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' 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

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  \
```

```
      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]
```

Step 2: Train the Models

In [3]:

```python
from sklearn.preprocessing import StandardScaler

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

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

In [4]:

```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),
```

```
    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 [5]:

```
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):

    # 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:
```

```
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 = 0.007 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)
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 = 0.001 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.
```

```
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 Random Forest = 0.027 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 = 0.018 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 = 0.008 min.
```

	models	F1 Score	Balanced Accuracy	ROC AUC	Neg Log Loss
0	Logistic Regression	0.705002	0.706250	0.832812	-0.535090
1	KNN	0.669356	0.696875	0.835547	-0.574364
2	SVM	0.639647	0.631250	0.741797	-0.611696
3	Decision Tree	0.714471	0.687500	0.687500	-9.642188
4	Random Forest	0.728670	0.712500	0.807813	-0.559156
5	Gradient Boosting	0.709187	0.690625	0.789062	-0.628873
6	XGBClassifier	0.733194	0.721875	0.774219	-0.638473

In [6]:

```
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 = 0.001 min.
Time to cross-validate SVM = 0.001 min.
Time to cross-validate Decision Tree = 0.001 min.
Time to cross-validate Random Forest = 0.027 min.
Time to cross-validate Gradient Boosting = 0.017 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	-0.548995
3	Decision Tree	0.684726	0.650000	0.650000	-10.649576
4	Random Forest	0.708330	0.671875	0.805859	-0.535242
5	Gradient Boosting	0.727597	0.696875	0.792969	-0.623737
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 [7]:

```
model = LogisticRegression(multi_class='multinomial', max_iter=10000)
model.fit(X, Y)
```

Out[7]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='multinomial', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [8]:

```
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())
```

```
         PA162         R162         H162        HR162       RBI162       SB162  \
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

       CS162        BB162        SO162     BA  ...           PA            R  \
0  25.578947  477.473684  1219.263158  0.245  ...  6096.315789  750.315789
1  24.300000  526.500000  1644.300000  0.244  ...  5977.800000  842.400000
2  45.000000  513.000000  1305.000000  0.269  ...  6111.000000  864.000000
3  25.578947  426.315789  1492.105263  0.254  ...  6096.315789  724.736842
4  10.125000  658.125000  1620.000000  0.244  ...  6176.250000  840.375000

             H           HR          RBI           SB           CS           BB  \
0  1347.157895  127.894737  707.684211  34.105263  25.578947  477.473684
1  1312.200000  226.800000  826.200000  64.800000  24.300000  526.500000
2  1467.000000  243.000000  846.000000  63.000000  45.000000  513.000000
3  1415.368421  196.105263  682.105263  34.105263  25.578947  426.315789
4  1296.000000  232.875000  789.750000  50.625000  10.125000  658.125000

            SO           TB
0  1219.263158  2097.473684
1  1644.300000  2349.000000
2  1305.000000  2583.000000
3  1492.105263  2353.263158
4  1620.000000  2288.250000

[5 rows x 24 columns]
```

In [9]:

```
### Calculate predictions: predictions
predictions_2020 = model.predict(DF_2020)
```

```
predictions_2020 = [round(value) for value in predictions_2020]

print(predictions_2020)
```

```
[1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0]
```

Result

When trying to see which teams' offensive stats on 08/13/2020(normalized to 162 games) are worthy of getting into postseason on a traditional 10-team postseason format, it shows ARI, CIN, CLE, HOU, LAA, MIA, MIL, OAK, PIT, SEA, TEX.

Conclusion

Though the list of teams might not be exactly the powerhouse of MLB on 8/13, but we have to keep in mind that this research only considered the offensive part of baseball.