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 model

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  53.0  444.0  1354.0
```

```
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: Perform k-fold cross-validation on Logistic Regression Model

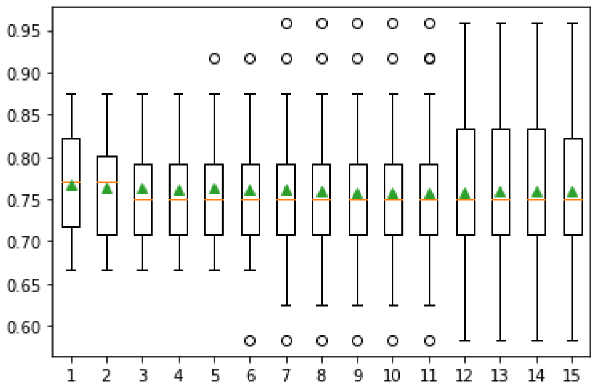
The expectation of repeated k-fold cross-validation is that the repeated mean would be a more reliable estimate of model performance than the result of a single k-fold cross-validation procedure. This may mean less statistical noise. One way this could be measured is by comparing the distributions of mean performance scores under differing numbers of repeats.

In [3]:

```python
# compare the number of repeats for repeated k-fold cross-validation
from scipy.stats import sem
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot


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

# evaluate a model with a given number of repeats
def evaluate_model(X, Y, repeats):
 # prepare the cross-validation procedure
 cv = RepeatedKFold(n_splits=10, n_repeats=repeats, random_state=1)
 # create model
 model = LogisticRegression()
 # evaluate model
 scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=-1)
 return scores

# configurations to test
repeats = range(1,16)
results = list()
for r in repeats:
 # evaluate using a given number of repeats
 scores = evaluate_model(X, Y, r)
 # summarize
 print('>%d mean=%.4f se=%.3f' % (r, mean(scores), sem(scores)))
 # store
 results.append(scores)
# plot the results
pyplot.boxplot(results, labels=[str(r) for r in repeats], showmeans=True)
pyplot.show()
```

```
>1 mean=0.7667 se=0.023
>2 mean=0.7625 se=0.015
>3 mean=0.7639 se=0.012
>4 mean=0.7615 se=0.010
>5 mean=0.7633 se=0.009
>6 mean=0.7618 se=0.008
```

```
>7 mean=0.7613 se=0.008
>8 mean=0.7589 se=0.008
>9 mean=0.7583 se=0.007
>10 mean=0.7579 se=0.007
>11 mean=0.7583 se=0.007
>12 mean=0.7583 se=0.007
>13 mean=0.7593 se=0.006
>14 mean=0.7598 se=0.006
>15 mean=0.7589 se=0.006
```



Ideally, we would like to select a number of repeats that shows both minimization of the standard error and stabilizing of the mean estimated performance compared to other numbers of repeats. 10 repeats seems like a good choice here.

In [4]:

```
# prepare the cross-validation procedure
cv = RepeatedKFold(n_splits=10, n_repeats=10, random_state=1)
# create model
model = LogisticRegression()
# evaluate model
scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.758 (0.073)
```

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 [5]:

```
import pandas as pd
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 [6]:

```
### Calculate predictions: predictions
model.fit(X, Y)

predictions_2020 = model.predict(DF_2020)

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

print(predictions_2020)
```

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

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Result

The model we built has a roughly 75.8% accuracy on training data set(data from 2011-2019). When trying to see which teams' stats on 08/13/2020(normalized to 162 games) are worthy of getting into postseason on a traditional 10-team postseason format, it shows BAL, CHC, COL, HOU, LAA, LAD, MIA, MIN, NYM, NYY, OAK, PHI, TBR.

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 offense part of baseball. And it's definitely good to see teams like CHC, HOU, LAD, NYY, OAK, TBR making the list.