```
#List of packages needed
library(ggplot2)
library(tibble)
library(tidyr)
library(skimr)
library(tidyverse)
library(DataExplorer)
library(dplyr)
library(rpart)
library(rpart.plot)
library(caret)
library (neuralnet)
library(nnet)
library(stargazer)

# Need to import the data
data <- read.csv("high_diamond_ranked_10min.csv")

#Taking out the gameId
data <- data[,-1]

#Turning Blue Side Wins into a factor variable
data$blueWinsD <- factor(data$blueWins)

#Adding a variable for the difference of minions killed between the teams
data$blueMinionDiff <- data$blueTotalMinionsKilled - data$redTotalMinionsKilled

#Focusing on the Blue Side (Removing the 19 red-related variables)
blue <- data%>%select(-c('redWardsPlaced':'redGoldPerMin'))

#Eliminating columns that mean the same as others (blueCSPerMin * 10 =
blueTotalMinionsKilled, etc...)
blue <- blue %>% select (-blueCSPerMin, -blueGoldPerMin, -blueTotalExperience)

#Understanding the variables
skim_without_charts(blue)
summary(blue)
bhead(blue)

#Plotting and then eliminating the games with too many wards placed
histogram(data1$blueWardsPlaced)
wards <- mean(blue$blueWardsPlaced) + (3 * sd(blue$blueWardsPlaced))
blue <- filter(blue, blue$blueWardsPlaced <= wards)

#Understanding the variables again after changes
skim_without_charts(blue)
```

```
bluesummary <- summary(blue)
stargazer(blue, type = "html", out = "summary.html")
bhead(blue)
create_report(blue)


#GRAPHS ------------------------------------

#Correlation between TotalGold against GoldDiff
ggplot(data = blue) + geom_jitter(mapping = aes(x = blueTotalGold, y = blueGoldDiff)) +
geom_smooth(mapping = aes(x = blueTotalGold, y = blueGoldDiff), se = T)
cor(blue$blueTotalGold, blue$blueGoldDiff)

#Does GoldDiff work as a strong predictor?
ggplot(blue, aes(x=blueGoldDiff, fill = blueWinsD)) + geom_density(alpha = 0.5) +
xlab("Difference in Gold")+ ylab("Density") + theme(plot.title = element_text(hjust = 0.5))

#Relationship between TotalMinionsKilled and GoldDiff
ggplot(blue) + geom_jitter(mapping = aes(x = blueTotalMinionsKilled, y = blueGoldDiff, color
= blueWins), alpha = 0.6) +
  scale_color_gradient(low = "10", high = "20") + ggtitle("Scatterplot of Total Minions and Gold
Difference") + xlab("Total Minions Killed") +
  ylab("Gold Difference") + theme(plot.title = element_text(hjust = 0.5))

#What about MinionDiff and GoldDiff?
ggplot(blue) + geom_jitter(mapping = aes(x = blueMinionDiff, y = blueGoldDiff, color =
blueWins), alpha = 0.6) + scale_color_gradient(low = "10", high = "20")

ggplot(blue) + geom_jitter(mapping = aes(x = as.factor(blueHeralds) , y = blueGoldDiff, color =
blueWins), alpha = 0.6) +
  ggtitle("Gold Difference per Number of Heralds") + xlab("Number of Heralds")+ ylab("Gold
Difference") +
  scale_color_gradient(low = "10", high = "20") + theme(plot.title = element_text(hjust = 0.5)) +
  geom_hline(yintercept = -172) + geom_hline(yintercept = 832, linetype = "dashed")

#Checking if killing the Herald yields statistically significant results in Total Gold:
blue %>%
  group_by(as.factor(blueHeralds)) %>%
  summarise(average = mean(blueTotalGold))
t.test(blueTotalGold ~ as.factor(blueHeralds), data = blue)
wilcox.test(blueTotalGold ~ as.factor(blueHeralds), data = blue)

#Checking if killing the Herald yields statistically significant results in GoldDiff:
blue %>%
  group_by(as.factor(blueHeralds)) %>%
  summarise(average = mean(blueGoldDiff))
```

```
t.test(blueGoldDiff ~ as.factor(blueHeralds), data = blue)
wilcox.test(blueGoldDiff ~ as.factor(blueHeralds), data = blue)


#DECISION TREES ------------

#Eliminate the numerical blueWins variable
blue_dt <- blue%>%select(-1)

#Change categorical variables to Factors
blue_dt$blueFirstBlood <- factor(blue_dt$blueFirstBlood)
blue_dt$blueEliteMonsters <- factor(blue_dt$blueEliteMonsters)
blue_dt$blueDragons <- factor(blue_dt$blueDragons)
blue_dt$blueHeralds <- factor(blue_dt$blueHeralds)
blue_dt$blueTowersDestroyed <- factor(blue_dt$blueTowersDestroyed)

#Partition the data
blue_dt_index <- createDataPartition(blue_dt$blueWinsD, p = .3, list = FALSE)
blue_dt_test <- blue_dt[blue_dt_index,]
blue_dt_train <- blue_dt[-blue_dt_index,]

#Create the Decision Tree (dt)
blue_dt_0 <- rpart(formula = blueWinsD ~ ., data = blue_dt_train, method = "class")
summary(blue_dt_0, "cp")
dt_0 <- summary(blue_dt_0)

#Plot the Decision Tree
rpart.plot(blue_dt_0)
plotcp(blue_dt_0, upper = "splits")

#Predict using the train data set
predict_train0 <- predict(blue_dt_0, blue_dt_train, type = "class")
table(predict_train0)

#Confusion Matrix
blue_dt_train0_CM <- table(blue_dt_train$blueWinsD, predict_train0)
confusionMatrix(blue_dt_train0_CM)

#Predict using the test data set
predict_test0 <- predict(blue_dt_0, blue_dt_test, type = "class")
table(predict_test0)

#Confusion Matrix
blue_dt_test0_CM <- table(blue_dt_test$blueWinsD, predict_test0)
confusionMatrix(blue_dt_test0_CM)
```

```
#The decision tree only has two leaves, what happens when we force it to be bigger?
blue_dt_1 <- rpart(formula = blueWinsD ~ ., data = blue_dt_train, method = "class", control =
rpart.control(minbucket = 300, cp=0,001))
rpart.plot(blue_dt_1)

#Plot the Decision Tree 2
rpart.plot(blue_dt_1)
plotcp(blue_dt_1, upper = "splits")

#Predict using the train data set
predict_train1 <- predict(blue_dt_1, blue_dt_train, type = "class")
table(predict_train1)

#Confusion Matrix
blue_dt_train1_CM <- table(blue_dt_train$blueWinsD, predict_train1)
confusionMatrix(blue_dt_train1_CM)

#Predict using the test data set
predict_test1 <- predict(blue_dt_1, blue_dt_test, type = "class")
table(predict_test1)

#Confusion Matrix
blue_dt_test1_CM <- table(blue_dt_test$blueWinsD, predict_test1)
confusionMatrix(blue_dt_test1_CM)

# 10-Fold Cross Validation for the Decision Tree
trainParameters <- trainControl(method = "cv", number = 10, savePredictions = "final")
blue_dt_kfold <- train(blueWinsD ~ ., method = "rpart", trControl = trainParameters, data =
blue_dt_train, na.action = na.omit)
rpart.plot(blue_dt_kfold$finalModel)

#Predict using the train data set
predict_train2 <- predict(blue_dt_kfold, blue_dt_train, type = "raw")
table(predict_train2)

#Confusion Matrix
blue_dt_train2_CM <- table(blue_dt_train$blueWinsD, predict_train2)
confusionMatrix(blue_dt_train2_CM)

#Predict using the test data set
predict_test2 <- predict(blue_dt_kfold, blue_dt_test, type = "raw")
table(predict_test2)

#Confusion Matrix
blue_dt_test2_CM <- table(blue_dt_test$blueWinsD, predict_test2)
confusionMatrix(blue_dt_test2_CM)
```

```
#NEURAL NETWORKS
# --------------

#Transform the needed columns into factors and scaling the other columns
factor_columns <- c("blueWinsD", "blueFirstBlood", "blueEliteMonsters", "blueDragons",
"blueHeralds", "blueTowersDestroyed")
notfactor_columns <- c("blueWardsPlaced", "blueWardsDestroyed", "blueKills", "blueDeaths",
"blueAssists", "blueTotalGold", "blueAvgLevel", "blueTotalMinionsKilled",
"blueTotalJungleMinionsKilled", "blueGoldDiff", "blueExperienceDiff", "blueMinionDiff")
blue_nn <- data.frame(blue[,factor_columns], scale(blue[,notfactor_columns]))
blue_nn[,factor_columns] <- lapply(blue_nn[,factor_columns], as.factor)

#Partition the data
blue_nn_index <- createDataPartition(blue_nn$blueWinsD, p = .3, list = FALSE)
blue_nn_test <- blue_nn[blue_nn_index,]
blue_nn_train <- blue_nn[-blue_nn_index,]

#Create the Model Matrix due to factor variables
blue_nn_trainMM <- model.matrix(~ ., blue_nn_train)
blue_nn_trainMM <- blue_nn_trainMM[,-1]
blue_nn_testMM <- model.matrix(~ .-blueWinsD, blue_nn_test)
blue_nn_testMM <- blue_nn_testMM[,-1]

#Estimating the model
blue_nn_0 <- neuralnet(blueWinsD1 ~ ., blue_nn_trainMM, linear.output = FALSE, hidden =
c(2,1), threshold = 0.02)
plot(blue_nn_0)
blue_nn_0$result.matrix

#Creating a new dataframe and storing the predicted values
blue_nn_CM <- data.frame(predicted = blue_nn_0$net.result[[1]], actual =
blue_nn_train$blueWinsD)
blue_nn_CM$predicted <- as.numeric(blue_nn_CM$predicted > (1 -0.5))
blue_nn_CM <- table(blue_nn_CM)
confusionMatrix(blue_nn_CM)

#Predicting from the test data set
predict_nn0 <- as.numeric(predict(blue_nn_0, blue_nn_testMM) > (1 - 0.5))
predict_nn0_CM <- table(predict_nn0, blue_nn_test$blueWinsD)
confusionMatrix(predict_nn0_CM)

#Estimating a new model with more nodes
blue_nn_1 <- neuralnet(blueWinsD1 ~ ., blue_nn_trainMM, linear.output = FALSE, hidden =
c(3,3), threshold = 0.1)
plot(blue_nn_1)
```

```
blue_nn_1$result.matrix

#Creating a new dataframe and storing the predicted values
blue_nn_CM1 <- data.frame(predicted = blue_nn_1$net.result[[1]], actual =
blue_nn_train$blueWinsD)
blue_nn_CM1$predicted <- as.numeric(blue_nn_CM1$predicted > (1 -0.5))
blue_nn_CM1 <- table(blue_nn_CM1)
confusionMatrix(blue_nn_CM1)

#Predicting from the test data set
predict_nn1 <- as.numeric(predict(blue_nn_1, blue_nn_testMM) > (1 - 0.5))
predict_nn1_CM <- table(predict_nn1, blue_nn_test$blueWinsD)
confusionMatrix(predict_nn1_CM)

#USE K-FOLD CROSS VALIDATION (we need nnet as neuralnet only allows for regressions)

trainParameters_nn <- trainControl(method = "cv", number = 4, savePredictions = "final")
blue_nn_kfold <- train(blueWinsD ~ ., method = "nnet", trControl = trainParameters, data =
blue_nn_train)
blue_nn_kfold

#Predict with Train
predict_nnkfold_train <- table(predict(blue_nn_kfold, blue_nn_train),
blue_nn_train$blueWinsD)
confusionMatrix(predict_nnkfold_train)

#Predict with Test
predict_nnkfold_test <- table(predict(blue_nn_kfold, blue_nn_test), blue_nn_test$blueWinsD)
confusionMatrix(predict_nnkfold_test)
```