Southern Connecticut State University
DSC 101, Spring 2021

Machine Learning: Predicting NBA Playoffs Using K-Nearest Neighbor

James Weaver

May 13^{th}, 2021

Instructor ..... Raymond Mugno

Contents

1 Introduction

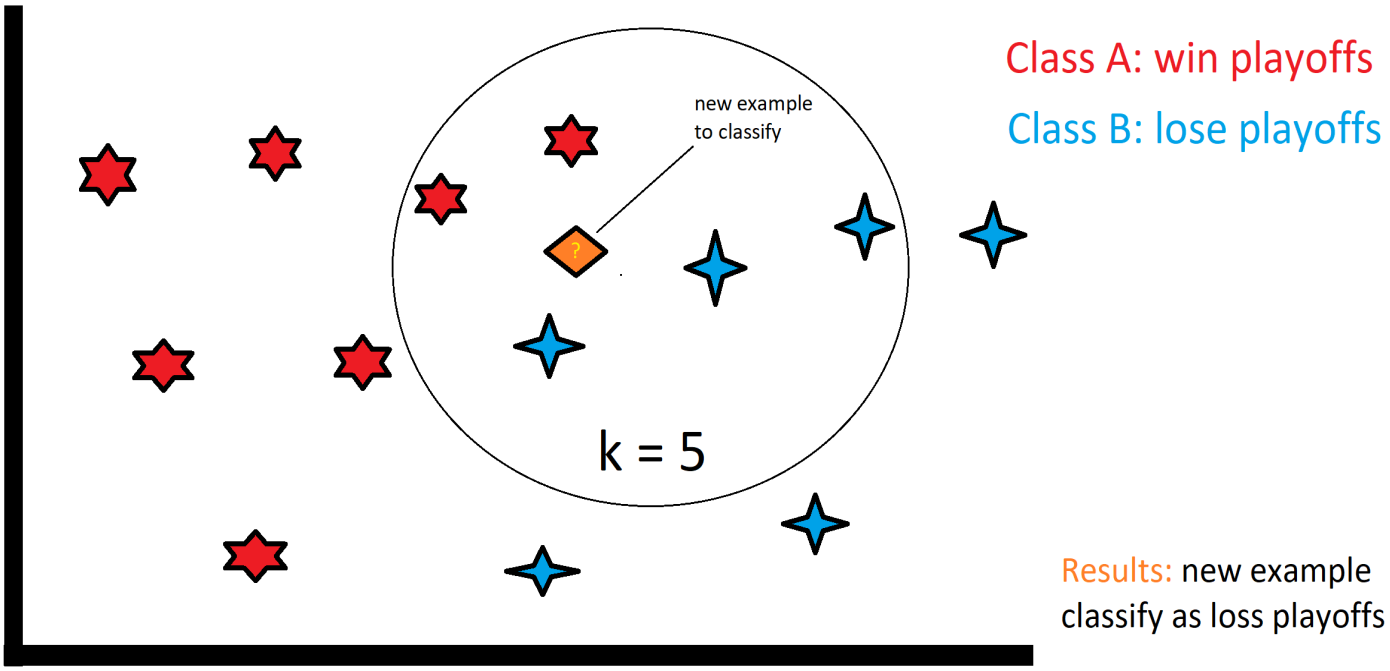
1.1 Purpose

Sports betting has been around for over hundreds of years. It started to become popular in the early 18th century when people would bet on horse races. Even further back, the earliest record of sports betting sets back over 2000 years ago when the Greeks would bet on athletic competitions during the Olympics. With the internet being as big as it ever has, sports betting has become profitable but is extremely risky because many factors can influence the outcome. One of the most popular sports that is used in sports betting is basketball. Hence, this research seeks to find a model to predict whether or not a particular basketball team will win the NBA playoffs based on a set of evaluation metrics.

1.2 K-Nearest Neighbor Model

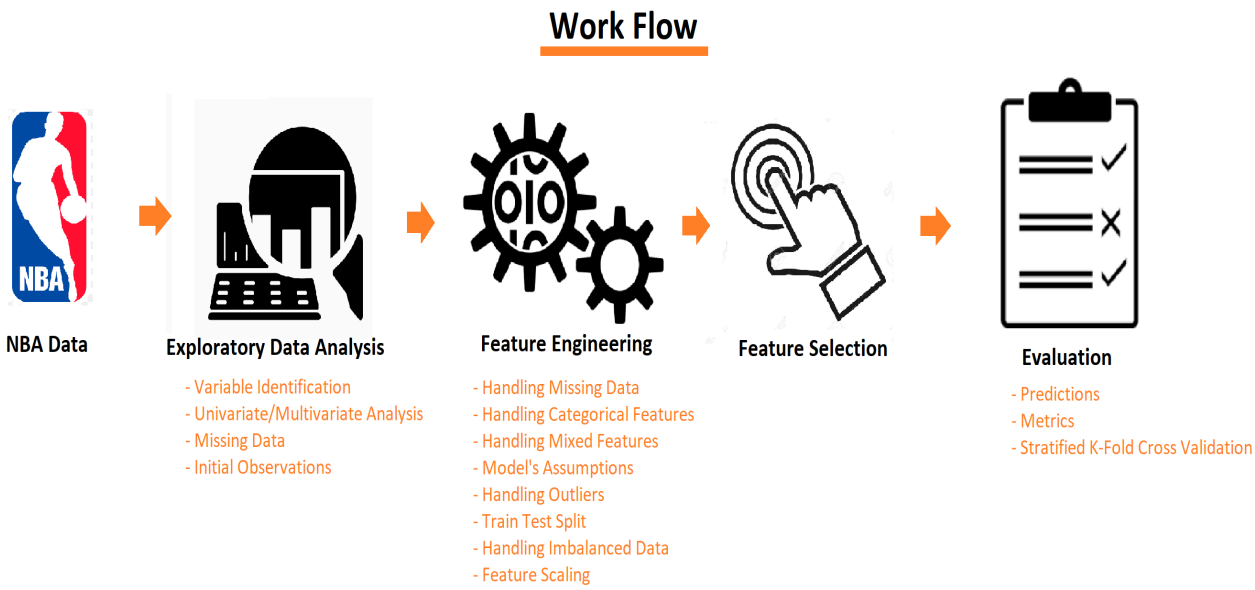
The algorithm that was used in this research to make the model was k-nearest neighbor(knn) classifier. Knn is one of the simplest machine learning algorithms that makes no mathematical assumptions. The only things it requires are the number k of neighbors and a distance metric. Knn determines the class of a data point by majority voting principle. If k is set to n, the classes of n closest points are checked. Prediction is done according to the majority class. I choose 5 as the k and euclidean as the distance metric.

Figure 1: *Knn Example*



1.3 Workflow

Figure 2: *Work Flow*



Here is the workflow on how to build the model. First the NBA data needs to be uploaded and have an exploratory data analysis on it to understand which features will help predict a team winning the NBA playoffs. For feature engineering, you want to make sure all features are numeric, there are no missing values(NaN), and fix similar issues or the model won't be able to work. After the feature engineering and training the model, feature selection is the next step, and the goal is to pick the most important features. Then the model is ready to be evaluated to determine how good it will perform on new data.

2 Exploratory Data Analysis

Figure 3: *First 5 Observations*

Observations	Season	TEAM	Conference Standings	All Stars	MVP	Offs Games	GP	W	L	WIN%	...	AST.1	OREB%	DREB%	REB%	TOV%	EFG%	TS%	PACE	PIE	POSS
1	2002-2003	Dallas Mavericks	3	2	NaN	10	82	60	22	0.732	...	17.3	27.7	67.7	47.8	12.3	49.8	54.3	93.75	54.6	7,726
2	2002-2003	San Antonio Spurs	1	1	1	16	82	60	22	0.732	...	15.9	31.1	68.7	50.8	17.1	49.7	54.1	91.46	54.9	7,558
3	2002-2003	Sacramento Kings	2	2	NaN	7	82	59	23	0.72	...	18.3	28.9	67.8	49.3	14.9	49.9	53.5	96.63	55.1	7,987
4	2002-2003	Minnesota Timberwolves	4	1	NaN	2	82	51	31	0.622	...	19	31	69.5	50.8	14.6	48.8	52.8	93.08	52.9	7,681
5	2002-2003	Detroit Pistons	1	1	NaN	8	82	50	32	0.61	...	16.2	29.4	70.7	49.9	15.1	47.3	52.3	88.26	52.5	7,318

The data contains 538 observations; however, only the first 5 are shown above. Since we are interested in predicting whether or not a basketball team will win the NBA playoffs, this will be the target variable. Here are the names of the potential features that will be used to predict the target variable: season, team, conference standings, all stars, mvp, playoffs games won, GP, W, L, win%', min, pts, fgm, fga, fg%, 3pm, 3pa, 3P%, ftm, fta, ft%, oreb, dreb, reb, ast, tov, stl, blk, blka, pf, ped, +/-, min.1, offrtg, defrtg, netrtg, ast%, ast/to ratio, ast.1, oreb%, dreb%', reb%, tov%, efg%, ts%, pace, pie, and poss. Since there are a large number of features, we will later discuss their descriptions in the feature selection where you eliminate unimportant features.

2.1 Variable Identification

Figure 4: *Variable Types*

Variable Identification			
Discrete Variables	Continuous Variables	Categorical Variables	Mixed Variables
All Stars	Win%	Team	Season
GP	Min		Poss
W	Pts		
L	Fgm		
MIN.1	Fga		
Conference Standings	Fg%		
MVP	3pm		
	3pa		
	3p%		
	Ftm		
	Fta		
	Ft%		
	Oreb		
	Dreb		
	Reb		
	Ast		
	Tov		
	Stl		
	Blk		
	Blka		
	Pf		
	Pfd		
	"+/-"		
	Offrtg		
	Defrtg		
	Netrtg		
	Ast%		
	Ast/To Ratio		
	Ast.1		
	Oreb%		
	Dreb%		
	Tov%		
	Efg%		
	Ts%		
	Pace		
	Pie		

2.2 Univariate And Multivariate Analysis

Figure 5: *Description Of Continuous Features*

	WIN%	MIN	PTS	FGM	FGA	FG%	3PM	3PA	3P%	FTM	...	AST/TO Ratio	AST.1	OREB%	DREB%	REB%	TOV%	EFG%	TS%	PACE	PIE
count	538	538	538	538	538	538	538	538	538	538	...	538	538	538	538	538	538	538	538	538	538
mean	0.499717	48.357993	100.813	37.5615	82.7342	45.3989	7.51952	21.0872	35.5106	18.1652	...	1.532862	16.8065	29.5545	70.4219	50.0004	15.1777	49.9041	54.0158	94.515	49.9907
std	0.15125	0.178816	6.39573	2.23166	3.7563	1.55908	2.53482	6.84482	1.88303	1.96876	...	0.192925	1.1694	2.76844	2.24405	1.35094	1.2594	2.3396	2.11046	3.64785	3.30887
min	0.106	48	84.2	32.4	74.3	40.8	2.7	7.8	27.8	12.2	...	1.01	14.1	21.6	65	45.3	11.9	42.8	46.9	87.39	41.1
25%	0.39	48.2	96.325	36.025	80.025	44.325	5.7	16.025	34.4	16.7	...	1.4	16	27.6	68.8	49.2	14.3	48.2	52.5	91.8225	47.8
50%	0.512	48.4	99.85	37.3	82.4	45.35	7	19.6	35.5	18.1	...	1.52	16.8	29.7	70.3	50	15.1	49.75	53.9	93.945	50.1
75%	0.61	48.5	104.575	38.9	85.4	46.4	9.2	25.4	36.7	19.4	...	1.66	17.5	31.4	71.9	50.9	16	51.5	55.4	96.825	52.475
max	0.89	49	118.7	44	94	50.4	16.1	45.4	41.6	24.1	...	2.12	21.2	37.3	77.5	54.1	20	56.9	60.3	105.51	58.1

Below are the histograms, boxplots, and scatterplots of the continuous features.

Figure 6: *Graphs Of Continuous Features*

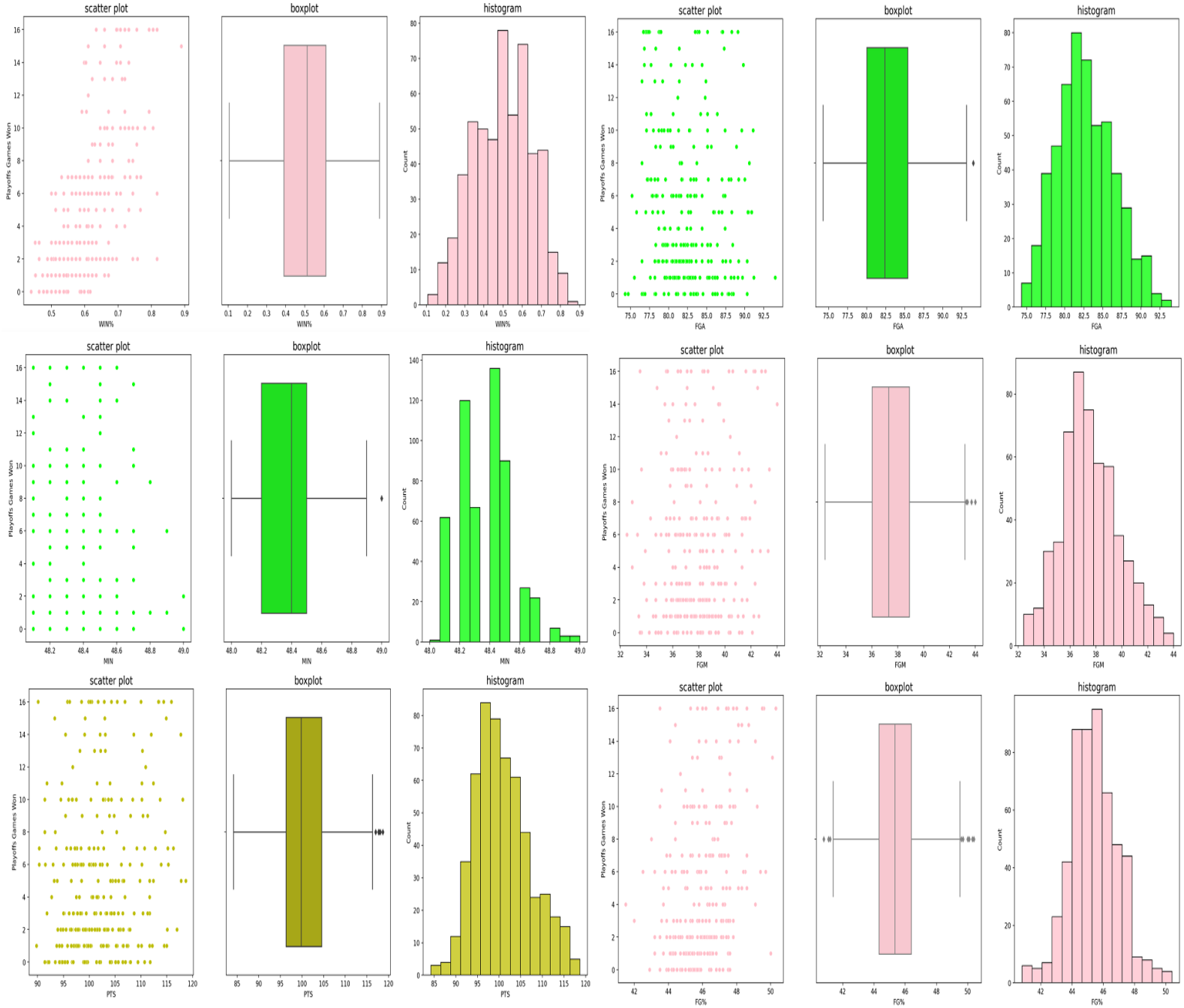


Figure 7: *Graphs Of Continuous Features*

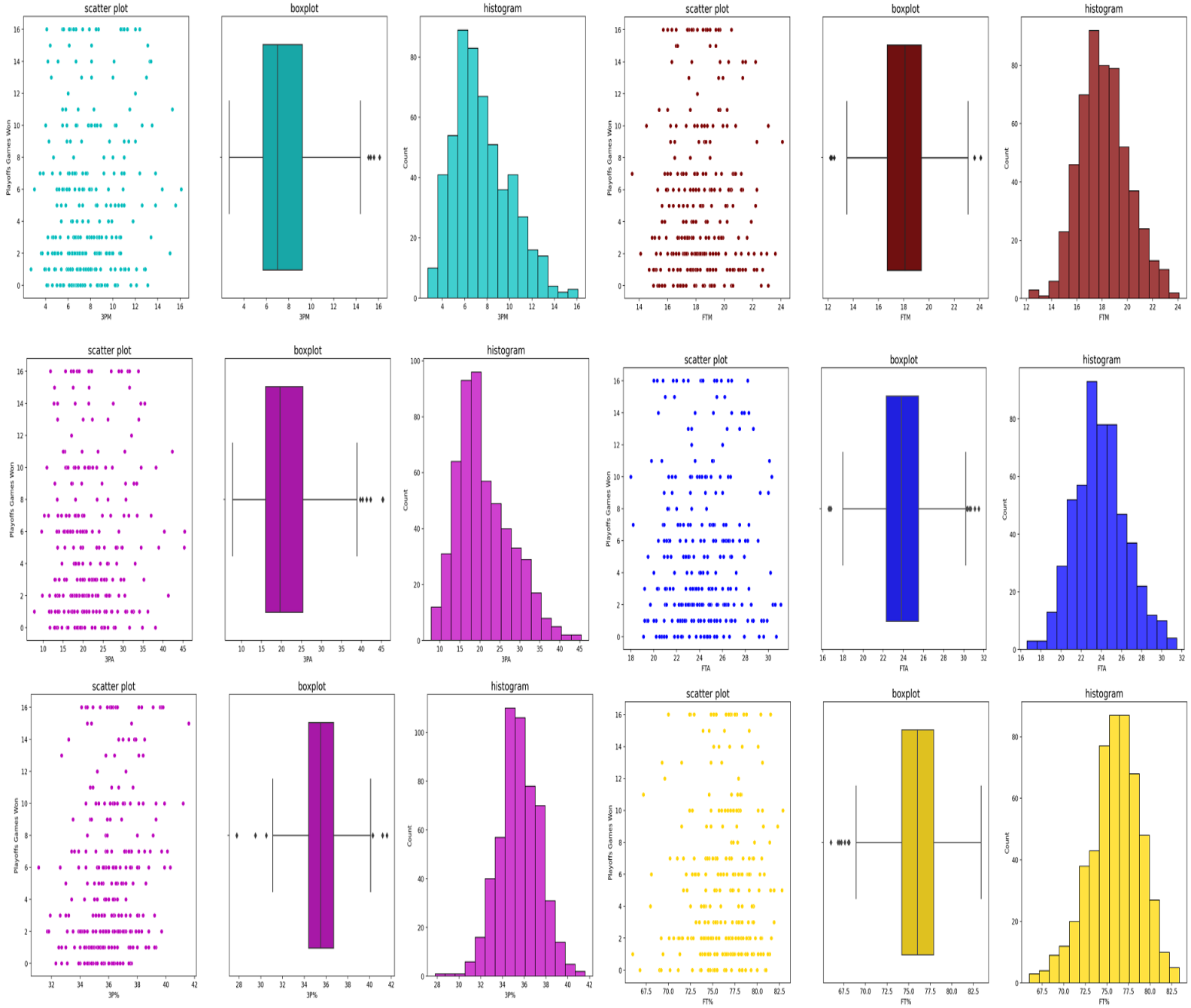


Figure 8: *Graphs Of Continuous Features*

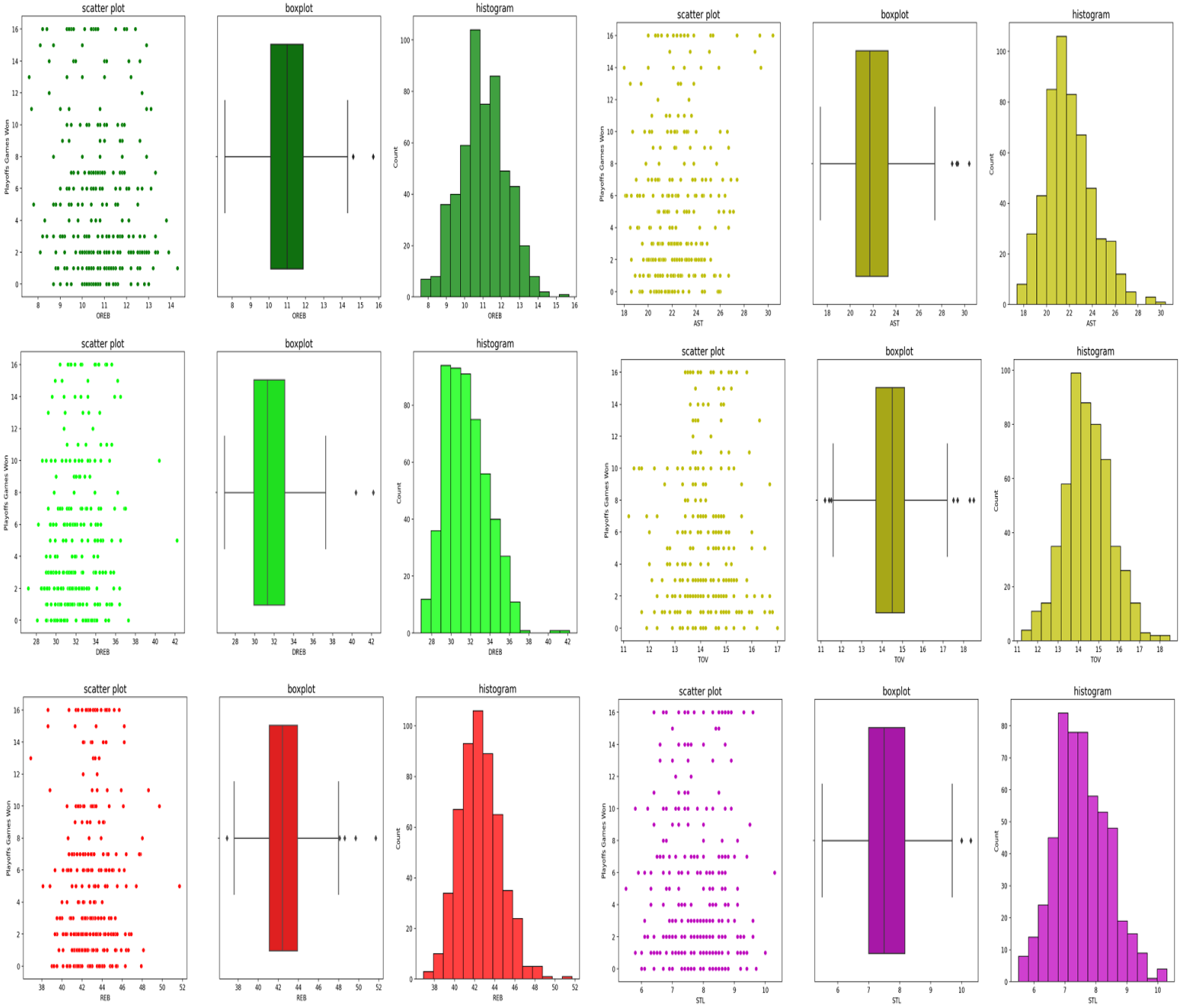


Figure 9: *Graphs Of Continuous Features*

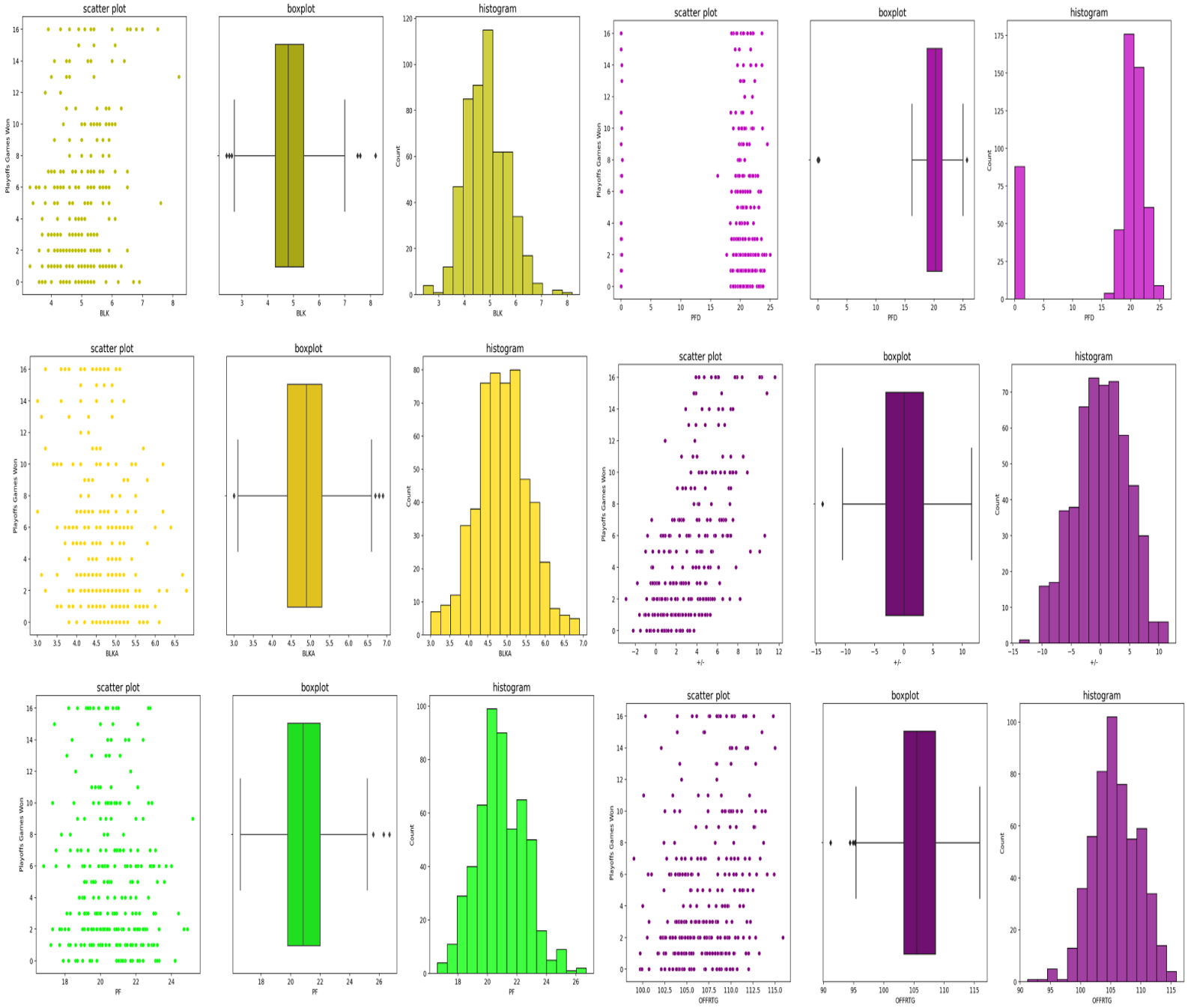


Figure 10: *Graphs Of Continuous Features*

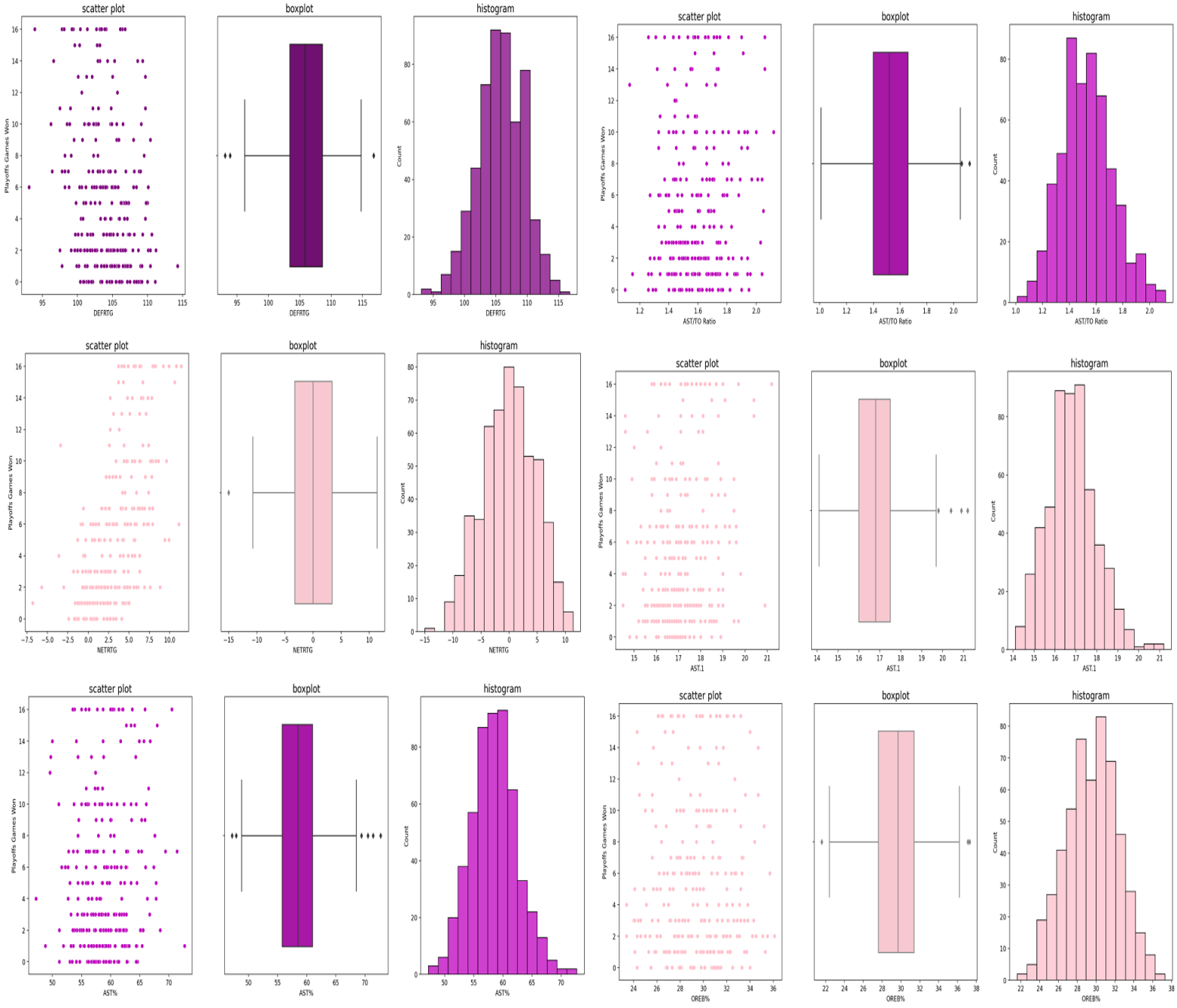


Figure 11: *Graphs Of Continuous Features*

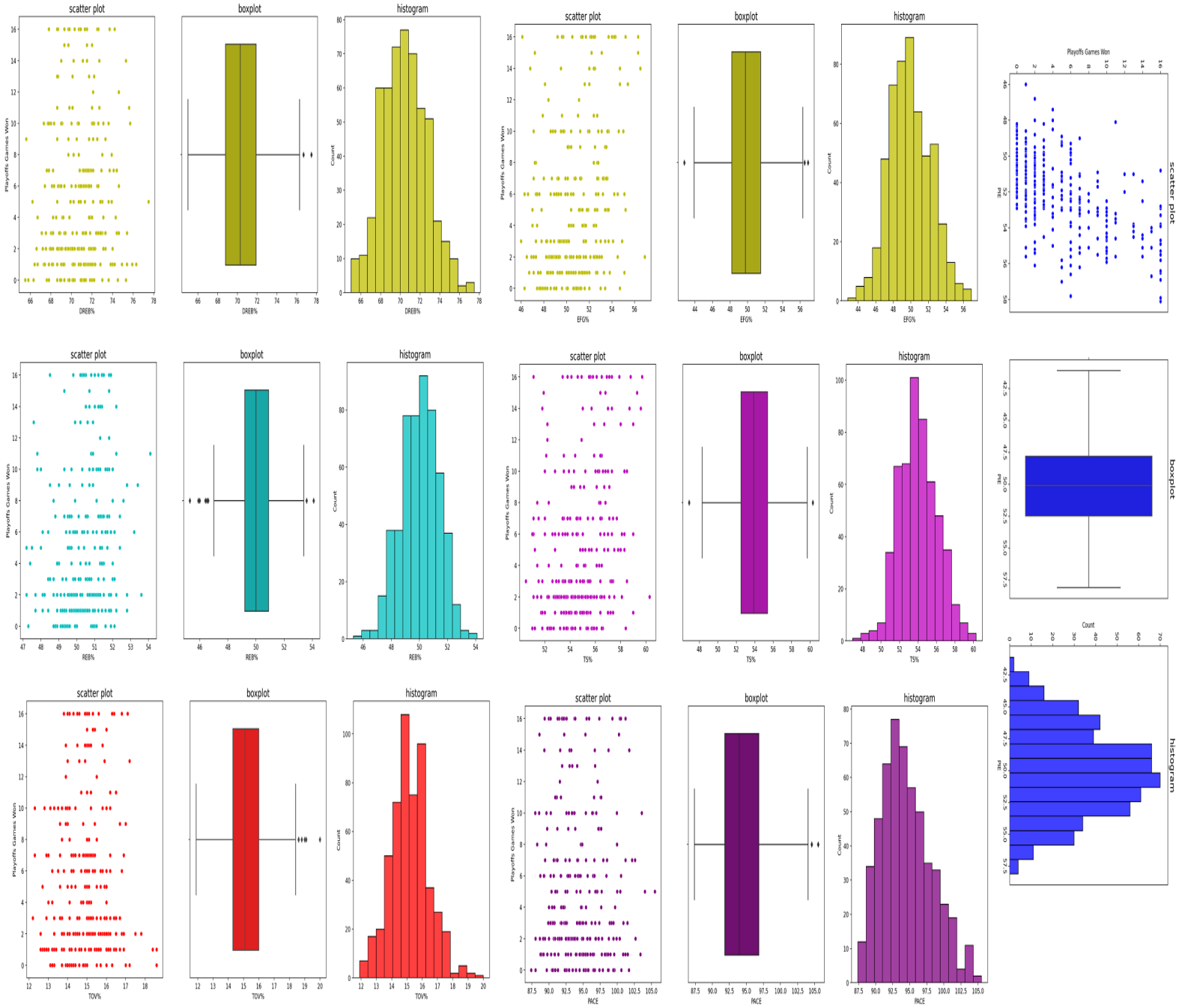
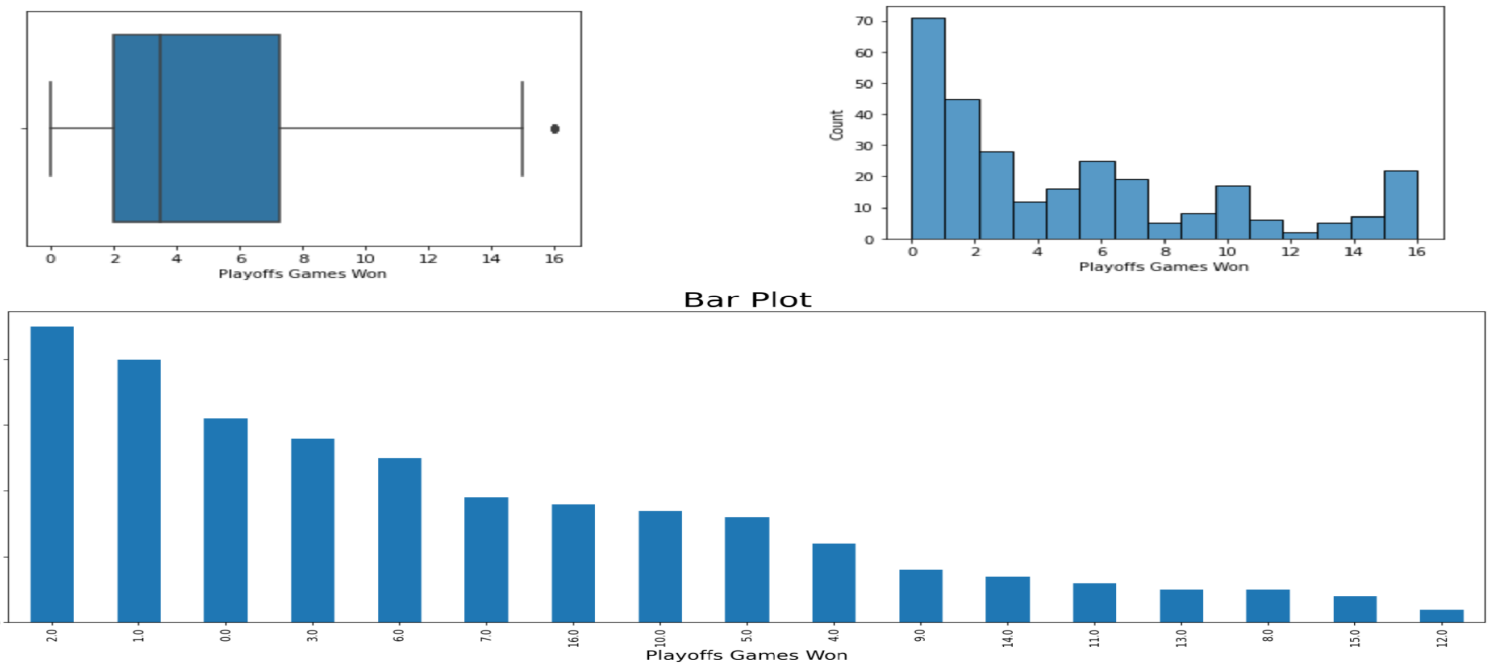


Figure 12: *Description Of Target Variable*



From looking at the graphs for the target variable, you can see that data is imbalanced. If a team wins 16 games, it indicates that they won the NBA playoffs. Otherwise, it means that they lost during the playoffs. Later in the feature engineering, the target variable needs to be converted into a binary(0 and 1), where 1 will represent teams winning the NBA playoffs, and 0 will represent teams that lost.

Figure 13: *Description Of Discrete Features*

	All Stars	GP	W	L	MIN.1	Conference Standings	MVP
count	538	538	538	538	538	288	18
mean	0.860595	80.4684	40.2342	40.2342	3891.569	4.5	1
std	0.870096	4.451259	12.42546	12.31829	215.3059	2.295276	0
min	0	64	7	9	3112	1	1
25%	0	82	32	31	3951	2.75	1
50%	1	82	41	40	3961	4.5	1
75%	1	82	50	49	3971	6.25	1
max	4	82	73	72	4011	8	1

Below are the histograms, boxplots, and scatterplots of the discrete features.

Figure 14: *Graphs Of Discrete Features*

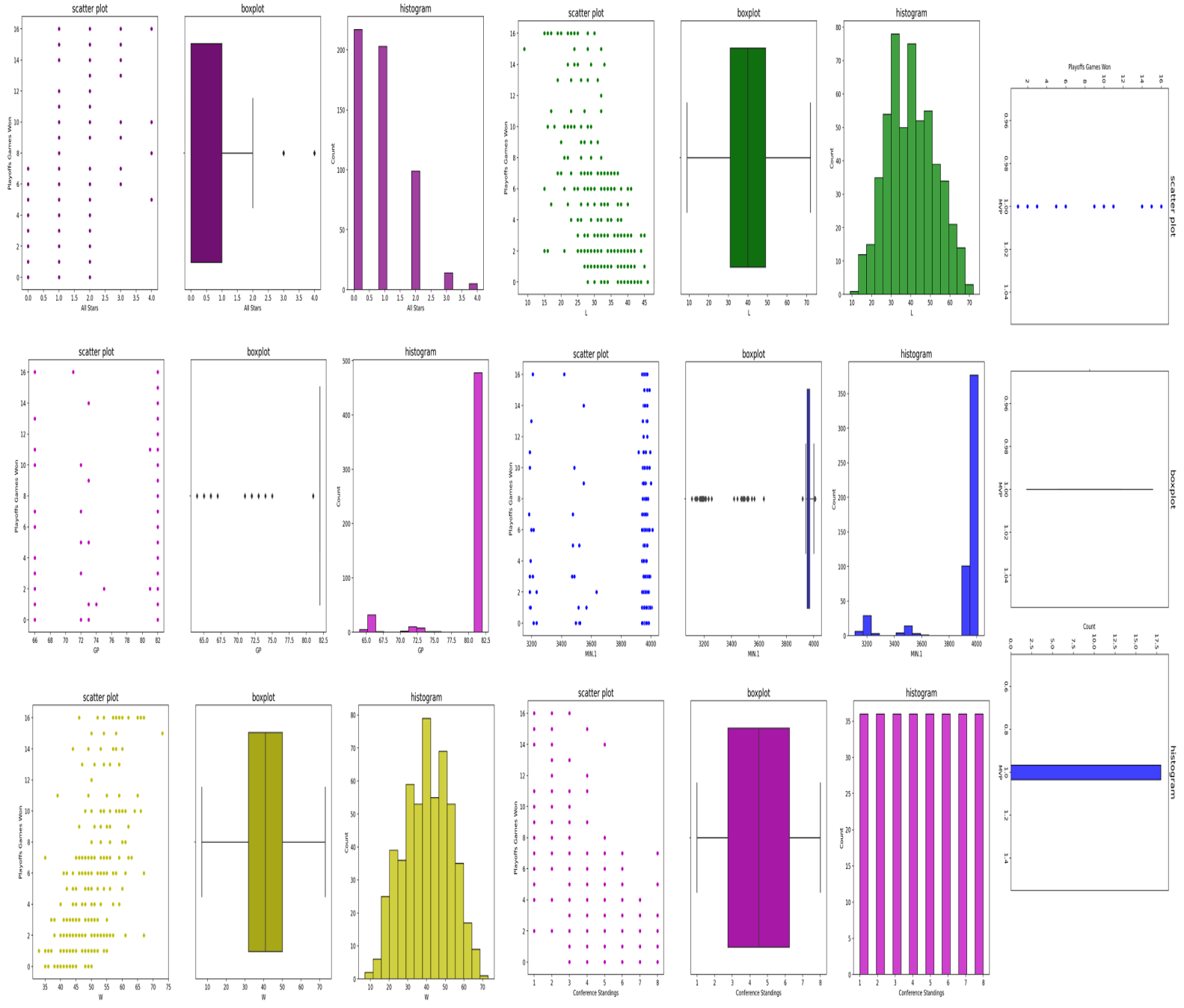
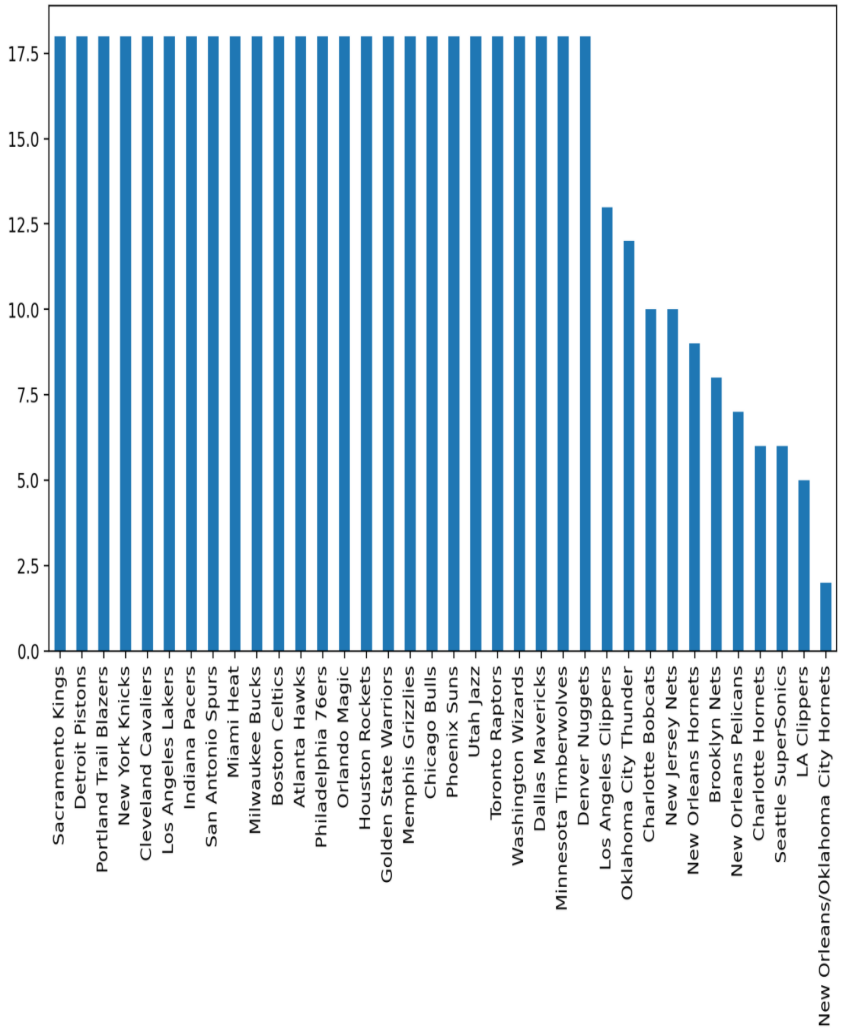
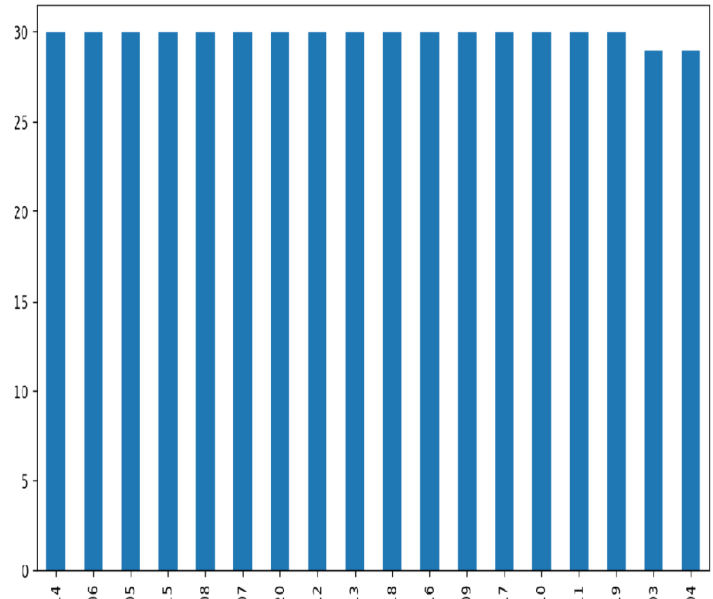


Figure 15: *Mixed And Categorical Features*

TEAM	Season	POSS
Dallas Mavericks	2002-2003	7,726
San Antonio Spurs	2002-2003	7,558
Sacramento Kings	2002-2003	7,987
Minnesota Timberwolves	2002-2003	7,681
Detroit Pistons	2002-2003	7,318





You can Poss contains commas and because of this, it's considered a mixed variable.

2.3 Missing Data

Figure 16: *Percentage Of Missing Values In The Columns*

Columns	Percentage% Of Missing Values
Season	0
Team	0
Conference Standings	46.468401
All Stars	0
Mvp	96.654275
Playoffs Games Won	46.468401
Gp	0
W	0
L	0
Win%	0
Min	0
Pts	0
Fgm	0
Fga	0
Fg%	0
3Pm	0
3Pa	0
3P%	0
Ftm	0
Fta	0
Ft%	0
Oreb	0
Dreb	0
Reb	0
Ast	0
Tov	0
Stl	0
Blk	0
Blka	0
Pf	0
Pfd	0
+/-	0
Min.1	0
Offrtg	0
Defrtg	0
Netrtg	0
Ast%	0
Ast/To Ratio	0
Ast.1	0
Oreb%	0
Dreb%	0
Reb%	0
Tov%	0
Efg%	0
Ts%	0
Pace	0
Pie	0
Poss	0

You can see only the columns *conference standings*, *mvp*, and *playoffs games won* have missing values.

2.4 Initial Observations

The data used in this research are statistics of NBA teams from 2003 to 2020 during a regular season. A regular season usually runs from October to April, with each team playing 82 games. When the regular season ends, the top 16 teams will play each other in the best of 7 elimination tournament, known as the NBA playoffs. The team that wins 16 games is considered the NBA champion. We are interested in finding which features from the data will help predict whether or not a basketball team will win the NBA playoffs based on the statistics from the regular season. At first glance, one can assume the winning percentage would be a great predictor because a high winning percentage would indicate that the team has a high probability of being the NBA champion.

3 Feature Engineering

3.1 Handling Missing Data

3.1.1 Conference Standings

About 46.47% of the *Conference Standings* column have missing values because these were the teams that did not qualify to play in the NBA playoffs. It would make more sense to focus on teams that played in the playoffs, so that the algorithm can learn from less noise. Therefore, I removed all teams that were not qualified. By removing these teams, the data went from 538 to 288 observations.

3.1.2 Playoffs Games Won

About 46.47% of the *Playoffs Games Won* column have missing values for the same reason as the *Conference Standings* column. Since I removed the missing values previously, this would also remove the missing values for this column.

3.1.3 Mvp

About 96.65% of the *Mvp* column have missing values because these teams did not have the NBA Mvp that season playing on their team . The NBA Mvp is the most valuable player of the regular season and is selected by various people from the sports media. To fix the missing data for this column, I used a method called arbitrary value imputation. In the Exploratory Data Analysis section, you can see if the Mvp was on a team, the values were 1's. Otherwise, the values were NaN's(missing data). The arbitrary value imputation method will replace all NaN's with 0's. In the picture, you can now see that there are no more missing values in the data.

Figure 17: *Arbitrary Value Imputation*

Will replace all missing values in the mvp column with zeros

In [903]:
```
# fixing MVP col
def replace_nan(df,variable,arbitrary):
    '''
    will replace the na by an arbitrary choice.
    df=dataframe
    variable=the varibale as a string we want
    arbitrary = the arbitrary numeric number
    '''
    df[variable]=df[variable].fillna(arbitrary)

replace_nan(df1,'MVP',0)
```

In [904]:
```
df1['MVP'].isnull().sum()
```

Out[904]:
```
0
```

In [905]:
```
df1.isnull().sum()
```

Out[905]:

Season	0
TEAM	0
Conference Standings	0
All Stars	0
MVP	0
Playoffs Games Won	0
GP	0
W	0
L	0
WIN%	0
MIN	0
PTS	0
FGM	0
FGA	0
FG%	0
3PM	0
3PA	0
3P%	0
FTM	0
FTA	0
FT%	0
OREB	0
DREB	0
REB	0
AST	0
TOV	0
STL	0
BLK	0
BLKA	0
PF	0
PFD	0
+/-	0
MIN.1	0
OFFRTG	0
DEFRTG	0
NETRTG	0
AST%	0
AST/TO Ratio	0
AST.1	0
OREB%	0
DREB%	0
REB%	0
TOV%	0
EFG%	0
TS%	0
PACE	0
PIE	0
POSS	0

dtype: int64

There are no more missing values

3.2 Handling Categorical Features

In the Exploratory Data Analysis, we saw that the variable *Team* was the only categorical feature. The cardinality for *Team* is 36. Since the cardinality is high, I decided not to use *Team* for the algorithm. For the target variable *Playoffs Games Won*, I used the arbitrary value imputation method again, where if a basketball team won 16 games, I replace the values with 1's, and any values not 16, I replaced the values with 0's. After replacing the target variable to binary, there were 36 teams winning the NBA playoffs and 252 teams losing in the playoffs. Later, we will discuss fixing the imbalanced data.

3.3 Handling Mixed Features

In the Exploratory Data Analysis, we can see that *Season* and *Poss* are mixed features. The cardinality for *Season* is 18. This feature is not important because the data is from 2003-2020, which is 18 seasons. The cardinality for *Poss* is 442. Even though the cardinality is high, the reason why this is a mixed feature is because it contains commas in the numbers. To fix this issue, I deleted all the commas and converted the numbers from strings to floats (numeric).

3.4 Model's Assumption

K-nearest neighbors has no assumptions to fulfill since it is a lazy non-parametric algorithm.

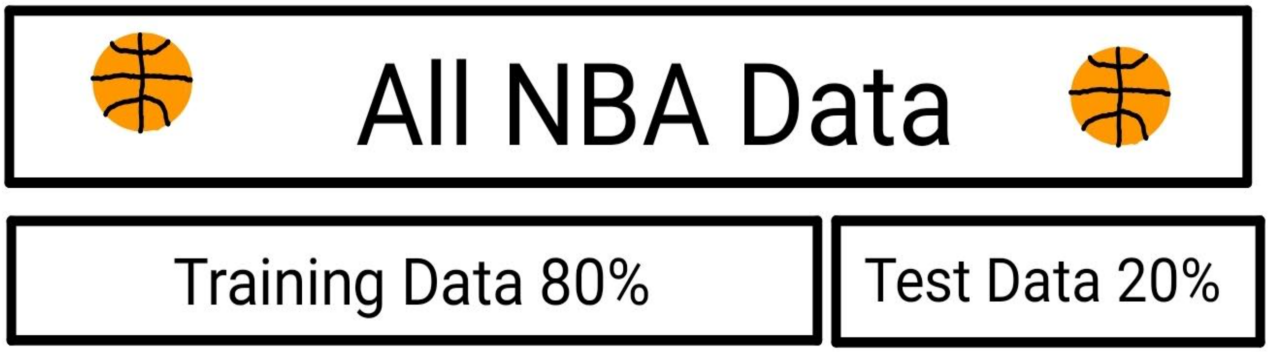
3.5 Handling Outliers

An outlier is an unusual observation. The k-nearest neighbor algorithm is sensitive to outliers because it selects its neighbors based on a distance criteria. I decided not to remove any outliers because I believe these outliers represent the popular of NBA teams that qualify to play in the playoffs since the dataset is small.

3.6 Train Test Split

To prevent the model from data leakage, I separated the entire data into a training set and testing set. An analogy of data leakage is the following scenario. Suppose a student studies for a test and the day before the exam, that student finds the real test online and studies that. After the exam, the student gets a perfect score because he already saw the exam. The student did not learn anything because he 'cheated'. In machine learning, the objective is for the model to perform well in the training set and to perform well on unseen data(testing set) without data leakage. Since the data is not small, I randomly split the entire data, where 80% of the data goes in the training set, and the remaining 20% goes in the testing set. Since the data is also unbalanced, I made sure that the proportion of the target variable in the entire dataset will be the same for the training and testing sets.

Figure 18: *Train Test Split*

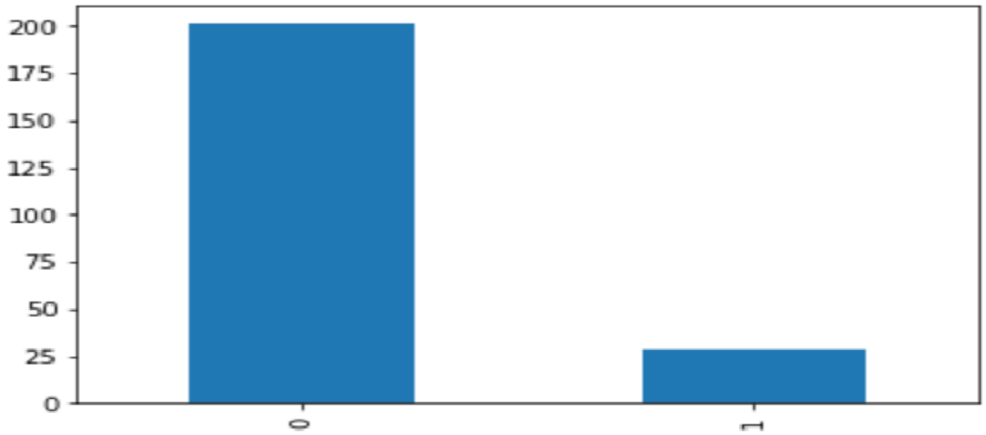



```
X = df1.drop(['Season','TEAM ','Playoffs Games Won'],axis=1)
y = df1['Playoffs Games Won']

# Library for train test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=65421,stratify=y)
# global random state=65421
# stratify will make sure the training/test set will have same  proportion of the target feature in the original data
```

3.7 Handling Imbalanced Data

Figure 19: *Imbalanced Training Set*

Not NBA Champion	201
NBA Champion	29

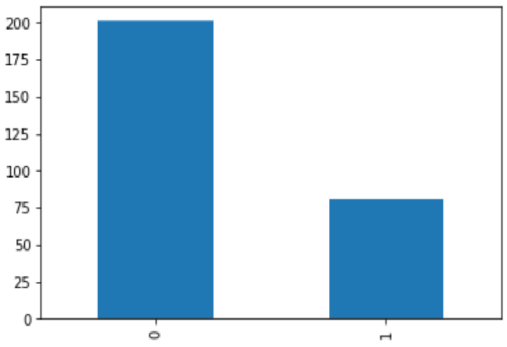


In the image, you can see that the training set is extremely unbalanced. One way to fix an imbalanced data is to use oversampling methods. Oversampling is the process of increasing the number of samples from the minority class, so that the data can be balanced. The oversampling method I used is Borderline SMOTE, which creates synthetic samples only from the observations in the minority class closer to the boundary with the majority class. Since the original data was extremely unbalanced, I decided to add 52 synthetic samples to the minority class.

Figure 20: *Oversampled Training Set*

```
from imblearn.over_sampling import BorderlineSMOTE

number1 = y_train.value_counts()[0] # the number of samples in the majority class
number2 = -int(-(number1*.40)//1) # 40 percent of the number of samples in the majority class

oversample = BorderlineSMOTE(sampling_strategy={0:number1,1:number2},random_state=65421)
X_train_os, y_train_os = oversample.fit_resample(X_train, y_train) # oversample training set
```

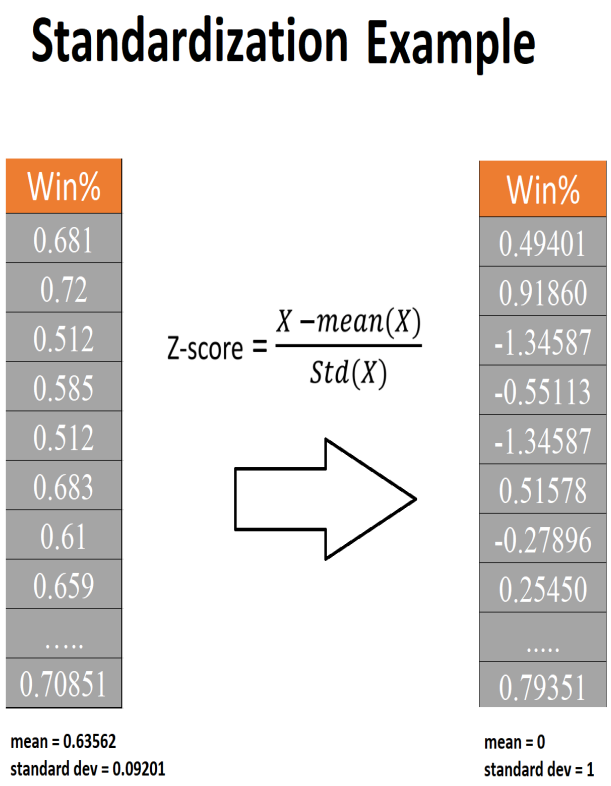
Not NBA Champion	201
NBA Champion	81



3.8 Feature Scaling

It is important to note that features with bigger magnitude dominate the ones with smaller magnitude. Since k-nearest neighbors uses euclidean distances, which are sensitive to feature magnitude, the features need to be scaled where their values will range within a similar scale.

Figure 21: *Feature Scaling Example*



To scale the features, I standardized them which will make each feature have a mean of 0 and standard deviation of 1.

Figure 22: *Scaled Oversample Training Set Code*

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_X_train_os = scaler.fit_transform(X_train_os)
scaled_X_test = scaler.transform(X_test)
```

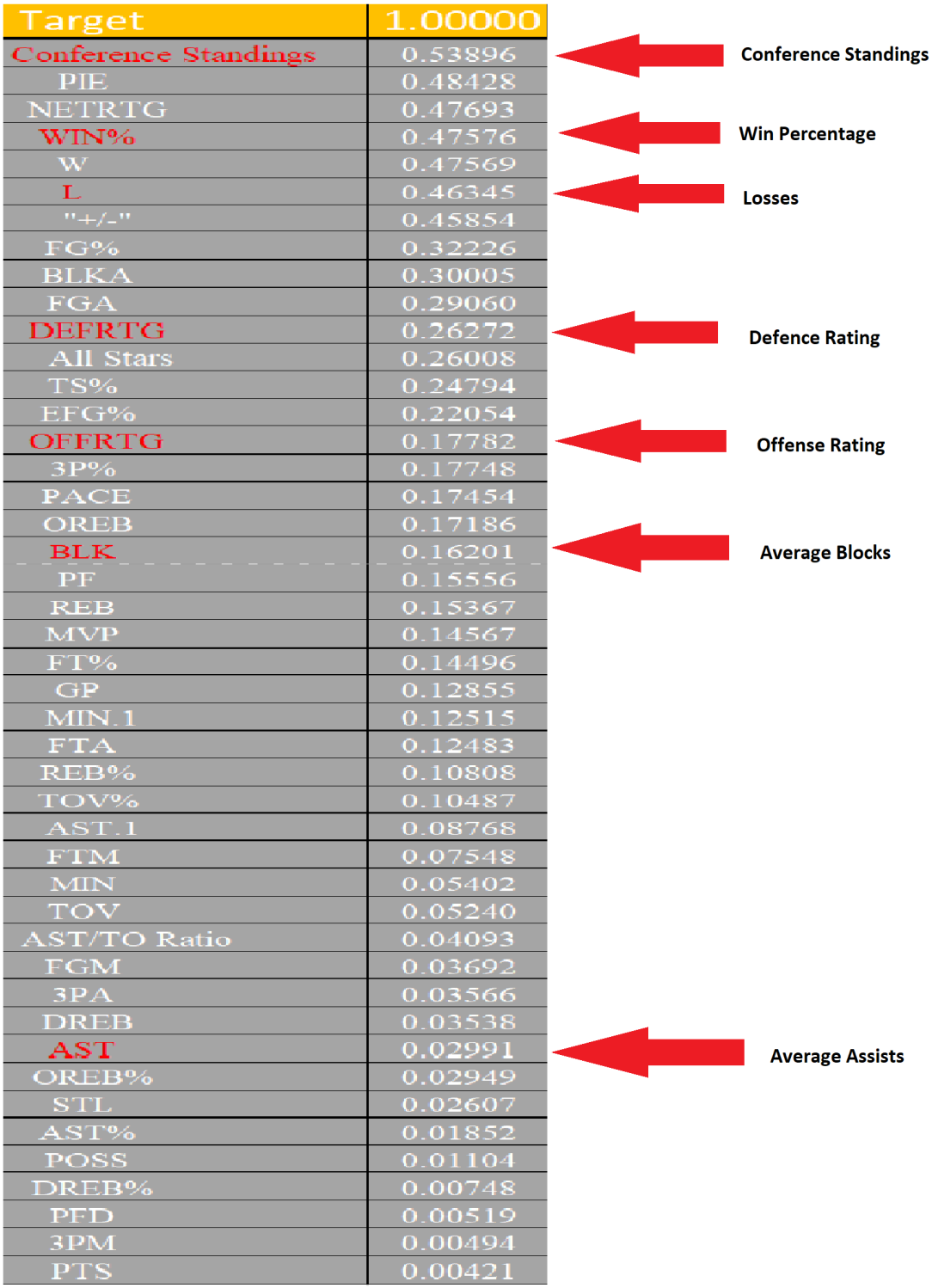
4 Feature Selection

Feature selection is the procedure of selecting a subset of important features for use in a machine learning model building. Since there are many features, picking the most significant ones will make the model easier to interpret, reduce feature redundancy, and shorten the time to train the model. There are numerous methods of feature selection and it also depends on the type of model to pick the appropriate method.

I used Pearson's correlation coefficient to remove redundant features and to select the best features that were correlated with the target variable. Pearson's coefficient values vary between -1 and 1, with 1 being highly positive correlated, and -1 being highly negative correlated. Since I am interested in highly correlated features with the target variable, I took the absolute value of Pearson's coefficient values so that the numbers can range from 0 to 1 making it easier to select. After using Pearson's correlation coefficient and some subjectivity on common basketball knowledge, I narrowed down the features to 7, which were: *conference standings*, *win percentage*, *total losses*, *defense rating*, *offense rating*, *average blocks*, and *average assists*. Below are the descriptions of the 7 features:

- Conferences Standings x_1: The rank on the performance on how well a team performed during the regular season with 1 being the highest rank and 8 being the lowest.(discrete),
- Win Percentage x_2: The percentage of games played that a team has won(continuous),
- Total Losses x_3: The number of games lost by a team(discrete),
- Defense Rating x_4: The number of points allowed per 100 possessions by a team(continuous),
- Offense Rating x_5: Measures a team's points scored per 100 possessions(continuous),
- Average Blocks x_6: blocks occur when an offensive player attempts a shot, and the defense player tips the ball, blocking their chance(continuous),
- Average Assists x_7: passes that lead directly to a made basket by a team (continuous),

Figure 23: *Feature Selection: Pearson's Correlation Coefficient*



5 Evaluation

So far the k-nearest neighbors model is fully trained with 7 features but the final step is to find out how well the model will perform on new data. Previously, I split the entire data where 80% is used only in the training set, and 20% is only used for the testing set. The goal is to make sure the model performs well on the training set, but also well on the testing set without data leakage, overfitting, and underfitting.

5.1 Metrics Glossary

Before discussing the evaluation of the model, here is a vocabulary that needs to be defined:

- True Negatives: The number of negative samples correctly classified.
- False Positives: The of negative samples incorrectly classified as positive.
- False Negatives: The number of positive samples incorrectly classified as negative.
- True positives: The number of positive samples correctly classified.
- Accuracy: The percentage of predictions that the model got right.

$$\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

,

- Precision: The percentage of predictions that were actually positive of the predictions that were classified as positive.

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

- Recall: The percentage of predictions that were classified positive of the predictions that were actually positive.

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

,

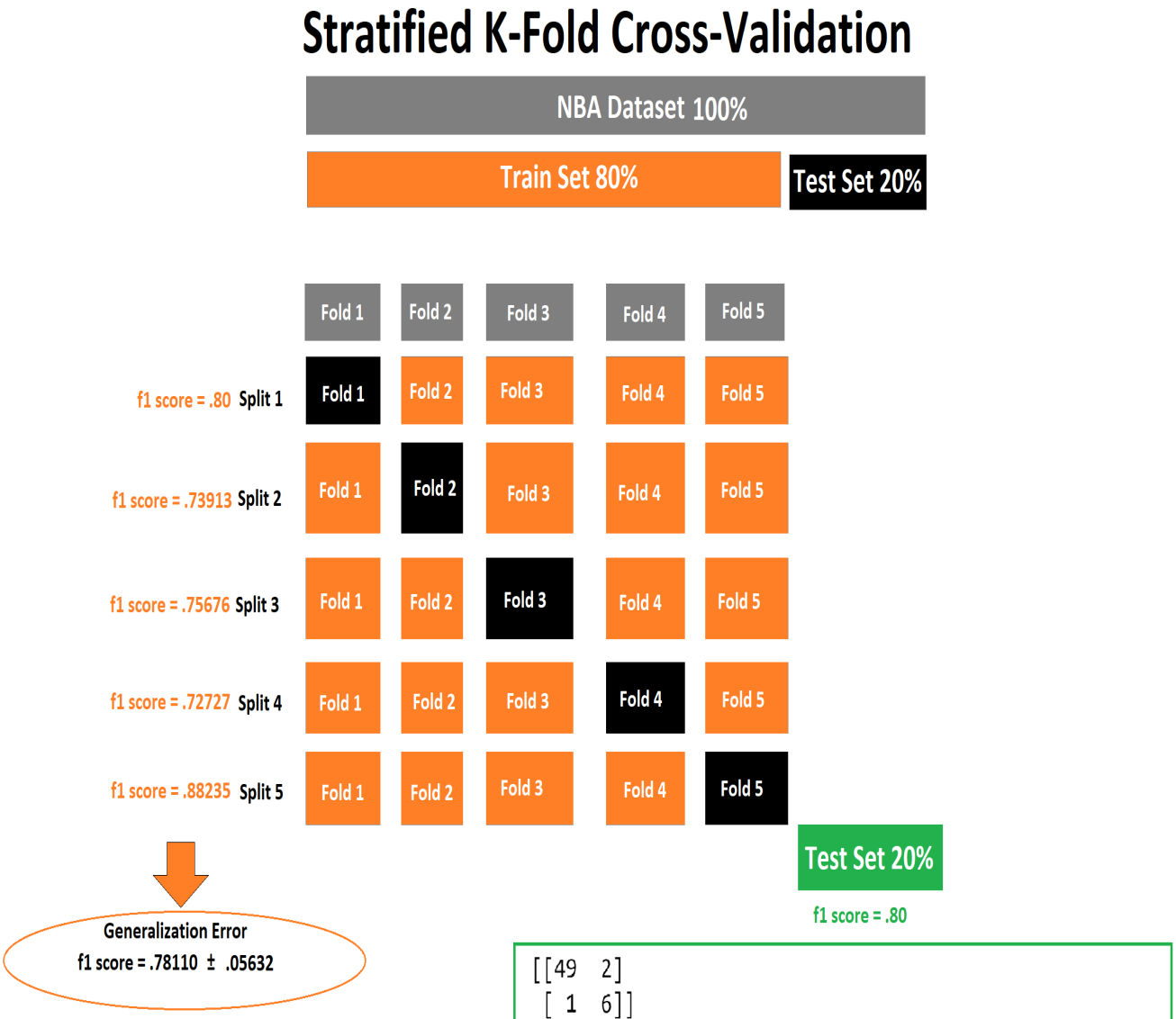
- F1 score: The f1 score is a weighted harmonic mean of precision and recall.

$$\text{F1 score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

5.2 Stratified K-Fold Cross-Validation

There are different methods to evaluate a model's performance but I used stratified k-fold cross-validation. In general, cross-validations estimate the generalization error for a given model. K-fold cross-validation is when the data is randomly split into independent k-folds without replacement. K-1 folds are used for the model training and one fold is used for performance evaluation. This process is repeated k times, so that we obtain the k number of the performance metrics picked for each iteration. Lastly, we get the mean and standard deviation of k number of that performance metric; this will be the generalization error for the model. K is usually 5 or 10 but since the data is small, I used 5. The performance metric I used is f1 score. Stratified k-fold cross-validation is similar to k-fold cross-validation but ensures that each fold has a similar proportion of observations of each class. This validation is useful when dealing with very imbalanced datasets. In the diagram, you can see a visual representation on how stratified k-fold cross-validation works.

Figure 24: *Stratified K-Fold Cross-Validation*




```
[[49  2]
 [ 1  6]]
              precision    recall  f1-score   support

           0       0.98      0.96      0.97        51
           1       0.75      0.86      0.80         7

    accuracy                           0.95        58
   macro avg       0.86      0.91      0.89        58
weighted avg       0.95      0.95      0.95        58
```

Figure 25: *Evaluation Code*

```
#########################################################################################################################
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=5,metric='euclidean')
knn_model.fit(scaled_X_train_os,y_train_os)
#########################################################################################################################

# evalute dataset
#########################################################################################################################
y_pred = knn_model.predict(scaled_X_test)

from sklearn.metrics import confusion_matrix,accuracy_score,classification_report

print(confusion_matrix(y_test,y_pred)) # check the model's confusion matrix
print(classification_report(y_test,y_pred)) # model classication report
#########################################################################################################################




# Stratified K-Fold Cross-Validation

from sklearn.model_selection import StratifiedKFold,cross_val_score

skfold=StratifiedKFold(n_splits=5,shuffle=True,random_state=65421)
# skfold=StratifiedKFold(n_splits=5)
# shuffle=True...will shuffle data before valdiation

scores=cross_val_score(knn_model,scaled_X_train_os,y_train_os,cv=skfold,scoring='f1')
print(scores)
print(np.mean(scores))
print(np.std(scores))
```

You can see that the generalization error has a f1 score mean of 0.78110 with a standard deviation of 0.05632. This means that the model is expected to perform within 0.78110 $\pm$ 0.05632. When I used the model on the testing set, the model had a f1 score of .80, which is within the generalization error that I got from the training set. This shows strong evidence that the model does not have overfitting or underfitting according to the results. In the testing set, there were a total of 7 teams that actually won the NBA playoffs and 51 teams that did not. The model was able to predict 6 teams winning the NBA playoffs correctly, but it misclassified 2 teams in winning the NBA playoffs and 1 team in not winning the playoffs. The model had an accuracy of .95, a precision score of .75, and a recall score of .86 but the f1 score is a better metric for imbalance data because it takes consideration of

the recall and precision.

6 Conclusion

After the exploratory data analysis, it was clear that the dataset needed to be fixed before training the knn model. These problems were solved in the feature engineering which included handling missing data, categorical features, and mixed features. Next, the model was trained by using 80% of the data and using the remaining 20% to test the model's performance. After splitting the data, we saw that the target variable classes were imbalanced and used an oversampling technique to fix this. Subsequently, from the 45 features, only 7 were selected by using Pearson's correlation coefficient to help remove unimportant and redundant features. Finally, the model was evaluated by using stratified k-fold cross-validation to get the generalization error and when using the testing data on the model, the performance fell within the error which shows strong evidence that the model does not overfit or underfit. For a person who is interested in sports betting for NBA games, this model may increase a person's chances of winning and should be used with caution.

7 Weaknesses

One weakness is the small dataset. In the feature engineering, 250 observations were removed due to teams not qualified in playing in the playoffs. The consequence of having a small dataset can lead to outliers, metric errors, and sampling bias. For the feature selection, there were limited methods to used . Unlike linear regression where certain assumptions need to be met, k-nearest neighbor does not have any strict requirements that could be helpful in eliminating irrelevant features. Another weakness is the data being imbalanced. For a dataset to be balanced, the class should approximately follow a 60/40 or 50/50 distribution. I could had added more synthetic samples but I decided not to because the results were good. Lastly, for the k-nearest neighbor model, there was no hyperparameter tuning like choosing the optimal k or changing the distance metric to increase the model's performance.