# installing the packages

installNewPackage <- function(packageName) {

if(packageName %in% rownames(installed.packages()) == FALSE)

{

install.packages(packageName, repos = "http://cran.us.r-project.org", dependencies=TRUE)

}

}

installNewPackage("Amelia")

installNewPackage("wordcloud")

installNewPackage("caret")

installNewPackage("corrplot")

installNewPackage("moments")

installNewPackage("e1071")

installNewPackage("elasticnet")

installNewPackage("earth")

installNewPackage("doParallel")

installNewPackage("doMC")

library(Amelia)

library(wordcloud)

library(caret)

library(corrplot)

library(moments)

library(e1071)

library(elasticnet)

library(earth)

library(doMC)

registerDoMC(4)

## Load the data

fifa\_2017 <- read.csv(file = "data/FullData.csv", na.strings=c("","NA"))

head(fifa\_2017)

fifa\_predictors <- fifa\_2017[,-which(names(fifa\_2017) %in% c("Rating", "National\_Position"))]

fifa\_targets <- fifa\_2017[,which(names(fifa\_2017) %in% c("Rating", "National\_Position"))]

## Dimensions of the Data

dim(fifa\_2017)

## Variables Word cloud

# Setting the seed

set.seed(1)

# Getting the column names

col\_name <- colnames(fifa\_2017)

# Generate random numbers for frequency for the word cloud

col\_name\_values <- runif(length(col\_name), min = 0, max = 1)

# Create the Word cloud for all variables

word\_cloud <- data.frame(col\_name, col\_name\_values)

wordcloud(word\_cloud$col\_name, freq = word\_cloud$col\_name\_values, random.order = TRUE, random.color = FALSE, scale = c(2, 0.05), col = brewer.pal(9, "Oranges"))

# Setting the seed

set.seed(1)

# Filtering the dataset

subset\_colclasses <- function(DF, colclasses="numeric") {

DF[,sapply(DF, function(vec, test) class(vec) %in% test, test=colclasses)]

}

fifa\_2017\_factor <- subset\_colclasses(fifa\_2017, c("factor"))

fifa\_2017\_integer <- subset\_colclasses(fifa\_2017, c("integer", "num"))

# Getting the column names

col\_name\_factor <- colnames(fifa\_2017\_factor)

col\_name\_integer <- colnames(fifa\_2017\_integer)

# Generate random numbers for frequency for the word cloud

col\_name\_factor\_values <- runif(length(col\_name\_factor), min = 0, max = 1)

col\_name\_integer\_values <- runif(length(col\_name\_integer), min = 0, max = 1)

# Create the Word cloud for all variables

word\_cloud\_factor <- data.frame(col\_name\_factor, col\_name\_factor\_values)

wordcloud(word\_cloud\_factor$col\_name\_factor, freq = word\_cloud\_factor$col\_name\_factor\_values, random.order = TRUE, random.color = FALSE,

scale = c(3.4, 0.5), col = brewer.pal(9, "Oranges"))

word\_cloud\_integer <- data.frame(col\_name\_integer, col\_name\_integer\_values)

wordcloud(word\_cloud\_integer$col\_name\_integer, freq = word\_cloud\_integer$col\_name\_integer\_values, random.order = TRUE, random.color = FALSE,

scale = c(2, 0.5), col = brewer.pal(9, "Oranges"))

## Missing values in the data

# Missmap

missmap(fifa\_2017, col = c("white", "brown3"), main = "Missing Map | Before Processing")

missing\_count <- sapply(fifa\_2017, function(x) sum(is.na(x)))

print(missing\_count)

# Get the missing variables alone

missing\_count[missing\_count > 0]

## Imputation of missing values

# Remove unwanted predictors

fifa\_2017$National\_Position <- NULL

fifa\_2017$National\_Kit <- NULL

# Impute values by median

fifa\_2017$Club\_Kit = ifelse(is.na(fifa\_2017$Club\_Kit),ave(fifa\_2017$Club\_Kit, FUN = function(x) median(x, na.rm = TRUE)),fifa\_2017$Club\_Kit)

fifa\_2017$Contract\_Expiry = ifelse(is.na(fifa\_2017$Contract\_Expiry),ave(fifa\_2017$Contract\_Expiry, FUN = function(x) median(x, na.rm = TRUE)),fifa\_2017$Contract\_Expiry)

# Impute categorical values by mode

Mode <- function(x) {

ux <- unique(x)

ux[which.max(table(match(x, ux)))]

}

fifa\_2017$Club\_Position = ifelse(is.na(fifa\_2017$Club\_Position), ave(fifa\_2017$Club\_Position, FUN = function(x) Mode(x)), fifa\_2017$Club\_Position)

fifa\_2017$Club\_Joining = ifelse(is.na(fifa\_2017$Club\_Joining), ave(fifa\_2017$Club\_Joining, FUN = function(x) Mode(x)),fifa\_2017$Club\_Joining)

fifa\_2017$Height <- as.numeric(gsub(fifa\_2017$Height,pattern = " cm",replacement = ""))

fifa\_2017$Weight <- as.numeric(gsub(fifa\_2017$Weight,pattern = " kg",replacement = ""))

## Missing values verification of data

missmap(fifa\_2017, col = c("white", "darkolivegreen3"),main = "Missing Map | After Processing")

## Skewness

# Intialize variable

skew = list()

# Selecting only numerical values

fifa\_num = sapply(X = fifa\_2017,is.numeric)

# Calculating Skewness for the variables

skew = round(abs(apply(fifa\_2017[,fifa\_num], 2, skewness)), 3)

high\_skew <- list()

high\_skew\_value <- list()

par(mfrow=c(3,3))

# Select skewness values greater than 1 and plot histogram

for(i in 1:length(skew)){

if(skew[i] > 1) {

high\_skew <- c(high\_skew, names(skew[i]))

high\_skew\_value <- c(high\_skew\_value, as.numeric(skew[i]))

hist(fifa\_2017[,names(skew[i])],main = paste(names(skew[i]), "|", skew[i]), cex.main = 3, xlab = names(skew[i]))

}

}

## Box Cox Transformation

# Preprocessing the data and applying Box Cox transformation for highly skewed variables

fifa\_trans <- preProcess(x = fifa\_2017[unlist(high\_skew)],method = "BoxCox")

fifa\_data <- predict(fifa\_trans,newdata = fifa\_2017[unlist(high\_skew)])

# Calculating Skewness after Box Cox transformation and Plotting histogram for the transformed variables

skew\_measures <- matrix()

par(mfrow = c(3,3))

for (i in 1:ncol(fifa\_data)){

skew\_measures[i] <- round(skewness(fifa\_data[,i]), 3)

hist(fifa\_data[,i], main = paste(colnames(fifa\_data)[i], "|", skew\_measures[i]), cex.main = 3, xlab = colnames(fifa\_data)[i] )

}

data.frame(cbind(column = colnames(fifa\_data), skew\_measures))

## Correlation between Continuous predictors

par(xpd = TRUE)

# Plot the correlation plot

corrplot(cor(fifa\_2017\_integer), order="hclust", tl.cex = 1)

# Finding out which predictors to elliminate since they have too large correlations

highCorr = findCorrelation(cor(fifa\_2017\_integer), cutoff = 0.90)

fifa\_2017\_integer\_rem <- fifa\_2017\_integer[,-highCorr]

removed\_col <- names(fifa\_2017\_integer)[highCorr]

# Matrix has no values > cutoff=0.9

corrplot(cor(fifa\_2017\_integer\_rem), order="hclust", tl.cex = 1)

# Print the list of variables removed

print("The variables that are removed after performing correlation check are")

print(removed\_col)

# Setting dummy Variables

dmy <- dummyVars(" ~ .", data = fifa)

fifa <- data.frame(predict(dmy, newdata = fifa))

# Near zero variance

fifa <- fifa[, -nearZeroVar(fifa)]

# Write the processed data for future reference

write.csv(fifa\_2017, file = "data/fifa\_processed.csv", row.names = FALSE)

## Load the processed data

fifa <- read.csv("data/fifa\_processed.csv")

## Model Creation

# Setting the seed for reproduciablity

set.seed(1)

# Performing data spliting

cv\_index <- createDataPartition(fifa$Rating, p = 0.8, list = FALSE)

fifaTrain <- fifa[cv\_index, ]

fifaTest <- fifa[-cv\_index, ]

X\_train <- fifaTrain[, !colnames(fifaTrain) %in% c("Rating")]

X\_test <- fifaTest[, !colnames(fifaTest) %in% c("Rating")]

y\_train <- as.numeric(fifaTrain[, "Rating"])

y\_test <- as.numeric(fifaTest[, "Rating"])

# Setting up the control parameter

ctrl <- trainControl(method = "cv", number = 5)

### Linear Models

### Linear model

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the KNN model

if(file.exists("models/lm\_model.rds")) {

lm\_model <- readRDS("models/lm\_model.rds")

} else {

lm\_model <- train(x = X\_train, y = y\_train, method = "lm", trControl = ctrl)

saveRDS(lm\_model, "models/lm\_model.rds")

}

# Print the model

lm\_model

# Predict the model

lm\_pred <- predict(lm\_model, X\_test)

# Plot the model

plot(lm\_pred, y\_test, cex = 1.3, pch = 16, col = "blue", xlab = "Predicted", ylab = "Observed", main = "Predicted vs Observed")

# Get the test Set performance metrics

postResample(pred = lm\_pred, obs = y\_test)

### Partial Least Square

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the PLS model

if(file.exists("models/pls\_model.rds")) {

pls\_model <- readRDS("models/pls\_model.rds")

} else {

pls\_model <- train(x = X\_train, y = y\_train, method = "pls", trControl = ctrl, preProcess = c("center", "scale"), tuneLength = 20)

saveRDS(pls\_model, "models/pls\_model.rds")

}

# Print the model

pls\_model

# Plot the results

plot(pls\_model, type = c("p", "g"), xlab = "Components", ylab = "RMSE")

# Make prediction on the test set

pls\_pred <- predict(pls\_model, X\_test, ncomp = as.numeric(pls\_model$bestTune))

# Get the test Set performance metrics

postResample(pred = pls\_pred, obs = y\_test)

### Ridge Regression

# Create the grid for the ridge model

ridge\_grid <- data.frame(.lambda = seq(0, .1, length = 20))

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the Ridge model

if(file.exists("models/ridge\_model.rds")) {

ridge\_model <- readRDS("models/ridge\_model.rds")

} else {

ridge\_model <- train(x = X\_train, y = y\_train, method = "ridge", trControl = ctrl, preProcess = c("center", "scale"), tuneGrid = ridge\_grid)

saveRDS(ridge\_model, "models/ridge\_model.rds")

}

# Print the model

ridge\_model

# Plot the results

plot(ridge\_model, type = c("p", "g"), xlab = "Lambda", ylab = "RMSE")

# Make prediction on the test set

ridge\_pred <- predict(ridge\_model, as.matrix(X\_test), s = as.numeric(ridge\_model$bestTune))

# Get the test Set performance metrics

postResample(pred = ridge\_pred, obs = y\_test)

### Lasso Regression

# Create the grid for the ridge model

lasso\_grid <- expand.grid(.fraction = seq(0.2, 1, length = 20))

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the lasso model

if(file.exists("models/lasso\_model.rds")) {

lasso\_model <- readRDS("models/lasso\_model.rds")

} else {

lasso\_model <- train(x = X\_train, y = y\_train, method = "lasso", trControl = ctrl, preProcess = c("center", "scale"), tuneGrid = lasso\_grid)

saveRDS(lasso\_model, "models/lasso\_model.rds")

}

# Print the model

lasso\_model

# Plot the results

plot(lasso\_model, type = c("p", "g"), xlab = "Fraction", ylab = "RMSE")

# Make prediction on the test set

lasso\_pred <- predict(lasso\_model, as.matrix(X\_test), s = as.numeric(lasso\_model$bestTune[1]), mode = "fraction")

# Get the test Set performance metrics

postResample(pred = lasso\_pred, obs = y\_test)

### Elastic Net

# Develop the grid

enetGrid <- expand.grid(.lambda = c(0, 0.01, .1), .fraction = seq(0.05, 1, length = 20))

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the enet model

if(file.exists("models/enet\_model.rds")) {

enet\_model <- readRDS("models/enet\_model.rds")

} else {

enet\_model <- train(x = X\_train, y\_train, method = "enet", tuneGrid = enetGrid, trControl = ctrl, preProc = c("center", "scale"))

saveRDS(enet\_model, "models/enet\_model.rds")

}

# Print the model

enet\_model

# Plot the paramter to see the best parameter

plot(enet\_model, type = c("p", "g"), xlab = "Fraction", ylab = "RMSE")

# Make prediction on the test set

enet\_pred <- predict(enet\_model, as.matrix(X\_test), s = as.numeric(enet\_model$bestTune), mode = "fraction")

# Get the test Set performance metrics

postResample(pred = enet\_pred, obs = y\_test)

### Non Linear Models

#### KNN

# Set the seed

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the KNN model

if(file.exists("models/knn\_model.rds")) {

knn\_model <- readRDS("models/knn\_model.rds")

} else {

knn\_model <- train(x = X\_train, y = y\_train, method = "knn", preProc = c("center", "scale"), tuneLength = 10, trControl = ctrl)

saveRDS(knn\_model, "models/knn\_model.rds")

}

# Print the model

knn\_model

# Plot the model

plot(knn\_model)

# Predict the model

knn\_pred <- predict(knn\_model, newdata = X\_test)

# Get the test Set performance metrics

postResample(pred = knn\_pred, obs = y\_test)

#### Neural Network

# Create the grid for the network

nn\_grid <- expand.grid(.decay = c(0, 0.01, 0.1), .size = 1:10)

# Set the seed

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the model

if(file.exists("models/nnet\_model.rds")) {

nnet\_model <- readRDS("models/nnet\_model.rds")

} else {

nnet\_model <- train(x = X\_train, y = y\_train, tuneGrid = nn\_grid, method = "nnet", preProc = c("center", "scale"), trControl = ctrl,

linout = TRUE, trace = FALSE, MaxNWts = 10 \* (ncol(X\_train)+1) + 10 + 1, maxit=500)

saveRDS(nnet\_model, "models/nnet\_model.rds")

}

# Print the model

nnet\_model

# Plot the model

plot(nnet\_model)

# Predict the test set

nnet\_pred <- predict(nnet\_model, newdata = X\_test)

# Get the test Set performance metrics

postResample(pred = nnet\_pred, obs = y\_test)

#### Averaged Neural Network

# Create the tune grid

tune\_grid <- expand.grid(.decay = c(0, 0.01, .1), .size = 1:10, .bag = FALSE)

# Setting the seed

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the model

if(file.exists("models/avg\_nnet\_model.rds")) {

avg\_nnet\_model <- readRDS("models/avg\_nnet\_model.rds")

} else {

avg\_nnet\_model <- train(x = X\_train, y = y\_train, tuneGrid = tune\_grid, method = "avNNet", preProc = c("center", "scale"),

linout = TRUE, trace = FALSE, MaxNWts = 10 \* (ncol(X\_train) + 1) + 10 + 1, maxit = 500)

saveRDS(avg\_nnet\_model, "models/avg\_nnet\_model.rds")

}

# Print the model

avg\_nnet\_model

# Plot the model

plot(avg\_nnet\_model)

# Make the prediction

avg\_nnet\_pred <- predict(avg\_nnet\_model, newdata = X\_test)

# Get the performance scores

postResample(pred = avg\_nnet\_pred, obs = y\_test)

#### Mars Model with no preprocessing

# Create the tune grid

tune\_grid <- expand.grid(.degree = 2:4, .nprune = 30:50)

# Setting the seed

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the model

if(file.exists("models/mars\_model.rds")) {

mars\_model <- readRDS("models/mars\_model.rds")

} else {

mars\_model <- train(x = X\_train, y = y\_train, trControl = ctrl, tuneGrid = tune\_grid, method = "earth")

saveRDS(mars\_model, "models/mars\_model.rds")

}

# Print the model

mars\_model

# Plot the model

plot(mars\_model)

# Make the prediction

mars\_pred <- predict(mars\_model, newdata = X\_test)

#### Test set

# Get the performance scores

postResample(pred = mars\_pred, obs = y\_test)

#### Support Vector Machine

# Setting the seed

set.seed(1)

# Check the file exists and load to variables

# else bulid and store the model

if(file.exists("models/svm\_model.rds")) {

svm\_model <- readRDS("models/svm\_model.rds")

} else {

svm\_model <- train(x = X\_train, y = y\_train, trControl = ctrl, tuneLength = 14, method = "svmRadial", preProc = c("center","scale"))

saveRDS(svm\_model, "models/svm\_model.rds")

}

# Print the model

svm\_model

# Plot the model

plot(svm\_model)

# Make the prediction

svm\_pred <- predict(svm\_model, newdata = X\_test)

# Get the performance scores

postResample(pred = svm\_pred, obs = y\_test)

#### Variable Importance

# Neural Network

nnet\_imp <- varImp(nnet\_model)

plot(nnet\_imp, top = 10, main = "Neural Network Important predictors")