Prediction of NBA Draft

Jinming Chen

Professor Andrew Catlin

Will Data-Driven Decision Making Lead to Better Draft Choices?

Project Goals

- Which statistical features best predict whether an NCAA player will be drafted into the NBA?
- Which accuracy metric is most relevant?
- Which machine learning model has the best accuracy?

Interview with the NCAA Super Star: Jordan Armstrong

- What I learned

Models Evaluated

- Logistic Regression
- Support Vector Machine
- K-nearest neighbors algorithm
- Decision Tree
- Random Forest
- XG Boost Classifier

Datasets

- Game statistics and combine data for drafted and top undrafted players for three NBA seasons.

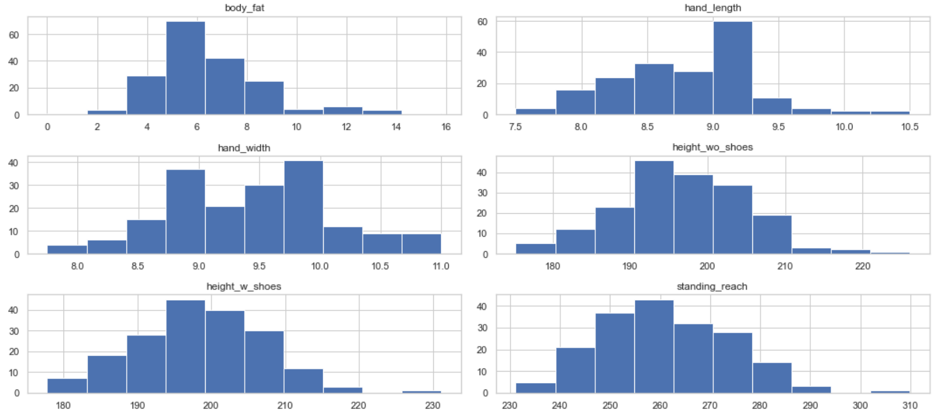
NBA Combine Data Source & Method

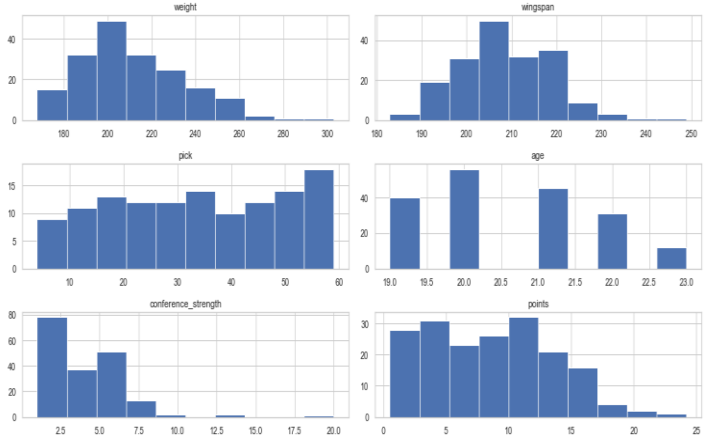
In [2]:
```
import requests
from bs4 import BeautifulSoup

with open("C:/Users/11946/NBA2018.html", "rb") as fp:
    soup = BeautifulSoup(fp, 'html.parser')
print(soup)
```

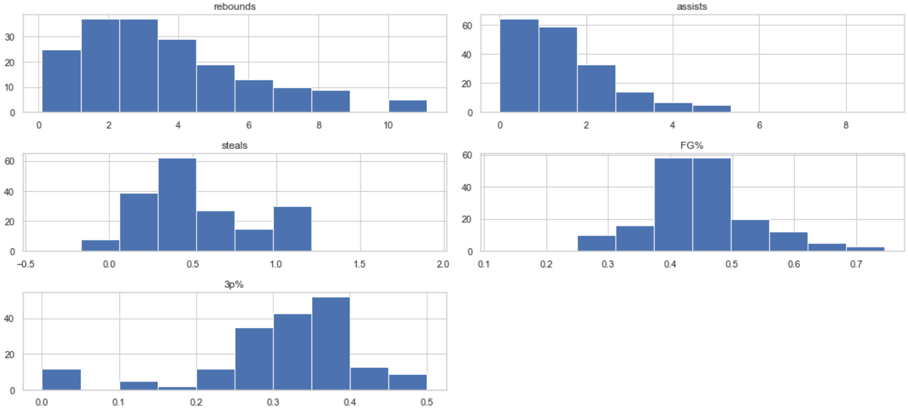
	Season	player	pos	body_fat	hand_length	hand_width	height_wo_shoes	height_w_shoes	standing_reach	weight	...	draft	pick	conference	conference_strength
0	2018-2019	Rawle Alkins	SG	8.90%	8.50	10.00	187.96	193.04	251.46	217.4	...	0	NaN	Pac-12	6.0
1	2018-2019	Grayson Allen	SG	5.55%	8.25	10.00	190.50	193.04	246.38	198.0	...	1	21.0	ACC	5.0
2	2018-2019	Kostas Antetokounmpo	PF	5.00%	9.25	9.50	205.74	208.28	279.40	194.8	...	1	20.0	ACC	5.0
3	2018-2019	Udoka Azubuike	C	7.95%	9.50	10.00	208.28	213.36	284.48	273.8	...	1	27.0	Big 12	3.0
4	2018-2019	Mohamed Bamba	C	6.20%	9.75	10.25	210.82	213.36	292.10	225.6	...	1	6.0	SEC	2.0

Exploratory Data Analysis

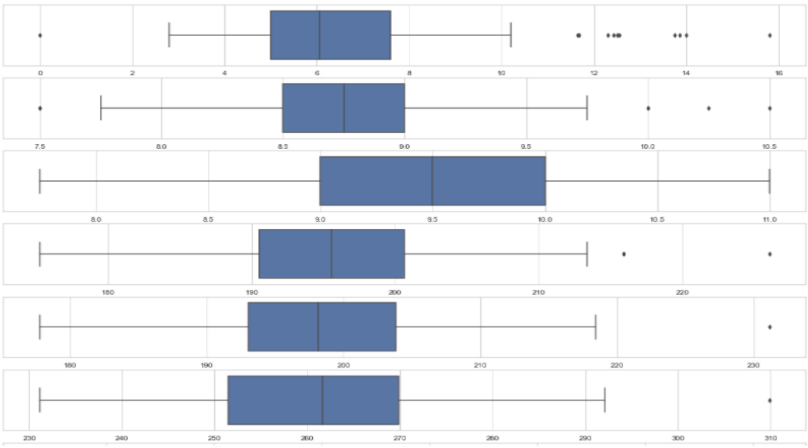
Histogram(numerical)

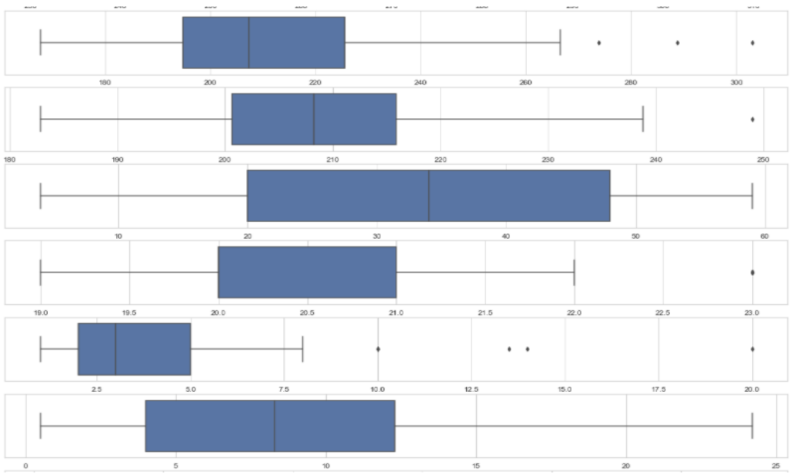


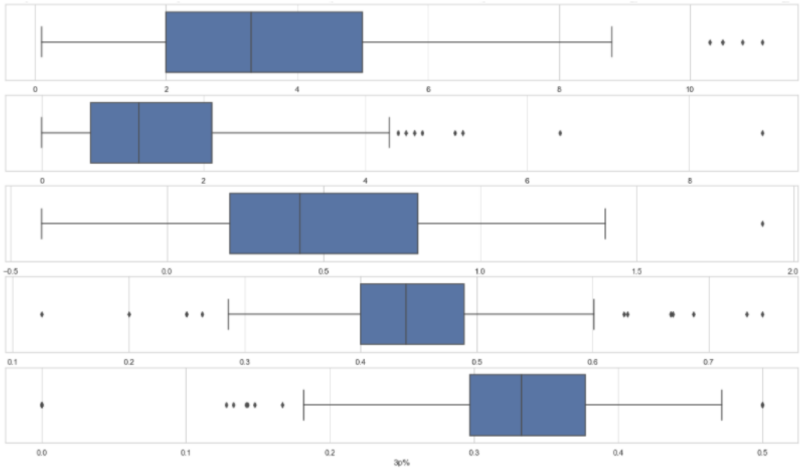




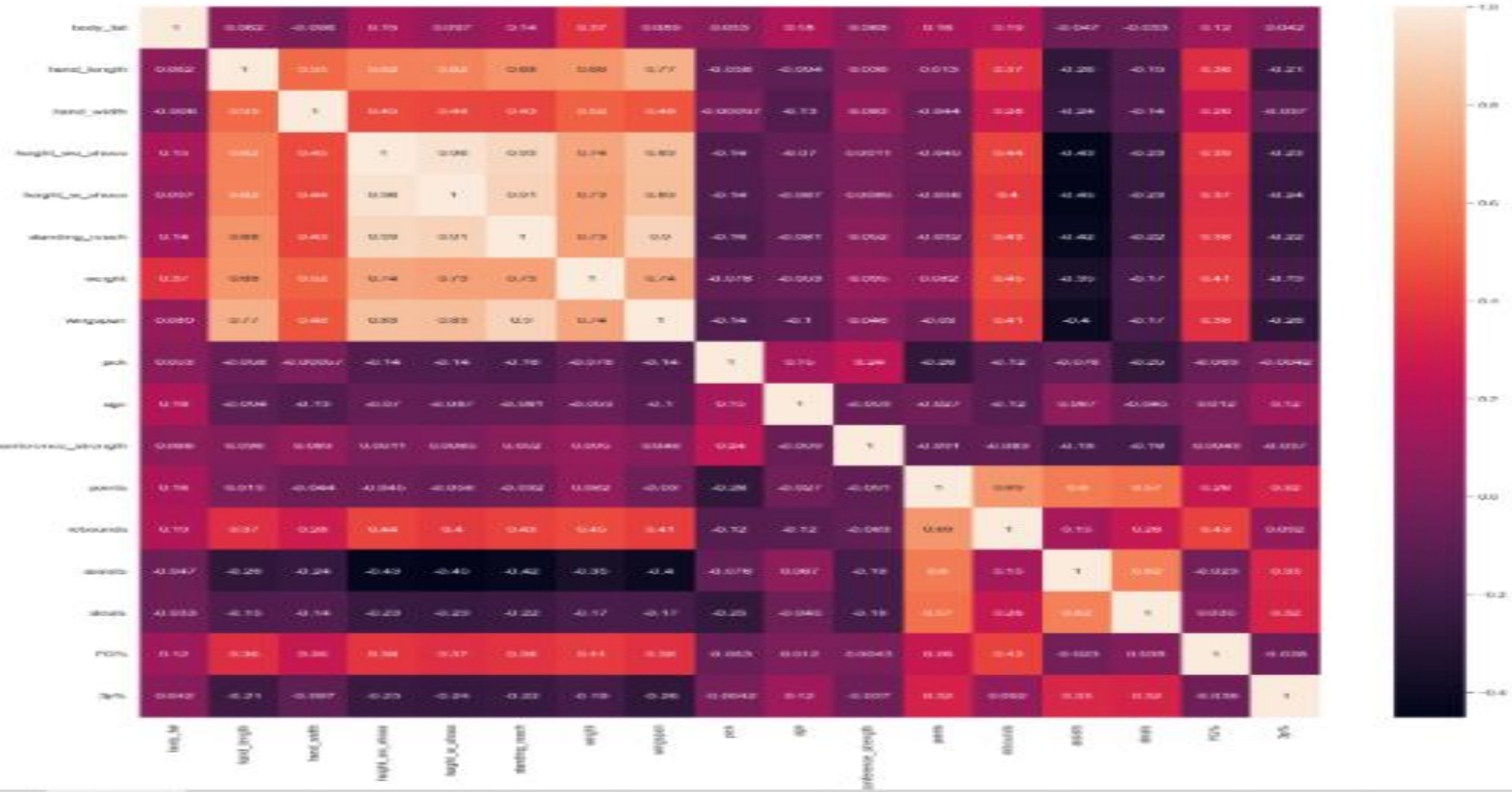
Box Plot



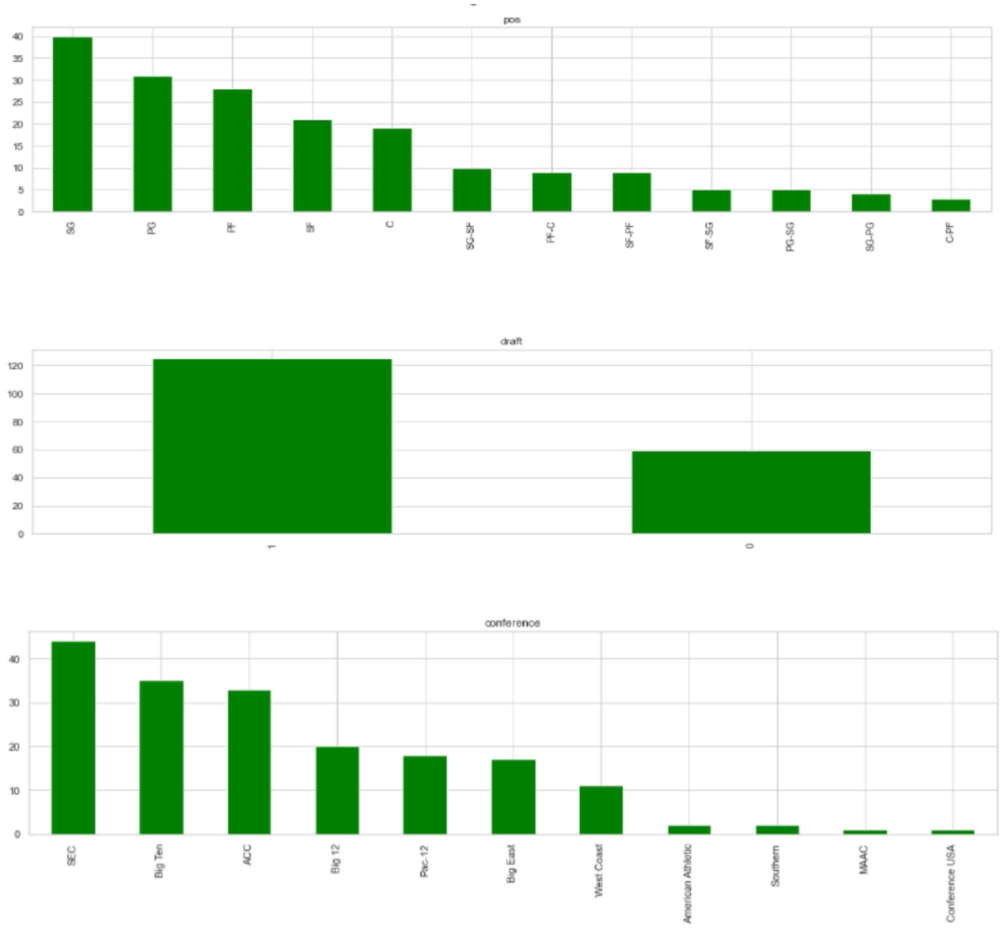




Correlation heatmap



Histogram (categorical)



Feature Selection

- Chi Square

```
#
col_cat3=['Season_2018-2019',
 'Season_2019-2020',
 'Season_2020-2021',
 'pos_C',
 'pos_C-PF',
 'pos_PF',
 'pos_PF-C',
 'pos_PG',
 'pos_PG-SG',
 'pos_SF',
 'pos_SF-PF',
 'pos_SF-SG',
 'pos_SG',
 'pos_SG-PG',
 'pos_SG-SF',
 'conference_ACC',
 'conference_American Athletic',
 'conference_Big 12',
 'conference_Big East',
 'conference_Big Ten',
 'conference_Conference USA',
 'conference_MAAC',
 'conference_Pac-12',
 'conference_SEC',
 'conference_Southern',
 'conference_West Coast']

#Import stats module to perform chi-square
from scipy import stats

#Perform chi-square test
chi_sq=[]
for i in range(0,26):
    chi_sq.append([stats.chi2_contingency(pd.crosstab(df2['draft'], df2[col_cat3].iloc[:,i]))[0:2],i])

#Chi_sq
chi_sq.sort(reverse=True)
chi_sq
```

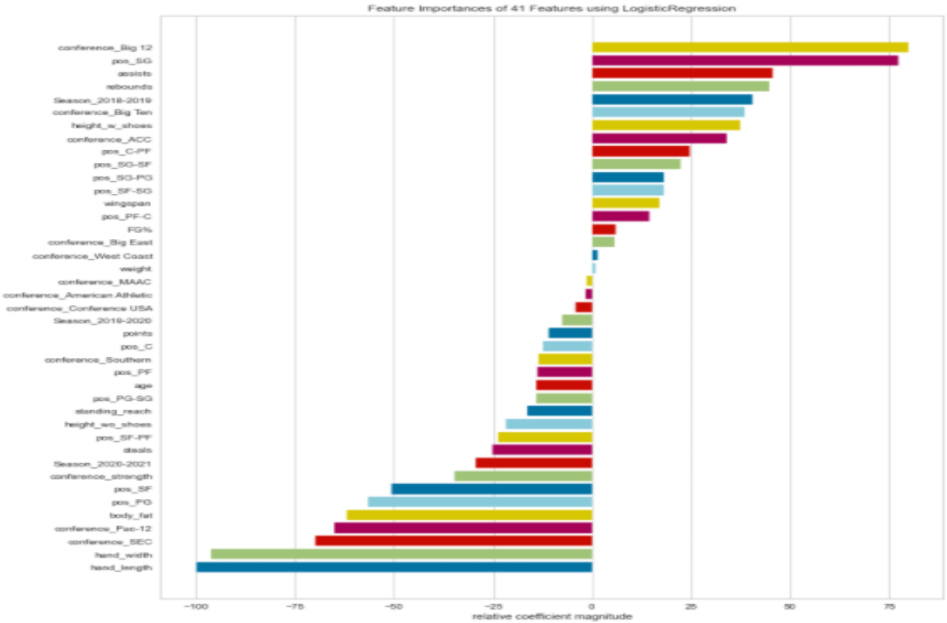
```
Out[22]: [[(6.319950990402289, 0.011938692863666398), 22],
 [(6.216111781727987, 0.012659282812268953), 17],
 [(5.304668241870745, 0.021268349767189342), 19],
 [(1.727548545116096 8, 0.1887242739977153 5), 25],
 [(1.7110731979884526, 0.1908461877398134), 24],
 [(1.6223186440677964, 0.20276878862699652), 12],
 [(1.2732842878239785, 0.2591514528057774), 2],
 [(0.7589281318057003, 0.3836641693271503), 8],
 [(0.7335050847457627, 0.3917494080994335), 0],
 [(0.7185355932203388, 0.39662468159193176), 13],
 [(0.43327782959365624, 0.5103849348178284), 7],
 [(0.3319661641227331, 0.564503626588041 7), 4],
 [(0.3273444815254835, 0.5672269532782037), 18],
 [(0.2423222287161504, 0.6225338869994104), 14],
 [(0.20227030400860913, 0.6528945291359994), 10],
 [(0.14846790775215338, 0.7000038009884428), 21],
 [(0.14846790775215338, 0.7000038009884428), 20],
 [(0.14490625232103876, 0.7034514431458619), 9],
 [(0.07985312886736613, 0.7774965446680049), 6],
 [(0.05263997019929219, 0.8185315960088253), 5],
 [(0.050805546995377426, 0.8216674438699264), 23],
 [(0.046334140435835336, 0.8295695525330888), 16],
 [(0.0447652258535398, 0.8324363188989854), 3],
 [(0.010063289461225252, 0.9200934965349331), 11],
 [(0.0011265421075725413, 0.9732248297841373), 15],
 [(0.00031264602183422836, 0.9858927005385975), 1]]
```

According to the chi-square test tells us the following columns should be the most significant categorical columns to include in our analysis:

- Column 22 - Conference_Pac-12
- Column 17 - Conference_Big 12
- Column 19 - Conference_Big Ten
- Column 25 - Conference_West Cost
- Column 24 - Conference_Southern
- Column 12 - pos_SG

Yellowbrick

```
predictors = ['body_fat','hand_length','hand_width','height_wo_shoes','height_w_shoes','standing_reach','weight','wingspan',
        'age','conference_strength','points','rebounds','assists','steals','FG%','Season_2018-2019','Season_2019-2020',
 'Season_2020-2021',
 'pos_C',
 'pos_C-PF',
 'pos_PF',
 'pos_PF-C',
 'pos_PG',
 'pos_PG-SG',
 'pos_SF',
 'pos_SF-PF',
 'pos_SF-SG',
 'pos_SG',
 'pos_SG-PG',
 'pos_SG-SF',
 'conference_ACC',
 'conference_American Athletic',
 'conference_Big 12',
 'conference_Big East',
 'conference_Big Ten',
 'conference_Conference USA',
 'conference_MAAC',
 'conference_Pac-12',
 'conference_SEC',
 'conference_Southern',
 'conference_West Coast']
X = df2[predictors]
y = df2.draft

lr = LogisticRegression(random_state=42)
fig, ax = plt.subplots(figsize=(12, 12))
fi_viz = FeatureImportances(lr)
fi_viz.fit(X, y)
fi_viz.poof()
```



According to the plot above, the top features:

- conference_Big 12
- pos_SG
- assists
- rebounds
- hand_length
- hand_width
- conference_SEC
- conference_Pac-12
- body_fat
- steals
- height_w_shoes

Predictors Groups

```
predictors1 = [ 'pos_C', 'pos_C-PF','pos_PF','pos_PF-C','pos_PG','pos_PG-SG','pos_SF','pos_SF-PF',
               'pos_SF-SG','pos_SG','pos_SG-PG','pos_SG-SF','assists', 'rebounds','hand_length']
predictors2 = [ 'pos_C', 'pos_C-PF', 'pos_PF', 'pos_PF-C', 'pos_PG', 'pos_PG-SG', 'pos_SF', 'pos_SF-PF',
               'pos_SF-SG', 'pos_SG', 'pos_SG-PG', 'pos_SG-SF',
               'conference_ACC', 'conference_American Athletic', 'conference_Big 12', 'conference_Big East',
               'conference_Big Ten', 'conference_Conference USA', 'conference_MAAC',
               'conference_Pac-12', 'conference_SEC', 'conference_Southern','conference_West Coast','assists',
               'steals','hand_length','conference_strength','body_fat']
```

Fix imbalance data and standardize data

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state = 420)
Xs_train, ys_train = sm.fit_resample(X_train, y_train.ravel())

#Standardize the data
scaler = StandardScaler()
scaler.fit(X_train)
Xs_train = scaler.transform(X_train)
Xs_test = scaler.transform(X_test)
```

Null Error Rate

```
#Null error rate
NRE = df2.draft.value_counts()[1]/(df2.draft.value_counts()[0] + df2.draft.value_counts()[1])
NRE
```

```
0.6793478260869565
```

After SMOTE the data:

```
#Training null error rate
training1_NRE = y_train.value_counts()[1]/(y_train.value_counts()[0] + y_train.value_counts()[1])
Null_Error_Rate= 1 - training1_NRE
Null_Error_Rate
```

```
0.5000488742628132
```

Model Structure
Logistic Regression Model

Model 1

```
X1 = df2[predictors1]
y1 = df2.draft

#Use 33% data to train
#Use random state number 20 to make sure result is fixed
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.33, random_state=20)

#Define the model type as logistic regression
model = LogisticRegression()

#Train the algorithm
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

Model 2

```
X2 = df2[predictors2]
y2 = df2.draft

#Use 33% data to train
#Use random state number 20 to make sure result is fixed
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.33, random_state=20)

#Define the model type as logistic regression
model = LogisticRegression()

#Train the algorithm
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

SVM model

Model 1

```
#Response and explanatory variables
X3 = df2[predictors1]
y3 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X3, y3, test_size=0.33, random_state=20)
```

```
from sklearn.svm import SVC

clf1=SVC(random_state = 42, C = 100, gamma = 0.1, kernel = "rbf")
clf1.fit(X3_train, y_train)
clf1.score(X3_train, y_train)
```

Model 2

```
#Response and explanatory variables
X4 = df2[predictors2]
y4 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X4, y4, test_size=0.33, random_state=20)
```

```
clf2=SVC(random_state = 42, C = 100, gamma = 0.1, kernel = "rbf")
clf2.fit(X4_train, y_train)
clf2.score(X4_train, y_train)
```

KNN Model

Model 1

```
#Response and explanatory variables
X5 = df2[predictors1]
y5 = df2.draft


#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X5, y5, test_size=0.33, random_state=20)

#Create a model
KNN_Classifier = KNeighborsClassifier(n_neighbors = 5, p=2, metric = 'euclidean')

#Train the model
KNN_Classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(metric='euclidean')
```

Model 2

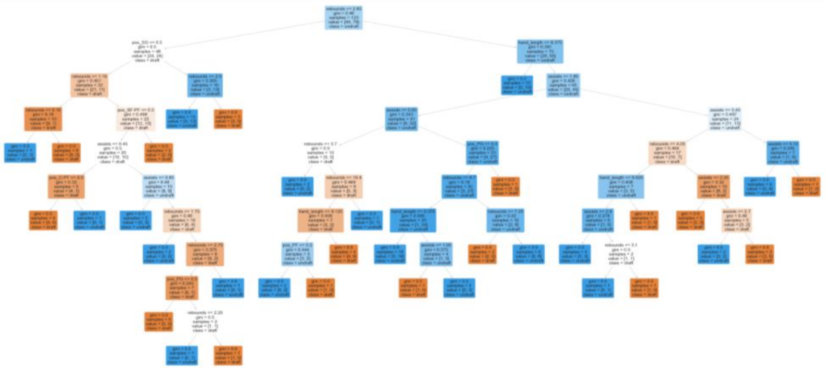
```
#Response and explanatory variables
X6 = df2[predictors2]
y6 = df2.draft


#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X6, y6, test_size=0.33, random_state=20)

#Create a model
KNN_Classifier = KNeighborsClassifier(n_neighbors = 5, p=2, metric = 'euclidean')

#Train the model
KNN_Classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(metric='euclidean')
```

Decision Tree

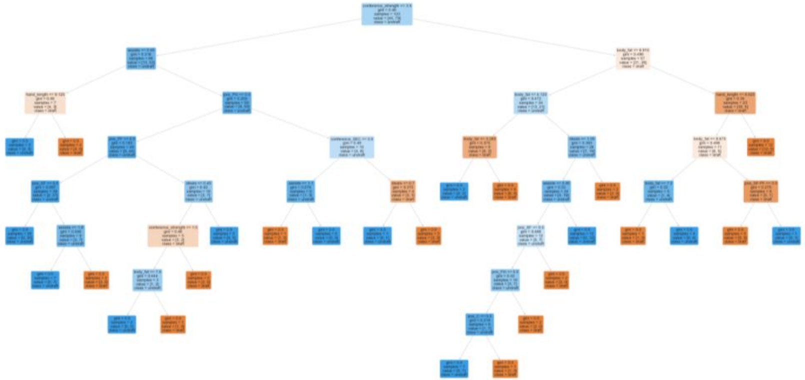
Model 1

```python
#Create decision tree and fit it to the training data
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

#Response and explanatory variables
X7= df2[predictors1]
y7 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X7, y7, test_size=0.33, random_state=20)


clf_dt1 = DecisionTreeClassifier(random_state=20, criterion='gini', max_depth = 15)
clf_dt1 = clf_dt1.fit(X_train, y_train)
```



Model 2

```python
X8= df2[predictors2]
y8 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X8, y8, test_size=0.33, random_state=20)


clf_dt2 = DecisionTreeClassifier(random_state=20, criterion='gini', max_depth = 15)
clf_dt2 = clf_dt1.fit(X_train, y_train)
```



Random Forest

Model 1

```
#Create first random forest model and fit it to the training data
from sklearn.ensemble import RandomForestClassifier


#Response and explanatory variables
X9= df2[predictors1]
y9 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state
X_train, X_test, y_train, y_test = train_test_split(X9, y9, test_size=0.33, random_state=20)

rnd_clf1 = RandomForestClassifier(n_estimators=500, max_leaf_nodes=15, random_state=420)
rnd_clf1.fit(X_train, y_train)
```

```
RandomForestClassifier(max_leaf_nodes=15, n_estimators=500, random_state=420)
```

Model 2

```
#Create first random forest model and fit it to the training data
from sklearn.ensemble import RandomForestClassifier

#Response and explanatory variables
X10= df2[predictors2]
y10 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X10, y10, test_size=0.33, random_state=20)

rnd_clf2 = RandomForestClassifier(n_estimators=500, max_leaf_nodes=15, random_state=420)
rnd_clf2.fit(X_train, y_train)
```

```
RandomForestClassifier(max_leaf_nodes=15, n_estimators=500, random_state=420)
```

XG Boost Classifier Model

```
#https://gist.github.com/wrwr/3f6b66bf4ee01bf48be965f60d14454d
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

#import xgboost as xgb

#param_grid = {
        #'silent': [False],
        #'max_depth': [6, 10, 15, 20],
        #'learning_rate': [0.001, 0.01, 0.1, 0.2, 0,3],
        #'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        #'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        #'colsample_bylevel': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        #'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
        #'gamma': [0, 0.25, 0.5, 1.0],
        #'reg_lambda': [0.1, 1.0, 5.0, 10.0, 50.0, 100.0],
        #'n_estimators': [100]}

#best_xgb = GridSearchCV(xgb.XGBClassifier(), param_grid, scoring='accuracy',
                            #n_jobs=4, refit=False, cv=2)

#best_xgb.fit(X_train, y_train)

#best_score = best_xgb.best_score_
#best_params = best_xgb.best_params_
#print("Best score: {}".format(best_score))
#print("Best params: ")
#for param_name in sorted(best_params.keys()):
    #print('%s: %r' % (param_name, best_params[param_name]))
```

This took about 46 minutes to run with the above specifications so I have commented it out. The results were as follows:

Best params	
colsample_bylevel:	0.7
colsample_bytree:	0.9
gamma:	0
learning_rate:	0.1
max_depth:	15
min_child_weight	3.0
n_estimators:	100
reg_lambda:	1.0
silent:	False
subsample:	0.8

Model 1

```
#Create first XGBoost Classifier and fit it to the training data using our ideal parameters.
import xgboost as xgb

xgb_clf1 = xgb.XGBClassifier(colsample_bylevel=0.7,
colsample_bytree=0.9,
gamma=0,
learning_rate=0.1,
max_depth=15,
min_child_weight=3.0,
n_estimators=100,
reg_lambda=1.0,
silent=False,
subsample=0.8,
random_state = 10)

X11= df2[predictors1]
y11 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X11, y11, test_size=0.33, random_state=20)


xgb_clf1.fit(X_train, y_train)
```

Model 2

```
#Create second XGBoost Classifier and fit it to the training data using our ideal parameters.
import xgboost as xgb

xgb_clf2 = xgb.XGBClassifier(colsample_bylevel=0.7,
colsample_bytree=0.9,
gamma=0,
learning_rate=0.1,
max_depth=15,
min_child_weight=3.0,
n_estimators=100,
reg_lambda=1.0,
silent=False,
subsample=0.8,
random_state = 10)

X12= df2[predictors2]
y12 = df2.draft

#Prepare data for classification process, use 2/3rds of the data to train, 1/3rd to test, random state of 20.
X_train, X_test, y_train, y_test = train_test_split(X12, y12, test_size=0.33, random_state=20)


xgb_clf2.fit(X_train, y_train)
```

A summary of all the models' performance

Metrics	LG 1	LG 2	SVM 1	SVM 2	KNN 1	KNN 2	DT 1	DC 2	RF 1	RF 2	XG 1	XG 2
Number of Features	15	27	15	27	15	27	15	27	15	27	15	27
Precision	72.1	78.6	90.0	100.0	75.8	76.0	100.0	100.0	87.0	92.0	82.0	87.0
Recall	73.0	77.0	92.0	100.0	76.0	75.0	100.0	100.0	83.0	90.0	82.0	87.0
F1 Score	82.0	83.3	94.0	100.0	82.8	82.0	100.0	100.0	81.0	90.0	82.0	87.0
Accuracy	73.2	77.2	92.0	100.0	75.6	75.0	100.0	100.0	83.0	90.0	82.0	87.0
Cross-validation	65.3	67.4	56.1	57.6	62.4	67.2	69.0	69.0	63.5	68.3	65.0	68.1

- From player's perspective, the accuracy score is more important.
- From GM or Coach's perspective, the recall score is more important.

Model Selection

```
#Predict values using test data and second Random Forestmodel.
predsm = rnd_clf2.predict(X_test)
```

```
#Import confusion matrix
from sklearn.metrics import confusion_matrix
confusion = confusion_matrix(y_test,predsm)
print('Confusion Matrix\n')
print(confusion)

from sklearn.metrics import classification_report
print('\nClassification Report\n')
print(classification_report(y_test, predsm, target_names=['draft','undraft']))
```

```
Confusion Matrix

[[ 2 13]
 [ 6 40]]

Classification Report

              precision    recall  f1-score   support

       draft       0.25      0.13      0.17        15
     undraft       0.75      0.87      0.81        46

    accuracy                           0.69        61
   macro avg       0.50      0.50      0.49        61
weighted avg       0.63      0.69      0.65        61
```

```
from sklearn.model_selection import cross_val_score
scoresF = cross_val_score(rnd_clf2, X_test, y_test, cv=10, scoring='accuracy')
print(scoresF)
print(scoresF.mean())
```

```
[0.57142857 0.66666667 0.66666667 0.33333333 0.66666667 0.83333333
 0.66666667 0.83333333 0.83333333 0.83333333]
0.6904761904761905
```

The second model of Random Forest: has 2 true negatives, 13 false positives, 6 false negatives, and 40 true positives.

- Precision is 63%
- Recall is 69%
- F1 score is 65%
- Accuracy is 69%
- Avg cross-validation is 69.0%

Key Findings

Random Forest model 2 which utilized conference, position, assists, steals, hand_length, conference_strength, and body_fat.

The value of conference level.

The strength of different machine learning models.

My Recommendations for Further Analysis

Inaccessible Data/feature: Medical Records

Building a Bigger Database

THANKS FOR LISTENING!

Reference

- https://www.nba.com/stats/draft/combine-anthro/?SeasonYear=2020-21
- https://www.warrennolan.com/basketball/2021/conferencenet
- From college to the NBA: what determines a player's success and what characteristics are NBA franchises overlooking?: Applied Economics Letters: Vol 25, No 5 (tandfonline.com)
- The Ranking Prediction of NBA Playoffs Based on Improved PageRank Algorithm (hindawi.com)
- Anchoring bias in the evaluation of basketball players: A closer look at NBA draft decision-making - Berger - 2021 - Managerial and Decision Economics - Wiley Online Library
- Left atrial size and strain in elite athletes: A cross-sectional study at the NBA Draft Combine - Cheema - 2020 - Echocardiography - Wiley Online Library