Fantasy Basketball League: Predicting Game Results

Ye Ri Park 3035712080, Pranay Sandeep Periwal 3035664697, Yashvardhan Srivastava 3035664659

1. Objectives:

Basketball fans love to discuss and guess which team consisting of their favourite players will win. Be it putting all time greats head to head or thinking about their abilities as the GM of a franchise, these discussions have never had conclusive answers. With our proposed AI application, we aimed to use statistics of past performances of players to predict the outcome of a game between any two teams with the players being chosen by users.

We also extended our model and applied it to a randomly generated fantasy basketball league (including the user's team) to let them see how they would fare against the various franchises in the league. Using the NBA Playoff elimination style, we let the users see how far they make into the final tournament.

At the end, we managed to create an application with relatively high accuracy that satisfyingly predicts the outcome of games depending on the players chosen.

2. Three Highlights of Our Work:

- ➢ Created a logistic regression model to predict NBA game outcomes using player statistics (from a Kaggle dataset)[1] with an accuracy of ~80%
- ➢ Conducted extensive data engineering - combining correlated features, one hot encoding, standardization of values, removing noise from data by dropping extraneous features - to increase accuracy from 50% to 80% (interim prototype to final)
- ➢ Tested with four models (logistic regression, SVM, XGBoost, and neural network) to find the most appropriate and accurate model for our project

3. Data Exploration:

The datasets are from Kaggle NBA games data. There are two datasets we worked with. The first dataframe consists of the statistics of players for each game, and the second dataframe consists of the match results - whether it is a win or loss for the **home** team.

[1] https://www.kaggle.com/nathanlauga/nba-games?select=games_details.csv

A. Dataframe 1: Player Statistics

The player statistics dataset originally had 602,767 rows x 28 columns. For the final dataset, we dropped the following columns from the dataset: 'PLAYER_ID', 'TEAM_ABBREVIATION', 'TEAM_CITY', 'PLAYER_NAME', 'COMMENT', 'FGM', 'FGA', 'FG3M', 'FG3A', 'FTM', 'FTA', 'TO', 'PF', 'PTS', 'PLUS_MINUS', 'OREB', and 'DREB'.

After dropping features for the final dataset to be used for training (for more details, refer to section 3), we have 'GAME_ID', 'TEAM_ID', 'START_POSITION', 'MIN', 'FG_PCT', 'FG3_PCT', 'FT_PCT', 'REB', 'AST', 'STL', and 'BLK'.

Labels of Each Feature

- 'FG': Field goals (shots)
- 'FT': Free throws
- 'FG3': 3PT field goals
- 'FGM': Field goals made
- 'FGA': Field goals attempted
- 'FG_PCT': Field goal percentage (made/attempted)
- 'FG3_PCT': Field goal (3 PT) percentage (made/attempted)
- 'FT_PCT': Free throw percentage (made/attempted)
- 'REB': Total rebounds (catching missed shots)
- 'OREB/DREB': Offensive/defensive rebounds
- 'AST': Assists
- 'STL': Steals
- 'BLK': Blocks
- 'TO': Turnovers (ball lost to the other team)
- 'PF': Player fouls (number of fouls committed)
- 'PTS': Points
- 'PLUS_MINUS': Net points (points - points due to turnovers etc.)

On observing the data, we find that certain rows have no statistics for some players. This is because the given player did not participate in the game. Hence, these rows were dropped. We also dropped players who had 0 seconds in the 'SEC' column.

This is what the first dataframe after dropping extraneous columns and rows looks like.

	GAME_ID	TEAM_ID	START_POSITION	MIN	FG_PCT	FG3_PCT	FT_PCT	OREB	DREB	REB	AST	STL	BLK	TO	PF	PTS	PLUS_MINUS
0	22000645	1610612754	F	41:45	0.455	0.455	0.000	0.0	5.0	5.0	2.0	2.0	2.0	2.0	2.0	15.0	8.0
1	22000645	1610612754	F	35:06	0.385	0.000	1.000	2.0	9.0	11.0	1.0	3.0	0.0	5.0	6.0	17.0	-2.0
2	22000645	1610612754	C	29:58	0.500	0.600	0.833	1.0	5.0	6.0	1.0	0.0	5.0	2.0	3.0	16.0	6.0
3	22000645	1610612754	G	34:42	0.278	0.333	1.000	0.0	3.0	3.0	3.0	1.0	0.0	0.0	1.0	13.0	1.0
4	22000645	1610612754	G	40:35	0.235	0.273	0.500	3.0	3.0	6.0	10.0	1.0	0.0	2.0	1.0	12.0	6.0

B. Dataframe 2: Game Results

The player statistics dataset originally had 24195 rows x 28 columns. The reason there are less rows for game results is that one row of game results represents one game, while multiple rows of player statistics represent one game[2].

We dropped all columns besides 'GAME_ID', 'HOME_TEAM_ID', 'VISITOR_TEAM_ID', 'HOME_TEAM_WINS', and 'SEASON'. We also dropped games that have been dropped in the first dataframe. Later, only 'HOME_TEAM_WINS' was used for the input for training. 'HOME_TEAM_WINS' represents 1 for win and 0 for loss for the home team.

This is what the second data frame looks like.

	GAME_ID	HOME_TEAM_ID	VISITOR_TEAM_ID	HOME_TEAM_WINS	SEASON
22776	11500001	1610612746	1610612743	1	2015
22774	11500002	1610612753	1610612766	0	2015
22775	11500003	1610612754	1610612740	0	2015
22773	11500004	1610612747	1610612762	0	2015
22772	11500005	1610612761	1610612746	1	2015

C. Comparing Two Dataframes for Data Compatibility

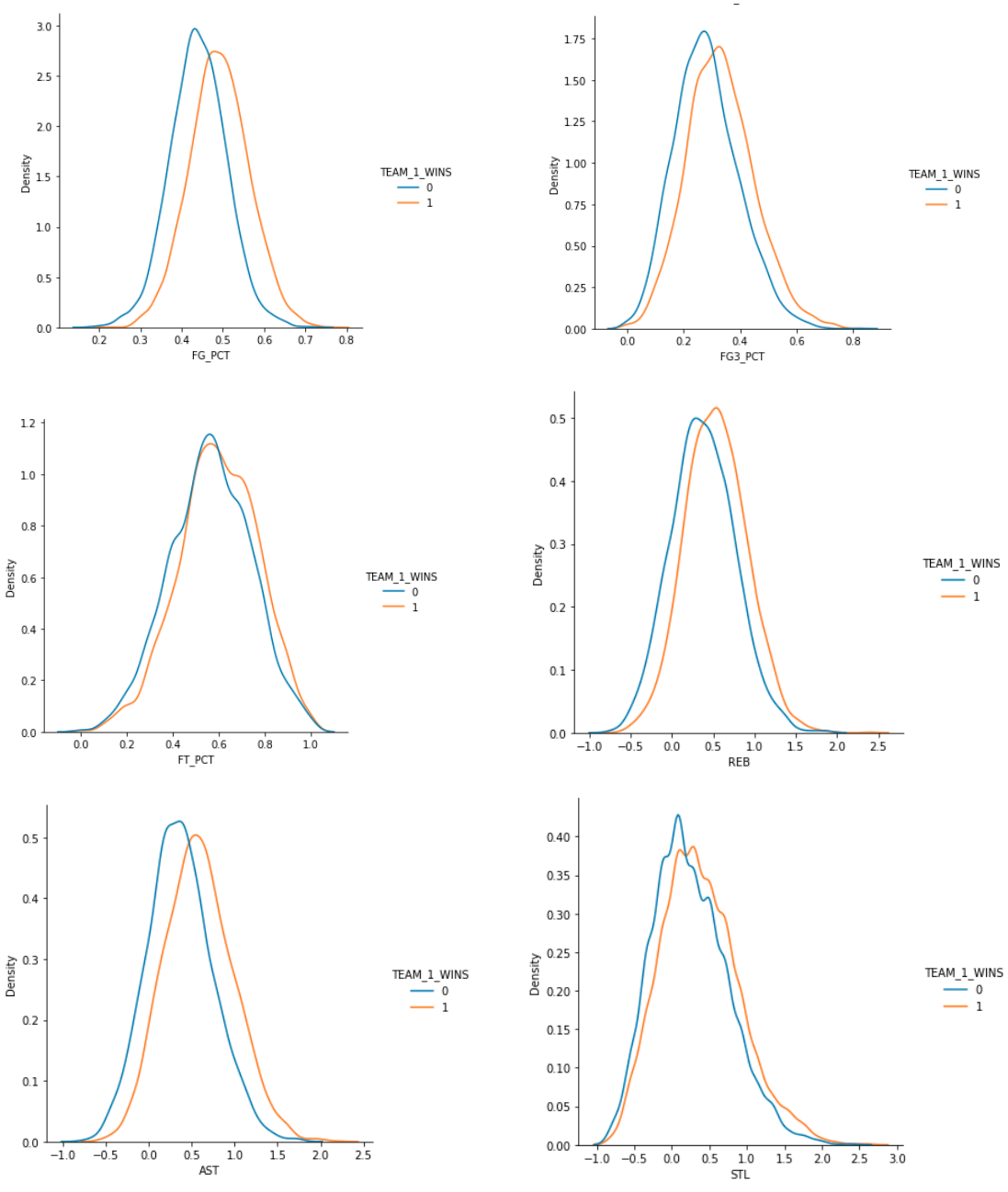
We sorted both tables by 'GAME_ID' in an ascending order so that the orders matched for both dataframes. While sorting, the 'HOME_TEAM_WINS' was converted to represent win or loss for the team with a lower ID. This is because the original win or loss represents the result for the home team and our player statistics data frame is sorted in an ascending order of 'TEAM_ID' within each 'GAME_ID'. We also renamed 'HOME_TEAM_WINS' to 'TEAM_1_WINS' since the win or loss does not represent the home team anymore.

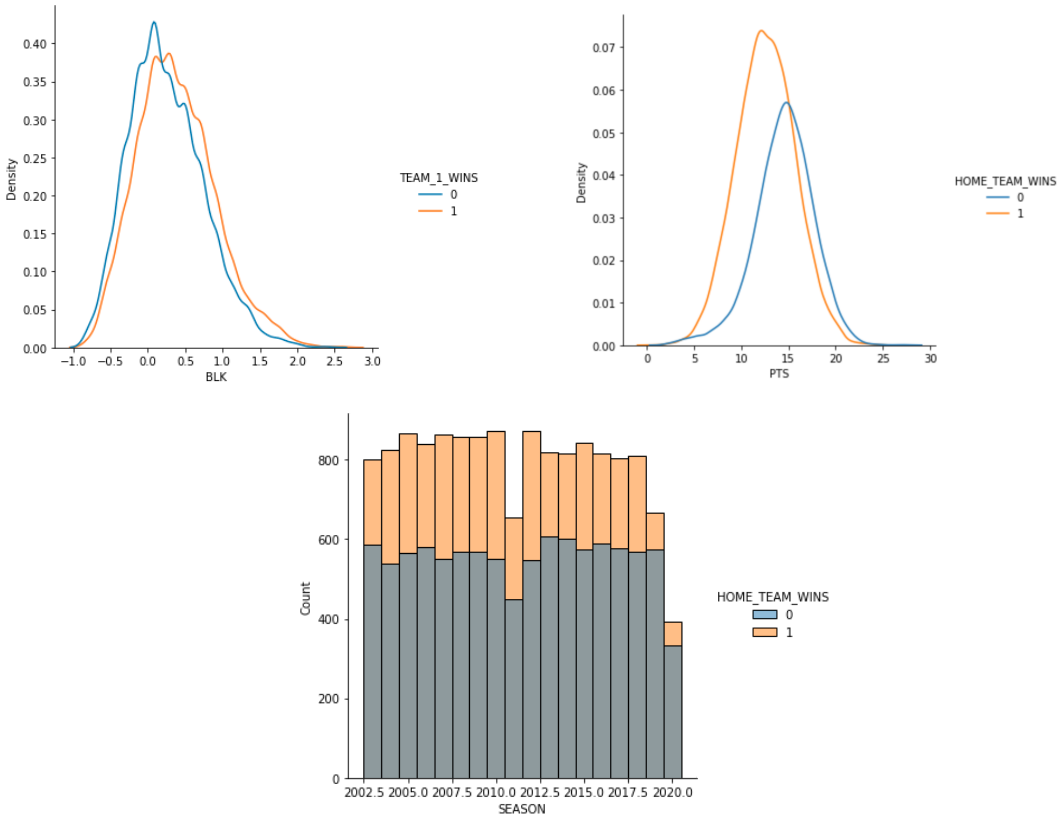
We made sure that the number of matches and 'GAME_ID's are the same for the first and the second dataframe. There were also some same rows duplicated in the second dataframe, so we dropped those rows. We managed to have the data sets match in terms of 'GAME_ID' for training and testing.

[2] https://www.kaggle.com/nathanlauga/nba-games?select=games.csv

4. Data Visualization and Data Analysis

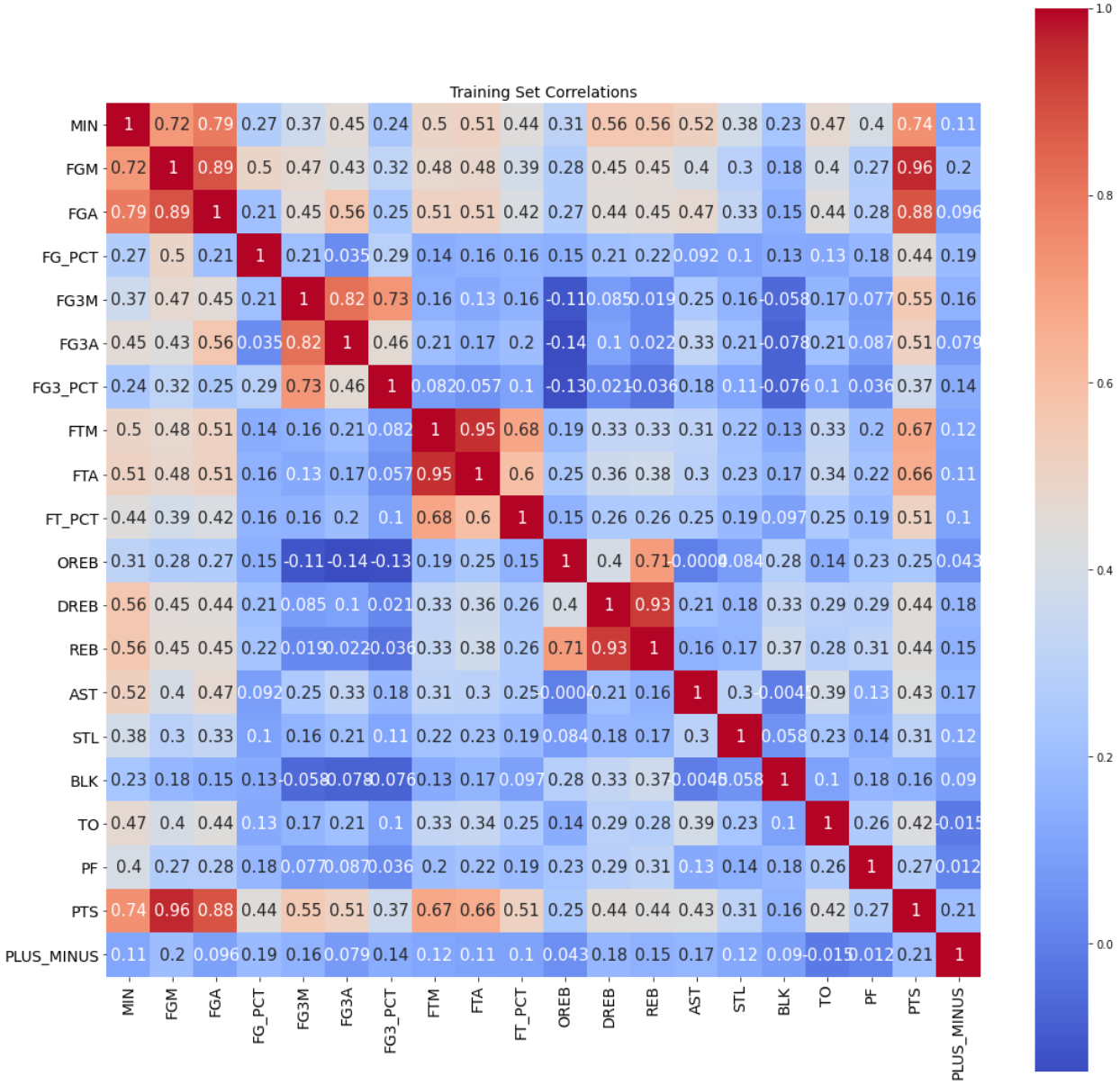
Then, the following graphs are shown in the order of FG_PCT, FG3_PCT, FT_PCT, REB, AST, STL, BLK, PTS, START POSITIONS for guard, forward, center, and bench. Each graph represents the density or counts of each feature for both win and loss. The blue represents the loss, and the orange represents the win for team 1.





As seen above, the win and the loss graphs are quite similar to each other, but they are still different. The win graphs tend to have a peak at a higher value of continuous features. We conducted data engineering to figure out whether these features help the models find the patterns.

The heat-map of correlation between each feature is shown below.



The correlation heat-map reveals high correlation between the following features: FGM and FGA, FG3 and FGM, FTA and FTM, DREB and REB, PTS and FGM, PTS and FGA, PTS and MIN. Most of these correlations are obvious such as FGM and FGA (the more field goals you attempt, the more field goals you'll likely make) or DREB and REB (REB = DREB + OREB) or PTS and FGM (the more field goals made, the more points you'll have). However, the correlation between PTS and MIN provides an insight - the more time a player plays for, the more points they can score. Further, the PTS statistic alone does not show the impact of each player on the game. Hence, PTS and MIN (time played) may be wrapped up into one feature.

5. Data Engineering:

There were three stages of our dataset: interim prototype, further testing, and final dataset.

Stage	Number of Players Per Team	Number of Features (Player Statistics)	Range of Seasons
A. Interim Prototype Dataset	8	18	All Seasons (2002-2020)
B. Further Testing Dataset	5	18	All Seasons (2002-2020)
C. Final Dataset	5	12	2015-2020

For our final dataset, we reduced the input data by narrowing our seasons to 2015-2020 (down from 2002-2020). This was done as the play style of players in the NBA has changed drastically in the past 6 years and using all the available seasons made it difficult for the model to identify patterns (as seen in the low accuracy with previous datasets, refer to 7). Rows with incomplete data were dropped.

The columns ['PLAYER_ID', 'TEAM_ABBREVIATION', 'TEAM_CITY', 'PLAYER_NAME', 'COMMENT', 'FGM', 'FGA', 'FG3M', 'FG3A', 'FTM', 'FTA', 'DREB', 'OREB', 'TO', 'PF', 'PLUS_MINUS'] were dropped as these features are either irrelevant such as Player ID, Team Abbreviation, already exist in another form in the dataset such as FGM and FGA (FG_PCT = FGM/FGA, OREB+DREB = REB), or do not correlate much to the game result such as TO, and PF.

The positions of the players were one hot encoded to aid the model find correlations between starting positions and player statistics because certain positions emphasize on particular statistics (Guards on shooting and passing, forwards on rebounds and blocks etc.). Firstly, non starting/bench players were assigned the position 'NO'. Then, the encoder created new features **START_POSITION_1, START_POSITION_2, START_POSITION_3, START_POSITION_4** corresponding to G, F, C and NO respectively.

From (4) Data Analysis, we noticed a correlation between points and time played. Using this, we created a new feature 'PTS/SEC'. This adjusts the points scored for the time that they played and provides a more accurate representation of their impact on the game.

To normalize the values, we tried 3 different methods - scaling, dividing by the mean and standardization. Through testing, we observed that standardization provided the best accuracy

(refer to (7) Experiment Results). As such, we standardized the columns **'PTS/SEC', 'REB', 'AST', 'STL' and 'BLK'** for our final dataset.

The remaining three features '**FG_PCT**', '**FG3_PCT**', and '**FT_PCT**' are percentages of field goals and free throws. As such, they do not need to be normalized as they are already between 0 and 1.

We restructured the output dataset to match the input dataset to indicate the W/L outcome of the game for the team with the smaller team ID (as our input dataset is sorted by team ID).

We spliced the data set created so far by season. We used the 2020 season as our test dataset and seasons from 2015 to 2019 as our training/development dataset. Finally, we converted the data frame into a NumPy array.

In the preparation for the datasets for the models, we dropped 'TEAM_ID', 'GAME_ID' from the first dataframe and 'GAME_ID', 'HOME_TEAM_ID', and 'HOME_TEAM_ID' as they are not needed for the model's predictions. We're left with 12 features at this point.

Also, for the final dataset, there are 5 players (chosen by time) in each team to correspond to the starting players of a basketball game. We also dropped to 5 players to help the model identify the patterns within the data more easily. Further, these 5 players also have the maximum impact on the game (as they played the most)

The dimensions of the inputs and the outputs are different for the classifiers and the neural network. The variable 'n' represents the number of games.

	Dimensions	
	Inputs	Outputs
1) Scikit Classifiers	2-dimensional (n, 120)	1-dimensional (n)
2) Neural Network	3-dimensional (n, 2, 60)	2-dimensional (n, 2)

The inputs are 1) a 2-dimensional numpy array where the first axis represents the number of games and the second axis represents the player statistics for each player in the game (10 players*12 features = 120) and 2) a 3-dimensional numpy array where the first axis represents games, the second axis represents the teams (the 2 teams playing in that match), and the third axis represents the game statistics for each player (12 features*5 players = 60).

The outputs are 1) 1-dimensional numpy array, represented by 0 for loss and 1 for win for team 1, which is the team with a smaller team_id and 2) 2-dimensional numpy array with the first axis

representing games and the second axis representing win or loss (1 or 0) for both teams in the input.

6. Models Implemented:

The scikit models we implemented are logistic regression, support vector machines, and extreme gradient boosting classifiers . We chose these models as they are the go-to methods for binary classification problems.

Logistic Regression Model:

```
lr = LogisticRegression(solver='lbfgs', max_iter=10000)
```

Support Vector Machine:

```
clf = svm.SVC(kernel='linear')
```

Extreme Gradient Boosting Model:

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1, random_state=0).fit(X_train, y_train)
```

The sequential neural network was implemented using TensorFlow and Keras. Neural networks are capable of identifying complex patterns within the dataset. Given the diversity of the data points within our data, the neural network model can find patterns and predict outcomes with a high accuracy.

This is the final neural network model. The model was trained with 30 epochs and 200 batch size.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 2, 60)	3660
dropout (Dropout)	(None, 2, 60)	0
dense_1 (Dense)	(None, 2, 30)	1830
dropout_1 (Dropout)	(None, 2, 30)	0
output (Dense)	(None, 2, 1)	31

7. Experimental Results:

A. Interim Prototype (Refer to 3 for the dataset structure)

	Logistic Regression	SVM	Neural Network
Accuracy (%)	48.55	48.05	51.1

B. Further Testing (Refer to 3 for the dataset structure)

	Normalization Methods	Logistic Regression	SVM	XGBoost	Neural Network
Accuracy (%)	MinMaxScaler	50.16	49.86	49.46	50.64
	Dividing by Mean	50.24	50.15	49.99	50.04
	Standardization	50.42	50.16	49.73	50.23

C. Final (Refer to 3 for the dataset structure)

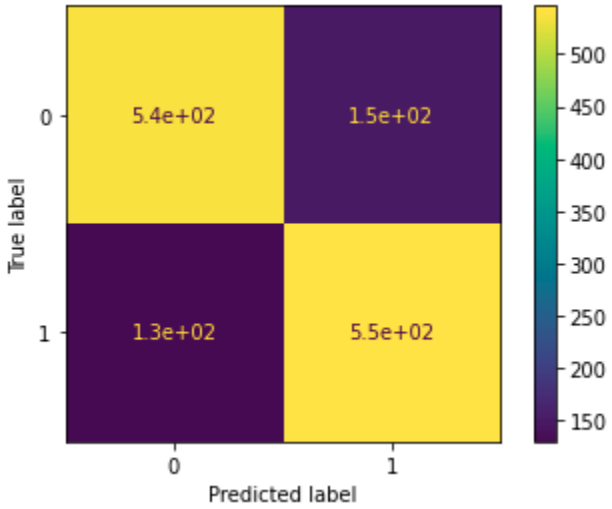
	Normalization Methods	Logistic Regression	SVM	XGBoost	Neural Network
Accuracy (%)	MinMaxScaler	77.83	77.62	76.89	63.94
	Dividing by Mean	79.31	79.01	77.19	64.51
	Standardization	79.41	78.97	77.51	67.84

From the above table, we can see that standardizing the features led to the highest accuracies.

	Dataset	Logistic Regression	SVM	XGBoost	Neural Network
Accuracy (%)	Dev Set	79.41	78.97	77.51	67.84
	Test Set	79.56	78.67	76.00	69.04

The logistic regression model led to the highest accuracy of **79.41%** with the development set and **79.56%** with the test set.

8. Model Evaluation and Comparison:



Confusion Matrix for Logistic Regression

This confusion matrix represents how our Logistic Regression model performed in terms of the prediction and its accuracy. As you can see, we had false positives ~21% of the times and false negatives ~19%. This was a great improvement from the previous precision which was 50% (random prediction).

Our new numbers are[3] :
Accuracy = 79.41% (Correct predictions/All predictions)
Accuracy helps us decide how many correct predictions our model is making, win or loss. This is very relevant for us as in sports betting, one can bet that a team would win or lose. Hence, accuracy is important.

Precision = 78.57% (true positions / (true positives + false positives))
Precision is important for us as it helps us understand what is the probability we bet that a team would win, and it does win, as opposed to a false positive. It is important for people betting on team 1 winning using our model.

We followed the industry standard of splitting our data into train/dev/test datasets. Over 2015 to 2020, we had 7,497 games. Around 18% of the data was used for the dev set, 9% for the test set, and the remaining 73% for the train set. The reason why we could not increase the size of the test set is that we were using only the 2020 season for testing to simulate a real-world scenario where we need to predict results for the current season based on previous seasons.

[3] https://www.jeremyjordan.me/evaluating-a-machine-learning-model/

Our accuracy being ~79% has proven to be quite competitive to other models already existing and trying to predict NBA win outcomes. Research done by Josh Weiner in the article linked[4] says that the maximum accuracy of most models they found was a 74.1% (for playoff outcomes). Josh Weiner, and his model, was able to achieve ~67.15% accuracy. Hence, we are quite satisfied with our results from Logistic Regression, having beaten existing models.

9. Possible Issues Analysis:

A. Upsets refer to when a relatively stronger team loses out to a "weaker" team. This often occurs in the NBA, especially during the playoffs. This is unfortunately not possible in our current model as we do not account for upsets.

B. Player statistics are not really the only factor which determines the success of teams. Other external factors such as coaching, and team chemistry affect the performance. One way to resolve the issue would be to find the co-occurrence of different players in the games where a team won. This is an effort to find a pattern for whether specific players perform well together as opposed to other players.

C. We also cannot account for different emotional and psychological factors that may affect a player such as some kind of personal issue going on in someone's life. This may cause them to underperform in a game.

10. Application of the Model

We wanted to build an application which would be fun for 2 friends, who are fans of the NBA, to play against each other. They could also use this application to help simulate games taking place in real time if they wish to get into sports betting. Hence, we built 2 types of applications. The first one is where 2 teams are selected by 2 users, and they can play against each other. The second is when after the 2 users have selected their teams, we have those 2 teams compete in a tournament to see if their teams can come out to be the winners.

The process starts with the application taking input of the 2 teams both the players wish to play with. Each person also picks 5 team players. After this we give the players the choice of running either of the 2 applications, or both if they so choose.

---

4 https://towardsdatascience.com/predicting-the-outcome-of-nba-games-with-machine-learning-a810bb768f20

The first type of game allows them to judge how well their team would perform against the other's. This could potentially help them judge real games simulating an upcoming match and choosing a roster for the game. The second option is more for fun. We try and simulate how the playoffs happen in the actual NBA. There is an Eastern conference and a Western conference. Both these conferences are knockout type competitions, 8 teams in each conference, 1 winner. The winners of both conferences would play against each other to see who is crowned as champion.

Currently, to simplify the process of which teams play in the conferences, we randomly choose 14 teams and sort them into 2 different arrays. Then we add team 1 to the 1st array and team 2 to the 2nd array. Each array represents teams playing in the different conferences. We keep the 2 player-selected teams separate so as to not have them compete against each other at any stage, unless they both make it to the finals. We have added some textual output for each game to make it more fun and seem like the games are taking place as the program runs.

There are however definitely improvements we can make to parts of the application. For instance, we could pick the top 16 teams that are actually going to be playing in the Eastern or Western conference if we simulate all the matches that come before the conferences. This would make things a lot more real as compared to just random teams. We could also come up with a way to change the teams a person has picked during a game i.e. making a team selection dynamic simulating real games by allowing substitutions to be made. This would require a lot more complexity and hence we avoided going into it right now.

11. Possible Limitations and Future upgrades

A. Our current model does not account for home court advantage which basically means that the team playing at home has an edge because of the supporters, the familiarity, etc. Since we sort based on TEAM_ID, we do not account for home court teams. A simple way we could mitigate this is by including another column with a 1 or 0 for home court. However, this does not factor in how important it is playing on home court. Hence, there should be a more sophisticated way to calculate that.

B. Since we are only taking into account 5 players with the highest PTS/SEC, we are not able to simulate how an actual NBA game works with substitutes and bench players and the effect that they may have on the team. This is also hard to factor in because there is no consistency in the number of players on the roster for each team. Some teams may have 12 players while another may have 8. Hence, we need to make compromises to help our model make predictions.

C. In NBA teams change very often, hence our model and dataset would have to be learning and upgrading often. It is hard for us to predict how our model would perform in future seasons because we used only 1 season for our test set.

D. There is also an issue that if a substitute player comes off the bench in a game and plays only for a few seconds but manages to score a few points, the PTS/SEC would be quite high for the player, although the impact was not as much as others. To circumvent this issue, we could consider players who played for a short duration of time such as 1 min to 2 min, then we should decrease the weightage of PTS/SEC for the player. Further work is required to find a suitable solution for this problem.

12. Conclusion

Our project has been very successful in predicting NBA game outcomes, based on player statistics, achieving an accuracy of 79.41%. This accuracy is greater than other published models trying to achieve the same purpose. We carefully engineered our data to ensure that our model can identify patterns within the data. Such as, by using PTS/SEC instead of PTS and SEC, dropping columns which were represented in other numbers or were adding noise to the model, exploring data and found correlations between them, etc, we were able to improve on our interim prototype substantially. While we started out with random prediction (50% accuracy), with a lot of data engineering and dataset manipulation, we were able to use Logistic Regression optimally to serve our goal.