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['19B$']
where Tm is not null and Tm not in ('WSN','LAD','MIL','ATL','STL','HOU','NYY','MIN','TBR','OAK', 'LgAvg')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['18B$']
where Tm is not null and Tm not in ('BOS','LAD','MIL','ATL','CHC','HOU','NYY','CLE','COL','OAK', 'LgAvg')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['17B$']
where Tm is not null and Tm not in ('BOS','LAD','COL','WSN','CHC','HOU','NYY','CLE','ARI','MIN', 'LgAvg')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['16B$']
where Tm is not null and Tm not in ('TOR','CLE','BOS','BAL','TEX','NYM','CHC','LAD','WSN','SFG', 'LgAvg')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['15B$']
where Tm is not null and Tm not in ('TOR','KCR','HOU','NYY','TEX','NYM','CHC','LAD','STL','PIT', 'LgAvg')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['14B$']
where Tm is not null and Tm not in ('BAL','KCR','OAK','LAA','DET','WSN','STL','LAD','PIT','SFG', 'LgAvg')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['13B$']
where Tm is not null and Tm not in ('BOS','TBR','OAK','CLE','DET','ATL','STL','LAD','PIT','CIN', 'LgAvg')
UNION ALL
select Tm, PA, R, H, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB
from [dbo].['12B$']
where Tm is not null and Tm not in ('TEX','BAL','OAK','NYY','DET','ATL','STL','SFG','WSN','CIN', 'LgAvg')
'''
df = pd.read_sql(query, sql_conn)

#stored as df_npost
df_npost = df

#add each dataframe a new column named POST, which imply whether the team made the postseason that year
df_post['POST']= 1
df_npost['POST']= 0

#append two dataframes together
df_com=df_post.append(df_npost)
```

```
#take a look at the table we got
print(df_com)
```

```
      Tm      PA      R       H     HR    RBI     SB    CS     BB      SO  \
0    ATL  6302.0  855.0  1432.0  249.0  824.0   89.0  28.0  619.0  1467.0
1    HOU  6394.0  920.0  1538.0  288.0  891.0   67.0  27.0  645.0  1166.0
2    LAD  6282.0  886.0  1414.0  279.0  861.0   57.0  10.0  607.0  1356.0
3    MIL  6309.0  769.0  1366.0  250.0  744.0  101.0  25.0  629.0  1563.0
4    MIN  6392.0  939.0  1547.0  307.0  906.0   28.0  21.0  525.0  1334.0
..   ...     ...    ...     ...    ...    ...    ...   ...    ...     ...
155  PIT  6014.0  651.0  1313.0  170.0  620.0   73.0  52.0  444.0  1354.0
156  SDP  6112.0  651.0  1339.0  121.0  610.0  155.0  46.0  539.0  1238.0
157  SEA  6057.0  619.0  1285.0  149.0  584.0  104.0  35.0  466.0  1259.0
158  TBR  6105.0  697.0  1293.0  175.0  665.0  134.0  44.0  571.0  1323.0
159  TOR  6094.0  716.0  1346.0  198.0  677.0  123.0  41.0  473.0  1251.0

        BA    OBP    SLG    OPS      TB  POST
0    0.258  0.336  0.452  0.789  2514.0     1
1    0.274  0.352  0.495  0.848  2781.0     1
2    0.257  0.338  0.472  0.810  2593.0     1
3    0.246  0.329  0.438  0.767  2429.0     1
4    0.270  0.338  0.494  0.832  2832.0     1
..     ...    ...    ...    ...     ...   ...
155  0.243  0.304  0.395  0.699  2138.0     0
156  0.247  0.319  0.380  0.699  2060.0     0
157  0.234  0.296  0.369  0.665  2027.0     0
158  0.240  0.317  0.394  0.711  2128.0     0
159  0.245  0.309  0.407  0.716  2231.0     0

[240 rows x 16 columns]
```

Take a brief look at the summary

In [3]:

```
print(df_com.describe())
```

```
                PA           R            H          HR         RBI  \
count   240.000000  240.000000   240.000000  240.000000  240.000000
mean   6158.929167  712.929167  1397.520833  178.191667  679.350000
std      97.271646   77.051138    71.511707   40.684477   75.512603
min    5905.000000  513.000000  1199.000000   95.000000  485.000000
25%    6085.250000  652.500000  1346.000000  148.000000  622.500000
50%    6154.500000  709.500000  1390.500000  174.500000  675.000000
75%    6224.250000  761.000000  1446.000000  211.000000  728.000000
max    6475.000000  943.000000  1625.000000  307.000000  906.000000

              SB          CS          BB           SO          BA         OBP  \
count  240.00000  240.000000  240.000000   240.000000  240.000000  240.000000
mean    87.53750   33.195833  499.750000  1296.412500    0.252933    0.319117
std     28.46018    8.737884   63.716652   130.141413    0.010588    0.011853
min     19.00000   10.000000  375.000000   973.000000    0.226000    0.292000
25%     66.00000   27.000000  452.000000  1204.000000    0.245000    0.311000
50%     86.00000   33.000000  500.000000  1290.500000    0.252000    0.319000
75%    105.25000   38.250000  545.500000  1384.500000    0.260000    0.327000
max    181.00000   61.000000  656.000000  1595.000000    0.283000    0.352000

              SLG         OPS           TB        POST
count  240.000000  240.000000   240.000000  240.000000
mean     0.409925    0.729004  2264.937500    0.333333
std      0.026815    0.036499   163.359296    0.472390
min      0.335000    0.627000  1810.000000    0.000000
25%      0.391000    0.702750  2152.500000    0.000000
50%      0.409000    0.728000  2256.500000    0.000000
75%      0.428250    0.752000  2364.000000    1.000000
max      0.495000    0.848000  2832.000000    1.000000
```

Take a brief look at the correlation table

In [4]:

```
df_corr = df_com.corr()
```

```
print(df_corr)
```

```
            PA         R         H        HR       RBI        SB        CS  \
PA    1.000000  0.728568  0.603097  0.404630  0.724551 -0.075825 -0.275590
R     0.728568  1.000000  0.627253  0.757987  0.996604 -0.077307 -0.286587
H     0.603097  0.627253  1.000000  0.168541  0.622822  0.010873 -0.096667
HR    0.404630  0.757987  0.168541  1.000000  0.773527 -0.226804 -0.338698
RBI   0.724551  0.996604  0.622822  0.773527  1.000000 -0.101979 -0.300592
SB   -0.075825 -0.077307  0.010873 -0.226804 -0.101979  1.000000  0.563940
CS   -0.275590 -0.286587 -0.096667 -0.338698 -0.300592  0.563940  1.000000
BB    0.672470  0.572770  0.031033  0.490024  0.570726 -0.060325 -0.240242
SO    0.004639  0.060778 -0.417156  0.423937  0.068943 -0.128711 -0.030335
BA    0.512469  0.614537  0.978840  0.149331  0.609907  0.044830 -0.045988
OBP   0.790239  0.828347  0.741023  0.438346  0.823876 -0.004032 -0.162219
SLG   0.582243  0.922851  0.598798  0.864843  0.931042 -0.141123 -0.270421
OPS   0.685823  0.948734  0.681851  0.779146  0.953297 -0.105815 -0.251054
TB    0.658199  0.926030  0.685324  0.816982  0.933121 -0.148223 -0.286691
POST  0.461904  0.488744  0.339784  0.281205  0.481147 -0.079983 -0.224696

            BB        SO        BA       OBP       SLG       OPS        TB  \
PA    0.672470  0.004639  0.512469  0.790239  0.582243  0.685823  0.658199
R     0.572770  0.060778  0.614537  0.828347  0.922851  0.948734  0.926030
H     0.031033 -0.417156  0.978840  0.741023  0.598798  0.681851  0.685324
HR    0.490024  0.423937  0.149331  0.438346  0.864843  0.779146  0.816982
RBI   0.570726  0.068943  0.609907  0.823876  0.931042  0.953297  0.933121
SB   -0.060325 -0.128711  0.044830 -0.004032 -0.141123 -0.105815 -0.148223
CS   -0.240242 -0.030335 -0.045988 -0.162219 -0.270421 -0.251054 -0.286691
BB    1.000000  0.236172  0.035724  0.647519  0.440775  0.534829  0.400129
SO    0.236172  1.000000 -0.463401 -0.188190  0.143080  0.044561  0.101463
BA    0.035724 -0.463401  1.000000  0.769802  0.595888  0.688866  0.655888
OBP   0.647519 -0.188190  0.769802  1.000000  0.735855  0.866479  0.746457
SLG   0.440775  0.143080  0.595888  0.735855  1.000000  0.975517  0.985919
OPS   0.534829  0.044561  0.688866  0.866479  0.975517  1.000000  0.968787
TB    0.400129  0.101463  0.655888  0.746457  0.985919  0.968787  1.000000
POST  0.431907 -0.105900  0.356648  0.546741  0.405289  0.477010  0.403341

          POST
PA    0.461904
R     0.488744
H     0.339784
HR    0.281205
RBI   0.481147
SB   -0.079983
CS   -0.224696
BB    0.431907
SO   -0.105900
BA    0.356648
OBP   0.546741
SLG   0.405289
OPS   0.477010
TB    0.403341
POST  1.000000
```

Step 2: Train the Models

There's one other consideration worth making -- the distribution of outcomes is somewhat imbalanced. Teams getting into postseason each year are less than those not. I tried an oversampling technique to see how it affected the models. Oversampling techniques are usually applied to datasets where an outcome is significantly less common. That might be a little bit of a stretch for this scenario, but I think it's worth at least checking if an oversampling technique would help. I tried fitting my different models twice -- with and without oversampling. For oversampling, I used SMOTE (synthetic minority oversampling technique).

use 4 metrics to evaluate the models, which together should give a good picture of the best overall model:

F1 score (weighted by instances of each label) ROC AUC (computed by label and weighted by frequency) balanced accuracy (for imbalanced datasets) log loss

In [5]:

```
from sklearn.preprocessing import StandardScaler

# split data into X and y
X = df_com.loc[:,'PA':'TB']
Y = df_com.loc[:,'POST']
```

```python
# scale and center numeric columns
X = StandardScaler().fit_transform(X)
```

In [6]:

```python
from imblearn.pipeline import Pipeline
from sklearn.model_selection import cross_validate
from sklearn.metrics import f1_score, accuracy_score, log_loss, roc_auc_score, make_scorer
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier


# scoring metrics
scoring = {
    'f1_weighted': 'f1_weighted',
    'accuracy': 'balanced_accuracy',
    'roc_auc': 'roc_auc_ovr_weighted',
    'neg_log_loss': 'neg_log_loss'
    }

# for results df
eval_cols = [
    'models',
    'F1 Score',
    'Balanced Accuracy',
    'ROC AUC',
    'Neg Log Loss'
    ]

# define classifier models
classifiers = [
    LogisticRegression(multi_class='multinomial', max_iter=10000),
    KNeighborsClassifier(n_neighbors=50),
    SVC(probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    GradientBoostingClassifier(),
    XGBClassifier()
    ]

# classifier names
clf_names = [
    'Logistic Regression',
    'KNN',
    'SVM',
    'Decision Tree',
    'Random Forest',
    'Gradient Boosting',
    'XGBClassifier'
    ]
```

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\externals\six.py:31: FutureWarning: The module
is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for
Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sk
learn.neighbors.base module is  deprecated in version 0.22 and will be removed in version 0.24.
The corresponding classes / functions should instead be imported from sklearn.neighbors. Anything
that cannot be imported from sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
```

In [7]:

```python
from imblearn.over_sampling import SMOTE
import time as time
import numpy as np

f1, acc, roc_auc, log_loss = [], [], [], []
for clf, clf_nm in zip(classifiers, clf_names):
```

```python
    # setup pipeline to oversample, then fit model
    pipe = Pipeline([
        ('smote', SMOTE()),
        ('classify', clf)
        ])

    start = time.time()

    # cross-validate 5 times
    res_smote = cross_validate(pipe, X, Y, cv=5, scoring=scoring)
    results_smote = pd.DataFrame(res_smote)

    stop = time.time()

    print('Time to cross-validate %s = %0.3f min.' % (clf_nm, (stop - start) / 60))

    # save average scores
    f1.append(np.mean(results_smote.test_f1_weighted))
    acc.append(np.mean(results_smote.test_accuracy))
    roc_auc.append(np.mean(results_smote.test_roc_auc))
    log_loss.append(np.mean(results_smote.test_neg_log_loss))

# save results to df
model_eval_smote = pd.DataFrame(data=zip(clf_names, f1, acc, roc_auc, log_loss),
                                columns=eval_cols)

display(model_eval_smote)
```

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
```

```
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

```
Time to cross-validate Logistic Regression = 0.004 min.
Time to cross-validate KNN = 0.002 min.
```

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

```
Time to cross-validate SVM = 0.002 min.
Time to cross-validate Decision Tree = 0.002 min.
```

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

```
Time to cross-validate Random Forest = 0.026 min.
```

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
```

```
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

```
Time to cross-validate Gradient Boosting = 0.019 min.
```

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and will be
removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

```
Time to cross-validate XGBClassifier = 0.009 min.
```

	models	F1 Score	Balanced Accuracy	ROC AUC	Neg Log Loss
0	Logistic Regression	0.718629	0.721875	0.827734	-0.536031
1	KNN	0.658151	0.684375	0.824414	-0.578825
2	SVM	0.647821	0.634375	0.742578	-0.609196
3	Decision Tree	0.671690	0.637500	0.637500	-11.225236
4	Random Forest	0.727879	0.712500	0.813672	-0.545912
5	Gradient Boosting	0.708470	0.690625	0.780078	-0.679895
6	XGBClassifier	0.734760	0.721875	0.785547	-0.617131

In [8]:

```python
import numpy as np
import time as time
f1, acc, roc_auc, log_loss = [], [], [], []
for clf, clf_nm in zip(classifiers, clf_names):

    start = time.time()

    # cross-validate 5 times
    res = cross_validate(clf, X, Y, cv=5, scoring=scoring)
    results = pd.DataFrame(res)

    stop = time.time()

    print('Time to cross-validate %s = %0.3f min.' % (clf_nm, (stop - start) / 60))

    # save average scores
    f1.append(np.mean(results.test_f1_weighted))
    acc.append(np.mean(results.test_accuracy))
    roc_auc.append(np.mean(results.test_roc_auc))
    log_loss.append(np.mean(results.test_neg_log_loss))
```

```
# save results to df
model_eval = pd.DataFrame(data=zip(clf_names, f1, acc, roc_auc, log_loss),
                          columns=eval_cols)

display(model_eval)
```

```
Time to cross-validate Logistic Regression = 0.003 min.
Time to cross-validate KNN = 0.002 min.
Time to cross-validate SVM = 0.002 min.
Time to cross-validate Decision Tree = 0.001 min.
Time to cross-validate Random Forest = 0.035 min.
Time to cross-validate Gradient Boosting = 0.018 min.
Time to cross-validate XGBClassifier = 0.005 min.
```

	models	F1 Score	Balanced Accuracy	ROC AUC	Neg Log Loss
0	Logistic Regression	0.748284	0.715625	0.828516	-0.491554
1	KNN	0.711105	0.675000	0.830469	-0.520010
2	SVM	0.731454	0.684375	0.751563	-0.547907
3	Decision Tree	0.674239	0.643750	0.643750	-10.937406
4	Random Forest	0.712223	0.675000	0.803516	-0.539415
5	Gradient Boosting	0.723963	0.693750	0.792578	-0.626625
6	XGBClassifier	0.734542	0.703125	0.785547	-0.602285

It looks like not performing oversampling is the way to go here. The non-SMOTE'd data built models that slightly outperformed the SMOTE'd models. sklearn defines balanced accuracy as the average of recall on each class. Recall only considers false negatives and true positives -- and since SMOTE creates more data to help a model recognize minority classes, it should reduce false negatives. So, it's no surprise the SMOTE'd data performs better in balanced accuracy.

Overall, the Logistic Regression model was the best.

Step 3: Make Predictions with Logistic Regression model

import 2020 team stats as of 08/14/2020 normalized to 162 games, try to see which teams' stats on 08/13/2020 is worthy of getting into postseason on a traditional 10-team postseason format.

In [9]:

```
model = LogisticRegression(multi_class='multinomial')
model.fit(X, Y)
```

Out[9]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='multinomial', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [10]:

```
model.coef_
```

Out[10]:

```
array([[ 0.20563564,  0.35558267, -0.43472342, -0.07174802, -0.16508224,
        -0.01143693, -0.18435707,  0.00724   , -0.18083768,  0.20025544,
         0.54154686,  0.04143439,  0.35595693, -0.26669742]])
```

In [11]:

```
df_2020=pd.read_excel(r'C:\Users\allen\Desktop\Baseball research\Postseason or bust\2020 projection for 0813.xlsx')
```

```
DF_2020=df_2020.loc[:,['PA162', 'R162', 'H162', 'HR162', 'RBI162', 'SB162', 'CS162', 'BB162', 'SO16
2', 'BA', 'OBP', 'SLG', 'OPS', 'TB162']]
DF_2020['PA']=DF_2020['PA162']
DF_2020['R']=DF_2020['R162']
DF_2020['H']=DF_2020['H162']
DF_2020['HR']=DF_2020['HR162']
DF_2020['RBI']=DF_2020['RBI162']
DF_2020['SB']=DF_2020['SB162']
DF_2020['CS']=DF_2020['CS162']
DF_2020['BB']=DF_2020['BB162']
DF_2020['SO']=DF_2020['SO162']
DF_2020['TB']=DF_2020['TB162']
DF_2020=DF_2020.loc[:, ['PA','R','H','HR','RBI','SB','CS','BB','SO','BA','OBP','SLG','OPS','TB']]
print(DF_2020.head())
```

```
            PA           R            H          HR         RBI         SB  \
0  6096.315789  750.315789  1347.157895  127.894737  707.684211  34.105263
1  5977.800000  842.400000  1312.200000  226.800000  826.200000  64.800000
2  6111.000000  864.000000  1467.000000  243.000000  846.000000  63.000000
3  6096.315789  724.736842  1415.368421  196.105263  682.105263  34.105263
4  6176.250000  840.375000  1296.000000  232.875000  789.750000  50.625000

          CS          BB           SO     BA    OBP    SLG    OPS           TB
0  25.578947  477.473684  1219.263158  0.245  0.315  0.382  0.697  2097.473684
1  24.300000  526.500000  1644.300000  0.244  0.316  0.437  0.753  2349.000000
2  45.000000  513.000000  1305.000000  0.269  0.337  0.473  0.810  2583.000000
3  25.578947  426.315789  1492.105263  0.254  0.316  0.422  0.738  2353.263158
4  10.125000  658.125000  1620.000000  0.244  0.344  0.431  0.776  2288.250000
```

In [12]:

```
DF_2020_1 = StandardScaler().fit_transform(DF_2020)
predictions_2020_proba = model.predict_proba(DF_2020_1)
predictions_2020_pred = model.predict(DF_2020_1)
data_result = {'Team': df_2020['Tm'],
        'Probability': predictions_2020_proba[:,1]*100,
              'Prediction': predictions_2020_pred}
prediction_table = pd.DataFrame(data_result)
print(prediction_table)
```

```
   Team  Probability  Prediction
0   ARI    31.150825           0
1   ATL    17.553153           0
2   BAL    53.163528           1
3   BOS    12.661454           0
4   CHC    83.040529           1
5   CHW    17.058149           0
6   CIN    43.136621           0
7   CLE    11.326538           0
8   COL    70.816525           1
9   DET    13.061752           0
10  HOU    60.793937           1
11  KCR     3.868289           0
12  LAA    50.827103           1
13  LAD    57.108112           1
14  MIA    27.250176           0
15  MIL     4.160920           0
16  MIN    30.270392           0
17  NYM    77.672358           1
18  NYY    87.175147           1
19  OAK    47.563303           0
20  PHI    90.666869           1
21  PIT     1.140761           0
22  SDP    24.507434           0
23  SEA    10.759323           0
24  SFG    14.734669           0
25  STL     0.988297           0
26  TBR    76.495594           1
27  TEX     8.408854           0
28  TOR     1.349251           0
29  WSN     5.305377           0
```

In order get more accurate result. I decided to adjust my model to only include the variables that are more significantly correlated to predicting the postseason birth predicting the postseason birth.

The Logistic Regression model from sklearn doesn't provide p-value automatically, so I turned to the logistic regression model from statsmodel to see which variables are less significantly correlated to predicting the postseason birth.

In [13]:

```
import statsmodels.api as sm
log_reg = sm.Logit(Y, X).fit()
print(log_reg.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.501195
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                   POST   No. Observations:                  240
Model:                          Logit   Df Residuals:                      226
Method:                           MLE   Df Model:                           13
Date:                Fri, 06 Nov 2020   Pseudo R-squ.:                  0.2126
Time:                        17:41:51   Log-Likelihood:                -120.29
converged:                       True   LL-Null:                       -152.76
Covariance Type:            nonrobust   LLR p-value:                 6.723e-09
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.3429      0.974      1.378      0.168      -0.567       3.252
x2             3.2892      2.137      1.539      0.124      -0.900       7.478
x3            -7.7348      5.924     -1.306      0.192     -19.346       3.877
x4             0.0444      0.823      0.054      0.957      -1.568       1.657
x5            -2.9628      2.257     -1.313      0.189      -7.386       1.461
x6            -0.0494      0.209     -0.237      0.813      -0.459       0.360
x7            -0.3901      0.230     -1.694      0.090      -0.842       0.061
x8            -0.3932      0.764     -0.515      0.607      -1.890       1.104
x9            -0.2961      0.239     -1.241      0.215      -0.764       0.172
x10            6.4508      4.922      1.311      0.190      -3.195      16.097
x11           -5.2292      4.448     -1.176      0.240     -13.947       3.489
x12          -17.1446     10.577     -1.621      0.105     -37.875       3.586
x13           17.7719     12.821      1.386      0.166      -7.356      42.900
x14            4.6839      8.959      0.523      0.601     -12.875      22.243
==============================================================================
```

Judging from the p-value, I decided to use only those p-value under 0.18, which is PA, R, CS, SLG, OPS. A surprising discovery here is that HR(p=0.957) is wildly non-siginificantly correlated with postseason birth, which is a bit the contrary of what teams pursue recently.

In [14]:

```
X2 = df_com[['PA', 'R', 'CS', 'SLG', 'OPS']]
Y2 = df_com['POST']
X2 = StandardScaler().fit_transform(X2)

model2 = LogisticRegression(multi_class='multinomial')
model2.fit(X2, Y2)
```

Out[14]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='multinomial', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [15]:

```
DF_2020_2 = DF_2020[['PA', 'R', 'CS', 'SLG', 'OPS']]
DF_2020_2 = StandardScaler().fit_transform(DF_2020_2)
predictions_2020_proba2 = model2.predict_proba(DF_2020_2)
predictions_2020_pred2 = model2.predict(DF_2020_2)
data_result2 = {'Team': df_2020['Tm'],
        'Probability': predictions_2020_proba2[:,1]*100,
            'Prediction': predictions_2020_pred2}
```

```
prediction_table2 = pd.DataFrame(data_result2)
print(prediction_table2)
```

```
   Team  Probability  Prediction
0   ARI    30.528836           0
1   ATL    34.945587           0
2   BAL    50.611437           1
3   BOS    24.343055           0
4   CHC    79.979493           1
5   CHW    39.400260           0
6   CIN    22.450180           0
7   CLE     6.953635           0
8   COL    81.358858           1
9   DET    26.703734           0
10  HOU    67.514209           1
11  KCR     7.929352           0
12  LAA    51.155164           1
13  LAD    56.421238           1
14  MIA    31.429079           0
15  MIL     9.461973           0
16  MIN    38.962639           0
17  NYM    71.531353           1
18  NYY    63.583383           1
19  OAK    46.468942           0
20  PHI    64.933070           1
21  PIT     3.358363           0
22  SDP    22.172423           0
23  SEA    14.899682           0
24  SFG    18.638615           0
25  STL     1.257867           0
26  TBR    71.328421           1
27  TEX     5.664947           0
28  TOR     3.660027           0
29  WSN     2.925560           0
```

The list of teams were the same but there were slight differences for the probability value.

Conclusion

The result was quite satisfying given the list of teams consumes most of the powerhouse of MLB, but we still have to keep in mind that this research only considered the batting part of stats and the stats on 8/13. In my last prediction model it only considered 'PA', 'R', 'CS', 'SLG', 'OPS', which may give us a look at what matters most in teams probability of getting into postseason.

Prediction using classifiers other than Logistic Regression can be found here: https://github.com/Allen-Ho-0302/2020PostseasonPrediction-DeepLearning_XGBoost_ClassificationTree_LogisticRegression.