PSTAT 131 Project

Raymond Lee

Contents

Introduction

This project attempts to create a model that predicts whether the Denver Nuggets will win or lose a regular season game based on Nikola Jokic's performance.

Background and Research Questions

As of writing this report in 2022, Nikola Jokic is the reigning back-to-back MVP in the NBA and has produced historic stats and numbers in the process. He is arguably the best player in the league and undeniably the best player on his team. Therefore, I was interested in uncovering exactly how great and valuable Jokic is. Can I identify any trends regarding his performance? Does his performance alone truly determine whether his team wins or loses? Is he as valuable as what the media and his accolades suggest? These are some of the questions that I sought to answer.

The Data

I obtained the data from https://www.basketball-reference.com/. The website records basic and advanced stats for basketball players and teams. The data is official data that are collected by SportRadar, the official statistics provider of the NBA. The data set is quite extensive, so it may be helpful to consult the codebook. The data set includes various stats that are recorded throughout a basketball game and information about the particular game played such as date, location, and opponent.

Data Cleaning

Stats for each regular season were in separate data sets. Basic and advanced stats for each season were also in separate datasets. The data was cleaned before performing the initial split:

- For each season's data sets, unnecessary or repetitive columns were removed. Column names were also modified into more legible ones. The values for the column that indicated the game's location were also changed to "Home" and "Away." The basic and advanced data sets for each season were then merged to have complete stats for each season.

```
library(dplyr)
library(tidyr)
library(stringr)

set.seed(1114)
```

```
reg1516 = read.csv("15-16.txt")
reg1516 = reg1516[-5]
reg1516[5][reg1516[5] == "@"] = "Away"
reg1516[5][reg1516[5] == ""] = "Home"
new_colnames1 = list("Location", "Result", "FG%", "3P", "3PA", "3P%", "FT%", "PM")
colnames(reg1516)[c(5, 7, 12:15, 18, 29)] = new_colnames1

adv1516 = read.csv("adv15-16.txt")
adv1516 = adv1516[-c(2:10, 23)]
new_colnames2 = list("TS%", "eFG%", "ORB%", "DRB%", "TRB%", "AST%", "STL%", "BLK%",
                     "TOV%", "USG%")
colnames(adv1516)[c(2:11)] = new_colnames2

stats1516 = merge(reg1516, adv1516)

reg1617 = read.csv("16-17.txt")
reg1617 = reg1617[-5]
reg1617[5][reg1617[5] == "@"] = "Away"
reg1617[5][reg1617[5] == ""] = "Home"
colnames(reg1617)[c(5, 7, 12:15, 18, 29)] = new_colnames1

adv1617 = read.csv("adv16-17.txt")
adv1617 = adv1617[-c(2:10, 23)]
colnames(adv1617)[c(2:11)] = new_colnames2

stats1617 = merge(reg1617, adv1617)

reg1718 = read.csv("17-18.txt")
reg1718 = reg1718[-5]
reg1718[5][reg1718[5] == "@"] = "Away"
reg1718[5][reg1718[5] == ""] = "Home"
colnames(reg1718)[c(5, 7, 12:15, 18, 29)] = new_colnames1

adv1718 = read.csv("adv17-18.txt")
adv1718 = adv1718[-c(2:10, 23)]
colnames(adv1718)[c(2:11)] = new_colnames2
```

```
stats1718 = merge(reg1718, adv1718)

reg1819 = read.csv("18-19.txt")
reg1819 = reg1819[-5]
reg1819[5][reg1819[5] == "@"] = "Away"
reg1819[5][reg1819[5] == ""] = "Home"
colnames(reg1819)[c(5, 7, 12:15, 18, 29)] = new_colnames1

adv1819 = read.csv("adv18-19.txt")
adv1819 = adv1819[-c(2:10, 23)]
colnames(adv1819)[c(2:11)] = new_colnames2

stats1819 = merge(reg1819, adv1819)

reg1920 = read.csv("19-20.txt")
reg1920 = reg1920[-5]
reg1920[5][reg1920[5] == "@"] = "Away"
reg1920[5][reg1920[5] == ""] = "Home"
colnames(reg1920)[c(5, 7, 12:15, 18, 29)] = new_colnames1

adv1920 = read.csv("adv19-20.txt")
adv1920 = adv1920[-c(2:10, 23)]
colnames(adv1920)[c(2:11)] = new_colnames2

stats1920 = merge(reg1920, adv1920)

reg2021 = read.csv("20-21.txt")
reg2021 = reg2021[-5]
reg2021[5][reg2021[5] == "@"] = "Away"
reg2021[5][reg2021[5] == ""] = "Home"
colnames(reg2021)[c(5, 7, 12:15, 18, 29)] = new_colnames1

adv2021 = read.csv("adv20-21.txt")
adv2021 = adv2021[-c(2:10, 23)]
colnames(adv2021)[c(2:11)] = new_colnames2

stats2021 = merge(reg2021, adv2021)

reg2122 = read.csv("21-22.txt")
reg2122 = reg2122[-5]
reg2122[5][reg2122[5] == "@"] = "Away"
reg2122[5][reg2122[5] == ""] = "Home"
colnames(reg2122)[c(5, 7, 12:15, 18, 29)] = new_colnames1

adv2122 = read.csv("adv21-22.txt")
adv2122 = adv2122[-c(2:10, 23)]
colnames(adv2122)[c(2:11)] = new_colnames2

stats2122 = merge(reg2122, adv2122)

stats1516 = stats1516[-1]
stats1617 = stats1617[-1]
stats1718 = stats1718[-1]
```

```
stats1819 = stats1819[-1]
stats1920 = stats1920[-1]
stats2021 = stats2021[-1]
stats2122 = stats2122[-1]
```

- The datasets for each season were then merged into one dataset.

```
stats = rbind(stats1516, stats1617, stats1718, stats1819, stats1920,
              stats2021, stats2122)
```

- Each game's result and point differential were in the same column, so the two were separated. Plus signs were also removed from column values that utilized them to make the data easier to read.

```
stats = separate(stats, Result, c("Result", "Differential"), sep = " ")
stats$Differential = gsub("[()]", "", stats$Differential)
stats$Differential = gsub("\\+", "", stats$Differential)
stats$PM = gsub("\\+", "", stats$PM)
```

- The column that indicated whether Jokic started the game was removed since it was an unnecessary detail. Columns that should be numeric were also coded to be numeric.

```
stats = stats[-8]
stats[c(1, 7, 9:41)] = lapply(stats[c(1, 7, 9:41)], as.numeric)
```

- `Date` was coded as and separated into `Year`, `Month`, `Day`. The days in `Age` were also dropped.

```
library(lubridate)

stats$Date = ymd(stats$Date)
stats$Age = gsub("-.*", "", stats$Age)
stats[3] = lapply(stats[3], as.numeric)
stats = stats %>%
  mutate(year=year(Date), month=month(Date), day=mday(Date))
```

- Games in which Jokic did not play were removed. `MP` (minutes played) was also coded as and separated into `Minutes` and `Seconds`.

```
stats = stats %>%
  filter(str_detect(MP, ":")) %>%
  mutate(Time=parse_date_time(MP, "M:S"), Minutes=minute(Time), Seconds=second(Time))
```

- Unnecessary columns, such as `Day` and `Time`, were removed once more, and the columns were reordered. Character columns were also coded as factors.

```
stats = stats[-c(2, 8, 44, 45)]
stats = stats[,c(1, 40:41, 2:4, 42:43, 5:39)]

stats$Location = as.factor(stats$Location)
stats$Opp = as.factor(stats$Opp)
stats$Result = as.factor(stats$Result)
```

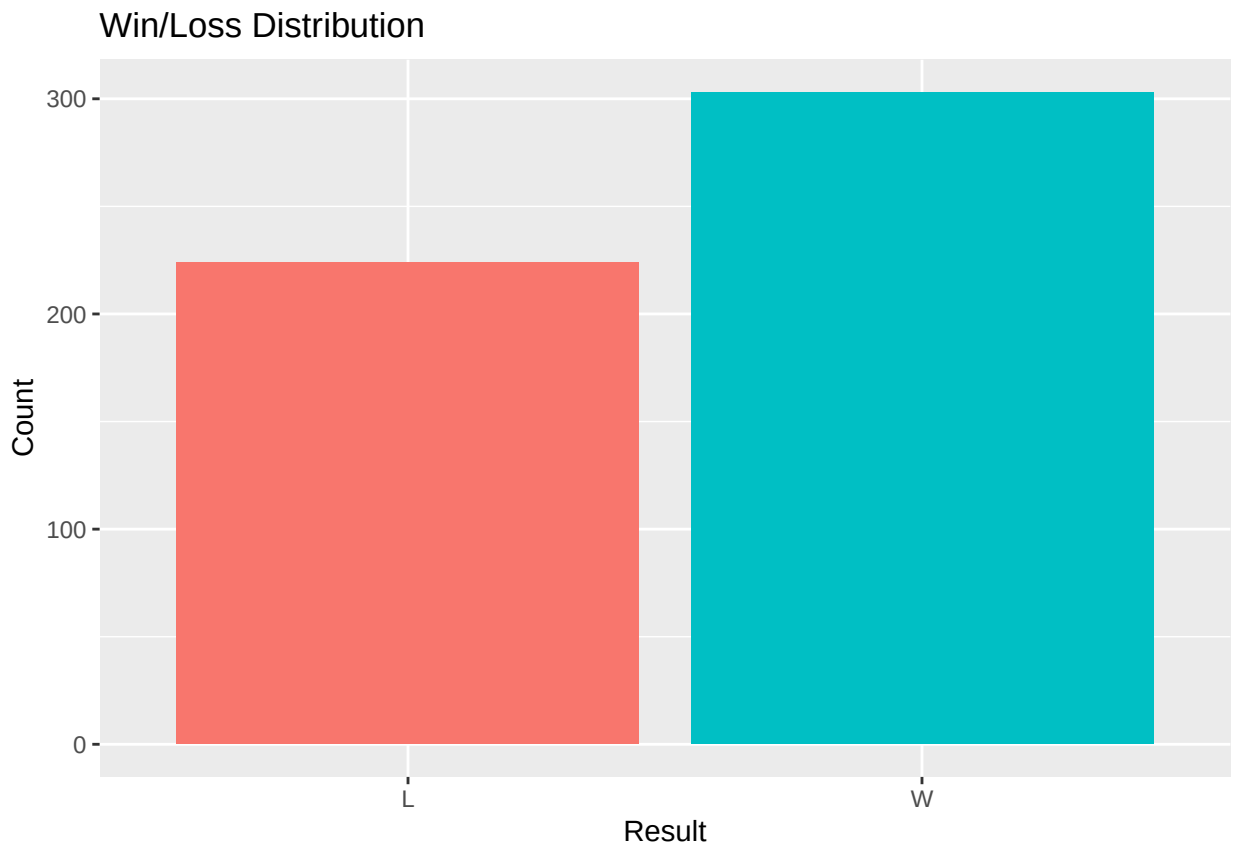
- The column names were then cleaned. Also, separate data set that preserves the `Differential` variable was created to aid with EDA in the next section. Then, it was dropped from the original data set.

```
library(janitor)
stats = stats %>%
  clean_names()

names(stats)[14] = 'three_pts'
names(stats)[15] = 'three_pa'
names(stats)[16] = 'three_pts_percent'
names(stats)[32] = 'efg_percent'
names(stats)[41] = 'ortg'
names(stats)[42] = 'drtg'

stats_with_diff = stats
stats = stats[-10]

write.csv(stats, "C:/Users/Raymond/Desktop/PSTAT 131 Project/PSTAT 131 Project/data/stats.csv")
```

Exploratory Data Analysis

Win/Loss Distribution

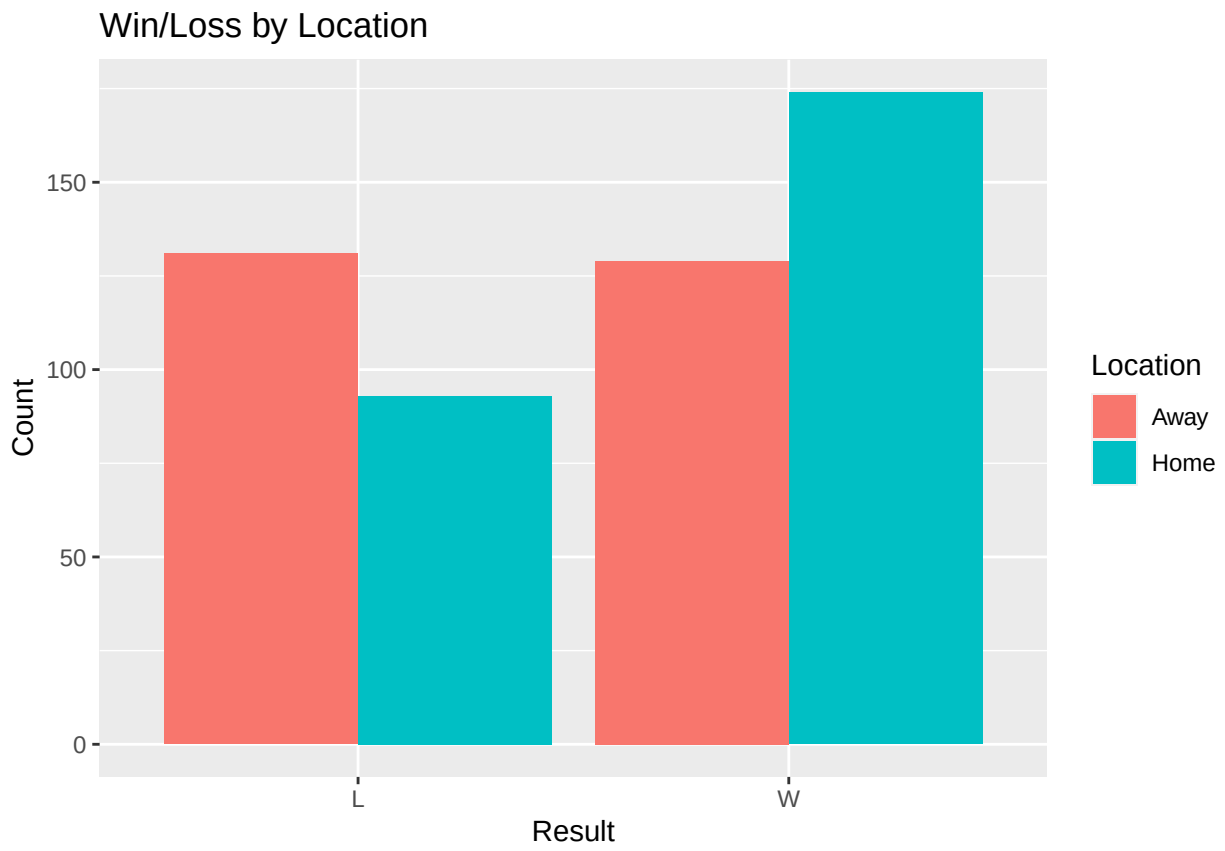
```
library(tidymodels)
library(tidyverse)
tidymodels_prefer()

ggplot(stats_with_diff, aes(result, fill=result)) + geom_bar() + ggtitle('Win/Loss Distribution') + xlab
  theme(legend.position='none')
```



This graphic displays the distribution of wins vs. losses in the data set. As we can see, Jokic and the Nuggets have more wins than losses for games that Jokic has played in. This suggests that Jokic is a winning player, which is expected of an MVP.

Win/Loss Grouped by Location

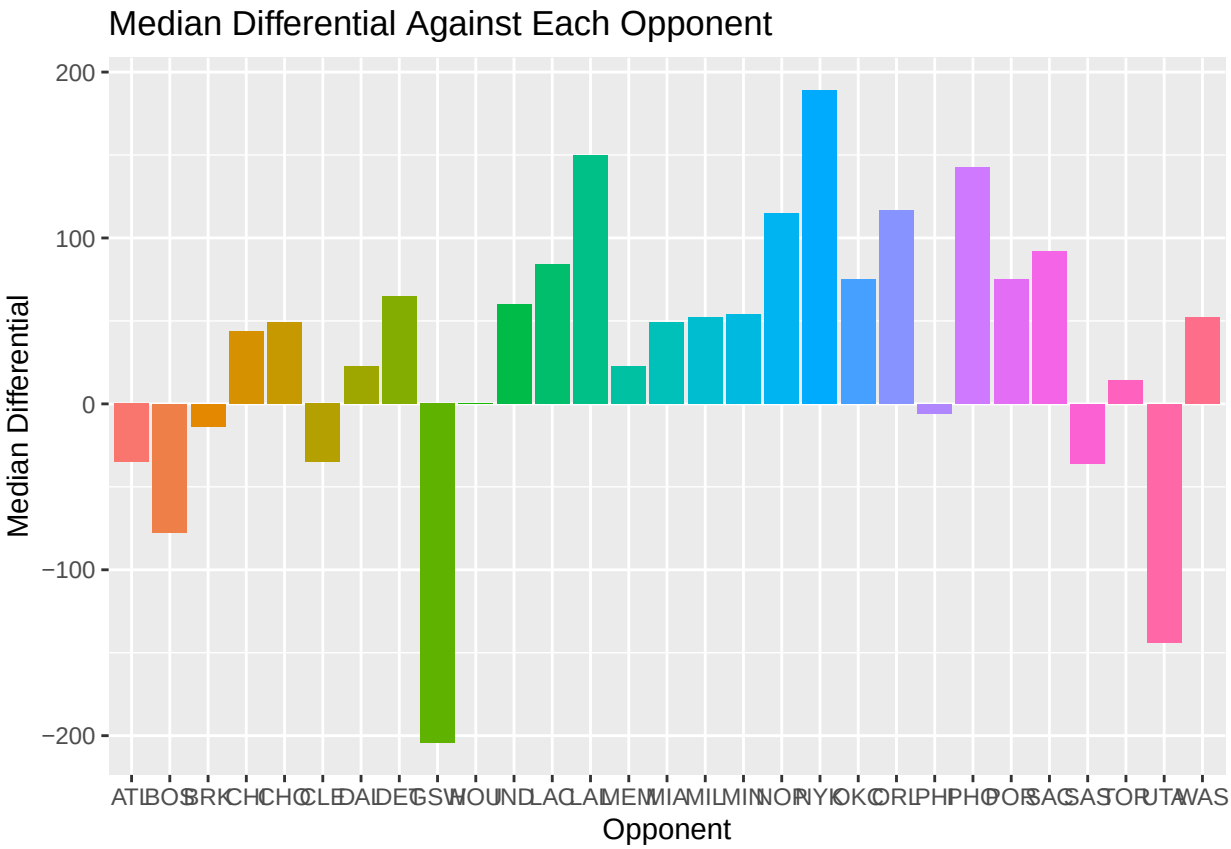
```
ggplot(stats_with_diff, aes(result, fill=location)) + geom_bar(position='dodge') + ggtitle('Win/Loss by
  xlab('Result') + ylab('Count') + guides(fill=guide_legend(title='Location'))
```



This graphic shows the win/loss distribution by the location of the game. We can see that Jokic and the Nuggets have won more games at home but have lost more games at away. This may be a sign of the existence of home court advantage.

Median Differential Grouped by Opponent

```
stats_with_diff %>%
  group_by(opp) %>%
  mutate(median_diff=median(differential)) %>%
  ungroup() %>%
  ggplot(aes(x=opp, y=median_diff, fill=opp)) + geom_bar(stat='identity') +
  ggtitle('Median Differential Against Each Opponent') + xlab('Opponent') + ylab('Median Differential')
  theme(legend.position='none')
```

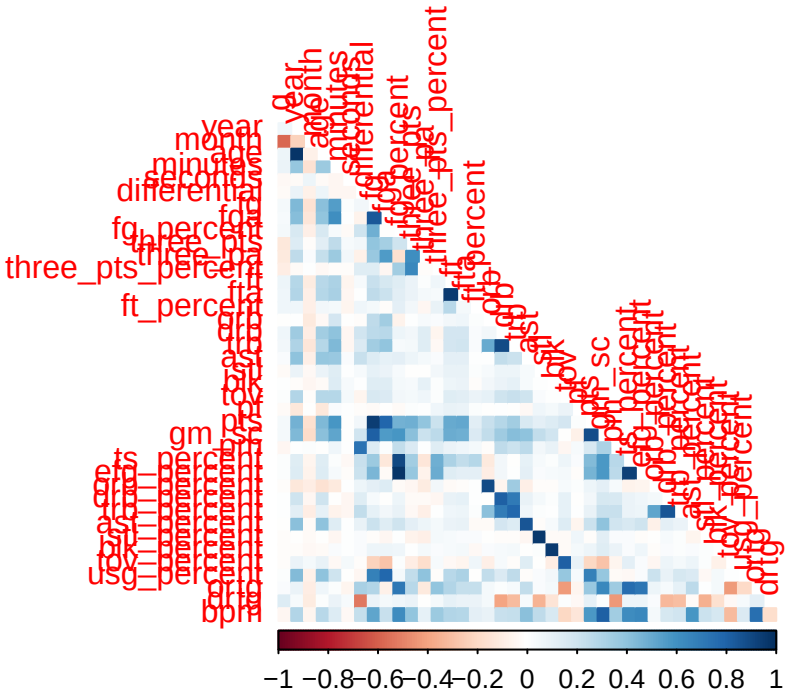


This graphic displays the median `differential` at the end of a game against each opponent. A positive `differential` would mean a win while a negative `differential` would mean a loss. I decided to use the median to prevent any influence from one-sided differentials. We can see that Jokic and the Nuggets win with the greatest point disparity against the New York Knicks. However, they lose with the greatest point disparity against the Golden State Warriors.

Correlation Between Stats

A correlation matrix was created for the numeric variables. A separate data set with no missing values was created to accomplish this. 128 observations were omitted through this process, so this may have a slight negative impact on the accuracy of this analysis.

```
library(corrplot)

eda_stats = na.omit(stats_with_diff)

eda_stats %>% select(where(is.numeric)) %>%
  cor() %>%
  corrplot(type = 'lower', diag = FALSE, method = 'color')
```



`minutes` are strongly positively correlated with `fg` (field goals), `fga` (field goal attempts), and `pts` (points). This is intuitive because more time on the court would allow for more shots taken and points scored. `minutes` are also strongly positively correlated with `gm_sc` (Game Score), suggesting that the more that Jokic plays, the more productive he ends up being.

`year` is strongly positively correlated with `fg`, `fga`, `trb` (total rebounds), `ast` (assists), `pts`, `gm_sc`, `ast_percent` (assist percentage), and `usg_percent` (usage percentage). This suggests that Jokic improves offensively as time goes on. He is also more involved in team plays and provides more assists as time goes on.

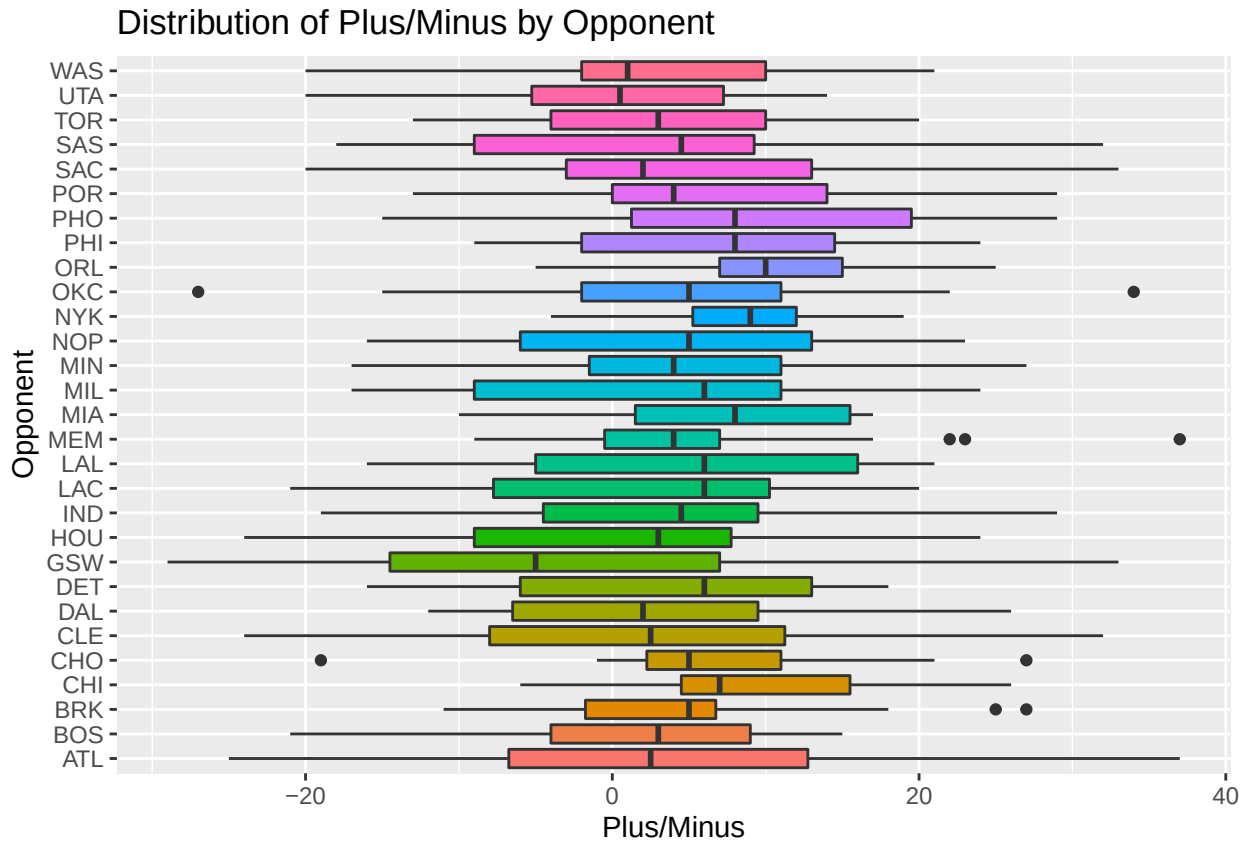
`usg_percent` is also strongly positively correlated with `fg`, `fga`, and `pts`. As he becomes more involved in his team's plays, he attempts and makes more shots. However, `usg_percent` is also moderately correlated with `tov` (turnovers), which is intuitive because he simply would have a higher chance of losing possession of the ball if he has it in his hands more.

`ortg` (offensive rating) is moderately negatively correlated with `tov` and `tov_percent` (turnover percentage). `drtg` (defensive rating) is strongly negatively correlated with `differential` and moderately negatively correlated with `drb` (defensive rebounds), `trb`, `stl` (steals), and their respective percentages. These make sense because more turnovers would lead to less points produced, and more rebounds and steals would lead to less points allowed.

Finally, `differential` is slightly positively correlated with `ast`, `ts_percent` (true shooting percentage), `efg_percent` (effective field goal percentage), and `trb_percent` (total rebound percentage). Therefore, it may be possible that when Jokic has more assists, has better overall shooting numbers, and grabs more rebounds that are available, he and his team are more likely to win. What is interesting is that despite Jokic being praised as one of the best passers in the NBA today (and most likely in NBA history by the time he retires), `ast` and `ast_percent` are not strongly positively correlated with `differential`.

Distribution of Plus/Minus Grouped by Opponent

```
stats_with_diff %>%
  ggplot(aes(x=pm, y=opp, fill=opp)) + geom_boxplot() +
  ggtitle('Distribution of Plus/Minus by Opponent') + xlab('Plus/Minus') + ylab('Opponent') +
  theme(legend.position='none')
```



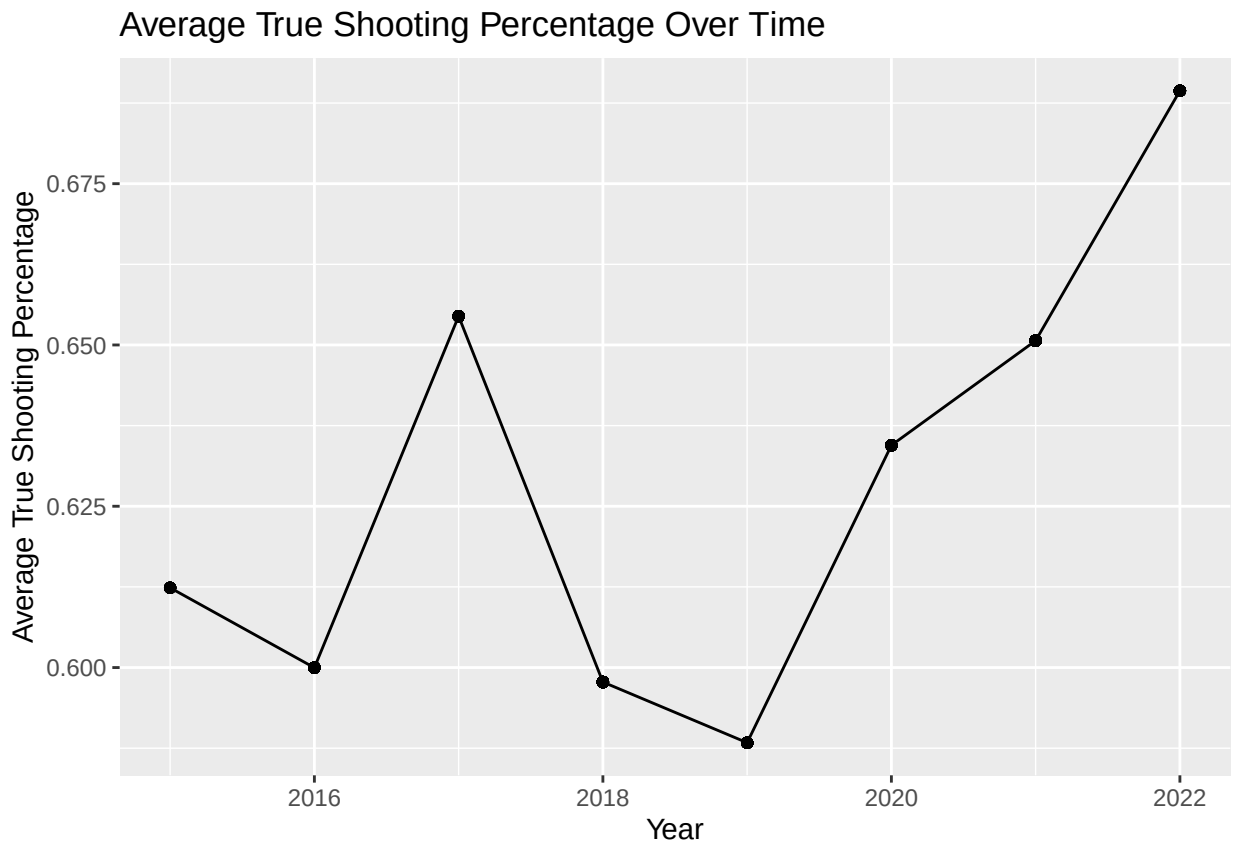
This plot displays the distribution of Jokic's `pm` against each opponent. `pm` (plus/minus) is not the greatest stat to use in determining a player's impact because it does not consider the impact of other players on the court, but it is still useful in gaining a rudimentary understanding of a player's value. At a glance, this graphic shows that the spread for the distribution of Jokic's `pm` is greatest when he plays against the Atlanta Hawks, the Cleveland Cavaliers, and Golden State Warriors. In other words, he plays the most inconsistently against these teams compared to other teams. Jokic has his highest median plus/minus when he plays against the Orlando Magic and has his lowest median `pm` when he plays against the Golden State Warriors. In other words, his team's typical lead when he is playing is greatest against the Orlando Magic but lowest against the Golden State Warriors.

Average True Shooting Percentage Over Time

The data set without missing values was used for this section. Again, this may have a slight negative impact on the accuracy of this analysis.

```
eda_stats %>%
  group_by(year) %>%
```

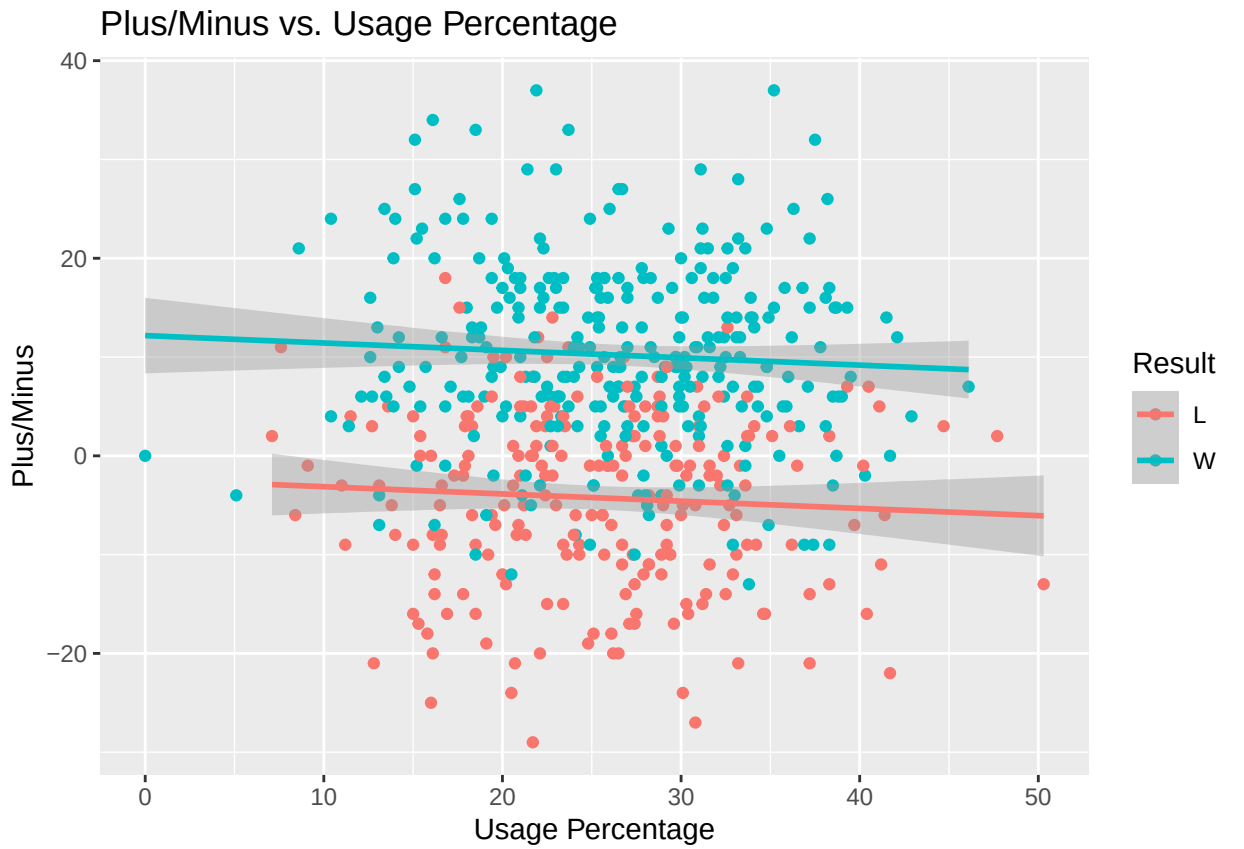
```
  mutate(avg_ts_percent=mean(ts_percent)) %>%
  ungroup() %>%
  ggplot(aes(x=year, y=avg_ts_percent)) + geom_point(stat='identity') + geom_line() +
  ggtitle('Average True Shooting Percentage Over Time') + xlab('Year') + ylab('Average True Shooting Pe
```



This graphic shows that Jokic's average scoring efficiency (considering 2-pointers, 3-pointers, and free throws) has generally been improving. Even during the visible decline from 2017 to 2019, his average `ts_percent` was still above 0.5875. This means that he was successful in over half of all of his scoring attempts! In particular, we can see that Jokic's average `ts_percent` was over a ridiculously elite percentage of 65% in 2021 and 2022, which is when he won his two MVP awards. Therefore, there has generally been an upward trend during his career so far regarding his performance.

Scatterplot of Plus/Minus vs. Usage Percentage

```
stats_with_diff %>%
  ggplot(aes(x=usg_percent, y=pm, color=result)) + geom_point() + geom_smooth(method='lm') +
  ggtitle('Plus/Minus vs. Usage Percentage') + xlab('Usage Percentage') + ylab('Plus/Minus') +
  guides(color=guide_legend(title='Result'))
```



This graphic plots Jokic's `pm` against his `usg_percent` for each game and displays a trend line for the points. It reveals that as Jokic's `usg_percent` increases, his `pm` tends to drop slightly regardless of the game's result. That is, as Jokic becomes more involved his team's plays, the lead that his team has while he is playing decreases slightly. Therefore, there seems to be a point of diminishing returns on scoring as his team relies on him more.

Data Splitting and Cross Validation

The data were split into a 70% training and 30% test split. Stratified sampling was performed due to the distribution of `result` being skewed. The training set consists of 368 observations whereas the test set consists of 159 observations.

```
stats_split = stats %>%
  initial_split(prop=.7, strata='result')

stats_train = training(stats_split)
stats_test = testing(stats_split)
```

The training set was then folded into 10 folds for cross-validation. Cross-validation is when we split the training set into folds to use in assessing model performance and selecting the best model. This way, less data are used to figure out the best model, allowing us to set aside enough data for the test set.

```
stats_folds = vfold_cv(stats_train, v=10)
```

Model Fitting

Creating the Recipe

Before fitting any models, there was a problem with the data set that had to be addressed: missing values. Certain predictor values were missing due to how Jokic plays in a game. For example, if he does not shoot a free throw at all (0/0), the value for free throw percentage would be missing. Unfortunately, removing all observations with missing values would result in the exclusion of a significant portion of the data. We would also lose all other information besides the missing values in each observation.

I decided to impute the missing values using the medians of the respective predictors. Therefore, readers should understand that any conclusions made in this project involve the following caveat: missing values for any predictor were replaced with the median for that particular predictor. This may either inflate or deflate Jokic's overall performance in each game with missing values.

I created a recipe used to fit the models after identifying which predictors had missing values.

```
# Predictors with missing values
colnames(stats)[colSums(is.na(stats)) > 0]
```

```
## [1] "fg_percent"        "three_pts_percent" "ft_percent"
## [4] "ts_percent"        "efg_percent"       "tov_percent"
```

```
recipe = recipe(result ~., data = stats_train) %>%
  step_impute_median(fg_percent, three_pts_percent, ft_percent, ts_percent, efg_percent, tov_percent) %
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
```

Logistic Regression Model

I first fit a logistic regression model to the training data. As an aside, I got a warning message describing that some predicted probabilities were extremely close to 0 or 1.

```
log_reg = logistic_reg() %>%
  set_engine('glm') %>%
  set_mode('classification')

log_wf = workflow() %>%
  add_model(log_reg) %>%
  add_recipe(recipe)

control = control_resamples(save_pred = TRUE)

log_fit = fit_resamples(log_wf, stats_folds, control=control)
```

Support Vector Machines

I then fit a linear support vector machines (SVM) model.

```
svm_spec = svm_poly(degree = 1) %>%
  set_mode('classification') %>%
  set_engine('kernlab', scaled = FALSE)
```

I then set up a tuning grid while setting `levels = 10`.

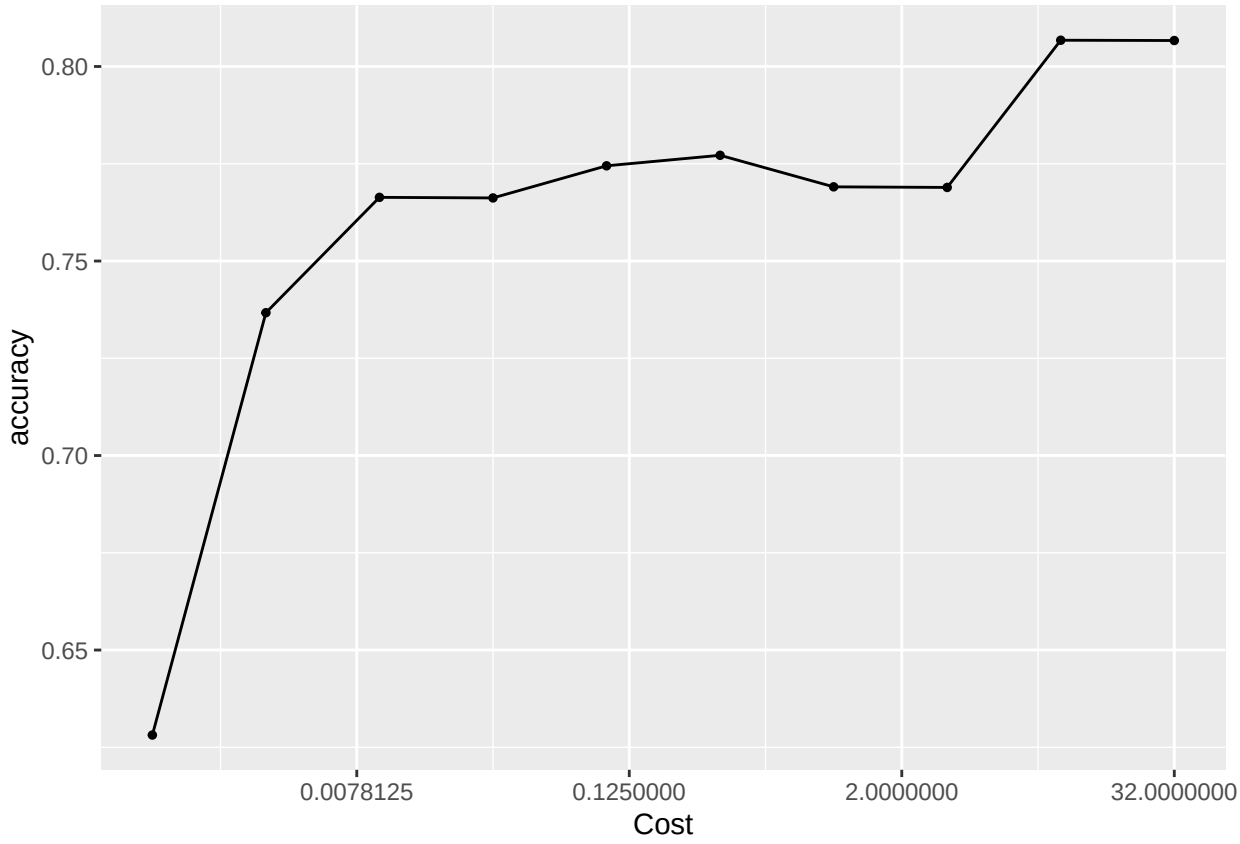
```
svm_grid = grid_regular(cost(), levels = 10)
```

Next, I created a workflow while tuning `cost` and then tuned the model.

```
svm_wf <- workflow() %>%
  add_model(svm_spec %>% set_args(cost = tune())) %>%
  add_recipe(recipe)

# svm_tune_res <- tune_grid(svm_wf, resamples = stats_folds, grid = svm_grid, metrics=metric_set(accura
load('svm.rda')
```

I also printed an autoplot() of the results. It appears that accuracy is highest when `cost` is 10.07936840.

```
autoplot(svm_tune_res)
```




```
show_best(svm_tune_res)
```

```
## # A tibble: 5 x 7
##      cost .metric  .estimator  mean     n std_err .config
##     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 10.1    accuracy binary     0.807    10  0.0215 Preprocessor1_Model09
## 2 32      accuracy binary     0.807    10  0.0227 Preprocessor1_Model10
```

```
## 3  0.315  accuracy binary     0.777    10  0.0209 Preprocessor1_Model06
## 4  0.0992 accuracy binary     0.774    10  0.0238 Preprocessor1_Model05
## 5  1      accuracy binary     0.769    10  0.0243 Preprocessor1_Model07
```

Finally, I fit the model and selected the best performing SVM model.

```
best_svm = select_best(svm_tune_res, metric='accuracy')

svm_final = finalize_workflow(svm_wf, best_svm)

svm_final_fit = fit(svm_final, data = stats_train)
```

Random Forest Model

I then fit a random forest model. I set the model while tuning `mtry`, `trees`, and `min_n`.

```
rf_spec = rand_forest(mtry=tune(), trees=tune(), min_n=tune()) %>%
  set_engine('ranger', importance='impurity') %>%
  set_mode('classification')
```

I then set up a tuning grid while setting `mtry` to a range from 1 to 6, `trees` to range from 100 to 500, `min_n` to a range from 1 to 20, and `levels = 10`.

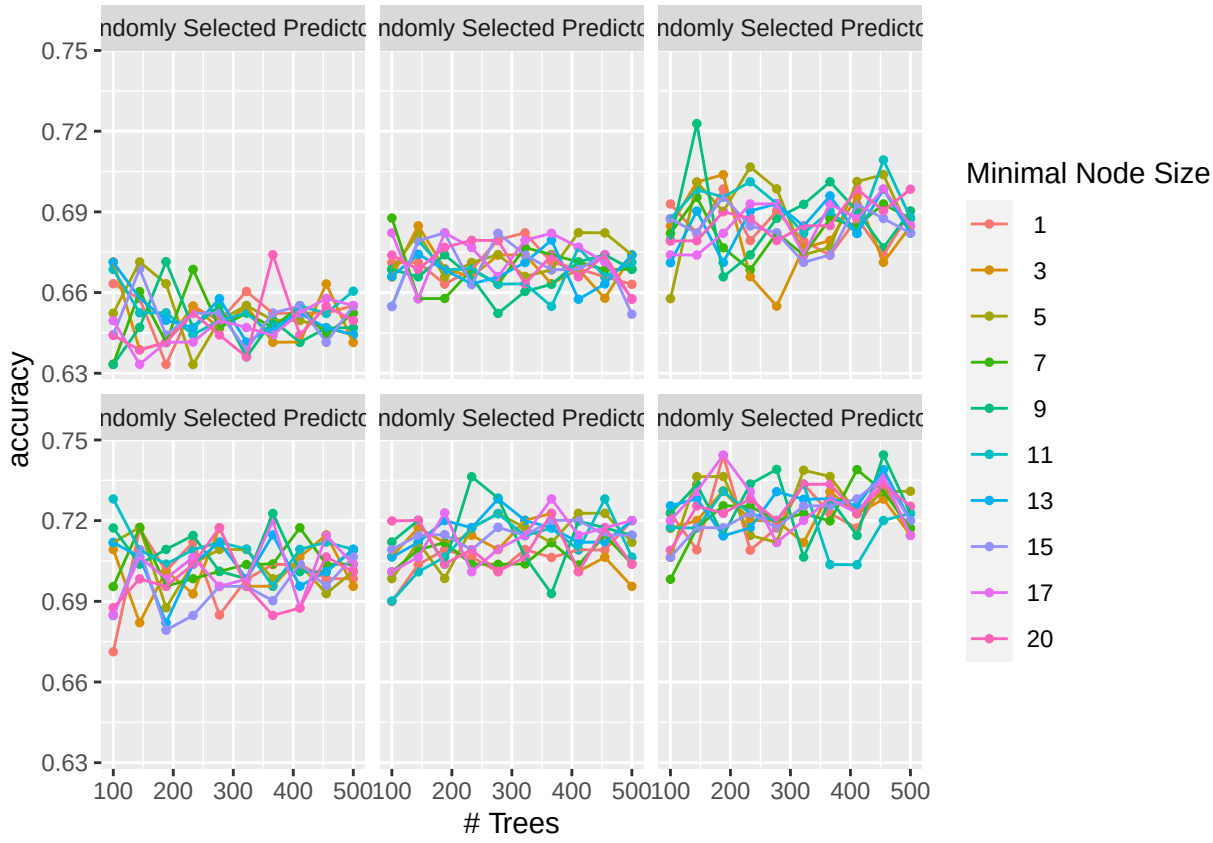
```
rf_grid = grid_regular(mtry(range=c(1, 6)), trees(range=c(100, 500)), min_n(range=c(1, 20)), levels=10)
```

I then created a workflow for the random forest model and tuned the model.

```
rf_wf = workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(recipe)

# rf_tune_res = tune_grid(rf_wf, resamples=stats_folds, grid=rf_grid, metrics=metric_set(accuracy))
load('rf.rda')
```

I also printed an autoplot() of the results. It appears that accuracy is highest when `mtry` is 6, `trees` is 455, and `min_n` is 9.

```
autoplot(rf_tune_res)
```




```
show_best(rf_tune_res)
```

```
## # A tibble: 5 x 9
##    mtry trees min_n .metric  .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     6   455     9 accuracy binary     0.744    10  0.0231 Preprocessor1_Model~
## 2     6   188    17 accuracy binary     0.744    10  0.0217 Preprocessor1_Model~
## 3     6   188     1 accuracy binary     0.744    10  0.0288 Preprocessor1_Model~
## 4     6   277     9 accuracy binary     0.739    10  0.0237 Preprocessor1_Model~
## 5     6   411     7 accuracy binary     0.739    10  0.0257 Preprocessor1_Model~
```

Finally, I fit the model and selected the best performing random forest model.

```
best_rf = select_best(rf_tune_res, metric='accuracy')

rf_final = finalize_workflow(rf_wf, best_rf)

rf_final_fit = fit(rf_final, data = stats_train)
```

Boosted Trees Model

I also fit a boosted trees model. I set the model while tuning `trees` and `learn_rate`.

```
boost_spec <- boost_tree(trees=tune(), learn_rate=tune()) %>%
  set_engine('xgboost') %>%
  set_mode('classification')
```

I then set up a tuning grid while setting `trees` to a range from 10 to 2000, `learn_rate` to a range from -5 to 5, and `levels = 10`.

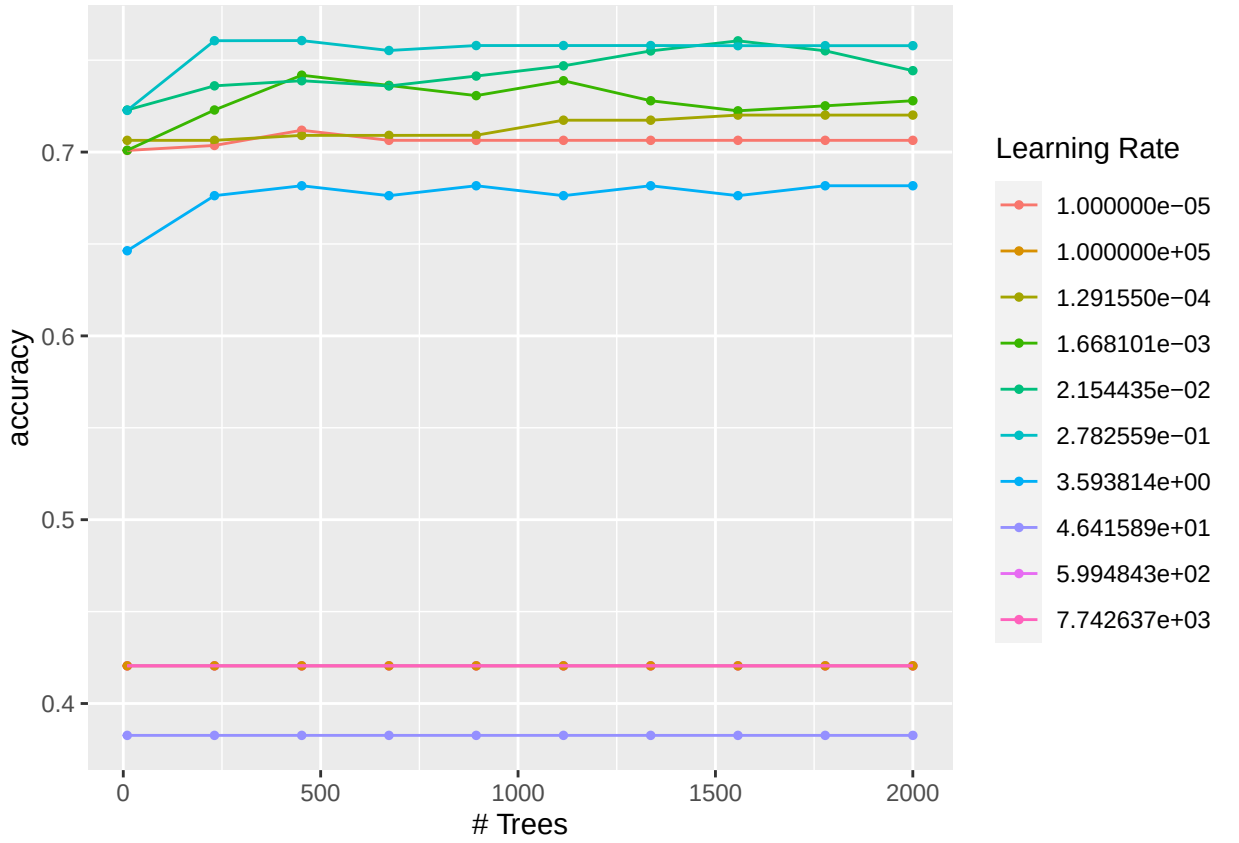
```
boost_grid = grid_regular(trees(range = c(10, 2000)), learn_rate(range = c(-5, 5)), levels = 10)
```

I then created a workflow for the boosted trees model and tuned the model.

```
boost_wf = workflow() %>%
  add_model(boost_spec) %>%
  add_recipe(recipe)

# boost_tune_res = tune_grid(boost_wf, resamples=stats_folds, grid=boost_grid, metrics=metric_set(accur
load('boost.rda')
```

I also printed an autoplot() of the results. It appears that accuracy is highest when `trees` is 452 and `learn_rate` is 0.27825594.

```
autoplot(boost_tune_res)
```



```
show_best(boost_tune_res)
```

```
## # A tibble: 5 x 8
##   trees learn_rate .metric  .estimator  mean     n std_err .config
##   <int>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1   452     0.278  accuracy binary     0.761    10  0.0221 Preprocessor1_Model0~
## 2   231     0.278  accuracy binary     0.761    10  0.0225 Preprocessor1_Model0~
## 3  1557     0.0215 accuracy binary     0.761    10  0.0233 Preprocessor1_Model0~
## 4   894     0.278  accuracy binary     0.758    10  0.0216 Preprocessor1_Model0~
## 5  1115     0.278  accuracy binary     0.758    10  0.0216 Preprocessor1_Model0~
```

Finally, I fit the model and selected the best performing boosted trees model.

```
best_boost = select_best(boost_tune_res, metric='accuracy')

boost_final = finalize_workflow(boost_wf, best_boost)

boost_final_fit = fit(boost_final, data = stats_train)
```

Model Performance

Selecting the Best Model

To compare model performances, I created a table that displays the accuracy rates for all four models using the training set and then using the test set. Accuracy is defined as the average number of correct predictions that the model made.

```
log_train_acc = augment(log_fit) %>%
  accuracy(truth = result, estimate = .pred_class)

svm_train_acc = augment(svm_final_fit, new_data = stats_train) %>%
  accuracy(truth = result, estimate = .pred_class)

rf_train_acc = augment(rf_final_fit, new_data = stats_train) %>%
  accuracy(truth = result, estimate = .pred_class)

boost_train_acc = augment(boost_final_fit, new_data = stats_train) %>%
  accuracy(truth = result, estimate = .pred_class)

accuracies <- c(log_train_acc$.estimate, svm_train_acc$.estimate,
                rf_train_acc$.estimate, boost_train_acc$.estimate)
models <- c("Logistic Regression", "SVM", "Random Forest", "Boosted Trees")
results <- tibble(training_accuracy = accuracies, models = models)
results %>%
  arrange(-accuracies)
```

```
## # A tibble: 4 x 2
##   training_accuracy models
##               <dbl> <chr>
## 1             1     Random Forest
```

```
## 2             1     Boosted Trees
## 3             0.932 SVM
## 4             0.802 Logistic Regression
```

```
log_test_recipe = recipe(result ~., data = stats_test) %>%
  step_impute_median(fg_percent, three_pts_percent, ft_percent, ts_percent, efg_percent, tov_percent) %
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())

log_test_wf = workflow() %>%
  add_model(log_reg) %>%
  add_recipe(log_test_recipe)

control = control_resamples(save_pred = TRUE)

log_test_fit = fit_resamples(log_test_wf, stats_folds, control=control)

log_test_acc = augment(log_test_fit) %>%
  accuracy(truth = result, estimate = .pred_class)

svm_test_acc = augment(svm_final_fit, new_data = stats_test) %>%
  accuracy(truth = result, estimate = .pred_class)

rf_test_acc = augment(rf_final_fit, new_data = stats_test) %>%
  accuracy(truth = result, estimate = .pred_class)

boost_test_acc = augment(boost_final_fit, new_data = stats_test) %>%
  accuracy(truth = result, estimate = .pred_class)

accuracies <- c(log_test_acc$.estimate, svm_test_acc$.estimate,
                rf_test_acc$.estimate, boost_test_acc$.estimate)
models <- c("Logistic Regression", "SVM", "Random Forest", "Boosted Trees")
results <- tibble(test_accuracy = accuracies, models = models)
results %>%
  arrange(-accuracies)
```

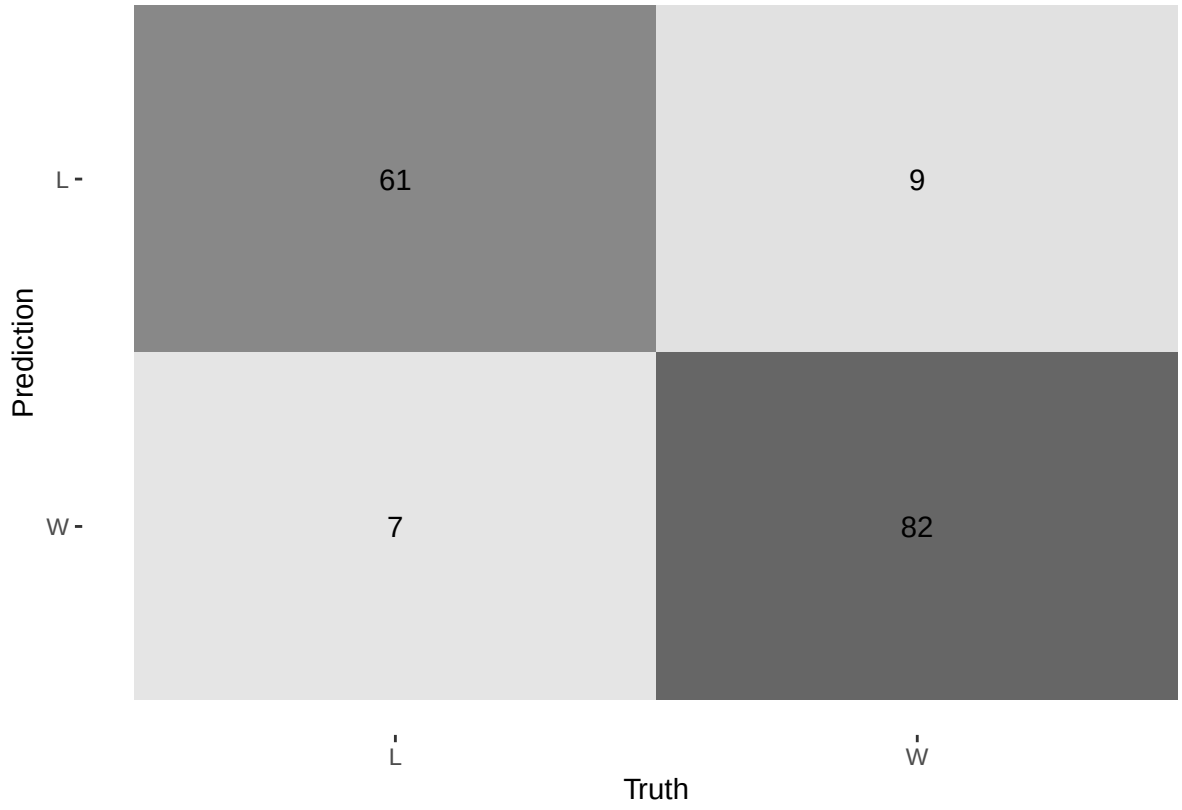
```
## # A tibble: 4 x 2
##   test_accuracy models
##           <dbl> <chr>
## 1         0.899 SVM
## 2         0.836 Boosted Trees
## 3         0.811 Random Forest
## 4         0.802 Logistic Regression
```

As we can see, the training accuracy for the random forest and boosted trees model came out to be 1, which unfortunately indicates that the models overfitted to the training set. I determined that the SVM model was the best performing model because it had the highest accuracy for the test set. However, because the accuracy for the test set was greater than that for the training set, I concluded that the model did indeed overfit to the training set.

Analysis of the Best Model and Test Set

I created a confusion matrix for the SVM model on the test set.

```
predicted_data = augment(svm_final_fit, new_data = stats_test) %>%
  select(result, starts_with('.pred'))

predicted_data %>% conf_mat(truth = result, estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```



As we can see, the model does a great job with correctly predicting the result of the game. The model correctly predicted about 90.1% of wins and about 89.7% of losses.

Conclusion

In summary, I fit a logistic regression, support vector machines, random forest, and boosted trees model to Nikola Jokic's stats from 2015 to 2022 to predict the outcome of a game. I concluded that the SVM model performed the best among the four models and that it predicts `result` extremely well. Therefore, I have found that it may very well be possible to predict whether the Denver Nuggets will win a game solely based on Jokic's performance. This demonstrates that Jokic is immensely valuable to his team. Also, while performing this analysis, I have learned that Jokic's stats are indeed well-deserving of back-to-back MVP awards. He has demonstrated elite efficiency and a winning performance against opposing teams throughout his career.

However, there are some caveats to address. First, as mentioned earlier, observations with missing values were omitted from the data set to aid with EDA. Missing values for variables were also imputed with their respective median values. This may have exaggerated (or even underestimated) some findings. In addition, because games in which Jokic did not play were also omitted, I ended up analyzing his performance in games played while excluding the effect of his presence or absence in games. Second, signs of overfitting to the

training set were observed, meaning that a significant portion of the noise in the training set was captured by the model. This may have had negative effects on the model's performance. Finally, the data set I used does not include some other useful and interesting player stats such as player efficiency rating (PER) and data regarding shot charts. PER is a per-minute rating of a player's performance that adds up all of a player's positive stats and subtracts all of the negative stats. It allows us to compare players despite a difference in minutes played between them. Shot charts provide detailed information on the shot that was attempted by a player. This includes information such as type of shot, area, and distance. Analyzing these variables in addition to the ones in this project may have provided more insight into Jokic's performance and value.

It is worth noting that the model may have been able to perform extremely well due to Jokic's status as the best and most influential player on his team. The model might have more difficulty predicting `result` based on the performance of a player who does not have as big of a role on the team.

It is also worth noting that although I concluded that the SVM model performed the best, the accuracy rates for the other models are also quite high. Since all of the models performed quite well, this may be evidence that it is indeed possible and effective to predict the result of a game based on Jokic's performance. In fact, this may also imply that it is possible to predict `result` for any team based on its best player.

Future analyses or projects should attempt to compare Jokic's stats to those of other players who have won MVP. This would provide insight into exactly how well Jokic is performing and how he compares to the greatest players of all time. It may also be interesting to fit a model that predicts `result` for a playoff game and compare its performance to the current one which deals with regular season games. Playoff games tend to be more competitive and volatile, so it might be more difficult to predict the `result`. Finally, it may be interesting for a future analysis to examine which variables contribute the most to a win.

I would like to thank Dr. Katie Coburn and Hanmo Li of the University of California, Santa Barbara for their instruction and guidance throughout this project.