Project Report: NFL Rushing Data

Jack Francis

November 2019

1 Introduction

The objective of my project is to predict the success of running plays in the NFL. In the NFL, rushing plays are a fundamental part of the game, accounting for about 40% of all offensive plays. In a standard rushing play, the quarterback will hand off the ball to the running back, who will then try to run without being tackled by the defense. Thus, the skill of the running back and defense are critical in determining the success of the run. In addition, there are other factors that can affect the success of the run, such as offensive/defensive formation, weather, and game scenario.

2 Data Description

The dataset for my project is 2 years of NFL rushing plays provided by the NFL and Next Gen Stats.

2.1 Input/Output Variables

The dataset I used included relevant information for all of these important factors. For each rushing play, the location of each of the 22 players on the field at the time the ball is handed off is given. In addition, the speed, acceleration, and direction of each player is recorded by comparing the location of each player at the time the ball is snapped and the time the ball is handed off to the running back. Each player has a unique NFLID, which allows for identifying each player by a numeric ID. Next, the dataset provides information about the current game scenario. This includes the location of the start of the play, the time remaining on the clock, the down and distance needed to get a first down, and the current game score. Next, the formation of both the offense and defense is given along with the number of "defenders in the box". The defenders in the box refers to the number of defensive players that are near to the line of scrimmage and likely to be part of the defense's effort to stop the run. An example is shown in Figure 1. Next, information about the player, including height, weight, college attendance, and age is given. For each play, the weather information and location of the game is also given. Finally, the dataset contains the number of yards that were gained (or lost) on the play.

2.2 Data Format

For the dataset, there are many continuous and categorical variables. Some of the continuous variables include the location information of each player, the current down and distance, the current score, number of defenders in the box, and the number of yards the rushing play gained or lost. Categorical variables include the positions of each player, the offensive and defensive formations, the stadium type, weather conditions, and the turf. Most columns are numeric data, however some are text data (team name, player name, offensive/defensive formation). Additionally, for this dataset, each rushing play is captured by 22 rows of data, because one row corresponds to one player on one rushing play. Thus, although there are 509762 rows in total, only 23,171 rushing plays are in the dataset, since each play is replicated 22 times for each player on the field.

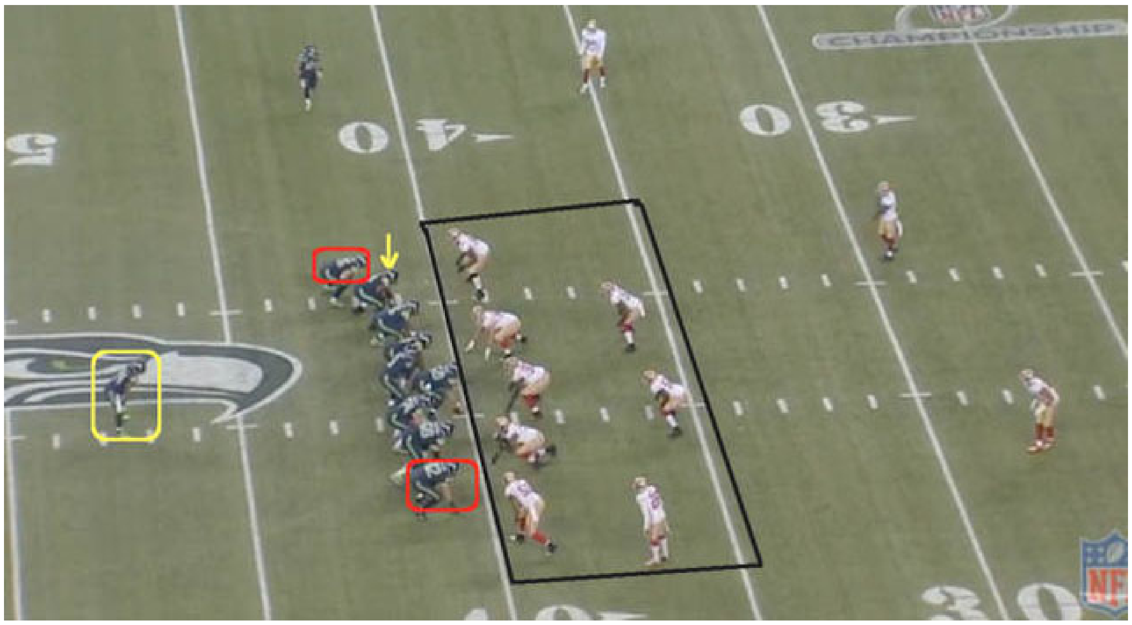


Figure 1: The defenders in the box are captured in the black rectangle. The running back is shown in the yellow rectangle.

2.3 Missing Data

Overall, there is little data missing from the entire dataset. For most of the columns, only 1-3 rushing plays (corresponding to 22-66 rows) have missing data. However, the columns regarding weather have a significant amount of missing data. This is because games played indoors (i.e. in dome stadiums) do not record the temperature, wind direction, or wind speed, since these are artificially controlled in the dome stadium. However, out of these three factors the only one that is likely to be significant is the temperature, which for indoor stadiums is usually set to 70 °F, so the missing data can be filled in accurately.

3 Data Visualization

3.1 Distribution of Yards Gained/Lost

Understanding the distribution of yards gained/lost for all rushing plays is important in determining what type of modeling approach to use. The smoothed distribution is shown in Figure 2. Most rushing plays gain between 0-5 yards, but there are significant outliers making the range extremely large (-14, 99).

3.2 Distribution of Defenders in the Box

The number of defenders in the box is important in predicting how well the defense will perform at stopping the run. The most common defensive formations include 3-3-5, 3-4-4, 4-3-4, and 4-4-3, where the sum of the first two numbers corresponds to the number of defenders in the box. The prevalence of these common schemes is shown when plotting a bar chart of number of defenders in the box, shown in Figure 3. Plays with less than 6 defenders in the box are typically scenarios where the offense is expected to pass. Plays with more than 8 defenders in the box are typically seen when the offense is close to the goal line. In these scenarios, the defense is focused on stopping the run, because it is likely to occur.

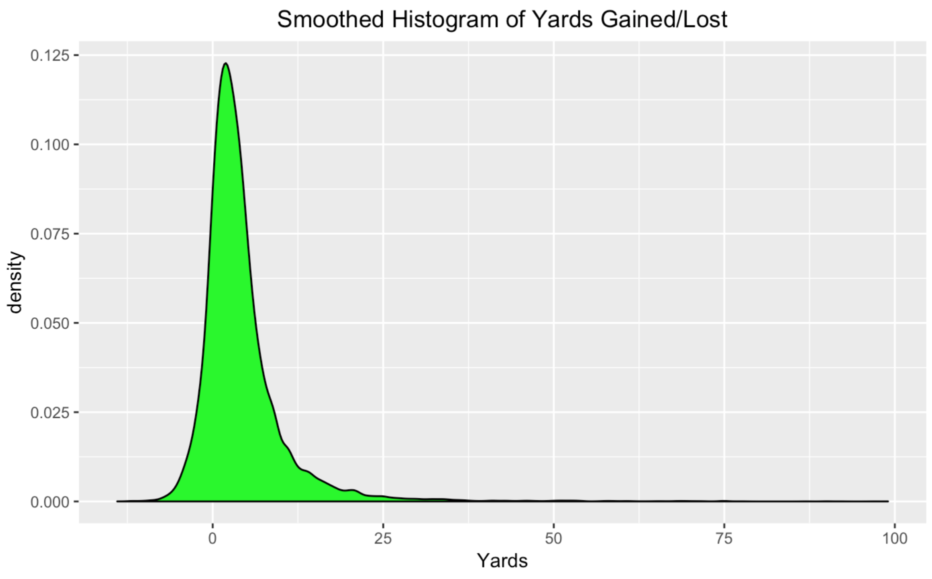


Figure 2: Smoothed histogram of the number of yards gained/lost for all plays in the dataset.

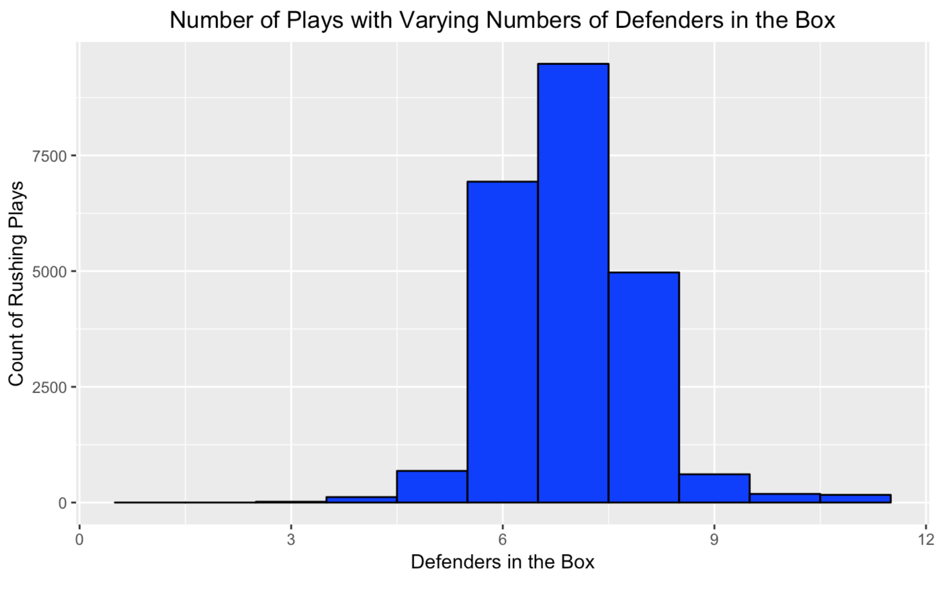


Figure 3: Bar chart of the number of defenders in the box for all rushing plays.

3.3 Distribution of Rushing Plays by Quarter and Down

Finding out when teams are likely to run the ball is important in determining the relative success of a rushing play. For example, near the end of the game, teams that are winning will often prefer rushing plays because the game clock will continue to run. Similarly, on 3rd down plays with a long distance (10+ yards) to get a first down, many teams will run the ball. This strategy helps avoid possible interceptions on deep passes, which are much more detrimental to a team than having to punt on 4th down. Bar charts for both rushing plays by quarter and down are given in Figures 4 and 5. Rushing plays are highest in the 1st quarter and on 1st down.

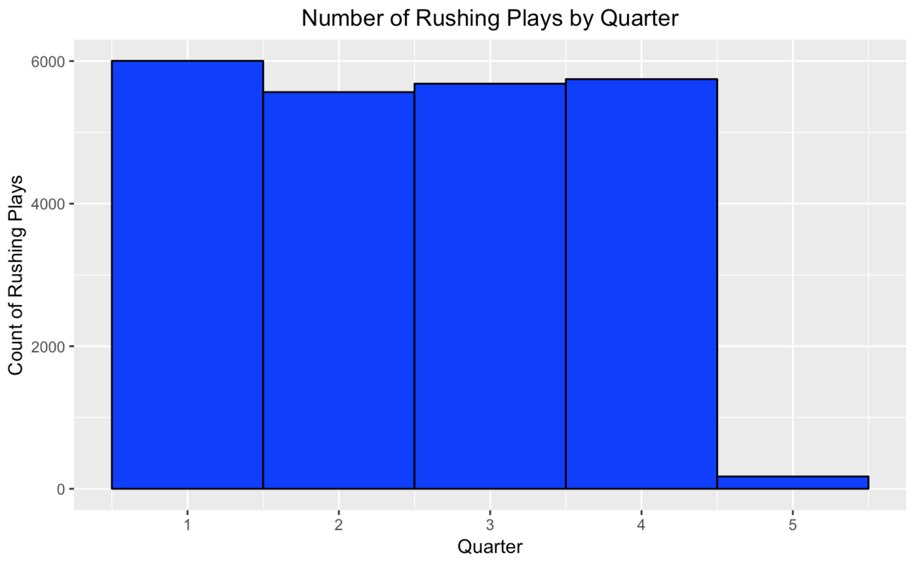


Figure 4: Number of rushing plays attempted by quarter. Note that quarter 5 refers to rushing plays performed in overtime.

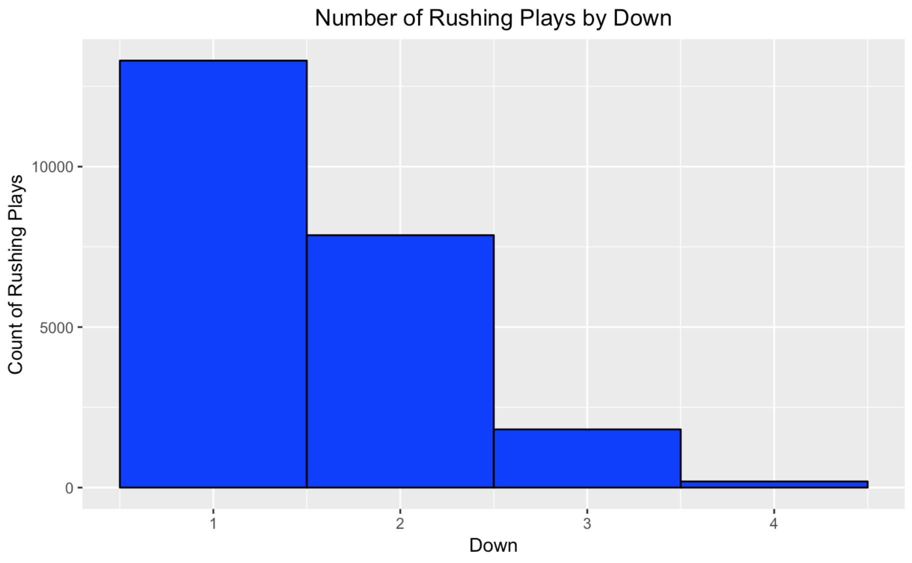


Figure 5: Number of rushing plays attempted by down.

3.4 Player Location Visualization

This dataset provides the xy location of each player on the field at the time the ball is handed off to the running back. This information is even more valuable than number of defenders in the box, because the distance between players can be accurately measured. Additionally, in some defensive schemes, a player will start in the box but then drop into coverage. When this occurs, the player will likely not be involved in the running play. An example of the xy location of all players is shown in Figure 6. The defense is shown in red, the offense is shown in blue, and the running back is shown in green.

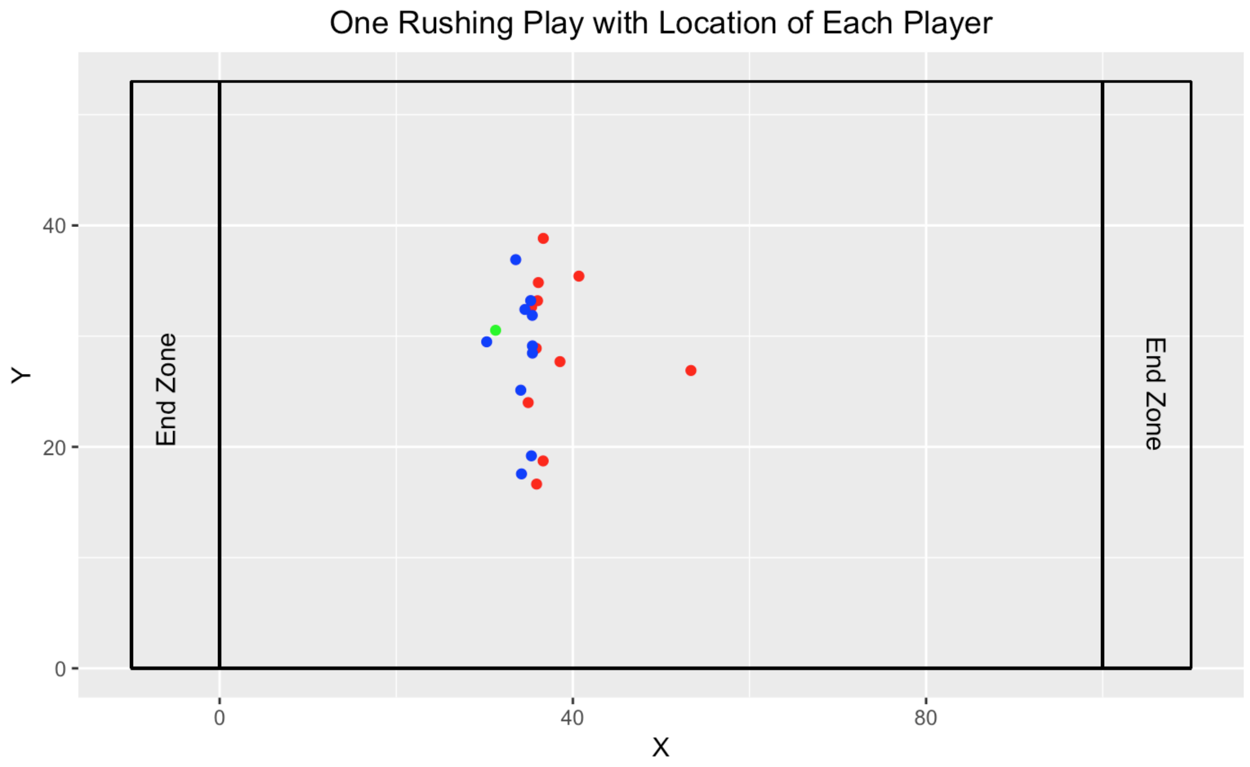


Figure 6: Example of the player location data for a single rushing play.

4 Methods and Performance Comparison

To get the data into a usable form for a regression model, a number of preprocessing steps were completed. This included: converting columns with text data into a usable numeric form, creating dummy variables for categorical variables, and creating columns that used data from multiple columns. A list of the features and type of feature is shown in Table 1. The full R code for preprocessing is in the Appendix.

Once the data was preprocessed, 80% of the data was used for training and the remaining 20% was used as a test set. The objective of the project is to outperform a model that always predicts a 4 yard gain, which is the median number of yards gained. This model has a RMSE of 6.43 yards, so the objective is to create a model that has a lower RMSE.

Feature	Type of Variable
Is Indoor Stadium	Binary
Play direction	Binary
Down	Categorical
Offensive Team Winning	Categorical
Offensive Team Losing	Categorical
Offense formation	Categorical
Quarter	Categorical
Current distance to the endzone	Continuous
Number of defensive linemen	Continuous
Number of defensive linebackers	Continuous
Number of defenders in the box	Continuous
Temperature	Continuous
Time remaining in quarter (seconds)	Continuous
Yards to gain a first down	Continuous

Table 1: Features created from original NFL dataset to build the linear regression models

4.1 Linear Regression Model

The first model trained was a linear regression model using all of the features directly. This model has a very low R^2_{adj} of 1.9% and a RMSE of 5.94. While this RMSE is high, this still outperforms the initial objective RMSE of the project. The code used to fit the model is shown below in Listing 1 and the full output is shown in the Appendix.

Listing 1: Linear Regression Model

```
# Fit a Linear Regression Model
lin_reg <- lm(Yards ~. , data = NFL_Train)

# Display Summary of the Model
summary(lin_reg)

# Calculate RMSE on the test set
rmse(NFL_Test$Yards, predict(lin_reg, NFL_Test))
```

Viewing the summary of this model, a large number of features were not statistically significant. In fact, only 8 of the 25 features were significant at the level of 0.1. The factors that were significant related to the number of defenders in the box, the yards needed to gain a first down, game situation (quarter, down, winning/losing), and the position of the team on the field. Based on prior knowledge of the NFL, it is not surprising that these factors are significant. Because there were many features that were not significant, I chose to next use a stepwise linear regression model to see if removing features would improve performance.

4.2 Stepwise Linear Regression Model

Next, I used a subset selection model that chooses the best model based on AIC. The final subset model included 10 of the original 25 features, but performed similarly in both R^2_{adj}=1.9% and RMSE of 5.94. The code used to fit the model is shown below in Listing 2.

Listing 2: Stepwise Linear Regression Model

```
# Stepwise Fit the Model
stepwise_model <- step(lin_reg, direction = c('both'), trace=0)

# Show which terms were included in the final model
attr(summary(stepwise_model)$terms, "term.labels")

# Display the adj R^2
summary(stepwise_model)$adj.r.squared

# Calculate RMSE on the test set
rmse(NFL_Test$Yards, predict(stepwise_model, NFL_Test))
```

In addition to the features that were statistically significant in the original Linear Regression Model, the stepwise model also used information about the offensive formation. The formations selected for the stepwise model are offensive formations where runs occur very often compared to the other formations. However, due to the extremely low R^2_{adj} and similar RMSE to the full linear regression model, the incorporation of these factors may be due to fitting noise in the dataset.

4.3 Elastic Net Model

Finally, an optimized elastic net model was used to fit the data. To find the optimal α and λ cross-validation of the training set was used. Once the optimal α and λ were found, the elastic model was trained and performance was measured on the test set. The code for fitting the elastic net model is shown below in Listing 3.

Listing 3: ElasticNet Model

```
# Fit the optimized glmnet model and determine RMSE and R^2
elastic_net <- glmnet(NFL_Train_X_matrix, NFL_Train_Y, alpha = best_alpha,
                      lambda = best_lambda, standardize = TRUE)

# Make predictions for test data
y_pred <- predict(elastic_net, NFL_Test_X_matrix)

# Calculate RMSE for the test set
rmse(NFL_Test_Y, y_pred)
```

The elastic model achieved similar performance to the linear regression and stepwise regression models with a RMSE of 5.94. Ultimately the models low R^2_{adj} show that the relationship between the features and yards gained is not linear. Performing subset selection and regularization did not meaningfully help the model perform better. This is likely due to the relationship between the various features and yards gained being highly non-linear. To improve the fit, a non-linear modelling framework would be needed to accurately capture the relationship.

5 Possible Improvements

There are many areas for improvement for the current model to predict rushing yards per play. A few possible improvements are listed below with a brief explanation for each in the following subsections.

1. Tracking player performance week to week and incorporating the trend of each player into the model
2. Using a stronger modeling framework (such as neural networks or gradient boosted machines)
3. Using a graph-based approach to model the relative position of each player on the field

5.1 Tracking Player Performance

There are two ways to use past performance data to provide better predictions on yards gained for a rush play. First, taking the effectiveness of the runner in the past few weeks into account would likely improve the model. When runners are performing well, defenses will likely focus more on stopping them, leading to less yards gained on each rush play. On the other hand, if the team's quarterback has been very effective at pass plays, then the defense will focus less on the runner, which could lead to more yards gained on each rush play. Similarly, the recent skill of the defense to stop running plays will significantly alter how the offense plans to play against the defense. Finally, injuries to important players will effect the running back's ability to gain yards as well. An example of this was seen earlier this year, when the New York Jets star running back Leveon Bell was not very successful while their quarterback Sam Darnold was injured. Once Sam Darnold returned from injury, Leveon Bell was more successful on running plays.

5.2 Stronger Modeling Framework

The NFL rushing data has many features that are inherently non-linear. Feature engineering can be done to create non-linear features (splines, higher-order terms, or interaction terms), but will likely not be as effective as modeling frameworks that can 1) learn features through training or 2) determine the relative importance of features during training to boost model performance. An example of the first type is a neural network, where the hidden layers within a neural network can learn complicated feature embeddings that may not have an equivalent physical interpretation but are nonetheless excellent at improving predictive performance. An example of the second type of modeling framework is a gradient boosted machine (such as XGBOOST, LGBM, or CATBOOST). These algorithms are an ensemble of weak learners (typically decision trees). In the first iteration a decision tree is fit to the data and the relative performance is determined. Next, gradient boosted machines will weight data that was poorly classified and develop a new decision tree that better classifies this data. The algorithm continues adding trees (up to a predetermined limit) and ensembles the results of each individual decision tree. As gradient boosted machines learn, the relative importance of each feature can be identified.

To test this hypothesis I fit a more complex model to the data. Using a small neural network the RMSE on the test set was reduced to 0.99. The code for fitting a neural network in Python is shown below in Listing 4. Ultimately, this shows that the relationship can be modeled accurately using data analysis tools, however the relationship is non-linear and thus linear models will not provide quality predictions.

Listing 4: Neural Network Model

```python
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from math import sqrt

train_df = pd.read_csv("NFL_Training_Data.csv")
test_df = pd.read_csv("NFL_Testing_Data.csv")

X_train = train_df.iloc[:, :-1]
y_train = train_df.iloc[:, -1]
X_test = test_df.iloc[:, :-1]
y_test = test_df.iloc[:, -1]


def build_regressor():
    regressor = Sequential()
    regressor.add(Dense(units=12, input_dim=26))
    regressor.add(Dense(units=4))
    regressor.add(Dense(units=1))
    regressor.compile(optimizer='adam', loss='mean_squared_error', metrics=['mse'])
    return regressor


regressor = KerasRegressor(build_fn=build_regressor, batch_size=64, epochs=200)
results = regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
rmse = sqrt(np.square(np.subtract(y_test, y_pred)).mean())
```

5.3 Graph-Based Modelling Approach

An interesting improvement for the model would be to use a graph-based approach to the player's positions on the field. Figure 6 shows the location of each player on the field. A Voronoi diagram, where the region of the closest player is shaded, could be generated for each play. Because football plays begin fairly similarly, analyzing these diagrams using a Convolutional Neural Network may lead to interesting results. For example, similar diagrams may lead to similar rushing success for the running back. Analyzing the area of of each cell in the Voronoi diagram for the running back may also lead to interesting relationships that help to better predict rushing performance.

6 Conclusion

Rushing plays in the NFL are an integral part of the game but have had little analysis in the past. These plays are highly variable due to the skill of players involved, formations, and the small differences between a successful/unsuccessful run. Predicting the success of a running play in the NFL is thus extremely difficult. In this project I used various linear regression models to try to predict the yards gained for a given runner based on a number of features about the game situation. While I was able to beat the baseline model of a constant prediction of 4 yards, ultimately the linear model was not complex enough to model the relationship. A small decrease in RMSE was seen by using the linear models but ultimately a more complex modeling structure would be needed to better understand the relationship between game features and yardage gained for running plays in the NFL

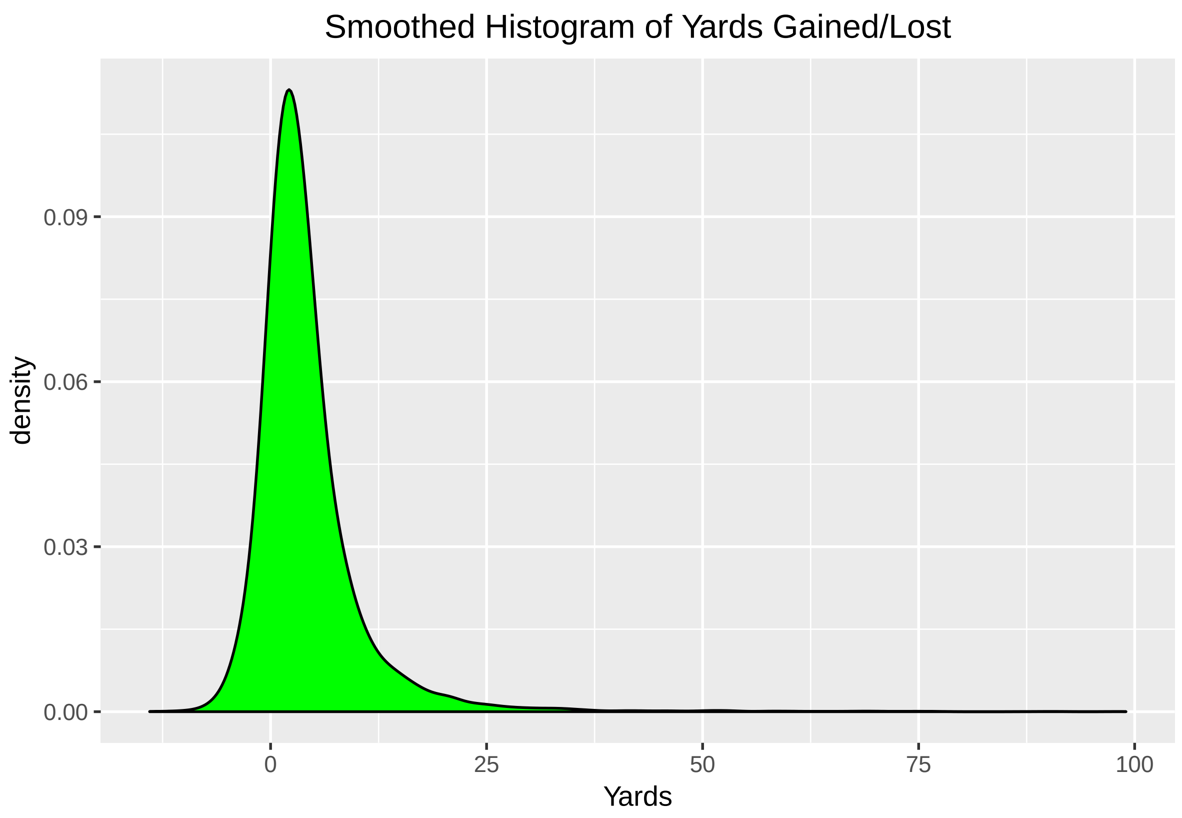
Appendix: R Markdown Output for Data Visualization and Model Fitting

EDA_NFL_Rushing_Data

```r
library(ggplot2)
library(Metrics)
library(plyr)
library(lubridate)
library(caret)
library(dplyr)
library(glmnet)
set.seed(31)
```

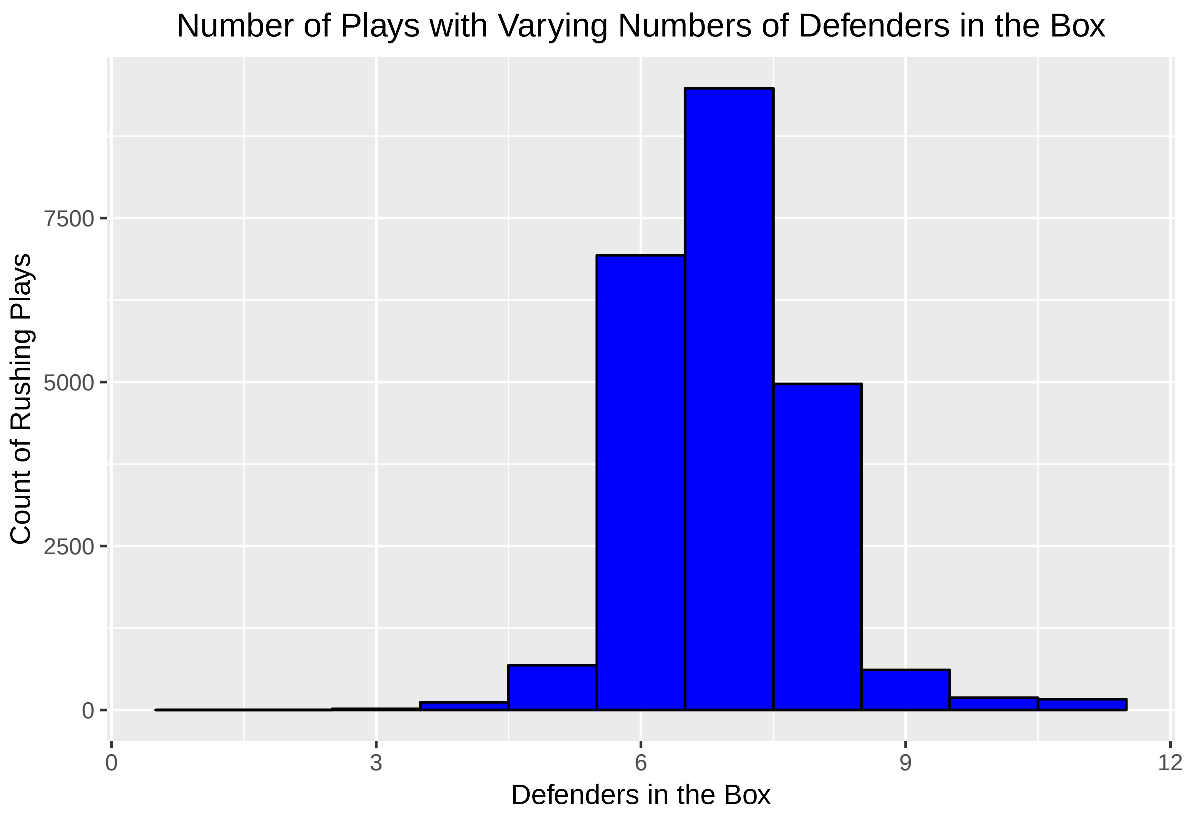
```r
NFL_Rushing_Data_2 <- read.csv("~/OneDrive/GradResearch/Grad School/Advanced_Data_Analytics/Project/NFL_

NFL_Rushing_Data_2_rushing_plays = NFL_Rushing_Data_2[seq(1, nrow(NFL_Rushing_Data_2), 22), ]
```

```r
# Change the width of bins
ggplot(NFL_Rushing_Data_2_rushing_plays, aes(x=Yards)) +
  geom_density(adjust=3, fill="green") +
  labs(
    title="Smoothed Histogram of Yards Gained/Lost"
  )+
  theme(plot.title = element_text(hjust = 0.5))
```

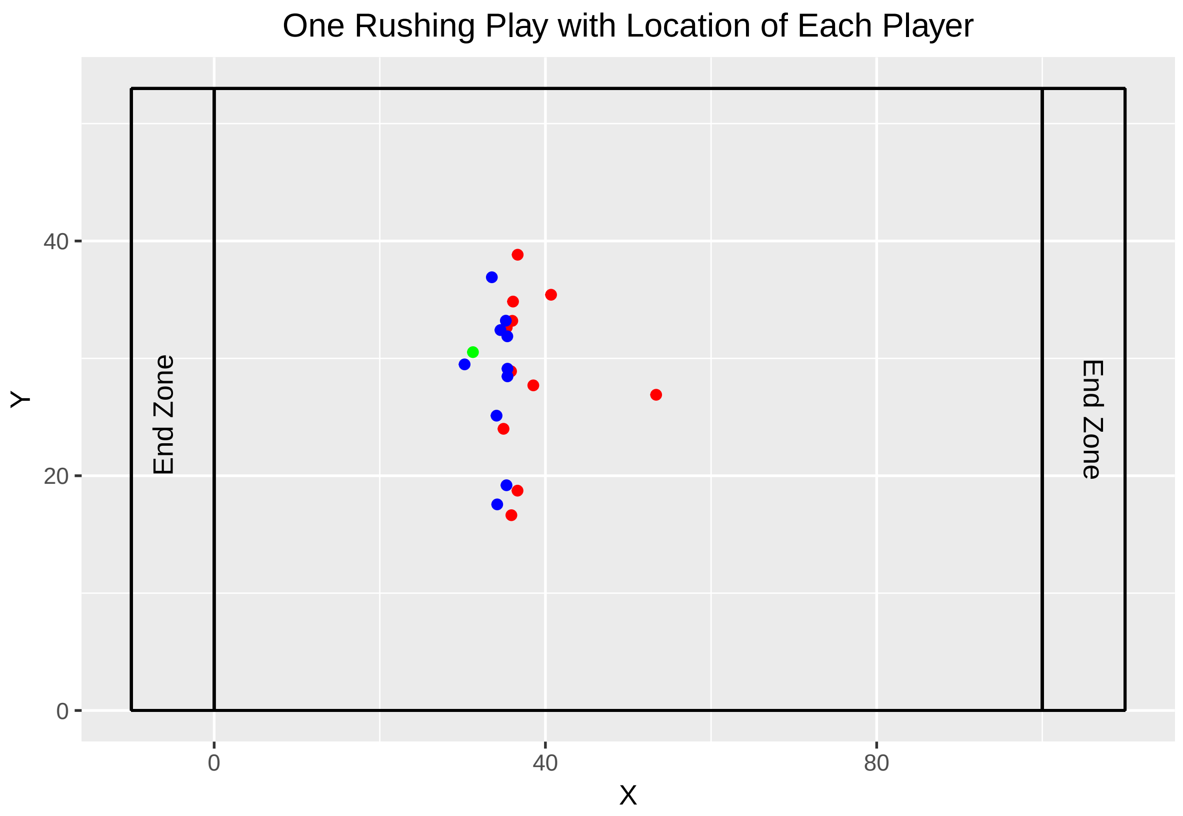


```r
# Change the width of bins
ggplot(NFL_Rushing_Data_2_rushing_plays, aes(x=DefendersInTheBox)) +
  geom_histogram(binwidth = 1, color="black", fill="blue") +
  labs(
    title="Number of Plays with Varying Numbers of Defenders in the Box"
  )+
  xlab("Defenders in the Box") +
  ylab("Count of Rushing Plays") +
  theme(plot.title = element_text(hjust = 0.5))
```

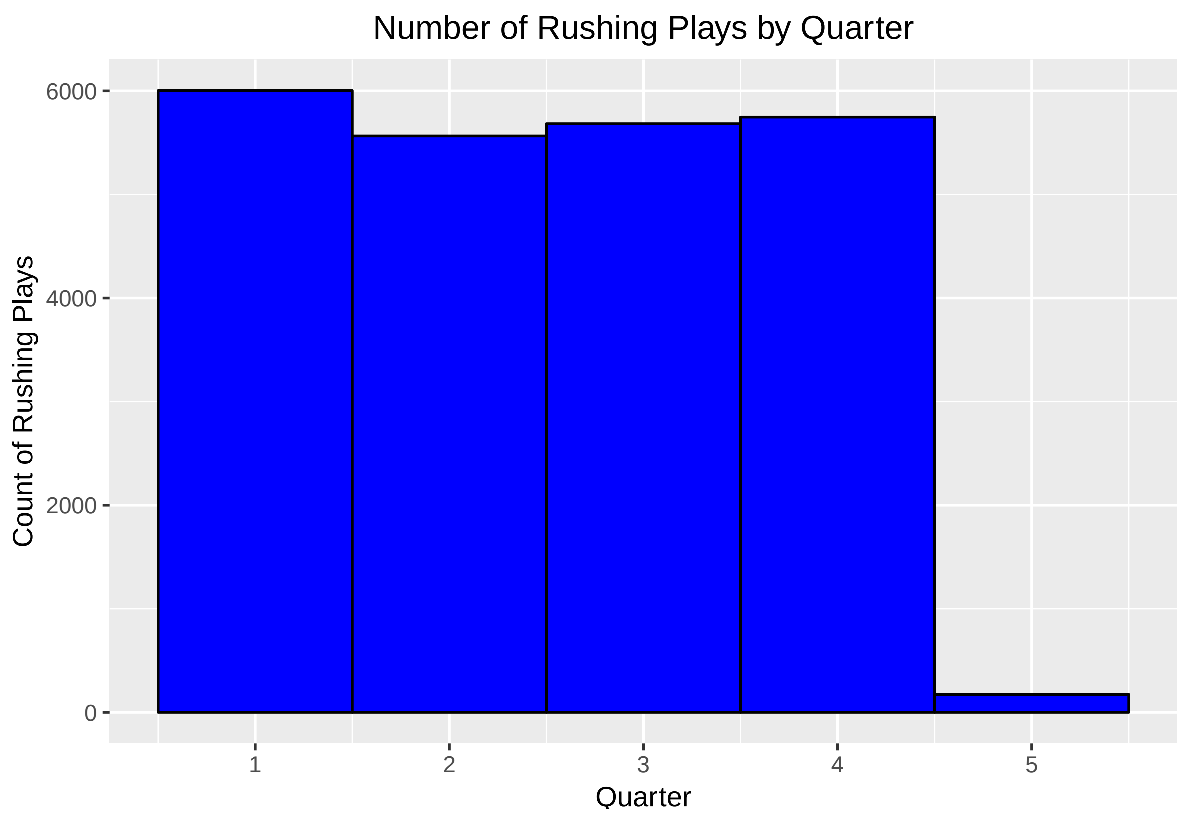
```
Warning: Removed 3 rows containing non-finite values (stat_bin).
```



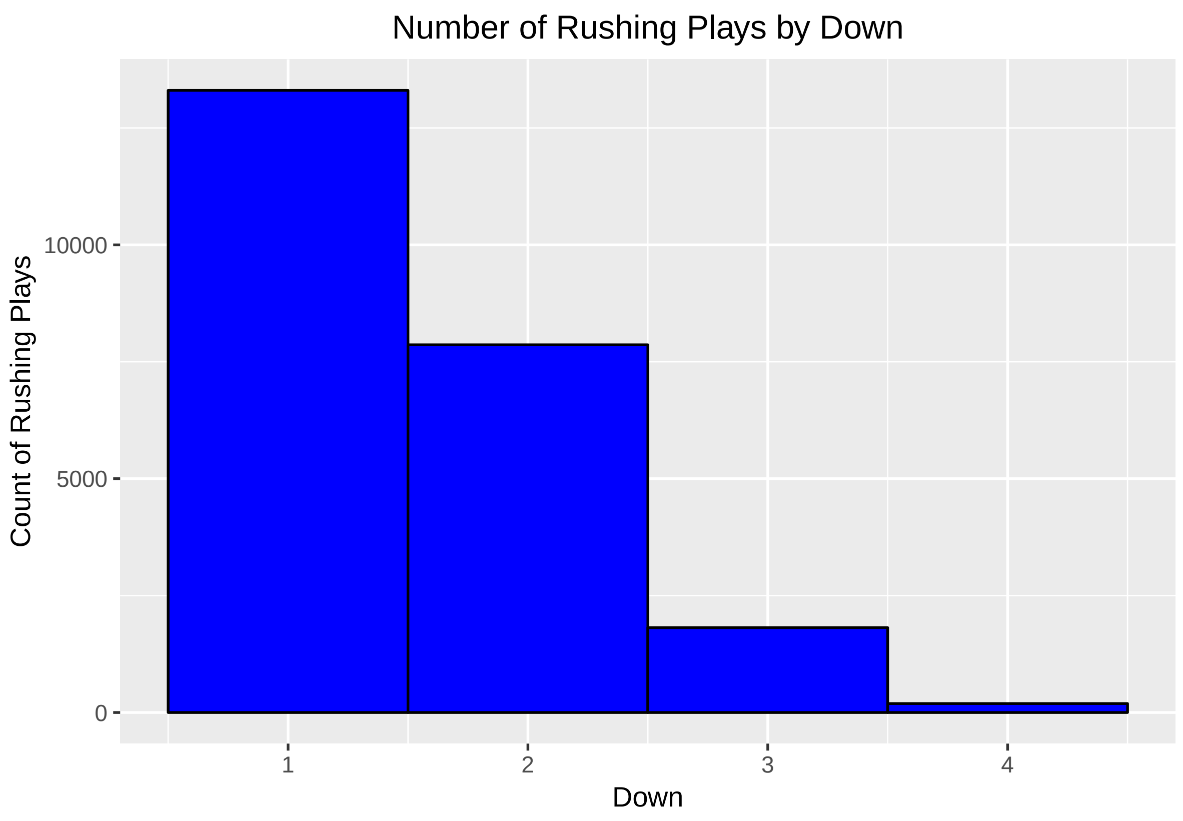
```
one_play_df <- NFL_Rushing_Data_2[1:22,]
one_play_df$Color_Scheme <- ifelse(one_play_df$Team =="away", "red", "blue")
one_play_df$Color_Scheme[19] <- "green"
one_play_df$X <- 100 - one_play_df$X + 10
ggplot(one_play_df, aes(x=X, y=Y)) + geom_point(color=one_play_df$Color_Scheme) +
  xlim(-10,110) +
  ylim(0, 53) +
  ggtitle("One Rushing Play with Location of Each Player") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_segment(aes(x = 0, y = 0, xend = 0, yend = 53)) +
  geom_segment(aes(x = 100, y = 0, xend = 100, yend = 53)) +
  geom_segment(aes(x = -10, y = 0, xend = 110, yend = 0)) +
  geom_segment(aes(x = -10, y = 53, xend = 110, yend = 53)) +
  geom_segment(aes(x = -10, y = 0, xend = -10, yend = 53)) +
  geom_segment(aes(x = 110, y = 0, xend = 110, yend = 53)) +
  annotate(geom = 'text', label = 'End Zone', x = -5, y = 20, hjust = 0, vjust = 0, angle=90) +
  annotate(geom = 'text', label = 'End Zone', x = 105, y = 30, hjust = 0, vjust = 0, angle=270)
```



```
# Change the width of bins
ggplot(NFL_Rushing_Data_2_rushing_plays, aes(x=Quarter)) +
  geom_histogram(binwidth = 1, color="black", fill="blue") +
  labs(
    title="Number of Rushing Plays by Quarter"
  )+
  xlab("Quarter") +
  ylab("Count of Rushing Plays") +
  theme(plot.title = element_text(hjust = 0.5))
```



```
# Change the width of bins
ggplot(NFL_Rushing_Data_2_rushing_plays, aes(x=Down)) +
  geom_histogram(binwidth = 1, color="black", fill="blue") +
  labs(
    title="Number of Rushing Plays by Down"
  )+
  xlab("Down") +
  ylab("Count of Rushing Plays") +
  theme(plot.title = element_text(hjust = 0.5))
```



```r
NFL_Data_Full <- NFL_Rushing_Data_2[NFL_Rushing_Data_2$NflId==NFL_Rushing_Data_2$NflIdRusher,]

# Choose columns that will be used in the analysis
NFL_Data <- subset(NFL_Data_Full, select = c("Team", "PossessionTeam", "FieldPosition",
                                             "Quarter", "YardLine","GameClock", "Down",
                                             "Distance", "StadiumType", "OffenseFormation",
                                             "DefensePersonnel", "PlayDirection",
                                             "HomeScoreBeforePlay", "VisitorScoreBeforePlay",
                                             "Temperature"))

# Find if running back is home or away team
NFL_Data$FieldPosition <- ifelse(NFL_Data$FieldPosition == "",
                                 as.character(NFL_Data$PossessionTeam),
                                 as.character(NFL_Data$FieldPosition))

# Determine if yards to endzone needs to be adjusted
NFL_Data$YardsToEndzone <- ifelse(NFL_Data$PossessionTeam == NFL_Data$FieldPosition,
                                  100 - NFL_Data$YardLine, NFL_Data$YardLine)

# Create Dummy Variables for each quarter except overtime
NFL_Data$Quarter_1 <- ifelse(NFL_Data$Quarter == 1, 1, 0)
NFL_Data$Quarter_2 <- ifelse(NFL_Data$Quarter == 2, 1, 0)
NFL_Data$Quarter_3 <- ifelse(NFL_Data$Quarter == 3, 1, 0)
NFL_Data$Quarter_4 <- ifelse(NFL_Data$Quarter == 4, 1, 0)

# Time remaining in Quarter
NFL_Data$TimeRemaining <- (as.integer(substr(NFL_Data$GameClock, 1, 2)) * 60
                           + as.integer(substr(NFL_Data$GameClock, 4, 5)))

# Down
NFL_Data$Down_1 <- ifelse(NFL_Data$Down == 1, 1, 0)
NFL_Data$Down_2 <- ifelse(NFL_Data$Down == 2, 1, 0)
NFL_Data$Down_3 <- ifelse(NFL_Data$Down == 3, 1, 0)

# Yards to Gain First Down
NFL_Data$YardsToGain <- NFL_Data$Distance

# Which team is currently winning
NFL_Data$RunningTeamWinning <- ifelse((NFL_Data$HomeScoreBeforePlay >
                                        NFL_Data$VisitorScoreBeforePlay &
                                       NFL_Data$Team == "home") |
                                       (NFL_Data$VisitorScoreBeforePlay >
                                          NFL_Data$HomeScoreBeforePlay &
                                       NFL_Data$Team == "away"),1,0)
NFL_Data$RunningTeamLosing <- ifelse((NFL_Data$HomeScoreBeforePlay >
                                       NFL_Data$VisitorScoreBeforePlay &
                                       NFL_Data$Team == "away") |
                                       (NFL_Data$VisitorScoreBeforePlay >
                                          NFL_Data$HomeScoreBeforePlay &
                                       NFL_Data$Team == "home"),1,0)

# Offense Formation: Need to be categorical because they are significantly different
NFL_Data$Ace_Formation <- ifelse(NFL_Data$OffenseFormation == "ACE", 1, 0)
NFL_Data$Empty_Formation <- ifelse(NFL_Data$OffenseFormation == "EMPTY", 1, 0)
NFL_Data$I_Formation <- ifelse(NFL_Data$OffenseFormation == "I_FORM", 1, 0)
NFL_Data$Jumbo_Formation <- ifelse(NFL_Data$OffenseFormation == "JUMBO", 1, 0)
NFL_Data$Pistol_Formation <- ifelse(NFL_Data$OffenseFormation == "PISTOL", 1, 0)
NFL_Data$Shotgun_Formation <- ifelse(NFL_Data$OffenseFormation == "SHOTGUN", 1, 0)
NFL_Data$Singleback_Formation <- ifelse(NFL_Data$OffenseFormation == "SINGLEBACK", 1, 0)

# Play Direction
NFL_Data$Play_Left <- ifelse(NFL_Data$PlayDirection == "left",1,0)


# Defensive Personnel: Clean up the few that have OL which is most likely an additional DL
NFL_Data$DefensePersonnel <- mapvalues(NFL_Data$DefensePersonnel,
                                  from=c("4 DL, 5 LB, 1 DB, 1 OL",
                                         "2 DL, 4 LB, 4 DB, 1 OL",
                                         "5 DL, 4 LB, 1 DB, 1 OL",
                                         "3 DL, 4 LB, 3 DB, 1 OL",
                                         "5 DL, 3 LB, 2 DB, 1 OL"),
                                  to = c("5 DL, 5 LB, 1 DB",
                                         "3 DL, 4 LB, 4 DB",
                                         "6 DL, 4 LB, 1 DB",
                                         "4 DL, 4 LB, 3 DB",
                                         "6 DL, 3 LB, 2 DB"))

# Parse the DefensePersonnel to get the number for each position
NFL_Data$DefLine <- as.integer(substr(NFL_Data$DefensePersonnel, 1, 1))
NFL_Data$DefLB <- as.integer(substr(NFL_Data$DefensePersonnel, 7, 7))
```

```r
# NFL_Data$DefDB <- as.integer(substr(NFL_Data$DefensePersonnel, 13, 13))

# Clean up stadium data -- Outdoor Venues -> 0
NFL_Data$StadiumType <- mapvalues(NFL_Data$StadiumType,
                                  from=c("Outdoor", "Outdoors", "Open",
                                         "Indoor, Open Roof","Outddors",
                                         "Outdoor Retr Roof-Open", "Oudoor",
                                         "Ourdoor", "Retr. Roof-Open",
                                         "Heinz Field", "Outdor", "Retr. Roof - Open",
                                         "Domed, Open", "Domed, open", "Cloudy",
                                         "Bowl", "Outside"),
                                  to = c(rep(0,17)))

# Clean up stadium data -- Indoor Venues -> 1
NFL_Data$StadiumType <- mapvalues(NFL_Data$StadiumType,
                                  from=c("Indoor", "Indoors", "Retractable Roof",
                                         "Retr. Roof-Closed", "Retr. Roof - Closed",
                                         "Domed, closed", "Dome, closed", "Domed",
                                         "Closed Dome", "Dome", "Indoor, Roof Closed",
                                         "Retr. Roof Closed" ),
                                  to = c(rep(1,12)))

# Move stadium to end of dataframe for easy removal of first columns
NFL_Data$IndoorStadium <-  NFL_Data$StadiumType

# Number of defenders in the box
NFL_Data$DefendersInTheBox <- NFL_Data_Full$DefendersInTheBox

NFL_Data$DefendersInTheBox[is.na(NFL_Data$DefendersInTheBox)] <- (NFL_Data$DefLine +
                                                                  NFL_Data$DefLB)
```

```
Warning in NFL_Data$DefendersInTheBox[is.na(NFL_Data$DefendersInTheBox)]
<- (NFL_Data$DefLine + : number of items to replace is not a multiple of
replacement length
```

```r
NFL_Data$Temperature[is.na(NFL_Data$Temperature)] <- 70

# Number of Yards the rusher gained/lost
NFL_Data$Yards <- NFL_Data_Full$Yards

# Delete beginning columns used for feature engineering
NFL_Data <- NFL_Data[, -c(1:14)]

# Prepare a training and testing set
training.samples <- NFL_Data$Yards %>% createDataPartition(p=0.8, list=FALSE)
NFL_Train <- NFL_Data[training.samples, ]
NFL_Test <- NFL_Data[-training.samples, ]
```

```r
# Fit Linear Regression Model
lin_reg <- lm(Yards ~. , data = NFL_Train)

# Display Summary
summary(lin_reg)
```

```
Call:
lm(formula = Yards ~ ., data = NFL_Train)

Residuals:
    Min      1Q  Median      3Q     Max 
-18.933  -3.201  -1.181   1.348  93.645 

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)    
(Intercept)           8.3639279  1.2725641   6.573 5.08e-11 ***
Temperature          -0.0061293  0.0028810  -2.128 0.033391 *  
YardsToEndzone        0.0177587  0.0020165   8.807  < 2e-16 ***
Quarter_1            -0.7249480  0.5595110  -1.296 0.195101    
Quarter_2            -0.6611078  0.5683200  -1.163 0.244736    
Quarter_3            -0.6869278  0.5705754  -1.204 0.228635    
Quarter_4            -0.9440254  0.5714203  -1.652 0.098538 .  
TimeRemaining         0.0002417  0.0001875   1.289 0.197458    
Down_1                0.6886212  0.5426830   1.269 0.204486    
Down_2                0.6959691  0.5377830   1.294 0.195631    
Down_3                0.9611788  0.5547459   1.733 0.083175 .  
YardsToGain           0.0529603  0.0156583   3.382 0.000720 ***
RunningTeamWinning    0.4285086  0.1507282   2.843 0.004475 ** 
RunningTeamLosing     0.1293820  0.1438649   0.899 0.368489    
Ace_Formation        -1.1518560  6.5299730  -0.176 0.859985    
Empty_Formation       1.0504569  1.6544633   0.635 0.525486    
I_Formation           0.4115786  0.8078656   0.509 0.610433    
Jumbo_Formation       0.2946302  0.8806446   0.335 0.737959    
Pistol_Formation      0.1727530  0.8531865   0.202 0.839544    
Shotgun_Formation    -0.0286932  0.8029901  -0.036 0.971496    
Singleback_Formation  0.1961568  0.8019332   0.245 0.806765    
Play_Left             0.0648133  0.0952821   0.680 0.496371    
DefLine              -0.2776792  0.1073322  -2.587 0.009686 ** 
DefLB                -0.3471402  0.1024453  -3.389 0.000704 ***
IndoorStadium0       -0.1708973  0.1959371  -0.872 0.383108    
IndoorStadium1       -0.3295760  0.2126746  -1.550 0.121237    
DefendersInTheBox    -0.4947497  0.0694135  -7.128 1.06e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.479 on 18512 degrees of freedom
Multiple R-squared:  0.02012,   Adjusted R-squared:  0.01875 
F-statistic: 14.62 on 26 and 18512 DF,  p-value: < 2.2e-16
```

```r
# Calculate RMSE on the test set
rmse(NFL_Test$Yards, predict(lin_reg, NFL_Test))
```

```
[1] 5.936956
```

```r
# Stepwise fit the model
stepwise_model <- step(lin_reg, direction = c('both'), trace=0)

# Show which terms were included in the final model
attr(summary(stepwise_model)$terms, "term.labels")
```

```
 [1] "Temperature"          "YardsToEndzone"       "Quarter_4"
 [4] "YardsToGain"          "RunningTeamWinning"   "I_Formation"
 [7] "Singleback_Formation" "DefLine"              "DefLB"
[10] "DefendersInTheBox"
```

```
# Display the adj R^2
summary(stepwise_model)$adj.r.squared
```

```
[1] 0.0189212
```

```
# Make predictions
rmse(NFL_Test$Yards, predict(stepwise_model, NFL_Test))
```

```
[1] 5.937868
```

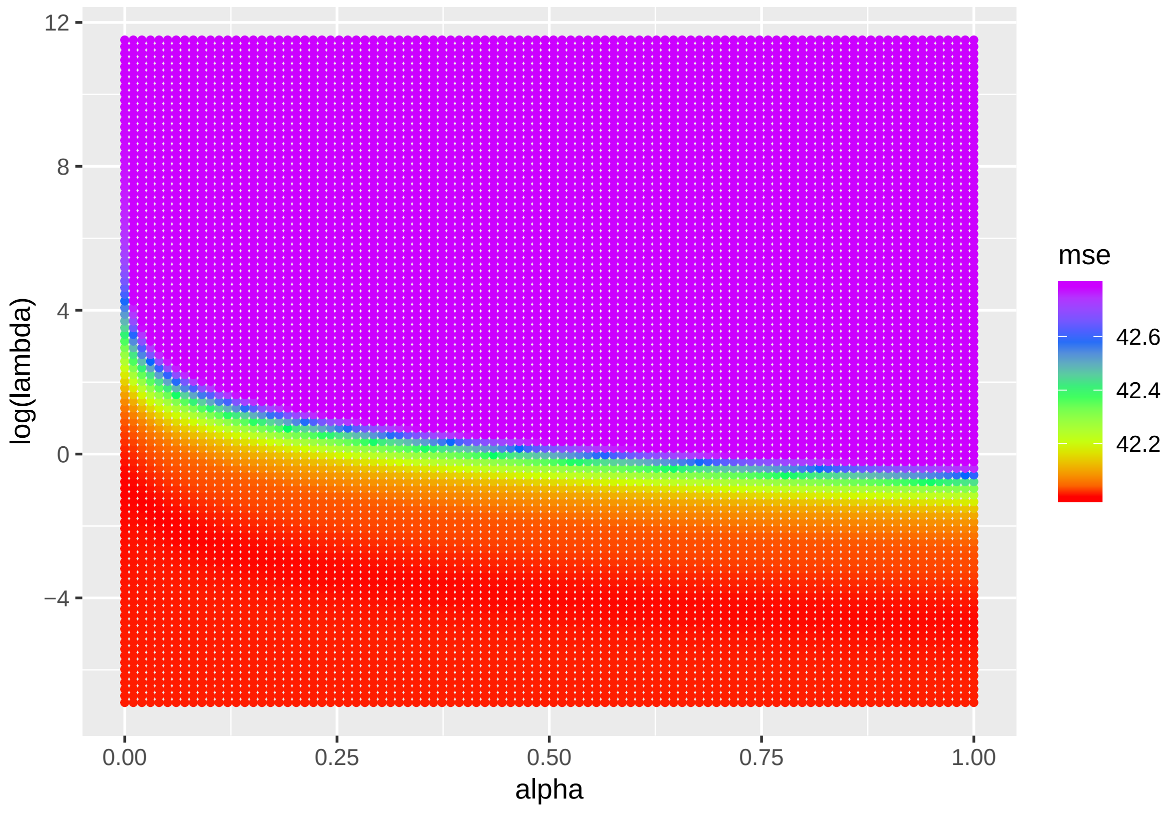
```
NFL_Train_X_matrix <- data.matrix(subset(NFL_Train, select = -Yards))
NFL_Train_Y <- data.matrix(subset(NFL_Train, select = Yards))

NFL_Test_X_matrix <- data.matrix(subset(NFL_Test, select = -Yards))
NFL_Test_Y <- data.matrix(subset(NFL_Test, select = Yards))

cv_fold <- 5
fold_sequence <- rep(1:cv_fold, length.out = length(NFL_Train_Y))
fold_seq_shuffled <- sample(fold_sequence, length(fold_sequence),
                            replace = FALSE)
```

```
# Test 100 values of alpha
alphas_to_try <- seq(0,1,length.out=100)

# Test 100 values of lambda
lambdas_to_try <- 10^seq(-3, 5, length.out = 100)

# Iterate through each alpha and lambda combination to find the optimal MSE
for (i in 1:length(alphas_to_try)){
  elastic_net <- cv.glmnet(NFL_Train_X_matrix, NFL_Train_Y, foldid = fold_seq_shuffled,
                           alpha = alphas_to_try[i], lambda = lambdas_to_try,
                           standardize = TRUE, nfolds = cv_fold)
  result_matrix <- matrix(nrow = 100, ncol = 3)

  # Save the mse for each alpha and lambda combination
  result_matrix[,1] <- alphas_to_try[i]
  result_matrix[,2] <- lambdas_to_try

  # cv.glmnet sorts the cvm call by lambda largest to smallest
  result_matrix[,3] <- rev(elastic_net$cvm)

  # For first iteration, initialize matrix, after that add rows to end
  if (i == 1){
    results <- result_matrix
  } else {
    results <- rbind(results, result_matrix)
  }
}

# Name columns in preparation for conversion to data frame
colnames(results) <- c("alpha", "lambda", "mse")

# Convert matrix to data frame so ggplot can be used
```

```r
results_df <- as.data.frame(results)

# Find the minimum MSE
min(results_df$mse)
```

```
[1] 42.00346
```

```r
# Find the location of minimum MSE to identify the best alpha and lambda value
min_index <- which(results_df$mse == min(results_df$mse, na.rm = TRUE),
                   arr.ind = TRUE)

# Optimal Alpha value
best_alpha <- results_df$alpha[min_index]

# Optimal Lambda value
best_lambda <- results_df$lambda[min_index]

# Plot alpha vs log(lambda) and color based on MSE
ggplot(results_df, aes(x=alpha, y=log(lambda), color=mse)) +
  geom_point(size = 1) +
  scale_color_gradientn(colours = rainbow(5))
```



```r
# Fit the optimized glmnet model and determine RMSE and R^2
elastic_net <- glmnet(NFL_Train_X_matrix, NFL_Train_Y, alpha = best_alpha,
                      lambda = best_lambda, standardize = TRUE)

# Make predictions for test data
y_pred <- predict(elastic_net, NFL_Test_X_matrix)
```

```r
# Calculate RMSE for the test set
rmse(NFL_Test_Y, y_pred)
```

```
[1] 5.936321
```