Hall Of Fame Baseball

By: Brian McCabe

Purpose

Identifying Hall Of Fame Baseball Players

Batting, Fielding, Pitching, All-star and Awards Statistics

Comparing Binary Classification Algorithms

K-Nearest Neighbors

Logistic Regression

SVM

Decision Trees

Random Forest

Naïve Bayes

XGBoost

Significance

What makes a Hall Of Fame Baseball Player?

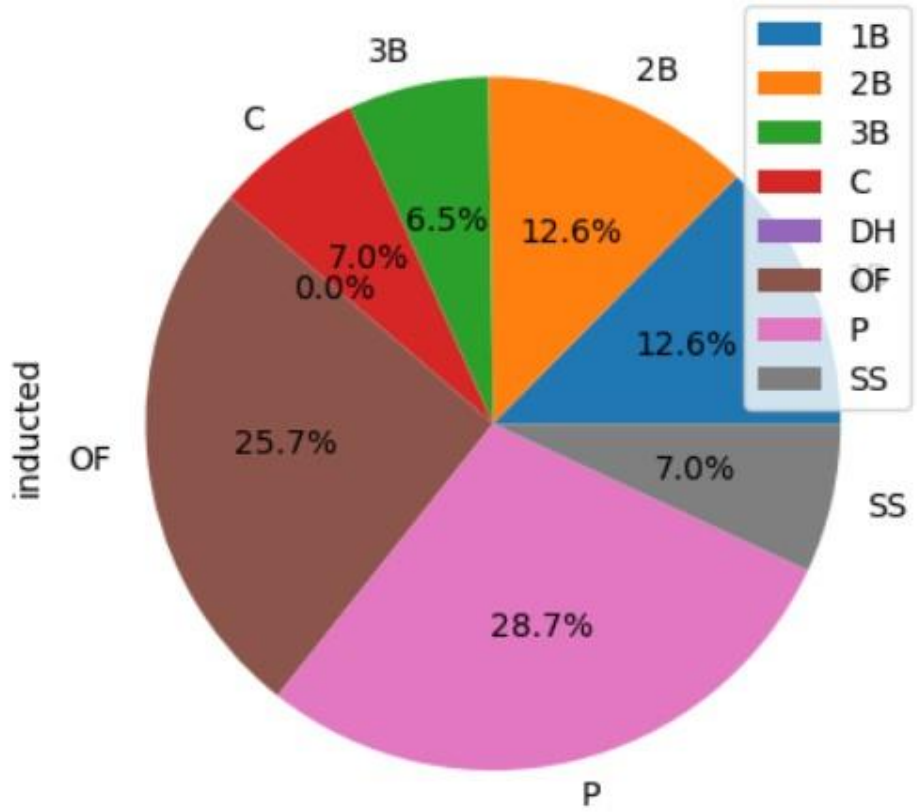
How can this be used?

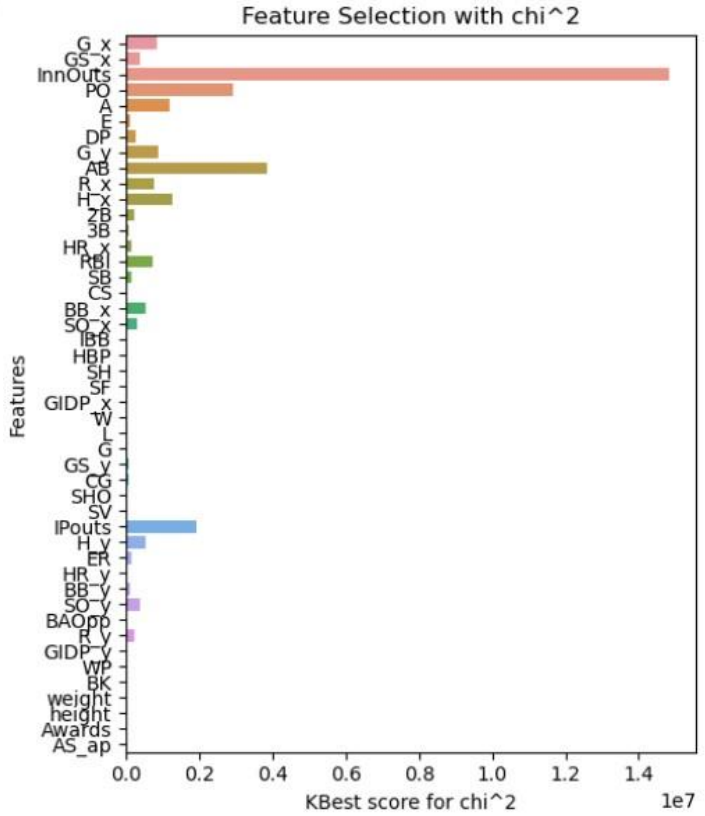
Who would benefit from this analysis?

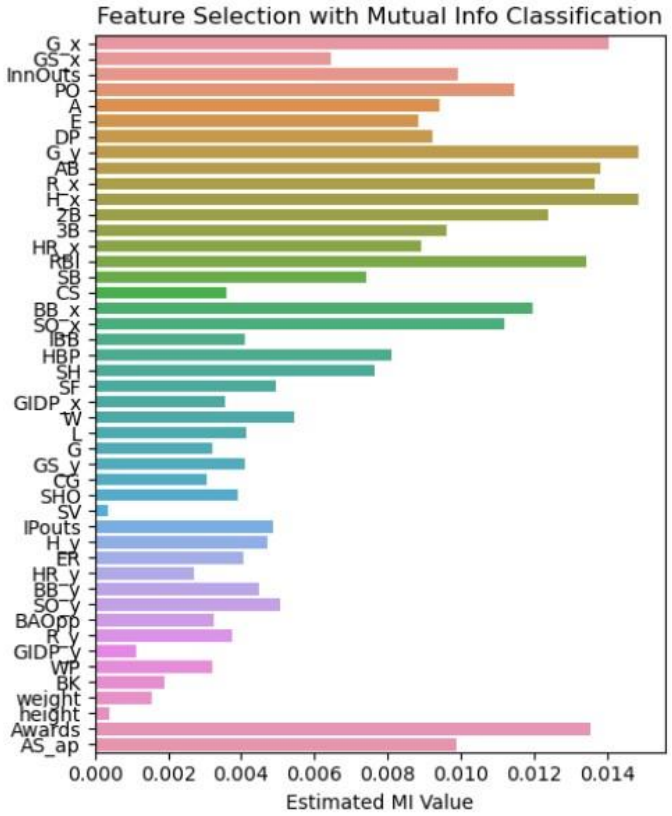
Data Description

- 46 input variables
- Batting
  - At Bats
  - Runs
  - Hits
  - Home Runs
  - Walks
  - Runs Batted In
  - Walks
  - Strike Outs
- Pitching
  - Wins
  - Loses
  - Shut Outs
  - Strike Outs
  - Runs Allowed
- Fielding
  - Games Started
  - Put Outs
  - Assist
  - Errors
  - Inn Outs
  - Double Plays
- Awards
  - All Star Appearances
  - Awards Won
- 1 Out-Put Variable: HOF Status
  - 1 if player is in HOF
  - 0 if player is not in HOF

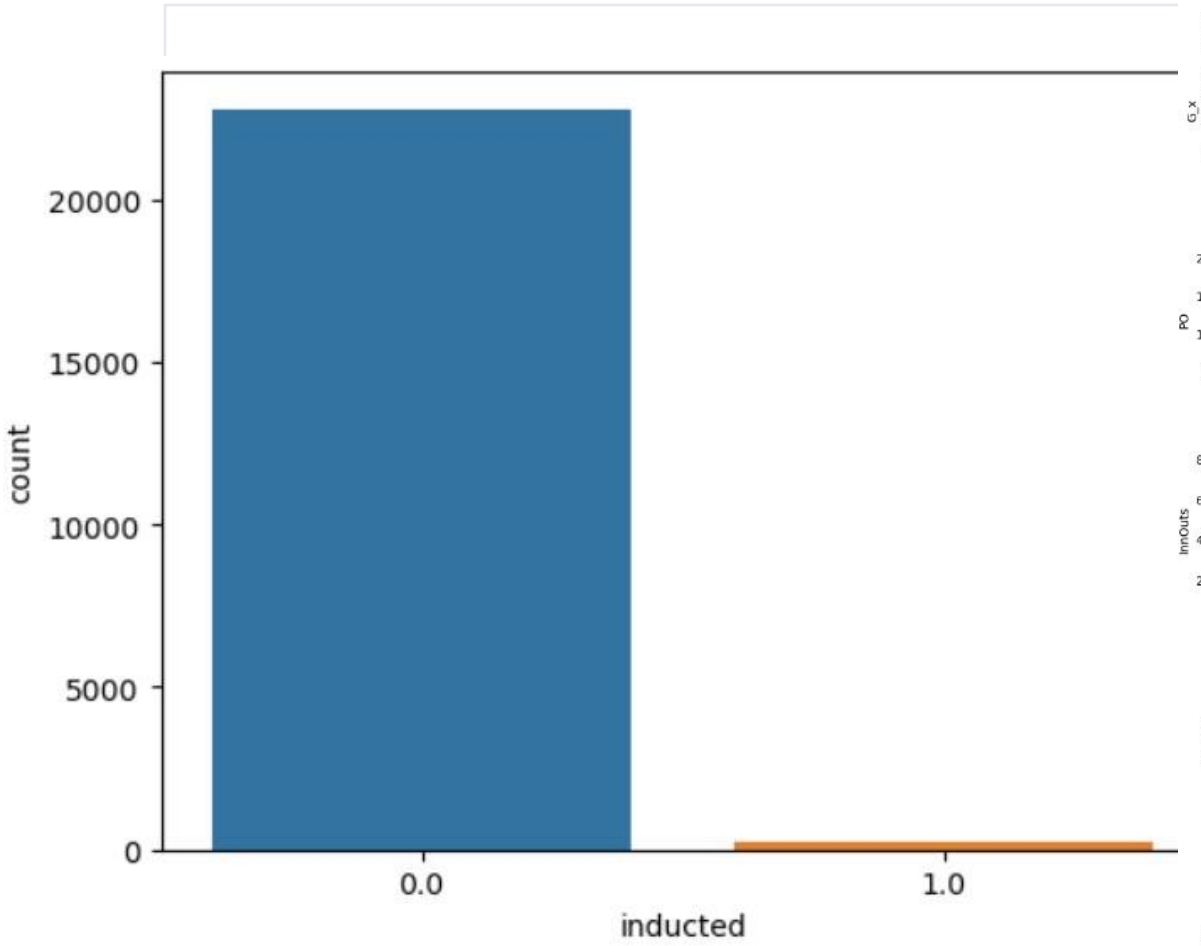
Exploratory

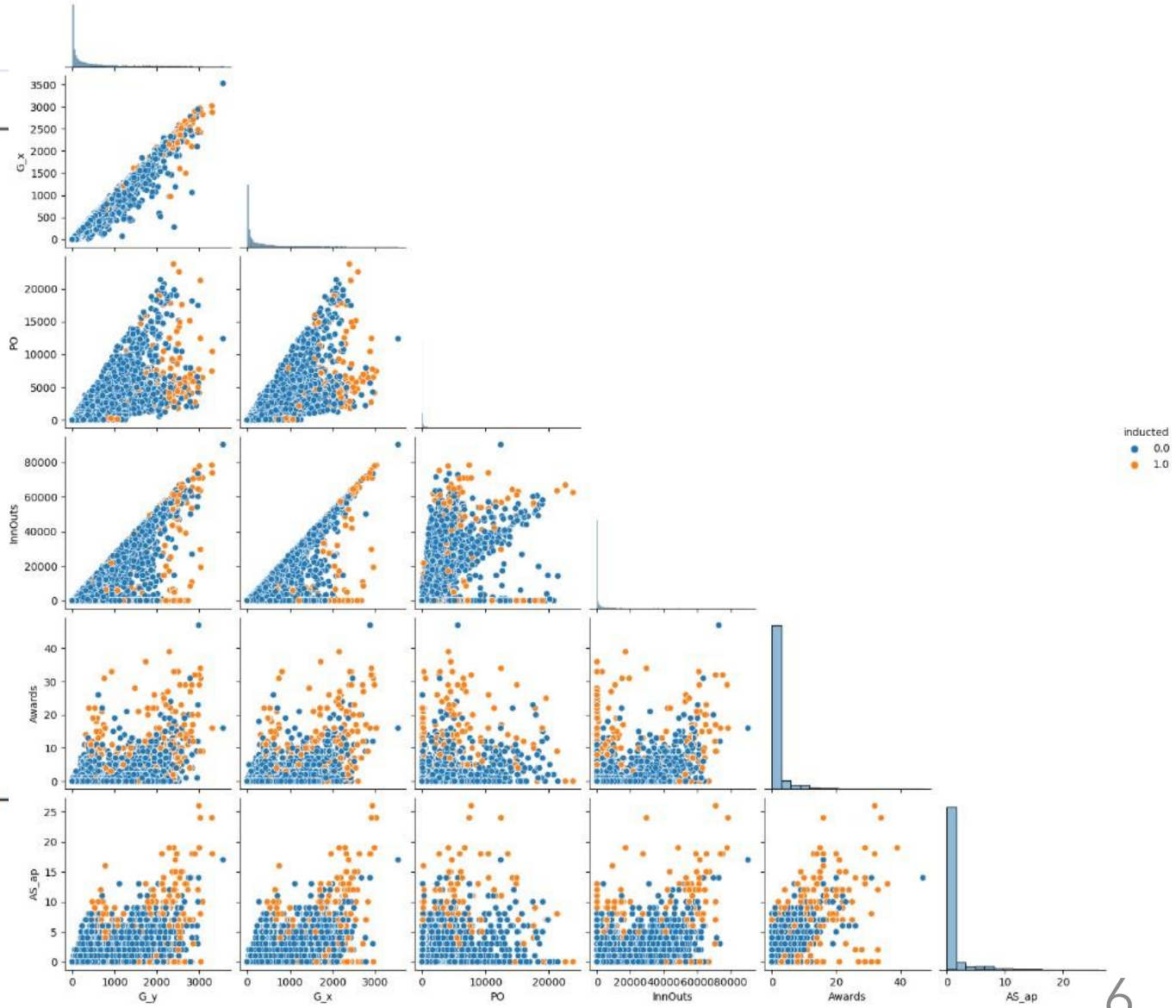




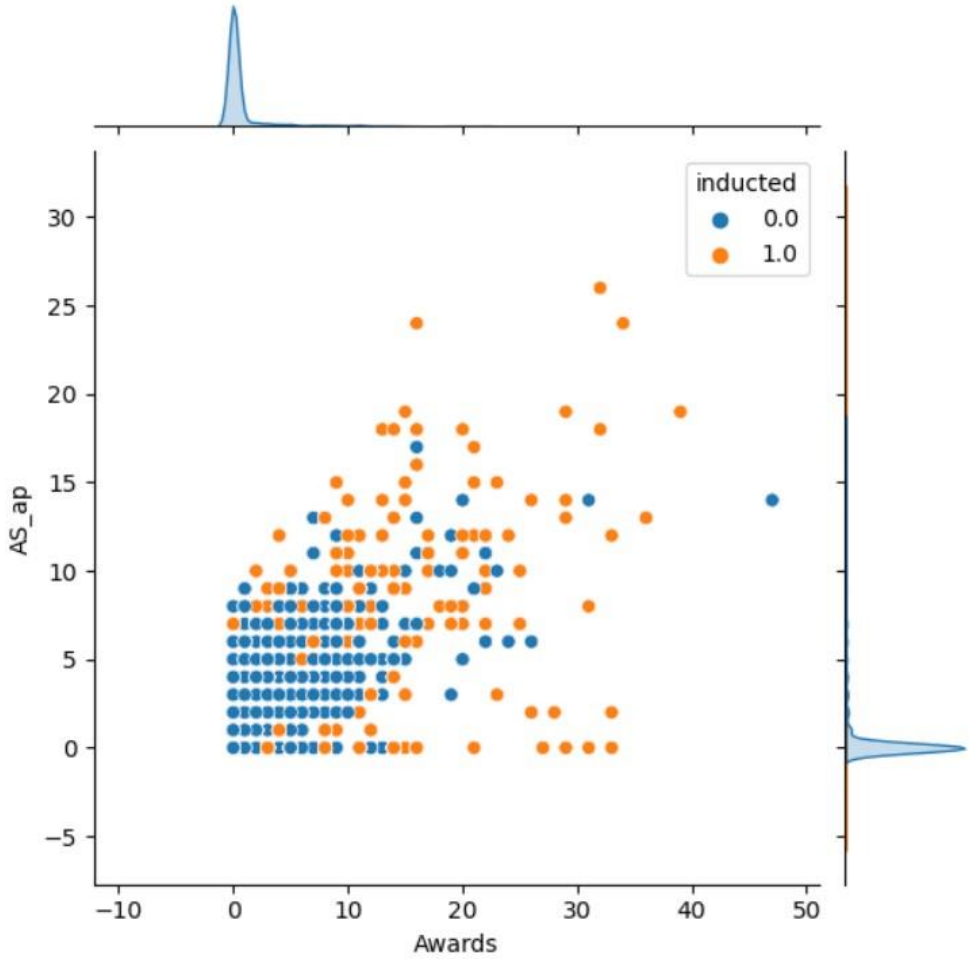


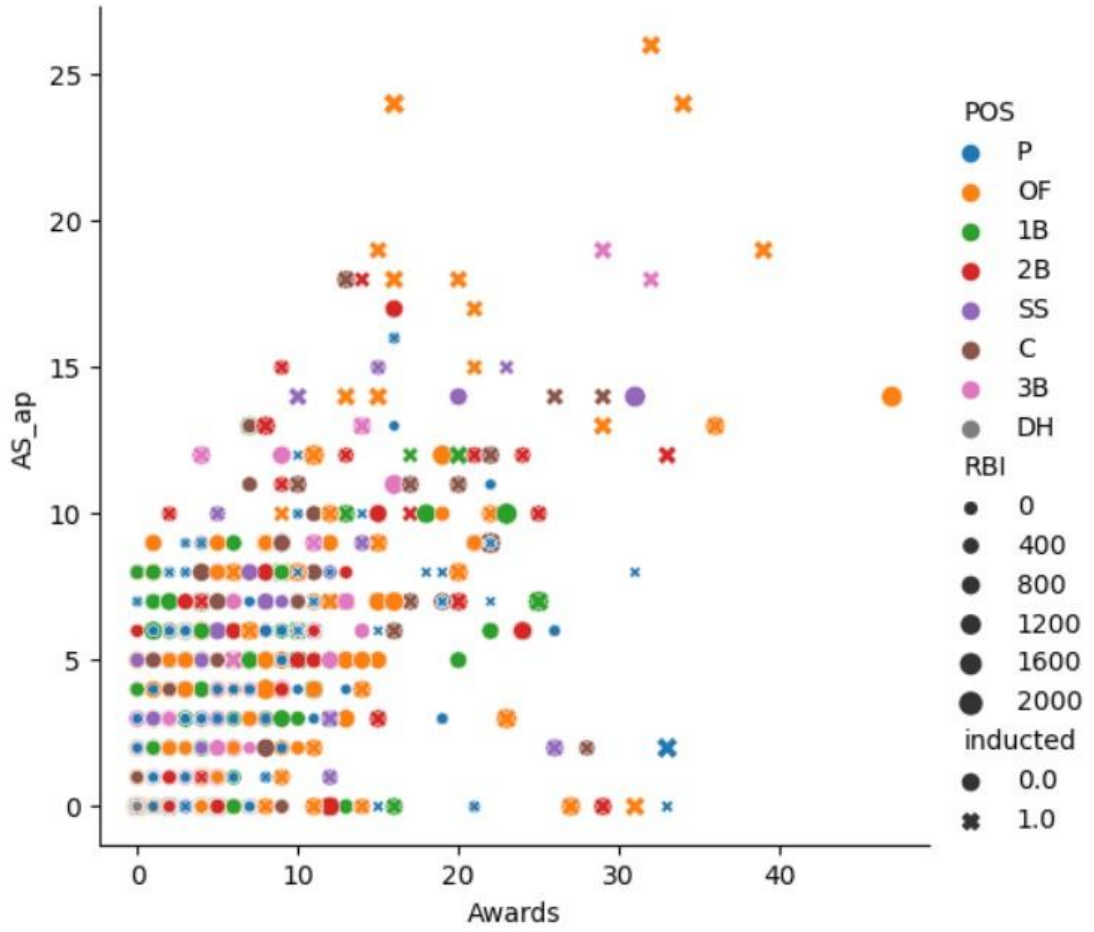
Exploratory





Exploratory





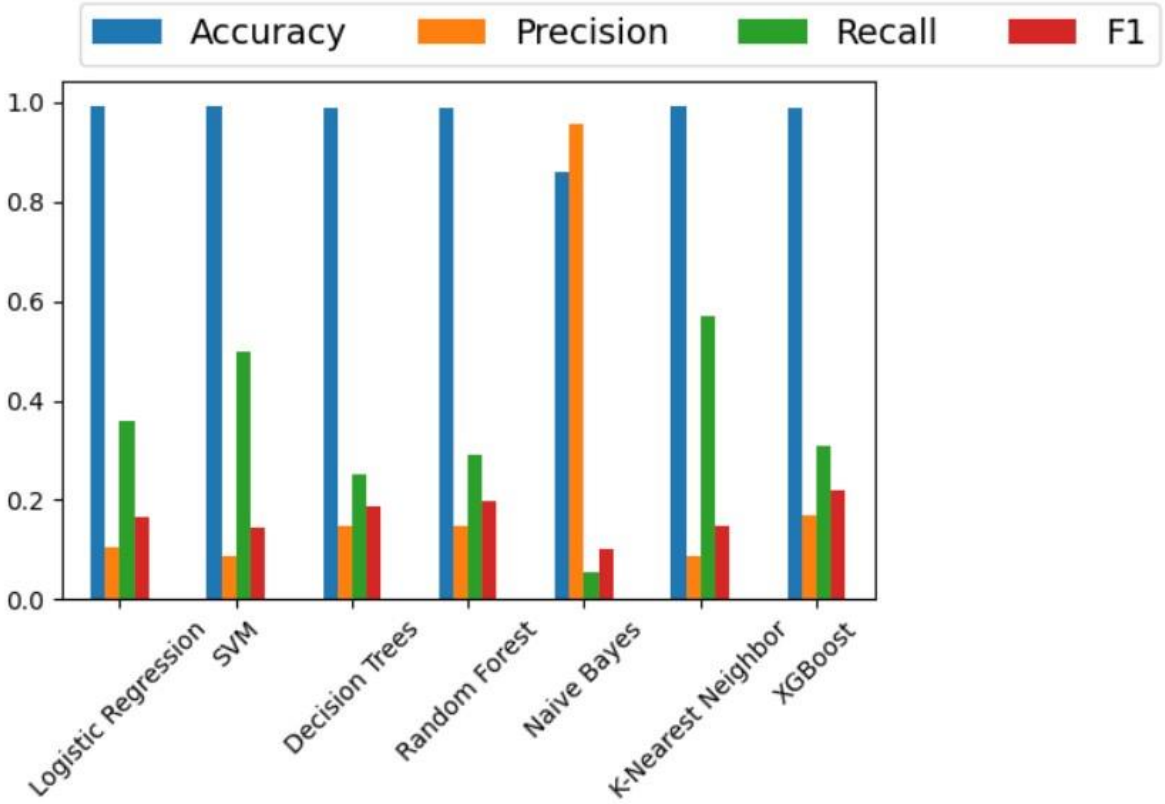
Data Preparation

- Combined records on player ID
- Used Aggregate Functions
- Merged all 7 Data sets into one 23005 x 50
- Changed out-put variable to binary
- Filled NANs with Zeros
- 75/25 Train Test Split
- Standard Scaler

```
aggregation_functions_f = {'POS': 'first', 'G': 'sum','GS': 'sum', 'InnOuts':'sum',
                           'PO':'sum','A':'sum','E':'sum','DP':'sum'}
aggregation_functions_b = {'G': 'sum','AB': 'sum', 'R':'sum','H':'sum',
                           '2B':'sum','3B':'sum','HR':'sum','RBI':'sum','SB':'sum',
                           'CS':'sum','BB':'sum','SO':'sum','IBB':'sum','HBP':'sum',
                           'SH':'sum','SF':'sum','GIDP':'sum'}
aggregation_functions_p = {'W': 'sum','L': 'sum', 'G':'sum','GS':'sum',
                           'CG':'sum','SHO':'sum','SV':'sum','IPouts':'sum','H':'sum',
                           'ER':'sum','HR':'sum','BB':'sum','SO':'sum','BAOpp':'sum',
                           'R':'sum','GIDP':'sum','WP':'sum','BK':'sum'}
combined_f = fielding.groupby(fielding['playerID']).aggregate(aggregation_functions_f)
combined_b = batting.groupby(batting['playerID']).aggregate(aggregation_functions_b)
combined_p = pitching.groupby(pitching['playerID']).aggregate(aggregation_functions_p)
merged_df = pd.merge(combined_f,combined_b, on='playerID',how = 'outer')
merged_df = pd.merge(merged_df,combined_p, on='playerID',how='outer')
merged_df['POS'].fillna("DH",inplace=True)
info = info[['playerID','nameFirst','nameLast','weight','height']]
hof['inducted'] = hof['inducted'].map({'Y': 1, 'N': 0})
hof = hof[hof['category']=='Player']
hof = hof[['playerID','inducted']]
awards_clean = pd.DataFrame(awards.groupby(['playerID'])['playerID'].count())
awards_clean.rename(columns={"playerID": "Awards"},inplace=True)
as_clean = pd.DataFrame(allstar.groupby(['playerID'])['playerID'].count())
as_clean.rename(columns={"playerID": "AS_ap"},inplace=True)
merged_df = pd.merge(merged_df,info, on='playerID',how = 'outer')
merged_df = pd.merge(merged_df,hof, on='playerID',how='outer')
merged_df = pd.merge(merged_df,awards_clean, on='playerID',how='outer')
merged_df = pd.merge(merged_df,as_clean, on='playerID',how='outer')
merged_df = merged_df.fillna(0)
merged_df = merged_df[merged_df['POS']!=0]
merged_df = merged_df.set_index('playerID')
merged_df
```

```
y = merged_df['inducted']
X = merged_df.drop(['POS','nameLast','nameFirst','inducted'], axis=1)
standardizer = StandardScaler()
X = standardizer.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size=0.25, random_state=0)
```

Initial analysis



	Accuracy	Precision	Recall	F1
Logistic Regression	0.991134	0.106383	0.357143	0.163934
SVM	0.991829	0.085106	0.500000	0.145455
Decision Trees	0.989395	0.148936	0.250000	0.186667
Random Forest	0.990090	0.148936	0.291667	0.197183
Naive Bayes	0.859875	0.957447	0.053004	0.100446
K-Nearest Neighbor	0.992003	0.085106	0.571429	0.148148
XGBoost	0.990090	0.170213	0.307692	0.219178

Tuning

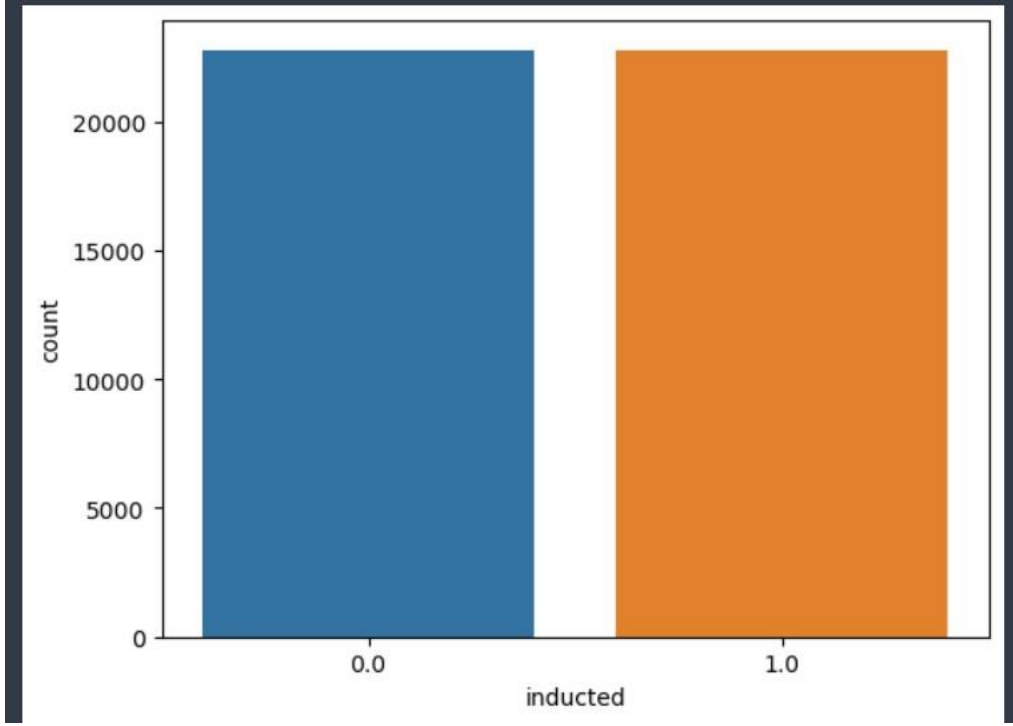
- Using Up Sampling to tune models
- Used Threshold Technique to tune best model

```
df_majority = merged_df[(merged_df['inducted']==0)]
df_minority = merged_df[(merged_df['inducted']==1)]
df_minority_upsampled = resample(df_minority, replace=True,n_samples= 22775,random_state=42)
df_upsampled = pd.concat([df_minority_upsampled, df_majority])
```

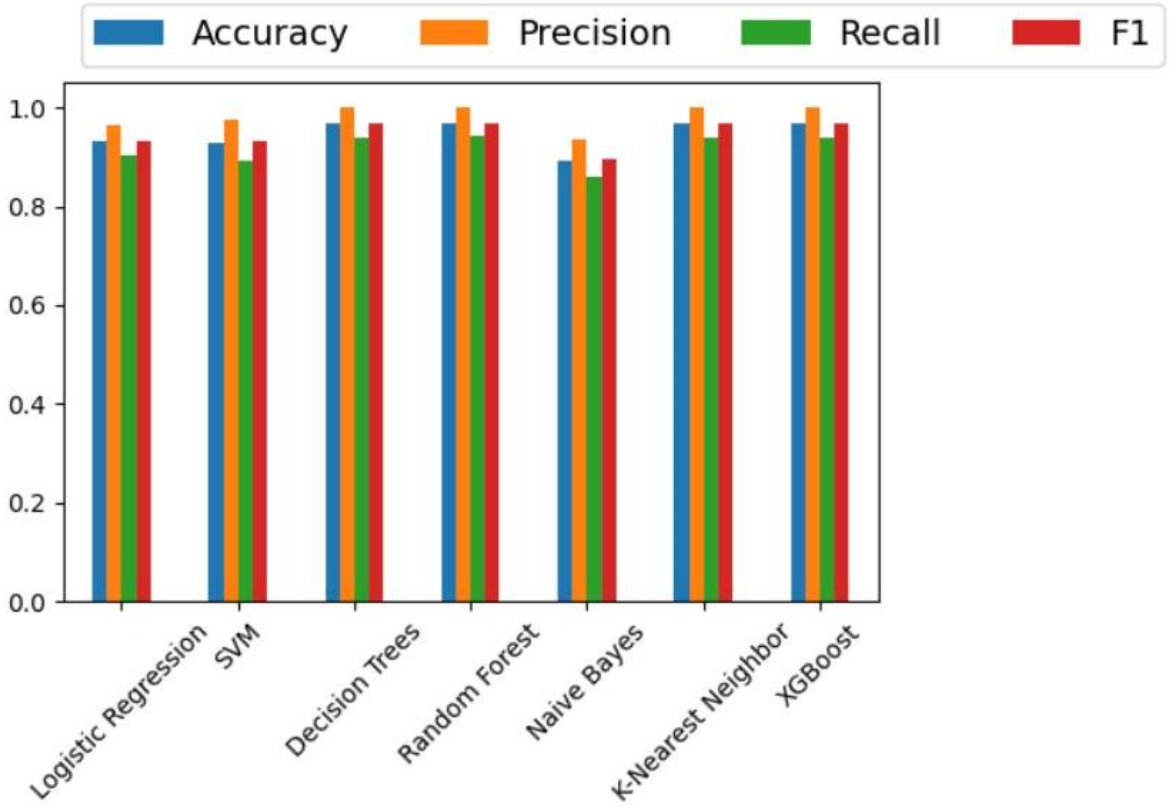
```
#Data is now evenly distributed
print(df_upsampled['inducted'].value_counts())
sns.countplot(df_upsampled['inducted'])
```

```
1.0    22775
0.0    22775
Name: inducted, dtype: int64


<AxesSubplot:xlabel='inducted', ylabel='count'>
```

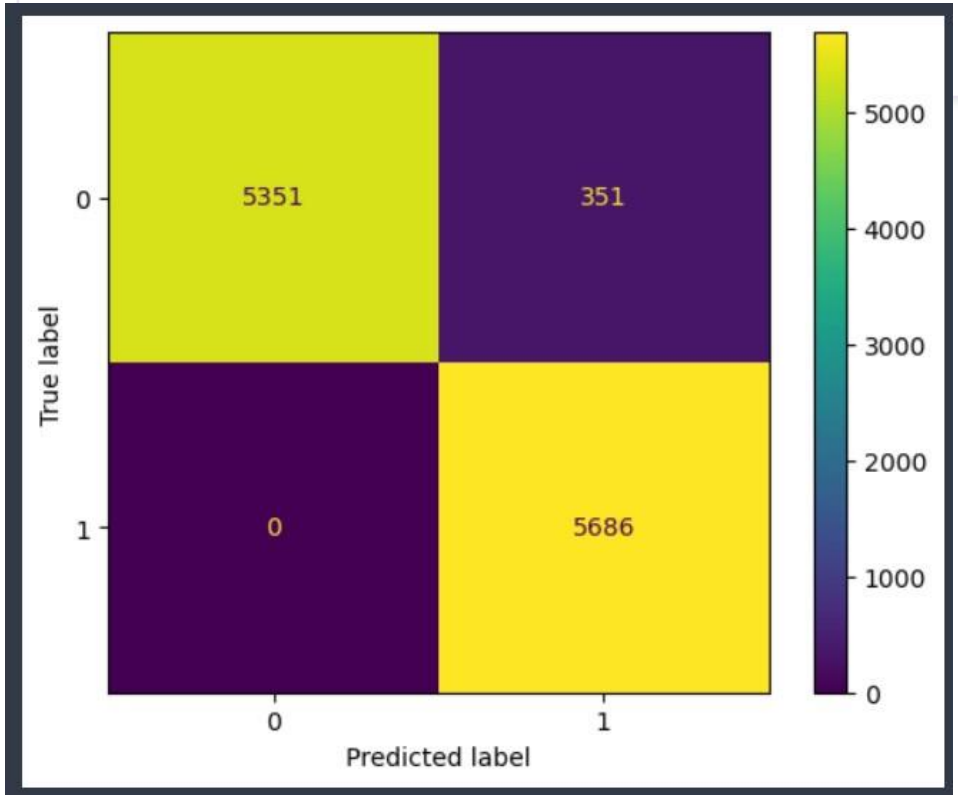


Model Analysis

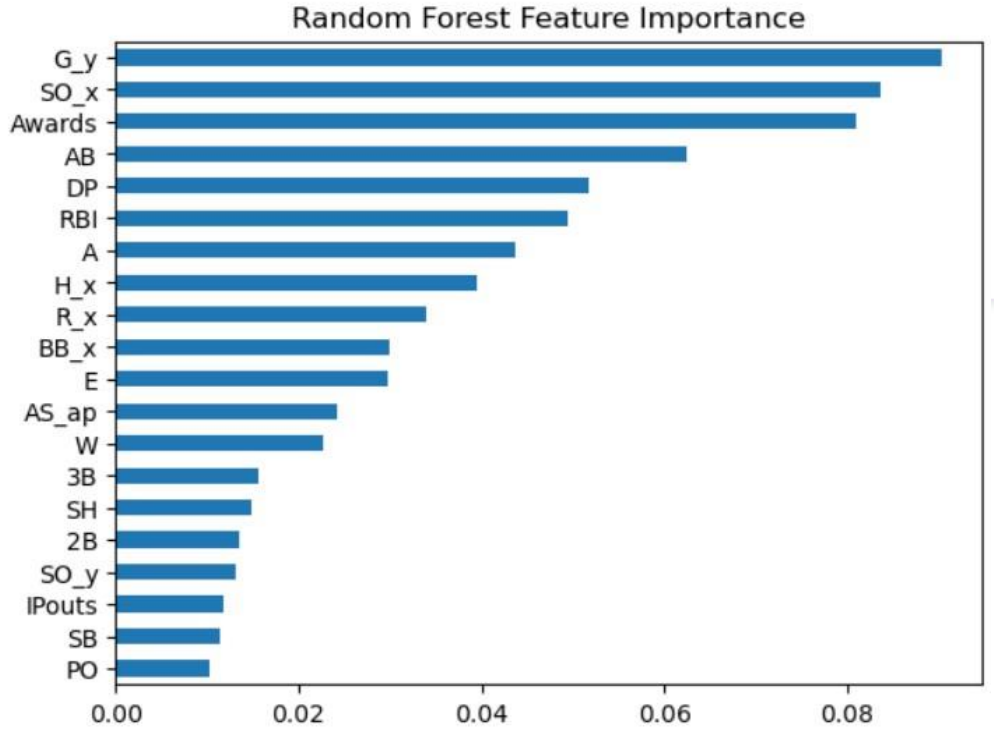


	Accuracy	Precision	Recall	F1
Logistic Regression	0.930453	0.965529	0.902070	0.932722
SVM	0.928082	0.973444	0.892310	0.931113
Decision Trees	0.967597	1.000000	0.939059	0.968572
Random Forest	0.968739	1.000000	0.941079	0.969645
Naive Bayes	0.891113	0.936159	0.858548	0.895676
K-Nearest Neighbor	0.967422	1.000000	0.938749	0.968407
XGBoost	0.968476	1.000000	0.940612	0.969397

Conclusion



	precision	recall	f1-score	support
0.0	0.94	1.00	0.97	5343
1.0	1.00	0.94	0.97	6045
accuracy			0.97	11388
macro avg	0.97	0.97	0.97	11388
weighted avg	0.97	0.97	0.97	11388



	Accuracy	Precision	Recall	F1
Logistic Regression	0.930453	0.965529	0.902070	0.932722
SVM	0.928082	0.973444	0.892310	0.931113
Decision Trees	0.967597	1.000000	0.939059	0.968572
Random Forest	0.968739	1.000000	0.941079	0.969645
Naive Bayes	0.891113	0.936159	0.858548	0.895676
K-Nearest Neighbor	0.967422	1.000000	0.938749	0.968407
XGBoost	0.968476	1.000000	0.940612	0.969397
RF.7	0.969178	1.000000	0.941859	0.970059

Testing Model

```
#Testing model on some well know players and up and coming players
dj = merged_df[ (merged_df['nameFirst'] == 'Derek') & (merged_df['nameLast'] == 'Jeter') ].drop(['POS','nam
do = merged_df[ (merged_df['nameFirst'] == 'David') & (merged_df['nameLast'] == 'Ortiz') ].drop(['POS','nam
dw = merged_df[ (merged_df['nameFirst'] == 'David') & (merged_df['nameLast'] == 'Wright') ].drop(['POS','na
ha = merged_df[ (merged_df['nameFirst'] == 'Hank') & (merged_df['nameLast'] == 'Aaron') ].drop(['POS','name
cy = merged_df[ (merged_df['nameFirst'] == 'Cy') & (merged_df['nameLast'] == 'Young') ].drop(['POS','nameLa
aj = merged_df[ (merged_df['nameFirst'] == 'Aaron') & (merged_df['nameLast'] == 'Judge') ].drop(['POS','nam
aj = merged_df[ (merged_df['nameFirst'] == 'Mike') & (merged_df['nameLast'] == 'Piazza') ].drop(['POS','nam

x = pd.concat([do,dj,dw,ha,cy,aj])
```

```
#Using my .7 threshold all 0's were predicted
predicted_proba_test = models['Random Forest'].predict_proba(x)
predicted_proba_test
predicted = (predicted_proba_test[:,1] >= .7).astype('int')
print(predicted)
```

```
[0 0 0 0 0 0 0 0 0 0]
```

```
#Had better results without the threshold
predictions = models[key].predict(x)
print(predictions)
```

```
[0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
```

Questions

Thank you