Contents

1. Library package

library(MLmetrics) # used for Accuracy();Specificity();Sensitivity()

library(caret) # used to split data into train and test ; to impute NA values using preProcess(df, method = 'medianImpute')

```
set.seed(2000)
train_test_split <- createDataPartition(df$target, p = 0.7) # 70 % in train 30% target.
train <- df[train_test_split$Resample1,]
validation <- df[-train_test_split$Resample1,]
```

library(ROCR) # for prediction() ; performance() for logistic regression

library(car) # to get vif values vif()

library(MASS) # to get stepAIC function stepAIC()

library(DMwR)# for central imputation

library(glmnet) # for Lasso and Elastic-Net Regularized Generalized Linear Models

library(dummies) # to do dummification

library(sqldf) # to write SQL-like commands in R to aggregate the data [data = sqldf("SQL Cmd")]

library(zoo) # {Used in Time Series}NA value replacing each NA with the most recent non-NA prior to it

library("factoextra")

```
# install.packages("devtools")

devtools::install_github("kassambara/factoextra"
```

```
library(MLmetrics)  # used for Accuracy();Specificity();Sensitivity()
library(caret)  # used to split data into train and test   ; to impute NA values using preProcess(df, method = 'medianImpute' )
library(ROCR)  # for prediction() ; performance() for logistic regression
library(car) # to get vif values vif()
library(MASS) # to get stepAIC function  stepAIC()
library(DMwR)# for central imputation
library(glmnet)  #  for  Lasso and Elastic-Net Regularized Generalized Linear Models
library(dummies) # to do dummification
library(sqldf) # to write SQL-like commands in R to aggregate the  data [data = sqldf("SQL Cmd")]
library(zoo) # {Used in Time Series}NA value replacing each NA with the most recent non-NA prior to it
library("factoextra")
```

```
# Library for the timeseries work / model
library(quantmod)
library(sqldf)
library(zoo)
library(TTR)
library("forecast")
```

2. Read the data

Clear the global variable and set the working directory

```
rm(list=ls(all=T))
setwd("C:/Users/Adi/Desktop/INSOFE/Batch 43/20180519/lab/")
```

Load variables if already saved in rstudio For now we don't use the load function

```
#save.image("C:/Users/Adi/Desktop/INSOFE/Batch 43/20180520/lab/savethedata.RData")

load("Data.RData")
```

To save the environment variables use save.image

3. See the data

```
# to get the all the dataframe info
dataSummary<- function(df) {

  # to clear console we use ctrl+L the code is "\014"
  cat("\014")
  cat("\n\n=======================Summary====================\n\n")
  print(summary(df))
  cat("\n\n=======================structure==================\n\n")
  print(str(df))
  cat("\n\n=======================Check for no of NA in Data set==================\n\n")
  print(sum(is.na(df)))
  cat("\n\n=======================Check for no of NA in each column of
dataset==================\n\n")
  print(sapply(df, function(x) sum(is.na(x))))
  cat("\n\n=======================Check for no of unique values in each column of
dataset==================\n\n")
  print(sapply(df, function(x) length(unique(x))))
  cat("\n\n=======================Check for no of empty values in each column of
dataset==================\n\n")
  print(sapply(df,function(x) length(which(x==''))))
cat("\n\n
==========================================================================================
===========\n ")
  }
```

Remove columns which has more than 70% unique values

```
uni <- sapply(df, function(x) length(unique(x)))
dim(df)
R = names(uni[ uni > nrow(df)*0.7 ])
df <- df[ , !(names(df) %in% R)]
dim(df)
```

Remove Duplicate rows after removing indices

```
# Check Uniqueness features and remove them from Train & Test
unique_values.train <- apply(df_train, 2, function(x) length(unique(x)))
unique_values.train
# Drop feature "PersonID"
drop_cols.train     <- which(unique_values.train == nrow(df_train))
df_train          <- subset(df_train, select = -c(drop_cols.train))
df_test           <- subset(df_test, select = -c(drop_cols.train))
str(df_train) # 11 Features
str(df_test)  # 10 Features
```

```
# Check for duplicates
```

```
table(duplicated(df_train)) #425 datapoints are duplicates
#Remove Dulictes
df_train <- distinct(df_train)
nrow(df_train)
```

4. Deal with NA

Once we get to know we have 2 options

Option a:- Remove the NA rows and build the model do this is no of NA rows is less than 10% else we need to go for option b to impute the NA values .

Remove NA rows from Data Frame:-

```
df = na.omit(df)
```

Option b:- Impute the NA values

I. No information is present or can't infer what need to be replaced use

a. Mean and build a model for numeric or int data type

```
df$Variable_NAME[is.na(df$Variable_NAME)] <- mean(df$Variable_NAME,na.rm=T)
```

b. Media and build a model for numeric or int data type

```
df$Variable_NAME[is.na(df$Variable_NAME)] <- median(df$Variable_NAME,na.rm=T)
```

c. Mode for the categorical variable.

```
df$Variable_NAME[is.na(df$Variable_NAME)] <-
names(which.max(table(df$Variable_NAME)))
```

II. Impute the NA values with following options

a. centralImputation

This function fills in any NA value in all columns of a data frame with the statistic of centrality (given by the function centralvalue()) of the respective column

```
centralImputation(df)
```

b. Knn Imputation

Function that fills in all NA values using the k Nearest Neighbours of each case with NA values. By default it uses the values of the neighbours and obtains an weighted (by the distance to the case) average of their values to fill in the unknows. If meth='median' it uses the median/most frequent value, instead.

```
knnImputation(df, k = 10, scale = T, meth = "weighAvg", distData = NULL)
```

We use same knn imputation for train and the impute it for validate and test

```
library(DMwR)
train_imputed <- knnImputation(train[,!names(train) %in% 'targetFeature'], k = 5, scale
= T)
test_imputed <- knnImputation(test[,!names(test) %in% 'mpg'], k = 5, scale = T,
                        distData = train[,!names(train) %in% 'mpg'])
```

c. Impute using other features with in the data frame

First get how strong is the dependence with feature which has NA and other feature

Dependence test is aov pvalue <0.05 means both features have strong significance.

```
print(summary(aov(featureWithNA~feature,  data  = df)))
```

```
df %>% group_by(feature{which has empty data/NA }) %>% summarise(mean =
mean(feature))
```

Impute the value from the other feature

featureName which has empty value

```
df[which(df$featureName==''),'featureName'] <- value {'c',2}
```

d. Impute using preprocess package **{Do imputation after dividing train and test }**

```
preprocess_object <- preProcess(train, method = 'medianImpute' )
train <- predict(preprocess_object, newdata = train)
validation <- predict(preprocess_object, newdata = validation)
```

e. Impute for time Series

For time series we need zoo package na.locf(df$feature)

```
df$MIN_PRICE=(na.locf(df$MIN_PRICE) + na.locf(df$MIN_PRICE,fromLast = TRUE) )/2
```

5. Data Pre Processing

Remove unique attributes from data frame

```
#Lets check the uniqueness in attribute values
unique_values <- apply(train, 2, function(x) length(unique(x)))
#unique_values_test <- apply(test, 2, function(x) length(unique(x)))
drop_cols <- names(which(unique_values == nrow(train))) #Drop 'PassengerId' and 'Name'
# drop_cols_test <- names(which(unique_values == nrow(test)))
train[,drop_cols] <- test[,drop_cols] <- NULL
```

Convert a list of features into a factor

```
here Survived is the target so we don't go to factor this in test
factor_vars <- c('Survived','Pclass')
train[,factor_vars] <- as.data.frame(lapply(train[,factor_vars], function(x) as.factor(x)))
test$Pclass <- as.factor(test$Pclass)
```

Check how many empty cells are there if its above 70% get rid of the column

```
print(length(which(df$feature==''))/sum(table(df$feature)))
df$feature <- NULL
```

table():- to get th e frequency or a table form

prop.table(table()):- to get the probability of table

6. Build Model

a. Get the Correlation Matrix
b. Build model with all features
c. Build Model after removing outliers or clipping outliers
d. Do vif and get rid of some features
e. Do stepAIC
f. Do standardization and then through to run regression

a. Linear Model

Build Model with All variables in it
cor(train) to get correlation matrix;

```
#### Get Correlation from given data
C <- cor(df,df$target)
names(C) <- cNames # cnames means column names
C[C > 0.5 | C < -0.5]
corrplot::corrplot(cor(df),title = "Correlaton Matrix of DF",method = "circle",type = "upper",diag = F )
corrplot::corrplot(cor(df),title = "Correlaton Matrix ",method = "number",type = "upper",diag = F )
```

i. Modeling

Model with only intercept

```
#Model0 - linear model with only intercept - least complex model
Model0 <- lm(mpg ~ 1, data = train)
```

Model with all variables

```
control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 45
metric <- "RMSE"
set.seed(seed)
lm_default <- train(Amount ~ . , data = train, method="lm", metric=metric, trControl=control)
print(lm_default)
model1 <- lm(Amount ~ . , data = train)
summary(model1)
```

```
LinReg <- lm(Target~., data=train)
summary(LinReg)
par(mfrow=c(2,2))
plot(LinReg)
#Error verification on train data
regr.eval(train$TotalRevenueGenerated, LinReg$fitted.values)
#Error verification on test data
Pred<-predict(LinReg,newdata = validation)
regr.eval(validation$TotalRevenueGenerated, Pred)
```

ii. Model feature Selection (Check for multi collinerarity ans stepAIC)

WE need to check for Multicollinearity and Variance inflation factor which are present in car package

```
#---------- model based feature selection ----------

#Multicollinearity check
#Variance inflation factor

vif(LinReg)
LinReg_VIF <- lm(TotalRevenueGenerated~.-FrquncyOfPurchase-NoOfGamesBought, data=train)
summary(LinReg_VIF)
# thumbrule: any factor which has more than value 5/10 could be ignored. which is 80% or 90% rsquare.

step <- stepAIC(LinReg_VIF, direction="both")
names(train)
step$anova
LinReg_Aic<- lm(formula = TotalRevenueGenerated ~ City + MinAgeOfChild + Tenure +
NoOfUnitsPurchased +
        FrequencyOFPlay + NoOfGamesPlayed + FavoriteChannelOfTransaction +
        FavoriteGame, data = train)
summary(LinReg_Aic)
```

```
## notice that there is no change in R-square or model significance even after the model parmeters
## have been reduced from 12 to 8. It is always better to have
## few attributes explaining about the target attribute


#Error verification on train data
regr.eval(train$TotalRevenueGenerated, LinReg_Aic$fitted.values)
#Error verification on test data
Pred<-predict(LinReg_Aic,validation)
regr.eval(validation$TotalRevenueGenerated, Pred)
```

iii. Residual Analysis

After A linear model is build we go a residual analysis to see how fix is the model

In residual analysis graphs we shouldn't see any partern and residual valuses should be homoscratic

```
#Analyzing residuals
graphics.off()
par(mfrow=c(2,2))
plot((LinReg))
```

iv. Remove Residual outliers

```
#To remove the Residual outliers
residuals = LinReg$residuals
outliers <- boxplot(residuals,plot=T)$out
sort(outliers)
tr_outliers <- train[residuals%in% outliers,]

#Influential Observations
#Leverage
lev= hat(model.matrix(LinReg))
plot(lev)
# Suppose we would like to remove
# records with leverage values greater than 0.2
train[lev>0.2,]
train<-train[lev<0.2,]

#Identify records with high cooks distance
cook = cooks.distance(LinReg)
graphics.off()
plot(cook,ylab="Cooks distances")
max=as.numeric(which.max(cook))
points(max,cook[max],col='red', pch=19)
```

```
train[max,]
cook[max]
names(cook[max])
train <- train[row.names(train) != names(cook[max]),]
#train <- train[-max,]

train$Target <- log(train$TotalRevenueGenerated)
```

v. Error Matrix Function

Error Matrix function will store the model name , rmse values of the test and train data

```
#Error Metrics vector
model_name <- vector()
rmse_train <- vector()
rmse_test <- vector()


model_name <- append(model_name, “modelName” )
rmse_train <- append(rmse_train, RMSE(predict(model, train),train$target))
rmse_test <- append(rmse_test, RMSE(predict(model, test), test$target))
```

Adding all models would be like below..

```
model_df <- data.frame('model_name' = model_info[[1]],
                       'rmse_train' = model_info[[2]],
                       'rmse_test' = model_info[[3]])
```

vi. Dummification and add back to test and train

We are going to do dummification and then add back .cyl and orign are dummied

```
dummy_obj <- dummyVars(~ cyl + origin, data = train)
dummy_vars_train <- predict(dummy_obj, newdata = train)
train$cyl <- train$origin <- NULL
train <- cbind(train, dummy_vars_train)
dummy_vars_test <- predict(dummy_obj, newdata = test)
test$cyl <- test$origin <- NULL
test <- cbind(test, dummy_vars_test)
```

vii. Lasso and Ridge Regression

Here we will look at how the lasso and ridge regression happens for both the regression the command is same, only alpha values changes if alpha is 1 its Lasso if alpha is 0 its ridge

```
model_ridge <- glmnet(as.matrix(train), as.matrix(train[,'mpg']), family = 'gaussian', alpha = 0,
                      standardize = TRUE)
plot(model_ridge, xvar = 'lambda')
ridge_cv <- cv.glmnet(as.matrix(train), as.matrix(train[,'mpg']), family = 'gaussian', alpha = 0,
                      standardize = TRUE)
plot(ridge_cv)
model_ridge_cv <- glmnet(as.matrix(train), as.matrix(train[,'mpg']), family = 'gaussian', alpha = 0,
                         standardize = TRUE, lambda = ridge_cv$lambda.1se)
lambda_val <- ridge_cv$lambda.1se
RMSE(predict(model_ridge, as.matrix(train), s = lambda_val), train$mpg)
RMSE(predict(model_ridge, as.matrix(test), s = lambda_val), test$mpg)
```

viii. Standardize

We need to exclude the target variable and also exclude the categorical variables

```
###Apply standardizations

pre1<-preProcess(train[,setdiff(colnames(train),targetVariable)])
train_scale<-predict(pre1,train[,setdiff(colnames(train), targetVariable)])
val_scale<-predict(pre1,val[,setdiff(colnames(val), targetVariable)])

                                    (OR)

ind_attr <- names(pre_train)[names(pre_train) != target11]
std_method <- preProcess(pre_train[,ind_attr], method = c("center", "scale"))
train_Data <- predict(std_method, pre_train)
test_Data <- predict(std_method, pre_test)
```

Scaling can be done in another way like as below imp note is that train dataset should contain only numerical

```
train_sc <- scale(train)
```

ix. Polynomial Model

To build a polynomial model like $y= x+x_2\text{^}3 +x_3\text{^}4$

```
#Complex models here for example I have kept mpg which is a column name for the dataset and
target is mpg
```

```
run_model <- function(degree, data_train, data_test){
  for(i in 1:degree){
    x <- as.matrix(data_train[,!names(data_train) %in% c('mpg','feature which are categorical')])
    x2 <- poly(x, degree = i)
    final_df <- data.frame(x2, data_train[,c('mpg','feature which are categorical')])
    x_test <- as.matrix(data_test[,!names(data_test) %in% c('mpg','feature which are
categorical')])
    x2_test <- poly(x_test, degree = i)
    final_df_test <- data.frame(x2_test, data_test[,c('mpg','mpg','feature which are categorical')])
    fit <- lm(mpg ~ ., data = final_df)
    model_name <- append(model_name, paste('m',i, sep = ''))
    rmse_train <- append(rmse_train, RMSE(predict(fit, final_df), final_df$mpg))
    rmse_test <- append(rmse_test, RMSE(predict(fit, final_df_test), final_df_test$mpg))
  }
  return(list(model_name,rmse_train,rmse_test))
}

model_info <- run_model(4, train, test)
```

x. PCA

In this we have all features present ; we will not drop features. Also PCA doesn't consider target variable for getting components. Components contain mix of all features.
Before applying PCA we need to standardize the train and test data sets.
Train_scale variable is the standardize train ; and total_UPDRS is the target variable.
PCA does center scaling (difference with mean only)

```
pca <- princomp(d)
pca$loadings  # these are the eigenvectors
pca$scores # this is the scoresCent .
```

Applying PCA to a data frame after getting train and validate

```
###Apply standardizations
library("factoextra")

pre1<-preProcess(train[,setdiff(colnames(train),"total_UPDRS")])
train_scale<-predict(pre1,train[,setdiff(colnames(train),"total_UPDRS")])
val_scale<-predict(pre1,val[,setdiff(colnames(val),"total_UPDRS")])

# Running a PCA using princomp function only on independent variables
prcomp_train <- princomp(train_scale)


# transformed data in new components
train_data<-prcomp_train$scores
```

```
# get only highest variation components and bind target
train_data<-data.frame(train_data[,comp_Names],"UPDRS"=train$total_UPDRS)

# apply same transformation on test
val_data<-predict(prcomp_train,val_scale)

summary(prcomp_train)
# from here we will decide how many components do we need to build a model
#Cumulative Proportion  from summary tell what would be the threshold value if we want 90%
#see the cumulative proportion of the component which has just crossed 0.90 till the
#components we will keep in the model and remove the rest


#subset the components only which are in train_data and bind target
val_data<-data.frame(val_data[,comp_Names],"UPDRS"=val$total_UPDRS)

model=lm(UPDRS~.,data=train_data)
summary(model)

#error metrics on train_data
preds<-predict(model,train_data)
preds_val<-predict(model,val_data)

library(DMwR)
regr.eval(train_data$UPDRS,preds)
regr.eval(val_data$UPDRS,preds_val)

e <- regr.eval(val_data$UPDRS,preds_val)
e[["rmse"]]
```

xi. PCA

b. Logistic Model

Build Model with All variables in it

```
sum(is.na(train))
sum(is.na(validatation))
#Build logistic regression model
logistic_model <- glm(Survived~., data = train, family = binomial)
summary(logistic_model)
```

Model is ready now predict it on train and test data

```
# Predicting on train data
prob_train <- predict(logistic_model, newdata = train, type = "response")
# Predicting on validation data
prob_val <- predict(logistic_model, newdata = validation, type="response")
```

Predicted Values will be in probability values and values can vary from 0 to 1
We will decide one threshold that if p >0.5 its 1 else its 0

```
#Hard code threshold
pred_train <- ifelse(prob_train < 0.5, 0, 1)
pred_val <- ifelse(prob_val < 0.5, 0, 1)
```

Write a Function to get our Accuracy, specificity, sensitivity
Accuracy = (TruePositive + TrueNegative) / Total
Sensitivity = TruePositive/Overall Positive {all actual positives}
specificity = TrueNegative/Overall Negative {all actual Negatives}

```
library(MLmetrics)
metrics.func <- function(preds, actuals){
  acc <- Accuracy(preds, actuals)
  spec <- Specificity(actuals, preds, positive = '1')
  sens <- Sensitivity(actuals, preds, positive = '1')
  return(list(acc,spec,sens))
}
```

Now get the accuracy, specificity and sensitivity rates for our predictions

```
metrics.func(pred_train, train$Survived)
metrics.func(pred_val, validation$Survived)
```

Now that we have one base model we will do ROC and do a plot of ROC
Tpr -> true Positive Rate ; fpr – False Positive Rate

```
library(ROCR)
pred <- prediction(prob_train, train$Survived)
perf <- performance(pred, measure="tpr", x.measure="fpr")
#Plot the ROC curve using the extracted performance measures (TPR and FPR)
plot(perf, col=rainbow(10), colorize=T, print.cutoffs.at=seq(0,1,0.05))
```

Do Area under Curve (AUC) to get which p value gives max area

```
perf_auc <- performance(pred, measure="auc")
auc <- perf_auc@y.values[[1]];
```

```
auc
```

From graph we get the p-value which needs to be taken and auc is the max area which can be obtained. We get the ROC graph from prob_test and prob_train which was the result of the predictions from our base model.
Here from Graph we got cut off p-value as 0.55 so we adjust our predictions; Before doing ROC we used the p-value as 0.5 and stored in pred_train and pred_test.

```
pred_train_roc <- ifelse(prob_train > 0.55, 1, 0)
pred_val_roc <- ifelse(prob_val > 0.55, 1, 0)
```

Now get the Accu, Specificity and sensitivity

```
metrics.func(pred_train_roc, train$Survived)
metrics.func(pred_val_roc, validation$Survived)
```

We have got our base model. Now we will try to see if we cn improve the accuracies and AUC
Use vif to find any multi-collinearity

```
library(car)
log_reg_vif <- vif(logistic_model)
print(log_reg_vif)
```

Improve the model using stepAIC from our Base Model

```
library(MASS)
log_reg_step <- stepAIC(logistic_model, direction = "both")
```

Now Predict value with the stepAIC Model and get the model metric

```
# Predicting on train data
prob_train_aic <- predict(log_reg_step, newdata = train, type = "response")
# Predicting on validation data
prob_val_aic <- predict(log_reg_step, newdata = validation, type="response")
```

Get the ROC curve for the step AIC model

```
pred_aic <- prediction(prob_train_aic, train$Survived)
perf_aic <- performance(pred, measure="tpr", x.measure="fpr")
#Plot the ROC curve using the extracted performance measures (TPR and FPR)
plot(perf_aic, col=rainbow(10), colorize=T, print.cutoffs.at=seq(0,1,0.5))

perf_aic_auc <- performance(pred, measure="auc")
auc <- perf_aic_auc@y.values[[1]];
perf_aic_auc@y.values[[1]]
```

Now hard code the threshold and get metrics

```
#Hard code threshold
pred_train_aic <- ifelse(prob_train_aic < 0.5, 0, 1)
pred_val_aic <- ifelse(prob_val_aic < 0.5, 0, 1)

metrics.func(pred_train_aic, train$Survived)
metrics.func(pred_val_aic, validation$Survived)
```

Now run this model on the test data

```
titanic_test <- predict(preprocess_object, newdata = titanic_test)
prob_test <- predict(log_reg_step, newdata = titanic_test, type = 'response')
pred_test <- ifelse(prob_test < 0.5, 0, 1)

test_predictions_df <- data.frame('PassengerId' = passenger_id, 'Survived' = pred_test)
```

Write our result to a file so that we can export it

```
# write.csv(test_predictions_df, file = 'titanic_predictions_test_centralimputation.csv',
row.names = F)
```

c. Time Series Model