Should I Buy a Ticket? Predicting Wins in College Football

Kenneth Geers
HarvardX: PH125.9x

April 14, 2020

Contents

Introduction

Everyone Loves a Winner

College Football is one of America's most popular sports. However, it is expensive to buy a ticket and to travel to a game. Therefore, this paper asks whether it is possible, using R code, machine learning, and a variety of on-field performance metrics, to build a model that can predict the number of games your favorite team will win in a season - so that you do not waste your time or money.

We construct three models:

1. Logistic Regression (LR)
2. K-Nearest Neighbors (KNN)
3. Random Forest (RF)

In this study, LR provided the best initial results, with KNN in second place. The RF algorithm came in third, but was still better than blind guessing. In the future, however, it is possible that other models, or fine tuning these models, will yield a different result.

Data & Methods

Libraries

The following R libraries were used in this research: tidyverse, dplyr, ggplot2, Hmisc, ggpubr, corrplot, nnet, ROCR, caret, pROC, mlbench, and randomForest.

Data Source

This analysis uses the 2019 College Football statistics from the NCAA website, which are grouped by 130 Division I teams, and can be found here: https://www.ncaa.com/stats/football/fbs.

These data were originally downloaded and organized into one text file by Jeff Gallini, who posted the data to Kaggle for others to analyze: https://www.kaggle.com/jeffgallini/college-football-team-stats-2019.

For this research, we modified Gallini's data slightly, in particular by separating the "Win-Loss" column into "Wins" and "Losses". The new data set is contained in this file: CFB2019.txt.

```
FB_Analysis <- read.csv("CFB2019.txt")
```

Data Dimensions

This code retrieves the dimensions of our new variable: FB_Analysis.

```
dim(FB_Analysis)
```

```
## [1] 130 147
```

There are 130 observations (representing 130 football teams), and 147 variables (representing 146 specific on-field performance metrics for each team).

Data Structure

Here is the structure of our data. "Team" is a factor, followed by a wide variety of variables (performance metrics) in both integer and numeric formats.

```
str(FB_Analysis [1:15])
```

```
## 'data.frame':    130 obs. of  15 variables:
##  $ Team              : Factor w/ 130 levels "Air Force","Akron",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ Games             : int  13 12 13 14 12 13 12 13 13 13 ...
##  $ Wins              : int  11 0 11 13 4 8 2 8 5 9 ...
##  $ Losses            : int  2 12 2 1 8 5 10 5 8 4 ...
##  $ Off.Rank          : int  51 130 6 39 30 94 111 33 89 64 ...
##  $ Off.Plays         : int  881 725 842 969 873 842 783 925 869 947 ...
##  $ Off.Yards         : int  5483 2918 6640 6064 5281 4837 4081 5711 4929 5285 ...
##  $ Off.Yards.Play    : num  6.22 4.02 7.89 6.26 6.05 5.74 5.21 6.17 5.67 5.58 ...
##  $ Off.TDs           : int  55 14 76 65 42 34 28 50 51 51 ...
##  $ Off.Yards.per.Game: num  422 243 511 433 440 ...
##  $ Def.Rank          : int  17 83 20 26 120 67 110 124 30 28 ...
##  $ Def.Plays         : int  752 871 877 940 889 928 827 1018 789 905 ...
##  $ Yards.Allowed     : int  4155 4967 4218 4705 5657 5113 5408 6218 4450 4381 ...
##  $ Yards.Play.Allowed: num  5.53 5.7 4.81 5.01 6.36 5.51 6.54 6.11 5.64 4.84 ...
##  $ Off.TDs.Allowed   : int  32 51 24 34 53 39 54 55 37 31 ...
```

First Rows of Data

This code shows the first 6 rows and 8 columns of data, ordered by team.

```
head(FB_Analysis [1:8])
```

```
##              Team Games Wins Losses Off.Rank Off.Plays Off.Yards Off.Yards.Play
## 1       Air Force    13   11      2       51       881      5483           6.22
## 2           Akron    12    0     12      130       725      2918           4.02
## 3         Alabama    13   11      2        6       842      6640           7.89
## 4 Appalachian St.    14   13      1       39       969      6064           6.26
## 5         Arizona    12    4      8       30       873      5281           6.05
## 6     Arizona St.    13    8      5       94       842      4837           5.74
```

Methods

We will use R code, machine learning, and the data shown above to build three models designed to predict the number of games each team will win in a season.

1. Logistic Regression (LR)
2. K-Nearest Neighbors (KNN)
3. Random Forest (RF)

We have two goals: to achieve a result that is better than blind guessing, and to select the best overall model.

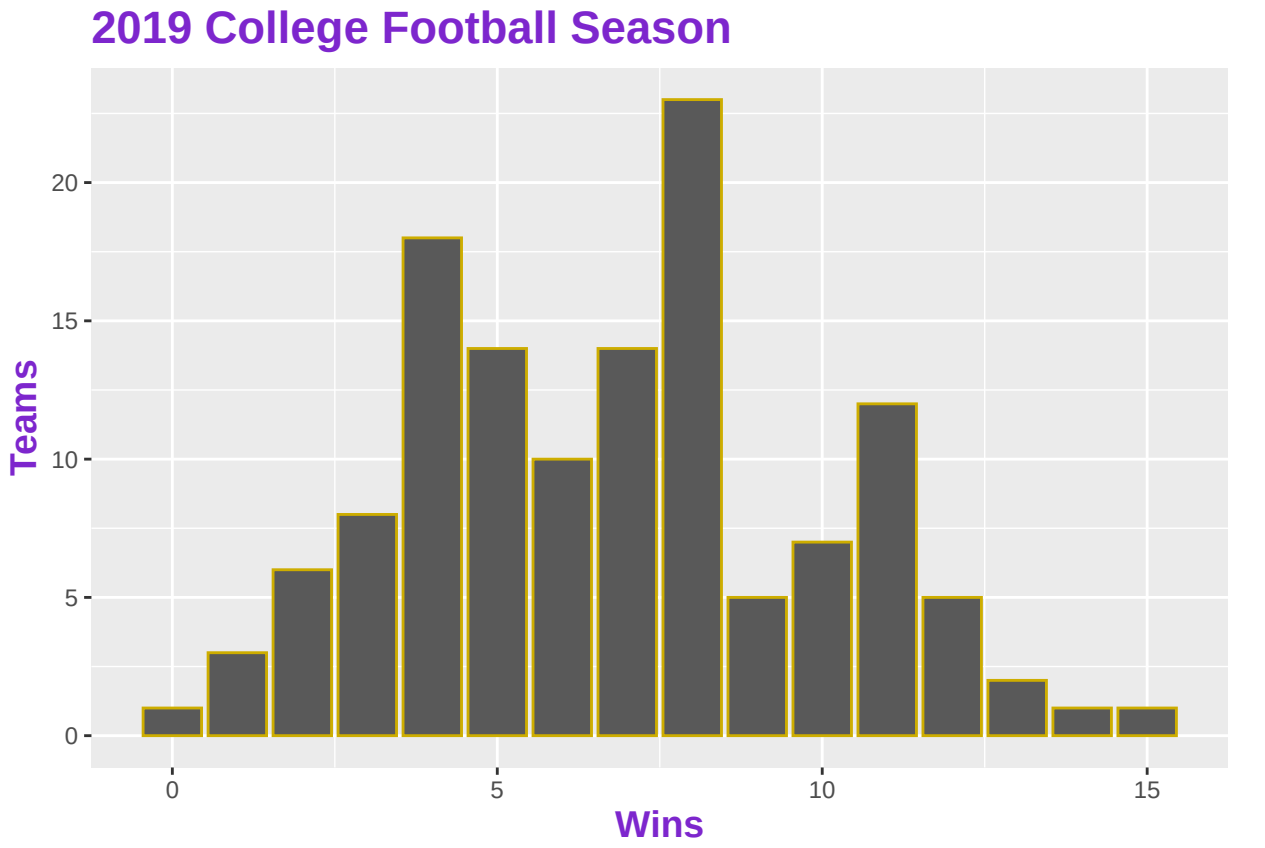
Visualization

Team Wins

First, let's visualize some of our primary data points.

The chart below shows the number of wins for all 130 teams in 2019.

```
ggplot(data = FB_Analysis) +
  geom_bar(aes(x = Wins), col = "gold3") +
  labs(title = "2019 College Football Season", x = "Wins", y = "Teams") +
  theme(title = element_text(color = "Purple3", face = "bold", size = 14))
```



The single most common number of wins was 8, which happened for 23 teams.

This means that if we guess blindly, our most accurate prediction could only be correct 17.69% of the time (23/130).

Our machine-learning task is to improve our predictive capability.

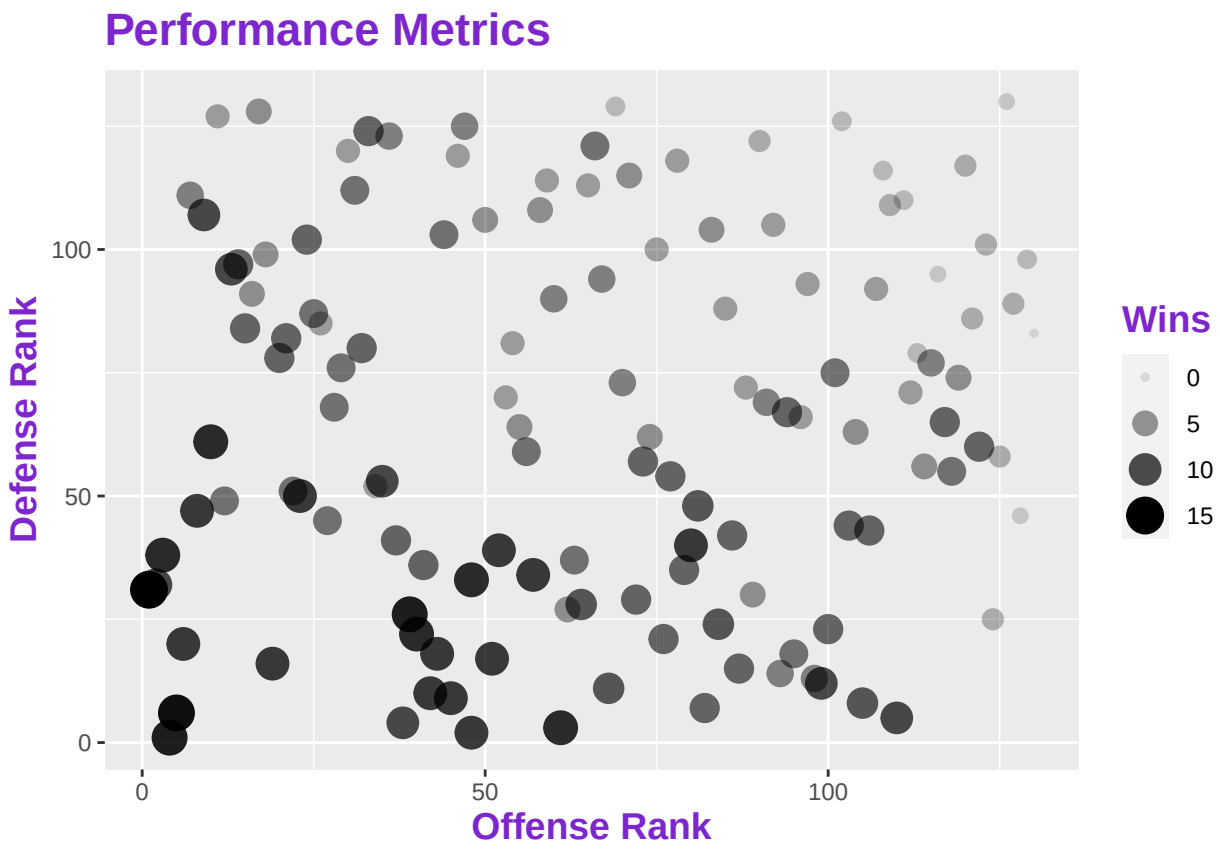
Performance Metrics

In the graph below, we establish something that is otherwise intuitive: better on-field performance usually yields more wins.

Each circle represents one of the 130 football teams. The size and shade of the circle represents that team's number of wins in 2019.

The x-axis represents offensive rank, and the y-axis is defensive rank (out of 146 total performance metrics). A lower rank means better relative performance.

```
ggplot(data = FB_Analysis) +
  geom_point(aes(x = Off.Rank, y = Def.Rank, alpha = Wins, size = Wins)) +
  labs(title = "Performance Metrics", x = "Offense Rank", y = "Defense Rank") +
  theme(title = element_text(color = "Purple3", face = "bold", size = 14))
```



The primary takeaway from this graph is that there is a strong, positive correlation between better performance and more wins.

The winningest teams have the best on-field statistics - and vice versa.

In our analysis below, we build machine-learning models to exploit this relationship.

They should discover the best mix of metrics to predict how many wins a team will have in a season.

Analysis & Results

Ranked Observations

The goal of this research is to use machine learning to predict how many wins a football team will have. If we build an effective model, it could help coaches, players, fans, gamblers, and investors to spend their time, money, and energy more wisely.

The NCAA provides 146 unique performance metrics, all of which may eventually be useful for analysis. For this initial essay, however, we will only use 24 of these metrics - those which have already been ranked for us, by team, from 1-130.

Here is the code to generate a new variable, "Ranked_Observations", and associate each metric with its team's win total:

```
Ranked_Observations <- FB_Analysis %>% select(matches("Rank|Wins"))
```

Here are the first 8 performance metrics (starting with the second column):

```
head(Ranked_Observations [1:9])
```

```
##   Wins Off.Rank Def.Rank First.Down.Rank First.Down.Def.Rank X4th.Down.Rank
## 1   11       51       17              45                   9              5
## 2    0      130       83             130                  58            124
## 3   11        6       20              15                  40             12
## 4   13       39       26              32                  51              1
## 5    4       30      120              63                 116             19
## 6    8       94       67             100                  87             55
##   X4rd.Down.Def.Rank Kickoff.Return.Def.Rank Kickoff.Return.Rank
## 1                 44                      82                129
## 2                 74                       1                 53
## 3                 11                      11                 55
## 4                 94                      29                 18
## 5                  5                      95                 57
## 6                 80                      74                  7
```

Note that Ranked_Observations does not contain specific numbers of yardage, points, time, etc., but only a team's relative rank (1-130), vis-à-vis the other teams, for 24 performance metrics.

As any football fan will know, some of the key ranked metrics include:

- Scoring: number of points earned
- Rushing: how well a team runs the ball
- Passing: how well a team throws the ball
- Kickoff: how well a team kicks the ball
- First Down: how far a team moves forward on its first try
- 4th.Down: how far a team moves forward on its last try
- Redzone: how well a team performs when about to score
- Turnover: how often a team involuntarily surrenders ball possession
- Penalty: number of infractions
- Time of Possession: how well a team maintains ball control

Logistic Regression Model

Our first model uses logistic regression (LR), which is a statistical model (typically of a binary dependent variable) to determine event probabilities, such as whether a patient might be healthy or sick.

First, we split the data into a 'train' set (80%) and a 'test' set (20%).

```
set.seed(111)
ind <- sample(2, nrow(Ranked_Observations), replace = TRUE, prob = c(0.8, 0.2))
train <- Ranked_Observations[ind==1,]
test <- Ranked_Observations[ind==2,]
```

Here you can see the first six rows of each data set. Notice that the test set has been randomly chosen from the larger data set.

```
head(train [1:3])
```

```
##   Wins Off.Rank Def.Rank
## 1   11       51       17
## 2    0      130       83
## 3   11        6       20
## 4   13       39       26
## 5    4       30      120
## 6    8       94       67
```

```
head(test [1:3])
```

```
##    Wins Off.Rank Def.Rank
## 18    8      117       65
## 25    4       34       52
## 40    6      115       77
## 44    3       90      122
## 48    5       17      128
## 55    3      109      109
```

Even in this data snippet, you can see how teams with better offensive and defensive rankings typically have more wins.

LR Model

This code runs our LR model, with the wins predicted from all 24 Ranked_Observations.

```
LRM_model <- lm(Wins ~ ., data = train)
summary(LRM_model)
```

```
##
## Call:
## lm(formula = Wins ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9106 -0.6604  0.0999  0.6891  3.3754
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)              14.7394668  0.9922215  14.855  < 2e-16 ***
## Off.Rank                  0.0032552  0.0130846   0.249  0.80418
## Def.Rank                  0.0301596  0.0262001   1.151  0.25315
## First.Down.Rank          -0.0066245  0.0101463  -0.653  0.51572
## First.Down.Def.Rank       0.0372505  0.0067518   5.517 4.22e-07 ***
## X4th.Down.Rank           -0.0056390  0.0041795  -1.349  0.18112
## X4rd.Down.Def.Rank       -0.0102264  0.0040202  -2.544  0.01292 *
## Kickoff.Return.Def.Rank -0.0064198  0.0035894  -1.789  0.07752 .
## Kickoff.Return.Rank       0.0064328  0.0039388   1.633  0.10641
## Passing.Off.Rank         -0.0030008  0.0091313  -0.329  0.74330
## Pass.Def.Rank            -0.0320927  0.0131954  -2.432  0.01727 *
## Penalty.Rank             -0.0035743  0.0041124  -0.869  0.38740
## Punt.Return.Rank         -0.0027892  0.0035312  -0.790  0.43196
## Punt.Return.Def.Rank      0.0007976  0.0037643   0.212  0.83273
## Redzone.Def.Rank         -0.0026527  0.0040587  -0.654  0.51527
## Redzone.Off.Rank         -0.0014798  0.0047300  -0.313  0.75522
## Rushing.Def.Rank         -0.0377034  0.0169981  -2.218  0.02942 *
## Rushing.Off.Rank         -0.0013211  0.0074410  -0.178  0.85954
## Sack.Rank                -0.0110155  0.0059246  -1.859  0.06671 .
## Scoring.Def.Rank         -0.0247355  0.0114135  -2.167  0.03323 *
## Scoring.Off.Rank         -0.0252099  0.0102129  -2.468  0.01573 *
## Tackle.for.Loss.Rank     -0.0033434  0.0048852  -0.684  0.49574
## X3rd.Down.Rank           -0.0013108  0.0058445  -0.224  0.82312
## Time.of.Possession.Rank -0.0158368  0.0051563  -3.071  0.00292 **
## Turnover.Rank            -0.0030444  0.0046257  -0.658  0.51235
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.187 on 79 degrees of freedom
## Multiple R-squared:   0.89,  Adjusted R-squared:  0.8566
## F-statistic: 26.63 on 24 and 79 DF,  p-value: < 2.2e-16
```

LR calculated that the starred variables (noted to the right of the P values) were those that had the most influence in determining the number of wins.

The top two performance metrics were “First Down Defense” (how well a team stops the other team’s first offensive try) and “Time of Possession” (how long a team is able to maintain control of the football while on offense).

In particular, note that 4 of the 5 remaining starred metrics are defensive in nature.

To improve our LR model (for "LRM_model2"), we choose to leverage all 7 starred P values from above.

```
LRM_model2 <- lm(Wins ~ First.Down.Def.Rank + X4rd.Down.Def.Rank + Pass.Def.Rank + Rushing.Def.Rank + S
summary(LRM_model2)
```

```
##
## Call:
## lm(formula = Wins ~ First.Down.Def.Rank + X4rd.Down.Def.Rank +
##     Pass.Def.Rank + Rushing.Def.Rank + Scoring.Def.Rank + Scoring.Off.Rank +
##     Time.of.Possession.Rank, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0116 -0.8089  0.0409  0.8156  3.3859
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)              13.020936   0.374474  34.771  < 2e-16 ***
## First.Down.Def.Rank       0.036342   0.005115   7.105 2.11e-10 ***
## X4rd.Down.Def.Rank       -0.006383   0.003570  -1.788 0.076919 .
## Pass.Def.Rank            -0.021078   0.004952  -4.257 4.83e-05 ***
## Rushing.Def.Rank         -0.029083   0.007468  -3.895 0.000182 ***
## Scoring.Def.Rank         -0.025735   0.007776  -3.309 0.001317 **
## Scoring.Off.Rank         -0.035860   0.003572 -10.038  < 2e-16 ***
## Time.of.Possession.Rank -0.011752   0.004033  -2.914 0.004445 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.196 on 96 degrees of freedom
## Multiple R-squared:  0.8643, Adjusted R-squared:  0.8544
## F-statistic: 87.33 on 7 and 96 DF,  p-value: < 2.2e-16
```

Thus, LRM_model2 seems to affirm the college football adage that "offense may sell tickets, but defense wins championships".

Next, we use our new model to guess the number of wins each team will have in the train data:

```
LRM_predictions_train <- predict(LRM_model2, train, type = 'response')
head(LRM_predictions_train [1:7])
```

```
##         1         2         3         4         5         6
## 10.497367  1.436406 11.867583 11.407014  4.840217  7.138075
```

This Confusion Matrix shows three things about LR's predicted win total for each football team:

1. Incorrect guesses (top row)
2. Correct guesses (bottom row)
3. Total guesses (sum of both rows)

```
LRM_Conf_train <- ifelse(round(LRM_predictions_train) == train$Wins, 1, 0)
LRM_Conf_Tab_train <- table(Predicted = LRM_Conf_train, Actual = train$Wins)
LRM_Conf_Tab_train
```

```
##           Actual
## Predicted  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##         0  1  2  1  4  9  9  6 10 12  1  3  8  3  1  1  1
##         1  0  0  3  1  4  2  2  2  8  2  2  4  2  0  0  0
```

Here is the percentage of wins LR correctly predicted in the train data.

```
options(digits = 5)
sum(LRM_Conf_Tab_train[2,])/sum(LRM_Conf_Tab_train)
```

```
## [1] 0.30769
```

We achieved a significant improvement: 30.77%.

This is far better than the maximum 17.69% that we could have achieved by randomly guessing.

Next, let's try our LR model against the test data:

```
LRM_predictions_test <- predict(LRM_model2, test, type = 'response')
head(LRM_predictions_test [1:7])
```

```
##     18     25     40     44     48     55
## 6.6610 4.5939 6.0813 3.4499 5.3737 3.4957
```

This Confusion Matrix shows the results:

```
LRM_Conf_test <- ifelse(round(LRM_predictions_test) == test$Wins, 1, 0)
LRM_Conf_Tab_test <- table(Predicted = LRM_Conf_test, Actual = test$Wins)
LRM_Conf_Tab_test
```

```
##           Actual
## Predicted 1 2 3 4 5 6 7 8 9 10 13
##         0 1 2 1 4 2 1 2 3 1  2  0
##         1 0 0 2 1 1 1 0 0 1  0  1
```

Here is our percentage score against the test data:

```
sum(LRM_Conf_Tab_test[2,])/sum(LRM_Conf_Tab_test)
```

```
## [1] 0.26923
```

For the test data, we achieved a somewhat lower result: 26.92%. However, this is still a tangible improvement over blind guessing.

K-Nearest Neighbors Model

Our second model uses the K-Nearest Neighbors (KNN) machine learning algorithm, which examines all known cases and classifies new cases based on a similarity measure such as a distance function, ultimately choosing the new classification based on a plurality vote.

We first split our data set into a 'train' set (80%) and a 'test' set (20%):

```
set.seed(222)
ind2 <- sample(2, nrow(Ranked_Observations), replace = TRUE, prob = c(0.8, 0.2))
train2 <- Ranked_Observations[ind2==1,]
test2 <- Ranked_Observations[ind2==2,]
```

The test data is randomly chosen from the complete data set.

```
head(train2 [1:3])
```

```
##   Wins Off.Rank Def.Rank
## 2    0      130       83
## 3   11        6       20
## 4   13       39       26
## 7    2      111      110
## 8    8       33      124
## 9    5       89       30
```

```
head(test2 [1:3])
```

```
##    Wins Off.Rank Def.Rank
## 1    11       51       17
## 5     4       30      120
## 6     8       94       67
## 15    3      120      117
## 21   11       80       40
## 26    5      114       56
```

Again, you can see that teams with better performance metrics tend to have more wins.

We will leverage that demonstrated relationship in building our KNN model, which will try to find the ideal combination of metrics for the most accurate predictive capability.

Here is the code to build our KNN model.

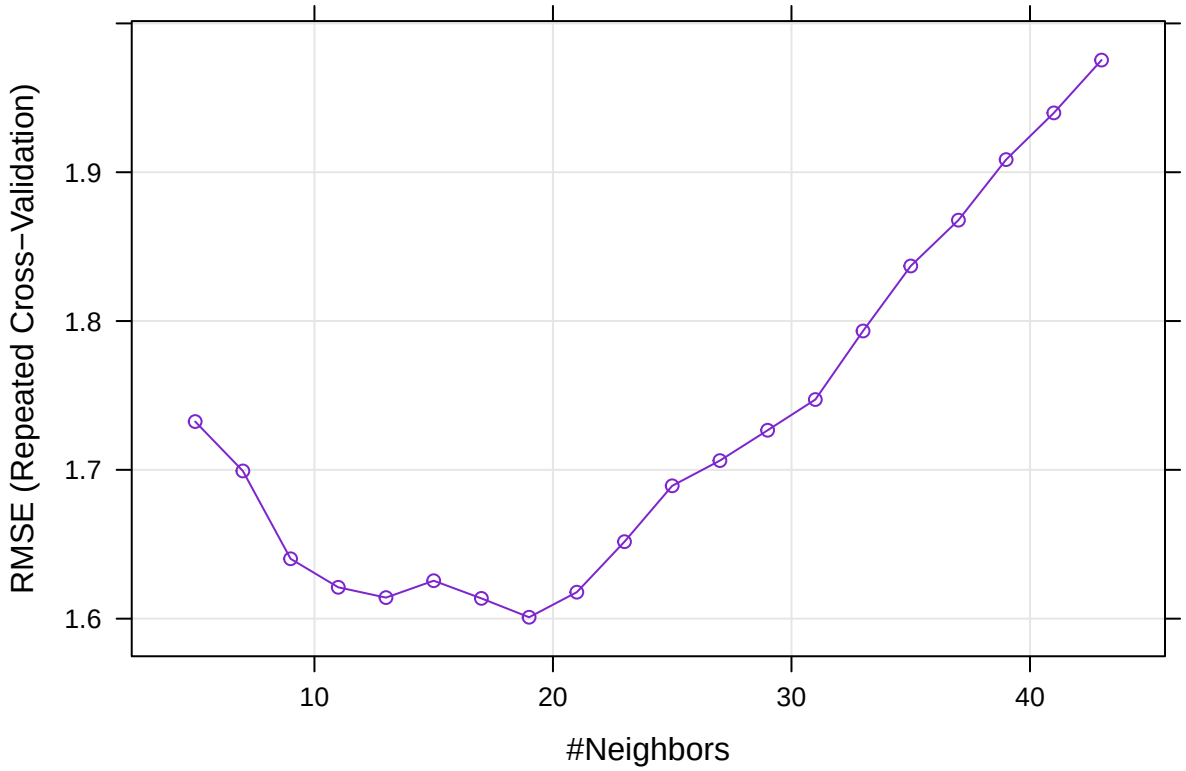
```
trControl <- trainControl(method = "repeatedcv",
                          number = 10,
                          repeats = 3)
set.seed(2222)
KNN_model <- train(Wins ~ .,
                   data = train2,
                   method = 'knn',
                   tuneLength = 20,
                   trControl = trControl,
                   preProc = c("center", "scale"))
KNN_model
```

```
## k-Nearest Neighbors
##
## 104 samples
##  24 predictor
##
## Pre-processing: centered (24), scaled (24)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 94, 95, 93, 93, 94, 93, ...
## Resampling results across tuning parameters:
##
##   k   RMSE    Rsquared  MAE
##    5  1.7325  0.73419   1.4326
##    7  1.6992  0.75118   1.4204
##    9  1.6403  0.78032   1.3686
##   11  1.6211  0.78816   1.3334
##   13  1.6141  0.78800   1.3253
##   15  1.6255  0.78816   1.3505
##   17  1.6136  0.79248   1.3534
##   19  1.6009  0.80643   1.3544
##   21  1.6178  0.80986   1.3680
##   23  1.6517  0.80646   1.3891
##   25  1.6893  0.80252   1.4225
##   27  1.7062  0.80634   1.4305
##   29  1.7266  0.80743   1.4450
##   31  1.7472  0.81015   1.4634
##   33  1.7933  0.80359   1.4965
##   35  1.8370  0.79615   1.5286
##   37  1.8678  0.79749   1.5507
##   39  1.9085  0.79335   1.5871
##   41  1.9399  0.79311   1.6144
##   43  1.9754  0.79160   1.6444
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 19.
```

KNN selected the smallest Root Mean Square Error (RMSE), at k = 19, for its optimal model.

Here is a visualization of the RMSE selection process.

```
plot(KNN_model, col = "Purple3")
```



Below, you can see that the RMSEs in both the train and test sets are quite similar.

```
KNN_predict_train <- predict(KNN_model, newdata = train2)
KNN_predict_test <- predict(KNN_model, newdata = test2)
RMSE(KNN_predict_train, train2$Wins)
```

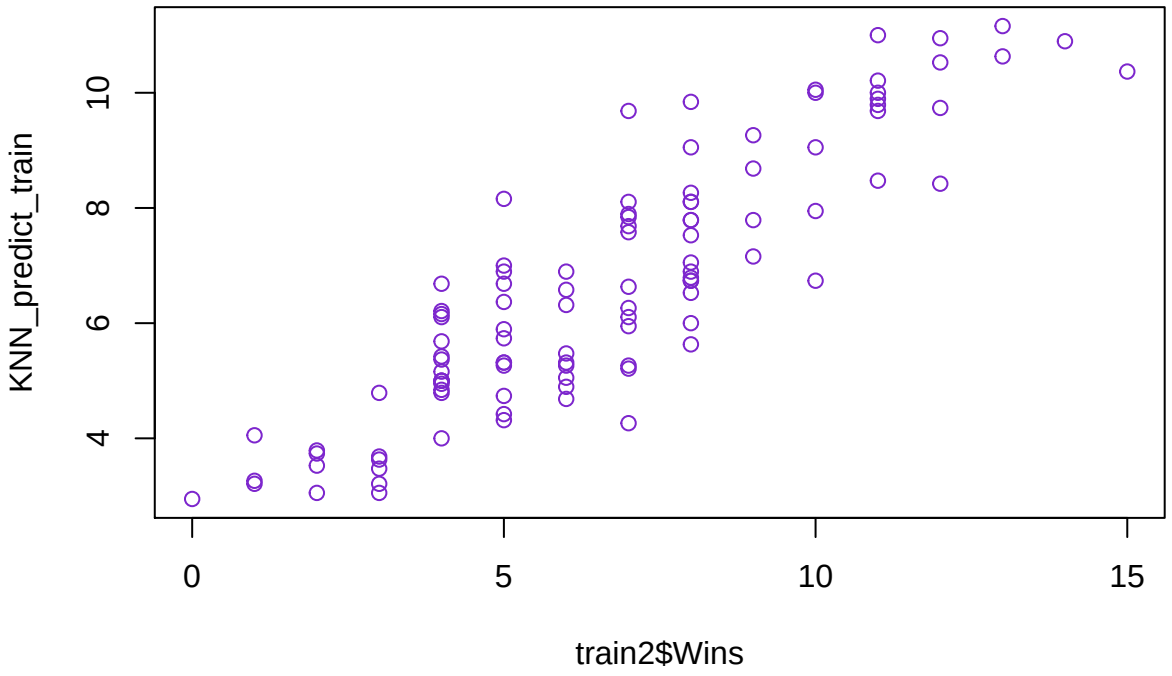
```
## [1] 1.5586
```

```
RMSE(KNN_predict_test, test2$Wins)
```

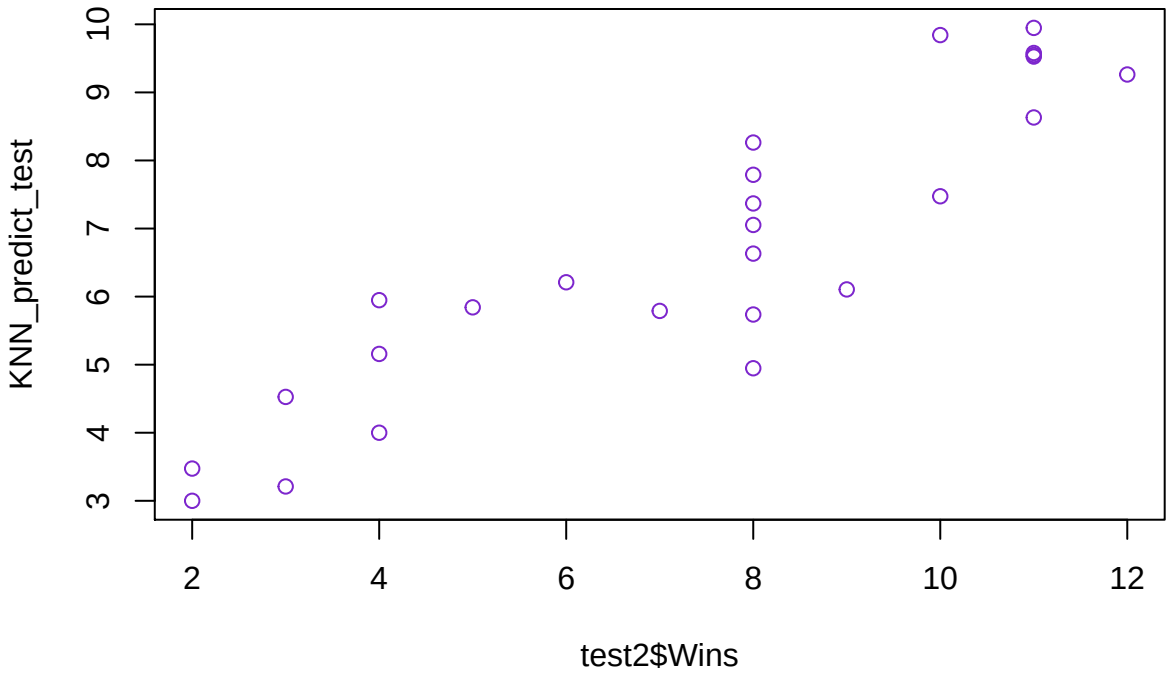
```
## [1] 1.5922
```

These two charts show the similarity of the KNN predictive capability in both the train and test sets.

```
plot(KNN_predict_train ~ train2$Wins, col = "Purple3")
```




```
plot(KNN_predict_test ~ test2$Wins, col = "Purple3")
```



In the table below, KNN calculates variable importance, sorting the top 20 in descending order.

```
varImp(KNN_model)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 24)
##
##                          Overall
## Scoring.Off.Rank           100.0
## Scoring.Def.Rank            97.3
## First.Down.Rank             97.2
## X3rd.Down.Rank              71.7
## Def.Rank                    70.8
## Rushing.Def.Rank            70.2
## Off.Rank                    55.9
## Sack.Rank                   54.8
## Turnover.Rank               46.3
## Redzone.Off.Rank            42.4
## Redzone.Def.Rank            33.3
## Passing.Off.Rank            28.9
## Tackle.for.Loss.Rank        26.9
## Pass.Def.Rank               21.9
## X4th.Down.Rank              21.1
## Rushing.Off.Rank            20.7
## Time.of.Possession.Rank     20.1
## X4rd.Down.Def.Rank          18.5
## Kickoff.Return.Def.Rank     18.2
## Kickoff.Return.Rank         11.8
```

Two interesting aspects immediately jump out.

First, score is naturally the ultimate deciding factor. However, we must dig deeper into the list to understand exactly how the final game scores were achieved. Further, it is worth noting that our first LR model rated "First Down Defense" and "Time of Possession" as even more important than scoring.

Second, we find the LR and the KNN models seem to have different philosophical approaches to winning football games. Remember that for LR, 5 of the top 7 statistics were on defense. KNN, however, appears to favor offense, with "First Down Rank" in third place, just after the two scoring metrics. And "Third Down Rank" (representing a team's typical penultimate offensive attempt to gain a new first down) is fourth. Thus, KNN strongly recognizes the value of a first down, which after all represents the latent potential of *four* downs (after which the other team typically gains possession).

This difference in approach (between LR and KNN) could be the subject of further research.

Now let's see how well KNN is able to predict a team's win total for the season. Here is the code for our train data.

```
KNN_CM_train <- predict(KNN_model, train2, type = 'raw')
head(KNN_CM_train [1:9])
```

```
## [1]  2.9474 11.0000 10.6316  3.5263  6.0000  6.8947
```

Here is the KNN Confusion Matrix for the train data:

```
KNN_CM_train2 <- ifelse(round(KNN_CM_train) == train2$Wins, 1, 0)
KNN_CM_train3 <- table(Predicted = KNN_CM_train2, Actual = train2$Wins)
KNN_CM_train3
```

```
##           Actual
## Predicted  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##         0  1  3  4  3 14  9  8 12 10  2  3  6  4  2  1  1
##         1  0  0  0  3  1  4  1  1  6  2  2  1  0  0  0  0
```

How well did it work?

```
options(digits = 5)
sum(KNN_CM_train3[2,])/sum(KNN_CM_train3)
```

```
## [1] 0.20192
```

KNN's predictive capability in the train data was 20.19% - far short of LR's 30.77%.

Now let's run KNN against the test data.

```
KNN_CM_test <- predict(KNN_model, test2, type = 'raw')
KNN_CM_test
```

```
##  [1] 8.6316 5.9474 6.6316 3.2105 9.5263 5.8421 9.5500 5.7895 9.8421 7.7895
## [11] 9.9474 5.7368 4.9474 6.2105 9.5789 9.2632 7.3684 5.1579 4.5263 3.0000
## [21] 7.4737 6.1053 3.4737 4.0000 7.0526 8.2632
```

Here is the KNN Confusion Matrix for the test data:

```
KNN_CM_test2 <- ifelse(round(KNN_CM_test) == test2$Wins, 1, 0)
KNN_CM_test3 <- table(Predicted = KNN_CM_test2, Actual = test2$Wins)
KNN_CM_test3
```

```
##           Actual
## Predicted 2 3 4 5 6 7 8 9 10 11 12
##         0 2 1 2 1 0 1 5 1  1  5  1
##         1 0 1 1 0 1 0 2 0  1  0  0
```

And the KNN percentage for the test data.

```
options(digits = 5)
sum(KNN_CM_test3[2,])/sum(KNN_CM_test3)
```

```
## [1] 0.23077
```

The final result against the test set is 23.08%.

This is better than blindly guessing, but not as good as LR (26.92%).

Random Forest Model

Our third and final model employs the Random Forest (RF) method, which works by creating decision trees that provide results in the form of classification or prediction.

First we split the data into a 'train' set (80%) and a 'test' set (20%).

```
set.seed(333)
ind3 <- sample(2, nrow(Ranked_Observations), replace = TRUE, prob = c(0.8, 0.2))
train3 <- Ranked_Observations[ind3==1,]
test3 <- Ranked_Observations[ind3==2,]
```

The split data sets are shown below.

The test set is randomly chosen from the complete data set.

```
head(train3 [1:3])
```

```
##   Wins Off.Rank Def.Rank
## 1   11       51       17
## 2    0      130       83
## 4   13       39       26
## 5    4       30      120
## 6    8       94       67
## 7    2      111      110
```

```
head(test3 [1:3])
```

```
##    Wins Off.Rank Def.Rank
## 3    11        6       20
## 13   12       48       33
## 16    8       82        7
## 18    8      117       65
## 30   11       23       50
## 37    3      127       89
```

Again, you can see that better on-field performance typically leads to a higher number of wins.

This code creates our RF model:

```
rf <- randomForest(Wins ~ ., data = train3)
print(rf)
```

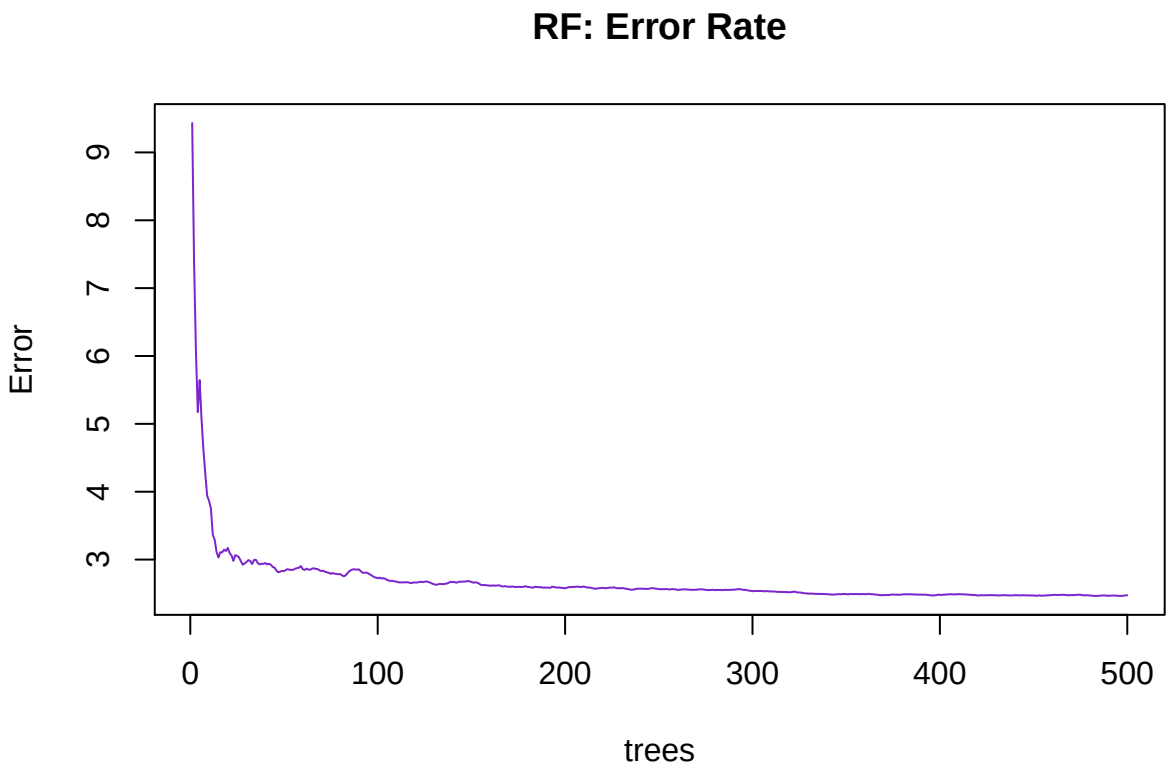
```
##
## Call:
##  randomForest(formula = Wins ~ ., data = train3)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 2.4723
##                     % Var explained: 73.24
```

It is a regression RF with 500 trees.

The number of variables attempted at each split was 8.

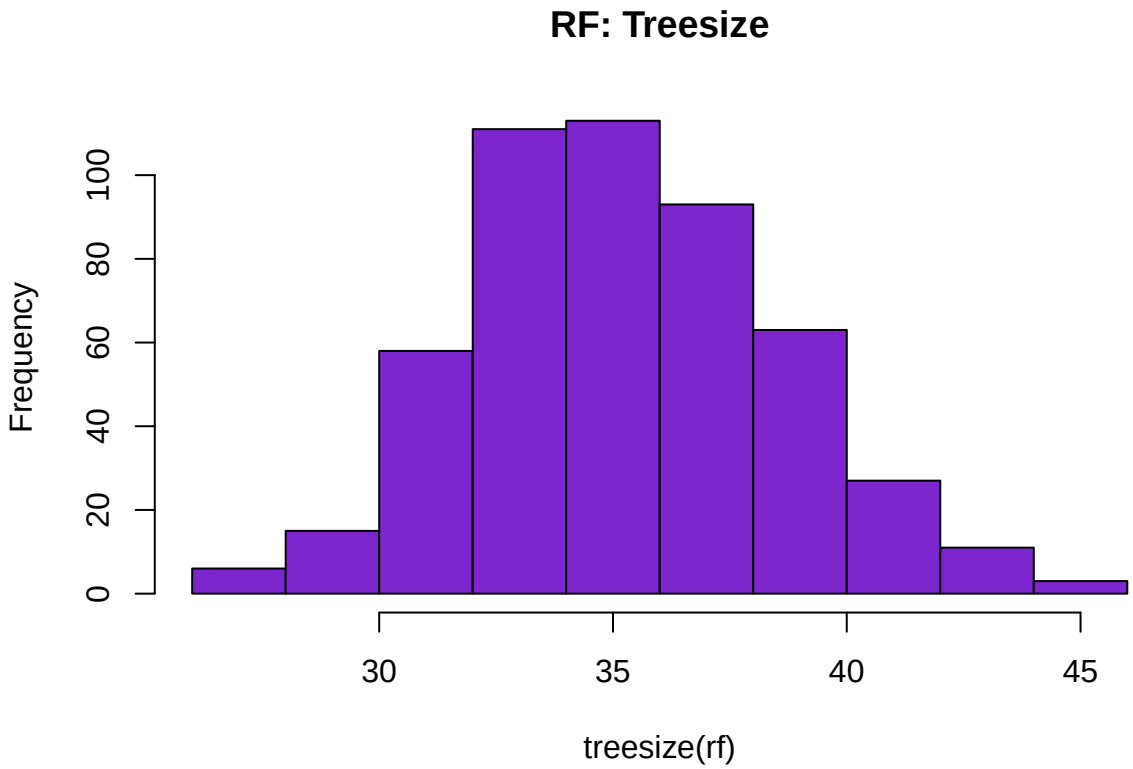
The model's error rate can be seen in the plot below.

```
plot(rf, main = "RF: Error Rate", col = "Purple3")
```



Here is a histogram of the treesize, or number of nodes.

```
hist(treesize(rf), main = "RF: Treesize", col = "Purple3")
```



Here are the model's list of attributes, such as "importance", which is the extractor function for variable importance measures as produced by randomForest.

```
attributes(rf)
```

```
## $names
##  [1] "call"            "type"            "predicted"       "mse"
##  [5] "rsq"             "oob.times"       "importance"      "importanceSD"
##  [9] "localImportance" "proximity"       "ntree"           "mtry"
## [13] "forest"          "coefs"           "y"               "test"
## [17] "inbag"           "terms"
##
## $class
## [1] "randomForest.formula" "randomForest"
```

Let's run the importance function, which provides a table of node "purity" within the model.

```
importance(rf)
```

```
##                           IncNodePurity
## Off.Rank                        32.8925
## Def.Rank                        84.7160
## First.Down.Rank                120.5030
## First.Down.Def.Rank             12.9646
## X4th.Down.Rank                  12.3383
## X4rd.Down.Def.Rank               8.3194
## Kickoff.Return.Def.Rank          7.2000
## Kickoff.Return.Rank              8.2662
## Passing.Off.Rank                 7.9238
## Pass.Def.Rank                   15.7526
## Penalty.Rank                     7.5655
## Punt.Return.Rank                 6.7077
## Punt.Return.Def.Rank             6.5755
## Redzone.Def.Rank                11.6126
## Redzone.Off.Rank                22.4696
## Rushing.Def.Rank                59.5110
## Rushing.Off.Rank                 8.8061
## Sack.Rank                       34.0468
## Scoring.Def.Rank               164.4578
## Scoring.Off.Rank               180.1863
## Tackle.for.Loss.Rank            16.2058
## X3rd.Down.Rank                  55.6519
## Time.of.Possession.Rank          6.5523
## Turnover.Rank                   16.0698
```

Compared to LR and KNN, RF has chosen a hybrid approach to winning football games.

Whereas LR favored defense, and KNN favored offense, RF has chosen a more balanced mix of both.

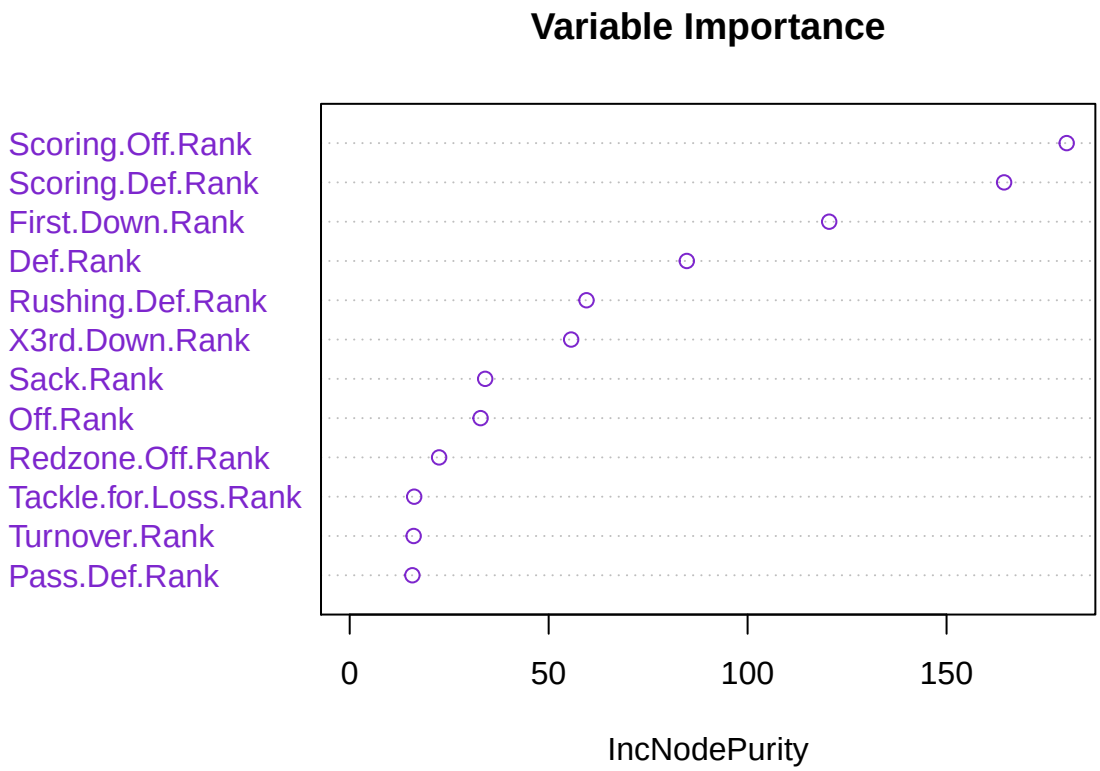
After scoring, RF picked the following performance metrics as its most influential:

1. First.Down.Rank (OFF)
2. Def.Rank (DEF)
3. Rushing.Def.Rank (DEF)
4. X3rd.Down.Rank (OFF)
5. Sack.Rank (DEF)
6. Off.Rank (OFF)
7. Redzone.Off.Rank (OFF)
8. Tackle.for.Loss.Rank (DEF)

Note that the top 8 variables are evenly split between offense and defense.

Let's visualize RF's choices in a dotchart.

```
varImpPlot(rf, n.var=12, main = "Variable Importance", col = "Purple3")
```



You can see that, as with KNN, the first down metric is third-most important in the RF model. However, 3 of the next 4 metrics are defensive in nature.

Now it is time to see how well RF's hybrid approach can predict the number of a team's wins.

```
RF_CM_train <- predict(rf, train3, type = 'response')
head(RF_CM_train [1:9])
```

```
##       1       2       4       5       6       7
## 10.5328  1.2705 11.9973  4.7076  7.5381  2.4653
```

Here is the Confusion Matrix for the RF train set:

```
RF_CM_train2 <- ifelse(round(RF_CM_train) == train3$Wins, 1, 0)
RF_CM_train3 <- table(Predicted = RF_CM_train2, Actual = train3$Wins)
RF_CM_train3
```

```
##           Actual
## Predicted  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
##         0  1  2  1  2  9  5  0  3  6  2  4  4  3  1  1
##         1  0  0  5  4  9  6  7  8 11  2  0  4  1  0  0
```

And the percentage of correct predictions for the train set:

```
options(digits = 5)
sum(RF_CM_train3[2,])/sum(RF_CM_train3)
```

```
## [1] 0.56436
```

Wow: this is our best prediction by far: 56.44%!

Surely this predictive capability cannot hold for the test data. Otherwise, we could get rich quickly by betting real money on this algorithm.

Let's see whether this success can be duplicated in the test set.

```
RF_CM_test <- predict(rf, test3, type = 'response')
head(RF_CM_test [1:9])
```

```
##       3      13      16      18      30      37
## 11.5915 10.5494  9.5321  6.5373  9.7568  2.3168
```

Here is the Confusion Matrix for the RF test set:

```
RF_CM_test2 <- ifelse(round(RF_CM_test) == test3$Wins, 1, 0)
RF_CM_test3 <- table(Predicted = RF_CM_test2, Actual = test3$Wins)
RF_CM_test3
```

```
##           Actual
## Predicted 1 3 5 6 7 8 9 10 11 12 13 15
##         0 1 2 1 3 2 6 0  2  3  1  1  1
##         1 0 0 2 0 1 0 1  1  1  0  0  0
```

And our final percentage:

```
options(digits = 5)
sum(RF_CM_test3[2,])/sum(RF_CM_test3)
```

```
## [1] 0.2069
```

Unfortunately, the extremely high percentage we achieved in the train set did not hold true for the test set.

Here, we tallied a mere 20.69%, which is in fact worse than both LR and KNN.

At the time of writing, it is unclear to this author why there should be such a large discrepancy.

Perhaps it is due to RF's hybrid approach to prediction, but more likely there is simply an error in coding, or a bias in the train set.

To gather a bit more data on this, let's check the RMSE of both the train set and test set.

```
RF_predict_train <- predict(rf, newdata = train3)
RF_predict_test <- predict(rf, newdata = test3)
RMSE(RF_predict_train, train3$Wins)
```

```
## [1] 0.637
```

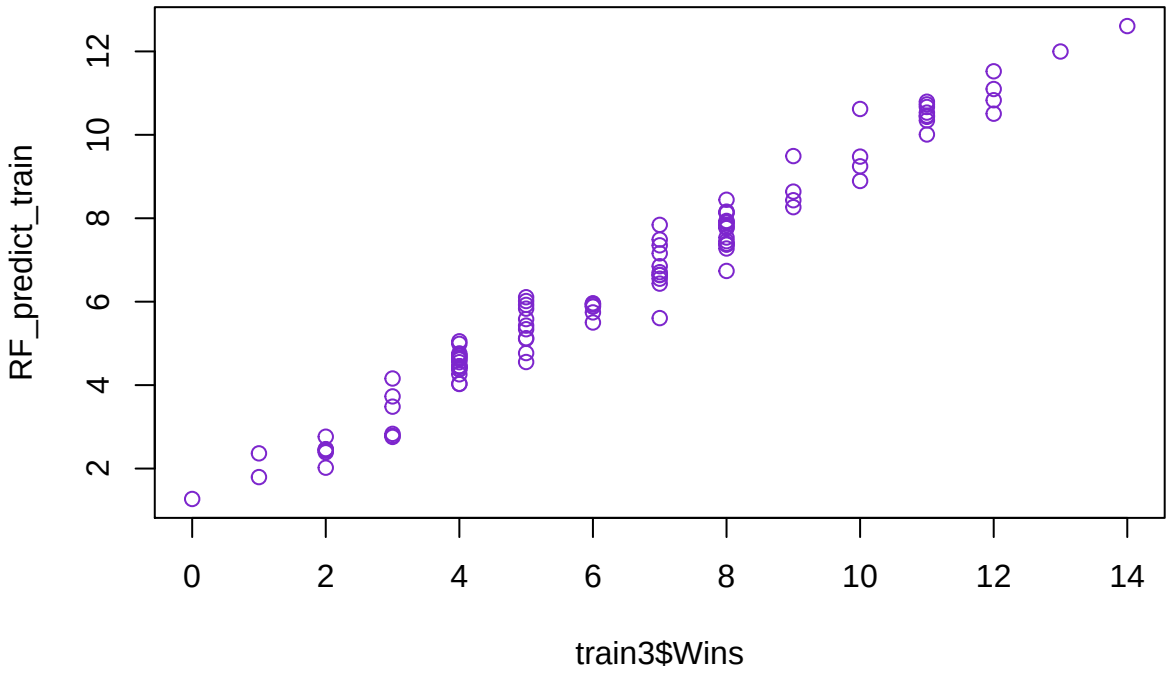
```
RMSE(RF_predict_test, test3$Wins)
```

```
## [1] 1.3444
```

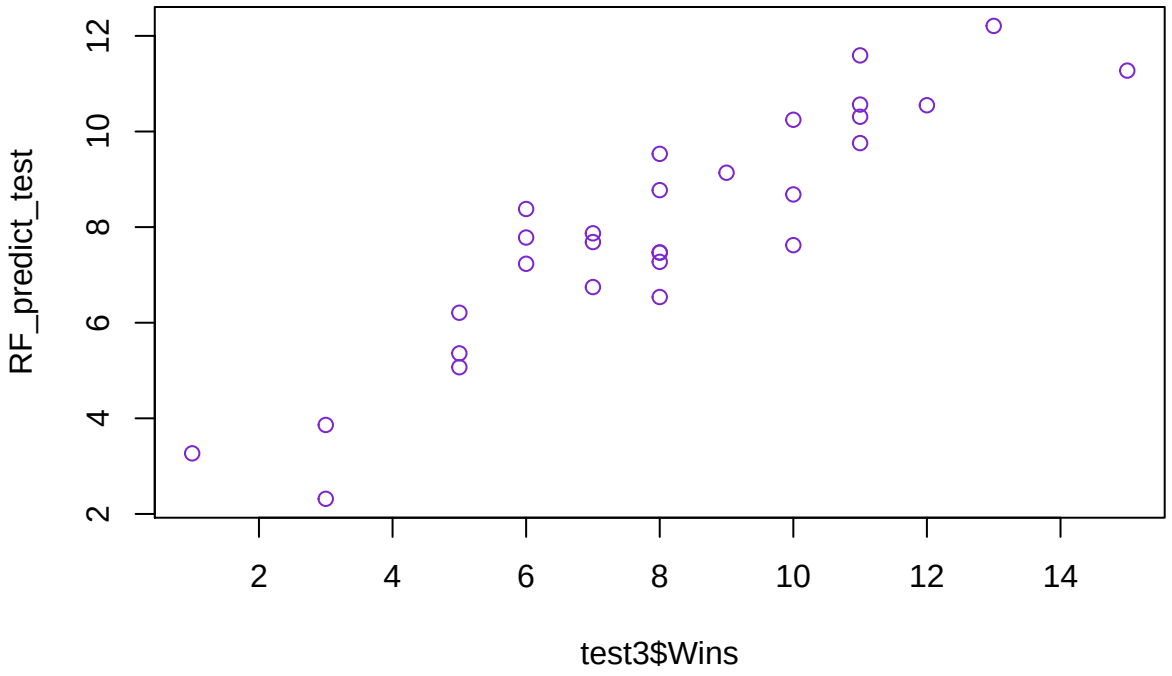
Indeed. Unlike for the KNN model, these numbers are quite far apart.

We can plot the RMSE disparity thus:

```
plot(RF_predict_train ~ train3$Wins, col = "Purple3")
```




```
plot(RF_predict_test ~ test3$Wins, col = "Purple3")
```



The first plot shows a rather tight group of data points, while the second shows a much greater room for error.

Conclusion

What Did We Learn?

In this paper, we used R code and machine learning to predict how many wins all 130 American college football teams would have in a season.

We built three models: Logistic Regression (LR), K-Nearest Neighbors (KNN), and Random Forest (RF).

The best prediction we could make via blind guessing was 17.69%.

Therefore, our goal was twofold: first, to achieve a better percentage than we could via guessing; and second, to choose the best overall model.

Here are the final results of all three models against the test data:

- LR: 26.92%
- KNN: 23.08%
- RF: 20.69%

In this initial analysis, LR provided the best predictive capability, with KNN second, and RF third. In terms of algorithmic choices, there were some interesting differences.

LR placed its emphasis on defense, which not only put it in first place among the algorithms, but also affirmed the college football adage that "offense may sell tickets, but defense wins championships".

KNN appeared to favor an offensive approach to winning, and easily took second place. RF took a hybrid approach, which might initially seem to be a smart move, but in fact landed it in last place.

In the future, it is possible that other models, or even fine tuning these models, will yield different results. There are many more models and variables to consider, and so far, we have only scratched the surface.