## Final Project Submission

- Student name: Noah-John Hizon
- Student pace: part time
- Scheduled project review date/time: 5/11 2:30 PM
- Instructor name: Abhineet Kulkarni
- Blog post URL:

# Predicting the Outcomes of NBA Games

## Overview

The focus of this project is to provide insight for NBA analysts towards understanding the most critical aspects that contribute to an NBA game's resulting final score. This project will be able to accurately predict how an NBA game will turn out given a set list of team attributes. Data regarding NBA games played in the 2019-2022 span (3 seasons) will be utilized to compose a final classifier model. The results of this modeling process will change the perspective on how analysts look at resulting game outcomes, and the factors that most affect them.

## Business Problem

The National Basketball Association (NBA) has changed to a great degree, especially in the past 3 years. While this is in large part due to the effects of the COVID-19 pandemic, the player base that has grown to prominence undoubtedly has the greatest effect. Players like Nikola Jokic, Stephen Curry, Kevin Durant, and Lebron James have changed the game to a great degree. With each passing year in the NBA, they set a new standard for future generations to live up to/surpass. The quality of an NBA player/team is not only defined by how well they shoot the ball or block shots; there is a myriad of other atrributes that emphasize overall contribution. Even from a holistic perspective, new coaches bring in new strategies into the mix that prioritize around certain players, methods of scoring, or managing game pace.

In an ever-fluctuating game state, NBA team general managers (or GMs) may have a hard time figuring out how to create a winning NBA franchise from the ground up. Key players departing due to either retirement, trade demands, or career-ending injuries, can leave a team in a state of disarray. Investments to acquire players or optimal NBA draft slots may prove to be unprofitable in the selected players' resulting performances. Lack of sound decision making could cost GMs their overall credibility, which may lead to ridicule or job loss, in extreme cases.

Data from the NBA 2019-2022 seasons (3 total seasons) will be aggregated/summarized in order to determine the most impactful characteristics that contribute to NBA game wins, and develop a machine learning model that can accurately predict the outcome of any game. Understanding the effect of every possible contributing game stat/contributor would help general managers make more informed predictions on what players to prioritizez. The output of this project would help build a GM's reputation while also educating the fanbase on the numbers that truly matter.

# Data Understanding

As seen below, the imported NBA data contains a variety of information related to the defining metrics for each game played in the 3 individual seasons (2019-2020, 2020-2021, 2021-2022). The data was originally exported on the player level of granularity, but was aggregated/cleaned via Excel, in order to have each row represented as an individual game. Players were aggregated and included in the overall dataset if their minutes played value was not equal to 0.

In [148]:
```
plot as plt
s
model import LinearRegression
election import GridSearchCV, train_test_split, cross_validate, cross_val_score,
pi as sm
_selection import RFECV, RFE
essing import StandardScaler
import mean_squared_error, make_scorer
rmula.api as smf
s stats
tats.api as sms
t combinations
port DecisionTreeClassifier
import accuracy_score, roc_curve, auc, confusion_matrix, plot_confusion_matrix,
e import BaggingClassifier, RandomForestClassifier
model import LogisticRegression
XGBClassifier

v('20192022NBAGAMES3.csv',header=0)
```

In [149]:
```
bothteams=nbadata[['CLEANED NAME','GAME RESULT','POINT DIFFERENCE','PACE']]
bothteams.head()
```

Out[149]:

| | CLEANED NAME | GAME RESULT | POINT DIFFERENCE | PACE |
|---|---|---|---|---|
| **0** | 20191022-H-LAC-A-LAL | 1 | 10 | 95.3 |
| **1** | 20191022-H-TOR-A-NOP | 1 | 8 | 102.2 |
| **2** | 20191023-H-BRK-A-MIN | 0 | 1 | 105.2 |
| **3** | 20191023-H-CHO-A-CHI | 1 | 1 | 101.6 |
| **4** | 20191023-H-DAL-A-WAS | 1 | 8 | 103.7 |

The data is comprised of 32 individual fields. There are 4 columns which represent cumulative game metrics across both teams, **CLEANED NAME, GAME RESULTS, POINT DIFFERENCE, and PACE.**

The **CLEANED NAME** is the unique identifier across the data set. Within the nomenclature, the field contains the date the game was played, the name of the home team, and the name of the away team. This field will not be relevant for the model build, but it will be useful in the future steps of the analysis to understand which teams won which game.

The **GAME RESULT** field will be the classifier that the model will eventually try to predict. This is a binary outcome classifier, represented as 1if the home team wins, and 0 if the away team wins.

The **POINT DIFFERENCE** is derived from taking the absolute value of the difference between the Home and Away team's final scores. For example, if the Home team scored 130 points, and the Away team scored 110 points, the point difference value would be 20. That being said, since the absolute value was taken within the calculation, every value in this column is an integer greater than zero.

The **PACE** of the game is represented by how many individual possessions take place across both teams. It is a metric that defines the speed at which the game is played, And is calculated by the formula below:

PACE= 48 * ((Home team Possessions + Away team Possessions) / (2 * (Minutes played / 5)))

48 is an arbitrary number, used to represent the total minutes in a standard 4 quarter NBA game.

In [150]:
```
1 nbadata.columns
```

Out[150]:
```
Index(['CLEANED NAME', 'GAME RESULT', 'POINT DIFFERENCE', 'PACE',
       'HOME TEAM OFFENSIVE RATING', 'HOME TEAM DEFENSIVE RATING',
       'HOME TEAM 2PT%', 'HOME TEAM 3PT%', 'HOME TEAM FT%',
       'HOME TEAM REBOUNDS', 'HOME TEAM ASSISTS', 'HOME TEAM STEALS',
       'HOME TEAM BLOCKS', 'HOME TEAM TURNOVERS',
       'HOME TEAM AVERAGE PLUS/MINUS', 'HOME TEAM DOUBLE DOUBLES',
       'HOME TEAM TOP 50 CONTRACT PLAYERS', 'AWAY TEAM OFFENSIVE RATING',
       'AWAY TEAM DEFENSIVE RATING', 'AWAY TEAM 2PT%', 'AWAY TEAM 3PT%',
       'AWAY TEAM FT%', 'AWAY TEAM REBOUNDS', 'AWAY TEAM ASSISTS',
       'AWAY TEAM STEALS', 'AWAY TEAM BLOCKS', 'AWAY TEAM TURNOVERS',
       'AWAY TEAM AVERAGE PLUS/MINUS', 'AWAY TEAM DOUBLE DOUBLES',
       'AWAY TEAM TOP 50 CONTRACT PLAYERS'],
      dtype='object')
```

The other 28 columns are split amongst the home and away teams, resulting in 14 fields represented for each team.

**OFFENSIVE RATING** represents each team's performance as a result of the following formula: Offensive Team Rating = (Points*Total FG%) + Point Difference= 1/5 of possessions - Times Fouled+ FTM* FT% * OAPOW (Official Adjusted Players Offensive Withstand).

**DEFENSIVE RATING** was represented in the initial data set on a player by player basis. The metric was averaged for any player that had minutes logged in the game, to get a cumulative team rating. The column was calculated by the following formula for each player: Defensive Player Rating = (Players Steals*Blocks) + Opponents Differential= 1/5 of possessions - Times blown by + Deflections * OAPDW( Official Adjusted Players Defensive Withstand

The respective field goal percentages, **2PT%**, **3PT%**, **FT%** are represented by the amount of made team field goals/field goal attempts for each shot type (2 point - shot within the 3 point line, 3 point - shot outside the 3 point line, free throw - at the line, after a shooting foul has occurred).

**OFFENSIVE REBOUNDS** represent rebounds made by the scoring team, keeping possession of the ball, while **DEFENSIVE REBOUNDS** represent those where the defending team takes possession after a scoring team's shot attempt.

An **ASSIST** represents the event where a player passes the ball to a teammate in a way that directly leads to a field goal score.

A **BLOCK** represents the event where a defending player legally deflects a field goal attempt by the scoring player.

A **STEAL** represents the event where a defensive player legally makes a turnover, causing a change in possession.

A **TURNOVER** represents a change in possession that occurs before a player on the offensive team attempts a shot.

**PLUS/MINUS** is an aggregated metric, averaged amongst the players in the original dataset with played minutes. This metric reflects how the team did while an individual player is on the court. For example, if a player has a +5, it means the team outscored the opposing team by 5 points while they were on the court.

A **DOUBLE-DOUBLE** represents the event where an individual player reaches double figures (10 or more) in 2 of the 5 main stat categories (points, rebounds, assists, steals, and blocks). A running count is aggregated for each home/away team.

The final metric, **TOP 50 CONTRACT PLAYERS**, is a derived metric based on ESPN’s player salary database. A list of the players with the top 50 salary contracts was compiled for the 3 seasons, then a distinct list was found through deduplication. When aggregating the games by player, a running count was summed for each home/away team.

In [151]:
```
1 nbadata.describe()
```

Out[151]:

| | GAME RESULT | POINT DIFFERENCE | PACE | HOME TEAM OFFENSIVE RATING | HOME TEAM DEFENSIVE RATING | HOME TEAM 2PT% | HC TEAM 3I |
|---|---|---|---|---|---|---|---|
| **count** | 3544.000000 | 3544.000000 | 3544.000000 | 3544.000000 | 3544.000000 | 3544.000000 | 3544.000 |
| **mean** | 0.545711 | 12.014391 | 98.438347 | 113.357619 | 111.954818 | 0.534781 | 0.36 |
| **std** | 0.497976 | 8.931961 | 4.889579 | 11.684333 | 11.731960 | 0.073262 | 0.084 |
| **min** | 0.000000 | 1.000000 | 82.200000 | 76.900000 | 65.777778 | 0.225806 | 0.088 |
| **25%** | 0.000000 | 5.000000 | 95.100000 | 105.200000 | 104.305769 | 0.486753 | 0.305 |
| **50%** | 1.000000 | 10.000000 | 98.300000 | 113.300000 | 112.133929 | 0.534483 | 0.360 |
| **75%** | 1.000000 | 16.000000 | 101.600000 | 121.400000 | 119.600000 | 0.584906 | 0.418 |
| **max** | 1.000000 | 73.000000 | 118.200000 | 159.300000 | 155.909091 | 0.782609 | 0.684 |

8 rows × 29 columns

In [152]:
```
nbadata=nbadata.drop(['CLEANED NAME'],axis=1)
nbadata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3544 entries, 0 to 3543
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   GAME RESULT                     3544 non-null   int64  
 1   POINT DIFFERENCE                3544 non-null   int64  
 2   PACE                            3544 non-null   float64
 3   HOME TEAM OFFENSIVE RATING      3544 non-null   float64
 4   HOME TEAM DEFENSIVE RATING      3544 non-null   float64
 5   HOME TEAM 2PT%                  3544 non-null   float64
 6   HOME TEAM 3PT%                  3544 non-null   float64
 7   HOME TEAM FT%                   3544 non-null   float64
 8   HOME TEAM REBOUNDS              3544 non-null   int64  
 9   HOME TEAM ASSISTS               3544 non-null   int64  
 10  HOME TEAM STEALS                3544 non-null   int64  
 11  HOME TEAM BLOCKS                3544 non-null   int64  
 12  HOME TEAM TURNOVERS             3544 non-null   int64  
 13  HOME TEAM AVERAGE PLUS/MINUS    3544 non-null   float64
 14  HOME TEAM DOUBLE DOUBLES        3544 non-null   int64  
 15  HOME TEAM TOP 50 CONTRACT PLAYERS  3544 non-null   int64  
 16  AWAY TEAM OFFENSIVE RATING      3544 non-null   float64
 17  AWAY TEAM DEFENSIVE RATING      3544 non-null   float64
 18  AWAY TEAM 2PT%                  3544 non-null   float64
 19  AWAY TEAM 3PT%                  3544 non-null   float64
 20  AWAY TEAM FT%                   3544 non-null   float64
 21  AWAY TEAM REBOUNDS              3544 non-null   int64  
 22  AWAY TEAM ASSISTS               3544 non-null   int64  
 23  AWAY TEAM STEALS                3544 non-null   int64  
 24  AWAY TEAM BLOCKS                3544 non-null   int64  
 25  AWAY TEAM TURNOVERS             3544 non-null   int64  
 26  AWAY TEAM AVERAGE PLUS/MINUS    3544 non-null   float64
 27  AWAY TEAM DOUBLE DOUBLES        3544 non-null   int64  
 28  AWAY TEAM TOP 50 CONTRACT PLAYERS  3544 non-null   int64  
dtypes: float64(13), int64(16)
memory usage: 803.1 KB
```

After removing the name identifier as our only categorical variable, the data is preprocessed and is ready to be modeled.

## Data Preparation

Due to all relevant fields already being in numerical form, no additional preprocessing is necessary before the modeling process is started. The initial, unprocessed dataset already had all of the information in numerical form to begin with, the aggregation was done in Excel to avoid redundancy/keep simplicity. If the data were to be preprocessed in Python, the groupby function would be used to summarize each numerical column, either summing or averaging relevant data certain columns, then grouping by the **CLEANED NAME** field as an identifier, while dropping the

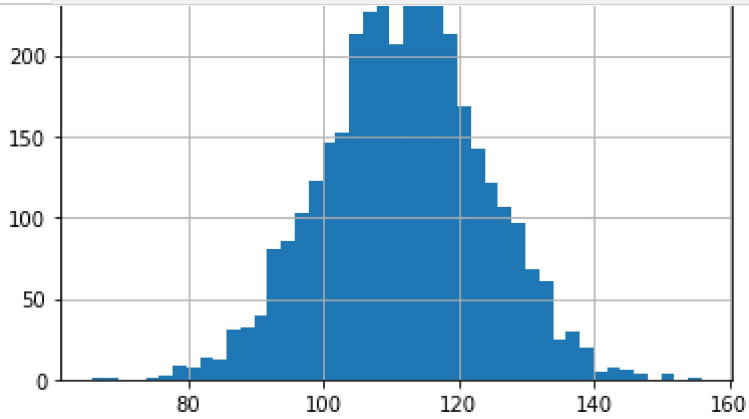
player name field. Players that had a **MINUTES PLAYED** value were the only ones aggregated in the overall metrics. Players listed in a game with no minutes played would inherently be dropped, as their inclusion would throw off the average metrics.

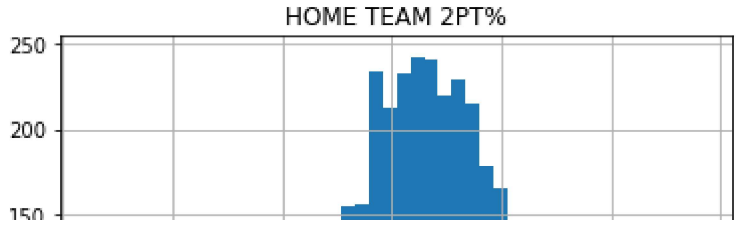
In [153]:
```
1 nbadata.dtypes
```

Out[153]:
```
GAME RESULT                          int64
POINT DIFFERENCE                     int64
PACE                               float64
HOME TEAM OFFENSIVE RATING         float64
HOME TEAM DEFENSIVE RATING         float64
HOME TEAM 2PT%                     float64
HOME TEAM 3PT%                     float64
HOME TEAM FT%                      float64
HOME TEAM REBOUNDS                   int64
HOME TEAM ASSISTS                    int64
HOME TEAM STEALS                     int64
HOME TEAM BLOCKS                     int64
HOME TEAM TURNOVERS                  int64
HOME TEAM AVERAGE PLUS/MINUS       float64
HOME TEAM DOUBLE DOUBLES             int64
HOME TEAM TOP 50 CONTRACT PLAYERS    int64
AWAY TEAM OFFENSIVE RATING         float64
AWAY TEAM DEFENSIVE RATING         float64
AWAY TEAM 2PT%                     float64
AWAY TEAM 3PT%                     float64
AWAY TEAM FT%                      float64
AWAY TEAM REBOUNDS                   int64
AWAY TEAM ASSISTS                    int64
AWAY TEAM STEALS                     int64
AWAY TEAM BLOCKS                     int64
AWAY TEAM TURNOVERS                  int64
AWAY TEAM AVERAGE PLUS/MINUS       float64
AWAY TEAM DOUBLE DOUBLES             int64
AWAY TEAM TOP 50 CONTRACT PLAYERS    int64
dtype: object
```

The data is now prepped for modeling. For this project, the plan is to use a classification model in order to assign a classifier to any given game, each game represented by a row in the dataset. The classification model will predict a binary outcome variable, represented by 0 or 1. 1 represents a win by the home team, and 0 represents a win by the away team.

In [154]:
```
for col in ['POINT DIFFERENCE', 'PACE',
       'HOME TEAM OFFENSIVE RATING', 'HOME TEAM DEFENSIVE RATING',
       'HOME TEAM 2PT%', 'HOME TEAM 3PT%', 'HOME TEAM FT%', 'HOME TEAM REBOU
       'HOME TEAM ASSISTS', 'HOME TEAM STEALS', 'HOME TEAM BLOCKS',
       'HOME TEAM TURNOVERS', 'HOME TEAM AVERAGE PLUS/MINUS',
       'HOME TEAM DOUBLE DOUBLES', 'HOME TEAM TOP 50 CONTRACT PLAYERS',
       'AWAY TEAM OFFENSIVE RATING', 'AWAY TEAM DEFENSIVE RATING',
       'AWAY TEAM 2PT%', 'AWAY TEAM 3PT%', 'AWAY TEAM FT%', 'AWAY TEAM REBOU
       'AWAY TEAM ASSISTS', 'AWAY TEAM STEALS', 'AWAY TEAM BLOCKS',
       'AWAY TEAM TURNOVERS', 'AWAY TEAM AVERAGE PLUS/MINUS',
       'AWAY TEAM DOUBLE DOUBLES', 'AWAY TEAM TOP 50 CONTRACT PLAYERS']:
    nbadata.hist(column=col, bins='auto')
```




Upon first glance, the majority of the variables seem well-distributed in their respective scales when plotted in histograms. This is promising as it means that there might not be a need to do extensive data manipulation in order to have a statistically valid resulting model

However, it should be noted that the scale of the variables is drastically different, with some being represented as solely positive integers, while some variables like average **PLUS/MINUS** may also contain negative values. That being said, a scaler will have to be applied in our initial regression model in order to improve validity and performance of our model.

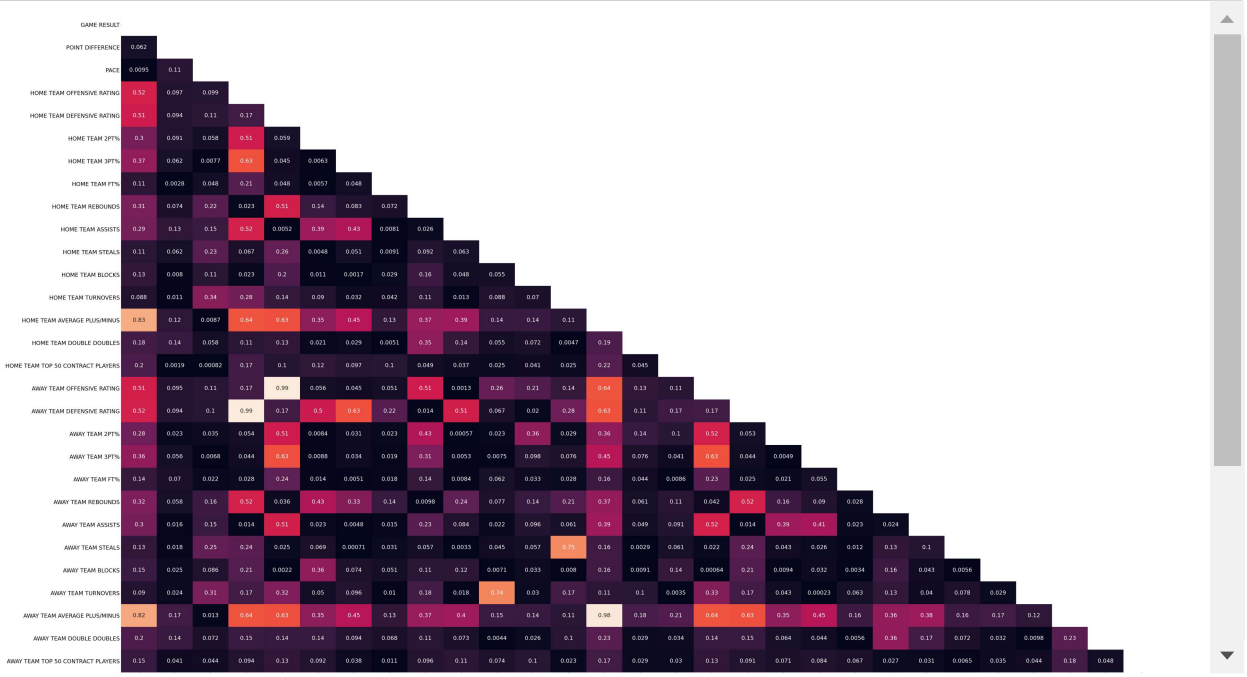
## Data Modeling

Now that the relevant columns are cleaned, modeling with a focus on 'GAME RESULT' as the target outcome variable can begin. A machine learning model is desired for this analysis, due to the necessity of a system that can predict wins given a set number of fields. The model's win predictions will be compared to the actual win results from the 2019-2022 seasons; an accurate model will reveal what the most impactful columns were on win rate.

First, a heat map is created to show the correlation between all of our variables.

In [155]:

```
correlation=abs(nbadata.corr())

fig, ax = plt.subplots(figsize=(100, 100))
heat=sns.heatmap(annot_kws={"fontsize":30},
    data=correlation,
    mask=np.triu(np.ones_like(correlation, dtype=bool)),
    ax=ax,
    annot=True,
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .2
)
ax.collections[0].colorbar.ax.tick_params(labelsize=50)
heat.set_xticklabels(heat.get_xmajorticklabels(), fontsize = 30)
heat.set_yticklabels(heat.get_ymajorticklabels(), fontsize = 30)
plt.show()
```




For the most part, it seems that most of our variables have little correlation with each other, seen by the abundance of combination correlations being below .4. There are some variable pairs that stand out as being highly correlated with each other. Due to the abundance of variables, the standout correlated pairs are difficult to pinpoint. The following code displays the top correlated variables within our dataset.

In [156]:
```
df=abs(nbadata.corr()>.75)
df=correlation.abs().stack().reset_index().sort_values(0,ascending=False)
df['paired']=list(zip(df.level_0,df.level_1))
df.set_index(['paired'],inplace=True)
df.drop(columns=['level_1','level_0'],inplace=True)
df.drop_duplicates(inplace=True)
df.head(20)
```

Out[156]:

| | 0 |
|---|---|
| **paired** | |
| **(GAME RESULT, GAME RESULT)** | 1.000000 |
| **(HOME TEAM DEFENSIVE RATING, AWAY TEAM OFFENSIVE RATING)** | 0.991417 |
| **(HOME TEAM OFFENSIVE RATING, AWAY TEAM DEFENSIVE RATING)** | 0.988798 |
| **(AWAY TEAM AVERAGE PLUS/MINUS, HOME TEAM AVERAGE PLUS/MINUS)** | 0.984488 |
| **(GAME RESULT, HOME TEAM AVERAGE PLUS/MINUS)** | 0.825814 |
| **(GAME RESULT, AWAY TEAM AVERAGE PLUS/MINUS)** | 0.818169 |
| **(HOME TEAM TURNOVERS, AWAY TEAM STEALS)** | 0.752596 |
| **(AWAY TEAM TURNOVERS, HOME TEAM STEALS)** | 0.743437 |
| **(HOME TEAM OFFENSIVE RATING, HOME TEAM AVERAGE PLUS/MINUS)** | 0.638910 |
| **(AWAY TEAM OFFENSIVE RATING, AWAY TEAM AVERAGE PLUS/MINUS)** | 0.636110 |
| **(HOME TEAM OFFENSIVE RATING, AWAY TEAM AVERAGE PLUS/MINUS)** | 0.636060 |
| **(HOME TEAM AVERAGE PLUS/MINUS, AWAY TEAM OFFENSIVE RATING)** | 0.635649 |
| **(HOME TEAM OFFENSIVE RATING, HOME TEAM 3PT%)** | 0.633629 |
| **(HOME TEAM AVERAGE PLUS/MINUS, AWAY TEAM DEFENSIVE RATING)** | 0.632439 |
| **(AWAY TEAM 3PT%, AWAY TEAM OFFENSIVE RATING)** | 0.631863 |
| **(AWAY TEAM AVERAGE PLUS/MINUS, AWAY TEAM DEFENSIVE RATING)** | 0.629339 |
| **(HOME TEAM DEFENSIVE RATING, AWAY TEAM AVERAGE PLUS/MINUS)** | 0.628658 |
| **(HOME TEAM DEFENSIVE RATING, AWAY TEAM 3PT%)** | 0.628533 |
| **(HOME TEAM AVERAGE PLUS/MINUS, HOME TEAM DEFENSIVE RATING)** | 0.628322 |
| **(HOME TEAM 3PT%, AWAY TEAM DEFENSIVE RATING)** | 0.625372 |

In order to avoid the issue of multicollinearity, aome of the attributes as defined above will need to be removed from the model. Multicollinearity is inherently a problem as performing a predictive model assumes that unit changes in each individual column has no effect on the others. That being said, a variable from each pairing with a correlation Factor greater than .75 will be removed from the model to mitigate this issue.

Since **OFFENSIVE RATING** and **DEFENSIVE RATING** are scored on a game by game basis, it is natural that a game with a high calculated Home team offensive rating would consequently have a low Away team defensive rating. The outcome variable is based on point metrics,**OFFENSIVE/DEFENSIVE RATINGS** will be removed for model validity.

Since **PLUS/MINUS** is an attribute calculated based on when either a home or away team player is on the floor, home and away team metrics will inherently have an inverse relationship. The more points an Away team member scores than a Home team member, the higher the away team's overall plus/minus would be and the lower the home team's overall plus/minus will be. Upon combing through the data, since the plus/minus metric is directly based on aggregate points scored, the comparison of home/away team plus/minus is associated with points scored by each team. The outcome variable is based on point metrics, we will remove the **PLUS/MINUS RATINGS** for model validity.

Similarly, **TURNOVERS** and **STEALS** are correlated, as a steal from a Home team player inherently counts as a legal turnover counted for the Away team.

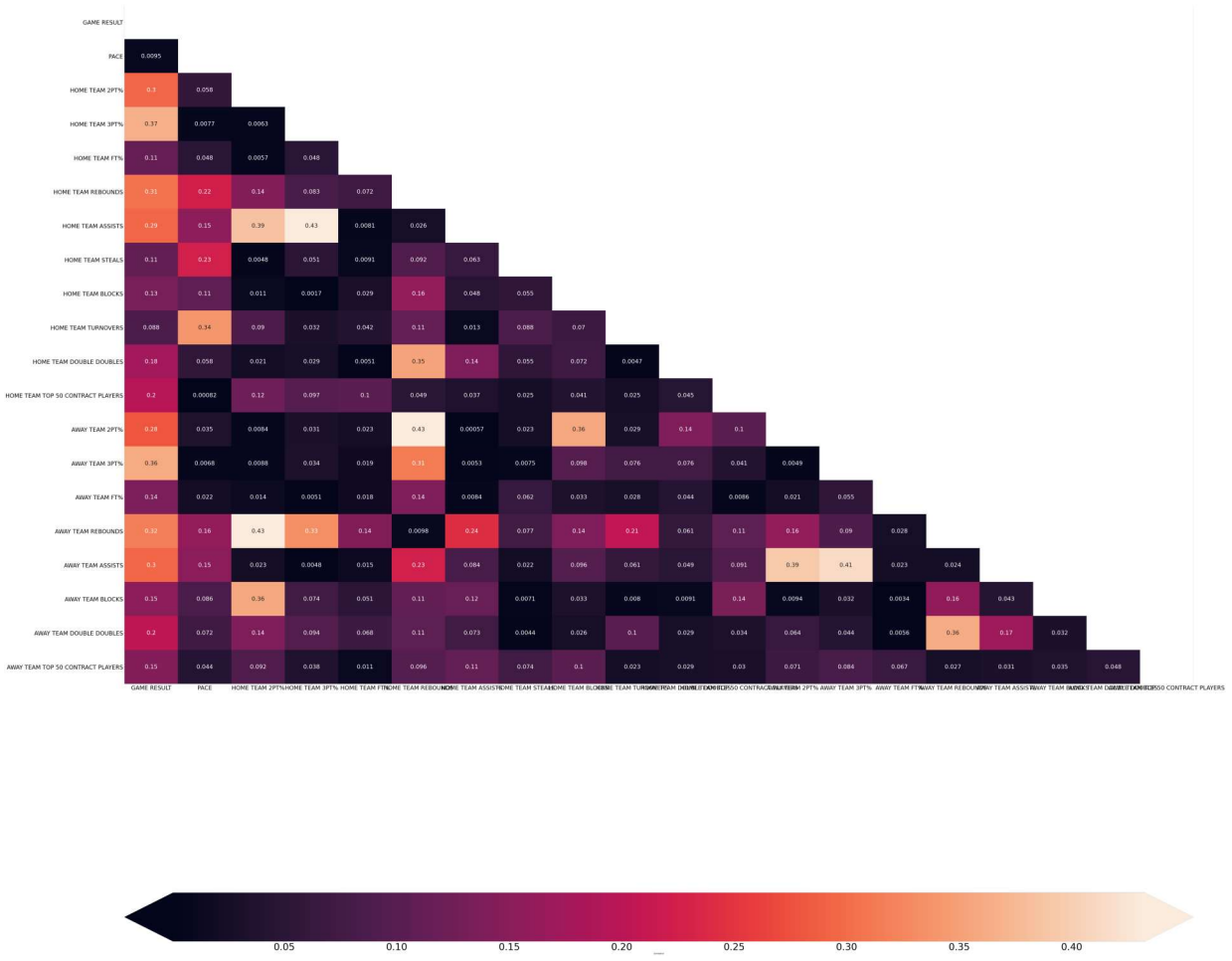
**POINT DIFFERENCE** was removed, as that metric is directly based on points, which is directly correlated to a win designation.

Below, the highly correlated variables in the dataset are dropped.

In [157]:
```
nbadata=nbadata.drop(['AWAY TEAM OFFENSIVE RATING','AWAY TEAM DEFENSIVE RATI
                      'AWAY TEAM AVERAGE PLUS/MINUS','AWAY TEAM STEALS',
                      'AWAY TEAM TURNOVERS','HOME TEAM AVERAGE PLUS/MINUS',
                      'HOME TEAM OFFENSIVE RATING','HOME TEAM DEFENSIVE RATI
                     'POINT DIFFERENCE'],axis=1)
```

In [158]:

```
corr2=abs(nbadata.corr())
fig, ax = plt.subplots(figsize=(100, 100))
heat2=sns.heatmap(annot_kws={"fontsize":30},
    data=corr2,
    mask=np.triu(np.ones_like(corr2, dtype=bool)),
    ax=ax,
    annot=True,
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .2
)
ax.collections[0].colorbar.ax.tick_params(labelsize=50)
heat2.set_xticklabels(heat2.get_xmajorticklabels(), fontsize = 30)
heat2.set_yticklabels(heat2.get_ymajorticklabels(), fontsize = 30)
plt.show()
```



After highly correlated variables are removed, the data can now be modeled.

Type *Markdown* and LaTeX: $\alpha^2$

## First Model

The first model will be created with simple classifier binary logistic regression, done to establish baseline performance of what a predictive model would look like. The following process depicts parsing out the data into training and test groups, done to perceive efficacy of a classifier across 2 different groups. The classifier will assign 1 as a prediction if the home team ends up winning, and 0 as a prediction if the away team ends up winning

In [159]:
```
nbadata.columns
```

Out[159]:
```
Index(['GAME RESULT', 'PACE', 'HOME TEAM 2PT%', 'HOME TEAM 3PT%',
       'HOME TEAM FT%', 'HOME TEAM REBOUNDS', 'HOME TEAM ASSISTS',
       'HOME TEAM STEALS', 'HOME TEAM BLOCKS', 'HOME TEAM TURNOVERS',
       'HOME TEAM DOUBLE DOUBLES', 'HOME TEAM TOP 50 CONTRACT PLAYERS',
       'AWAY TEAM 2PT%', 'AWAY TEAM 3PT%', 'AWAY TEAM FT%',
       'AWAY TEAM REBOUNDS', 'AWAY TEAM ASSISTS', 'AWAY TEAM BLOCKS',
       'AWAY TEAM DOUBLE DOUBLES', 'AWAY TEAM TOP 50 CONTRACT PLAYERS'],
      dtype='object')
```

In [160]:
```
y=nbadata['GAME RESULT']
X=nbadata.drop(columns=['GAME RESULT'], axis=1)

#normalizing data
```

The data is then normally scaled using the min/max of each column. This is done in order to optimize performance, since there is a high amount of variablility within the scale of our variables, as seen by the .describe() code.

In [161]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=0)

scaler=StandardScaler()
X_train1 =scaler.fit_transform(X_train)
X_test1 =scaler.transform(X_test)

logreg=LogisticRegression(fit_intercept=True, solver='liblinear')
logreg.fit(X_train1,y_train)
y_hat_train=logreg.predict(X_train1)
y_hat_test=logreg.predict(X_test1)
```

Above the LogisticRegression is instantiated. The regression model is used to predict using both the train and test data.

Below, results of the first model can be seen through the model's classification report. The classification report outputs 3 metrics of critical importance: precision, recall, accuracy, and f1 score.

These main metrics help understand how well the model performed in terms of classifying each game as a win/loss.

## Precision:

Precision measures how precise the predictions are. Precision is calculated by taking the number of true home team game wins, and dividing by the number of home team game wins predicted by the model.

## Recall:

Recall measures what % of the games identified as a win were actually identified correctly. Recall is calculated by dividing the number of total true home team game wins identified by the model, by the number of actual home team game wins.

## Accuracy:

Accuracy evaluates all predictions within the model. Accuracy is calculated by dividing the total correct predictions from the model (both wins/losses) by the total records in the original dataset.

## F1 Score:

F1 Score takes both precision and recall into account to get a cumulative score. The formula is calculated as follows: 2*(precision * recall)/(precision + recall). This metric is a good measure of overall model performance, as a high F1 score implies both precision and recall are both high as well.

In [162]:
```
reporttest1=classification_report(y_test, y_hat_test)
reporttrain1=classification_report(y_train, y_hat_train)

print('Test Data Classification Report')
print('_____________________________________')
print(reporttest1)
print('Training Data Classification Report')
print('_____________________________________')
print(reporttrain1)
```

```
Test Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.88      0.83      0.86       393
           1       0.87      0.91      0.89       493

    accuracy                           0.88       886
   macro avg       0.88      0.87      0.88       886
weighted avg       0.88      0.88      0.88       886

Training Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      1217
           1       0.91      0.92      0.92      1441

    accuracy                           0.91      2658
   macro avg       0.91      0.91      0.91      2658
weighted avg       0.91      0.91      0.91      2658
```

As seen above, the initial model performs relatively well on the test data, seen from the predicted values. F1 score is relatively high for both home team losses & wins classifications, implying succesful precision and recall values as well. Looking at the overall accuracy, across all observations, our model performed exceptionally well too.

The corresponding confusion matrices, seen below, re-interprets the information above in clean visuals, showing how the model classifies the test/train data using the logistic regression classifier trhough counts of true/false positives/negatives.

## LEGEND:

0 in True label, 0 in Predicted label - True Negative

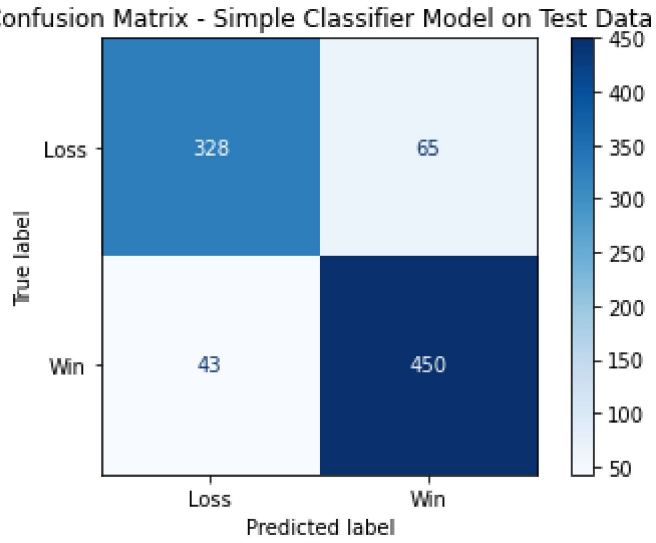
0 in True label, 1 in Predicted label - False Positive

1 in True label, 1 in Predicted label - True Positive

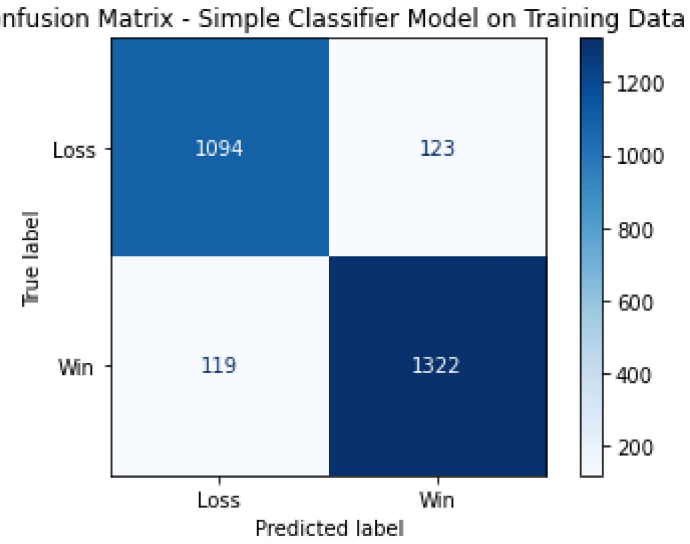
1 in True label, 0 in Predicted label - False Negative

In [163]:
```
ax=plt.subplot()
cnf_matrixtest1=plot_confusion_matrix(logreg, X_test1, y_test, display_label
ax.set_title('Confusion Matrix - Simple Classifier Model on Test Data')
```

Out[163]: Text(0.5, 1.0, 'Confusion Matrix - Simple Classifier Model on Test Data')




In [164]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - Simple Classifier Model on Training Data')
cnf_matrixtrain1=plot_confusion_matrix(logreg, X_train1, y_train, display_la
```




Seen from both plotted confusion matrices, the amount of accurately predicted counts is relatively proportional between the train and test data. The model appears to be slightly overfit on the training data, seen by the slightly higher accuracy % (91%). This implies that further application of parameters would help maintain overall model performance between train/test data while reducing overfitting.

## Second Model

A Decision Tree classifier is applied to the dataset, in order to get the perspective of a different model. A Decision Tree is comprised of different decisions that originate from the overall sample space. These decisions will be used to parse the data out, eventually getting to a granularity where the final nodes/groups of data can be classified. With metrics like 3PT% and rebounds being critical to success, decisions done based on these would be interesting to see. Below, the DecisionTreeClassifier is instantiated for use, and the game result is predicted based on the separated test data.

In [165]:
```
classifier=DecisionTreeClassifier(random_state=10)
classifier.fit(X_train,y_train)

y_hat_train2=classifier.predict(X_train)
y_hat_test2=classifier.predict(X_test)
```

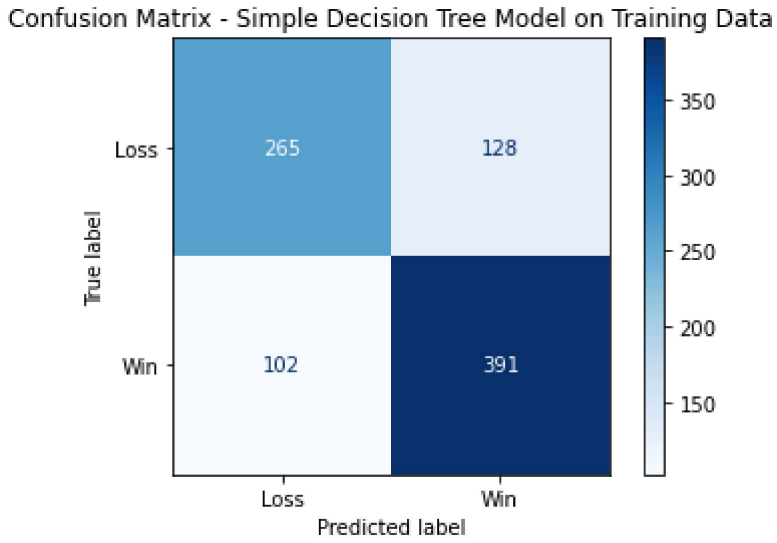
In [166]:
```
reporttest2=classification_report(y_test, y_hat_test2)
reporttrain2=classification_report(y_train, y_hat_train2)

print('Test Data Classification Report')
print('_____________________________________')
print(reporttest2)
print('Training Data Classification Report')
print('_____________________________________')
print(reporttrain2)
```

```
Test Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.72      0.67      0.70       393
           1       0.75      0.79      0.77       493

    accuracy                           0.74       886
   macro avg       0.74      0.73      0.74       886
weighted avg       0.74      0.74      0.74       886

Training Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1217
           1       1.00      1.00      1.00      1441

    accuracy                           1.00      2658
   macro avg       1.00      1.00      1.00      2658
weighted avg       1.00      1.00      1.00      2658
```

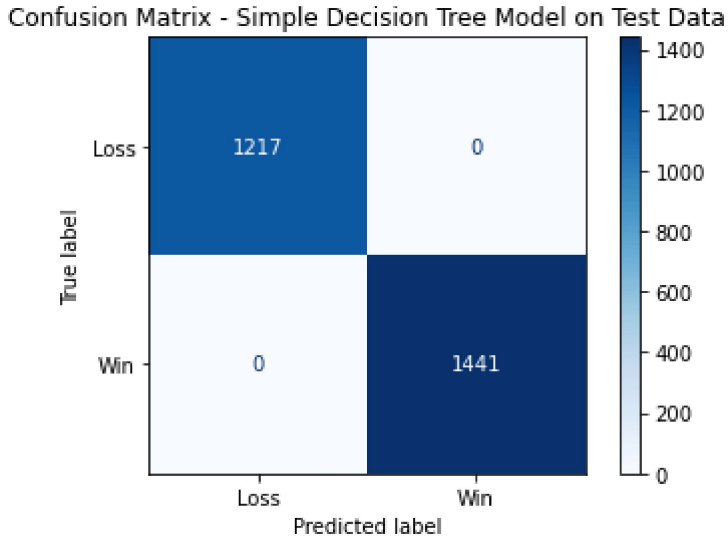
As seen above, the second model has dropped to ~.74 accuracy in predicting game outcomes. This may seem like a drastic change which invalidates the model, but as explained in later sections, an accuracy close to .75 is expected when predicting NBA games. The training data

scores clearly show the model is perfectly fit on the training data, with an accuracy score of 1. Parameter application in later model iterations will help reduce the overfitting.

In [167]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - Simple Decision Tree Model on Training Data
cnf_matrix_test2=plot_confusion_matrix(classifier, X_test, y_test, display_l

```




In [168]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - Simple Decision Tree Model on Test Data')
cnf_matrix_train2=plot_confusion_matrix(classifier, X_train, y_train,display_

```




The matrices show a slightly less accurate model overall on the test data compared to the first model, when using a Decision Tree classifier.

As seen by the above confusion matrix results, our prediction quality went down within the testing data. This is still a high performing model, but the fit will be optimized with the addition of hyperparameters. Due to the lack of applied hyperparameters, the Decision Tree fit on our training

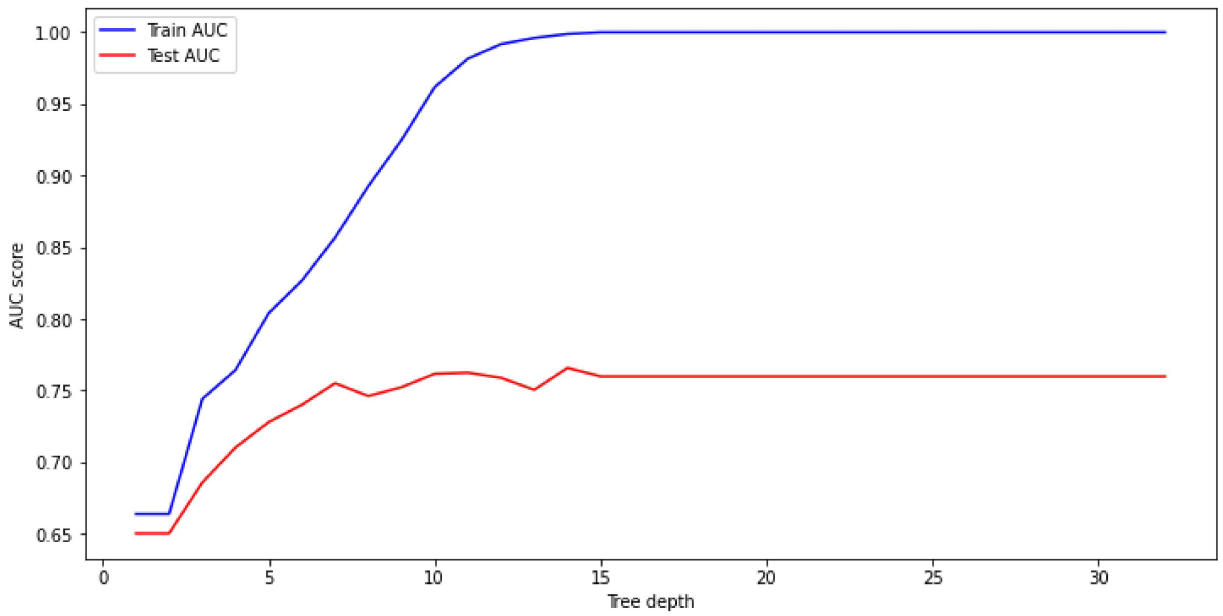
data had a perfect residual score of 1, quantifying it as overfit.

## Third Model

The application of hyperparameters will be observed in order to improve our initial Decision Tree model and mitigate overfitting on the training data. Below, the following charts are generated amongst the training/test data using the instantiated DecisionTreeClassifier in order to identify optimal hyperparameter values that maximize both training and testing AUC. The criterion 'entropy' is selected in order to prioritize minimizing messy data, while keeping predictive power.

In [169]:

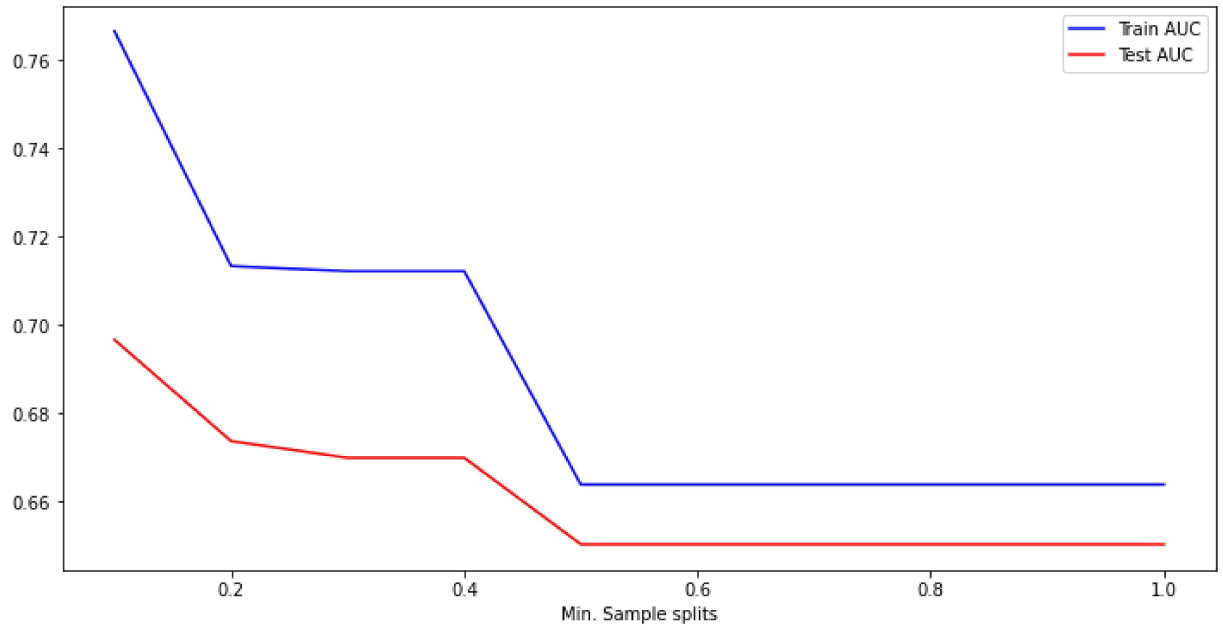
```
# Identify the optimal tree depth for given data
max_depths = np.linspace(1, 32, 32, endpoint=True)
train_results = []
test_results = []
SEED=22311
for max_depth in max_depths:
    dt = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth, ran
    dt.fit(X_train, y_train)
    train_pred = dt.predict(X_train)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,
    roc_auc = auc(false_positive_rate, true_positive_rate)


    train_results.append(roc_auc)


    y_pred = dt.predict(X_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y
    roc_auc = auc(false_positive_rate, true_positive_rate)


    test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_depths, train_results, 'b', label='Train AUC')
plt.plot(max_depths, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.legend()
plt.show()
```




As seen above, Train AUC maximizes around a tree depth of 15, with depth specified afterwards being a perfect fit on the training data. Test AUC peaks at ~5, but the corresponding Train AUC is only .75. There is a slight peak in Test AUC where tree depth is 10, so that value will be used as the max_depth hyperparameter.

In [170]:

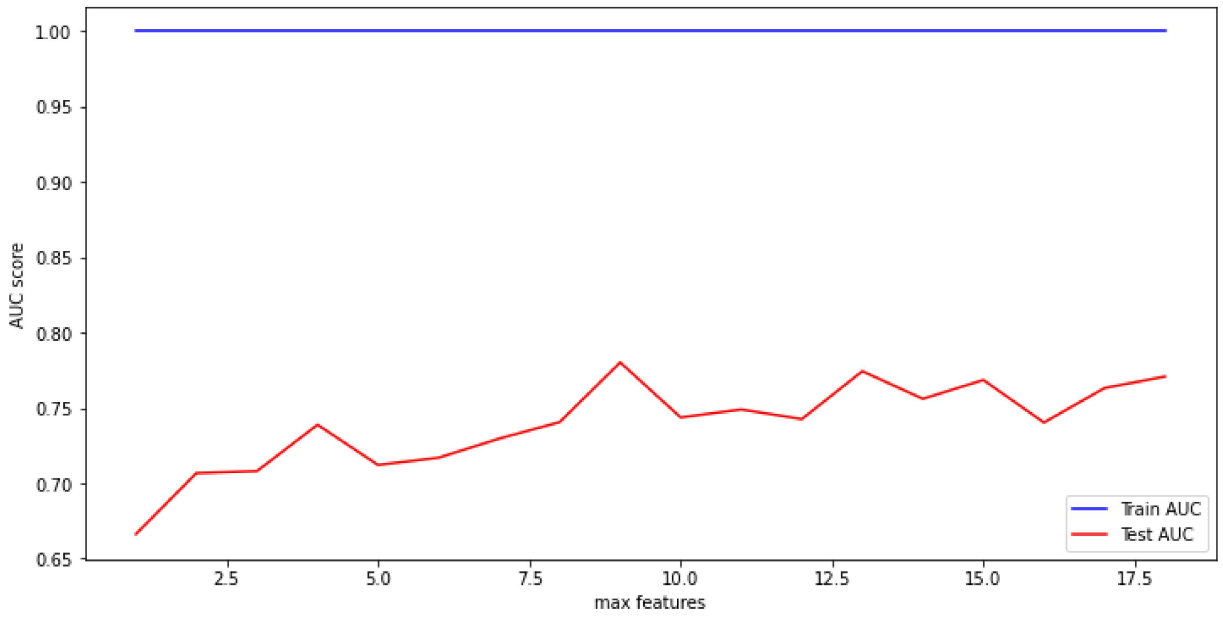
```
min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
train_results = []
test_results = []
for min_samples_split in min_samples_splits:
   dt = DecisionTreeClassifier(criterion='entropy', min_samples_split=min_sa
   dt.fit(X_train, y_train)
   train_pred = dt.predict(X_train)
   false_positive_rate, true_positive_rate, thresholds =    roc_curve(y_trai
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   y_pred = dt.predict(X_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(min_samples_splits, train_results, 'b', label='Train AUC')
plt.plot(min_samples_splits, test_results, 'r', label='Test AUC')
plt.xlabel('Min. Sample splits')
plt.legend()
plt.show()

```




As seen above, for both Train and Test data, a limit in min sample splits for both would severely decrease AUC for both sets. Therefore, this parameter will not be specified when instantiating the next DecisionTreeClassifier.

In [171]:

```
max_features=list(range(1, X_train.shape[1]))
train_results = []
test_results = []
for max_feature in max_features:
    dt = DecisionTreeClassifier(criterion='entropy', max_features=max_feature
    dt.fit(X_train, y_train)
    train_pred = dt.predict(X_train)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = dt.predict(X_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_features, train_results, 'b', label='Train AUC')
plt.plot(max_features, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('max features')
plt.legend()
plt.show()


print(test_results)
```



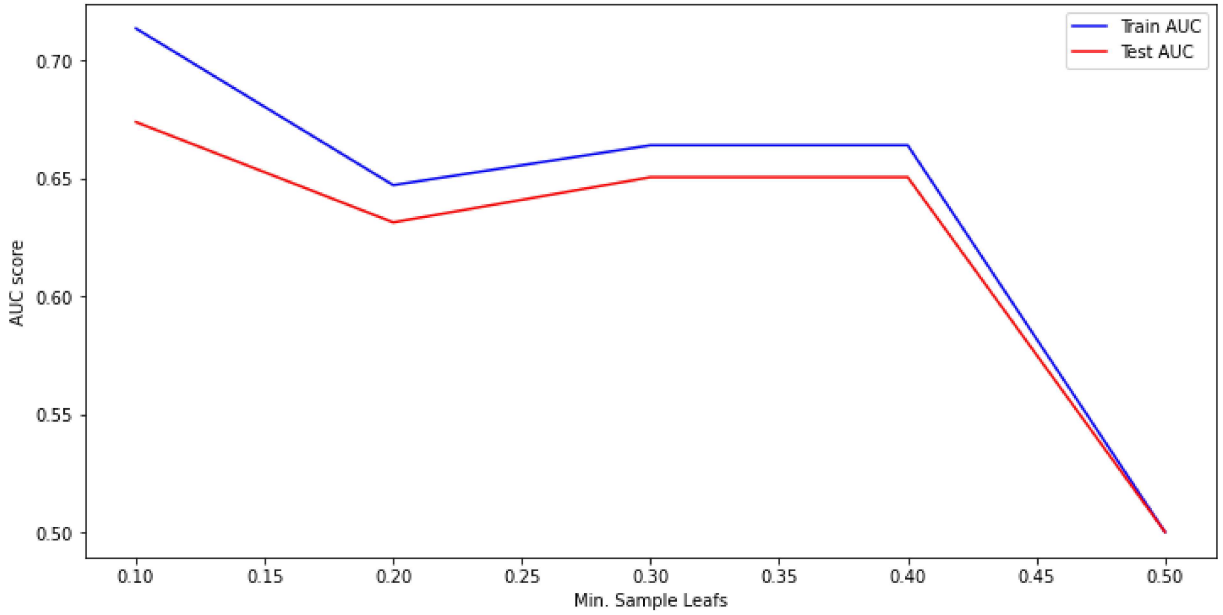

```
[0.6662485999927742, 0.7067804220925011, 0.7080707513329101, 0.73884510371666
43, 0.7121456110741216, 0.7169766037502129, 0.7296089270138169, 0.74059737082
```

```
51398, 0.7802233817980996, 0.7436941610021213, 0.7490051561556447, 0.74260770
37816968, 0.7743781903390469, 0.7560864830270092, 0.76855106348936, 0.7403212
403676922, 0.763203939117105, 0.7708194622939989]
```

For number of features, it seems like the Test AUC plateaus at ~.75, but there appears to be a bump in AUC at 9 features. Training AUC seems consistent no matter what value we choose, so 9 will be selected as the max_features hyperparameter.

In [172]:

```
min_samples_leafs=np.linspace(.1,.5,5,endpoint=True)
train_results=[]
test_results=[]
for min_samples_leaf in min_samples_leafs:
    dt=DecisionTreeClassifier(criterion='entropy', min_samples_leaf=min_samp
    dt.fit(X_train,y_train)
    train_pred=dt.predict(X_train)
    false_positive_rate, true_positive_rate, thresholds=roc_curve(y_train, t
    roc_auc=auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred=dt.predict(X_test)
    false_positive_rate, true_positive_rate, thresholds=roc_curve(y_test, y_
    roc_auc=auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)


plt.figure(figsize=(12,6))
plt.plot(min_samples_leafs, train_results, 'b', label='Train AUC')
plt.plot(min_samples_leafs, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('Min. Sample Leafs')
plt.legend()
plt.show()
```




Similar to the results seen in the plotted min_samples_split chart, a limit in min sample leafs for both would severely decrease AUC for both sets. In order to reduce overfitting, however, a limit of .1 will be applied in order to reduce the overfitting further.

In [173]:
```
dt2 = DecisionTreeClassifier(criterion='entropy', max_features=9, max_depth=
dt2.fit(X_train,y_train)
y_hat_test3=dt2.predict(X_test)
y_hat_train3=dt2.predict(X_train)
```

In [174]:
```
reporttest3=classification_report(y_test, y_hat_test3)
reporttrain3=classification_report(y_train, y_hat_train3)

print('Test Data Classification Report')
print('_____________________________________')
print(reporttest3)
print('Training Data Classification Report')
print('_____________________________________')
print(reporttrain3)
```

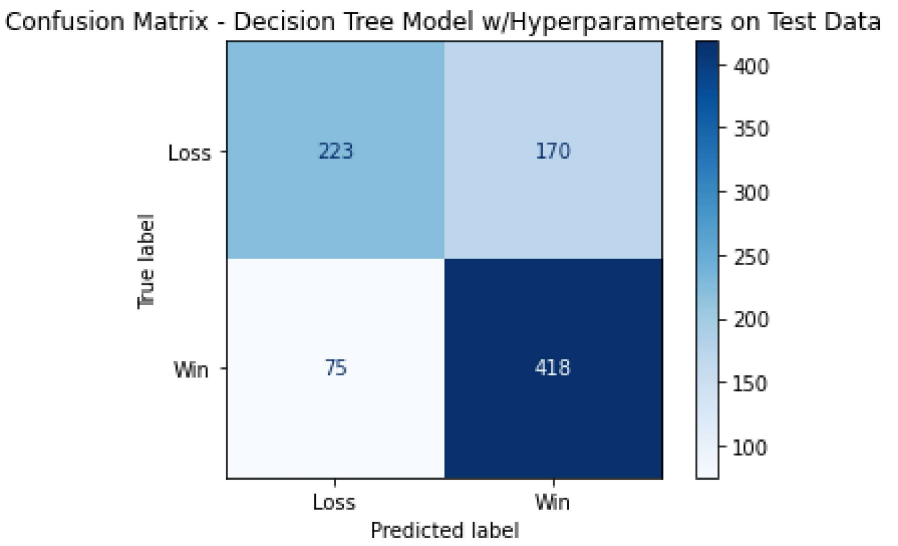
```
Test Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.75      0.57      0.65       393
           1       0.71      0.85      0.77       493

    accuracy                           0.72       886
   macro avg       0.73      0.71      0.71       886
weighted avg       0.73      0.72      0.72       886

Training Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.74      0.60      0.67      1217
           1       0.71      0.83      0.76      1441

    accuracy                           0.72      2658
   macro avg       0.73      0.71      0.71      2658
weighted avg       0.73      0.72      0.72      2658
```

As seen above, the application of all hyperparameters specified maintained the accuracy within the testing data relatively well, while also greatly reducing the overfitting seen within the training data.

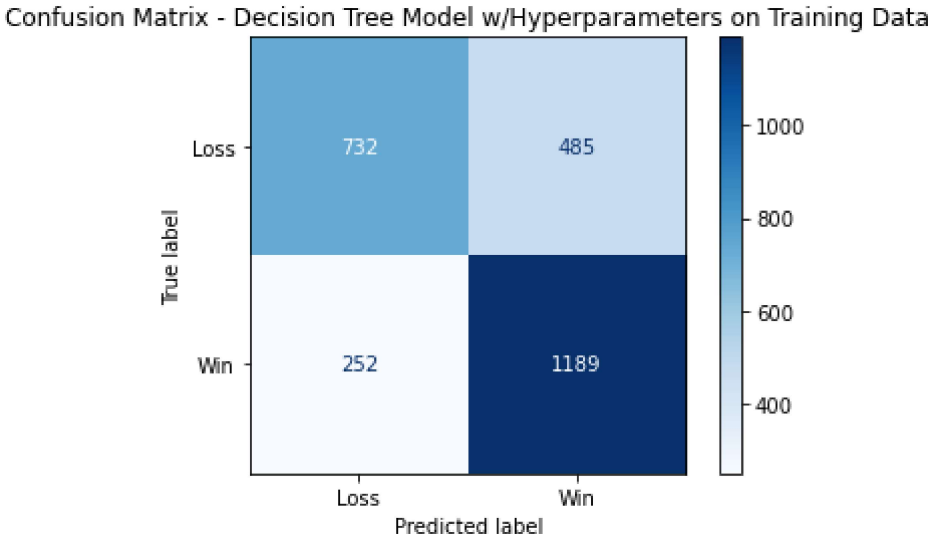
For the test data, recall was far higher than precision for win predictions, which implies the model was better at correctly identifying true game wins, rather than predicting whether a win happened. On the contrary, recall for loss predictions was far lower. This means the model tends to favor assigning false positive home team wins.

For the training data, similar trends can be seen within precision, recall, and f1-score for win/loss predictions, which makes the model performance relatively similar between train/test data. This means that the overfitting issue seen in the prior decision tree model has been fully mitigated.

In [175]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - Decision Tree Model w/Hyperparameters on Te
cnf_matrix_test3=plot_confusion_matrix(dt2, X_test, y_test, display_labels=[
```



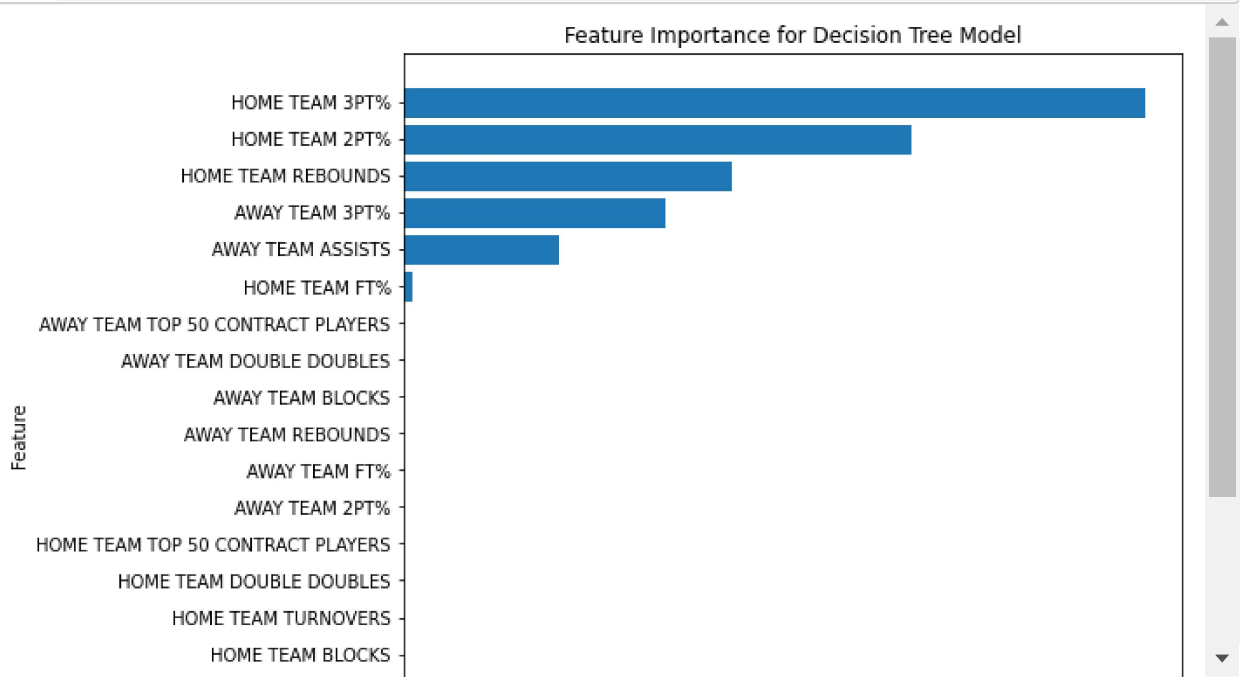

In [176]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - Decision Tree Model w/Hyperparameters on Tr
cnf_matrix_train3=plot_confusion_matrix(dt2, X_train, y_train, display_label
```




Resulting confusion matrices both illustrate the improvement in reduction of overfitting within the initial Decision Tree model when hyperparameters are added. The classifier now predicts with an accuracy of 72% with both the training data and test data.

Below, the defined **plot_feature_importances** function will allow us to see the individual contribution of each predictor in our model.

In [177]:
```
def plot_feature_importances(model,title):
    n_features = X.shape[1]
    plt.figure(figsize=(8,8))
    featdict={}
    for col,val in sorted(zip(X_train.columns, model.feature_importances_),k
        featdict[col]=val
    featuredf=pd.DataFrame({'Feature':featdict.keys(),'Importance':featdict.
    values=featuredf.Importance
    idx=featuredf.Feature
    plt.barh(idx, values, align='center')
    plt.xlabel('Feature Importance')
    plt.ylabel('Feature')
    plt.title('Feature Importance for {} Model'.format(title))

plot_feature_importances(dt2,'Decision Tree')
```




Seen above, the most impactful features in the model involve playmaking on the offensive end, and overall metrics that contribute to a team's final score. Since the 3 point shot offers the highest potential for scoring (4 points, if fouled on the made shot attempt) it is unsurprising that 3PT% is directly equated to win percentage. Similarly, 2 PT field goals (2PT%) are the standard/easiest route of scoring for most teams in the NBA. It seems that assists and rebounds are highlighted to a degree as well.

## Fourth Model

Next, the RandomForestClassifier will be tested for efficacy comparison amongst the other models. This model will be interesting to evaluate against, as it contains multiple Decision Trees in order to eliminate bias amongst individual trees. Each individual decision tree is based on a different subset of data so the resulting output will be a composite of several decisions. The nature of having

multiple trees assists with reducing overfitting within the eventual output. The plan is to use now use GridSearchCV to identify optimal hyperparameters within the forest, for a more optimized method in replicating the same overfitting-reduction results in the prior model.

In [178]:
```
depth = [i for i in range(6,11,1)]
features=[i for i in range(3,8,1)]

# Create the dictionary with parameters to be checked
parameters = dict(max_depth=depth, max_features=features, min_samples_leaf=[
forest2=RandomForestClassifier(random_state=11)

search=GridSearchCV(forest2,parameters,cv=3,return_train_score=True)
search.fit(X_train,y_train)

trainscore=np.mean(search.cv_results_['mean_train_score'])
testscore=search.score(X_test,y_test)
print(f"mean train score: {trainscore}")
print(f"mean test score: {testscore}")


search.best_params_

```

```
mean train score: 0.75945573112616
mean test score: 0.7708803611738149
```

Out[178]:
```
{'max_depth': 6,
 'max_features': 5,
 'min_samples_leaf': 0.1,
 'min_samples_split': 0.1}
```

As seen from the best_params_ output from the GridSearchCV, the ideal parameters are applied to the RandomForestClassifier below, seeing as they maximized both while reducing overfitting in the model.

{'max_depth': 10, 'max_features': 6, 'min_samples_leaf': 0.2, 'min_samples_split': 0.1}

In [179]:
```
forest=RandomForestClassifier(criterion='entropy', max_features=5, max_depth
forest.fit(X_train, y_train)
y_hat_test4=forest.predict(X_test)
y_hat_train4=forest.predict(X_train)

reporttest4=classification_report(y_test, y_hat_test4)
reporttrain4=classification_report(y_train, y_hat_train4)


print('Test Data Classification Report')
print('_____________________________________')
print(reporttest4)
print('Training Data Classification Report')
print('_____________________________________')
print(reporttrain4)

```

```
Test Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.81      0.64      0.71       393
           1       0.75      0.88      0.81       493

    accuracy                           0.77       886
   macro avg       0.78      0.76      0.76       886
weighted avg       0.78      0.77      0.77       886

Training Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.82      0.72      0.77      1217
           1       0.79      0.87      0.83      1441

    accuracy                           0.80      2658
   macro avg       0.81      0.80      0.80      2658
weighted avg       0.80      0.80      0.80      2658
```

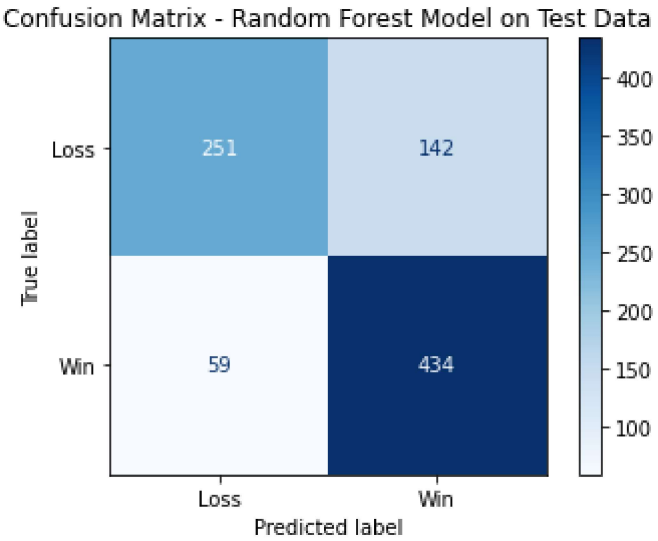
Compared to the prior Decision Tree model with applied hyperparameters, this is a clear improvement in performance. Recall in the loss predictions went up comparatively, making the overall accuracy of the test model ~80%, and the accuracy of the training model ~77%.

For the test data, recall was far higher than precision in the win predictions, which implies the model was better at correctly identifying true game wins, rather than predicting whether a win happened. The loss predictions show a drop in recall compared to the third model. This shows that our model is identifying a higher amount of false positive wins.

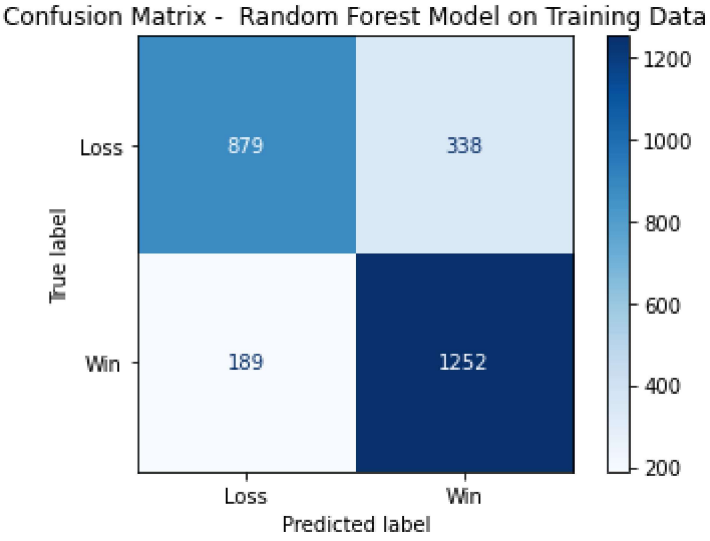
For the training data, a similar trend can be seen, but slightly higher precision in the loss predictions led to a higher overall f1 score & accuracy.

The ~3% discrepancy between train/test accuracy can be chalked up to noise/slight overfitting in the training data. This model still preserved accuracy seen in Model 3, while keeping loss recall relatively high.

In [180]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - Random Forest Model on Test Data')
cnf_matrix_test4=plot_confusion_matrix(forest, X_test, y_test, display_label
```



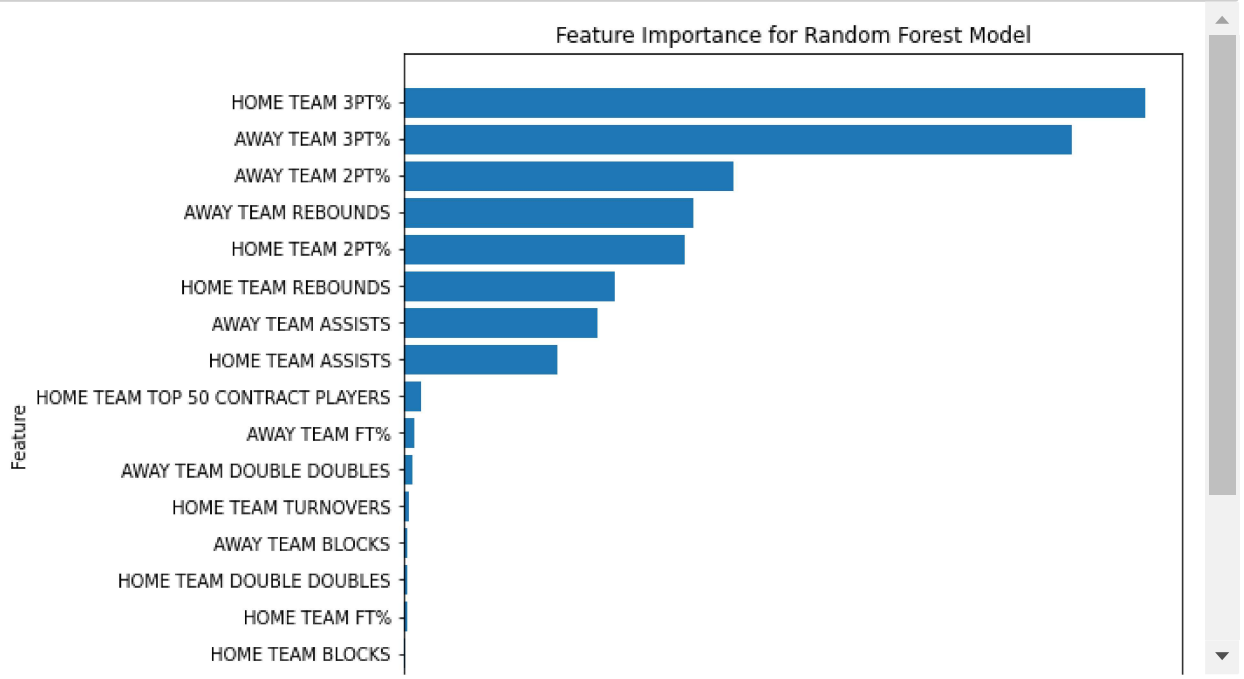

In [181]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix -  Random Forest Model on Training Data')
cnf_matrix_train4=plot_confusion_matrix(forest, X_train, y_train, display_la

```




Overall accuracy went up slightly for both sets. This may be due the inclusion of multiple trees having an influence in the decision making, rather than just one.

As shown by the confusion matrices, the model is now outputting a lower amount of false positive game wins, seen by the top right squares. This further validates the model, it being the most successful identifying true wins compared to prior models.

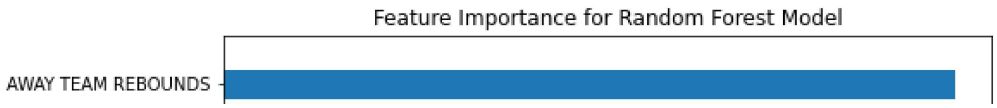
In [182]:
```
1 plot_feature_importances(forest, 'Random Forest')
```




Similarly to the initial Decision Tree model, the Random Forest model highlights the same features of importance within determining game wins: 3PT%, 2PT%, assists, and rebounds.

The **TOP 50 CONTRACT PLAYERS** being recognized as a feature of importance within the forest shows the effect a player with a max contract can have on game win %. Typically, NBA management will only give these kinds of contracts to players who they can build teams around. The max contract players will typically stay with the team for at least 2 years, and more often than not, are their main scorers.

In [183]:
```
rf_tree_1=forest.estimators_[0]
rf_tree_2=forest.estimators_[1]
rf_tree_3=forest.estimators_[2]
rf_tree_4=forest.estimators_[3]

plot_feature_importances(rf_tree_1, 'Random Forest')
plot_feature_importances(rf_tree_2, 'Random Forest')
plot_feature_importances(rf_tree_3, 'Random Forest')
plot_feature_importances(rf_tree_4, 'Random Forest')
```




When looking at the individual trees, we can see that each tree highlights a combination of important metrics in the overall forest contributing to team wins.

## Fifth Model

Finally, the XGBoost model will be used as a final comparison to our prior models. XGBoost is the top gradient boosting algorithm, used on trees in order to strengthen poor performing ones. A GridSearchCV parameter grid search will be used to tune the algorithm and identify ideal hyperparameters.

In [184]:

```
rate=[.4,.3,.5]
depth = [i for i in range(6,10,1)]

parameters2 = dict(learning_rate=rate, max_depth=depth,min_child_weight=[2,3
clf=XGBClassifier()

search2=GridSearchCV(clf,parameters2,cv=3,return_train_score=True)
search2.fit(X_train,y_train)

trainscore2=np.mean(search2.cv_results_['mean_train_score'])
testscore2=search2.score(X_test,y_test)
print(f"mean train score: {trainscore2}")
print(f"mean test score: {testscore2}")

search2.best_params_
```

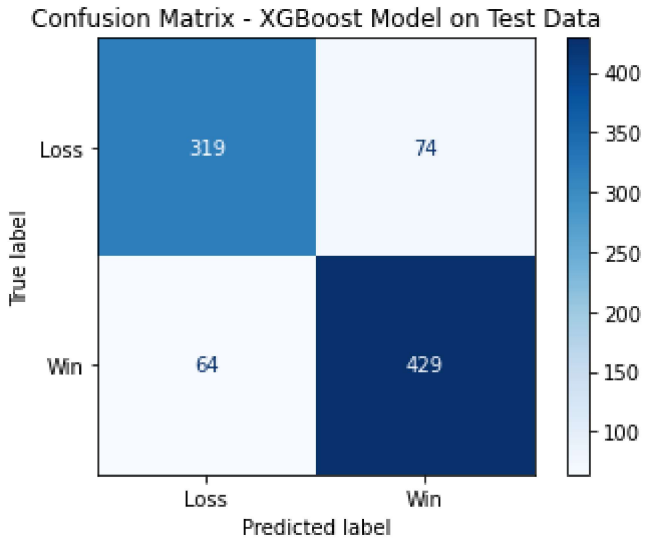
```
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
[21:27:21] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
[21:27:21] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
```

In [185]:
```
clf=XGBClassifier(max_depth=9,learning_rate=0.4,min_child_weight=3, subsampl
clf.fit(X_train,y_train)
y_hat_test5=clf.predict(X_test)
y_hat_train5=clf.predict(X_train)

reporttest5=classification_report(y_test, y_hat_test5)
print(reporttest5)

reporttrain5=classification_report(y_train, y_hat_train5)
print(reporttrain5)
```

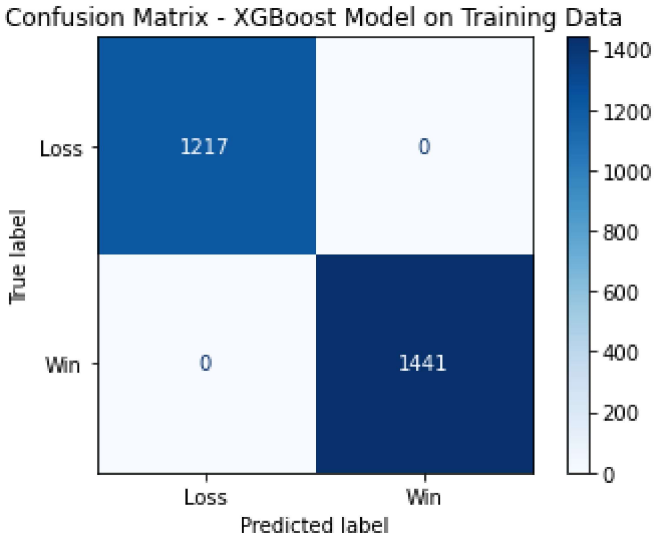
```
[21:27:49] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the defa
ult evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       393
           1       0.85      0.87      0.86       493

    accuracy                           0.84       886
   macro avg       0.84      0.84      0.84       886
weighted avg       0.84      0.84      0.84       886

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1217
           1       1.00      1.00      1.00      1441

    accuracy                           1.00      2658
   macro avg       1.00      1.00      1.00      2658
weighted avg       1.00      1.00      1.00      2658
```

While the test data shows the best modeled results thus far, the model is overfit on the training data, seen by the perfect accuracy score.

In [186]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - XGBoost Model on Test Data')
cnf_matrix_test5=plot_confusion_matrix(clf, X_test, y_test,display_labels=['
```

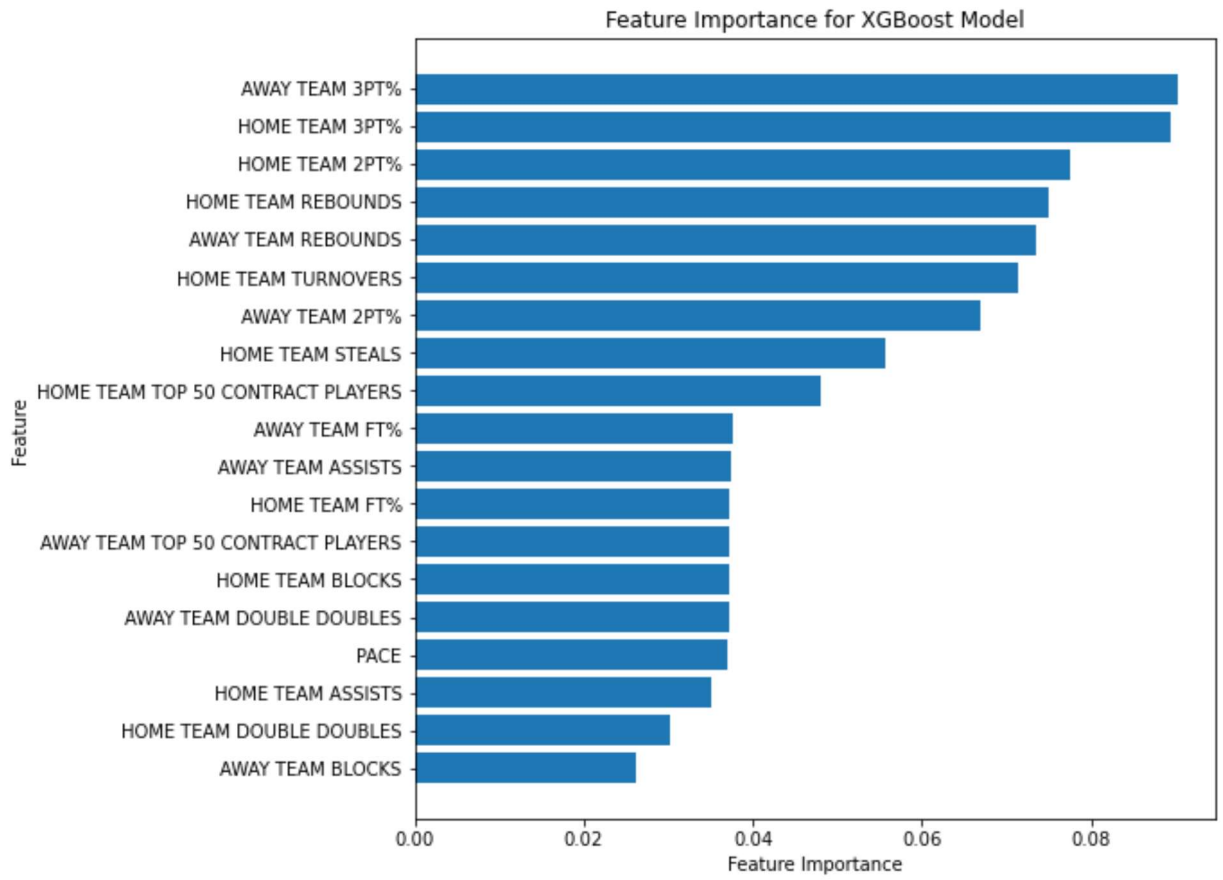



In [187]:
```
ax=plt.subplot()
ax.set_title('Confusion Matrix - XGBoost Model on Training Data')
cnf_matrix_test5=plot_confusion_matrix(clf, X_train, y_train,display_labels=
```




```
Confusion Matrices support the near perfect model fit, but the predictions
on the test data were the best overall, showing minimal false
positives/false negatives amongst the test data.
```

In [188]:
```
1 plot_feature_importances(clf, 'XGBoost')
```




Feature importance seems to take all possible metrics into account to a degree, but the prior metrics identified, 3PT%, 2PT%, and rebounds still appear to be the most prominent.

Seen from the above results, the model overfit on the training data, despite having a high accuracy when applying the model to the testing set. That being said, the Decision Tree (with identified hyperparameters) will be used as our final model, as it mitigated overfitting in the test/training data split while keeping overall accuracy high.

# Classifier Results Evaluation

There are several factors that go into determining a typical NBA game win. In order to find the ideal factors that are critical to the determination of game wins, the performed analysis/model creation using the 2019-2022 NBA dataset revealed the scale/importance of several of these key metrics. The final model created, a Random Forest Classifier with applied hyperparameters, was able to predict NBA game wins with an accuracy of 77% for test data and 80% for training data.

A classification report of the Random Forest classifier model can be seen below across the test/train datasets.

In [189]:
```
print('Test Data Classification Report')
print('_____________________________________')
print(reporttest4)
print('Training Data Classification Report')
print('_____________________________________')
print(reporttrain4)
```

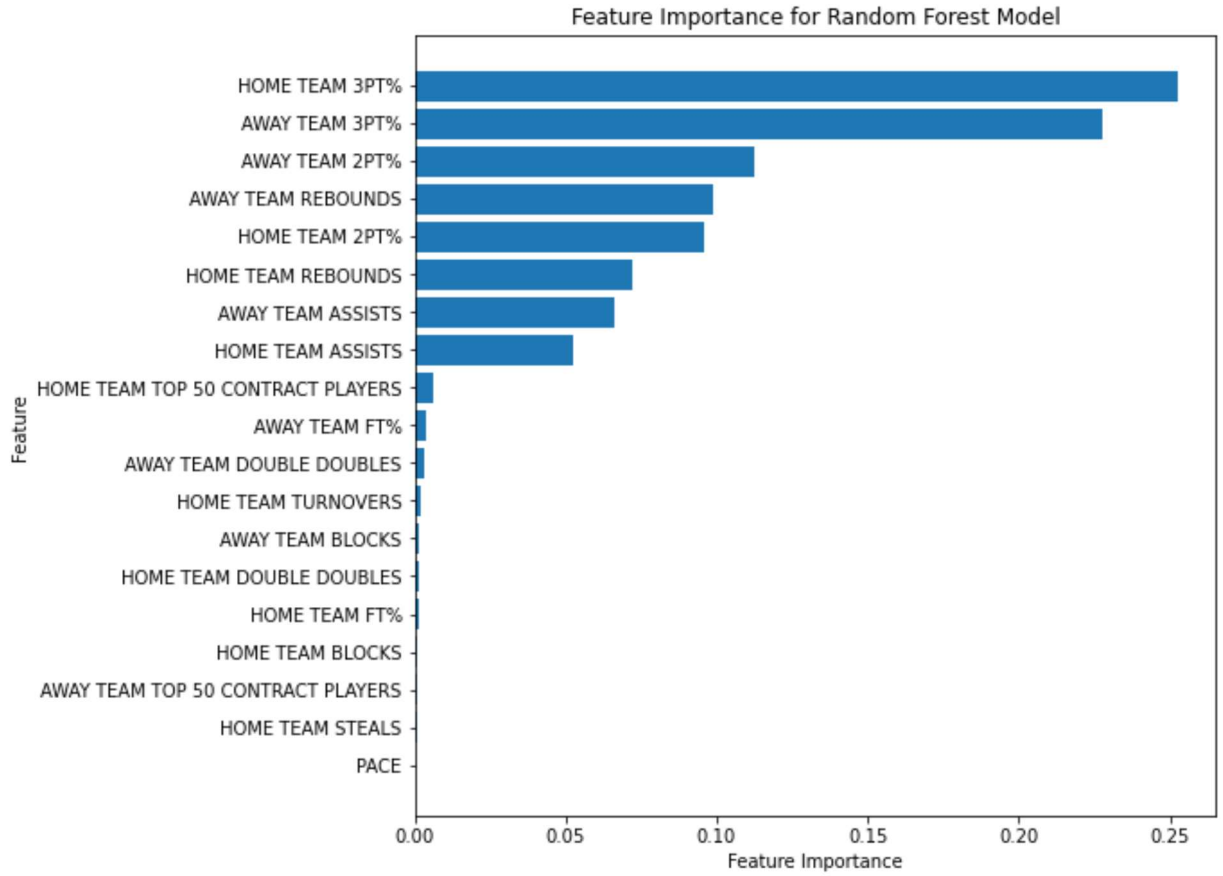
```
Test Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.81      0.64      0.71       393
           1       0.75      0.88      0.81       493

    accuracy                           0.77       886
   macro avg       0.78      0.76      0.76       886
weighted avg       0.78      0.77      0.77       886

Training Data Classification Report
_____________________________________
              precision    recall  f1-score   support

           0       0.82      0.72      0.77      1217
           1       0.79      0.87      0.83      1441

    accuracy                           0.80      2658
   macro avg       0.81      0.80      0.80      2658
weighted avg       0.80      0.80      0.80      2658
```

For the test data, recall was far higher than precision for win predictions, which implies the model was better at correctly identifying true game wins, rather than predicting whether a win happened. On the contrary, recall for loss predictions was far lower. This means the model tends to favor assigning false positive home team wins. The tendency to designate false positive wins can be overlooked since the recall % of overall home team wins was high, which is the main target metric of success within the model.

For the training data, similar trends can be seen within precision, recall, and f1-score for win/loss predictions, which makes the model performance relatively similar between train/test data. Both test/training predictions having ~80% accuracy implies that the overfitting issue seen in all prior models on the training data have been mostly mitigated. While an 80% accuracy across both test/training data may be seen as a poorly performing model, NBA games have a typical upset (where the non-favored team ends up winning) rate of 32%. This is understandable when looking at examples in recent years, such as the Toronto Raptors' 2019 NBA Finals win against the favored Golden State Warriors, and the Boston Celtics 4-0 playoff sweep against the favored Brooklyn Nets. That being said, the current accuracy rate with this model is to be expected.

Below, the calculated score importance of each feature in the dataset shows which features were the most impactful in making the model decisions.

In [190]:
```
1 plot_feature_importances(forest, 'Random Forest')
```




## 3PT and 2PT Percentages

Seen above, the most impactful features in the model have to do with playmaking on the offensive end. Since the 3 point shot offers the highest potential for scoring (4 points, if fouled on the made shot attempt) it is unsurprising that this percentage is most equated to win rate. Similarly, 2 PT field goals are the standard/easiest route of scoring for most teams in the NBA. Nearly all of NBA players have a higher average 2 PT % compared to their 3 PT %, so the importance is less impactful. It is far rarer to have a consistent and efficient 3 point shooter on most teams, so scoring ability in that regard is highly valued. Players that can take advantage of both field goal percentages inherently have more scoring potential, and are offensive assets. More often than not, these scorers typically have the highest value contracts, are all-star selections/candidates, and are starters for their team. Some of the most prolific scorers/shooters active in the NBA are Stephen Curry, Kevin Durant, Giannis Antentokounmpo, Joel Embiid, and Nikola Jokic. All of the named players have been on teams that have made deep NBA playoff runs in recent years.

Stephen Curry in particular is widely regarded as one of the most efficient 3PT shooters of all time, holding the current all-time NBA made 3PT shot record at 3,117. He has won NBA championships with the Warriors in 2015, 2017, and 2018.

Kevin Durant's play style, athleticism, and build have made him one of the most difficult NBA players to guard; his ability to create open shot opportunities from the 3PT and 2PT range is fundamental to his role as a scorer. He has played a pivotal role in the success of the Oklahoma City Thunder, Golden State Warriors, and Brooklyn Nets franchises.

Giannis Antentokounpo, Nikola Jokic, and Joel Embiid were all MVP (most valuable player) candidates for the 2021-2022 season; Jokic was awarded the title in 2021 and 2022, and Giannis was awarded the title in 2020 and 2019. The imposing builds of all 3 of these "big men" make them nearly impossible to guard when in the "paint"/driving to the rim.

## Rebounds

Total team rebounds are seen as an important factor as well. This makes sense, as the more that a defending team is able to regain possession during a game, the more opportunities that creates for that team to score on a fast break, before the other team has a chance to reset their defense. Similarly, the more that a team can rebound on the offensive end, the more they can prevent the other team from taking possession and scoring points. Current rebound leaders in the NBA include Rudy Gobert, Nikola Jokic, Giannis Antentokounmpo, and Joel Embiid.

Rudy Gobert has played a pivotal role for the Utah Jazz on the defensive end, having a career average of almost 12 rebounds per game. His high rebound/block rates have earned him the DPOY (Defensive Player of the Year) award 3 times in the past 4 years.

As previously mentioned, Giannis, Jokic, and Embiid were all MVP candidates for the 2021-2022 NBA season, with Jokic winning the title. It is natural that players who can be assets on both offensive and defensive ends of the court are highly valued.

## Assists

Assists are high in importance as well. Players that don't have as strong of an affinity for scoring can still be valued if they can create plays for their teammates through their passing ability/leadership. This leads to an increase in a team's resulting scores. Assists as a whole can be used to identify how well a team works together in order to score. The more assists a team has by the end of the game either highlights a team's consistent offense, or an opposing team's lackluster defense. Current assist leaders in the NBA include Chris Paul, Lebron James, and Luka Doncic.

Chris Paul has been in the top 20 season APG (assists per game) ranking list for the past 10 years. His ability to create shot opportunites for his teammates has led the Phoenix Suns to the NBA finals in 2021, and the playoffs in 2022.

Lebron James has been widely regarded as the GOAT (Greatest of all Time) by many NBA fans, and for good reason. His ability as a scorer/team leader has led him to 4 NBA championships with the Cleveland Cavaliers, the Miami Heat, and the Los Angeles Lakers.

## Recommendations/Conclusion

NBA General Managers(GMs) have the difficult job of determining ideal roster candidates in a game so highly sensitive as basketball. They are undoubtedly placed under a lot of scrutiny, as their job title inherently implies a strong understanding of the current game state. Several general managers, such as Elgin Baylor, have terrible draft picking making ability, resulting in very little success in the regular season and no appearances in the playoffs. Bad player investments can hinder a team's performance for years, or even decades. An GM’s teambuilding ability is directly correlated to their baseline understanding of NBA metrics and current/past NBA knowledge. Knowledge of game metrics and the effect they have on a win/loss result is critical to their success.

Knowledge of game metrics and the effect they have on a win/loss result is critical to their success. The created model to help analysts identify crucial metrics is able to predict game wins/losses for the home team given a total set of 19 metrics, with ~80% accuracy. The state of the model is optimal, as the typical NBA upset rate is ~32%, which means that 32% of the time, the predicted favored team (based on pre-existing metrics) will end up losing. Model performance is about the same between test/training sets which means that the model's success isn't heavily reliant on the data used to train it, and will perform relatively well in the face of new data.

The model identified the following stats as critical in determining game wins, in order of importance: 3 Point %, 2 Point %, Rebounds, and Assists. Although the feature importance diagram specifically lists home/away team designations, these designations should be ignored when evaluating overall model performance; it can be assumed that assists/rebounds generally both contribute to win rates.

Translating these findings to prioritizing player picks/trades players overall higher scoring percentages, particularly when it comes to 3PT shots should be idealized, especially in today's 3-point driven game state. If these metrics are similar/inconsistent across players, players with higher rebound rates should be prioritize. Rebounds are a clear indicator of how well a player is able to maintain ball possession, on both the offensive/defensive ends of the court. Finally, if scoring percentages/rebound rates are similar, players with higher average cumulative assists/game should be favored, as they can be seen as more cohesive/better at creating scoring opportunities.

Similarly, young NBA players that have aptitude for one/many of these metrics should be highlighted as key players to look out for in the future. For example, Zion Williamson, a young superstar on the New Orleans Pelicans, averaged 9 rebounds and 23 points during his time at Duke University. Similarly, Ja Morant, a current key offensive presence on the Memphis Grizzlies, averaged 25 points and 10 assists during his last year at Murray State.

## Next Steps

Due to the nature of the success metric, accuracy, next steps in improving this model would largely involve the addition of variables that help better represent/determine game wins. Upon finishing the analysis side of the project, it was found that there are four main metrics that are actually used to best determine win percentage: effective field-goal percentage, turnover percentage, offensive rebound percentage, and free throw rate.

Although two of those metrics are covered within our model (turnover percentage and free throw rate), the potential effect of effective field goal percentage and offensive rebound percentage on model performance/resulting accuracy would be interesting to see.

Currently, the **TOTAL TEAM REBOUNDS** within the model is a cumulative metric that takes both defensive/offensive rebounds into account. Offensive rebounds should be looked at in particular, as they reveals a team's tendency to keep possession of the ball, allowing them to better utilize remaining time on a shot clock, slow down game pace, and create different opportunities for scoring. In contrast, defensive rebounds involve regaining possession after a shot attempt has been made, but this metric doesn't inherently lead to scoring on the other end of the court.

Effective field-goal percentage is technically present in the current model state, and is calculated by evaluating 3-point percentage and 2-point percentage within one metric. The metric gives more weight to the 3-point field goal made, since it is valued higher than a normal 2-point shot. Using

this aggregated metric would be interesting to see, and may potentially improve/decrease resulting accuracy.

Injuries should be looked at as well, as a team missing their key offensive/defensive pieces has less player options to cycle through the duration a game. Considering the effect of the pandemic, several players were added to rosters on 10-day contracts. This sense of impermanence within a team could definitely throw off chemistry and game performance.

Finally, outside of the realm of general basketball statistics, a way to quantify external factors that involve NBA players would be of interest, whether they be interactions with media, other team players, or the city that a player is playing in. Some of these factors could have significant effects on a player's performance.

For example, Kyrie Irving on the Brooklyn Nets is widely regarded as one of the best point guards of all time. Controversy involving his departure from the Boston Celtics in 2019 caused him to be booed whenever he held the ball in TD Garden (Celtics home court). In an ideal scenario, a professional basketball player would not be affected by these factors, as they are expected to play under pressure at all times. However, as seen from Game 2 in the first round of the 2022 playoffs, Kyrie was held to an abysmal 31% FG percentage, with 4 out of 13 made attempts. While this could be a factor of relentless Celtics defense, the crowd's energy undoubtedly had an effect on his lackluster performance.

Russell Westbrook is a player who has been under a tremendous amount of scrutiny during the season since he joined the Lakers in 2021. The fans and media have started calling him "Westbrick" due to his recent trend of bad shot selection/playmaking. While this could be seen as a factor of Westbrook not yet getting adjusted to his role on the Lakers, the constant media/fan attention to the former MVP can be seen a contributing factor to his poor performance. This season, the Lakers noticeably struggled and failed to make the playoffs, despite being favored to win the 2021-2022 NBA championship by some fans/analysts.

Rivalries, whether on a player-by-player level or a team-team level undoubtedly have a factor on resulting game metrics. Long-standing rivalries like the Lakers and Celtics, or recently formed rivalries like the Atlanta Hawks/NY Knicks & Brooklyn Nets/Milwaukee Bucks, are inescapable factors that affect the energy of players and fanbases. On a player-by-player basis, the tension between James Harden and Giannis Antentokounmpo is a notable example of a recently formed "rivalry". When Harden was still an active player on the Rockets, he was noted as saying, "I wish I was 7-feet, and could run and just dunk. That takes no skill at all. I have to actually learn how to play basketball and how to have skill. I'll take that every day." The slight against Giannis undoubtedly fueled his 44-point performance against Harden in a Nets/Bucks matchup on March 30, 2021. Giannis was seen to address Harden's comment in that post-game interview saying, "It's good because I'm changing the narrative. I don't want to be the guy only that dunks and runs. I can make a three.".

In [ ]:
```
1
```