EXAMINING THE IMPACT OF TRADITIONAL BASKETBALL STATS ON THE GAME OUTCOME

Đorđe Martić, Jovan Samardžić

Mentor - Prof. Bojana Milošević
Faculty of Mathematics, University of Belgrade

Abstract

In this research, we will make a statistical model which illustrates basketball games and predicts the game outcome. Using two machine learning methods, Decision trees and Random forests, we will examine which traditional basketball stats have the biggest impact on winning. In an effort to get more precise results, we will generate three Decision trees with different training data sizes. Trees will be graphically represented.

Mentioned methods will be implemented using two R packages – `rpart` and `randomForest`, respectively. Data we will use can be found on the official NBA website.

The results we got tell us that defensive rebounding and three-point percentage are key to winning. Assists, turnovers, and personal fouls also affect the game outcome, but they are less impactful. Random forests give a much better accuracy – around 90%, while Decision trees have accuracies of around 80%.

Coaches can find our conclusions quite handy, since they can modify their practice style in a manner to emphasize improving skills which have an immense influence on winning. Basketball fans can find this project very useful, too — by interpreting our results correctly, they can have fun trying to predict which team will win the game.

Contents

1 Introduction

In the last decade, we have seen the rise of statistics utilization in basketball [1]. Many teams across the globe, but especially in North America, have recognized the potential and power of statistical models and started to implement them in various ways.

For example, many NBA teams executives use them to determine which players would be a good fit for their team. In other words, they use models to make decisions on which players to sign. Since players in the NBA are getting paid millions of dollars per year, statistics can help teams save a lot of money.

Many different projects inspired us to do this research, but one we found very interesting is Michael Armanious', *Men's U-Sports Basketball Analysis* [2]. The author had used data collected from Carleton University Raven's games and had shown which play styles impact winning games.

The purpose of our project is to determine how much **traditional basketball stats** influence winning games and which stat has the biggest influence on the game outcome. Those stats are:

- *Points*;
- *Field goals* – made, attempted & percentage;
- *Three-pointers* – made, attempted & percentage;
- *Free throws* – made, attempted & percentage;
- *Rebounds* – offensive & defensive;
- *Assists*;
- *Steals*;
- *Blocks*;
- *Turnovers*;
- *Personal fouls*.

To do so, we will use certain machine learning methods - **Decision trees** and **Random forests**. Our goal is to get prediction results as accurate as possible.

Since Random forests are a generalization of Decision trees, they should give much more accurate results. However, we will implement both, since Decision tree can be easily visualised and conclusions will be more obvious.

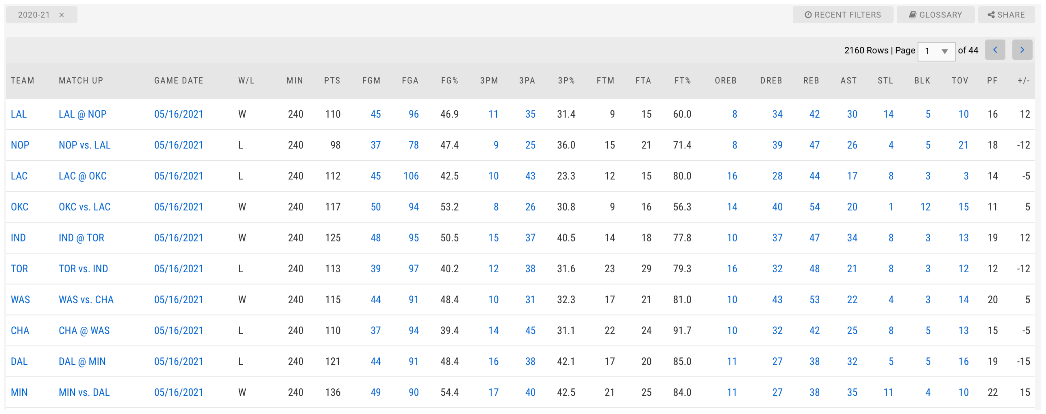
Considering usual accuracies [3] for mentioned methods, we aim to achieve more than 75% accuracy in predicting with Decision trees and 90% accuracy in predicting with Random forests.

This research can be very applicable, since coaches can get insight into which part of the game has the most effect on winning. Eventually, they can adjust practices, in order to improve specific player skills.

2 Materials and methods

2.1 Data

Raw data (figure 1) was collected from the official NBA website [4]. It contains box scores for each game of the 2020/21 NBA season. Each column represents a specific **stat** (e.g. column PF represents the total number of personal fouls the team committed during one game).



TEAM	MATCH UP	GAME DATE	W/L	MIN	PTS	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	FT%	OREB	DREB	REB	AST	STL	BLK	TOV	PF	+/-
LAL	LAL @ NOP	05/16/2021	W	240	110	45	96	46.9	11	35	31.4	9	15	60.0	8	34	42	30	14	5	10	16	12
NOP	NOP vs. LAL	05/16/2021	L	240	98	37	78	47.4	9	25	36.0	15	21	71.4	8	39	47	26	4	5	21	18	-12
LAC	LAC @ OKC	05/16/2021	L	240	112	45	106	42.5	10	43	23.3	12	15	80.0	16	28	44	17	8	3	3	14	-5
OKC	OKC vs. LAC	05/16/2021	W	240	117	50	94	53.2	8	26	30.8	9	16	56.3	14	40	54	20	1	12	15	11	5
IND	IND @ TOR	05/16/2021	W	240	125	48	95	50.5	15	37	40.5	14	18	77.8	10	37	47	34	8	3	13	19	12
TOR	TOR vs. IND	05/16/2021	L	240	113	39	97	40.2	12	38	31.6	23	29	79.3	16	32	48	21	8	3	12	12	-12
WAS	WAS vs. CHA	05/16/2021	W	240	115	44	91	48.4	10	31	32.3	17	21	81.0	10	43	53	22	4	3	14	20	5
CHA	CHA @ WAS	05/16/2021	L	240	110	37	94	39.4	14	45	31.1	22	24	91.7	10	32	42	25	8	5	13	15	-5
DAL	DAL @ MIN	05/16/2021	L	240	121	44	91	48.4	16	38	42.1	17	20	85.0	11	27	38	32	5	5	16	19	-15
MIN	MIN vs. DAL	05/16/2021	W	240	136	49	90	54.4	17	40	42.5	21	25	84.0	11	27	38	35	11	4	10	22	15

Figure 1: Raw data from NBA.com/stats

Before we dive into implementing methods and start making conclusions, we first need to sort out the provided data set. Initially, we deleted the following stats since we did not need them:

- *Team*: because we are predicting the game outcome based on stats, regardless of who is playing;
- *Match up*: same as above;
- *Game date*: because it does not affect the outcome in any form;
- *MIN*: this column is constant and equals 240.

After that, we removed every stat which can be represented as function of the other stats:

- *FGM*: number of made field goals ($FGA \cdot FG\%$);
- *3PM*: number of made three-pointers ($3PTA \cdot 3PT\%$);
- *REB*: number of total rebounds ($OREB + DREB$);

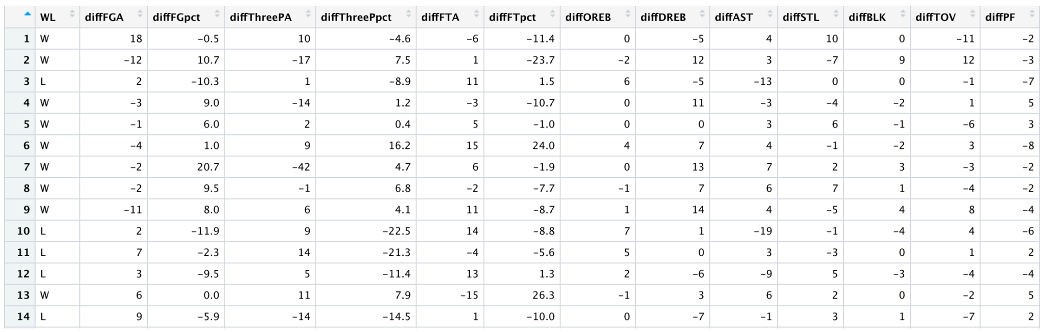
Then, we excluded `PTS` and `+/-`, because using this information would make any prediction on outcome trivial (whichever team scores more points, obviously wins). Also, we decided to exclude `FG%` as well, since it turns out that its importance on the outcome is enormous compared to the other stats.

Finally, we notice how each pair of rows of raw data (figure 1) represents the same game. For example, the first row portrays stats for LAL in their game against NOP and the second row portrays stats from the same game, but for NOP.

Taking this into account, we created a renewed data set, where each row contains information on a single game and columns represent differences between corresponding stats.

For example, if first row in our new data set represents LAL vs NOP game, then:

```
diffOREB(LALvsNOP) = OREB(LAL) - OREB(PEL).
```



	WL	diffFGA	diffFGpct	diffThreePA	diffThreePpct	diffFTA	diffFTpct	diffOREB	diffDREB	diffAST	diffSTL	diffBLK	diffTOV	diffPF
1	W	18	−0.5	10	−4.6	−6	−11.4	0	−5	4	10	0	−11	−2
2	W	−12	10.7	−17	7.5	1	−23.7	−2	12	3	−7	9	12	−3
3	L	2	−10.3	1	−8.9	11	1.5	6	−5	−13	0	0	−1	−7
4	W	−3	9.0	−14	1.2	−3	−10.7	0	11	−3	−4	−2	1	5
5	W	−1	6.0	2	0.4	5	−1.0	0	0	3	6	−1	−6	3
6	W	−4	1.0	9	16.2	15	24.0	4	7	4	−1	−2	3	−8
7	W	−2	20.7	−42	4.7	6	−1.9	0	13	7	2	3	−3	−2
8	W	−2	9.5	−1	6.8	−2	−7.7	−1	7	6	7	1	−4	−2
9	W	−11	8.0	6	4.1	11	−8.7	1	14	4	−5	4	8	−4
10	L	2	−11.9	9	−22.5	14	−8.8	7	1	−19	−1	−4	4	−6
11	L	7	−2.3	14	−21.3	−4	−5.6	5	0	3	−3	0	1	2
12	L	3	−9.5	5	−11.4	13	1.3	2	−6	−9	5	−3	−4	−4
13	W	6	0.0	11	7.9	−15	26.3	−1	3	6	2	0	−2	5
14	L	9	−5.9	−14	−14.5	1	−10.0	0	−7	−1	3	1	−7	2

Figure 2: Working version of our data set

2.2 Methods

Earlier, we mentioned we will use two machine learning methods - **Decision trees** and **Random forests**.

Initially, like in every machine learning method, we need to split our data set into two disjunctive data sets – one for training and one to make predictions of. We will use three different splits of our data set: 90–10, 75–25 and 60–40.

Before fitting, we inspected the data to make sure that training and prediction data have a similar distribution of wins and losses.

Note that all of the codings for this research will be coded using R.

2.2.1 Decision tree

Decision tree is a method that makes decisions by generating a binary tree model in order to perform data classification. It is built by recursively splitting the data based on a single feature, in a way so that it is as homogeneous as possible.

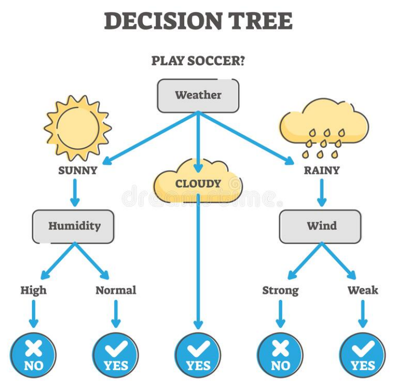


Figure 3: An example of the decision tree

Here we can see a simple example of a decision tree. At each node, we split the data according to the value of the certain feature. So first we divide by weather, if it is cloudy, we will play; if it is rainy our decision will depend on the amount of wind, if it is windy we will not play but if there is little wind we will. If weather is sunny, we look at humidity level: if it is high, we will not play, but if not we will.

There are various R packages which implement this method, but we opted for `rpart` [5].

Function in this package which generates decision tree model is:

```
rpart(formula, data, ...),
```

where `formula` is `Y` $\sim$ X, while `data` is our training data set.

`Y` represents the variable that we want to predict. In this case, that is `WL`.

X represents variables that we want the program to learn from. In this case, we want the program to use all available stats.

This function uses Gini impurity [6] in effort to find out by which stat should it start splitting our data set. It is calculated by formula:

$$\texttt{Gini} = 1 - \sum_{i=1}^{n} p_i^2,$$

where n is the number of different classes (in our case $n = 2$) and p_i is the probability of a instance from the node belonging to class i. In case every instance belongs the same class, `Gini` equals 0. In case distribution of the classes is uniform, `Gini` reaches its maximum. Of course, smaller values are preferable.

To determine how impactful each feature is in building the model, this function uses `variable.importance` parameter. As mentioned in the package long intro [8]:

"An overall measure of variable importance is the sum of the goodness of split measures for each split for which it was the primary variable."

In other words, a variable's importance is the sum of the improvement in the overall Gini measure produced by the nodes in which it appears.

In order to graphically present our model, we use `rpart.plot` function, which is contained in this package, as well.

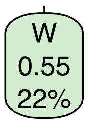


Figure 4: Node from a decision tree

As we see in figure 4, every node has three values written inside of it. Top value is letter `W` or letter `L` – it tells us if there are more wins or losses in that node. Middle value is a number between 0 and 1 – it tells us the winning percentage in that node. Bottom value tells us the percentage of games in this node out of all games from the initial data set.

On the one hand, we don't want to have too many leaf nodes, as it complicates our tree. But on the other hand, we want cross-validation relative error [7] to be as small as possible. Variable which connects these two is **complexity parameter** `cp`. As noted in package documentation [5]:

"A good choice of `cp` *for pruning is often the leftmost value for which the mean lies below the horizontal line."*

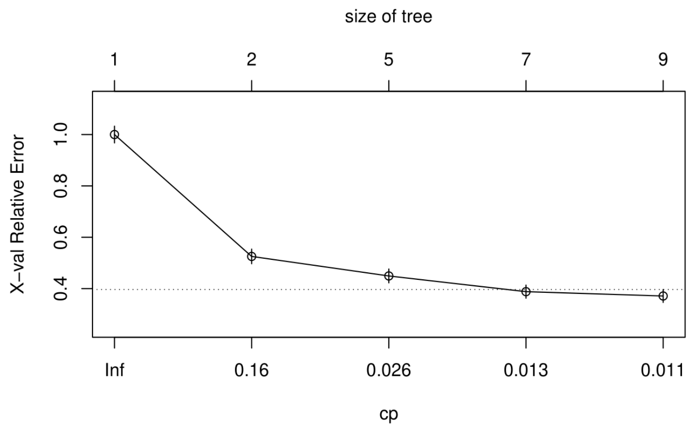


Figure 5: Example of complexity parameter

For example, on figure 5, good choice of `cp` would be 0.013.

After we determine optimal value of `cp`, we can now **prune** our tree. It means that we will transform it so it has an optimal size for predicting by snipping off the least important splits.

2.2.2 Random forest

Random forest, like it's name implies, consists of a large number of individual decision trees that operate as an ensemble. It is a method that uses **boosting** on the decision tree algorithm. Each individual tree in the random forest gives a class prediction and the class with the most votes becomes our model's prediction. This improves the accuracy of the model by doing extra computation. This helps us get a much more precise model that excludes mistakes that a single tree might make.

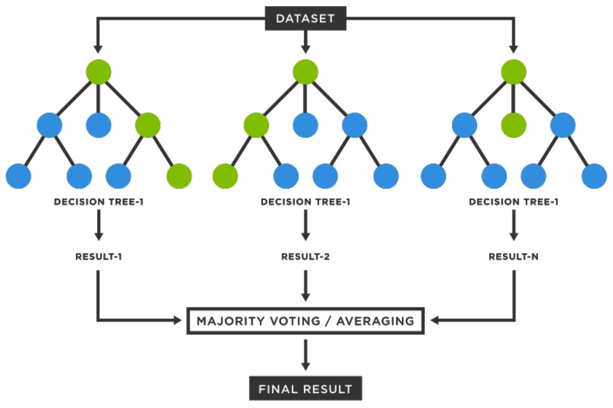


Figure 6: An example of the random forest

In order to implement this method, we have used `randomForest` function from the package of the same name [9]:

```
randomForest(formula, data, ntree, ...).
```

Here, formula and data are the same arguments as from `rpart` function in section 2.2.1, while `ntree` is a number of trees used. In this case, we will use the default number of trees, which is 500.

For the purpose, we will use `importance` function, which is similar to `variable.importance` from section 2.2.1, but it uses average over all trees in the forest.

3 Results

3.1 Decision tree results

3.1.1 Model with 90–10 data split ratio

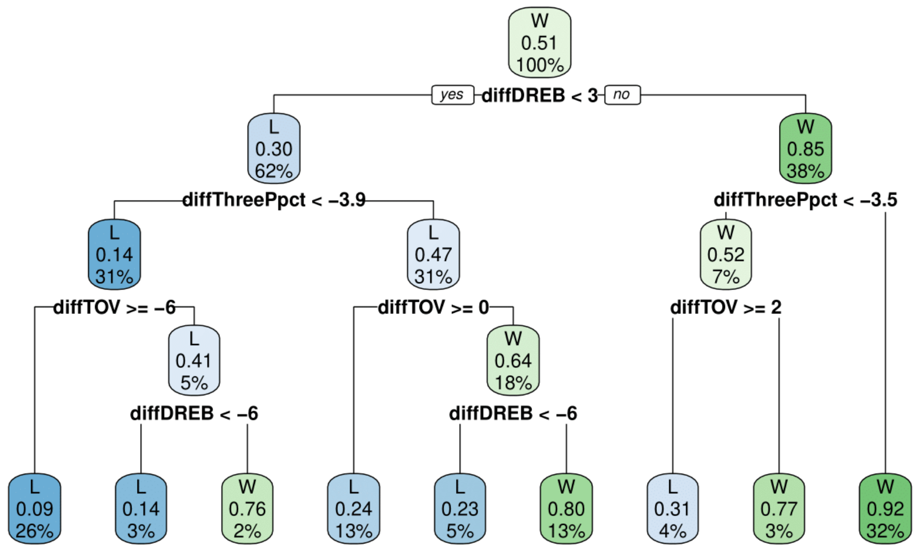


Figure 7: Decision tree based on 90% of data set

In *Figure 5*, we can see visual representation of the decision tree. It implies that `diffDREB` and `diffThreePpct` are the most important stats.

From each leaf node, we can find some information. For example, for the rightmost leaf, we can see that if `diffDREB` is greater than 3 and `diffThreePpct` is greater than -3.5, the game will result as a win in 92% of times and this occurs in 32% of the games.

Value of our model's `variable.importance` confirms these implications, as can be seen in the table 1 in appendix A. This tree uses 12 different variables. Variables `diffTOV`, `diffAST` and `diffFGA` also have significant importance value, while the others are more or less negligible.

By testing our model on 10000 different 90–10 random splits of data, we have accomplished mean prediction accuracy of 81.12%, with a standard deviation of around 3.5%. This accuracy is very good for a decision tree, but not great in general. Finally, different split doesn't mean different `variable.importance` – the order is similar every time and `diffDREB` is by far the most important.

3.1.2 Model with 75–25 data split ratio

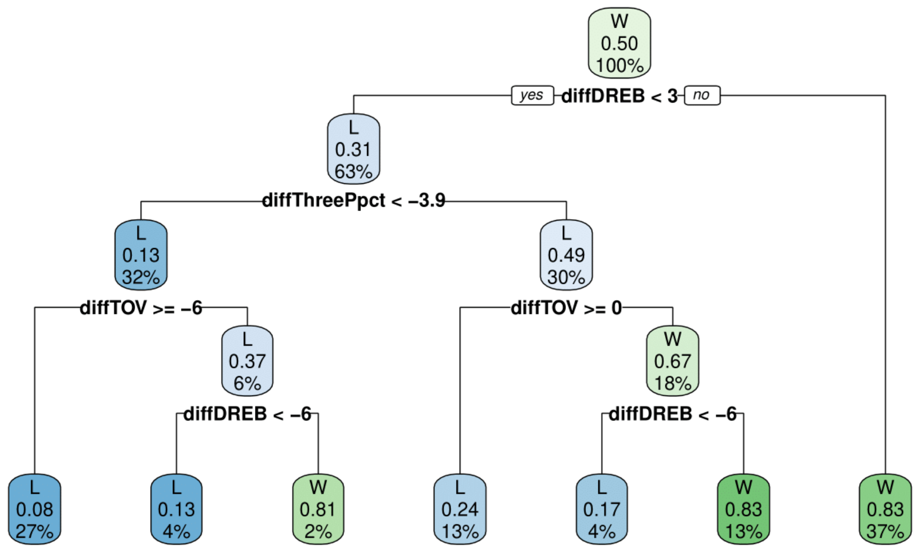


Figure 8: Decision tree based on 75% of data set

The results obtained from this split of data are rather similar to the results we got bit earlier. This tree uses 9 different variables, of which `diffDREB`, `diffThreePpct` and `diffTOV` stand out in `variable.importance`, as can be seen in the table 2 in appendix A.

Having said that, unsurprisingly, the resemblance between trees on figures 7 and 8 is fairly noticeable.

In the same way, as in the previous model, we have accomplished a prediction accuracy of 81.13% with a standard deviation of around 2.3%.

3.1.3 Model with 60–40 data split ratio

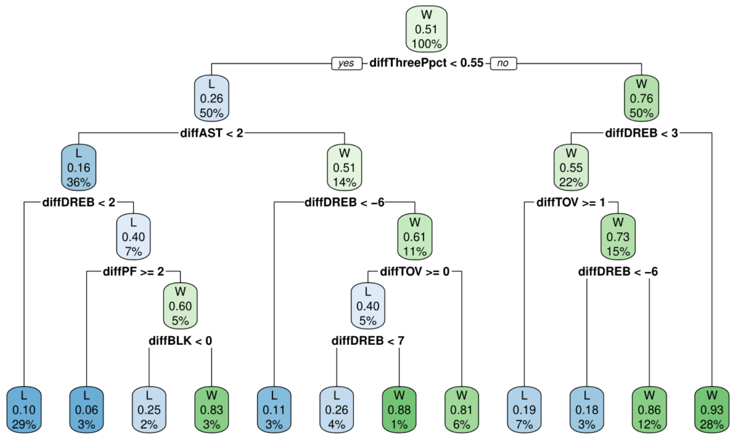


Figure 9: Decision tree based on 60% of data set

In this case, the tree differs from the previous two. It has more nodes, but in the table 3 in appendix A, we can see that it has 11 important variables, which is less than our first model.

In the same table, we see that no variable stands out as the most important. We have `diffThreePpct` and `diffDREB` at the top with almost same importance. Moreover, the other stats are closer to each other.

By testing our model, we have accomplished a prediction accuracy of 80.85% with a standard deviation of 1.9%.

3.2 Random forest results

Using random forest, with 75–25 split of data, we achieved a prediction accuracy of 88.52%.

As for variable importance, table 4 in appendix A shows us similar results as the ones we got with our decision trees. Again, we can see defensive rebounds and three-point percentage standing out. Also, assists have considerably larger importance than the rest of the stats.

4 Discussion

One thing that shows us the decency of our models is that the same stats are the most important in each model. However, the importance values are not as close as we have expected them to be.

In further discussion, we will use results obtained by the random forest model, as it was the best performing one, and try to come to a conclusion why those stats were as important.

We notice that **defensive rebounding** is the stat that has the major impact on winning. Indirectly, that makes sense - if a team has many defensive rebounds, that team gets more possessions. And more possessions give more chances of scoring. Moreover, the greater number of defensive rebounds implies that the opponent misses many shots. At last, if a team lets their opponents grab plenty of offensive rebounds instead, that might lead to plenty of *second–chance points* conceded, since they are often easy to score.

Next up, **three-point percentage** is also very impactful. This result was rather expected, as it is the most effective way to score. **Attempted free throws** and **free throw percentage** come up high as well, as free throws are an opportunity to score without interference from the defence.

Coming up next, we have **assists** – shots after a pass are generally less contested and hence have a higher chance of becoming points.
Finally, **turnovers** being much less impactful than defensive rebounds actually make sense – turnover only means lost possession, but it does not mean that opponents scored afterwards. So defensive rebounds lead to two negative effects and turnover gives only one negative effect.

On the other hand, it is surprising that **steals** have a lower impact, as they stop opponents' attacks and occasionally lead to fast breaks which then frequently lead to easy points.

5 Conclusion

Defensive rebounding has a major impact on the game outcome. Three-pointers are also a vital part of the modern game. Our results only confirm that they are also a crucial factor in winning. Other stats, such as turnovers and assists also have their impact, but in a much smaller portion.

As for accuracies, we confirmed that Random forests give much more precise results than Decision trees.

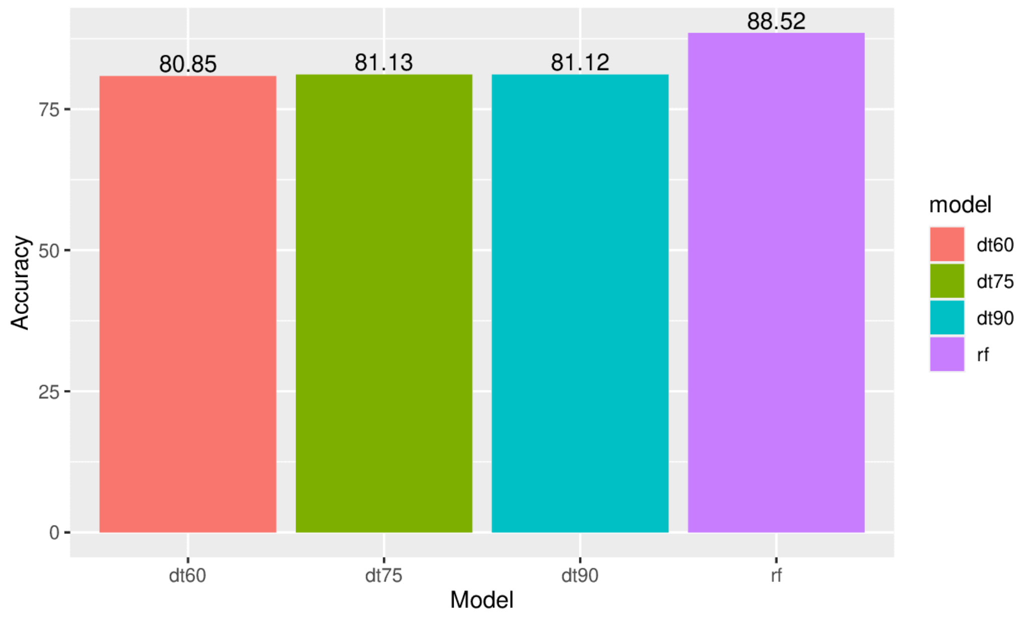


Figure 10: Accuracies based on implemented models

In the figure 11, blue points represent win, while red points represent loss. The first graph shows us the distribution of `diffDREB`, the second one is the distribution of `diffThreePpct` and the last one is the `diffPF` distribution.

It is fairly noticeable that wins really do have come with advantage in defensive rebounding, as we can clearly see that blue points are concentrated on the top. Similar conclusion can be made for three-point percentage. However, when it comes to personal fouls, points are more mixed, so this only confirms that fouls are not as important stat as previous two.

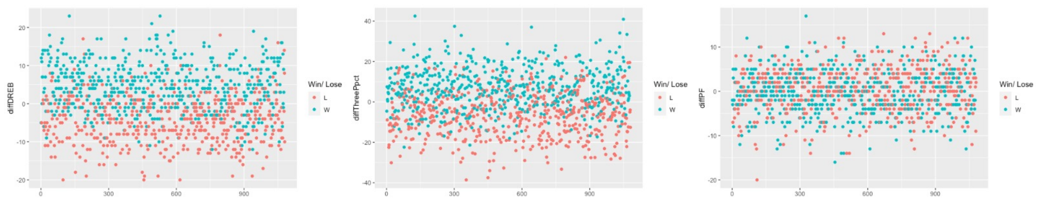


Figure 11: Dependence of `WL` stat of different stats

In the figure 12, we grouped our data by `WL`, in effort to see difference which defensive rebounding can make. Red line shows mean of number of defensive rebounds in corresponding groups.

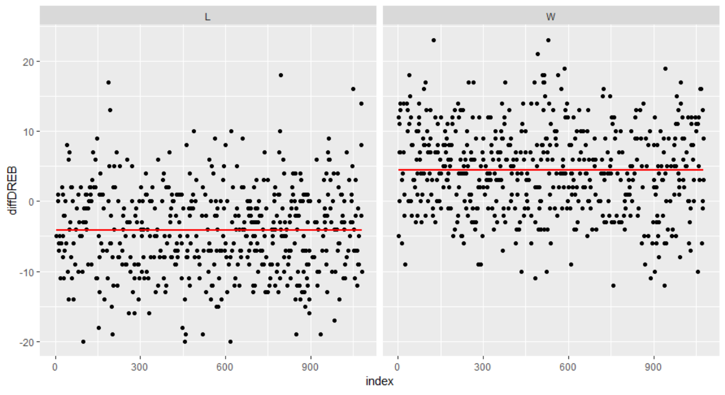


Figure 12: Means of `diffDREB` depending on `WL`

We completed our initial goal - we did check which traditional basketball stats influence winning games. Regarding future directions, we could include more advanced stats, such as deflections or screen assists. In addition, we could include many more factors, such as match up strength or home-court advantage.

A Appendix A

Variable	Importance
diffDREB	168.0170
diffThreePpct	83.2751
diffTOV	52.5824
diffAST	30.7807
diffFGA	23.7486
diffSTL	11.4795
diffFTA	7.6791
diffThreePA	6.0762
diffOREB	5.8731
diffPF	4.9288
diffFTpct	3.3042
diffBLK	1.3903

Table 1:

Variable	Importance
diffDREB	136.7103
diffThreePpct	58.1982
diffTOV	40.4822
diffAST	24.5910
diffFGA	18.5162
diffSTL	8.7566
diffFTpct	8.4895
diffFTA	6.3285
diffOREB	4.0853
diffBLK	2.6357

Table 2:

Variable	Importance
diffThreePpct	83.5401
diffDREB	81.5468
diffAST	41.1398
diffTOV	28.9495
diffFGA	23.3318
diffSTL	15.4113
diffOREB	11.9373
diffFTA	10.5650
diffFTpct	3.9873
diffBLK	3.1119
diffPF	0.9586

Table 3:

Variable	Importance
diffDREB	94.4512
diffThreePpct	79.2461
diffAST	48.0623
diffTOV	28.8107
diffFTpct	26.7751
diffFTA	22.4229
diffSTL	21.6588
diffBLK	19.4870
diffPF	17.7972
diffThreePA	16.0133
diffFGA	16.0036
diffOREB	13.6384

Table 4:

B Codes

main.R

```r
setwd("/Users/jovan.samke/Downloads")
source("./libPrimatijada22.R")

df <- cleanupDataFrame(read_xlsx("./NBA_DataSet_Version1.xlsx"))
dfSimple <- simplifyDataFrame(df)

dt90 <- modelDecisionTree(dfSimple, 0.90, 0.013, 100, 100000)
dt75 <- modelDecisionTree(dfSimple, 0.75, 0.027, 100, 100000)
dt60 <- modelDecisionTree(dfSimple, 0.60, 0.016, 100, 100000)
rf <- modelRandomForest(dfSimple, 0.75, 1000, 500)

drawAccuracies(c(dt90, dt75, dt60, rf))

drawGrid(dfSimple)

drawMeans(dfSimple)
```

libPrimatijada22.R

```r
#install.packages("dplyr")
#install.packages("ggplot2")
#install.packages("gridExtra")
#install.packages("readxl")
#install.packages("rpart")
#install.packages("rpart.plot")
#install.packages("randomForest")

library("dplyr")
library("ggplot2")
library("gridExtra")
library("readxl")
library("rpart")
library("rpart.plot")
library("randomForest")

cleanupDataFrame <- function(df){
  df <- na.omit(df)
  df <- df[-c(1:3, 5:7, 9, 10, 13, 18, 24)]
  colnames(df)[c(1, 3:4, 6)] <- c("WL", "ThreePA", "ThreePpct",
   "FTpct")

  df$WL <- as.factor(df$WL)

  df[14] <- 1:2160
  colnames(df)[14] <- "index"
  dfopp <- df[df$index%%2==0, ]
  dfteam <- df[df$index%%2!=0, ]
  colnames(dfopp) <- paste(replicate(14, "opp"), colnames(dfopp)
    , sep="")
  df <- cbind(dfteam, dfopp)
  df <- df[-c(14,15,28)]

  return(df)
}

simplifyDataFrame <- function(df){
  df2 <- data.frame(df[, 1:13])
  df2[-1] <- df2[-1] - df[14:25]
  colnames(df2) <- paste(replicate(12, "diff"), colnames(df2),
    sep="")
  colnames(df2)[1] <- "WL"

  return(df2)
}

generateTrainingData <- function(df, q, s){
  set.seed(s)
  index <- sample(1:nrow(df), size = q*nrow(df))
  trainingData <- dfSimple[index,]
  predictionData <- dfSimple[-index,]

  prop.table(table(trainingData$WL))
  prop.table(table(predictionData$WL))

  lista <- list(trainingData, predictionData);
  return(lista)
}

calculateAccuracy <- function(model, predictionData){
  results <- predict(model, predictionData, type = "class")
  return(sum(results == predictionData$WL)/nrow(predictionData))
}

modelDecisionTree <- function(df, q, cp, N, MAX){
  accuracies <- replicate(N, 0)
  for (i in 1:N){
    lista <- generateTrainingData(df, q, sample(MAX, 1))
    trainingData <- lista[[1]]
    predictionData <- lista[[2]]

    model <- rpart(WL ~ ., data = trainingData)
    #print(model$variable.importance)
    #cat("\n")
    #plotcp(model, minline=TRUE, upper = "size")
    model <- prune.rpart(model, cp)
    #rpart.plot(model)
    accuracies[i] <- calculateAccuracy(model, predictionData)
  }

  print(mean(accuracies))
  print(sd(accuracies))

  return(mean(accuracies))
}

modelRandomForest <- function(df, q, s, n){
  lista <- generateTrainingData(dfSimple, q, s)
  trainingData <- lista[[1]]
  predictionData <- lista[[2]]

  model <- randomForest(WL~., data = trainingData, ntree=n)
  model$importance
  print(model)

  return(calculateAccuracy(model, predictionData))
}

drawGrid <- function(df){
  df <- cbind(df, index = 1:length(dfSimple$WL))

  p1 <- ggplot(df, aes(x = index, y = diffDREB)) +
    geom_point(aes(col = WL)) +
    labs(x = "", colour = "Win/Lose")

  p2 <- ggplot(df, aes(x = index, y = diffThreePpct)) +
    geom_point(aes(col = WL)) +
    labs(x = "", colour = "Win/Lose")

  p3 <- ggplot(df, aes(x = index, y = diffPF)) +
    geom_point(aes(col = WL)) +
    labs(x = "", colour = "Win/Lose")

  grid.arrange(p1, p2, p3, ncol=3)
}

drawMeans <- function(df){
  df <- cbind(df, index = 1:length(dfSimple$WL))

  df <- df %>%
    group_by(WL) %>%
    mutate(meanDREB = mean(diffDREB))

  ggplot(df) +
    geom_point(aes(x = index, y = diffDREB)) +
    geom_smooth(aes(x = index, y = meanDREB), col = "red") +
    facet_wrap(~WL)
}

drawAccuracies <- function(accuracies){
  accuracies <- round(accuracies, digits = 4)*100
  model <- c("dt90", "dt75", "dt60", "rf")
  podaci <- data.frame(model, accuracies)

  ggplot(podaci, aes(x = model, y=accuracies)) +
    geom_col(aes(fill=model)) +
    geom_text(position=position_dodge(0.5), vjust=-0.25, label=
   accuracies) +
    xlab("Model") +
    ylab("Accuracy")
}
```

C Acknowledgement

We would like to thank our mentor Bojana Milošević, who encouraged us to work on this project. She also reviewed it and suggested some new ideas. Without her help, this research would not be the same.

We would also like to thank our friend Tamara Topalov for grammar checking.

References

[1] https://towardsdatascience.com/nba-data-analytics-changing-the-game-a9ad59d1f116

[2] https://bookdown.org/michael_arman7/Basketball-Analysis/

[3] https://www.dovepress.com/the-random-forest-model-has-the-best-accuracy-among-the-four-pressure–peer-reviewed-fulltext-article-RMHP

[4] https://www.nba.com/stats/teams/boxscores/?Season=2020-21&SeasonType=Regular%20Season

[5] https://cran.r-project.org/web/packages/rpart/rpart.pdf

[6] Suthaharan, S., 2016. Decision tree learning. In Machine Learning Models and Algorithms for Big Data Classification (pp. 237-269). Springer, Boston, MA.

[7] Turney, P., 1994. A theory of cross-validation error. Journal of Experimental & Theoretical Artificial Intelligence, 6(4), pp.361-391.

[8] https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

[9] https://cran.r-project.org/web/packages/randomForest/randomForest.pdf