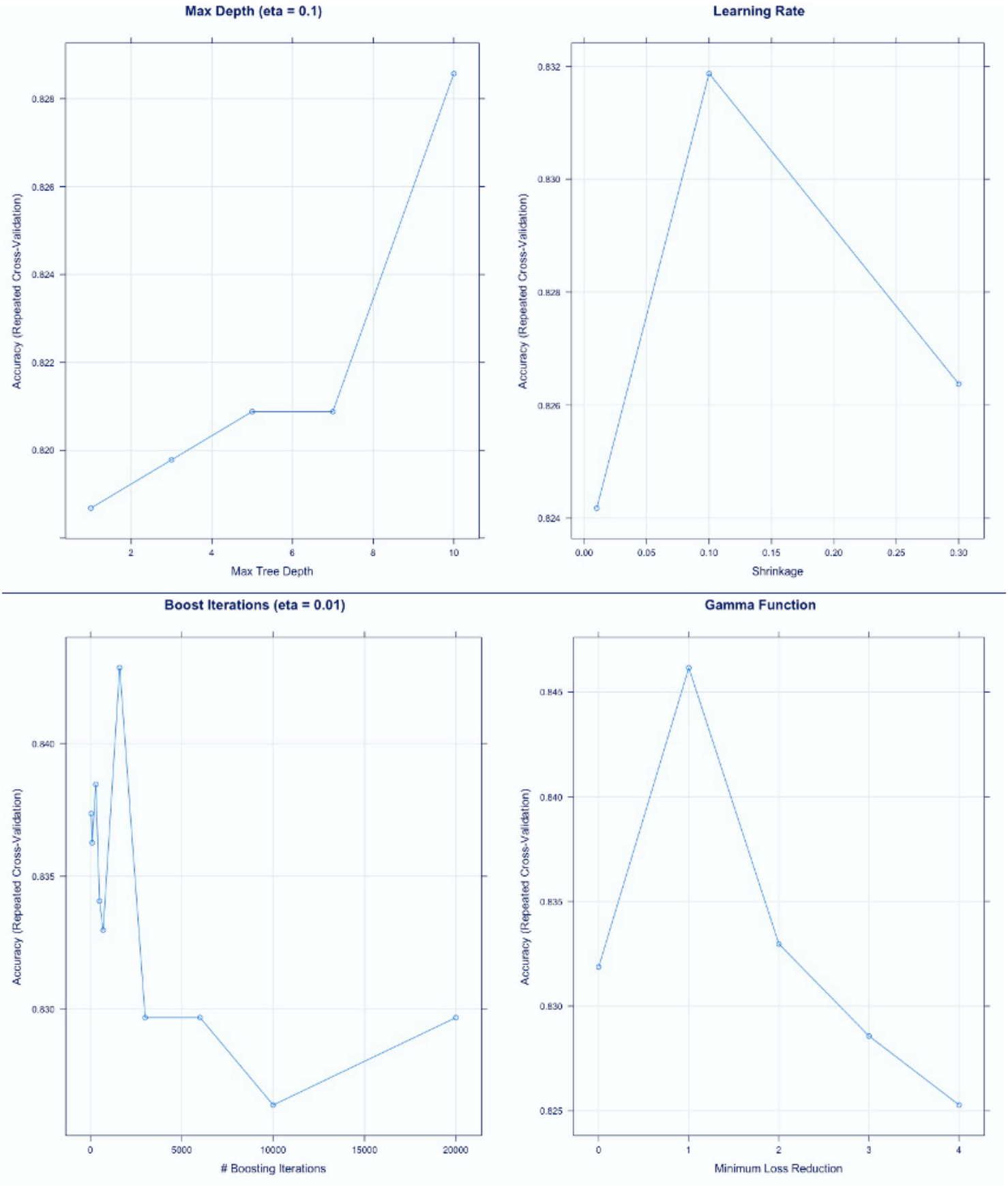
Appendix E109 Project

John Wu and Scotty Smith

5/4/2022

Appendix:

1. https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020 (https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020) (Kaggle dataset for Formula 1 and includes links to subsequent wikipedia dataset)
2. https://theoehrly.github.io/Fast-F1/ (https://theoehrly.github.io/Fast-F1/) (FastF1 python library)
3. https://en.wikipedia.org/wiki/Formula_One (https://en.wikipedia.org/wiki/Formula_One) (history of Formula 1)
4. https://fansided.com/2022/04/06/f1-driver-salaries-2022-formula-1-drivers-paid/ (https://fansided.com/2022/04/06/f1-driver-salaries-2022-formula-1-drivers-paid/) (Salary information)
5. https://github.com/jywu86/F1-Stats-Project (https://github.com/jywu86/F1-Stats-Project) (gitHub repo used for collaboration)
6. Boost Model Parameters


Max Depth (eta = 0.1)
Accuracy (Repeated Cross-Validation)
0.820
0.822
0.824
0.826
0.828
2
4
6
8
10
Max Tree Depth
Learning Rate
Accuracy (Repeated Cross-Validation)
0.824
0.826
0.828
0.830
0.832
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Shrinkage
Boost Iterations (eta = 0.01)
Accuracy (Repeated Cross-Validation)
0.830
0.835
0.840
0
5000
10000
15000
20000
Boosting Iterations
Gamma Function
Accuracy (Repeated Cross-Validation)
0.825
0.830
0.835
0.840
0.845
0
1
2
3
4
Minimum Loss Reduction

7.

```
# R Code Dump

############## CODE CLEANING ######################

library(dplyr)
library(tidyverse)

# Reading in data
results <- read.csv('Raw_Data/results.csv')
races <- read.csv('Old_Transformation/Relevant_Races.csv')
driver_id <- read.csv('Raw_Data/drivers.csv')
circuits <- read.csv('Old_Transformation/Relevant_Circuits_Final.csv')
qualify <- read.csv('Raw_Data/qualifying.csv')
team_id <- read.csv('Raw_Data/constructors.csv')
r_qualify <- read.csv('Old_Transformation/Relevant_Qualify_2.csv')
r_qualify <- select(r_qualify, -X.1, -X) # removing X.1 and X columns

# Removing columns from results that are not needed
results <- results[,c('resultId','raceId','driverId',
                      'constructorId','number','grid',
                      'position','positionText','positionOrder','time')]


# Merging data and combining columns
r_races <- races[,c('year','raceId','circuitId','Air.Temp','Track.Temp','Rainfall','W
ind.Speed')] # creating a reference dataframe
final_results <- merge(x=results, y=r_races, by='raceId',all.y=TRUE)

# Merging Final Results with Driver Name
#driver_id <- driver_id[,c('driverId','driverRef')]
#final_results <- merge(x =driver_id, y = final_results, by = 'driverId', all.y=TRUE)

#race_id <- r_races[,c('raceId','year')]

# Adding track_group (track speed) to results
circuits <- transform(circuits, track_group = ifelse(Avg_Speed_MPH<127.6, 'Low_Speed'
,ifelse(Avg_Speed_MPH<152.4, 'Med_Speed','High_Speed')))
speed_cols <- circuits[,c('circuitId','track_group')]
final_results <- merge(x=final_results, y=speed_cols, by='circuitId',all.x=TRUE)

r_qualify
r_qualify2 <- r_qualify[c("raceId","driverId","Fastest_Qual")]
r_qualify2

# Filtering out only relevant qualifying data
qualify_filter <- r_qualify %>% distinct(raceId)
qualify_filter
```

```
final_results
final_results <- merge(x=final_results, y = qualify_filter, by.x='raceId',all.y=TRUE)
final_results
# Removing results where people DQ'ed or Retired
final_results <- filter(final_results, positionText != 'R')
final_results2 <- merge(x=final_results,y=r_qualify2,by=c("raceId","driverId"))
final_results2
# Finding Top 10 drivers
driver_group <- final_results2 %>% group_by(driverId) %>% summarise(n=n()) %>%  arran
ge(desc(n))
driver_group <- merge(x=driver_group, y= driver_id, by='driverId',all.x=TRUE)
driver_group <- driver_group %>% arrange(desc(driver_group$n))

# Filtering by Top 10 Drivers (with most races) and merging with Driver Names
affected_drivers <- driver_group[1:10,c('driverId','driverRef')] # filtering top 10 d
rivers
final_results2 <- merge(x=final_results2, y=affected_drivers, by='driverId',all.y=TRU
E) # filtering out only results with top 8 drivers

final_results2
# Creating points system for positions 1-22 (Maybe not needed)
#points_model <- c(22,21,20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1)
#positionOrder <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22)
#position_df <- data.frame(positionOrder,points_model)
#final_results <- merge(x=final_results, y=position_df, by='positionOrder',all.x=TRUE
) # adding points to modeling data

# Creating team_rank by finding fastest average qualifying
team_rank <- r_qualify[,c('raceId','position_team','constructorId')]
team_rank <- team_rank %>% distinct(raceId,constructorId, .keep_all=TRUE)

race_rank <- team_rank %>%
  arrange(raceId, constructorId, position_team) %>%
  group_by(raceId) %>%
  mutate(team_rank = rank(position_team))

race_rank <- race_rank[,c('team_rank','raceId','constructorId')]

final_results2 <- merge(x=final_results2,y=race_rank, by=c('constructorId','raceId'),
all.x=TRUE)

final_results2 <- final_results2 %>%
  arrange(raceId, constructorId, team_rank) %>%
  group_by(raceId) %>%
  mutate(team_rank = rank(team_rank, ties.method='min'))

final_results2$position <- as.numeric(final_results2$position)
```

```
final_results2 <- final_results2 %>%
  arrange(raceId, driverId,position) %>%
  group_by(raceId) %>%
  mutate(final_position = rank(position, ties.method='min'))

# Adding team name
final_results2 <- merge(x=final_results2, y=team_id[,c('constructorId','constructorRe
f')], by='constructorId',all.x=TRUE)

# Adding track name
final_results2 <- merge(x=final_results2, y=circuits[,c('circuitId','circuitRef')], b
y='circuitId',all.x=TRUE)

# Converting Dry and Wet to bimodal
final_results2$Rainfall2[final_results2$Rainfall != 'Dry'] <- 'Wet'
final_results2$Rainfall2[final_results2$Rainfall == 'Dry'] <- 'Dry'

# Creating only modeling data
modeling_results <- subset(final_results2, select = -c(circuitId,constructorId,
                                                       driverId,number,
                                                       positionText,
                                                       positionOrder,time, Rainfall))


#write.csv(final_results2,'Modeling_Data_Full.csv', row.names=FALSE)
write.csv(modeling_results, 'Modeling_Only.csv', row.names=FALSE)

## Adding normalized qualifying

circuits <- read.csv("Raw_Data/circuits.csv")
ModelData <- read.csv("Modeling_Data_Full.csv")

circuits_year <- read.csv("Raw_Data/circuits_year.csv")
circuits_year <- circuits_year[,c('circuitId','Turns','Sharp.Turns','year','dist_s_tu
rns','dist.mi','dist_turns','turns_s_mile','turns_mile','Type')]
ModelData <- merge(ModelData,circuits_year,by=c('circuitId','year'))
ModelData
write.csv(ModelData,'Model_Data_w_Turns.csv',row.names=FALSE)

# creating normalized qualifying time

qualify <- read.csv('Raw_Data/qualifying2.csv')

average_qualify_time <- qualify %>% select(raceId,constructorId,Column2) %>% group_by
(raceId, constructorId) %>%
  summarise(qualify_time = mean(Column2))
```

```
qualify_time <- average_qualify_time %>% group_by(raceId) %>%
  mutate(qualifying_dif = qualify_time/min(qualify_time))

qualify_time <- subset(qualify_time, select = -c(qualify_time))

fulldata <- read.csv('Modeling_Data_Full.csv')

fulldata <- read.csv('Model_Data_w_Turns.csv')

fulldata <- subset(fulldata, select = -c(qualifying_dif))

fulldata2 <- merge(x=fulldata, y=qualify_time, by=c('raceId','constructorId'), all.x=
TRUE)

write.csv(fulldata2, 'Model_Data_w_Turns2.csv', row.names=FALSE)


########################## PURPOSE 1 Modeling ############################


library(randomForest)
library(caret)
library(lime)
library(tree)
library(pROC)
library(car)
library(MASS)
library(xgboost)
library(coefplot)

install.packages(randomForest)

# Importing Data and Prepping it for modeling
mydata_all <- read.csv('Model_Data_w_Turns.csv')

mydata_win <- subset(mydata_all, select= c(dist.mi,grid,position,Air.Temp,
                                           Track.Temp,Wind.Speed,driverRef,
                                           constructorRef,Rainfall2,
                                           qualifying_dif,team_rank,
                                           dist_turns,dist_s_turns,
                                           turns_mile,turns_s_mile,Turns,
                                           Sharp.Turns,Type))

mydata_top6 <- subset(mydata_all, select= c(dist.mi,grid,position,Air.Temp,
                                            Track.Temp,Wind.Speed,driverRef,
                                            constructorRef,Rainfall2,qualifying_dif,
```

```
                                            team_rank,dist_turns,dist_s_turns,
                                            turns_mile,turns_s_mile,
                                            Turns,Sharp.Turns,Type))

mydata_top3 <- subset(mydata_all, select= c(dist.mi,grid,position,Air.Temp,
                                            Track.Temp,Wind.Speed,driverRef,
                                            constructorRef,Rainfall2,qualifying_dif,
                                            team_rank,dist_turns,dist_s_turns,
                                            turns_mile,turns_s_mile,
                                            Turns,Sharp.Turns,Type))

mydata_top6$finish_tier[mydata_top6$position<=6] <-'Top6'
mydata_top6$finish_tier[(mydata_top6$position>6)] <- 'Back_Marker'

mydata_top3$finish_tier[mydata_top3$position<=3] <-'Podium'
mydata_top3$finish_tier[(mydata_top3$position>3)] <- 'Back_Marker'


mydata_top6 <- na.omit(mydata_top6)
mydata_top3 <- na.omit(mydata_top3)
mydata_win <- na.omit(mydata_win)


mydata_win$win[mydata_win$position ==1] <- 'Win'
mydata_win$win[mydata_win$position !=1] <- 'Lose'


############## Data Cleaning and dropping columns ################################

model_data_win <- subset(mydata_win, select = -c(turns_mile,turns_s_mile,Turns,
                                                 dist_turns,Sharp.Turns,position,
                                                 constructorRef,team_rank))
#str(model_data_win)
factor_cols_win <- c('Rainfall2','driverRef','win','Type')
model_data_win[,factor_cols_win] <- lapply(model_data_win[,factor_cols_win],factor)

model_data_top6 <- subset(mydata_top6, select = -c(turns_mile,turns_s_mile,Turns,
                                                   dist_turns,Sharp.Turns,
                                                   position,constructorRef,team_rank)
)
#str(model_data_top5)
factor_cols_top6 <- c('Rainfall2','driverRef','finish_tier','Type')
model_data_top6[,factor_cols_top6] <- lapply(model_data_top6[,factor_cols_top6],facto
r)

model_data_top3 <- subset(mydata_top3, select = -c(turns_mile,turns_s_mile,Turns,
                                                   dist_turns,Sharp.Turns,position,
```

```
                                                    constructorRef,team_rank))
#str(model_data_top3)
factor_cols_top3 <- c('Rainfall2','driverRef','finish_tier','Type')
model_data_top3[,factor_cols_top3] <- lapply(model_data_top3[,factor_cols_top3],facto
r)


#################### Win Lose Model ##############################################
# Training and Testing for Win Lose Model
set.seed(24)
ind <- sample(2, nrow(model_data_win), replace=T, prob=c(0.6,0.4))
train_win <- model_data_win[ind==1,]
test_win <- model_data_win[ind==2,]

############# logistic regression model for Win-Lose
log_win <- glm(win~., data=train_win, family='binomial')

# create prediction based on test
log_win_predict <- predict(log_win, test_win,type='response')

summary(log_win)

# ROC Curve for logistic model
rlog <- multiclass.roc(test_win$win,log_win_predict, percent=TRUE)
roc_log <- rlog[['rocs']]
r_log<-roc_log[[1]]
plot.roc(r_log,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'ROC Curve for Logistic (Win)')

log_win_predict_f<- as.factor(ifelse(log_win_predict_r>0.1,'Win','Lose'))
confusionMatrix(log_win_predict_f, test_win$win, positive='Win')

# creating pareto chart of total wins (for report)
win_data <- mydata_win %>%
  group_by(driverRef) %>%
  mutate(win_total = sum(win =='Win'))

driver_win <- unique(win_data[,c('driverRef','win_total')]) %>% arrange((win_total))
driver_win$driverRef <- gsub('max_verstappen','max',driver_win$driverRef)
driver_win$driverRef <- gsub('raikkonen','rai',driver_win$driverRef)
driver_win$reference <- 'no'
driver_win[8,'reference'] <- 'yes'
```

```
driver_win[3,'reference'] <- 'no'

driver_win <- arrange(driver_win,win_total)
driver_win$driverRef <- factor(driver_win$driverRef, levels=driver_win$driverRef)
win_pareto <-ggplot(driver_win, aes(x=driverRef,y=win_total, fill=reference)) +
  geom_col() +coord_flip() +
  theme(legend.position='none', plot.title = element_text(hjust = 0.5))+
  scale_fill_manual(values=c('grey69','tomato1')) +
  ggtitle('Wins by Driver') +
  xlab('Driver Name') +
  ylab('Total Wins (2018-2021)')

win_pareto

######### Forest Model for Win-Lose
cvcontrol <- trainControl(method ='repeatedcv',
                          number =5,
                          repeats=2,
                          allowParallel = TRUE)
forest_win <- train(win ~.,
                    data=train_win,
                    method='rf',
                    trControl = cvcontrol,
                    importance=TRUE,ntree=400)

# Create initial probability predictions based on test
forest_win_pred <- predict(forest_win, test_win, type='prob')
# ROC curve for random forest
rforest <- multiclass.roc(test_win$win,forest_win_pred$Win, percent=TRUE)
roc_forest <- rforest[['rocs']]
r_forest <-roc_forest[[1]]
plot.roc(r_forest,
         col = 'red',
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         main = 'ROC Curve for Forest')

coords(r_boost,"best", ret='threshold',transpose=FALSE)

# Prediction using ROC threshold
forest_win_factor <- as.factor(ifelse(forest_win_pred$Win>0.1,"Win","Lose"))

# Confusion Matrix
confusionMatrix(forest_win_factor, test_win$win, positive='Win')
```

```
plot(varImp(forest_win), main='Variable Importance for Race Win (Forest Model)')

######### Boost Model for Win-Lose
cvcontrol <- trainControl(method ='repeatedcv',
                          number =5,
                          repeats=2,
                          allowParallel = TRUE)
boost_win<- train(win ~.,
                  data = train_win,
                  method='xgbTree',
                  trControl = cvcontrol,
                  tuneGrid = expand.grid(nrounds = 50,
                                         max_depth = 3,
                                         eta = .1,
                                         gamma = 2,
                                         colsample_bytree =1,
                                         min_child_weight = 1,
                                         subsample =1 ))
# Prediction using test
boost_win_pred <- predict(boost_win, test_win, type='prob')
# ROC Curve
rboost <- multiclass.roc(test_win$win,boost_win_pred$Win, percent=TRUE)
roc_boost <- rboost[['rocs']]
r_boost <-roc_boost[[1]]
plot.roc(r_boost,
         print.auc = T,
         print.auc.cex=1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='Tomato',
         #show.thres = T,
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Boost (Win)')

boost_win_factor <- as.factor(ifelse(boost_win_pred$Win>0.05,"Win","Lose"))

confusionMatrix(boost_win_factor, test_win$win, positive='Win')

# Finding the best threshold
coords(r_boost,"best", ret='threshold',transpose=FALSE)
# coords(r_forest,"best", ret='threshold',transpose=FALSE)
# coords(r_log,"best", ret='threshold',transpose=FALSE)

plot(boostwin_depth, main = 'Max tree depth vs CV-Accuracy')

# plotting ROC curves for all
```

```
par(mfrow=c(1,3))
#par(mar= c(4,4,4,4)+.1)
plot.roc(r_log,
         print.auc = T,
         print.auc.cex=1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Logistic (Win)')
plot.roc(r_forest,
         print.auc = T,
         print.auc.cex=1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Forest (Win)')
plot.roc(r_boost,
         print.auc = T,
         print.auc.cex=1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='Tomato',
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Boost (Win)')

###################### Models used for Analysis in Win Lose
# StepAIC for logistic regression model
stepAIC(log_win)

## Logistic Reduced Model (Explanation in Report)
log_win_grid <- glm(win~grid+driverRef, data= train_win, family='binomial')
log_win_g_pred <- as.factor(ifelse(predict(log_win_grid, test_win,
                                           type='response')>0.1,'Win','Lose'))
confusionMatrix(log_win_g_pred, test_win$win, positive='Win')
saveRDS(log_win_grid,'log_win_model.rds')


summary(log_win_grid)
coefplot(log_win_grid, intercept=FALSE)

#################### PODIUM FINISH ANALYSIS ###################################
# Sampling and Training for Podium Models
```

```
str(model_data_top3)
set.seed(24)
ind <- sample(2, nrow(model_data_top3), replace=T, prob=c(0.6,0.4))
train_pod <- model_data_top3[ind==1,]
test_pod <- model_data_top3[ind==2,]

# Logistic Regression Podium
log_pod <- glm(finish_tier~., data=test_pod, family='binomial')

# Creating ROC Curve of Logistic
log_pod_predict <- predict(log_pod, test_pod,type='response')
rlog_pod <- multiclass.roc(test_pod$finish_tier,log_pod_predict, percent=TRUE)
roc_log_pod <- rlog_pod[['rocs']]
r_log_pod<-roc_log_pod[[1]]
plot.roc(r_log_pod,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'ROC Curve for Logistic (Podium)')

# Predicting with Threshold
log_pod_predict_f<- as.factor(ifelse(log_pod_predict>0.2,'Podium','Back_Marker'))
# Confusion Matrix
confusionMatrix(log_pod_predict_f, test_pod$finish_tier, positive='Podium')

# Forest Model Podium
cvcontrol <- trainControl(method ='repeatedcv',
                          number =5,
                          repeats=2,
                          allowParallel = TRUE)

f_pod <- train(finish_tier ~.,
               data=train_pod,
               method='rf',
               trControl = cvcontrol,
               importance=TRUE,ntree=400)

# ROC Curve
f_pod_pred <- predict(f_pod,test_pod, type='prob')
rf_pod <- multiclass.roc(test_pod$finish_tier,f_pod_pred$Podium, percent=TRUE)
roc_f_pod <- rf_pod[['rocs']]
r_f_pod <-roc_f_pod[[1]]
plot.roc(r_f_pod,
         print.auc = T,
         auc.polygon = T,
```

```
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         main = 'ROC Curve for Forest (Podium)')


# Forest Prediction with Threshold
f_pod_predict_f<- as.factor(ifelse(f_pod_pred$Podium>0.2,'Podium','Back_Marker'))
# Confusion Matrix
confusionMatrix(f_pod_predict_f, test_pod$finish_tier, positive='Podium')

plot(varImp(f_pod))
# Boost Model
cvcontrol <- trainControl(method ='repeatedcv',
                          number =5,
                          repeats=2,
                          allowParallel = TRUE)

b_pod <- train(finish_tier ~.,
               data = train_pod,
               method='xgbTree',
               trControl = cvcontrol,
               tuneGrid = expand.grid(nrounds = 100,
                                      max_depth =3,
                                      eta = 0.1,
                                      gamma = 0,
                                      colsample_bytree =1,
                                      min_child_weight = 1,
                                      subsample =1 ))

plot(b_pod, main='Different Gamma Function Podium Boost Model')

# ROC Curve
b_pod_pred <- predict(b_pod,test_pod, type='prob')
rb_pod <- multiclass.roc(test_pod$finish_tier,b_pod_pred$Podium, percent=TRUE)
roc_b_pod <- rb_pod[['rocs']]
r_b_pod <-roc_b_pod[[1]]
plot.roc(r_b_pod,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='Tomato',
         print.thres = T,
         main = 'ROC Curve for Boost (Podium)')

# Confusion Matrix
b_pod_predict_f<- as.factor(ifelse(b_pod_pred$Podium>0.1,'Podium','Back_Marker'))
```

```
confusionMatrix(b_pod_predict_f, test_pod$finish_tier, positive='Podium')


# creating pareto chart of podiums (for report)
pod_data <- mydata_top3 %>%
  group_by(driverRef) %>%
  mutate(pod_total = sum(finish_tier =='Podium'), race_total = n())

driver_pod <- unique(pod_data[,c('driverRef','pod_total','race_total')]) %>%
  arrange((pod_total))
driver_pod$driverRef <- gsub('max_verstappen','max',driver_win$driverRef)
driver_pod$driverRef <- gsub('raikkonen','rai',driver_win$driverRef)
driver_pod$highlight <- 'no'
driver_pod[8,'highlight'] <- 'yes'
driver_pod[3,'highlight'] <- 'no'
driver_pod[,'highlight'] <- 'no'

driver_pod$pod_percent <- (driver_pod$pod_total/driver_pod$race_total)*100


par(mfrow=c(1,1))
driver_pod <- arrange(driver_pod,pod_total)
driver_pod[8,'highlight'] <- 'yes'
driver_pod$driverRef <- factor(driver_pod$driverRef, levels=driver_pod$driverRef)
pod_pareto <-ggplot(driver_pod, aes(x=driverRef,y=pod_total, fill=highlight)) +
  geom_col() +coord_flip() +
  theme(legend.position='none', plot.title = element_text(hjust = 0.5))+
  scale_fill_manual(values=c('grey69','tomato1')) +
  ggtitle('Podiums by Driver') +
  xlab('Driver Name') +
  ylab('Total Podiums (2018-2021)')

pod_pareto


#### Models Used for Explanation

plot(varImp(f_pod))

f_pod_reduced <- train(finish_tier ~grid + qualifying_dif + driverRef +
                         Type +dist.mi,
                       data=train_pod,
                       method='rf',
                       trControl = cvcontrol,
                       importance=TRUE,ntree=400)
```

```
# ROC Curve and Confusion Matrix
f_pod_reduced_pred <- predict(f_pod_reduced,test_pod, type='prob')
rf_pod_reduced <- multiclass.roc(test_pod$finish_tier,f_pod_reduced_pred$Podium,
                                 percent=TRUE)
roc_f_pod_reduced <- rf_pod_reduced[['rocs']]
r_f_pod_reduced <-roc_f_pod_reduced[[1]]
plot.roc(r_f_pod_reduced,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         main = 'ROC Curve for Reduced Forest (Podium)')

f_pod_reduced_factor <- as.factor(ifelse(f_pod_reduced_pred$Podium>0.2,
                                         "Podium","Back_Marker"))

confusionMatrix(f_pod_reduced_factor, test_pod$finish_tier, positive='Podium')

imp_pod <- plot(varImp(f_pod_reduced), main='Variable Importance for Podium (Forest M
odel)')

log_pod_final <- glm(finish_tier ~ grid + driverRef + qualifying_dif, data=test_pod,
                     family='binomial')
log_pod_final_pred <- predict(log_pod_final, test_pod,type='response')
rlog_pod_final <- multiclass.roc(test_pod$finish_tier,log_pod_final_pred, percent=TRU
E)
roc_log_pod_final <- rlog_pod_final[['rocs']]
r_log_pod_final <-roc_log_pod_final[[1]]
plot.roc(r_log_pod_final,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'ROC Curve for Logistic-Reduced (Podium)')

install.packages("coefplot")
library(coefplot)
log_pod_s <- summary(log_pod_final)
log_pod_coeff <- log_pod_s$coefficients
log_pod_coeff <- data.frame(log_pod_coeff)
coefplot(log_pod_final,intercept=FALSE)
saveRDS(log_pod_final,'log_pod_model.rds')

# ROC Curves for Report
par(mfrow=(c(1,3)))
```

```
plot.roc(r_log_pod,
         print.auc = T,
         print.auc.cex = 1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Logistic (Podium)')
plot.roc(r_f_pod,
         print.auc = T,
         print.auc.cex = 1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         print.thres.cex = 1.05,
         main = 'ROC Curve for Forest (Podium)')
plot.roc(r_b_pod,
         print.auc = T,
         print.auc.cex =1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='Tomato',
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Boost (Podium)')

################## TOP TEAM (1-6 analysis) ##########################
str(model_data_top6)
set.seed(24)
ind <- sample(2, nrow(model_data_top6), replace=T, prob=c(0.6,0.4))
train_tm <- model_data_top6[ind==1,]
test_tm <- model_data_top6[ind==2,]

# Logistic Regression Podium
log_tm <- glm(finish_tier~., data=test_tm, family='binomial')

# Creating ROC Curve of Logistic
log_tm_predict <- predict(log_tm, test_tm,type='response')
rlog_tm <- multiclass.roc(test_tm$finish_tier,log_tm_predict, percent=TRUE)
roc_log_tm <- rlog_tm[['rocs']]
r_log_tm<-roc_log_tm[[1]]
plot.roc(r_log_tm,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
```

```
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'ROC Curve for Logistic (Top 6)')

# Confusion Matrix
log_tm_predict_f<- as.factor(ifelse(log_tm_predict>0.5,'Top6','Back_Marker'))
confusionMatrix(log_tm_predict_f, test_tm$finish_tier, positive='Top6')

# Forest Model Podium
cvcontrol <- trainControl(method ='repeatedcv',
                          number =5,
                          repeats=2,
                          allowParallel = TRUE)

f_tm <- train(finish_tier ~.,
              data=train_tm,
              method='rf',
              trControl = cvcontrol,
              importance=TRUE,ntree=400)

# ROC Curve
f_tm_pred <- predict(f_tm,test_tm, type='prob')
rf_tm <- multiclass.roc(test_tm$finish_tier,f_tm_pred$Top6, percent=TRUE)
roc_f_tm <- rf_tm[['rocs']]
r_f_tm <-roc_f_tm[[1]]
plot.roc(r_f_tm,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         main = 'ROC Curve for Forest (Top 6)')

plot(varImp(f_tm), main = 'Backmarker Model')

# Confusion Matrix
f_tm_predict_f<- as.factor(ifelse(f_tm_pred$Top6>0.6,'Top6','Back_Marker'))
confusionMatrix(f_tm_predict_f, test_tm$finish_tier, positive='Top6')

# Boost Model
b_tm <- train(finish_tier ~.,
              data = train_tm,
              method='xgbTree',
              trControl = cvcontrol,
              tuneGrid = expand.grid(nrounds = 100,
                                     max_depth =3,
                                     eta = 0.1,
```

```
                                      gamma = 0,
                                      colsample_bytree =1,
                                      min_child_weight = 1,
                                      subsample =1 ))

plot(b_pod, main='Different Gamma Function Podium Boost Model')

# ROC Curve
b_tm_pred <- predict(b_tm,test_tm, type='prob')
rb_tm <- multiclass.roc(test_tm$finish_tier,b_tm_pred$Top6, percent=TRUE)
roc_b_tm <- rb_tm[['rocs']]
r_b_tm <-roc_b_tm[[1]]
plot.roc(r_b_tm,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='Tomato',
         print.thres = T,
         main = 'ROC Curve for Boost (Top 6)')

# Confusion Matrix
b_tm_predict_f<- as.factor(ifelse(b_tm_pred$Top6>0.6,'Top6','Back_Marker'))
confusionMatrix(b_tm_predict_f, test_tm$finish_tier, positive='Top6')


# WINNING MODELS FOR Top6

# Boost Model Top6
plot(varImp(b_tm), main='Not Reduced Important Variables', ylab='Variables')

b_tm_rounds <- train(finish_tier ~ grid + qualifying_dif + Track.Temp + driverRef,
                     data = train_tm,
                     method='xgbTree',
                     trControl = cvcontrol,
                     tuneGrid = expand.grid(nrounds = c(50,100,300,500,700,1600,
                                                        3000,6000,10000,20000),
                                            max_depth =3,
                                            eta = 0.01,
                                            gamma = 0,
                                            colsample_bytree =1,
                                            min_child_weight = 1,
                                            subsample =1 ))

b_tm_eta <- train(finish_tier ~ grid + qualifying_dif + Track.Temp + driverRef,
                  data = train_tm,
                  method='xgbTree',
                  trControl = cvcontrol,
```

```
                  tuneGrid = expand.grid(nrounds = 1600,
                                         max_depth =5,
                                         eta = c(0.01,0.1,0.3),
                                         gamma = 0,
                                         colsample_bytree =1,
                                         min_child_weight = 1,
                                         subsample =1 ))

b_tm_depth <- train(finish_tier ~ grid + qualifying_dif + Track.Temp + driverRef,
                    data = train_tm,
                    method='xgbTree',
                    trControl = cvcontrol,
                    tuneGrid = expand.grid(nrounds = 1600,
                                           max_depth =c(1,3,5,7,10),
                                           eta = 0.1,
                                           gamma = 0,
                                           colsample_bytree =1,
                                           min_child_weight = 1,
                                           subsample =1 ))

b_tm_gamma <- train(finish_tier ~ grid + qualifying_dif + Track.Temp + driverRef,
                    data = train_tm,
                    method='xgbTree',
                    trControl = cvcontrol,
                    tuneGrid = expand.grid(nrounds = 1600,
                                           max_depth =5,
                                           eta = 0.1,
                                           gamma = c(0,1,2,3),
                                           colsample_bytree =1,
                                           min_child_weight = 1,
                                           subsample =1 ))

b_tm_reduced<- train(finish_tier ~ grid + qualifying_dif + Track.Temp + driverRef,
                     data = train_tm,
                     method='xgbTree',
                     trControl = cvcontrol,
                     tuneGrid = expand.grid(nrounds = 300,
                                            max_depth =5,
                                            eta = 0.1,
                                            gamma = 1,
                                            colsample_bytree =1,
                                            min_child_weight = 1,
                                            subsample =1 ))

par(mfrow=(c(2,2)))

# Creating XGBoost trees to analyze
```

```
library(DiagrammeR)
xgb.plot.tree(model=b_tm$finalModel, trees=50)
xgb.plot.tree(model=b_tm$finalModel, trees=1)

plot(varImp(b_tm), main='Variable Importance for Boost Model (Top 6)')


# ROC Curve and Confusion Matrix
b_tm_reduced_pred <- predict(b_tm_reduced,test_tm, type='prob')
rb_tm_reduced <- multiclass.roc(test_tm$finish_tier,b_tm_reduced_pred$Top6,
                                percent=TRUE)
rocb_tm_reduced <- rb_tm_reduced[['rocs']]
r_b_tm_reduced <-rocb_tm_reduced[[1]]
plot.roc(r_b_tm_reduced,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         main = 'ROC Curve for Boost (Top6)')

b_tm_reduced_factor <- as.factor(ifelse(b_tm_reduced_pred$Top6>0.7,"Top6",
                                        "Back_Marker"))

confusionMatrix(b_tm_reduced_factor, test_tm$finish_tier, positive='Top6')

log_pod_final_pred_f<- as.factor(ifelse(log_pod_final_pred>0.3,'Podium','Back_Marker'
))
confusionMatrix(log_pod_final_pred_f, test_pod$finish_tier, positive='Podium')

# creating pareto chart of podiums (for report)
pod_data <- mydata_top3 %>%
  group_by(driverRef) %>%
  mutate(pod_total = sum(finish_tier =='Podium'), race_total = n())

driver_pod <- unique(pod_data[,c('driverRef','pod_total','race_total')]) %>%
  arrange((pod_total))
driver_pod$driverRef <- gsub('max_verstappen','max',driver_win$driverRef)
driver_pod$driverRef <- gsub('raikkonen','rai',driver_win$driverRef)
driver_pod$highlight <- 'no'
driver_pod[6,'highlight'] <- 'yes'
driver_pod[3,'highlight'] <- 'yes'

driver_pod$pod_percent <- (driver_pod$pod_total/driver_pod$race_total)*100


par(mfrow=c(1,1))
```

```
driver_pod <- arrange(driver_pod,pod_total)
driver_pod$driverRef <- factor(driver_pod$driverRef, levels=driver_pod$driverRef)
pod_pareto <-ggplot(driver_pod, aes(x=driverRef,y=pod_total, fill=highlight)) +
  geom_col() +coord_flip() +
  theme(legend.position='none', plot.title = element_text(hjust = 0.5))+
  scale_fill_manual(values=c('grey69','tomato1')) +
  ggtitle('Podiums by Driver') +
  xlab('Driver Name') +
  ylab('Total Podiums (2018-2021)')

pod_pareto


# ROC Curve and Confusion Matrix
f_pod_reduced_pred <- predict(f_pod_reduced,test_pod, type='prob')
rf_pod_reduced <- multiclass.roc(test_pod$finish_tier,f_pod_reduced_pred$Podium,
                                 percent=TRUE)
roc_f_pod_reduced <- rf_pod_reduced[['rocs']]
r_f_pod_reduced <-roc_f_pod_reduced[[1]]
plot.roc(r_f_pod_reduced,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         main = 'ROC Curve for Reduced Forest (Podium)')

f_pod_reduced_factor <- as.factor(ifelse(f_pod_reduced_pred$Podium>0.4,"Podium",
                                         "Back_Marker"))

confusionMatrix(f_pod_reduced_factor, test_pod$finish_tier, positive='Podium')

plot(varImp(f_pod_reduced), main='Variable Importance for Podium (Forest Model)')

# ROC Curves for Report
par(mfrow=(c(1,3)))
plot.roc(r_log_tm,
         print.auc = T,
         print.auc.cex=1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Logistic (Top 6)')
plot.roc(r_f_tm,
         print.auc = T,
```

```
         print.auc.cex=1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightgoldenrod',
         print.thres = T,
         print.thres.cex =1.05,
         main = 'ROC Curve for Forest (Top 6)')
plot.roc(r_b_tm,
         print.auc = T,
         print.auc.cex =1.1,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='Tomato',
         print.thres = T,
         print.thres.cex=1.05,
         main = 'ROC Curve for Boost (Top 6)')



str(model_data_top6)
set.seed(24)
ind <- sample(2, nrow(model_data_top6), replace=T, prob=c(0.6,0.4))
train <- model_data_top3[ind==1,]
test <- model_data_top3[ind==2,]

cvcontrol <- trainControl(method ='repeatedcv',
                          number =5,
                          repeats=2,
                          allowParallel = TRUE)

forest3 <- train(finish_tier ~.,
                 data=train,
                 method='rf',
                 trControl = cvcontrol,
                 importance=TRUE,ntree=400)

# ROC Curve
f3_predict <- predict(forest3,test, type='prob')
r <- multiclass.roc(test$finish_tier,f3_predict$Podium, percent=TRUE)
roc <- r[['rocs']]
r1 <-roc[[1]]
#plot.roc(r1, col='red',lwd=3, main= 'ROC Curve for Forest 3 (Podium)')
plot.roc(r1,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
```

```
         print.thres = T,
         main = 'ROC Curve for Forest 3 (Top 5 Finish)')



boost3 <- train(finish_tier ~.,
                data = train,
                method='xgbTree',
                trControl = cvcontrol,
                tuneGrid = expand.grid(nrounds = 1000,
                                       max_depth =5,
                                       eta = 0.3,
                                       gamma = 2,
                                       colsample_bytree =1,
                                       min_child_weight = 1,
                                       subsample =1 ))


######################### PURPOSE 2 MODELING ############################
library(randomForest)
library(caret)
library(lime)
library(tree)
library(pROC)
library(ROCR)
library(car)
library(MASS)
library(dplyr)
library(sjPlot)
library(sjlabelled)
library(sjmisc)
library(ggplot2)
library(tidyverse)
library(traineR)
library(rstanarm)
library(cowplot)
theme_set(theme_sjplot())


# Importing Data and Prepping it for modeling
mydata_all <- read.csv('Model_Data_w_Turns.csv')
str(mydata)
mydata_win <- subset(mydata_all, select= c(dist.mi,grid,position,Air.Temp,
                                           Track.Temp,Wind.Speed,driverRef,
                                           constructorRef,Rainfall2,
                                           qualifying_dif,team_rank,dist_turns,
                                           dist_s_turns,turns_mile,turns_s_mile,
```

```
                                          Turns,Sharp.Turns,Type,year))
mydata_win <- na.omit(mydata_win)
mydata_win$finish_tier[mydata_win$position ==1] <- 'Win'
mydata_win$finish_tier[mydata_win$position !=1] <- 'Lose'
mydata_team_win <- subset(mydata_win, select= c(dist.mi,grid,Air.Temp,
                                                Track.Temp,Wind.Speed,driverRef,
                                                constructorRef,Rainfall2,
                                                qualifying_dif,team_rank,
                                                turns_s_mile,Type,finish_tier,
                                                year))
mydata_ind_w <- subset(mydata_win, select= c(dist.mi,grid,Air.Temp,Track.Temp,
                                             Wind.Speed,driverRef,constructorRef
                                             ,Rainfall2,qualifying_dif,
                                             team_rank,
                                             turns_s_mile,Type,finish_tier,
                                             year))

mydata_3 <- subset(mydata_all, select= c(dist.mi,grid,position,Air.Temp,
                                         Track.Temp,Wind.Speed,driverRef,
                                         constructorRef,Rainfall2,
                                         qualifying_dif,team_rank,dist_turns,
                                         dist_s_turns,turns_mile,turns_s_mile,
                                         Turns,Sharp.Turns,Type,year))
mydata_3 <- na.omit(mydata_3)
mydata_3$finish_tier[mydata_3$position <=3] <- 'Top3'
mydata_3$finish_tier[mydata_3$position >3] <- 'Not'
mydata_ind_3 <- subset(mydata_3, select= c(dist.mi,grid,Air.Temp,Track.Temp,
                                           Wind.Speed,driverRef,constructorRef,
                                           Rainfall2,qualifying_dif,team_rank,
                                           turns_s_mile,Type,finish_tier,year))


###Creating two different factor functions. One for individuals. One for Teams.
factor2 <- function(team) {
  team$Rainfall2 <- as.factor(team$Rainfall2)
  team$finish_tier <- as.factor(team$finish_tier)
  team$driverRef <- as.factor(team$driverRef)
  team
}
factor <- function(ind) {
  ind$Rainfall2 <- as.factor(ind$Rainfall2)
  ind$finish_tier <- as.factor(ind$finish_tier)
  ind
}

### Setting up filters for data models
```

```
ferrari_team_win <- mydata_team_win %>%
  filter(constructorRef == 'ferrari'& (driverRef == "vettel" |
                                         driverRef == "leclerc"))
ferrari_team_win <- subset(ferrari_team_win,select = -c(constructorRef,
                           team_rank,year,dist.mi,Air.Temp,qualifying_dif,Type))
mercedes_team_win <- mydata_team_win %>% filter(constructorRef == 'mercedes')
mercedes_team_win <- subset(mercedes_team_win,select = -c(constructorRef,
                         team_rank,year,dist.mi,Air.Temp,qualifying_dif,Type))
botta_3 <-mydata_ind_3 %>% filter(driverRef == 'bottas')
botta_3 <- subset(botta_3,select= -c(constructorRef,driverRef,year,dist.mi,
                                     Air.Temp,Type,qualifying_dif))
max_w <-mydata_ind_w %>% filter(driverRef == 'max_verstappen')
max_w <- subset(max_w,select= -c(constructorRef,driverRef,year,dist.mi,
                                   Air.Temp,Type,qualifying_dif))
hamilton_w <-mydata_ind_w %>% filter(driverRef == 'hamilton')
hamilton_w <- subset(hamilton_w,select= -c(constructorRef,driverRef,year,
                                     dist.mi,Air.Temp,Type,qualifying_dif))

str(botta_w)

max_w <- factor(max_w)
hamilton_w <- factor(hamilton_w)
botta_3 <- factor(botta_3)
mercedes_team_win <- factor(mercedes_team_win)
ferrari_team_win <- factor(ferrari_team_win)


### Setting up Ferrari Models W/L Team
set.seed(24)
ind_f_t_w <- sample(2, nrow(ferrari_team_win), replace=T, prob=c(0.80,0.20))
train_pod_f_t_w <- ferrari_team_win[ind_f_t_w==1,]
test_pod_f_t_w <- ferrari_team_win[ind_f_t_w==2,]


cvcontrol <- trainControl(method ='repeatedcv',
                          number =5,
                          repeats=2,
                          allowParallel = TRUE)

f_m_w_rf <- train(finish_tier ~grid+Wind.Speed+Track.Temp+Rainfall2+driverRef+
                    turns_s_mile,data=train_pod_f_t_w,method='rf',
                  trControl = cvcontrol,importance=TRUE)
f_m_w_log <- train.glm(finish_tier~ grid+driverRef+Wind.Speed+Track.Temp+
                         Rainfall2+turns_s_mile,
                       data=train_pod_f_t_w,method='glm.fit')
```

```
f_m_w_rf$coefnames

f_m_w_rf_pred <- predict(f_m_w_rf,test_pod_f_t_w,type='prob')
f_m_w_log_pred <- predict(f_m_w_log,test_pod_f_t_w,type='prob')

summary(f_m_w_log)



plot(varImp(f_m_w_rf))
f_m_w <- multiclass.roc(test_pod_f_t_w$finish_tier,
                        f_m_w_rf_pred$Win, percent=TRUE)
f_m_w  <- f_m_w[['rocs']]
f_m_w  <-f_m_w[[1]]

f_m_w

stepAIC(f_m_w_log)
f_m_w_log2 <- train.glm(finish_tier~ grid+driverRef,
                        data=train_pod_f_t_w,method='glm.fit')
summary(f_m_w_log2)

dev.off()
f_m_w2 <- multiclass.roc(test_pod_f_t_w$finish_tier,
                         f_m_w_log_pred$prediction[,'Win'], percent=TRUE)
f_m_w2  <- f_m_w2[['rocs']]
f_m_w2  <-f_m_w2[[1]]
par(mfrow=c(2,2))
roc1 <- plot.roc(f_m_w,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Ferrari Win/Loss RF',  print.auc.cex = 1.5,
         print.thres.cex = 1.5,
         cex.main=1.5,
         cex.lab=1.3)
labels <- c('Grid','Wind Speed','Track Temp','Rained',
            'Driver Vettel','Sharp Turns/Mi')
values <- as.numeric(varImp(f_m_w_rf)[1]$importance[,'Win'])
data <- data.frame(labels,values)
?ggplot
dev.new(width=10,height=5,unit='in')
gg1 <- ggplot(data,aes(values,reorder(labels,+values)),width=100)+
  geom_bar(stat = "identity",width=0.2,fill='skyblue') +
  xlab('Importance') +
```

```
  ylab(NULL) +
  ggtitle('Variable Importance Ferrari W/L RF') +
  theme(text=element_text(size=10,color='grey3'),
        axis.text.y=element_text(colour='black'))
plot(varImp(f_m_w_rf),main='Var Imp Ferrari')
roc2 <- plot.roc(f_m_w2,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Ferrari Win/Loss Logit',
         print.auc.cex = 1.5,
         print.thres.cex = 1.5,
         cex.main=1.5,
         cex.lab=1.3)
set_theme(axis.textsize.y = 1.3,axis.textcolor.y = "Black",title.size = 2)
pm <- plot_model(f_m_w_log,show.values=TRUE,transform = NULL,ci.lvl=NA,
                 title='Ferrari Logit Coefficients',dot.size=1,value.size=5)
levels(pm$data$term)<- c('Sharp Turns/Mi','Rained','Track Temp',
                         'Wind Speed','Driver Vettel','Grid')
pm

### Setting up Mercedes Models W/L Team
set.seed(24)
ind_m_t_w <- sample(2, nrow(mercedes_team_win), replace=T, prob=c(0.7,0.3))
train_pod_m_t_w <- mercedes_team_win[ind_m_t_w==1,]
test_pod_m_t_w <- mercedes_team_win[ind_m_t_w==2,]



m_m_w <- train(finish_tier ~.,data=train_pod_m_t_w,method='rf',trControl =
                 cvcontrol,importance=TRUE)
m_m_w_log.glm <- train.glm(finish_tier~ .,
                           data=train_pod_m_t_w,method='glm.fit')


summary(m_m_w_log.glm)

m_m_w_pred <- predict(m_m_w,test_pod_m_t_w,type='prob')
m_m_w_lpred.glm <- predict(m_m_w_log.glm,test_pod_m_t_w,type="prob")
par(mfrow=c(2,2))
plot(roc1)
plot(roc2)
roc3
roc4
```

```
m_m_wroc <- multiclass.roc(test_pod_m_t_w$finish_tier,
                           m_m_w_lpred.glm$prediction[,'Win'], percent=TRUE)
m_m_wroc  <- m_m_wroc[['rocs']]
m_m_wroc  <-m_m_wroc[[1]]
m_m_rocw <- multiclass.roc(test_pod_m_t_w$finish_tier,
                           m_m_w_pred$Win, percent=TRUE)
m_m_rocw  <- m_m_rocw[['rocs']]
m_m_rocw  <-m_m_rocw[[1]]


roc3 <- plot.roc(m_m_wroc,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Mercedes W/L Logit',  print.auc.cex = 1.5,
         print.thres.cex = 1.5,
         cex.main=1.5,
         cex.lab=1.3)
roc4 <- plot.roc(m_m_rocw,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Mercedes W/L Random Forest',  print.auc.cex = 1.5,
         print.thres.cex = 1.5,
         cex.main=1.5,
         cex.lab=1.3)
pm2 <- plot_model(m_m_w_log,show.values=TRUE,transform = NULL,
                  ci.lvl=NA,title='Mercedes Logit Coefficients',
                  dot.size=1,value.size=5)
levels(pm2$data$term)<- c('Sharp Turns/Mi','Rained','Driver Hamilton',
                          'Wind Speed','Track Temp','Grid')
pm2
labels2 <- c('Grid','Track Temp','Wind Speed','Driver Hamilton',
             'Rained','Sharp Turns/Mi')
values2 <- as.numeric(varImp(m_m_w)[1]$importance[,'Win'])
data2 <- data.frame(labels2,values2)
gg2<- ggplot(data2,aes(values2,reorder(labels2,+values2)),width=100)+
  geom_bar(stat = "identity",width=0.2,fill='skyblue') +
  xlab('Importance') +
  ylab(NULL) +
  ggtitle('Variable Importance Mercedes W/L RF') +
  theme(text=element_text(size=10,color='grey3'),axis.text.y=element_text
        (colour='black'))
```

```
plot(varImp(f_m_w_rf),main='Var Imp Ferrari')

plot_grid(pm,pm2)

plot_grid(gg1,gg2)
plot_grid(gg3,gg4,gg5,nrow=1)

plot_grid(pm3,pm4,pm5,nrow=1)

ggroc(c(roc1,roc2,roc3,roc4),aes="group")
### Setting up Max Individual Models W/L Team
par(mfrow=c(2,3))
set.seed(24)
ind_max_w <- sample(2, nrow(max_w), replace=T, prob=c(.7,.30))
train_max_w<- max_w[ind_max_w==1,]
test_max_w<- max_w[ind_max_w==2,]

max_rf_w <- train(finish_tier ~.,data=train_max_w,method='rf',trControl
                  = cvcontrol,importance=TRUE)
max_log_w <- train.glm(finish_tier~ .,data=train_max_w,method='glm.fit')

max_rf_wpred <- predict(max_rf_w,test_max_w,type='prob')
max_log_wpred <- predict(max_log_w,test_max_w,type="prob")
max_rf_wpred

summary(max_log_w)

summary(max_log_w)

max_w_roc <- multiclass.roc(test_max_w$finish_tier,max_rf_wpred$Win,
                            percent=TRUE)
max_w_roc <- max_w_roc[['rocs']]
max_w_roc <-max_w_roc[[1]]
max_w_rocl <- multiclass.roc(test_max_w$finish_tier,
                             max_log_wpred$prediction[,'Win'], percent=TRUE)
max_w_rocl <- max_w_rocl[['rocs']]
max_w_rocl <-max_w_rocl[[1]]
plot.roc(max_w_roc,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Max Verstappen W/L RF',print.auc.cex = 1.2,
         print.thres.cex = 1.2,
         cex.main=1.2,
         cex.lab=1.2)
```

```
plot.roc(max_w_rocl,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Max Verstappen W/L Logit',print.auc.cex = 1.2,
         print.thres.cex = 1.2,
         cex.main=1.2,
         cex.lab=1.2)

labels3 <- c('Grid','Track Temp','Wind Speed',
             'Rained','Team Rank','Sharp Turns/Mi')
values3 <- as.numeric(varImp(max_rf_w)[1]$importance[,'Win'])
data3 <- data.frame(labels3,values3)
gg3<- ggplot(data3,aes(values3,reorder(labels3,+values3)),width=100)+
  geom_bar(stat = "identity",width=0.2,fill='skyblue') +
  xlab('Importance') +
  ylab(NULL) +
  ggtitle('VI Max W/L RF') +
  theme(text=element_text(size=10,color='grey3'),
        axis.text.y=element_text(colour='black'))
gg3

pm3 <- plot_model(max_log_w,show.values=TRUE,transform = NULL,ci.lvl=NA,
                  title='Max W/L',dot.size=1,value.size=5)
levels(pm3$data$term)<- c('Sharp Turns/Mi','Team Rank','Rained',
                          'Wind Speed','Track Temp','Grid')
pm3
varImp(max_rf_w)[1]$importance
levels(pm$data$term)<- c('Sharp Turns/Mi','Rained','Track Temp',
                         'Wind Speed','Driver Vettel','Grid')
plot(varImp(max_rf_w))
### Setting up Lewis Individual Models W/L Team
set.seed(24)
ind_hamilton_w <- sample(2, nrow(hamilton_w), replace=T, prob=c(0.75,0.25))
train_hamilton_w<- hamilton_w[ind_hamilton_w==1,]
test_hamilton_w<- hamilton_w[ind_hamilton_w==2,]
str(train_hamilton_w)
hamilton_rf_w <- train(finish_tier ~.,data=train_hamilton_w,
                       method='rf',trControl = cvcontrol,importance=TRUE)
hamilton_log_w <- train.glm(finish_tier~ .,
                            data=train_hamilton_w,method='glm.fit')
plot_model(hamilton_log_w,show.values=TRUE,transform = NULL,ci.lvl=NA)
stepAIC(hamilton_log_w)
```

```
hamilton_rf_wpred <- predict(hamilton_rf_w,test_hamilton_w,type='prob')
hamilton_log_wpred <- predict(hamilton_log_w,test_hamilton_w,type="prob")

hamilton_w_roc <- multiclass.roc(test_hamilton_w$finish_tier,
                                 hamilton_rf_wpred$Win, percent=TRUE)
hamilton_w_roc <- hamilton_w_roc[['rocs']]
hamilton_w_roc <-hamilton_w_roc[[1]]
plot.roc(hamilton_w_roc,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Lewis Hamilton W/L RF',print.auc.cex = 1.2,
         print.thres.cex = 1.2,
         cex.main=1.2,
         cex.lab=1.2)
hamilton_w_rocl <- multiclass.roc(test_hamilton_w$finish_tier,
                                  hamilton_log_wpred$prediction[,'Win'],
                                  percent=TRUE)
hamilton_w_rocl <- hamilton_w_rocl[['rocs']]
hamilton_w_rocl <-hamilton_w_rocl[[1]]
plot.roc(hamilton_w_rocl,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Lewis Hamilton W/L Logit',print.auc.cex = 1.2,
         print.thres.cex = 1.2,
         cex.main=1.2,
         cex.lab=1.2)
plot_model(hamilton_log_w,show.values=TRUE,transform = NULL,ci.lvl=NA)
pm4 <- plot_model(hamilton_log_w,show.values=TRUE,transform = NULL,ci.lvl=NA,
                  title='Lewis W/L',dot.size=1,value.size=5)
levels(pm4$data$term)<- c('Sharp Turns/Mi','Team Rank','Rained','Wind Speed'
                          ,'Track Temp','Grid')
pm4
plot(varImp(hamilton_rf_w))


labels4 <- c('Grid','Track Temp','Wind Speed','Rained','Team Rank',
             'Sharp Turns/Mi')
values4 <- as.numeric(varImp(hamilton_rf_w)[1]$importance[,'Win'])
data4 <- data.frame(labels4,values4)
gg4<- ggplot(data4,aes(values4,reorder(labels4,+values4)),width=100)+
  geom_bar(stat = "identity",width=0.2,fill='skyblue') +
```

```
  xlab('Importance') +
  ylab(NULL) +
  ggtitle('VI Hamilton W/L RF') +
  theme(text=element_text(size=10,color='grey3')
        ,axis.text.y=element_text(colour='black'))
gg4
varImp(hamilton_rf_w)[1]$importance
### Setting up Bottas Individual Models Top3 Team

set.seed(24)
ind_botta_3 <- sample(2, nrow(botta_3), replace=T, prob=c(0.83,0.17))
train_botta_3<- botta_3[ind_botta_3==1,]
test_botta_3<- botta_3[ind_botta_3==2,]


botta_rf_3 <- train(finish_tier ~.,data=train_botta_3,method='rf',
                    trControl = cvcontrol,importance=TRUE)
botta_log_3 <- train.glm(finish_tier~ .,data=train_botta_3,method='glm.fit')
plot_model(botta_log_3,show.values=TRUE,transform = NULL,ci.lvl=NA)
botta_rf_3
summary(botta_log_3)

botta_rf_3pred <- predict(botta_rf_3,test_botta_3,type='prob')
botta_log_3pred <- predict(botta_log_3,test_botta_3,type="prob")


botta_3_roc <- multiclass.roc(test_botta_3$finish_tier,botta_rf_3pred$Top3,
                              percent=TRUE)
botta_3_roc <- botta_3_roc[['rocs']]
botta_3_roc <-botta_3_roc[[1]]
plot.roc(botta_3_roc,
         print.auc = T,
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Valtteri Bottas Top3 RF',print.auc.cex = 1.2,
         print.thres.cex = 1.2,
         cex.main=1.2,
         cex.lab=1.2)
botta_3_rocl <- multiclass.roc(test_botta_3$finish_tier,
                               botta_log_3pred$prediction[,'Top3'],
                               percent=TRUE)
botta_3_rocl <- botta_3_rocl[['rocs']]
botta_3_rocl <-botta_3_rocl[[1]]
plot.roc(botta_3_rocl,
         print.auc = T,
```

```
         auc.polygon = T,
         max.auc.polygon = T,
         auc.polygon.col ='lightblue',
         print.thres = T,
         main = 'Valtteri Bottas Top3 Logit',print.auc.cex = 1.2,
         print.thres.cex = 1.2,
         cex.main=1.2,
         cex.lab=1.2)

plot_model(botta_log_3,show.values=TRUE,transform = NULL,ci.lvl=NA)
pm5 <- plot_model(botta_log_3,show.values=TRUE,transform = NULL,ci.lvl=NA,
                  title='Bottas Top3',dot.size=1,value.size=5)
levels(pm5$data$term)<- c('Sharp Turns/Mi','Team Rank','Rained',
                          'Wind Speed','Track Temp','Grid')
pm5
plot(varImp(botta_rf_3))

labels5 <- c('Grid','Track Temp','Wind Speed','Rained','Team Rank',
             'Sharp Turns/Mi')
values5 <- as.numeric(varImp(botta_rf_3)[1]$importance[,'Top3'])
data5 <- data.frame(labels5,values5)
gg5<- ggplot(data5,aes(values5,reorder(labels5,+values5)),width=100)+
  geom_bar(stat = "identity",width=0.2,fill='skyblue') +
  xlab('Importance') +
  ylab(NULL) +
  ggtitle('VI Bottas Top3 RF') +
  theme(text=element_text(size=10,color='grey3'),axis.text.y=element_text
        (colour='black'))
gg5
varImp(botta_rf_3)[1]$importance
```