Team 4

Project Report: NBA Basketball Analytics and Game Predictor

Hruday Pratipathi, Hardeep Kainth, Lohit Kishore, Jimmy Valdez

CS 133 Section 01

November 29, 2025

Table Of Contents

I: Abstract

The NBA (National Basketball Association) is the largest professional basketball league in the world, being based in America and holding a total of 32 teams that play in an 82-game season, and afterwards, in a best of 7 playoff tournament to decide a team that wins an NBA championship. It contains millions of fans around the world, with millions of basketball players vying to get into the league. Alongside that, NBA recruiters look for the best talent they can find to fix holes in their team and try to figure out what they need to improve on in order to win more games, strengthen their team, and win the championship. To find trends among winning teams, we have taken a dataset containing information of player box scores from games dating from 2010 to 2024. Using this data, we want to answer questions regarding what statistics show the most correlation between winning games, such as defense, points per game measurements, being the home or the away team, and more. We use histograms, scatter plots, bar charts, and more to visualize this information and make clear solutions. Later on, we will also create a Machine Learning model that tries to predict who wins a game based off of team stat lines.

II: Background / Problem Definition

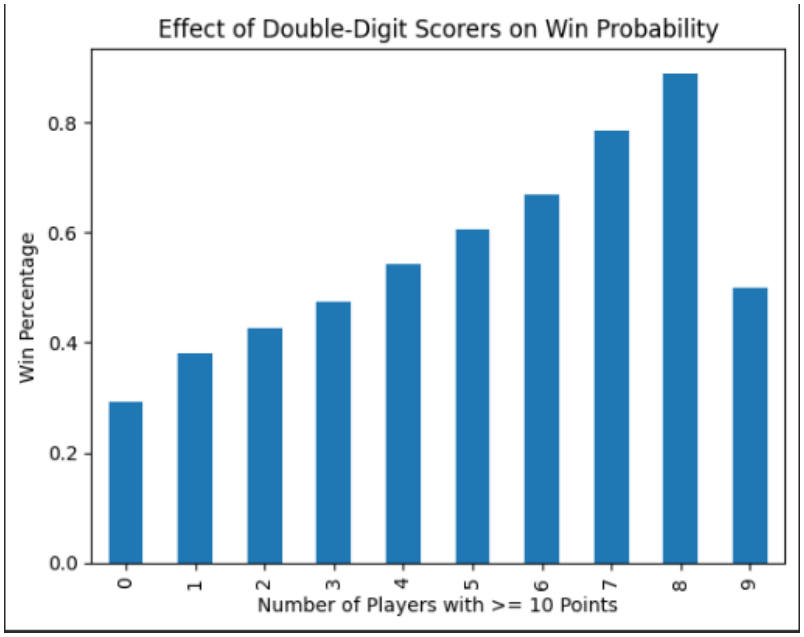
Our dataset comes from a GitHub repository that updates NBA Box Score statistics for data analysis every 6 months, and has been doing so since 2010 and 2024. The database comes from NocturneBear, and it comes from the repository located at https://github.com/NocturneBear/NBA-Data-2010-2024/tree/main. It contains three different player data sets, each containing around 150,000 rows of data, and once merged, contains 424,478 values with 33 columns. It also contains a total team dataset that contains 33,316 values with 56 columns. It contains the season year when the game was played, the date, matchup, the team's unique id, city, name, and tricode, the player's name, id, position, jersey number, and minutes played in a game, and game statistics. The game statistics include points, the number made, attempted, and percentage of field goals, three pointers, and free throws, as well as offensive, defensive, and total rebounds, assists, steals, blocks, turnovers, personal fouls, and finally, the plus-minus stat, which is the differential of the game score when that player is on the court. The team dataset contains the same

identifying team and game information, all of the same game statistic information except aggregated, as well as the rank for every statistic at that current time, which for our purposes, we didn't use.

This dataset contains all of the information necessary from 15 years worth of NBA basketball to analyze past trends and make analysis on impact of winning, which is our overall goal. Our target variable of if the game is won or lost gives us a statistic to correlate our analysis in order to observe potential impact. The five questions that we observed were if having multiple players scoring double digits statistically affected a team's chance to win, how defensive stats such as blocks, steals, defensive rebounds correlate with a team's chances of winning, how a higher PPG from a team's highest scorer correlate with their winning record, if homecourt advantage noticeably increase a team's chances of winning, and a full report of each roster's points per game in the form of a bar chart for each season and team. In the real world, people use this data in order to make conscious changes in the role of players in a team (defense/rebounding specialty players rather than scoring), and try to improve their team in the best way.

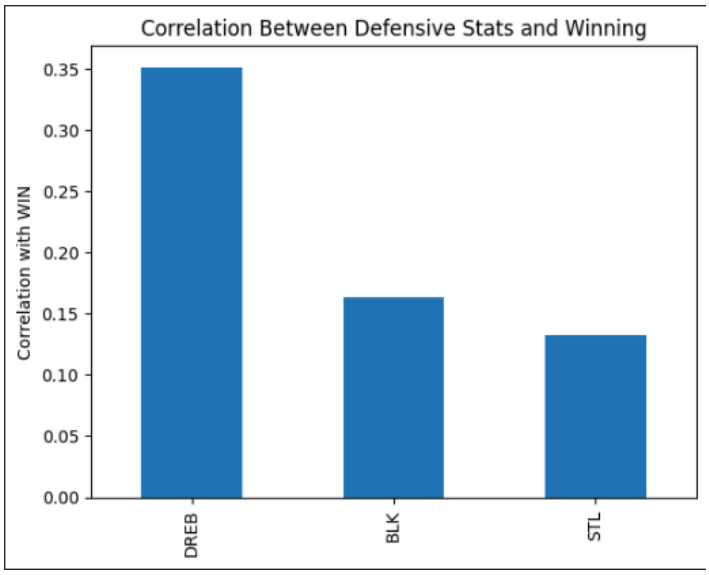
III. Data Visualization Strategy

For each of our questions that I mentioned above, we made visualizations that we can interpret to come to conclusions about the question. For the question of if more double digit scorers increases the chances of winning, we visualized this in a bar graph, and the output is shown here:



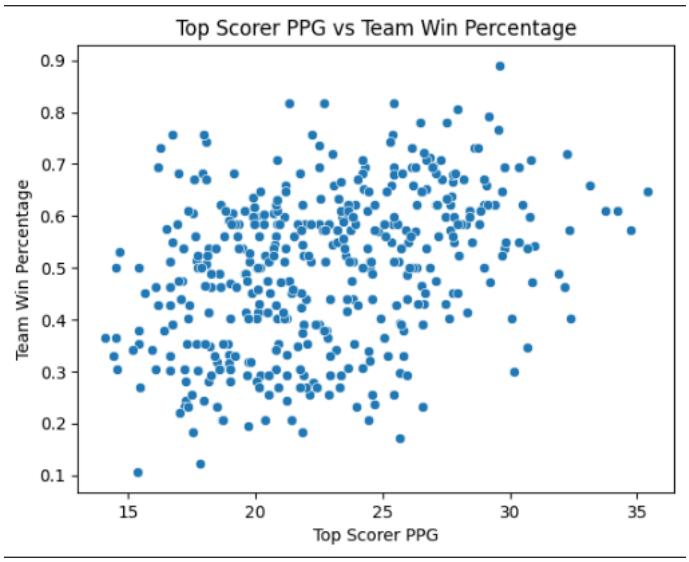
To get to here, we found the number of double digit scorers in the game by finding if a player scored more than 10 points, labelling that as a double digit scorer, and grouping by game to sum up double digit scorers. Now, in the team dataset, we can merge the gameIds of the dataframe with the double digit scorers to find the win proportion based on the number of wins while having an X number of scorers, and graph it as shown above. We see here that it seems the more double digit scorers you have, the more likely you will win, except for if you have 9 scorers.

For the question of defensive stats that correlate to winning, it was actually simple to do. We take our team dataset, take our defensive rows (STL, BLK, DREB), and use the pandas corr() function to find the correlation to the WIN variable. Our graph showed an interesting output:

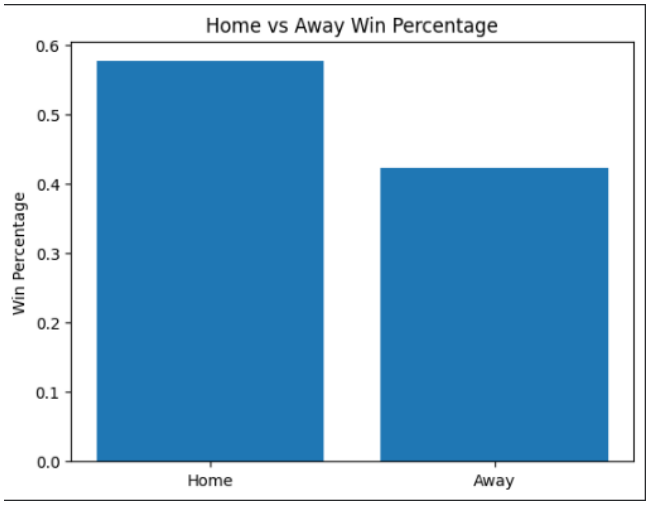


Defensive rebounds are actually more important to win compared to blocks and steals by a factor of almost two! This is definitely interesting to keep in mind!

Our next question is about if a higher PPG from a team's highest scorer correlates with a better win record can be analyzed by grouping by season, team, and personId, finding their points per game by summing points by the number of games played, finding the win proportion by aggregating the team's win graph by season, merging the two by the teamId, and finding the correlation between the two. Interestingly enough, our correlation number is 0.364, which shows a generally weak correlation. The scatter plot below supports this, as there is no clear and apparent line that shows that a higher PPG leads to a better win percentage. This is generally supported by the fact that well-rounded, young teams typically strive in the league.

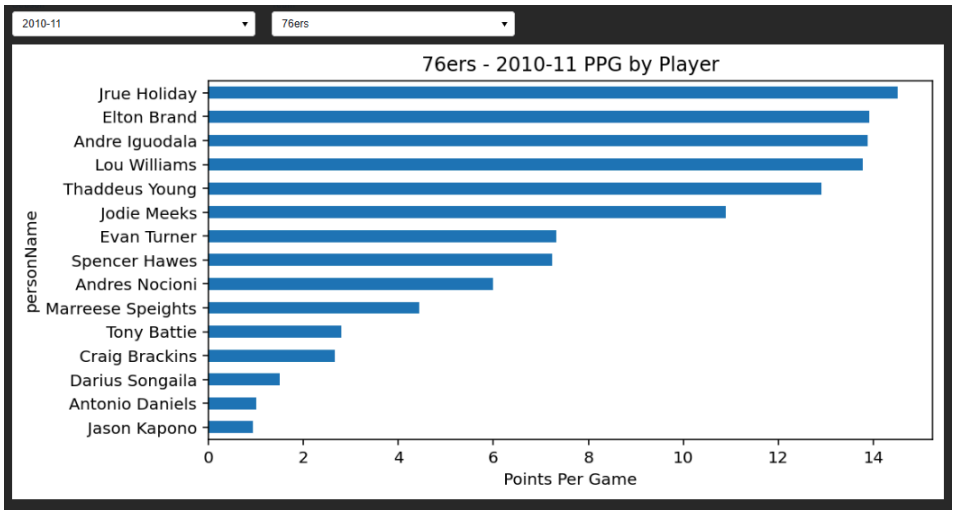


Our next question is very straightforward as well, showing if being at home vs away leads to a better win percentage. This uses the "matchup" variable, as typically, home games are labelled with "vs", for example. LAL vs DAL. LAL would be the home team in this case. When they are the away team, it is labelled with a @, meaning that they are playing @ a certain location. LAL @ DAL would mean that they are playing at the DAL venue. Using a lambda function to filter that out and calculating the win proportions, we find out the following



Home games are favored, but not by much.

Finally, our interactive graph is a display of the points per game of all players on a team's roster for a selected team and season. Using panel widgets, we took the unique team names and the season names, and filtered the player dataframe for performances that fit the selection, found the points per game, and graphed them on a bar graph. The interface looks as follows:



Clicking on the season and team will give a dropdown of options to choose from, and once a value is selected, the corresponding graph with team and player labels will be displayed.

IV: Machine Learning Pipeline and Evaluation

The machine learning question we want to try to predict is: "Can we predict whether an NBA team will win a game based on its and its opponent's statistics?" These predictions will use rolling averages of the games that have been played up to that point, and given the team's average and opponent's average, will decide who wins that game. First, we need to process our team dataframe to have our averages and include our opponent's statistics. We sorted our dataframe based on season and game date and used the pandas expanding() function to find rolling averages up to that game's date for specific game stat columns, the same ones we would use for the model. Once that was calculated, we needed to do a self-merge to get our opponent's statistics in the same row. We did a merge on gameId to get that.

However, it has an unintended side effect of merging its own row, so we need to get rid of the rows that have the same team name from the merge. After this, the columns that our model uses are shown below: ['WIN', 'PTS', 'PTS_opp', 'REB', 'REB_opp', 'AST', 'AST_opp', 'STL', 'STL_opp', 'BLK', 'BLK_opp', 'OREB', 'OREB_opp', 'DREB', 'DREB_opp', 'FGM', 'FGM_opp', 'FGA', 'FGA_opp', 'FG_PCT', 'FG_PCT_opp', 'FG3M', 'FG3M_opp', 'FG3A', 'FG3A_opp', 'FG3_PCT', 'FG3_PCT_opp', 'FTM', 'FTM_opp', 'FTA', 'FTA_opp', 'FT_PCT', 'FT_PCT_opp', 'TOV', 'TOV_opp', 'PF', 'PF_opp', 'PLUS_MINUS', 'PLUS_MINUS_opp']

“Win” is our target variable in this instance, and the rest will be in our feature dataframe. Shown below are some code snippets of our Machine Learning process

```
X = model_df[model_cols].drop('WIN', axis=1)
y = model_df['WIN']
```

After establishing X and y, we did a stratified 80/20 split to maintain the same proportion of wins and losses in the training and testing set.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y)
```

The three models that we elected to use were the Logistic Regression Classifier, Perceptron Classifier, and Random Forest Classifier models to see which one performed the best in this context. We applied the StandardScaler to features, because some of the stats operate in different scales, particularly the points and defensive stats like blocks and steals.

```
log_res = Pipeline([
        ('scaler', StandardScaler()),
        ('clf', LogisticRegression(max_iter=1000))
    ])
perceptron = Pipeline([
        ('scaler', StandardScaler()),
        ('clf', Perceptron(max_iter=1000, tol=1e-3))
    ])
random_forest = Pipeline([
        ('scaler', StandardScaler()),
        ('clf', RandomForestClassifier())
    ])

log_res.fit(X_train, y_train)
perceptron.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
```

Using 10-Fold Cross Validation to find the best model, we ran all three of them to find out that all of the models were quite close, however, the Logistic Regression performed the best, with the highest accuracy and lowest standard deviation between the losses.

```
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Evaluate Logistic Regression
log_scores = cross_val_score(log_res, X, y, cv=cv, scoring="accuracy")

# Evaluate Perceptron
percep_scores = cross_val_score(perceptron, X, y, cv=cv, scoring="accuracy")

# Evaluate Random Forest
rf_scores = cross_val_score(random_forest, X, y, cv=cv, scoring="accuracy")


print("Perceptron 10-fold accuracy:", np.mean(percep_scores), "±", np.std(percep_scores))
print("RandomForest 10-fold accuracy:", np.mean(rf_scores), "±", np.std(rf_scores))
print("Logistic Regression 10-fold accuracy:", np.mean(log_scores), "±", np.std(log_scores))
```

```
Perceptron 10-fold accuracy: 0.5687142782739583 ± 0.03184326608959654
RandomForest 10-fold accuracy: 0.6132228546970698 ± 0.008331318953990786
Logistic Regression 10-fold accuracy: 0.6352489990203098 ± 0.0075726827656048
```

With the lowest standard deviation, we can see that it performs a handful of percent higher with very good consistency, to the point where it's almost at around a hundredth of a percent of deviation. Testing the Logistic Regression on the test data, we get a similar accuracy of 64%, with other statistics like Precision, Recall, and F1 Score.

```
y_pred = log_res.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Accuracy: 0.6436599929750615
Precision: 0.6427076064200977
Recall: 0.6469968387776607
F1 Score: 0.6448450901452827
```

Looking at our confusion matrix, we can get a grasp of how our model performed in terms of classification.

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[1823 1024]
 [1005 1842]]
```

Confusion matrices measure the top two rows as true negatives and false positives, and the button as false negatives and true positives. We can see that our model predicted losses that were actually wins more often than wins that were losses, with 1823 true negatives vs 1024 false positives. In nearly the same proportion, the model evaluated the correct losses and correct wins.

Now, we can interpret the above metrics. Accuracy measures the proportion of correct predictions. The Precision answers true positives over all win classifications, or if a model predicts a win, how often is that correct. Recall answers how many of the wins the model successfully identified. The F1 Score is just a balance between precision and recall, which at this point, are so close it doesn't really change much.

V: References

Dataset Repo: https://github.com/NocturneBear/NBA-Data-2010-2024/tree/main

Pandas API: https://pandas.pydata.org/docs/reference/index.html

MatPlotLib Example Library: https://matplotlib.org/stable/gallery/index.html

Scikit Learn Library: https://scikit-learn.org/stable/api/index.html

Professor Westfall's Machine Learning Examples:

https://github.com/CS133-DataVisualization/F25-classfiles/tree/main/Module05_Machine-learning

Pandas merging frames with the same schema:

https://stackoverflow.com/questions/45914530/merge-two-dataframes-with-the-same-schema

Pandas corr() method: https://www.geeksforgeeks.org/pandas/python-pandas-dataframe-corr/

Panel select widget: https://panel.holoviz.org/reference/widgets/Select.html

Scikit Learn Pipelines: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html