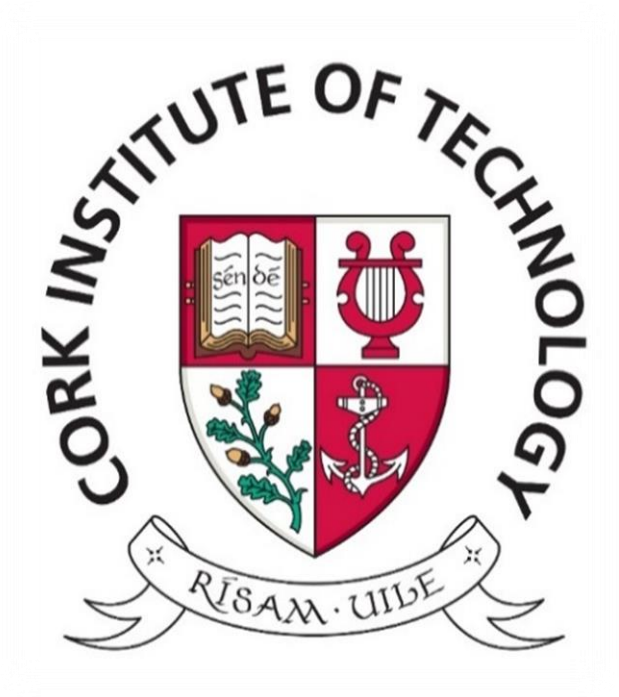
CORK INSTITUTE OF TECHNOLOGY



DEPARTMENT OF MATHEMATICS

MSC IN DATA SCIENCE & ANALYTICS 2019-2020

STUDENT NAME	**SHIVAANI KATRAGADDA**
STUDENT NUMBER	R00183214
MODULE NAME	DATA ANALYTICS & VISUALISATION
MODULE CODE	DATA9005
ASSIGNMENT NUMBER	2
PROFESSORS	ANGEUS DALY
DATE	16TH MAY 2020
DAY	SATURDAY

Declaration of Authorship

I, Shivaani Katragadda, declare that the work presented in this assignment is my own. I confirm that:

- This work was done wholly or mainly while in candidature for the Masters' degree at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- I understand that my project documentation may be stored in the library at CIT and may be referenced by others in the future.

Signed: SHIVAANI KATRAGADDA

Date: 16TH MAY 2020

In this assignment, I was given two datasets namely RF_TrainingDatasetA_Final, RF_ScoringDatasetA_Final.

The first dataset RF_TrainingDatasetA_Final contains of 2186 observations(rows) and 79 variables(columns).This dataset is completely used for building the models.

The second dataset RF_ScoringDatasetA_Final consists of 936 observations(rows) and 77 variables(columns).This dataset is used for predict the output by using the model which were built by using RF_TrainingDatasetA_Final.

RF_TrainingDatasetA_Final contains 79 columns and RF_ScoringDatasetA_Final contains 77 columns, both the dataset have almost same columns but the RF_ScoringDatasetA_Final dataset does not have two columns ENG_Class and Outcome columns which are present in RF_TrainingDatasetA_Final dataset.

The packages used in the assignment are as follows:

```
# install.packages("readxl")
# install.packages("DataExplorer")
# install.packages("dplyr")
# install.packages("psych")
# install.packages("caret")
# install.packages("DMwR")
# install.packages("skimr")
# install.packages("RANN")
# install.packages("visdat")
# install.packages("funModeling")
# install.packages("Hmisc")
# install.packages("inspectdf")
# install.packages("ggplot2")
# install.packages("corrplot")
# install.packages("ggcorrplot")
# install.packages("GGally")

library(readxl)

library(DataExplorer)

library(dplyr)

library(psych)

library(caret)

library(skimr)

library(RANN)

library(visdat)

library(funModeling)
```

```
library(Hmisc)
library(inspectdf)

library(ggplot2)
library(corrplot)

library(ggcorrplot)

library(GGally)

library(DMwR)
```

I loaded the first dataset RF_TrainingDatasetA_Final by using readr package which will be helpful for reading .xlsx files. And assigned file to Train variable.

```
#LOADING THE TRAINING DATASET

Train<-read_excel("F:/semester2/DA&V/DV2/RF_TrainingDatasetA_Final.xlsx")

#checking class of dataset
class(Train)

## [1] "tbl_df"     "tbl"        "data.frame"

#Converting to data frame
Train<- as.data.frame(Train)

class(Train)

## [1] "data.frame"
```

QUESTION A

Investigate the data by carrying out some exploratory data analysis (EDA). Perform the necessary data cleaning/data reduction tasks and outline how you do this in R. Word count 750.

SOLUTION

The aim of the question is to explore the data, clean the data, and perform data reduction tasks.

I will be performing the Exploratory Data Analysis(EDA) to know about the data.

EDA: Exploratory data analysis (EDA) is an approach to the study of data sets, often using visual tools, to outline their basic characteristics. Exploratory Data Analysis performs two main things: 1. It helps clean up a dataset. 2. It gives you a better view of the variables and their relationships. There are mainly three components of exploring data: (i)Understanding

your variables (Numerical data Analysis) (ii)Cleaning your dataset (iii)Analyzing relationships between variables (Graphical Data Analysis)

Firstly I want to know about the given dataset so I will be going through the dataset. In order to understand or to explore the dataset, I used different packages. For descriptive statistics, I used dplyr,skimr,funmodelling,Himsc

Firstly I used some functions such as dim,names,head,tail,str,summary,glimpse,sapply,df_status,freq,plot_num,profiling_num,describe,skim to know about the summary and descriptive statistics of the dataset.

Then I checked for duplicate columns and rows. There is no duplicated data present in the given dataset. To visualize the data I used DataExplorer,Inspectdf,visdat,corrplot,ggcorrplot and GGALLY. This is all about the numerical and graphical statistics of data.

There are three columns DpQ_R2,Fullmint1,Outcome in the RF_TrainingDatasetA_Final which contains missing values. Among these three DpQ_R2,Fullmint1 are numerical columns and the outcome is categorical.So I used the caret package to impute the values for the numerical data columns by using the knnImpute method and the categorical data is imputed by using the mode value. Therefore missing values are imputed and data is cleaned now. This is about the EDA and data cleaning process. Now I will be performing a Data reduction task by using the caret package. In the given dataset all the columns are not important for training the model and to predict the data, so whatever the unnecessary columns are present in the dataset those columns can be deleted by caret package. This process is called pre_processing,in this,there are so many methods such as Creating Dummy Variables,Zero- and Near Zero-Variance Predictors,Identifying Correlated Predictors,Linear Dependencies,The preProcess Function,Centering and Scaling,Imputation Transforming Predictors,Putting It All Together,Class Distance Calculations.

Among this all, I used dummy variables, zero and near zero-variance predictors, identifying correlated predictors

The dummyVars method can be used to produce an entire set of dummy variables from one or more factors. The function takes a formula and a set of numbers and outputs an attribute that can be used using the predict method to create the dummy variables. The dummy variables usually transform all of the categorical variables into numerical data.

In certain cases, the process that produces data will create predictors that have only one unique value. This can cause the model to crash or the suit to become unreliable for certain models. Likewise, predictors could only have a handful of specific values that occur at very low frequencies.

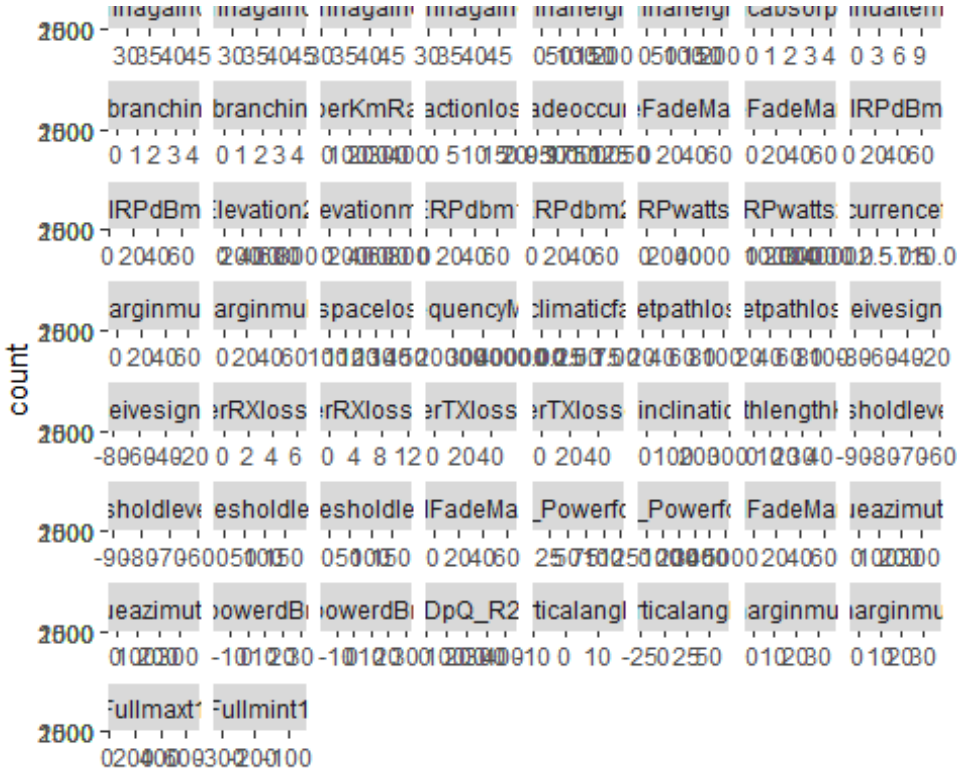
The problem here is that such predictors may become predictors with zero variance when the data is broken into sub-samples with cross-validation or that certain samples may have an improper effect on the model. Many predictors with “near-zero-variance” that need to be defined and removed before modeling. To identify these types of predictors, the following two metrics can be calculated the frequency of the most prevalent value over the second most frequent value which would be near one for well-behaved predictors and very

large for highly-unbalanced data and the "percent of unique values" is the number of unique values divided by the total number of samples that approaches zero as the granularity of the data increases the frequency ratio is greater than a pre-specified threshold and the unique value percentage is less than a threshold, we might consider a predictor to be near zero-variance.

We should not like to incorrectly classify data that has low granularity but is distributed uniformly, such as data from a distinct uniform distribution. These predictors should not be incorrectly identified using both parameters. By default, nearZeroVar will return to issue the positions of the variables that are flagged.

And then identification of correlated predictors While there are some models that thrive on correlated predictors, other models may benefit from decreasing the level of predictors correlation. The findCorrelation method used to find the correlation, I found the correlation of the data and removed all the variables having a correlation of more than 0.7. Finally, the data is reduced to 38 columns from 79 columns. Now the size of the dataset is **2186 rows and 38 columns.**

```
# Performing EXPLANATORY DATA ANALYSIS
#Knowing about the numerica summary of data
dim(Train)

## [1] 2186   79

plot_num(Train)
```




```
any(duplicated(Train))
```

```
## [1] FALSE
```

```
#checking for duplicates
which(duplicated(rownames(Train)))
```

```
## integer(0)
```

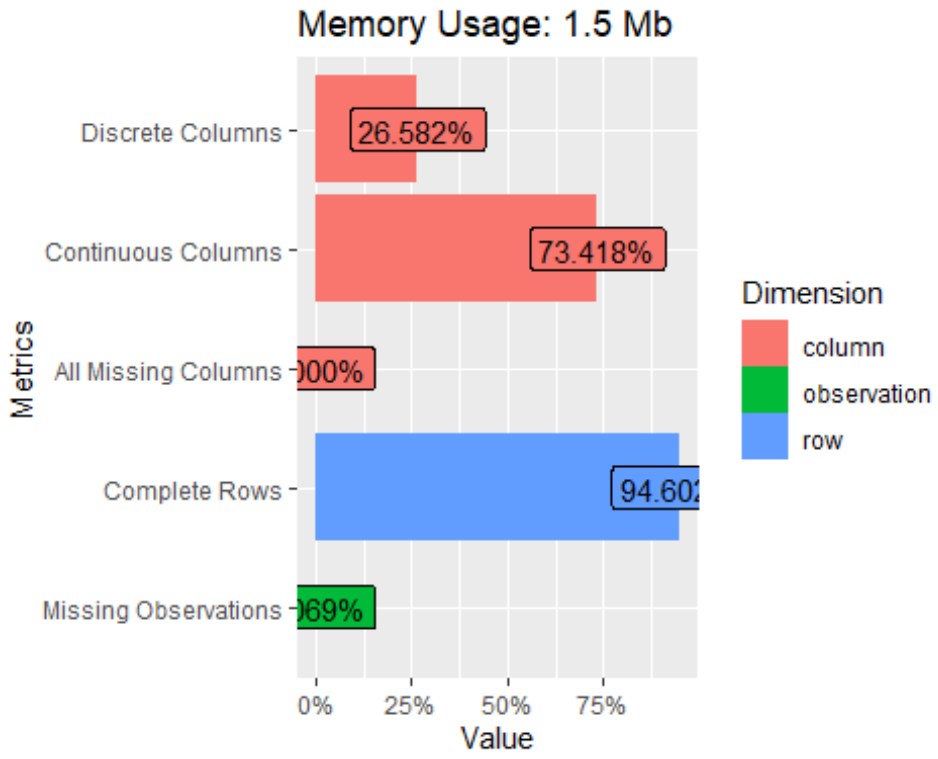
```
which(duplicated(colnames(Train)))
```

```
## integer(0)
```

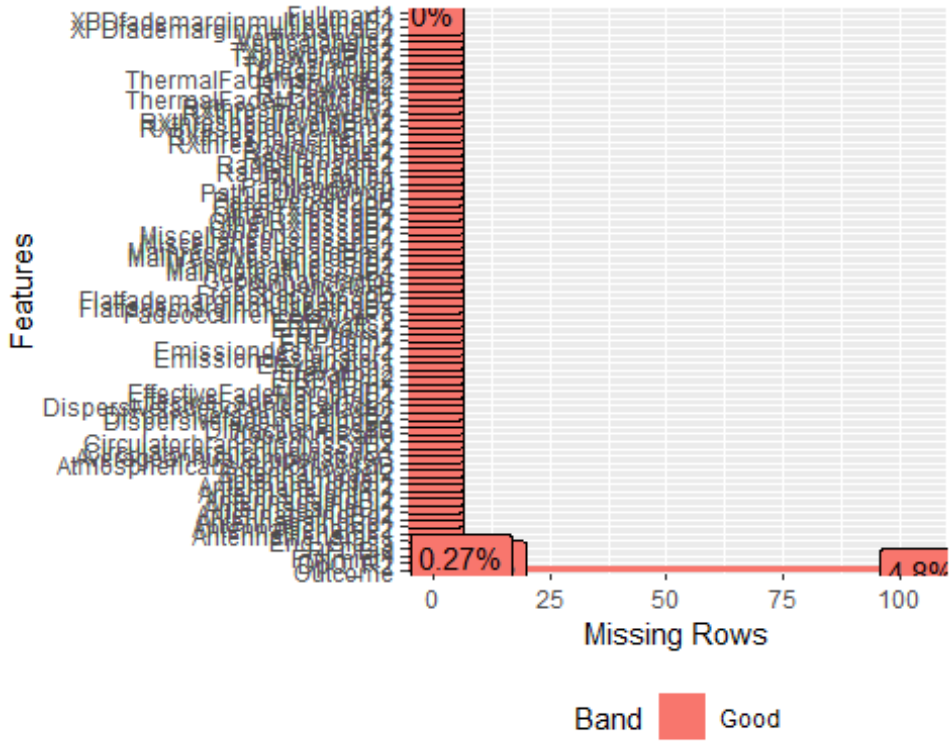
```
#Knowing about the data by using plots and tables

introduce(Train)#tells about the columns
```

```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 2186      79               21                 58                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                  120          2068              172694      1570160
```

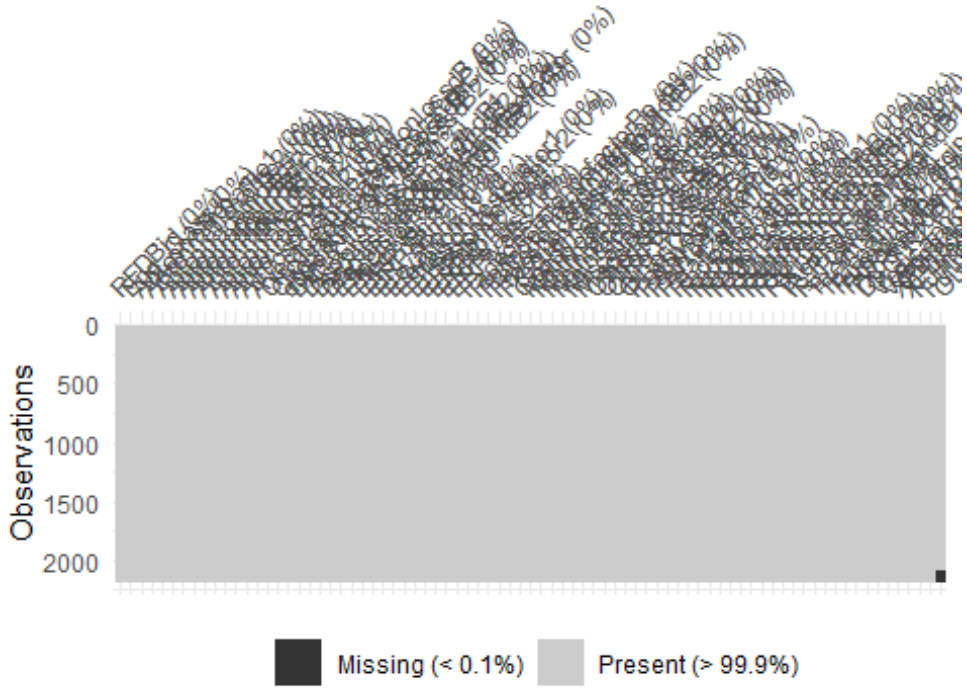
```
plot_str(Train)#plots the graphs by showing all the columns

plot_intro(Train)#gives information about the columns and rows
```



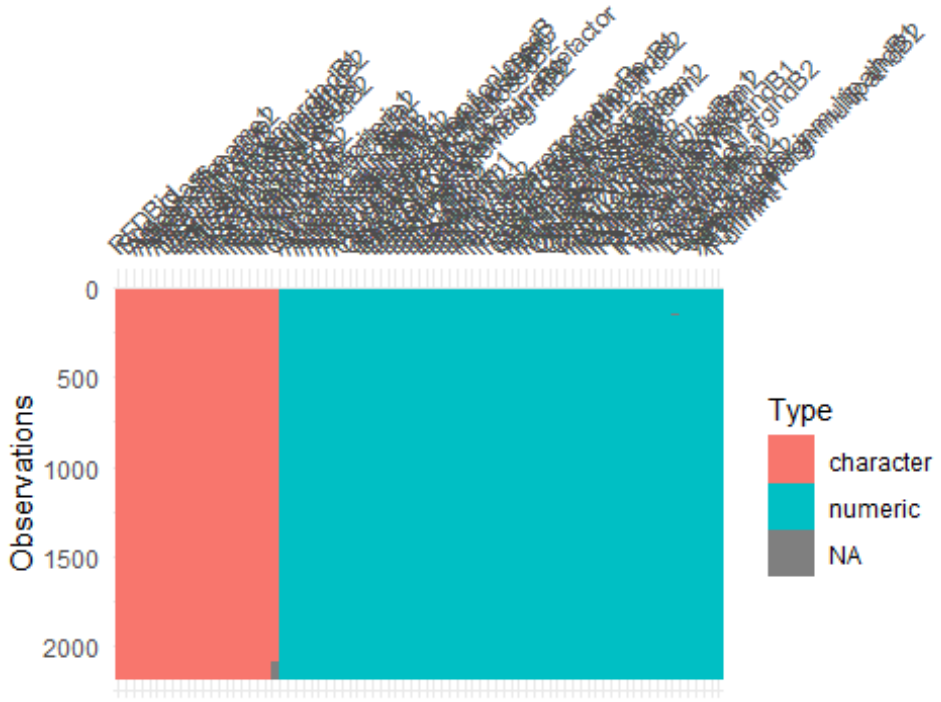

```
plot_missing(Train)#gives information about missing columns data
```



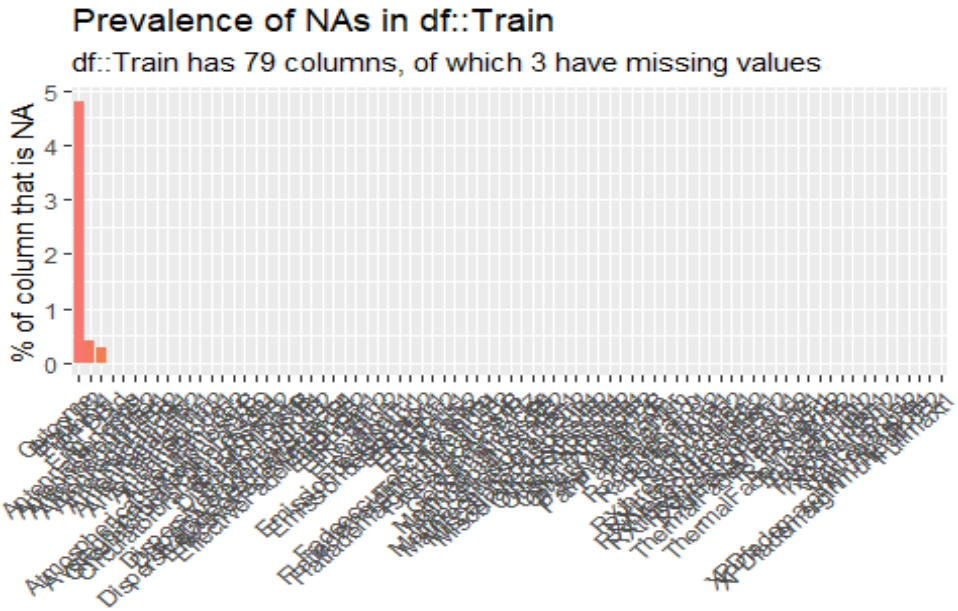

```
#tells about the missing data by usig graphs
vis_miss(Train)
```



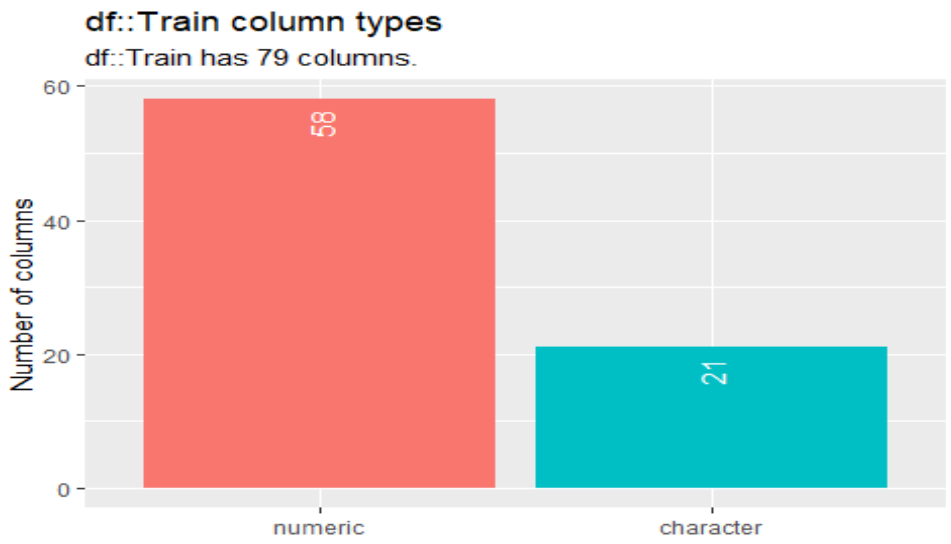
```
vis_dat(Train)
```



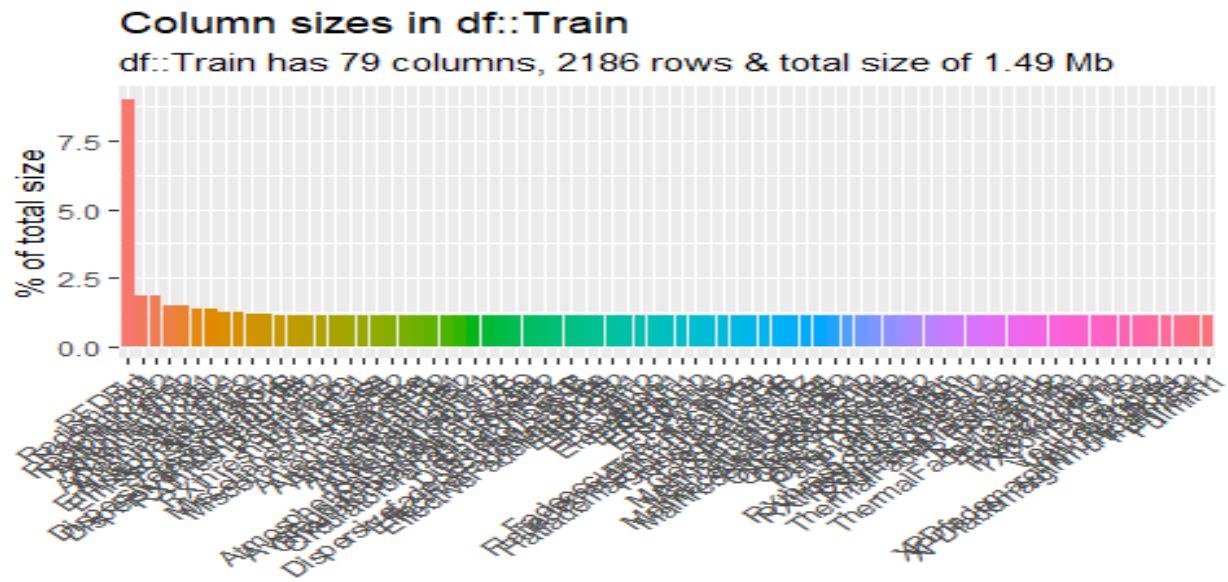

```
#tells about the missing values
b1<-inspect_na(Train)
show_plot(b1)
```



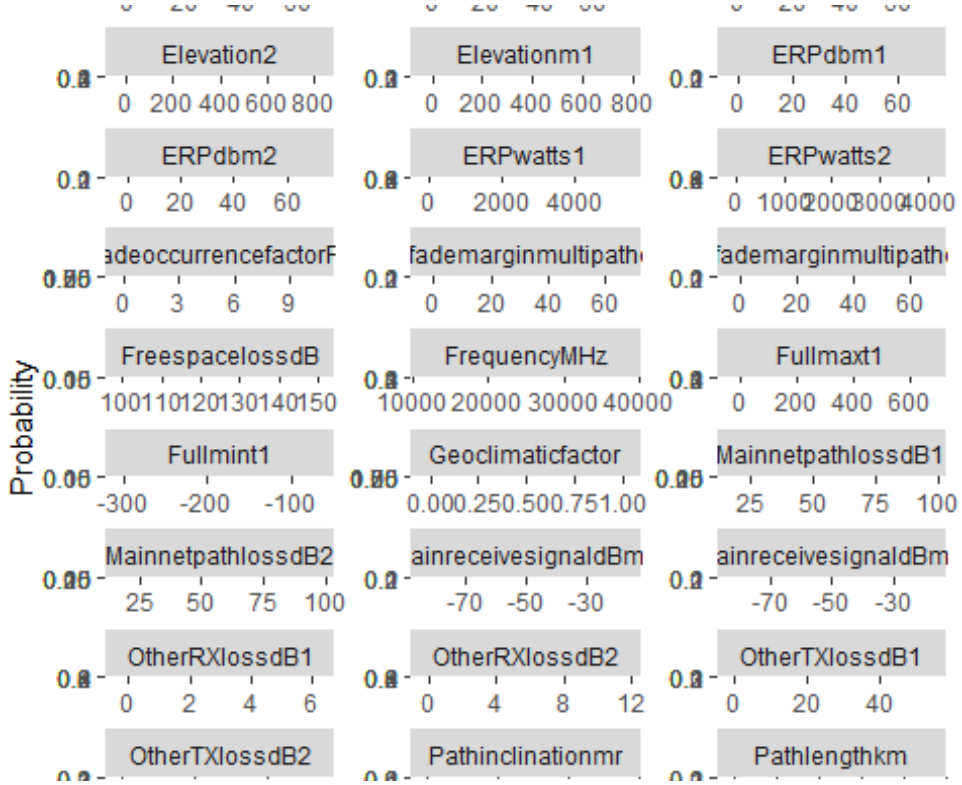
```
#*********************************************************
#Visualisation of dataset by using inspectdf package
#tells about the type of data
a<-inspect_types(Train)
show_plot(a)
```



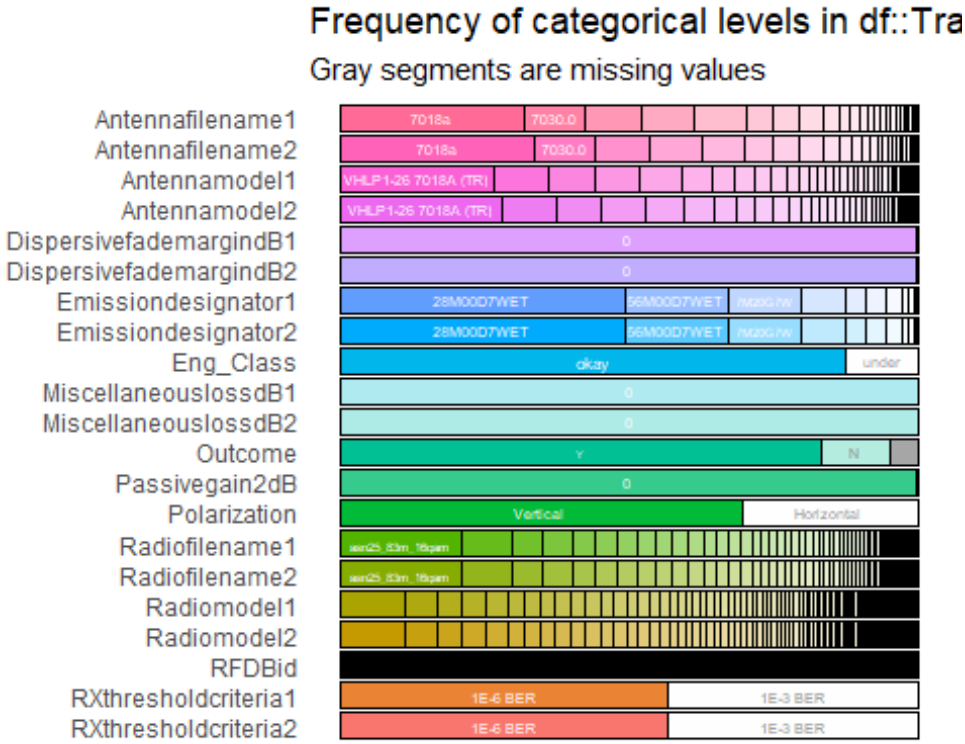

```
#tells about the columns and size
b<-inspect_mem(Train)
show_plot(b)
```



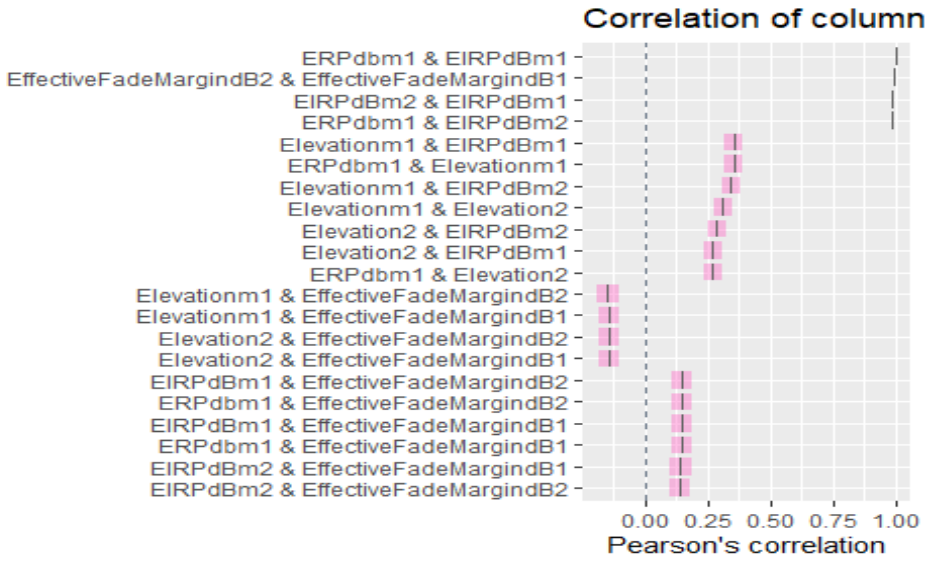

```
#inspect numerical data
b2<-inspect_num(Train)
show_plot(b2)
```



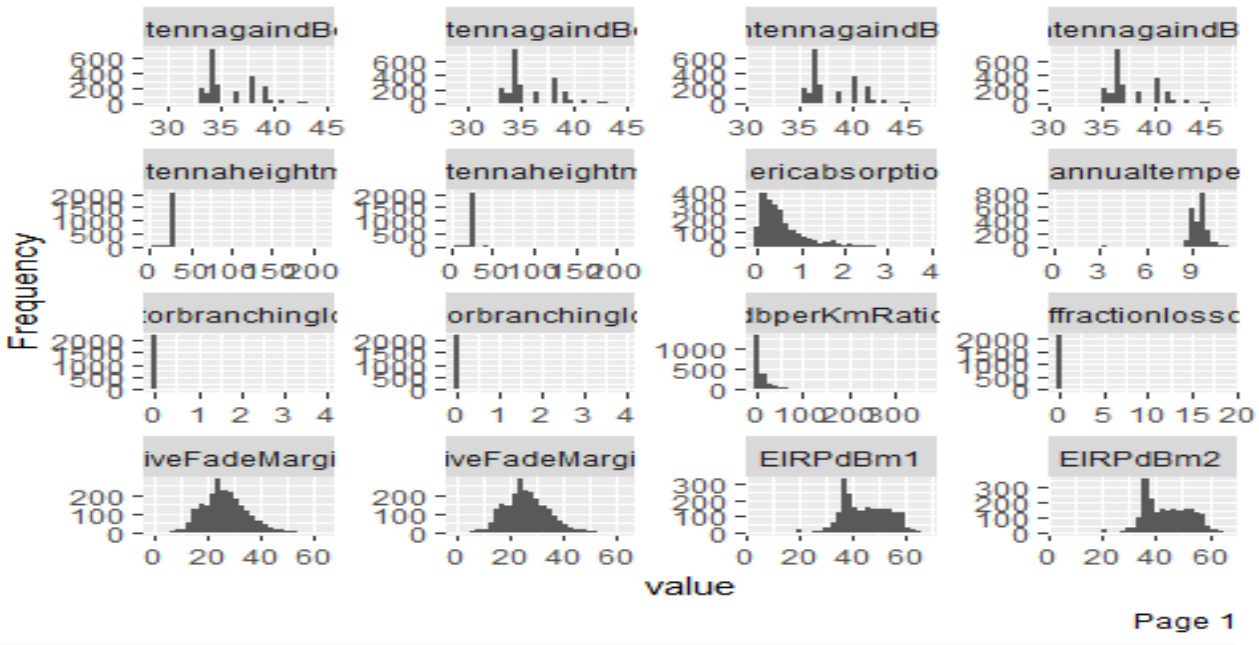

```
#inspect categorical data
b4<-inspect_cat(Train)
show_plot(b4)
```

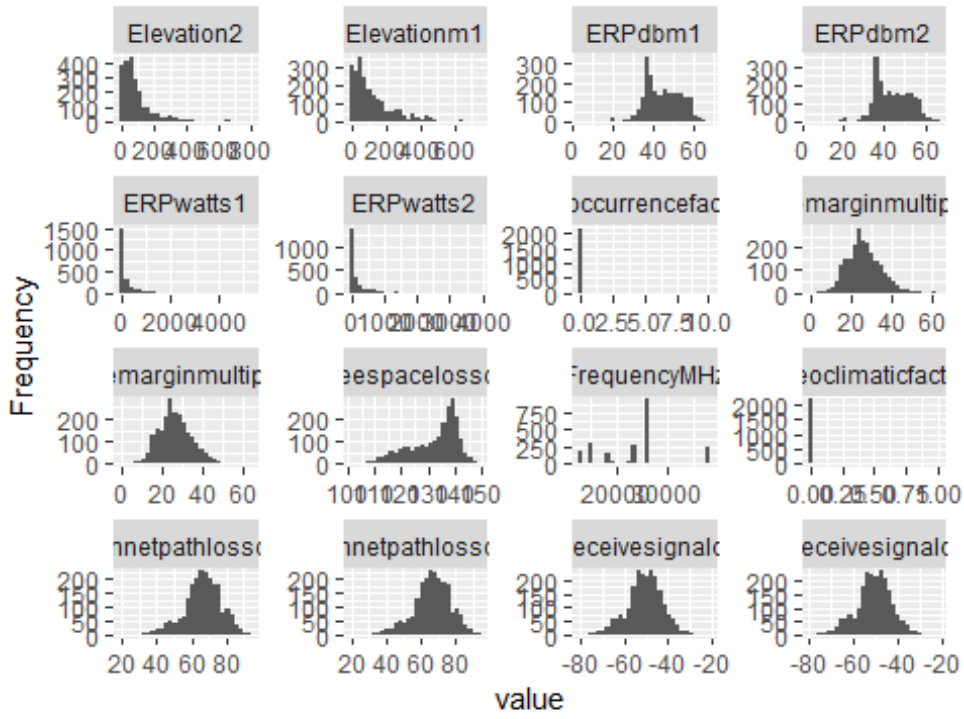


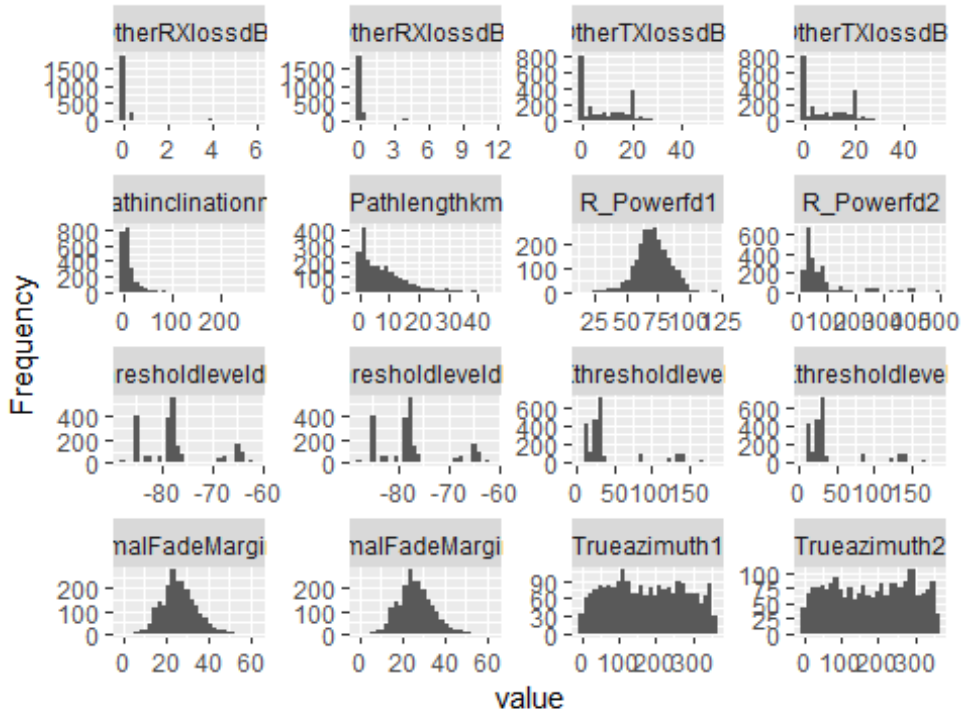
```
#considering some columns
z=Train[,c("EffectiveFadeMargindB1","EffectiveFadeMargindB2","EIRPdBm1","EIRP
dBm2","Elevation2","Elevationm1",
           "ERPdbm1")]
#inspects the correlation
b5<-inspect_cor(z)
show_plot(b5)
```

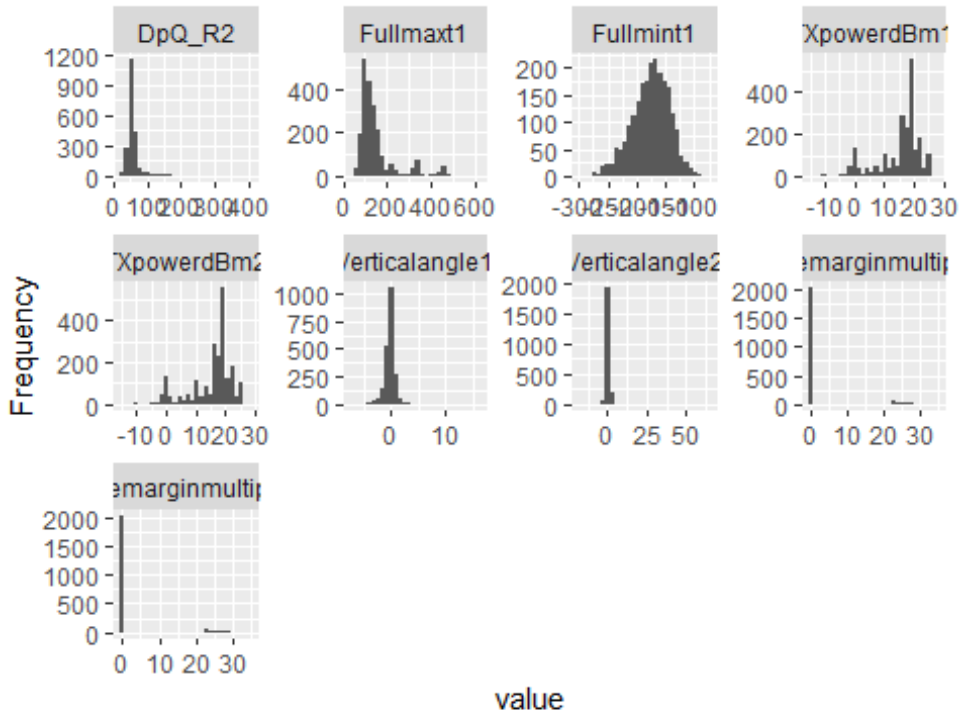



```
#****************************
#Visualisation of dataset by using DataExplorere package package
#histograms of numerical data
plot_histogram(Train)
```

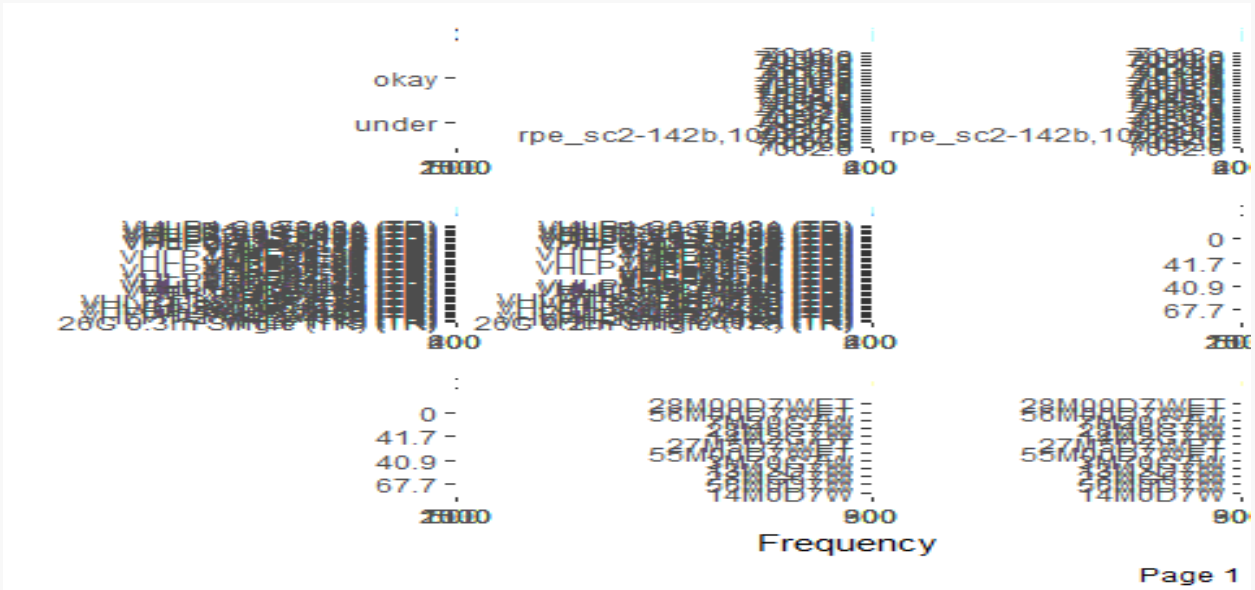


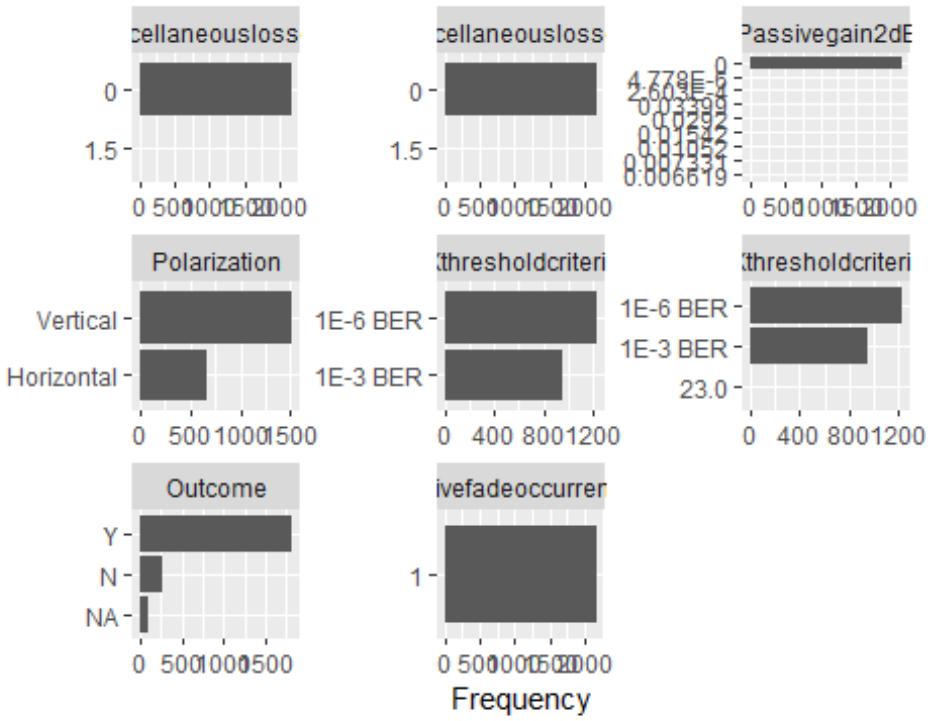

Elevation2
Elevationm1
ERPdbm1
ERPdbm2
ERPwatts1
ERPwatts2
marginmultip
Frequency
value


PathlengthkmR_Powerfd1R_Powerfd2
Trueazimuth1Trueazimuth2
Frequency
value

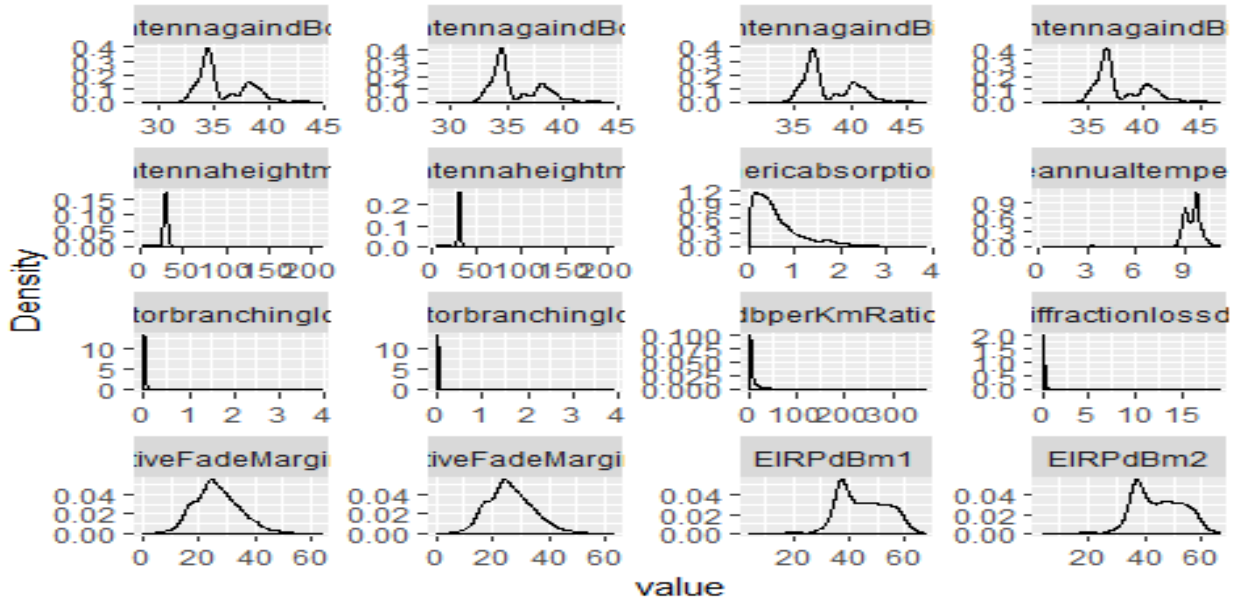


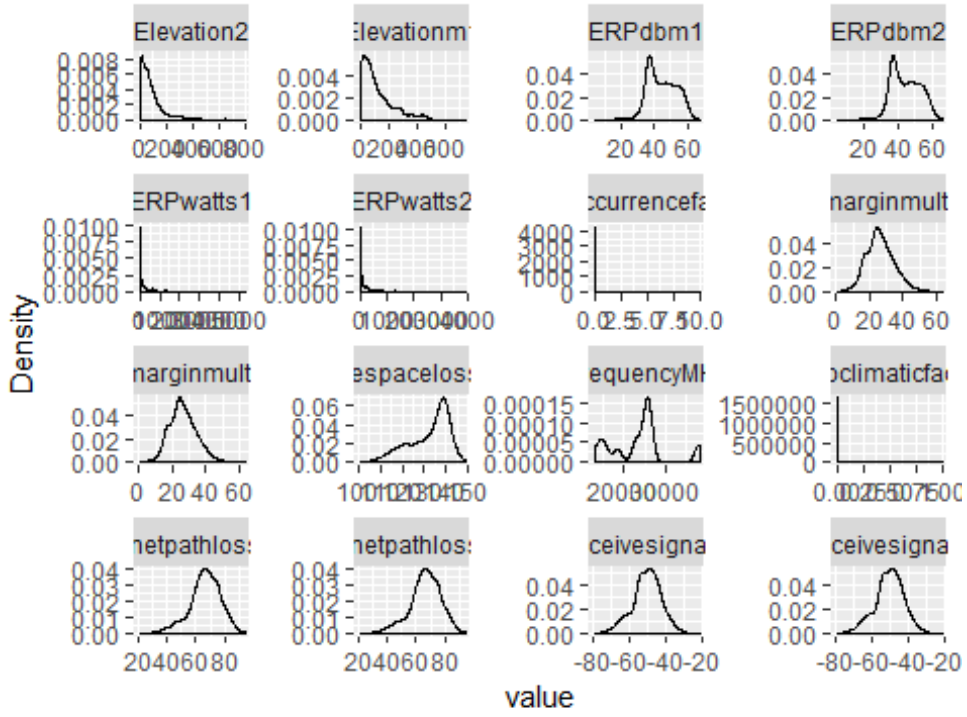

```
# bar plot of categorical data
plot_bar(Train)

## 5 columns ignored with more than 50 categories.
## RFDBid: 2186 categories
## Radiofilename1: 88 categories
## Radiofilename2: 88 categories
## Radiomodel1: 152 categories
## Radiomodel2: 152 categories
```

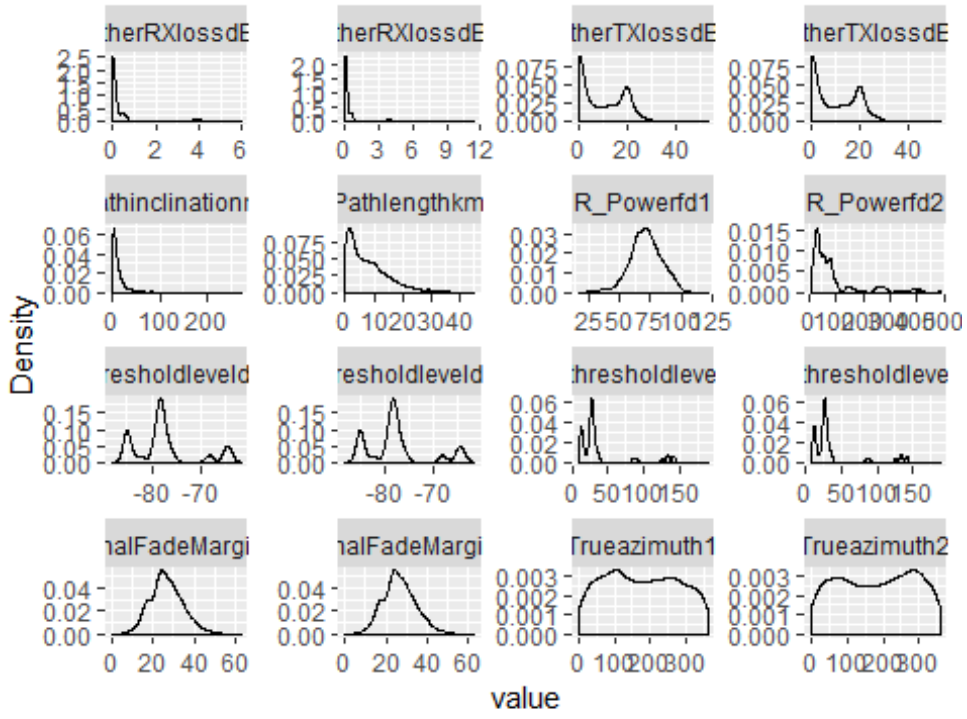


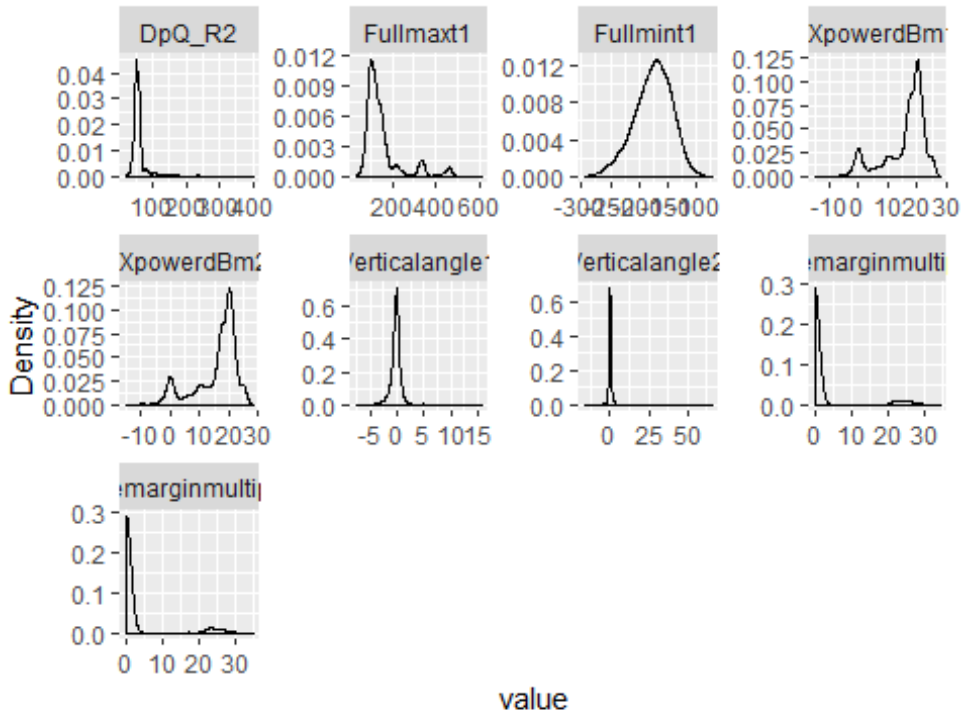



```
#density plot of numerical data
plot_density(Train)
```

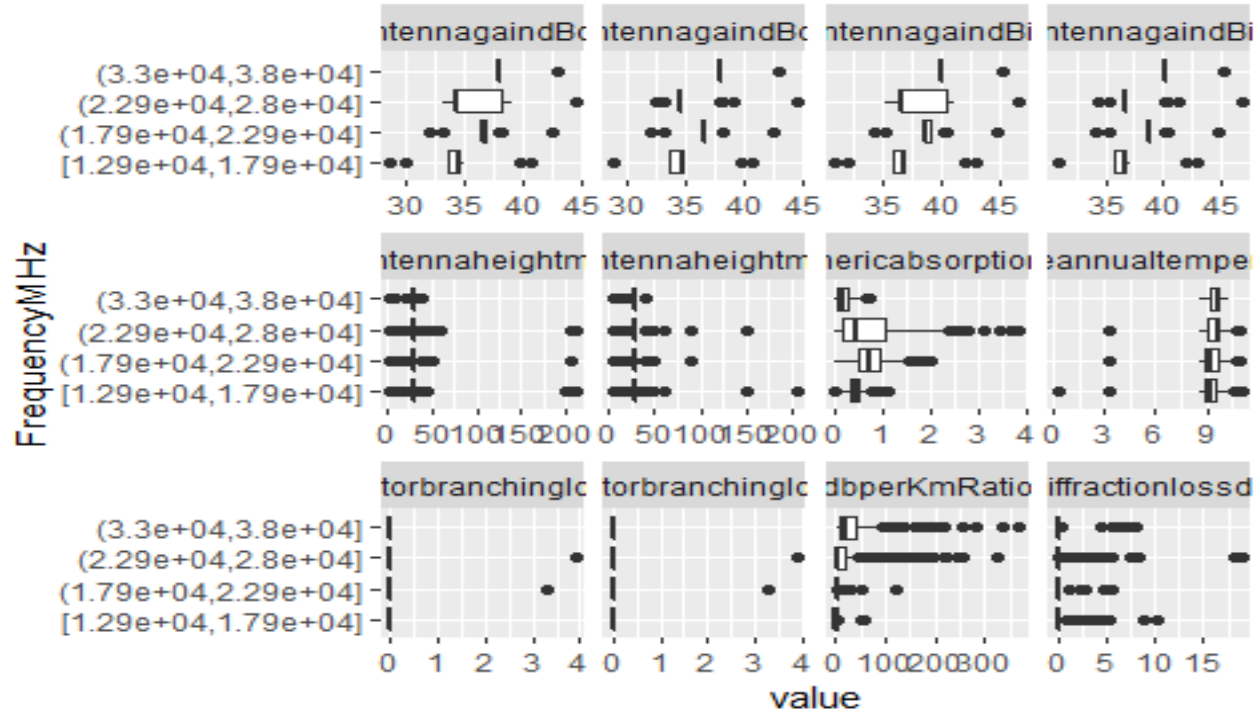


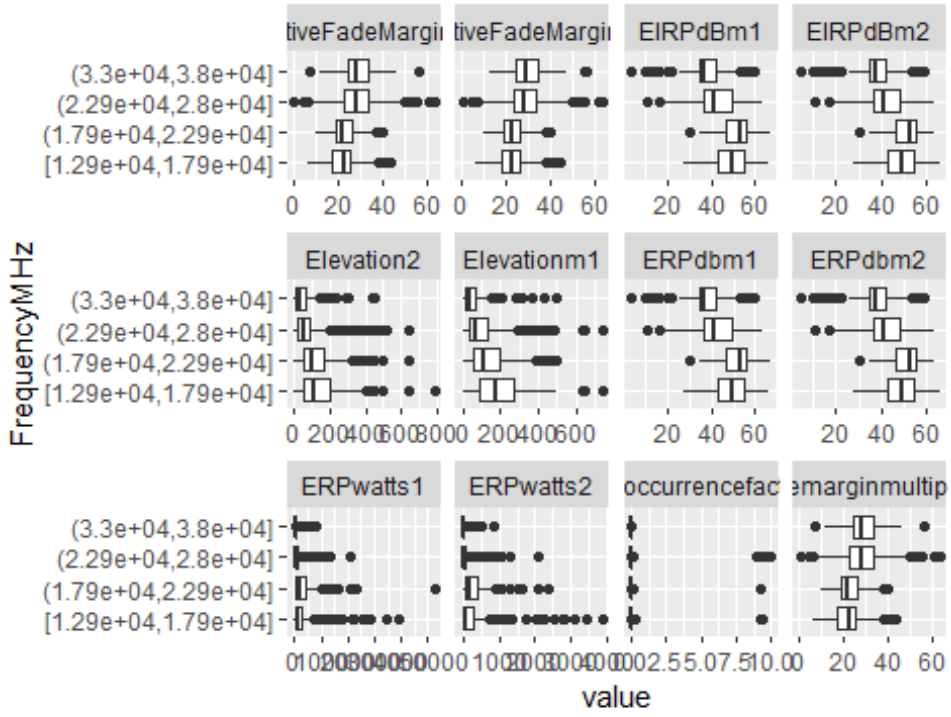

Elevation2
ERPdbm1
ERPdbm2
ERPwatts1
ERPwatts2
Density
value


R_Powerfd1
R_Powerfd2
Density
value

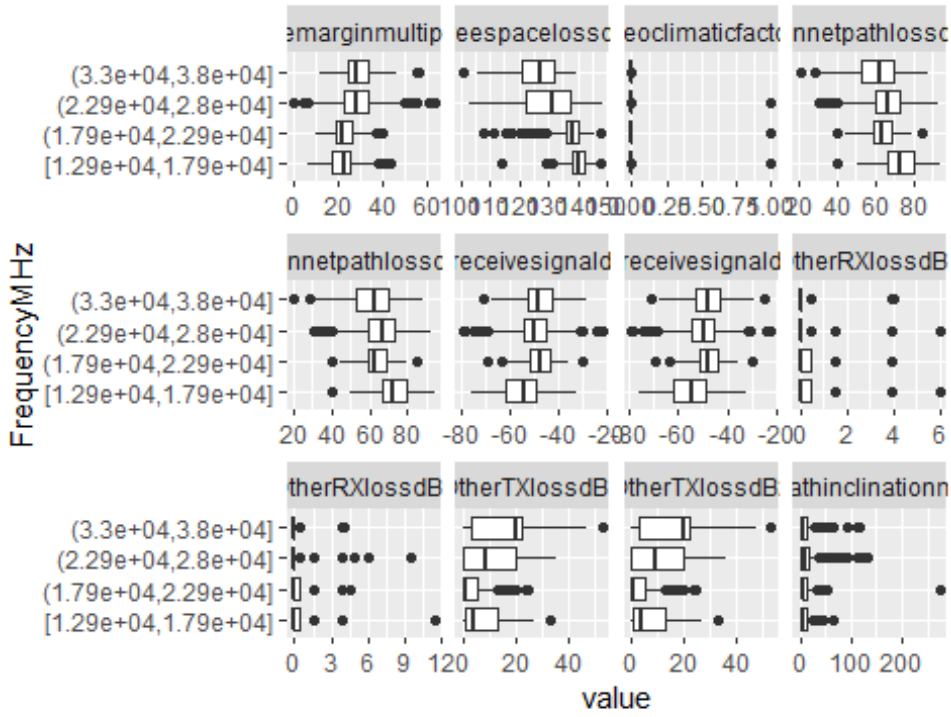



```
# boxplot

plot_boxplot(Train, by = "FrequencyMHz")
```

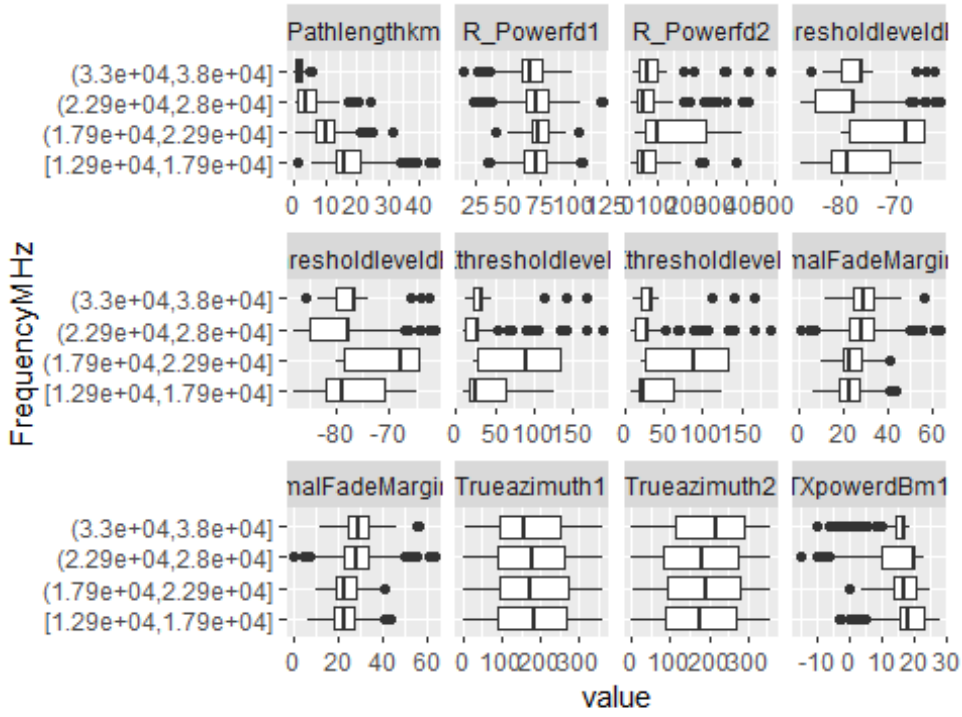



tiveFadeMargi
tiveFadeMargi
ElRPdBm1
ElRPdBm2
Elevation2
Elevationm1
ERPdbm1
ERPdbm2
ERPwatts1
ERPwatts2
occurrencefac
emarginmultip
(3.3e+04,3.8e+04]
(2.29e+04,2.8e+04]
(1.79e+04,2.29e+04]
[1.29e+04,1.79e+04]
FrequencyMHz
value

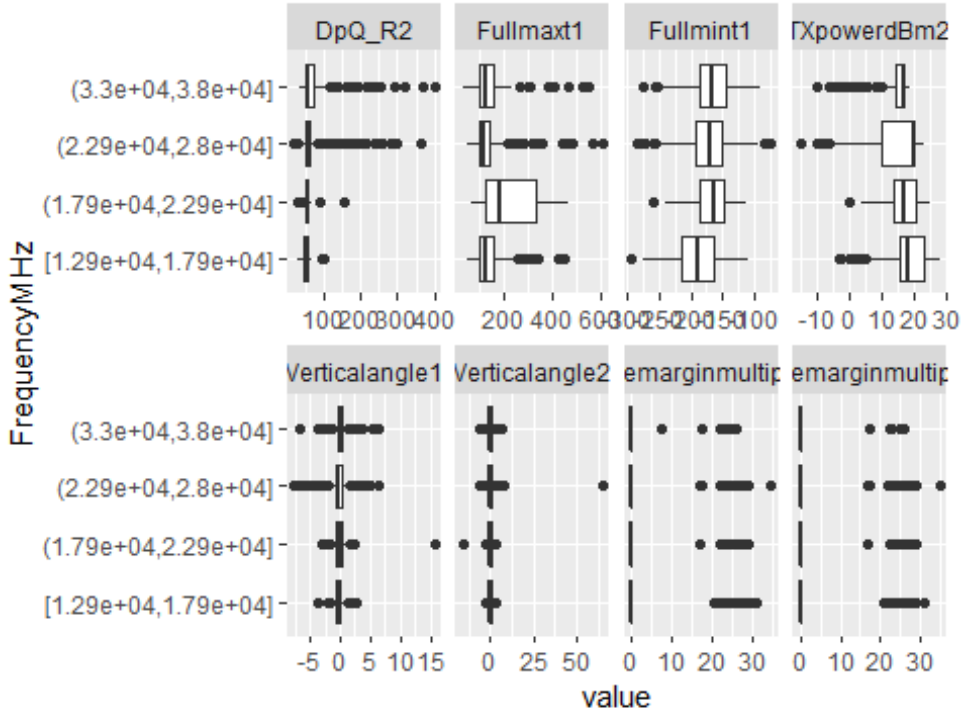
Page 2


emarginmultip
eespacelossc
eoclimaticfacto
nnetpathlossc
nnetpathlossc
receivesignald
receivesignald
therRXlossdB
therRXlossdB
therTXlossdB
therTXlossdB
athinclinationn
(3.3e+04,3.8e+04]
(2.29e+04,2.8e+04]
(1.79e+04,2.29e+04]
[1.29e+04,1.79e+04]
FrequencyMHz
value

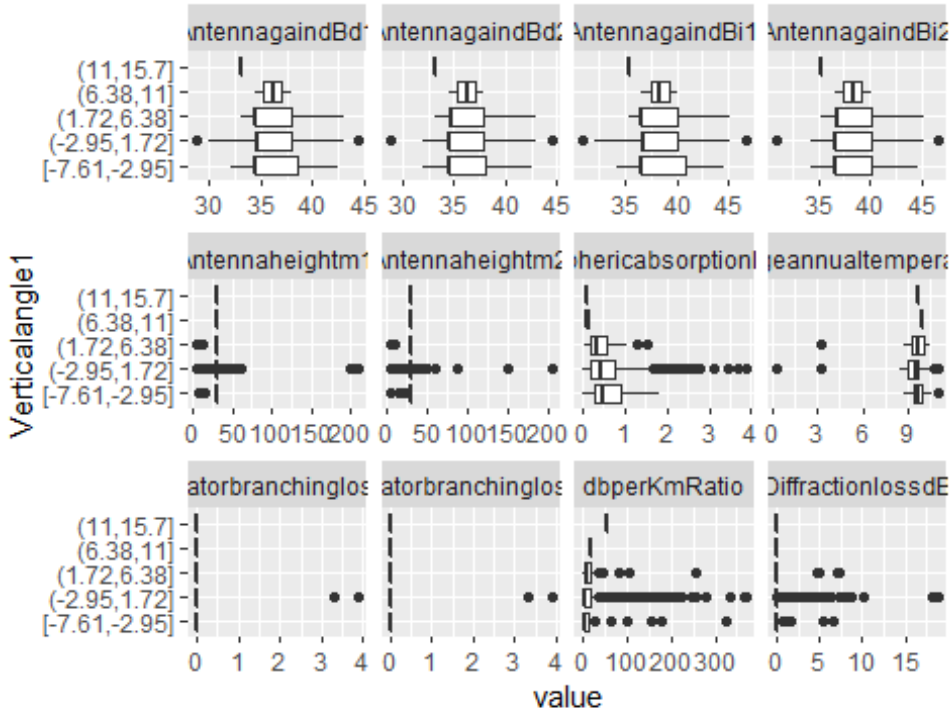
Page 3



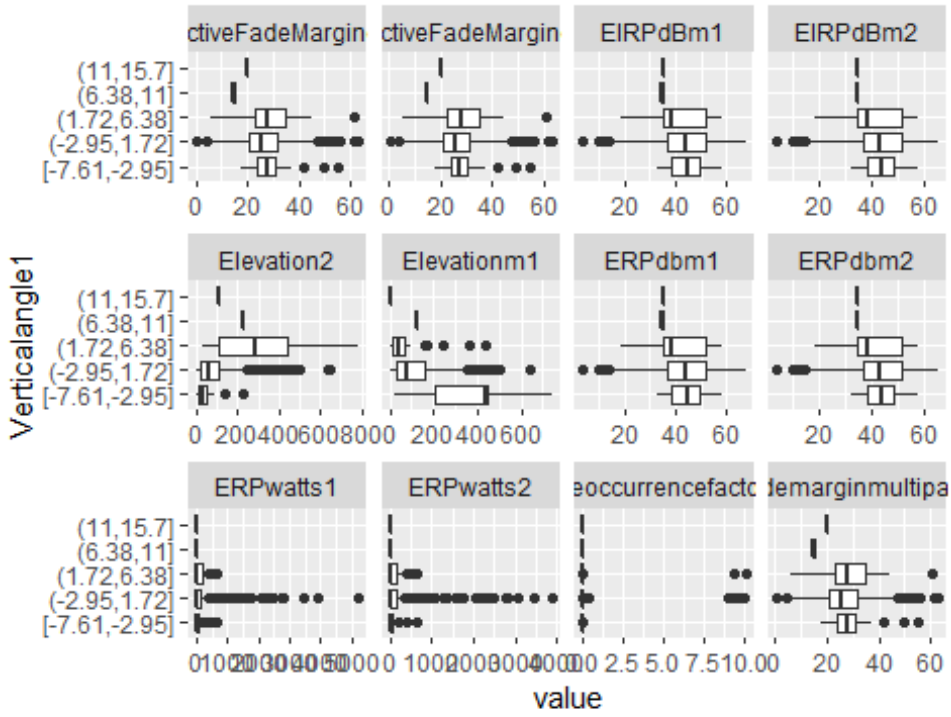

```
## Warning: Removed 15 rows containing non-finite values (stat_boxplot).
```



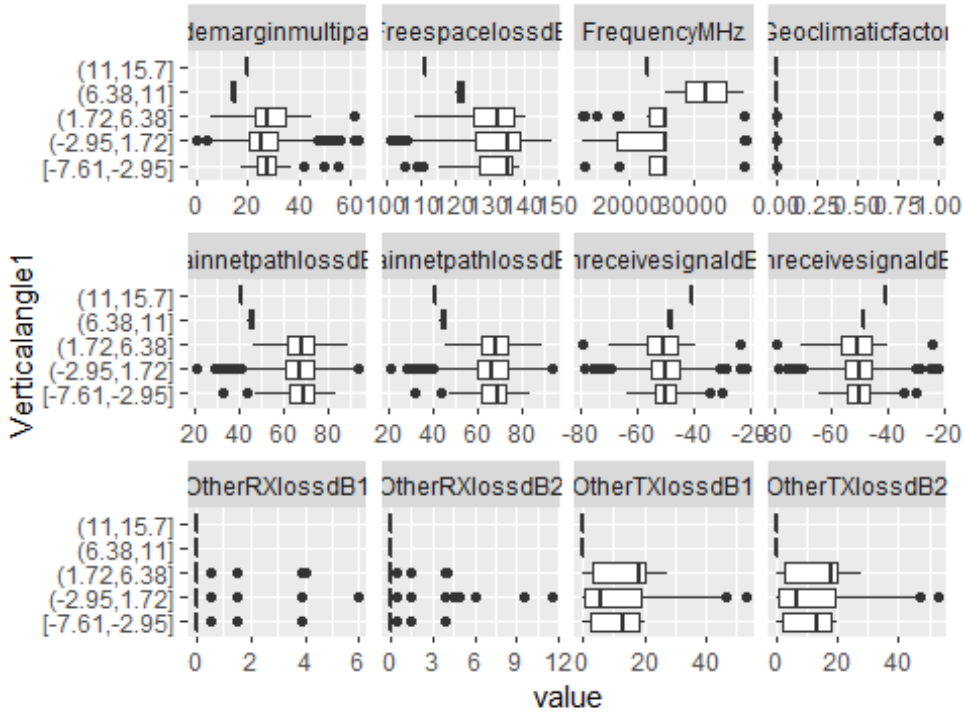

```
plot_boxplot(Train, by = "Verticalangle1")
```


ntennagaindBd1
ntennagaindBd2
AntennagaindBi1
AntennagaindBi2
(11,15.7]
(6.38,11]
(1.72,6.38]
(-2.95,1.72]
[-7.61,-2.95]
30 35 40 45
ntennaheightm1
ntennaheightm2
hericabsorptionl
eannualtempera
Verticalangle1
0 50 100 150 200
0 1 2 3 4
0 3 6 9
atorbranchinglos
atorbranchinglos
dbperKmRatio
DiffractionlossdE
0 1 2 3 4
0 100 200 300
0 5 10 15
value

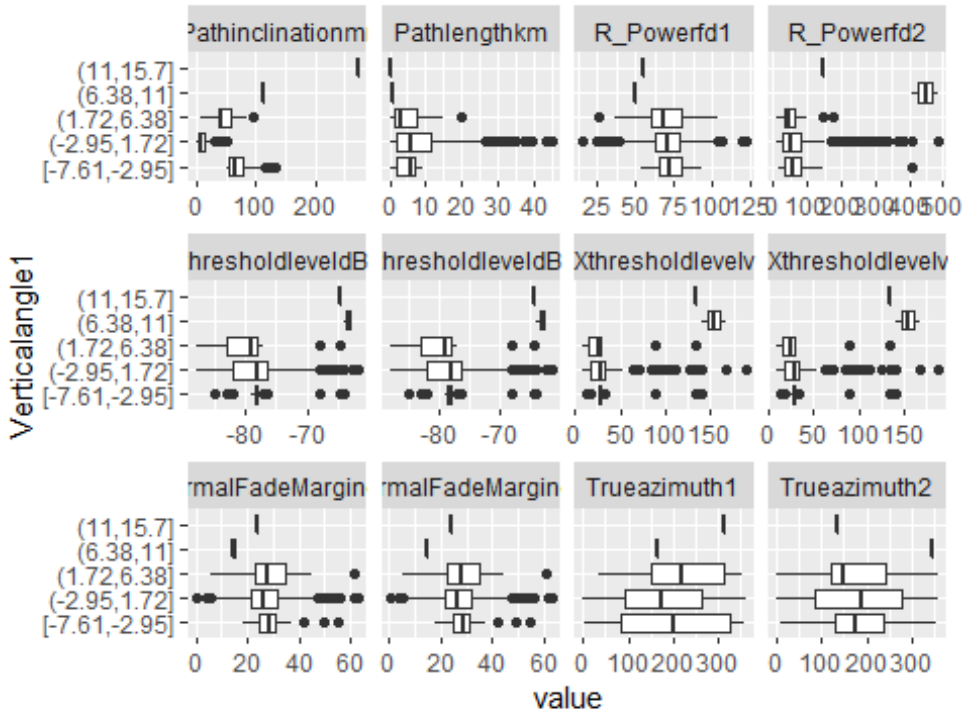
Page 1


ctiveFadeMargin
ctiveFadeMargin
EIRPdBm1
EIRPdBm2
(11,15.7]
(6.38,11]
(1.72,6.38]
(-2.95,1.72]
[-7.61,-2.95]
0 20 40 60
20 40 60
Elevation2
Elevationm1
ERPdbm1
ERPdbm2
Verticalangle1
0 2000 4000 6000 8000
200 400 600
ERPwatts1
ERPwatts2
eoccurrencefactc
demarginmultipa
0 2.5 5.0 7.5 10.00
value

Page 2

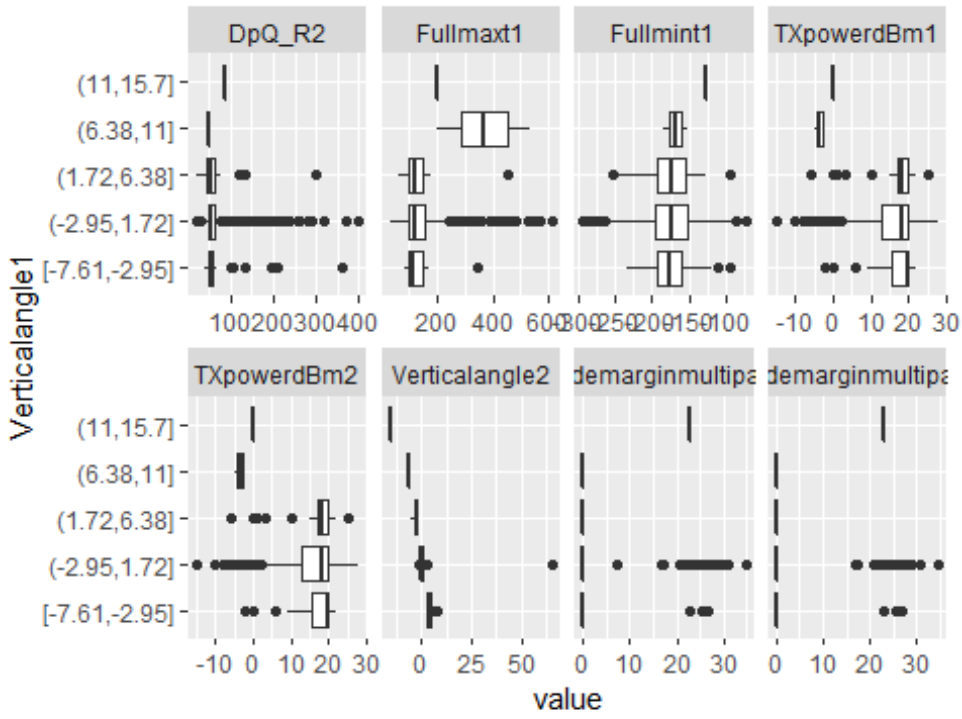


Page 3

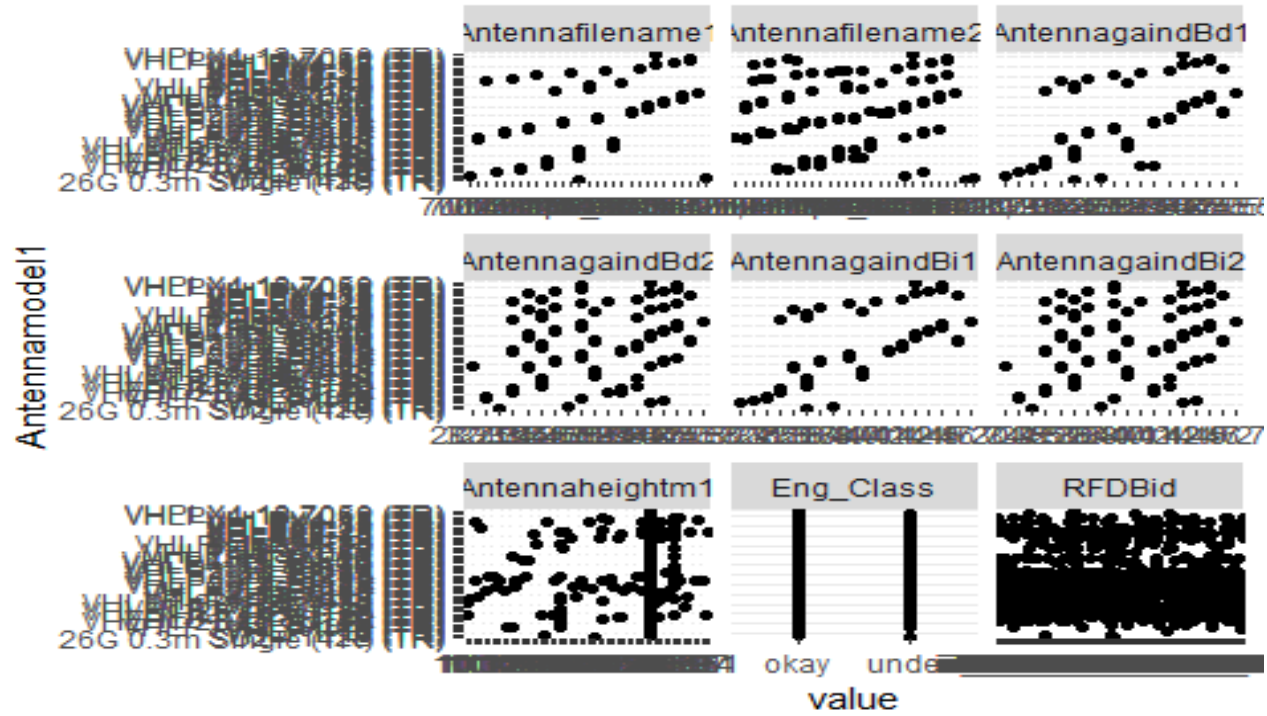


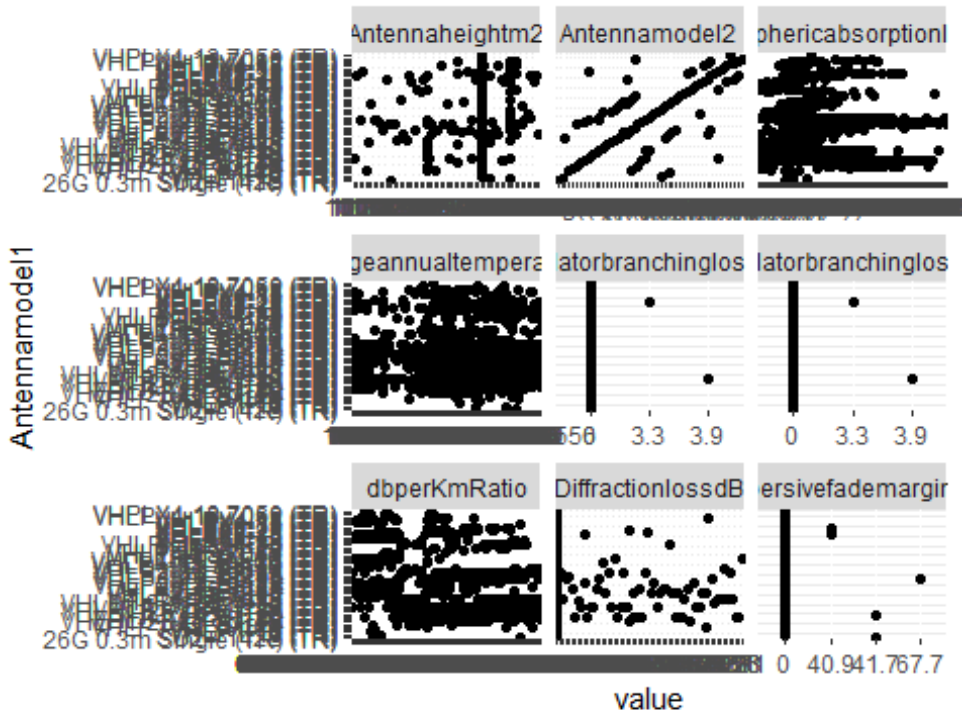
Page 4

```
## Warning: Removed 15 rows containing non-finite values (stat_boxplot).
```

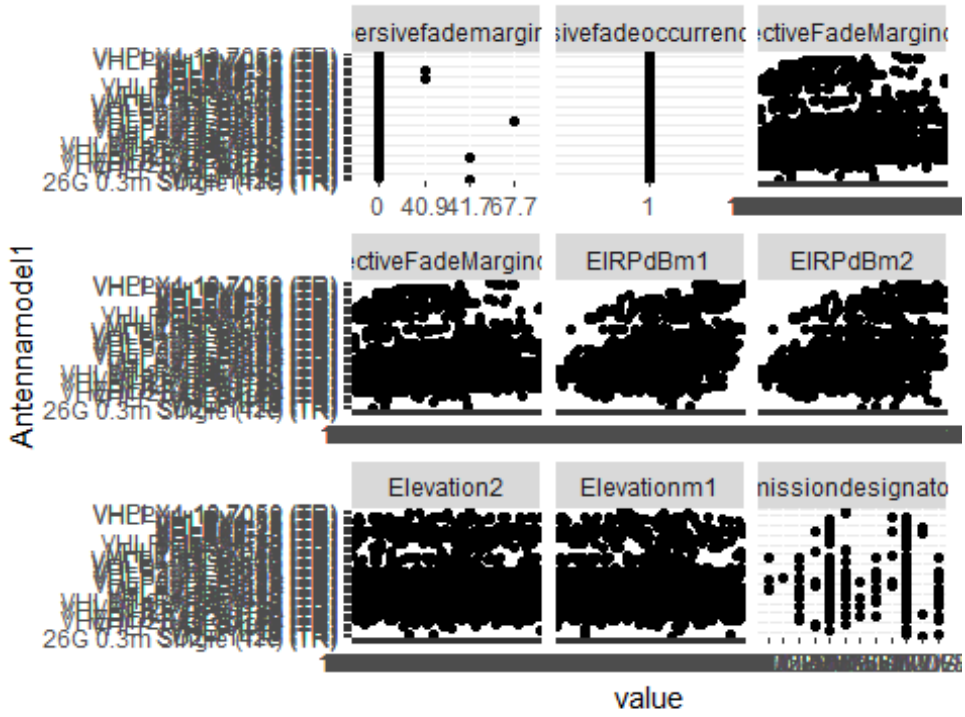



```
#Scatterplot
plot_scatterplot(Train,by = "Antennamodel1")
```

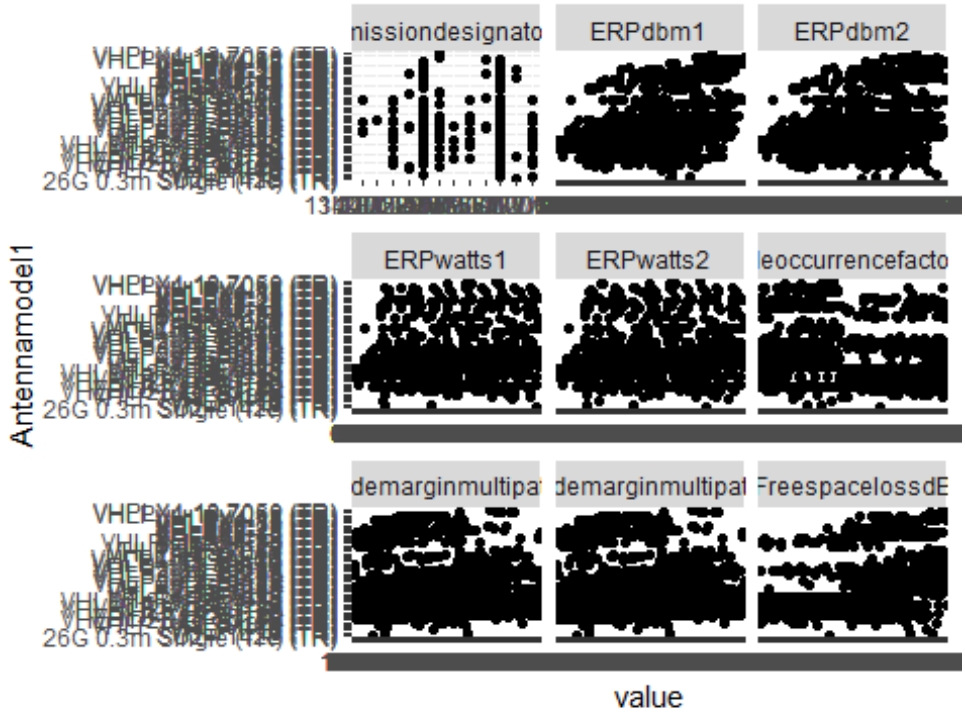



Antennaheightm2
Antennamodel2
phericabsorptionl
geannualtempera
atorbranchinglos
latorbranchinglos
Antennamodel1
0
3.3
3.9
dbperKmRatio
DiffractionlossdB
ersivefademargir
0
40.9
41.7
67.7
value

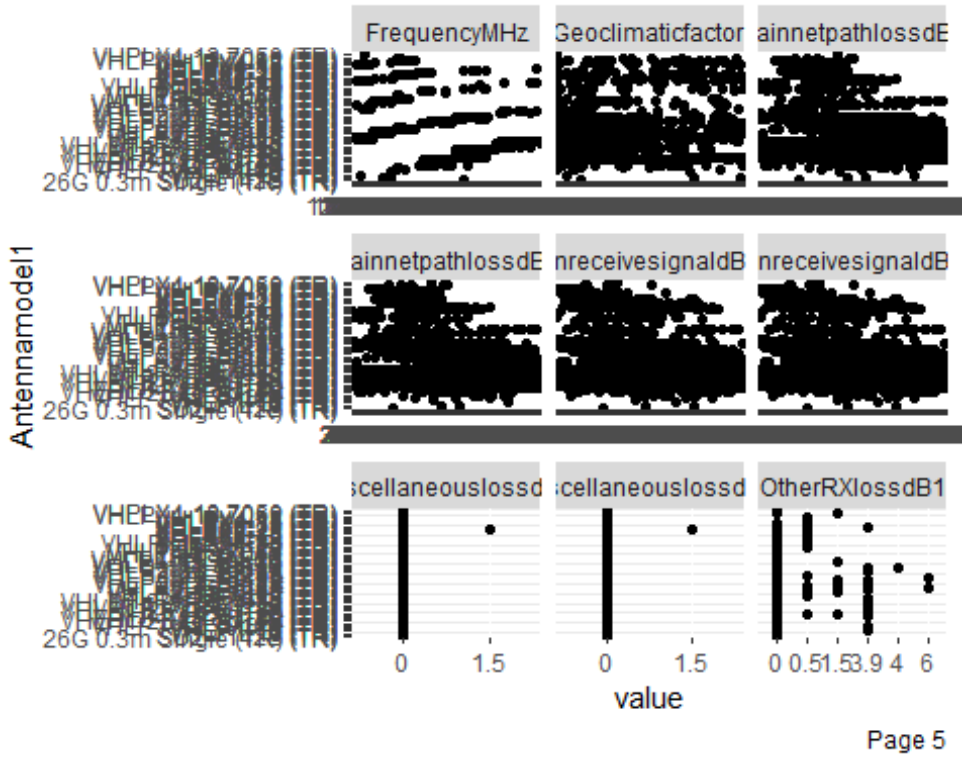
Page 2


ersivefademargir
sivefadeoccurrenc
ctiveFadeMarginc
0
40.9
41.7
67.7
1
ctiveFadeMarginc
EIRPdBm1
EIRPdBm2
Antennamodel1
Elevation2
Elevationm1
nissiondesignato
value

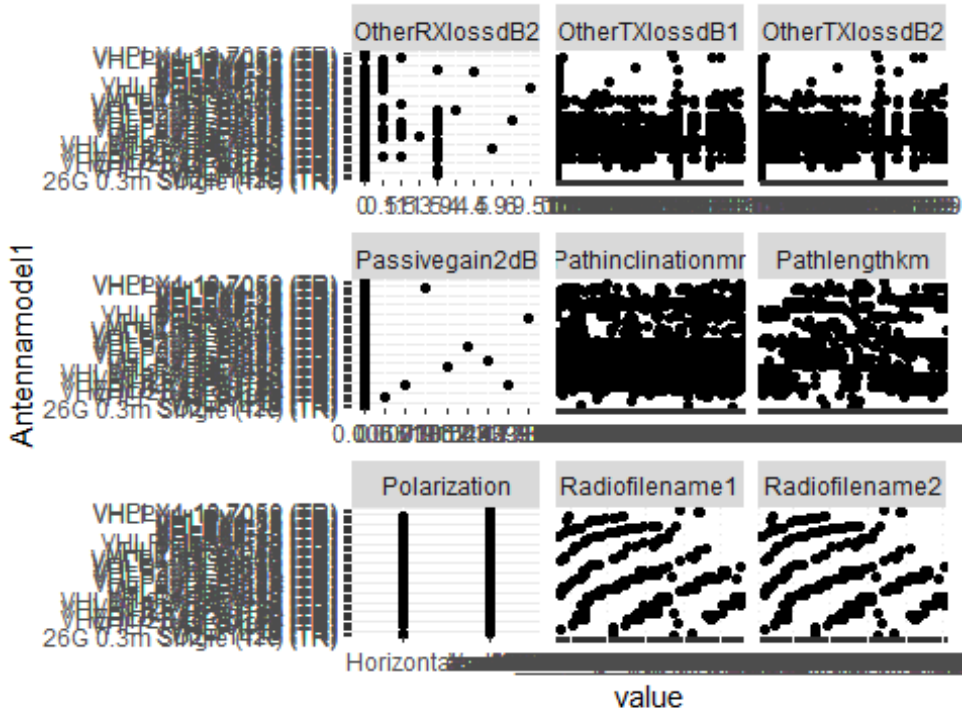
Page 3


nissiondesignato
ERPdbm1
ERPdbm2
ERPwatts1
ERPwatts2
eoccurrencefacto
demarginmultipa
demarginmultipat
FreespacelossdE
Antennamodel1
value

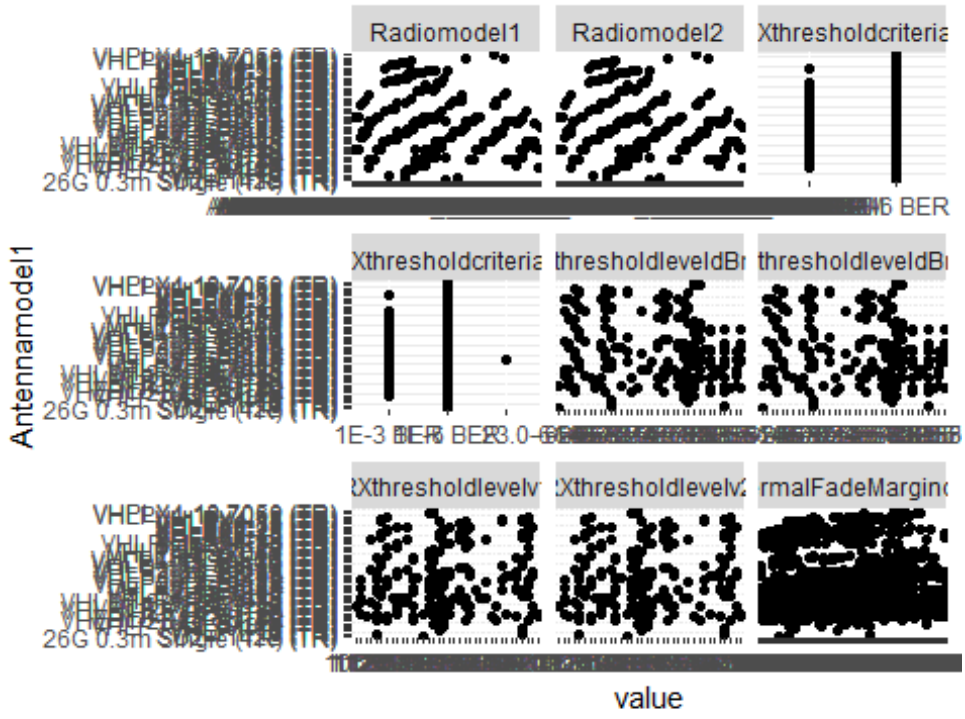
Page 4


FrequencyMHz
Geoclimaticfactor
ainnetpathlossdE
ainnetpathlossdE
nreceivesignaldB
nreceivesignaldB
cellaneouslossd
cellaneouslossd
OtherRXlossdB1
0
1.5
0
1.5
0 0.5 1.5 3.9 4 6
Antennamodel1
value

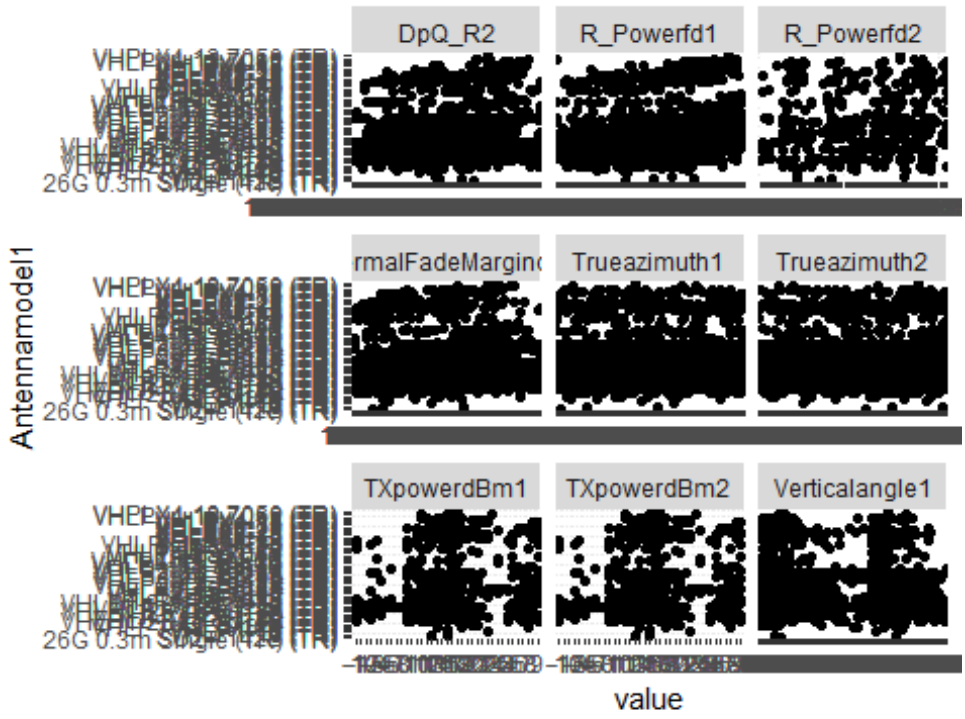
Page 5


OtherRXlossdB2
OtherTXlossdB1
OtherTXlossdB2
Passivegain2dB
Pathinclinationmr
Pathlengthkm
Polarization
Radiofilename1
Radiofilename2
Antennamodel1
value

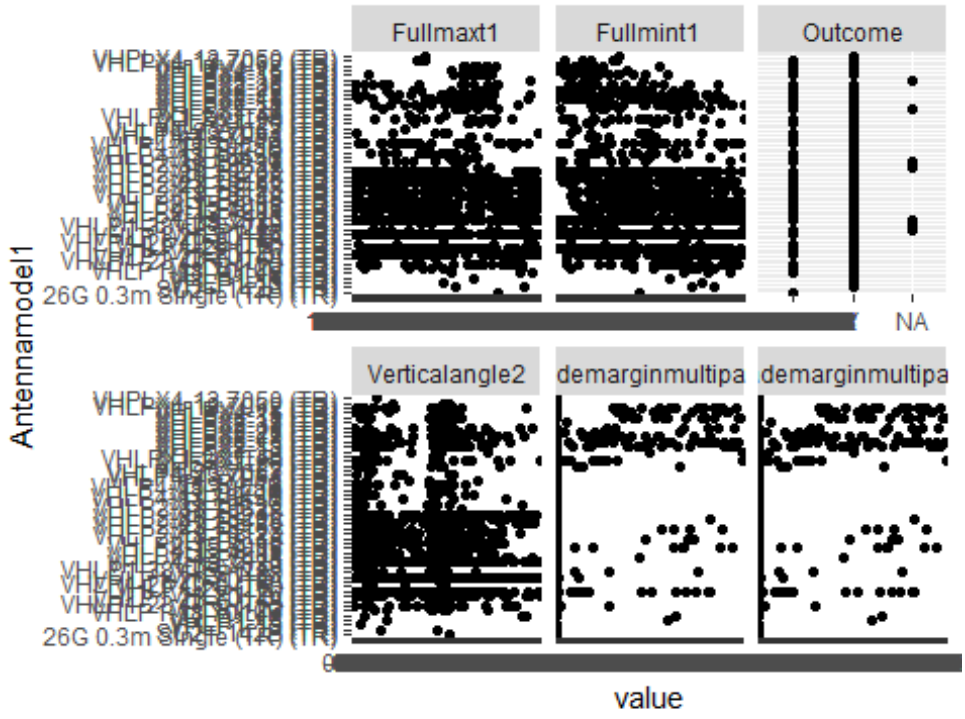
Page 6


Radiomodel1
Radiomodel2
Xthresholdcriteria
Antennamodel1
value

Page 7

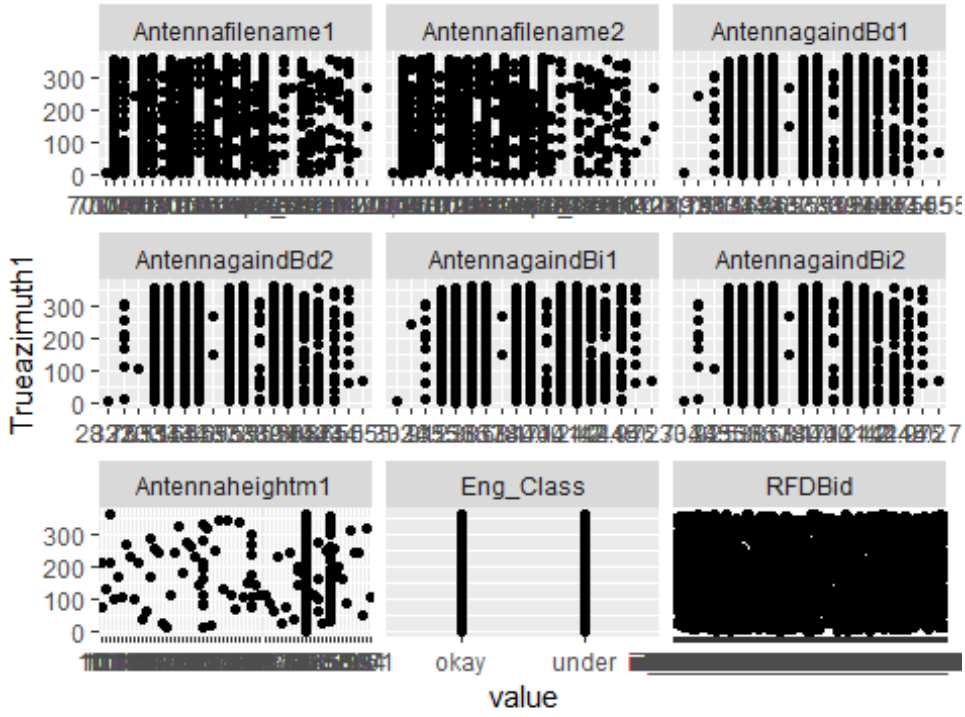


Page 8

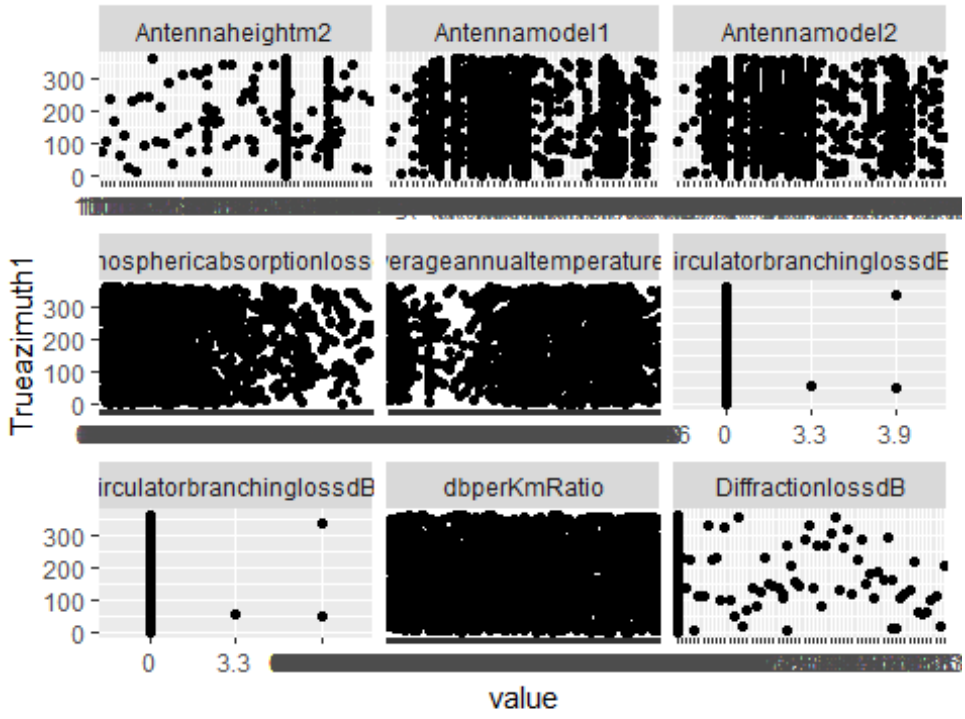


Page 9

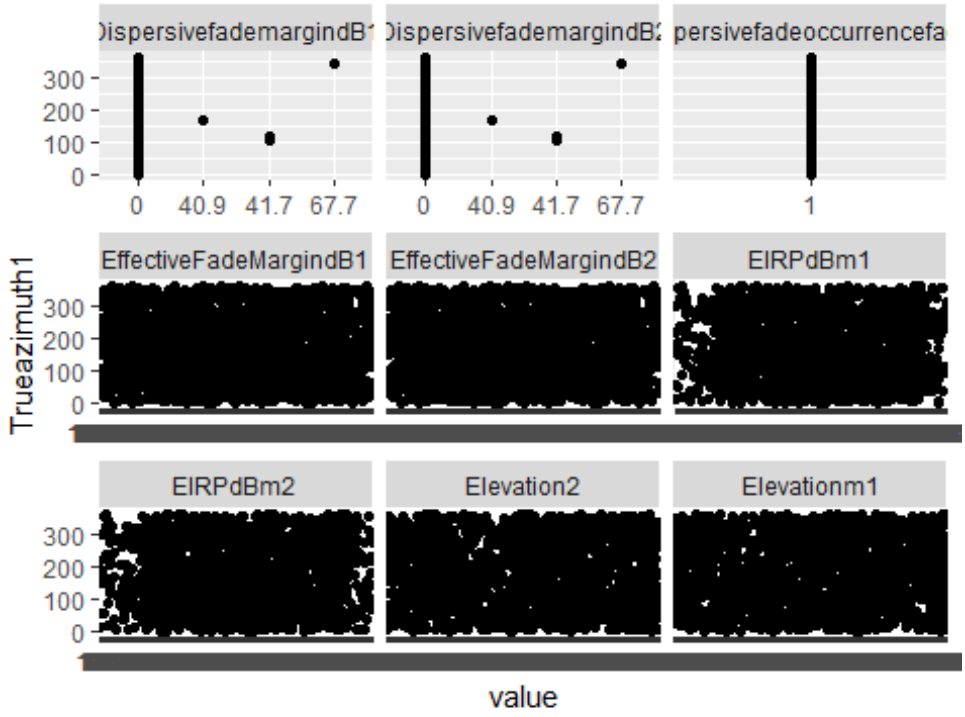
```
plot_scatterplot(Train,by = "Trueazimuth1")
```

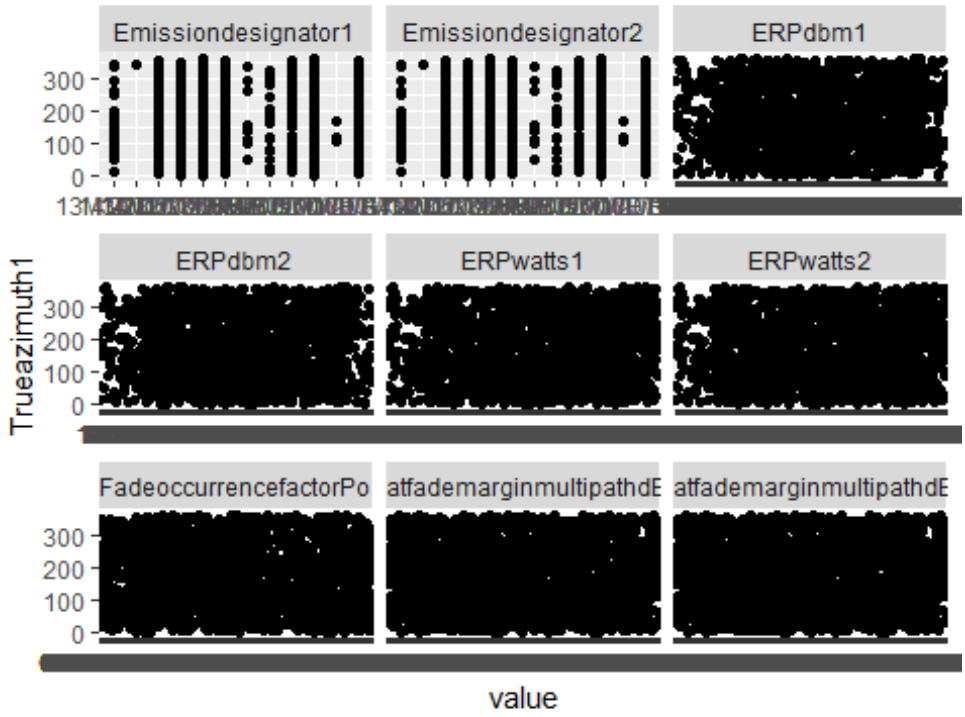

Antennafilename1
Antennafilename2
AntennagaindBd1
AntennagaindBd2
AntennagaindBi1
AntennagaindBi2
Antennaheightm1
Eng_Class
RFDBid
Trueazimuth1
300
200
100
0
okay
under
value

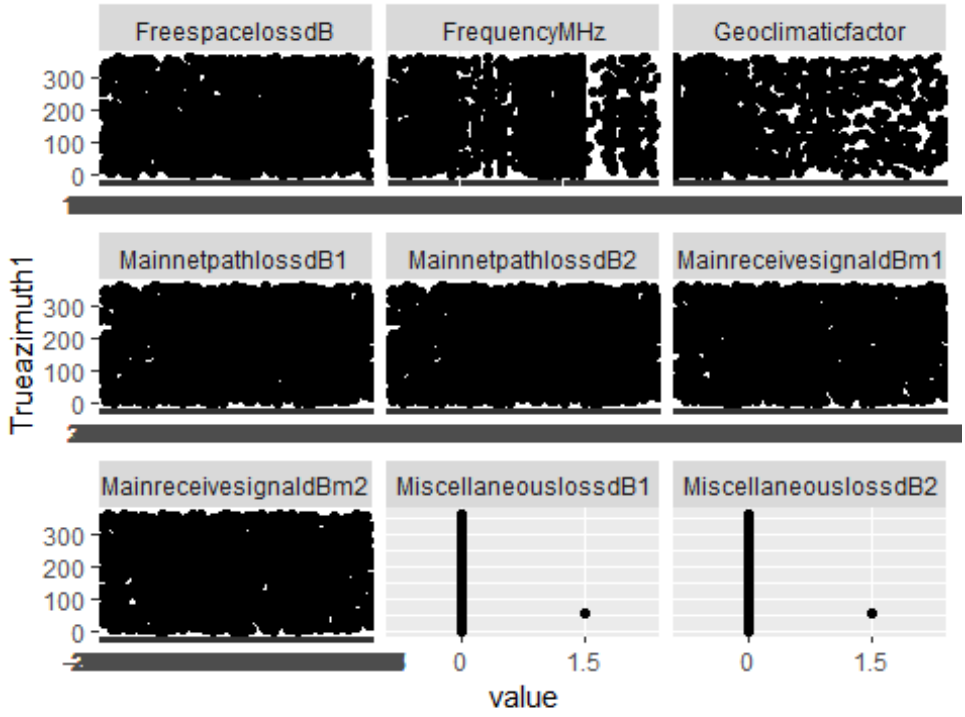
Page 1

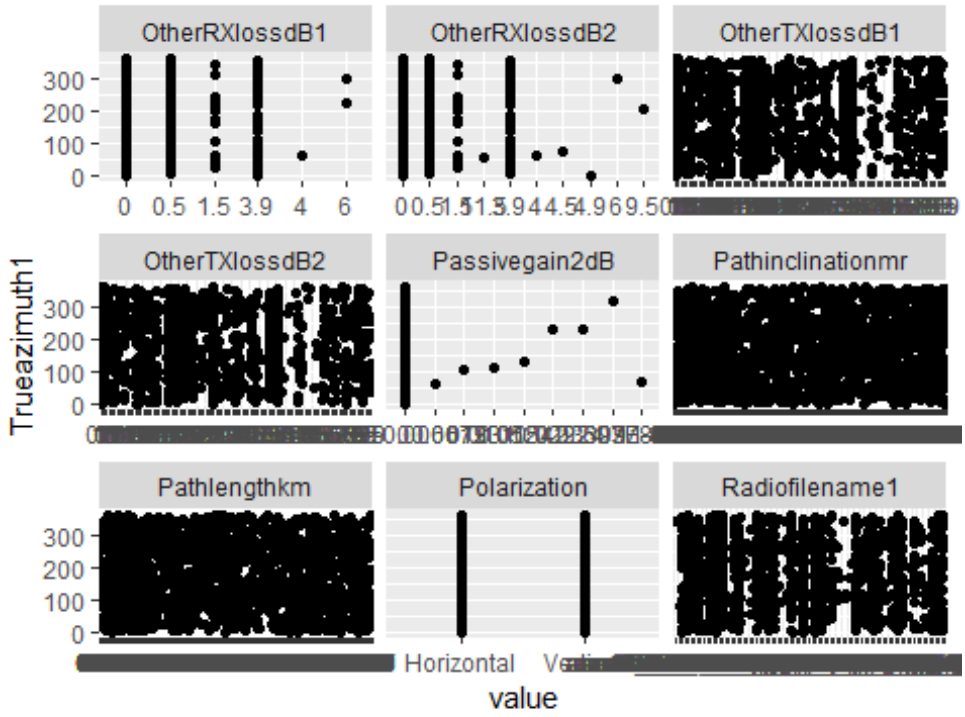

Antennaheightm2
Antennamodel1
Antennamodel2
hosphericabsorptionloss
erageannualtemperature
irculatorbranchinglossdB
irculatorbranchinglossdB
dbperKmRatio
DiffractionlossdB
Trueazimuth1
300
200
100
0
0
3.3
3.9
value

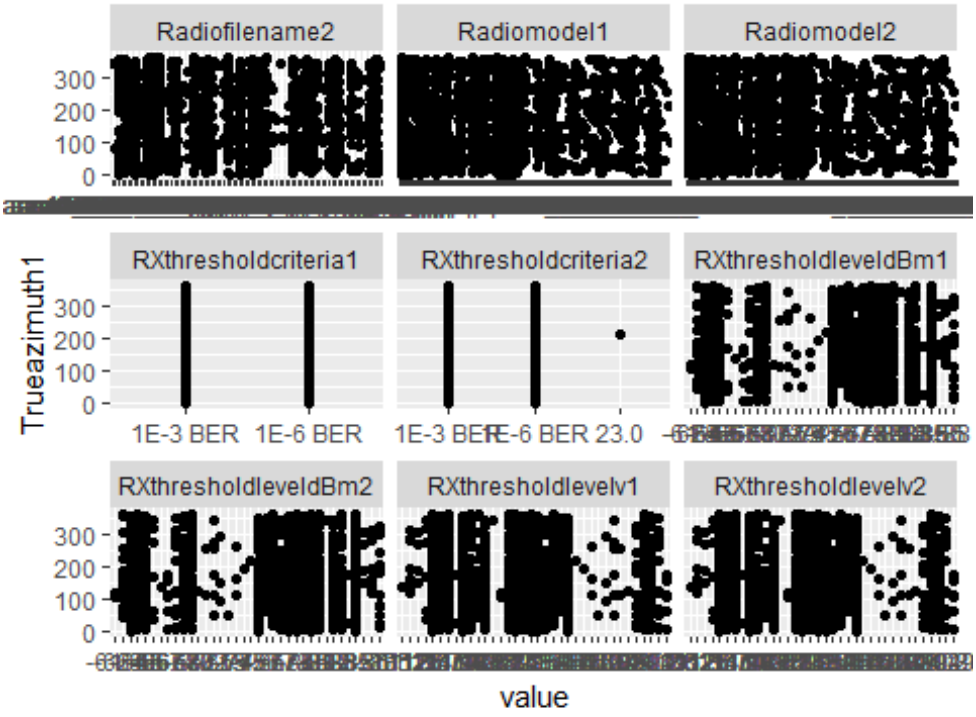
Page 2

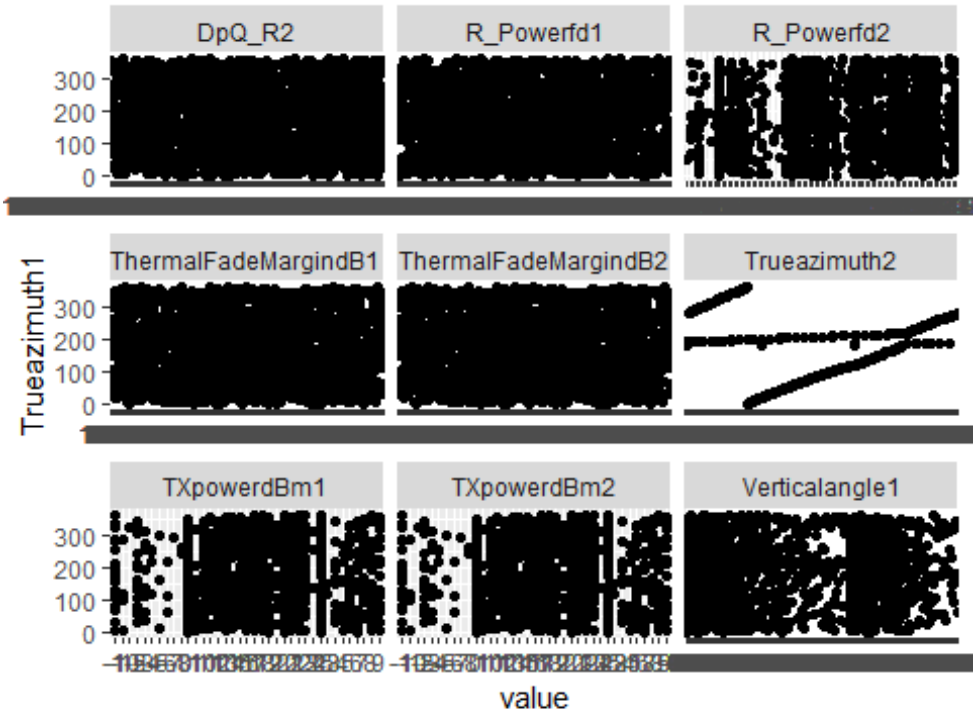

DispersivefademargindB1
DispersivefademargindB2
persivefadeoccurrencefa
300
200
100
0
0
40.9
41.7
67.7
1
EffectiveFadeMargindB1
EffectiveFadeMargindB2
EIRPdBm1
Trueazimuth1
EIRPdBm2
Elevation2
Elevationm1
value

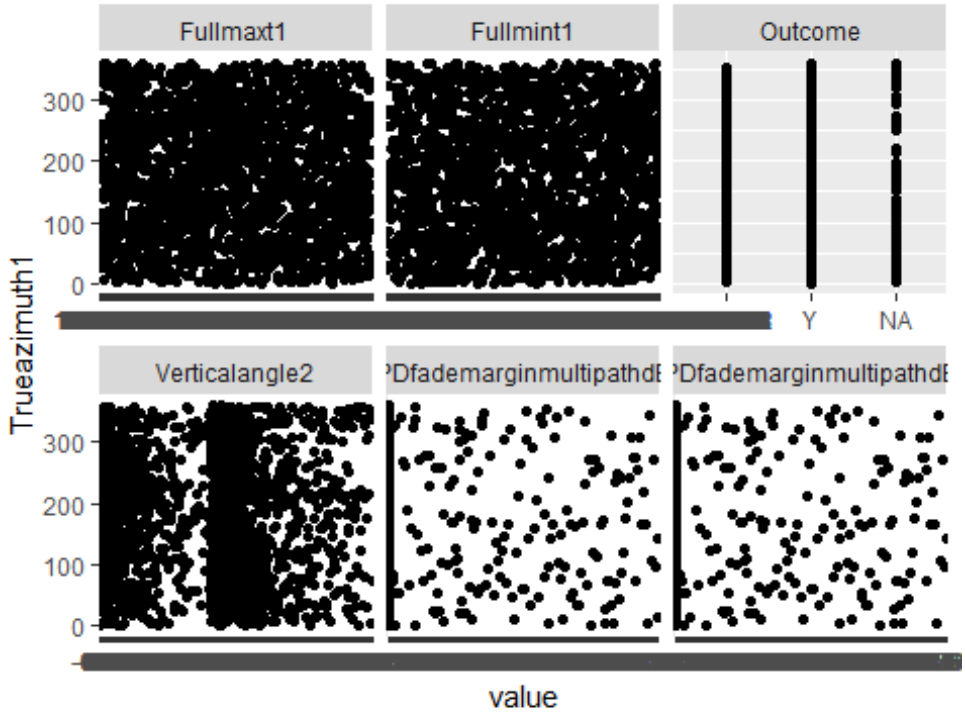

Emissiondesignator1
Emissiondesignator2
ERPdbm1
300
200
100
0
ERPdbm2
ERPwatts1
ERPwatts2
Trueazimuth1
FadeoccurrencefactorPo
atfademarginmultipathdE
atfademarginmultipathdE
value


FreespacelossdB
FrequencyMHz
Geoclimaticfactor
MainnetpathlossdB1
MainnetpathlossdB2
MainreceivesignaldBm1
MainreceivesignaldBm2
MiscellaneouslossdB1
MiscellaneouslossdB2
Trueazimuth1
300
200
100
0
0
1.5
value

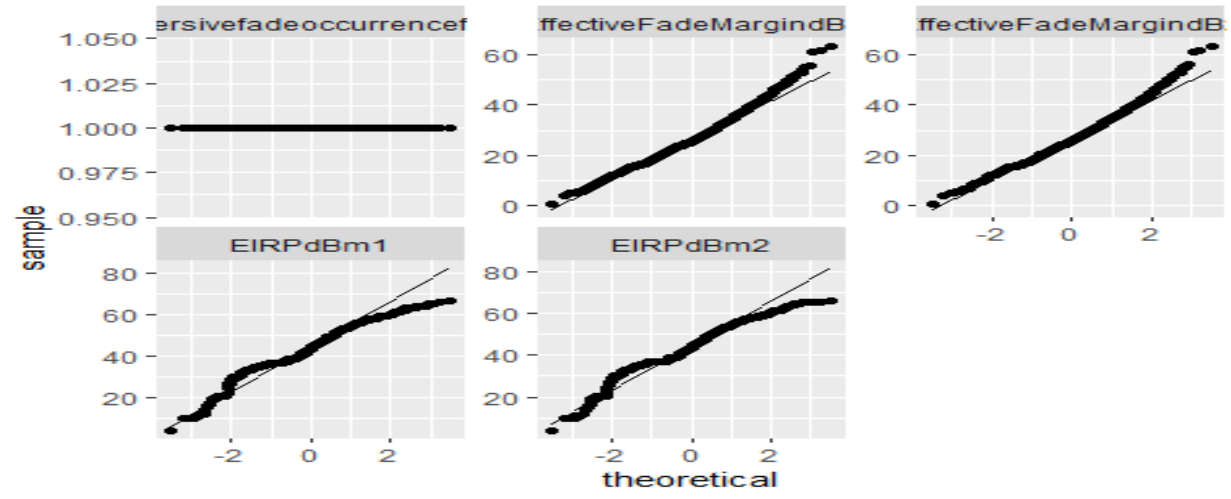

OtherRXlossdB1
OtherRXlossdB2
OtherTXlossdB1
OtherTXlossdB2
Passivegain2dB
Pathinclinationmr
Pathlengthkm
Polarization
Radiofilename1
Trueazimuth1
300
200
100
0
0 0.5 1.5 3.9 4 6
Horizontal
value


Radiofilename2
Radiomodel1
Radiomodel2
RXthresholdcriteria1
RXthresholdcriteria2
RXthresholdleveldBm1
RXthresholdleveldBm2
RXthresholdlevelv1
RXthresholdlevelv2
Trueazimuth1
300
200
100
0
1E-3 BER
1E-6 BER
1E-3 BER
1E-6 BER 23.0
value


DpQ_R2
R_Powerfd1
R_Powerfd2
ThermalFadeMargindB1
ThermalFadeMargindB2
Trueazimuth2
TXpowerdBm1
TXpowerdBm2
Verticalangle1
Trueazimuth1
300
200
100
0
value

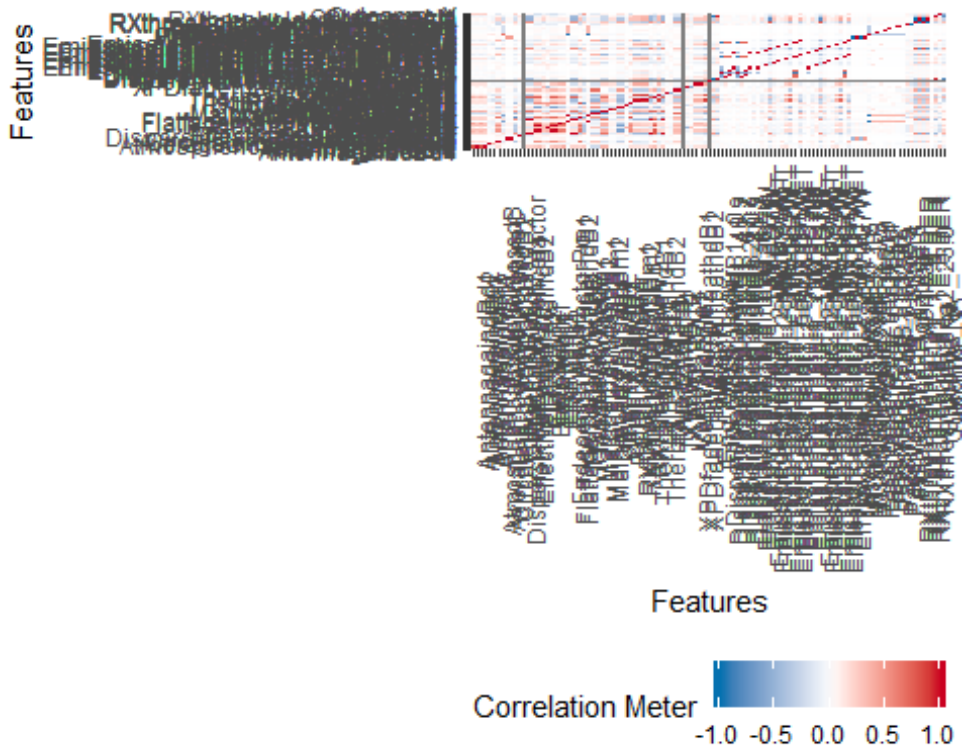



```
#qq plot
qq_data <-Train[, c("Dispersivefadeoccurrencefactor","EffectiveFadeMargindB1"
,"EffectiveFadeMargindB2","EIRPdBm1","EIRPdBm2")]

plot_qq(qq_data)#plotting qq plot for qq_data
```

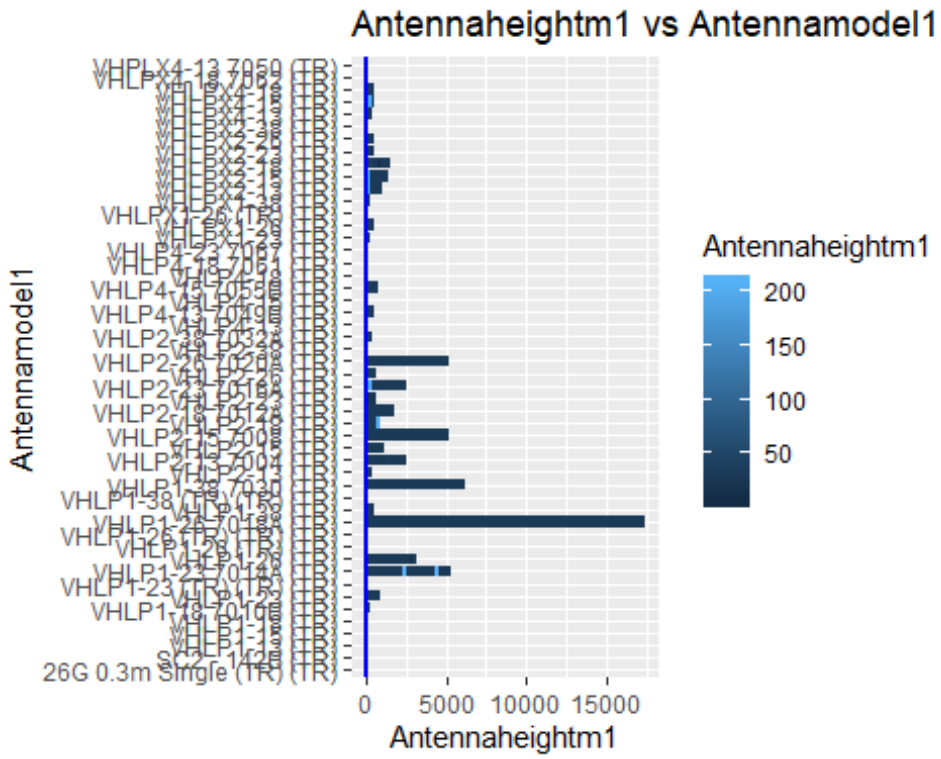



```
#correlation
plot_correlation(Train)
```

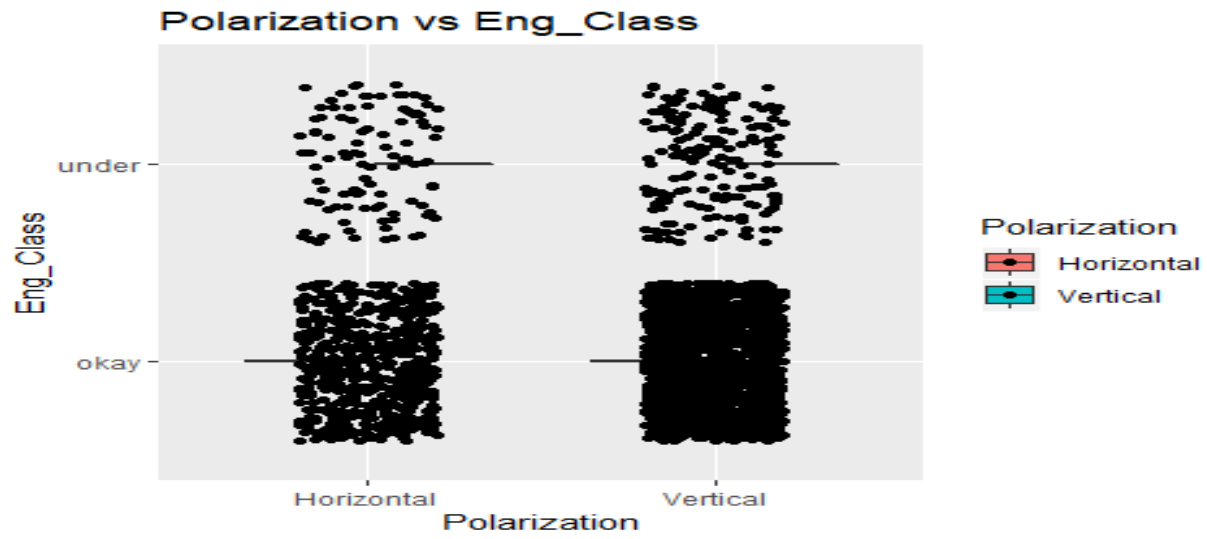
```
## 9 features with more than 20 categories ignored!
## RFDBid: 2186 categories
## Antennafilename1: 29 categories
## Antennafilename2: 29 categories
## Antennamodel1: 50 categories
## Antennamodel2: 49 categories
## Radiofilename1: 88 categories
## Radiofilename2: 88 categories
## Radiomodel1: 152 categories
## Radiomodel2: 152 categories

## Warning in cor(x = structure(list(AntennagaindBd1 = c(39.85, 33.65, 33.65,
: the
## standard deviation is zero
```




```
####********************************
#GGPLOT2
#ggplot for Antennamodel1,Antennaheightm1
ggplot(data=Train,aes(x=Antennamodel1,y=Antennaheightm1)) +
   geom_bar(stat ='identity',aes(fill=Antennaheightm1))+
   coord_flip() +
   theme_grey() +
   scale_fill_gradient(name="Antennaheightm1")+
   labs(title = 'Antennaheightm1 vs Antennamodel1',
        y='Antennaheightm1',x='Antennamodel1')+
   geom_hline(yintercept = mean(Train$Antennaheightm1),size = 1, color = 'blu
e')
```

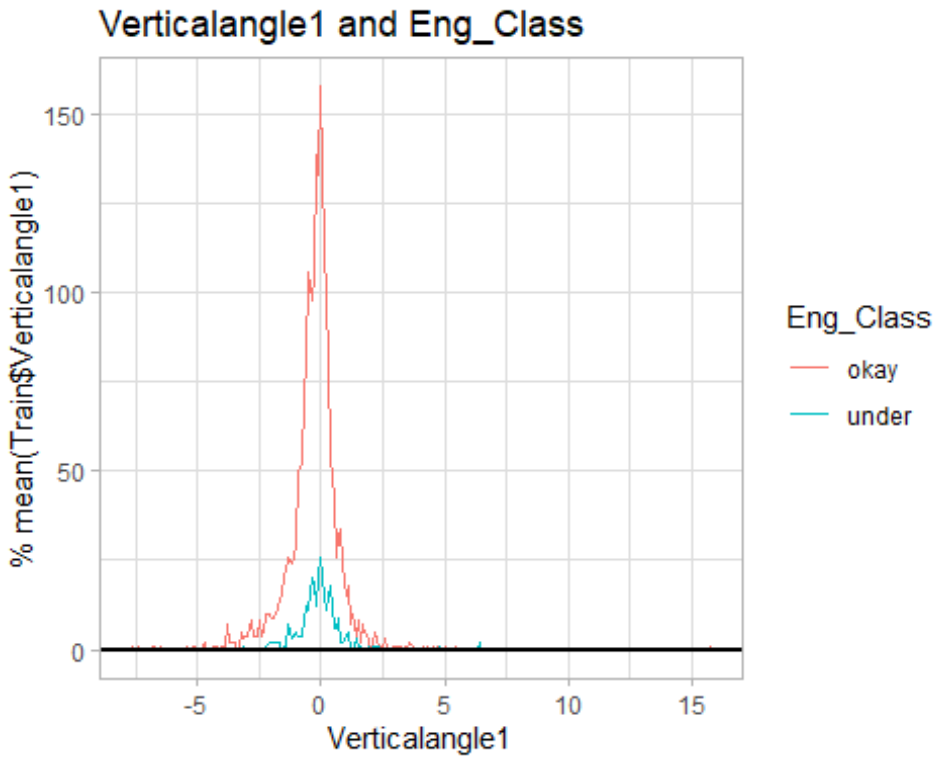



```
#ggplot for Polarization,Eng_Class
ggplot(data = Train, aes(x=Polarization,y=Eng_Class, fill=Polarization)) +
   geom_boxplot()+
   scale_color_brewer(palette="Dark2") +
   geom_jitter(shape=16, position=position_jitter(0.2))+
   labs(title = 'Polarization vs Eng_Class ',
        y='Eng_Class',x='Polarization')
```



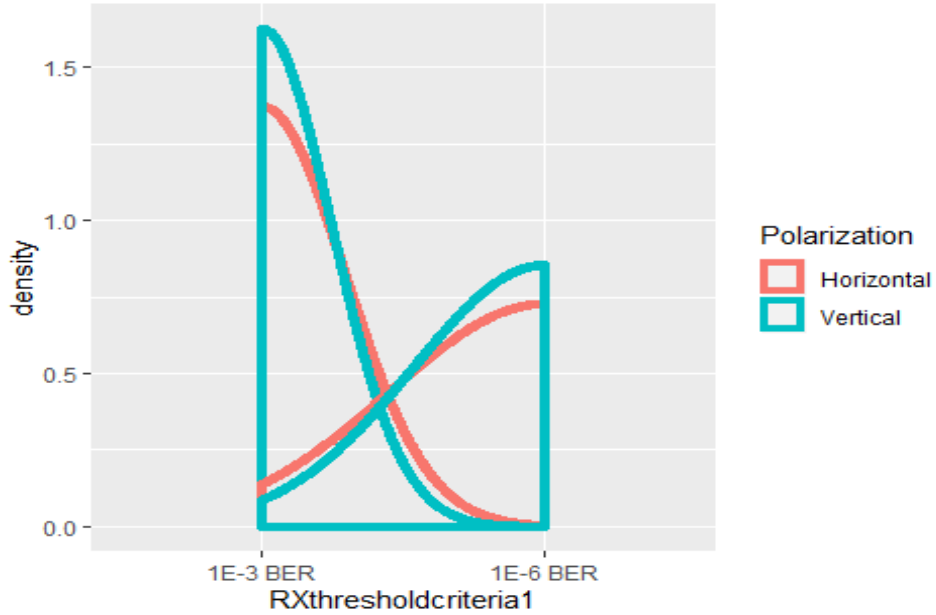

```
#ggplot for verticalangel1 and its mean
ggplot(data=Train,mapping= aes(x=Verticalangle1,colour=Eng_Class)) +
```

```
   geom_freqpoly(binwidth=0.1)+
   theme_light()+ geom_hline(yintercept = mean(Train$Verticalangle1), size=1,
color="black") +
   scale_fill_gradient() +
   labs(title="Verticalangle1 and Eng_Class", x="Verticalangle1", y="% mean(T
rain$Verticalangle1)")
```

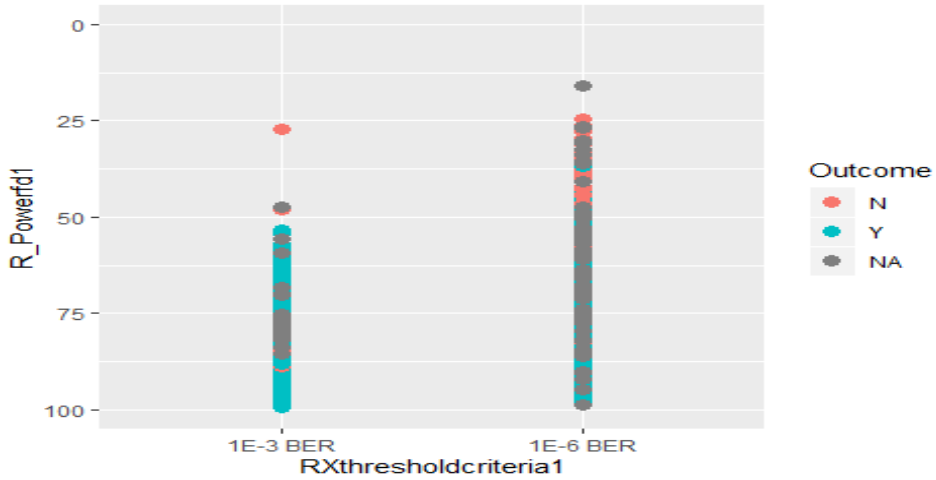



```
#ggplot for RXthresholdcriteria1 and Eng_Class
ggplot(Train, aes(x =  RXthresholdcriteria1, fill = Eng_Class)) +
   geom_bar()
```

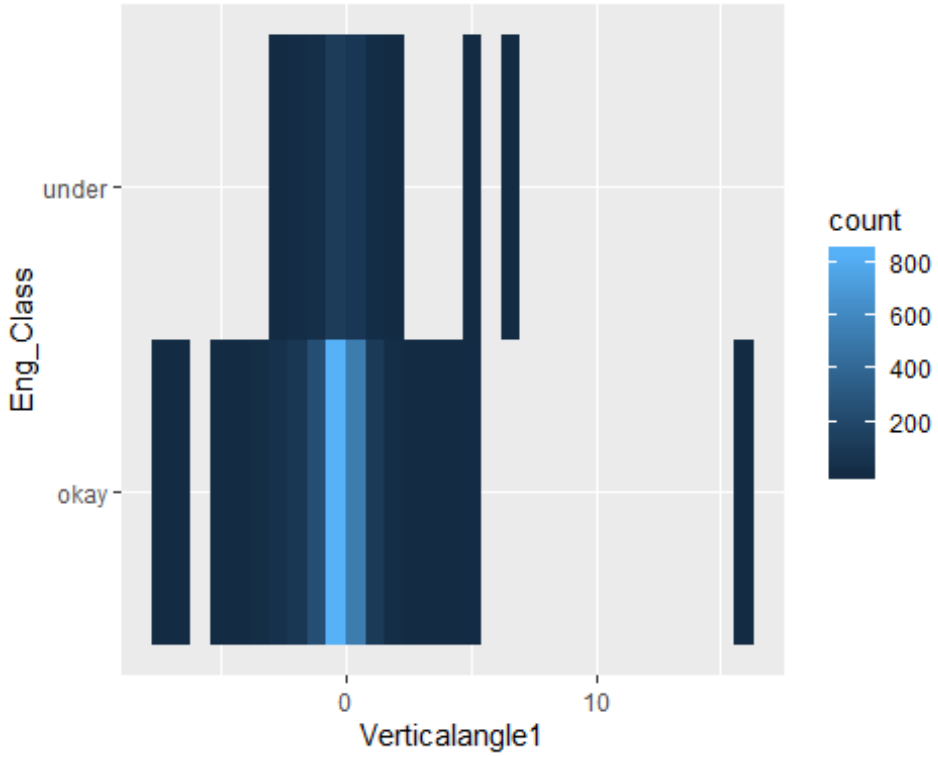
```
#ggplot for RXthresholdcriteria1 and Polarization
ggplot(Train, aes(x = RXthresholdcriteria1, color =Polarization)) +
   geom_density(size = 2)
```



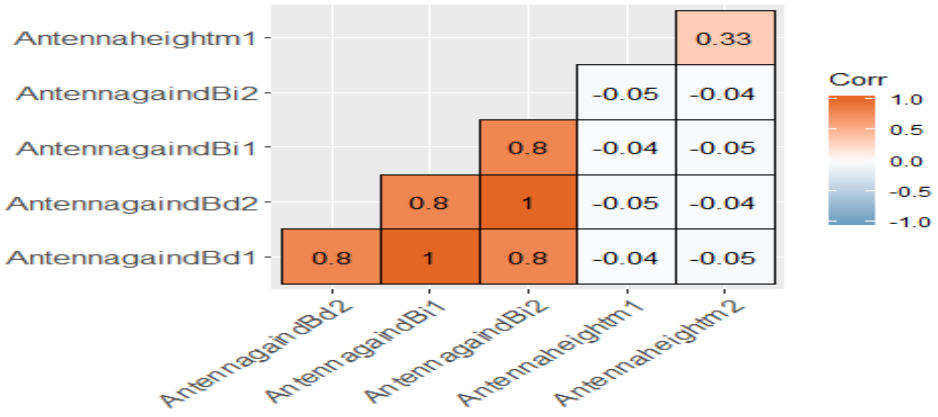

```
#ggplot for RXthresholdcriteria1 and R_Powerfd1
ggplot(Train, aes(x = RXthresholdcriteria1, y =R_Powerfd1 , color = Outcome))
+
   geom_point(size = 3) +
   ylim(100, 0)

## Warning: Removed 13 rows containing missing values (geom_point).
```




```
#ggplot for Eng_Class, Verticalangle1
ggplot(data = Train) +
   geom_bin2d(mapping = aes(y = Eng_Class, x = Verticalangle1))
```

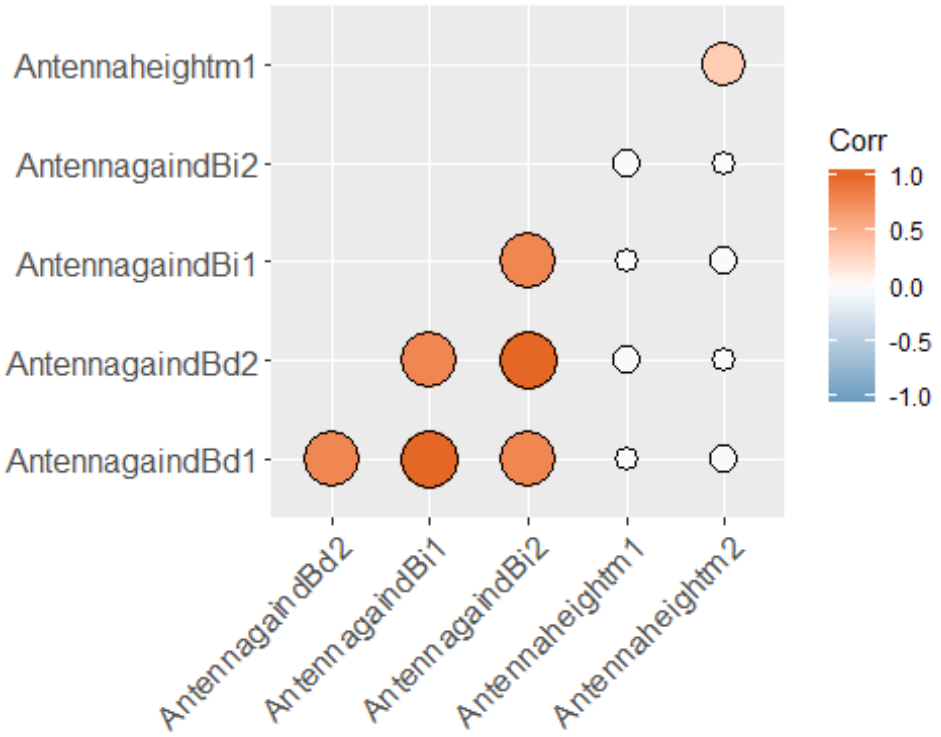



```
#selecting columns randomly
df = Train[,c(5,6,7,8,9,10)]
#correlation of df
res = cor(df)
#ggcorrplot for the df
ggcorrplot(res, type = "lower", outline.col = "black",
           lab=TRUE,
           ggtheme = ggplot2::theme_gray,
           colors = c("#6D9EC1", "white", "#E46726"))
```

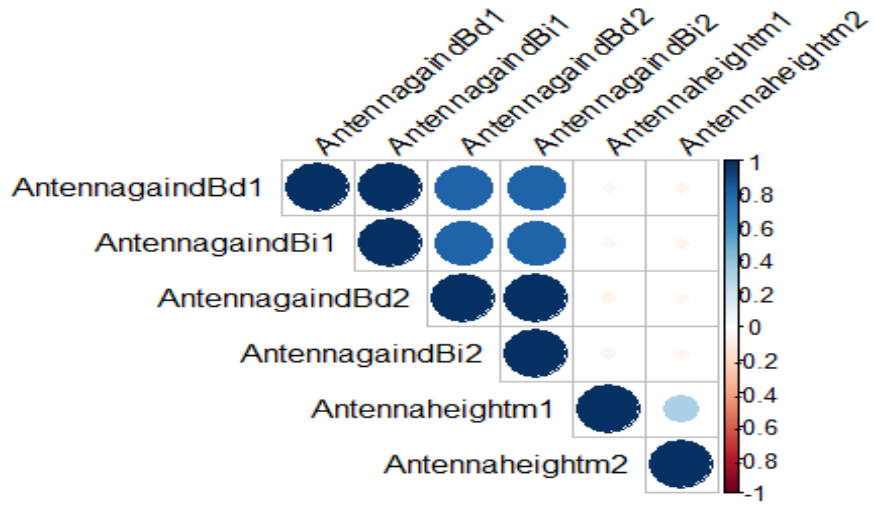



```
ggcorrplot(res, type = "lower", outline.col = "black",
           method="circle",
```

```
          ggtheme = ggplot2::theme_gray,
          colors = c("#6D9EC1", "white", "#E46726"))
```

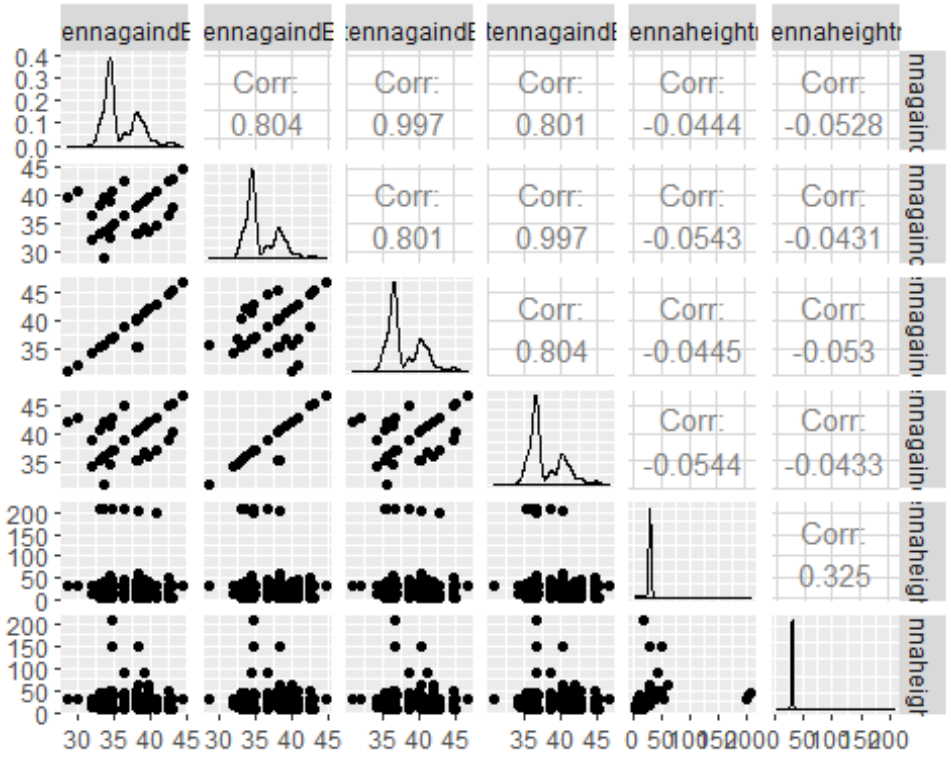



```
#corrplot
corrplot(res, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```




```
#*******************************************
#ggpairs
```

```
#The  ggpairs() function produces a matrix of scatter plots for visualizing t
he correlation between variables

ggpairs(df)#Make a matrix of plots with Train data set
```




```
#*****************************************************************************
******
#DATA CLEANING
# converting the categorical data to factors
Train<-mutate_if(Train,is.character,as.factor)

sum(is.na(Train))

## [1] 120

#printing the complete cases in the data
cat('The Complete cases in the dataset are : ', sum(complete.cases(Train)))

## The Complete cases in the dataset are :  2068

#printing the total number of null values present in the dataset
cat('Total null values in the Train dataset are : ' , sum(is.na(Train)))

## Total null values in the Train dataset are :  120

#printing the name of the column with null values
list_na <- colnames(Train)[apply(Train, 2, anyNA)]
cat('Columns with null values : ',list_na)
```

```
## Columns with null values :  DpQ_R2 Fullmint1 Outcome
```

```
#Data Preparation and Preprocessing
set.seed(214)
#impute missing values using preProcess
preProcValues <- preProcess(Train, method = c("knnImpute"))
```

```
## Warning in preProcess.default(Train, method = c("knnImpute")): These varia
bles
## have zero variances: Dispersivefadeoccurrencefactor
```

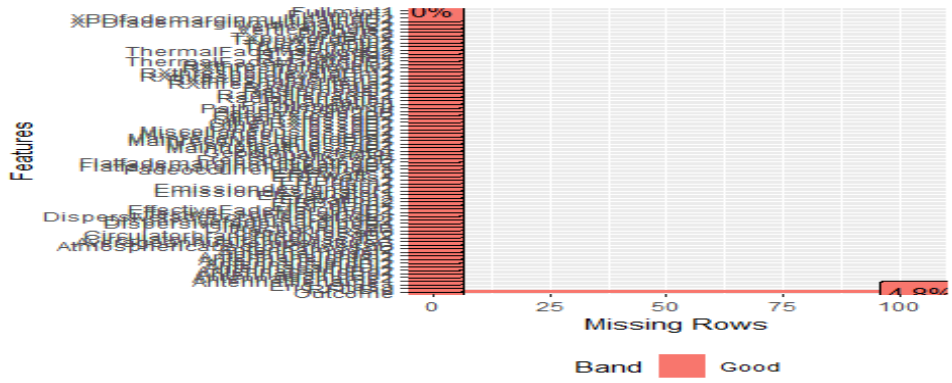
```
preProcValues
```

```
## Created from 2068 samples and 79 variables
##
## Pre-processing:
##   - centered (58)
##   - ignored (21)
##   - 5 nearest neighbor imputation (58)
##   - scaled (58)
```

```
train_processed <- predict(preProcValues, Train)
#checking for missing values after imputation
sum(is.na(train_processed))
```

```
## [1] 105
```

```
plot_missing(train_processed)
```




```
#filling NA for categorical data by using mode
table(is.na(train_processed$Outcome))#checking how many misssig values are th
ere in the column
```

```
##
## FALSE  TRUE
##  2081   105
```

```
sort(table(train_processed$Outcome))#sorting the column in order to get the t
otal number of observation for each domain in the column in ascending order
```

```
##
##    N    Y
##  258 1823
```

```
names(table(train_processed$Outcome))[table(train_processed$Outcome)==max(tab
le(train_processed$Outcome))] #now taking the domain which is having the high
value
```

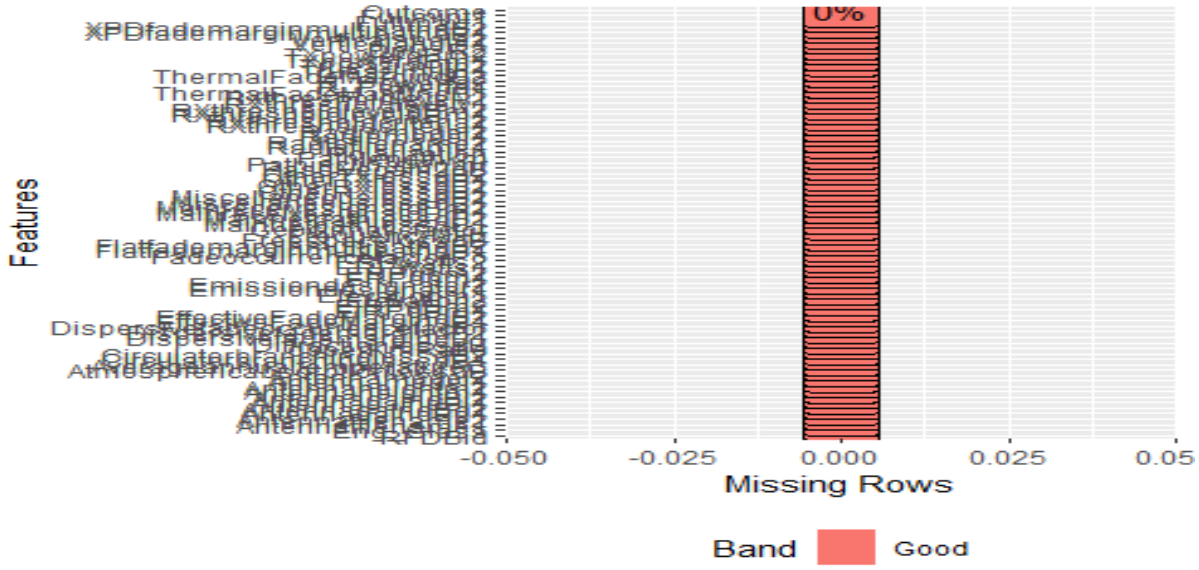
```
## [1] "Y"
```

```
train_processed$Outcome[is.na(train_processed$Outcome)] <- "Y"#Now assigining
the highest value to the missing rows in the column
```

```
table(is.na(train_processed$Outcome))#printing the table
```

```
##
## FALSE
##  2186
```

```
plot_missing(train_processed)#plotting the missing data graph to check whethe
r any missing values is present
```




```
#DATA REDUCTION

#How to create One-Hot Encoding (dummy variables)
#Converting Eng_Class variable to numeric
train_processed$Eng_Class<-ifelse(train_processed$Eng_Class=='under',0,1)
#Removing RFDBid,Antennafilename1,Antennafilename2 columns.
train_processed$RFDBid<-NULL
train_processed$Antennafilename1<-NULL
train_processed$Antennafilename2<-NULL
dim(train_processed)
```

```
## [1] 2186   76
```

```
#Creating Dummy Variables
dmy <- dummyVars(" ~ .", data = train_processed,fullRank = T)
train_transformed <- data.frame(predict(dmy, newdata = train_processed))
dim(train_transformed)
```

```
## [1] 2186  675
```

```
# Remove Zero and Near Zero-Variance Predictors
nzv <- nearZeroVar(train_transformed, saveMetrics = TRUE)

dim(train_transformed)
```

```
## [1] 2186  675
```

```
nzv <- nearZeroVar(train_transformed)
dat2 <- train_transformed[, -nzv]
dim(dat2)
```

```
## [1] 2186   87
```

```
# Identifying numeric variables
numericData <- dat2[sapply(dat2, is.numeric)]
dim(numericData)
```

```
## [1] 2186   87
```

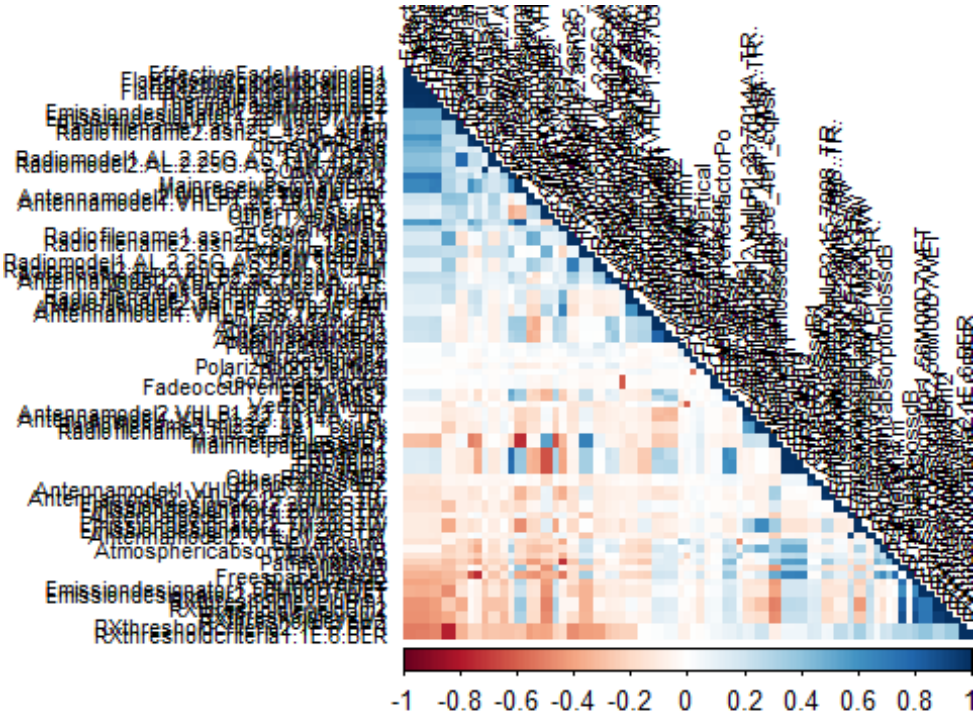
```
# Calculate correlation matrix
descrCor <- cor(numericData)

dim(descrCor)
```

```
## [1] 87 87
```

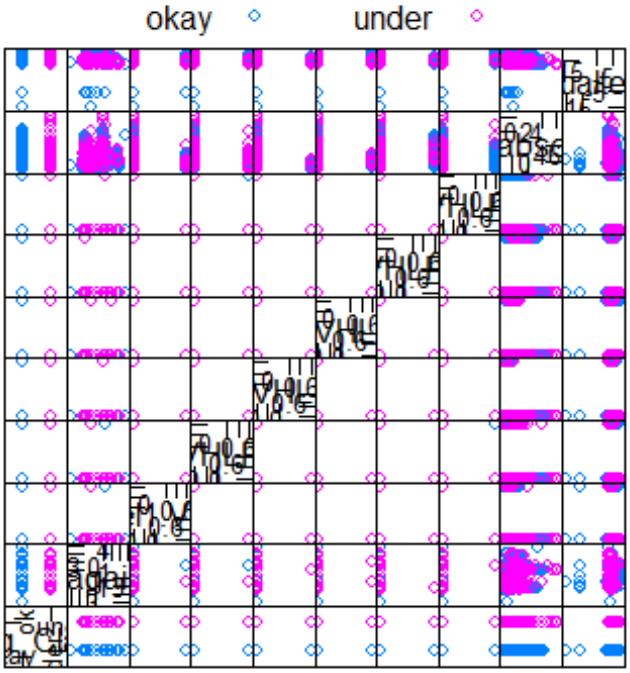
```
summary(descrCor[upper.tri(descrCor)])
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.77723 -0.10478 -0.01590  0.02535  0.13573  1.00000
```

```
# Check Correlation Plot
corrplot(descrCor, order = "FPC", method = "color", type = "lower", tl.cex =
0.7, tl.col = rgb(0, 0, 0))
```

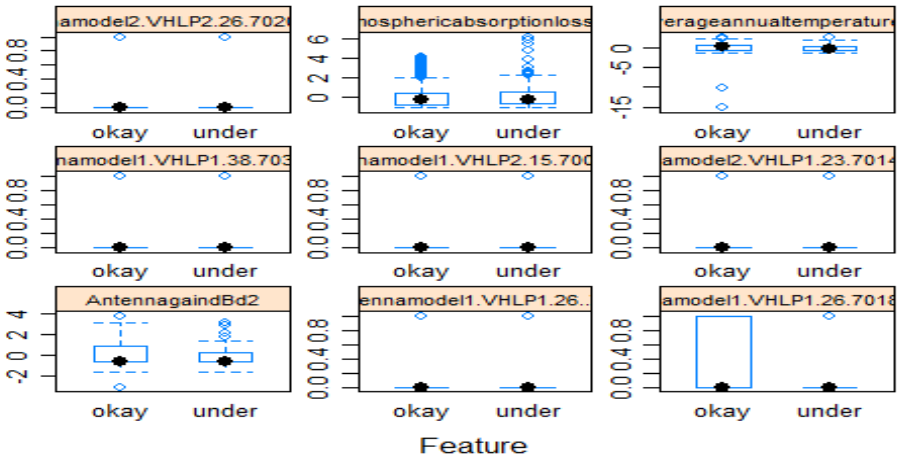



```
# find attributes that are highly corrected
highlyCorrelated <- findCorrelation(descrCor, cutoff=0.70)

# Indentifying Variable Names of Highly Correlated Variables
highlyCorCol <- colnames(numericData)[highlyCorrelated]

# Remove highly correlated variables and create a new dataset
dat3 <- dat2[, -which(colnames(dat2) %in% highlyCorCol)]
dim(dat3)

## [1] 2186   38

#converting the data to factors again
dat3$Eng_Class<-ifelse(dat3$Eng_Class==0,'under','okay')
dat3$Eng_Class<-as.factor(dat3$Eng_Class)

#checking for duplicates in dat3
which(duplicated(rownames(dat3)))

## integer(0)

which(duplicated(colnames(dat3)))

## integer(0)

#feature plots
featurePlot(x = dat3[1:10], y = dat3$Eng_Class, plot = 'pairs', auto.key = li
st(column = 2))
```



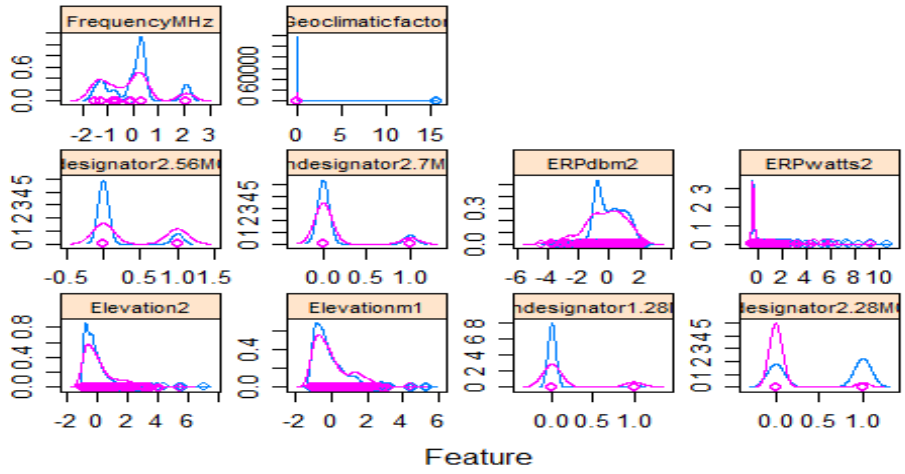
Scatter Plot Matrix

```
featurePlot(x = dat3[, 2:10],y = dat3$Eng_Class, plot = "box",
            strip=strip.custom(par.strip.text=list(cex=.7)),
            scales = list(x = list(relation="free"),
                          y = list(relation="free")))
```



Feature

```
featurePlot(x = dat3[, 11:20],
            y = dat3$Eng_Class,
            plot = "density",
            strip=strip.custom(par.strip.text=list(cex=.7)),
            scales = list(x = list(relation="free"),
                          y = list(relation="free")))
```



#**********

QUESTION B

Set up a training/testing methodology. Using a least 2 models, tune these models and compare your results. Give your best model and comment.

SOLUTION

Now I am going to build models. For building models I used the caret package. Initially, I will be making two samples by dividing the rows into 70% training data and 30% testing data probability. I will be taking the training data and train the model later for prediction the model I will be considering the testing data and predicting the data. Finally I will find the confusion matrix to know the accuracy of the model. We will be building a cross-validation system from the training set to test against our standard. It is important to rely more on the cross-validation set for model 's actual evaluation otherwise we may end up overfitting.To split data into two groups, we will use createDataPartition() function.

```
#MODEL BUILDING
#Spliting training set into two parts based on outcome: 70% and 30%

train.model.ind <- createDataPartition(dat3$Eng_Class, p = 0.7, list = FALSE)
#training
train.model <- dat3[train.model.ind,]
#testing
test.model <- dat3[-train.model.ind,]
```

I have splitted the data,now I am looking into the class imbalance of the target variable Eng_class. class Imblance: This is the problem in machine learning, where the total number of a (positive) data class is much less than the total number of another (negative) data class. This problem is relatively common in nature and can be used in multiple fields including fraud identification, identification of accidents, medical treatment, detection of oil spillage, facial recognition, etc. A difference in the frequencies of the observed classes can

have a significant negative impact on model fitting in classification problems. One approach of overcoming such a class imbalance is to sub-sample the training data in a way that mitigates the problems. The sampling methods for this purpose are: down-sampling: randomly subset all the groups in the training collection to fit the lesser dominant class with their individual frequencies. For eg, assume that 80 percent of the samples provided for the training is first class and the remaining 20 percent are second class. Down-sampling will select the first-class arbitrarily to be the same size as the second class (so that only 40% of the overall training set is used to match the model). Caret provides a function (downSample) to do this. up-sampling: a random sample (with replacement) of a minority class of the same size as the majority class. Caret provides a feature (upSample) to achieve this. hybrid methods: majority-class approaches such as SMOTE and ROSE down-sample and synthesize new minority-class data points. There are two packages that execute such methods (DMwRand ROSE).

I will be using SMOTE() funcion which is available in the DMWR package to know about the class imbalances.

```
#class imbalances
#library(DMwR)
set.seed(214)
#class imbalance for dat3
smote_train <- SMOTE(Eng_Class ~ ., data  = dat3)
table(smote_train$Eng_Class)

##
##  okay under
##  1096   822

#class imbalance for train.model

smote_train <- SMOTE(Eng_Class ~ ., data  = train.model)
table(smote_train$Eng_Class)

##
##  okay under
##   768   576

#class imbalance for test.model

smote_train <- SMOTE(Eng_Class ~ ., data  = test.model)
table(smote_train$Eng_Class)

##
##  okay under
##   328   246
```

This is possibly the part where Caret stands out from any other package available. It provides the capability of using consistent syntax to implements more than 200 machine learning algorithms. You can use: to receive a list of all the algorithms that Caret supports

```
# See available algorithms in caret
modelnames <- paste(names(getModelInfo()), collapse=',  ')
modelnames
```

```
## [1] "ada, AdaBag, AdaBoost.M1, adaboost, amdai, ANFIS, avNNet, awnb
, awtan, bag, bagEarth, bagEarthGCV, bagFDA, bagFDAGCV, bam, bartMach
ine, bayesglm, binda, blackboost, blasso, blassoAveraged, bridge, brnn
, BstLm, bstSm, bstTree, C5.0, C5.0Cost, C5.0Rules, C5.0Tree, cforest
, chaid, CSimca, ctree, ctree2, cubist, dda, deepboost, DENFIS, dnn,
dwdLinear, dwdPoly, dwdRadial, earth, elm, enet, evtree, extraTrees,
fda, FH.GBML, FIR.DM, foba, FRBCS.CHI, FRBCS.W, FS.HGD, gam, gamboost
, gamLoess, gamSpline, gaussprLinear, gaussprPoly, gaussprRadial, gbm_h
2o, gbm, gcvEarth, GFS.FR.MOGUL, GFS.LT.RS, GFS.THRIFT, glm.nb, glm,
glmboost, glmnet_h2o, glmnet, glmStepAIC, gpls, hda, hdda, hdrda, HYF
IS, icr, J48, JRip, kernelpls, kknn, knn, krlsPoly, krlsRadial, lars
, lars2, lasso, lda, lda2, leapBackward, leapForward, leapSeq, Linda,
lm, lmStepAIC, LMT, loclda, logicBag, LogitBoost, logreg, lssvmLinear,
lssvmPoly, lssvmRadial, lvq, M5, M5Rules, manb, mda, Mlda, mlp, mlpK
erasDecay, mlpKerasDecayCost, mlpKerasDropout, mlpKerasDropoutCost, mlpML
, mlpSGD, mlpWeightDecay, mlpWeightDecayML, monmlp, msaenet, multinom,
mxnet, mxnetAdam, naive_bayes, nb, nbDiscrete, nbSearch, neuralnet, nn
et, nnls, nodeHarvest, null, OneR, ordinalNet, ordinalRF, ORFlog, ORF
pls, ORFridge, ORFsvm, ownn, pam, parRF, PART, partDSA, pcaNNet, pcr
, pda, pda2, penalized, PenalizedLDA, plr, pls, plsRglm, polr, ppr,
PRIM, protoclass, qda, QdaCov, qrf, qrnn, randomGLM, ranger, rbf, rb
fDDA, Rborist, rda, regLogistic, relaxo, rf, rFerns, RFlda, rfRules,
ridge, rlda, rlm, rmda, rocc, rotationForest, rotationForestCp, rpart,
rpart1SE, rpart2, rpartCost, rpartScore, rqlasso, rqnc, RRF, RRFglobal
, rrlda, RSimca, rvmLinear, rvmPoly, rvmRadial, SBC, sda, sdwd, simp
ls, SLAVE, slda, smda, snn, sparseLDA, spikeslab, spls, stepLDA, ste
pQDA, superpc, svmBoundrangeString, svmExpoString, svmLinear, svmLinear2
, svmLinear3, svmLinearWeights, svmLinearWeights2, svmPoly, svmRadial,
svmRadialCost, svmRadialSigma, svmRadialWeights, svmSpectrumString, tan,
tanSearch, treebag, vbmpRadial, vglmAdjCat, vglmContRatio, vglmCumulativ
e, widekernelpls, WM, wsrf, xgbDART, xgbLinear, xgbTree, xyf"
```

Here I will be building three models. They are as follows :

1.KNN (k-Nearest Neighbors)

2.SVMLINEAR (Support Vector Machines with Linear Kernel)

3.RANDOM FOREST (rf)

KNN: K-Nearest Neighbors (KNN) is one of the simplest algorithms for regression and classification problems used in Machine Learning. KNN algorithms use data to identify new data objects, based on measures of similarities (e.g. distance). Classification of the neighbors is achieved by popular neighbors.

SVMLINEAR:SVM works by mapping data into a high-dimensional feature space so that data points can be categorized even if the data can not be separated linearly. There is a

separator between the categories, then the data is transformed in such a way that the separator can be drawn as a hyperplane.

RANDOM FOREST:Random forest is a supervised learning algorithm, used both for classification and regression. Similarly, random forest algorithm builds decision trees on data samples and then gets the prediction from each of them and eventually selects the best solution by voting.

Any step in the tuning cycle can be customized. The resampling technique used to test the model 's output using a collection of parameters in Caret is bootstrap, but it offers alternatives for using k-fold, repeated k-fold and Leave-one-out cross-validation (LOOCV) which can be defined using traincontrol(). I will be using 10-Fold Cross-validation repeated 3 times in this case and classProbs = TRUE,summaryFunction = twoClassSummary, savePredictions = TRUE,sampling="smote".

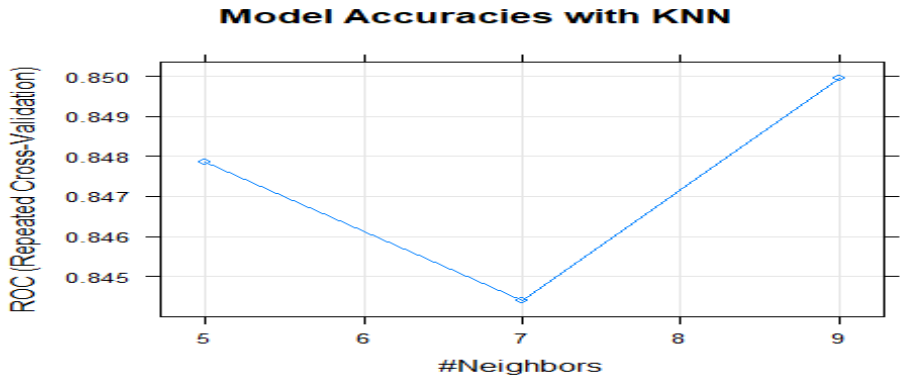
```
# setting conditions for trainControl
set.seed(214)
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 3,
                           ## Estimate class probabilities
                           classProbs = TRUE,
                           ## Evaluate performance using
                           ## the following function
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           sampling="smote")
```

1.Now building the first model i.e KNN

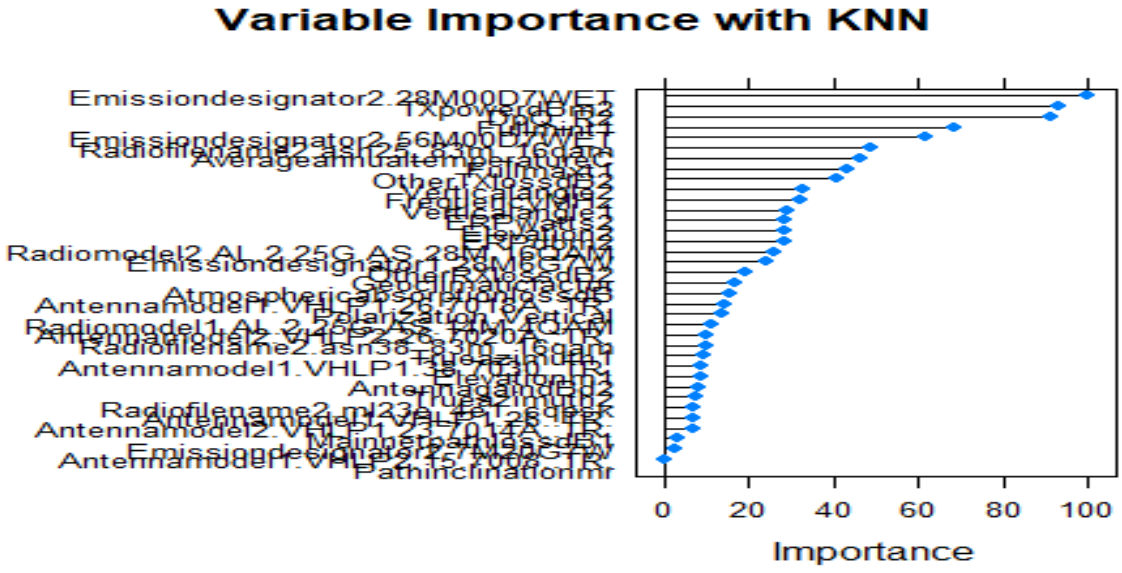
```
#1.KNN
modelLookup('knn')

##   model parameter      label forReg forClass probModel
## 1   knn         k #Neighbors   TRUE     TRUE      TRUE

# Set the seed for reproducibility
set.seed(214)

# Train the model using knn and predict on the training data itself.
#Basic Parameter Tuning
KNN_basic_tuning = train(Eng_Class ~ ., data = train.model,method='knn',trCon
trol=fitControl,metric="ROC",preProc = c("center", "scale"))
KNN_basic_tuning

## k-Nearest Neighbors
##
## 1531 samples
##   37 predictor
##    2 classes: 'okay', 'under'
##
```

```
## Pre-processing: centered (37), scaled (37)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1378, 1377, 1378, 1378, 1378, 1378, ...
## Addtional sampling using SMOTE prior to pre-processing
##
## Resampling results across tuning parameters:
##
##   k  ROC        Sens       Spec
##   5  0.8478497  0.8294505  0.6994737
##   7  0.8443920  0.8170015  0.6999123
##   9  0.8499594  0.8063180  0.7027193
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
plot(KNN_basic_tuning , main="Model Accuracies with KNN")
```



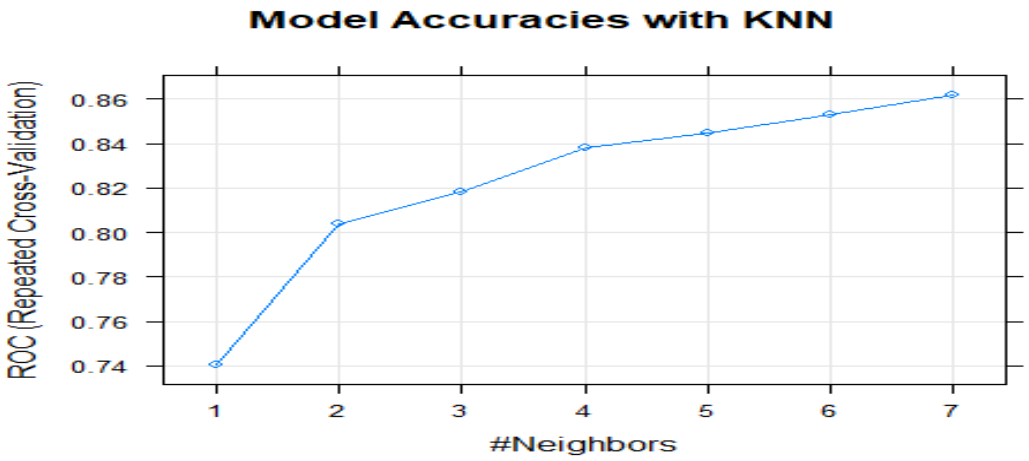

```
#variable importance of the model
varimp <- varImp(KNN_basic_tuning )
plot(varimp, main="Variable Importance with KNN")
```



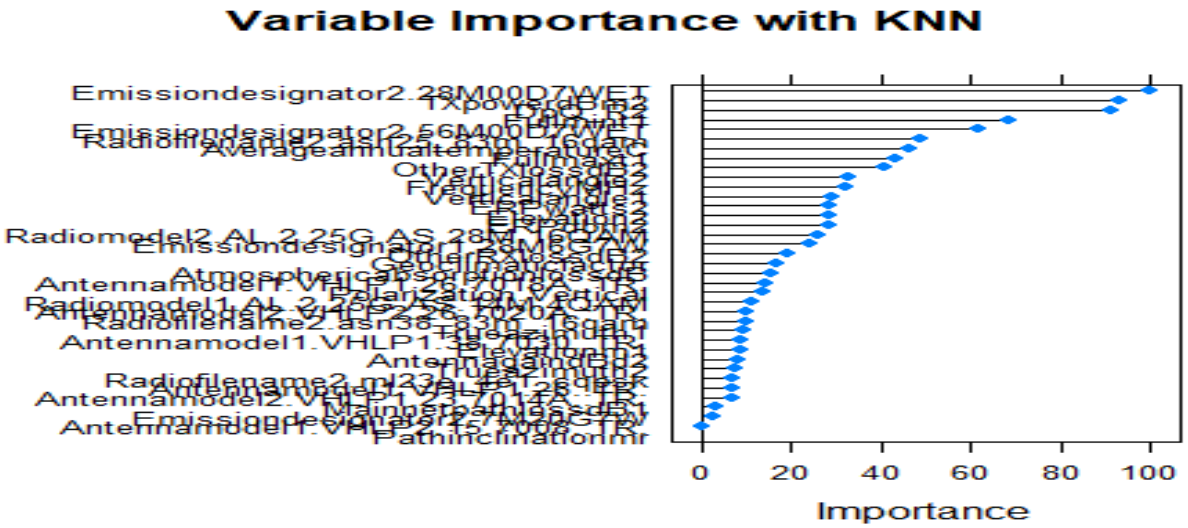
```r
#predicting by using test data and the confusion matrix
knn_predict_basic <- predict(KNN_basic_tuning ,test.model)
confusionMatrix(test.model$Eng_Class,knn_predict_basic)

#Alternate Tuning Grids
#setting seed value
set.seed(214)
#defining the grid
grid=expand.grid(k =1:7)
#settig the seed value
set.seed(214)
#building the model
KNN_alternate_tuning = train(Eng_Class ~ ., data = train.model, method='knn',
metric='ROC', tuneGrid = grid, trControl = fitControl)
KNN_alternate_tuning
```

```
## k-Nearest Neighbors
##
## 1531 samples
##   37 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1378, 1377, 1378, 1378, 1378, 1378, ...
## Addtional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   k  ROC        Sens       Spec
##   1  0.7402047  0.8389182  0.6414912
##   2  0.8040139  0.8359499  0.6535965
##   3  0.8183488  0.8409139  0.6840351
##   4  0.8383471  0.8317043  0.7013158
##   5  0.8449060  0.8349325  0.7171053
##   6  0.8530922  0.8369487  0.6892982
##   7  0.8621366  0.8322055  0.7296491
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```r
plot(KNN_alternate_tuning , main="Model Accuracies with KNN")
```




```r
#variable importance of the model
varimp1 <- varImp(KNN_alternate_tuning )
plot(varimp1, main="Variable Importance with KNN")
```

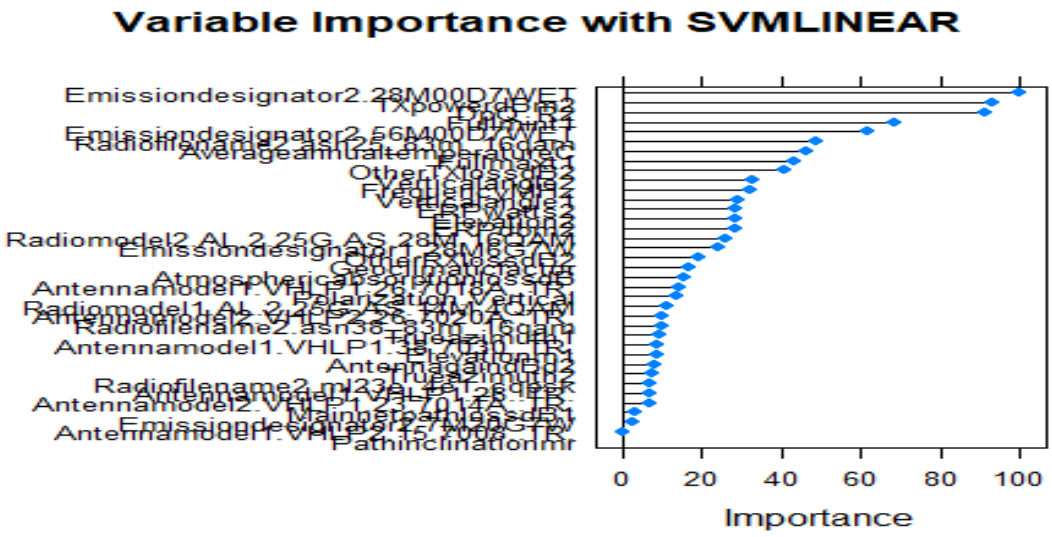



```r
#predicting by using test data and the confusion matrix
knn_predict_alternate <- predict(KNN_alternate_tuning, test.model)
confusionMatrix(test.model$Eng_Class,knn_predict_alternate)
```

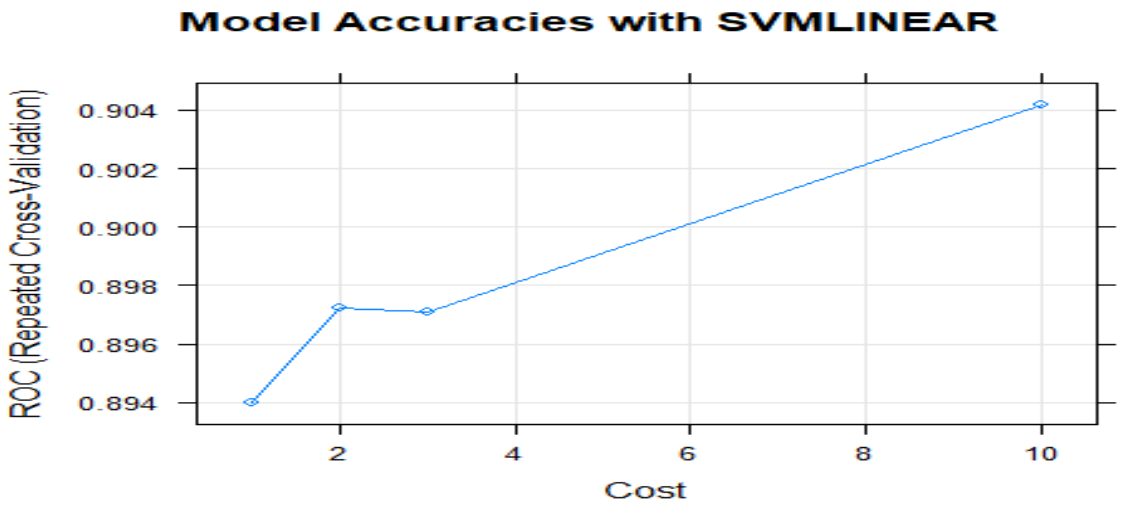
From the result we can see that the ROC was used to select the optimal model using the largest value. The final value used for the model was k = 5.Therefore the value k=5 is used for the model.This is the basic tuning of the model and the accuracy for this model is 82.9%. And then i performed alternate tuning by using tuneGrid, I have given values frim 1 to 7 for the grid i.e c(1:7). And the result for this is ROC was used to select the optimal model using the largest value. The final value used for the model was k = 7. The accuracy for this is 83.05%. Here we can observe that the accuracy of the model is increased after applying the alternate tuning model by applying tuneGrid value.

2.Now Building the second model i.e SVMLINEAR

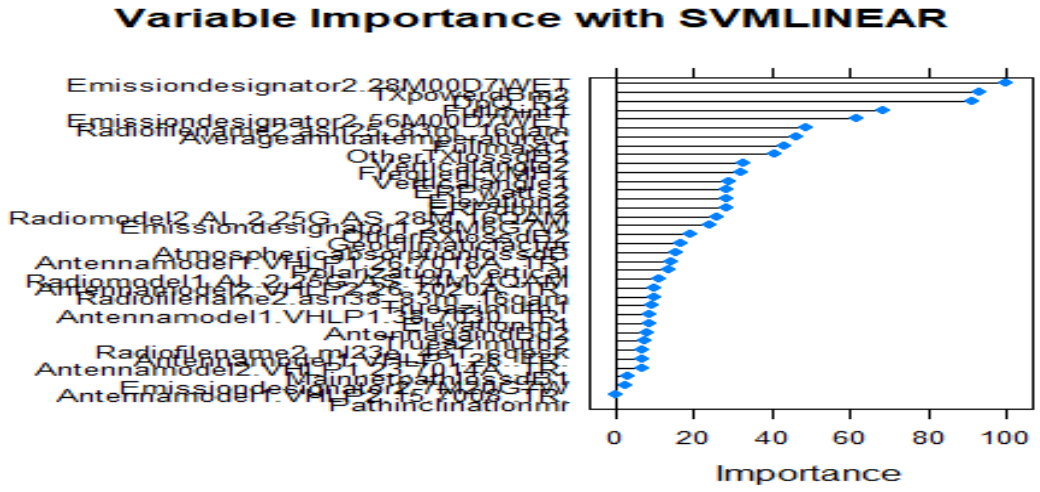
```
#2.SVMLINEAR

modelLookup('svmLinear')

##       model parameter label forReg forClass probModel
## 1 svmLinear         C  Cost   TRUE     TRUE      TRUE

#setting the seed value
set.seed(214)
#Basic Parameter Tuning
svm_basic_tuning <- train(Eng_Class ~ ., data = train.model,
                  method = "svmLinear",
                  metric = "ROC",
                  trControl = fitControl,
                  verbose = FALSE,preProc = c("center", "scale"))
svm_basic_tuning

## Support Vector Machines with Linear Kernel
##
## 1531 samples
##   37 predictor
##    2 classes: 'okay', 'under'
##
## Pre-processing: centered (37), scaled (37)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1378, 1377, 1378, 1378, 1378, 1378, ...
## Addtional sampling using SMOTE prior to pre-processing
##
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8999921  0.8677945  0.7410526
##
## Tuning parameter 'C' was held constant at a value of 1

#summary of the model
summary(svm_basic_tuning)

## Length  Class   Mode
##      1   ksvm     S4

#variable importance of the model
varimp2 <- varImp(svm_basic_tuning )
plot(varimp2, main="Variable Importance with SVMLINEAR")
```




```
#predicting by using test data and the confusion matrix

svm_predict_basic <- predict(svm_basic_tuning, test.model)
confusionMatrix(test.model$Eng_Class,svm_predict_basic)

#Alternate Tuning Grids
set.seed(214)
svm_alternate_tuning <- train(Eng_Class ~., data = train.model,
                              method = "svmLinear",
                          trControl=fitControl,
                          metric = "ROC",
                          verbose = FALSE,
                          tuneGrid=expand.grid(C = c(1,2,3,10)))



##   10  0.9041949  0.8700426  0.7320175
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 10.

#summary of the model

summary(svm_alternate_tuning)

## Length  Class   Mode
##      1   ksvm     S4

plot(svm_alternate_tuning , main="Model Accuracies with SVMLINEAR")
```




```
#variable importance of the model
varimp3 <- varImp(svm_alternate_tuning )
plot(varimp3, main="Variable Importance with SVMLINEAR")
```




```
#predicting by using test data and the confusion matrix

svm_predict_alternate<-predict(svm_alternate_tuning,test.model)
confusionMatrix(test.model$Eng_Class,svm_predict_alternate)
```

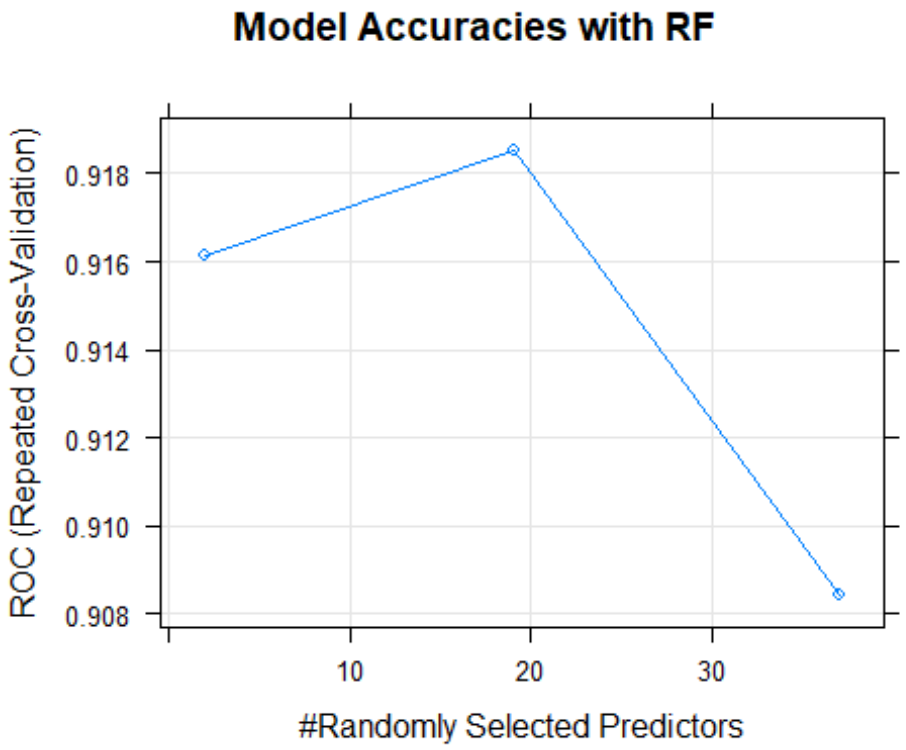
From the result we can see that Tuning parameter 'C' was held constant at a value of 1. This is the basic tuning of the model and the accuracy for this model is 84.58%. And then i performed alternate tuning by using tuneGrid c(1,2,3,10).And the result for this is ROC was used to select the optimal model using the largest value. The final value used for the model was C = 10.The accuracy for this alternate tuning model is 86.26%. Here we can observe that the accuracy of the model is increased after applying the alternate tuning model by applying tuneGrid value.

3.Now building the third model i.e RANDOM FOREST

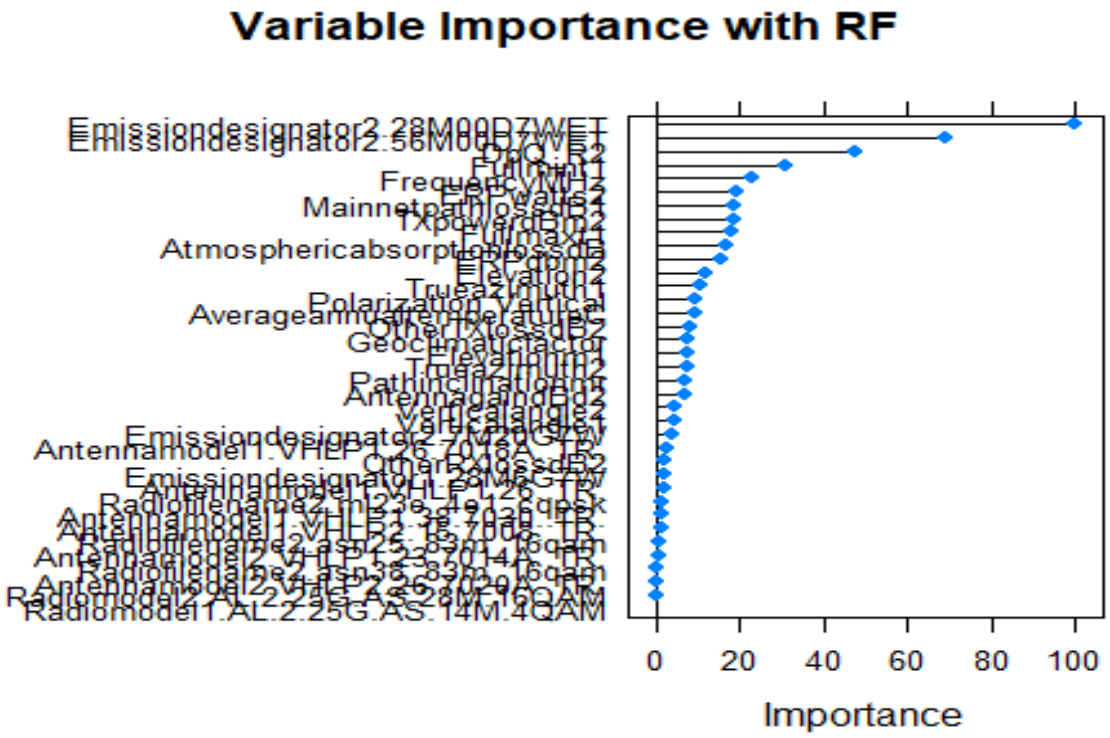
```
#3.RANDOM FOREST
set.seed(214)
#Model basic tuning

rf_basic_tuning <- train(Eng_Class ~ ., data = train.model,
                  method = "rf",
                  ## Specify which metric to optimize, by default, this is th
e accuracy
                  metric = "ROC",
                  trControl = fitControl,
                  verbose = FALSE)
rf_basic_tuning
```

```
## Random Forest
##
## 1531 samples
##   37 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1378, 1377, 1378, 1378, 1378, 1378, ...
## Addtional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.9161323  0.9307840  0.6407018
##   19    0.9185434  0.9297872  0.6810526
##   37    0.9084122  0.9210807  0.6615789
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 19.
```

```
plot(rf_basic_tuning, main="Model Accuracies with RF")
```




```r
#variable importance of the model
varimp5 <- varImp(rf_basic_tuning)
plot(varimp5, main="Variable Importance with RF")
```

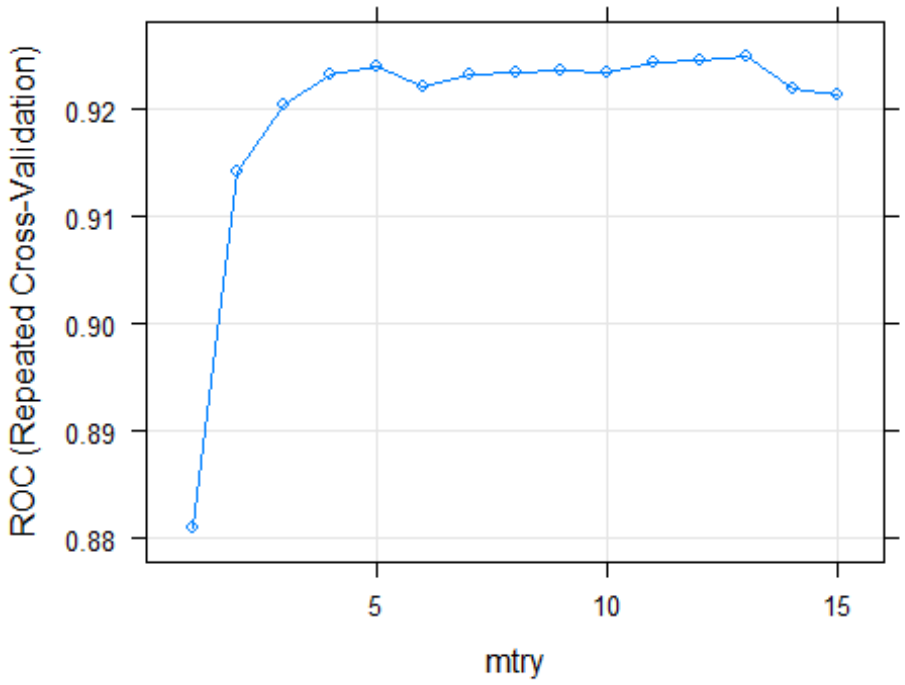


```r
#predicting by using test data and the confusion matrix
rf_basic_predict <- predict(rf_basic_tuning, test.model)
confusionMatrix(test.model$Eng_Class,rf_basic_predict)

#model alternate tuning

# setting conditions for trainControl
set.seed(214)
# setting different values of mtry for the model
man_grid <-  expand.grid(mtry = c(1:15))
# doing the grid search
set.seed(214)
# building the model
rf_alternate_tuning <- train(Eng_Class ~ ., data = train.model,
                        method = "rf",
                        ## Specify which metric to optimize, by default, this
is the                           accuracy
                        metric = "ROC",
                        trControl = fitControl,
                        verbose = FALSE,
                        tuneGrid = man_grid)

rf_alternate_tuning
```

```
## Random Forest
##
## 1531 samples
##   37 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1378, 1377, 1378, 1378, 1378, 1378, ...
## Addtional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    1    0.8808758  0.9058935  0.5938596
##    2    0.9141260  0.9315284  0.6497368
##    3    0.9203681  0.9312834  0.6750877
##    4    0.9233514  0.9273089  0.6978947
##    5    0.9239600  0.9317847  0.6809649
##    6    0.9222302  0.9320353  0.7015789
##    7    0.9232974  0.9295440  0.6982456
##    8    0.9234818  0.9335316  0.6704386
##    9    0.9236295  0.9297853  0.6861404
##   10    0.9235340  0.9280533  0.6966667
##   11    0.9243685  0.9290428  0.6858772
##   12    0.9246983  0.9337897  0.6736842
```

```
##   13    0.9250765  0.9332772  0.6837719
##   14    0.9219206  0.9315378  0.6718421
##   15    0.9214197  0.9322897  0.6753509
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 13.
```

```
plot(rf_alternate_tuning, xlab = 'mtry')
```




```
#Best tune of the model
rf_alternate_tuning$bestTune    # mtry = 10
```

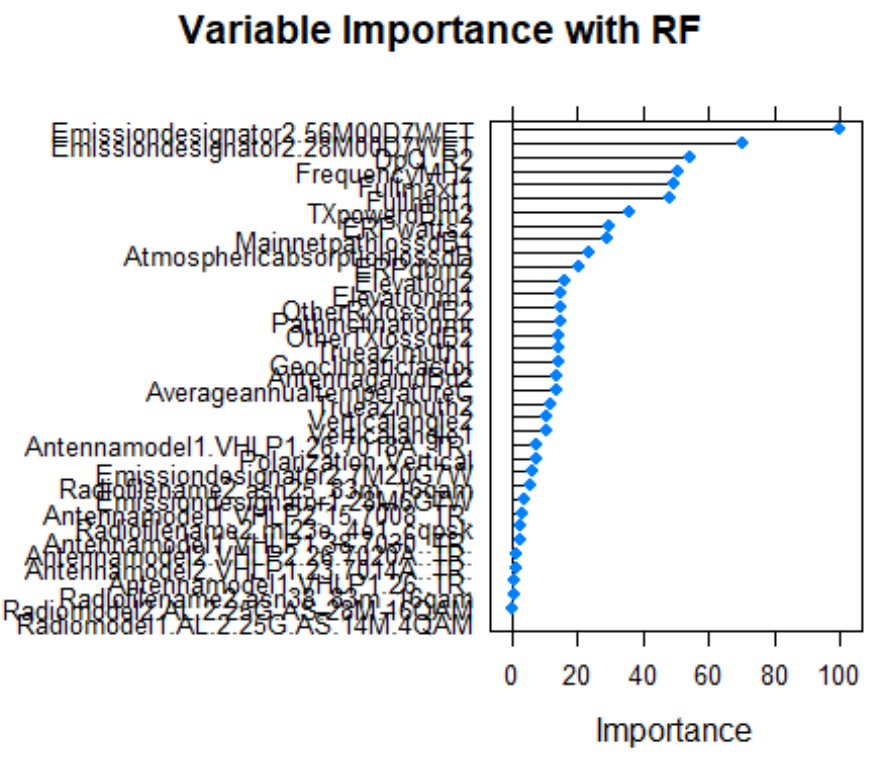
```
##    mtry
## 13   13
```

```
set.seed(214)

# building the model
# build the model by selecting the mtry given by the grid searches (mtry = 10
)
hyperparams <- expand.grid(mtry=10)

set.seed(214)
rf_alternate_tuning1 <- train(Eng_Class ~ ., data = train.model,
                        method = 'rf',
                        tuneGrid = hyperparams,
                        metric = "ROC",
```

```
                         trControl = fitControl,
                         verbose = F)
rf_alternate_tuning1
```

```
## Random Forest
##
## 1531 samples
##   37 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1378, 1377, 1378, 1378, 1378, 1378, ...
## Addtional sampling using SMOTE
##
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9241414  0.9310459  0.6687719
##
## Tuning parameter 'mtry' was held constant at a value of 10
```

```
#variable importance of the model
varimp6 <- varImp(rf_alternate_tuning1)
plot(varimp6, main="Variable Importance with RF")
```



```
#predicting by using test data and the confusion matrix

rf_alternate_predict <- predict(rf_alternate_tuning1, test.model)
confusionMatrix(test.model$Eng_Class,rf_alternate_predict)
```

From the result we can see that ROC was used to select the optimal model using the largest value. The final value used for the model was mtry = 19. This is the basic tuning of the model and the accuracy for this model is 89.47%.

And then i performed alternate tuning by using tuneGrid by giving range to the mtry from 1 to 15 i.e c(1:15).After building the model the best tune is for the mtry 10. The result is as follows ROC was used to select the optimal model using the largest value. The final value used for the model was mtry = 10. so i took that value and gave that value mtry=10 to tuneGrid and built the alternate tuning model. And the result for this model is Tuning parameter 'mtry' was held constant at a value of 10 Here we can observe that the accuracy of the model is increased after applying the alternate tuning model by applying tuneGrid value.

From the three model we can observe that the accuracy of each model is increasing after applying tunegrid i.e after applying alternate tuning.

Among all the three models the Random forest with the alternate tuning have the highest accuracy 89.62%

There among all the three models random forest model with alternate tuning is the best model.

#**********

QUESTION C

Perform feature selection on your model in c). Explain how you do this, giving a rational and comment on your results.

SOLUTION

I have to Perform feature selection on the best model from (b). My best model is random forest with alternate tuning,so i will be considering that model and performing feature selection. Feature selection: Feature Selection is the mechanism where you pick certain apps that most apply to your predictive element or performance you are interested in, automatically or manually. With irrelevant features in your results, the model's accuracy can be decreased and the model learned based on irrelevant features.

Key reasons for choosing the feature selection are:

–>It allows faster training of the machine learning algorithm.

–>This reduces a model 's complexity, which makes interpretation easier.

–>It increases a model's accuracy when the right subset is chosen.

–>It reduces the dependency on overfitting.

In feature selection there are some many methods such as filter methods,wrapper methods, Embedded Methods.

In this assignment I will be using Wrapper methods. We try to use a subset of features in wrapper models, and train a model using them. We agree to add or delete functionality from your subclass, depending on the inferences we generate from the previous iteration. In fact the problem is simplified to a search problem. These techniques are typically very costly in numerical terms. Some common examples of wrapper methods are forward feature selection, backward feature elimination, recursive feature elimination, etc.

Forward Selection:

Forward Selection is an iterative process with no function in the model. We keep inserting the function in each iteration that better enhances our model until the introduction of a new variable does not boost model efficiency.

Backward Elimination:

We start with all the features in reverse elimination and remove the least significant feature at each iteration which improves model efficiency. We repeat this until no change in the elimination of features is found.

Recursive Feature elimination:

This is a greedy optimization algorithm that tries to find the highest performing subset of features. It generates models repeatedly and leaves the better or worst performing function aside at every iteration. This builds the next iteration with the features on the left before all functions are removed. It then lists the features according to the order of their elimination.

Among this three i will be performing feature selection by using **Recursive Feature elimination(rfe)** There are several arguments in rfe().They are as follows:

x, a predictive matrix or data set of variables

Y, the resulting vector (numerical or factor)

Sizes, an integer variable for the unique dimension of the subset to be evaluated (which does not contain ncol(x))

RfeControl, a set of choices that can be used to determine the model and statistical approaches, rankings etc.

In rfeControl$functions a range of functions must be defined for a specific model. There are a variety of predefined collections of functions for different models, including: linear regression (in the lmFuncs object), random forest (rfFuncs), naive Bayes (nbFuncs), bagged trees (treebagFuncs), and functions that can be used for caret train feature (caretFuncs).

The above is useful because the model has parameters of tuning which have to be decided at each iteration.

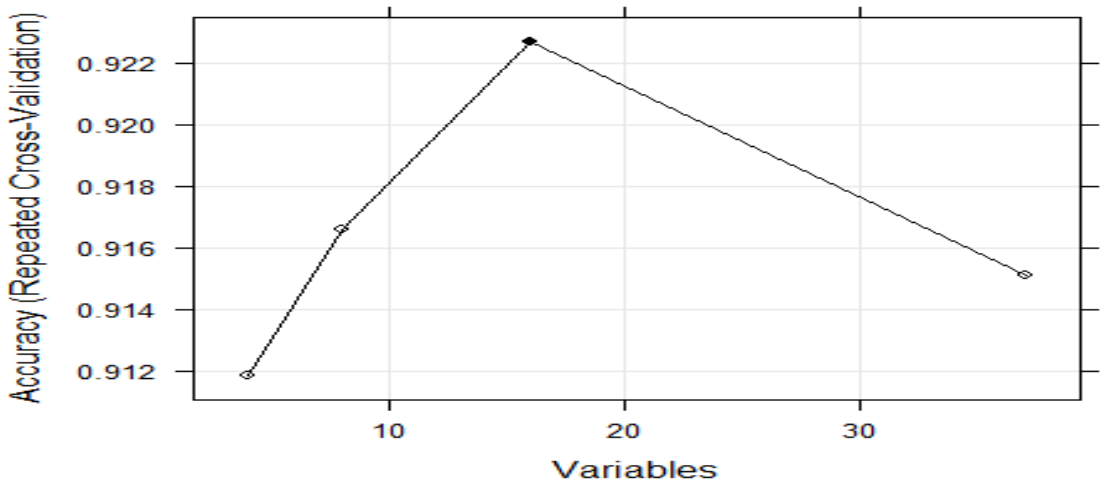
To fit random forset models, the rfFuncs set of functions can be used. To do this, a control object is created with the rfeControl function by using the parameters method = "repeatedcv", number = 10,repeats=3,returnResamp = "all",functions = rfFuncs,saveDetails = TRUE,verbose = FALSE. The verbose option prevents copious amounts of output from being produced.

```
#FEATURE SELECTION FOR THE BEST MODEL
#setting the seed
set.seed(214)
#defining the rfecontrol() function
rctrl1 <- rfeControl(method = "repeatedcv",
                     number = 10,
                     repeats=3,
                     returnResamp = "all",
                     functions = rfFuncs,
                     saveDetails = TRUE,verbose = FALSE)
#setting the seed value
set.seed(214)
#performing the feature selection using random forest method
model <- rfe(Eng_Class ~., data = train.model,
             method = "rf",
             trControl = trainControl(method = "cv",
                                      classProbs = TRUE),
             rfeControl = rctrl1)
#printing the model
model
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold, repeated 3 times)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          4   0.9118 0.5243    0.01352 0.08306
##          8   0.9166 0.5434    0.01774 0.10385
##         16   0.9227 0.5749    0.01775 0.11141        *
##         37   0.9151 0.5053    0.01568 0.11589
##
## The top 5 variables (out of 16):
##    DpQ_R2, Fullmaxt1, Fullmint1, TXpowerdBm2, MainnetpathlossdB1
```

```
#priting the total number of features
predictors(model)
```

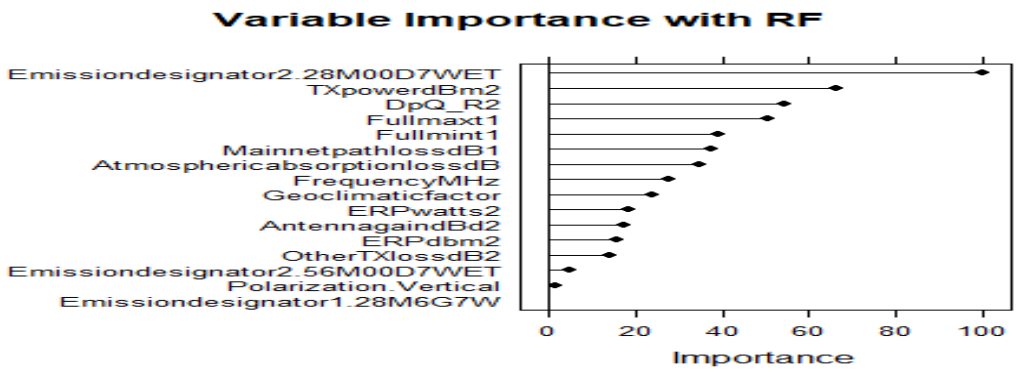
```
##  [1] "DpQ_R2"                                "Fullmaxt1"
##  [3] "Fullmint1"                             "TXpowerdBm2"
##  [5] "MainnetpathlossdB1"                    "AtmosphericabsorptionlossdB"
##  [7] "ERPwatts2"                             "ERPdbm2"
##  [9] "Emissiondesignator2.56M00D7WET"        "Emissiondesignator2.28M00D7WET"
## [11] "FrequencyMHz"                          "OtherTXlossdB2"
## [13] "Polarization.Vertical"                 "Antennamodel1.VHLP1.26.7018A..TR
."
## [15] "AntennagaindBd2"                       "Emissiondesignator1.28M6G7W"

#printing the model fir
model$fit

##
## Call:
##  randomForest(x = x, y = y, importance = TRUE, method = "rf",      trContr
ol = ..2)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 7.38%
## Confusion matrix:
##       okay under class.error
## okay  1319    20  0.01493652
## under   93    99  0.48437500

#printing the  head model$sample
head(model$resample)

##   Variables  Accuracy     Kappa .cell1 .cell2 .cell3 .cell4    Resample
## 1         4 0.9346405 0.6524307    132      2      8     11 Fold01.Rep1
## 2         8 0.9346405 0.6524307    132      2      8     11 Fold01.Rep1
## 3        16 0.9477124 0.7219446    133      1      7     12 Fold01.Rep1
## 4        37 0.9281046 0.6274076    131      3      8     11 Fold01.Rep1
## 5         4 0.9013158 0.5384615    126      7      8     11 Fold02.Rep1
## 6         8 0.8947368 0.4960630    126      7      9     10 Fold02.Rep1

#plotting the graph for the model
trellis.par.set(caretTheme())
plot(model, type = c("g", "o"))
```




```
#building the model by considering all the features after performing feature
selection by using random forest method.
rf_feature_selection <- train(Eng_Class ~Fullmaxt1+DpQ_R2+MainnetpathlossdB1+
Fullmint1+TXpowerdBm2+ERPwatts2+
                                ERPdbm2+Emissiondesignator2.28M00D7WET+Atmos
phericabsorptionlossdB+Emissiondesignator2.56M00D7WET+FrequencyMHz+Polarizati
on.Vertical+AntennagaindBd2+OtherTXlossdB2+Geoclimaticfactor+Emissiondesignat
or1.28M6G7W, data = train.model,
                              method = 'rf',
                              tuneGrid = hyperparams,
                              metric = "ROC",
                              trControl = fitControl,
                              verbose = F)
rf_feature_selection
```

```
## Random Forest
##
## 1531 samples
##   16 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1378, 1378, 1378, 1378, 1377, 1379, ...
## Addtional sampling using SMOTE
##
## Resampling results:
##
##   ROC       Sens       Spec
##   0.924697  0.9171025  0.7342982
##
## Tuning parameter 'mtry' was held constant at a value of 10
```

```
#summary of the model
summary(rf_feature_selection)

#variable importance of the model
varimp7 <- varImp(rf_feature_selection)
plot(varimp7, main="Variable Importance with RF")
```




```
#predicting by using test data and the confusion matrix
rf_feature_predict <- predict(rf_feature_selection, test.model)
confusionMatrix(test.model$Eng_Class,rf_feature_predict)
```

Therefore I performed feature selection by using random forest(rf) method and i got 16 features and they are as follows Fullmaxt1,DpQ_R2,MainnetpathlossdB1,Fullmint1,TXpowerdBm2,ERPwatts2, ERPdbm2,Emissiondesignator2.28M00D7WET,AtmosphericabsorptionlossdB,Emissiondesignator2.56M00D7WET,FrequencyMHz,Polarization.Vertical,AntennagaindBd2,OtherTXlossdB2,Geoclimaticfactor,Emissiondesignator1.28M6G7W.

After getting these features i built one model by using all these features by using random forest method to just have a look at the accuracy with these 16 features. And the result for the model is as follows Tuning parameter 'mtry' was held constant at a value of 10 And the accuracy is 90.69%

#**********

QUESTION D

For your best model explain to Daniel how this model works. Give and explain the cost/loss function used in your modelling. Word count 750.

SOLUTION

RANDOM FOREST: Random forest is a supervised learning algorithm, used both for classification and regression. Similarly, the random forest algorithm builds decision trees on data samples and then gets the prediction from each of them and eventually selects the best solution by voting. Implementation of the Algorithm: 1.Random sample collection

from the dataset. 2. Creating the decision tree with each sample and making outcomes of each decision tree predictive. 3. Get votes for every predicted outcome. 4. Selecting the predictive result as our final prediction, with the most votes.

COST FUNCTION: Cost functions in machine learning are functions that help determine the offset of predictions made during the training phase by a machine learning model with respect to the actual results. These are used in those supervised learning algorithms which use techniques of optimization. These algorithms are notable examples of regression, logistic regression, neural networks.cost functions are also known as the Loss functions. In machine learning, there are several cost functions, and each has its own use cases based on whether it is a problem of regression or classification.

Why Cost Functions in Machine Learning: The model assumes the initial weights arbitrarily during the training process and tries to make a guess about the training results. But how does the developer get to learn how far from the forecast it came from, So it obviously needs this knowledge so that, in the next iteration on training details, it can change the weight accordingly. This is where the cost function is used. The cost function is a function that takes a model and real outputs from all forecast outcomes and measures how much the model was incorrect in its prediction. Now with this quantifiable cost function information, the model tries to adjust its weight on training data for the next iteration to further reduce the error given by cost function. The goal of the training process is to establish weights that decrease this error in each repetition to the point that it cannot be further reduced. This is basically a problem of optimization.

This cost function is often the Gini index for classification problems, which measures the integrity of the data classes generated by the dividing point. In the case of a two-class classification problem, a Gini index of 0 is total purity where class values are completely divided into two classes. Split points are chosen in a decision tree by finding the attribute and the value of that attribute which results in the lowest cost. Finding the best split point in a decision tree involves estimating the cost of each value for each input variable in the training dataset.

This technique is performed for bagging and random forest on a subset of the testing dataset, made with substitution. Replacement sampling means the same row can be picked and added more than once to the sample. We can update the Random Forest procedure. Instead of mentioning all values in search of input attributes if the split is with the lowest cost, we can construct a selection of the input attributes to consider. A set of input attributes will be chosen randomly and without substitution, ensuring that each input attribute has to be weighed only once when looking for the lowest cost dividing point. Gini Impurity: Gini impurity is a measure of how often a randomly selected element from the set would be mislabeled if it were randomly labeled according to the label distribution in the subset. Gini impurity can be calculated by summing the probability {displaystyle p {i}}p {i} of an item with the {displaystyle i}i label that is selected times the probability {displaystyle sum {kneq i}p {k}=1-p {i}}}{displaystyle sum {kneq i}p {k}=1-p {i}}} of an error in categorizing that item. If all cases in the node fall into a single target group, it achieves its minimum (zero). Gini impurity is also a physical measure of information and corresponds to Tsallis Entropy with a coefficient of deformation {displaystyle q=2}q=2, and is correlated

in physics with the loss of information in out-of-equilibrium, non-extensive, dissipative and quantic systems. The normal Boltzmann-Gibbs or Shannon entropy is recovered for the limit {displaystyle qto 1}qto 1 The impurity of the Gini, in this case, is just a modification of the normal entropy test for decision trees. To compute Gini impurity for a set of items with {displaystyle J}J classes, suppose {displaystyle iin {1,2,...,J}}{displaystyle iin {1,2,...,J}}, and let {displaystyle p_{i}}p_{i} be the fraction of items labeled with class {displaystyle i}i in the set.

$$displaystyleoperatornameI_G(p) = sum_{i=1}^{J} p_i sum_{kneqi} p_k = sum_{i=1}^{J} p_i(1-p_i)$$
$$= sum_{i=1}^{J}(p_i - {p_i}^2) = sum_{i=1}^{J} p_i - sum_{i=1}^{J} {p_i}^2$$
$$= 1 - sum_{i=1}^{J} {p_i}^2 displaystyleoperatornameI_G(p) = sum_{i=1}^{J} p_i sum_{kneqi} p_k$$
$$= sum_{i=1}^{J} p_i(1-p_i) = sum_{i=1}^{J}(p_i - {p_i}^2) = sum_{i=1}^{J} p_i - sum_{i=1}^{J} {p_i}^2 = 1 - sum_{i=1}^{J} {p_i}^2$$

Out-of-bag (OOB) error is a method for measuring random forest prediction error to sub-sample data samples used for training.

```
# stastics of rf_alternate_tuning final model
rf_alternate_tuning$finalModel

##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, verbose = FALSE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 6.47%
## Confusion matrix:
##       okay under class.error
## okay   732    36  0.04687500
## under   51   525  0.08854167

#The OOB estimate of  error rate is 7.14%
```

The OOB estimate of error rate is 7.14%

#***

QUESTION E

Daniel is primarily concerned with finding the under engineered masts as these are the ones that cause outages, so incorrectly 'scoring' a mast as under when is it okay is not as bad as incorrectly 'scoring' a mast as okay when it is under; you can take the ratio here of misclassification 'costs' as 1:h, where h = {8, 16, 24}, i.e. h can take a value of 8, 16 or 24.Redo your modelling using your best model above and comment on your new results.

SOLUTION

MISSCLASSIFICATION COST:

Within several practical classification tasks, the performance of a classification model differs for specific classes of the target concept is not significant. Models showing apparently the same results in terms of the overall misclassification rate that greatly vary in actual usefulness based on which classes they effectively predict and for which they fail. This is particularly true for all types of diagnostic or anomaly detection tasks in which some model errors may be more severe or tolerable than others.

TO the criteria for classification models in these situations are sufficiently defined, true-estimated misclassification costs are used, assigned to different pairs of predicted and true classes. In order to make it cost-sensitive, they will also include additional performance measures for model evaluation, but also-and most significantly-be built into modeling. In other words, misclassification may occur because of property selection which is not suitable for classification. When all classes, groups, or categories of a variable have the same error rate or probability of misclassification then misclassification is considered to be the case.

The cost of misclassification is an important consideration for determining the efficiency of imbalanced datasets in classification. However, in certain cases solving the cost matrix for misclassification is not a straightforward process. A simple approach for identifying the costs of misclassification is to distribute them manually based on user expertise or inverse measure the costs based on the class distribution. More complex solutions can be found by applying the features' importance to adaptive equations.

Most machine learning algorithms assume all of a model's misclassification errors are equal.

Sometimes this is not the case with imbalanced classified problems where the lack of a positive or minority class situation is worse than the incorrect classification of an example from the negative or majority class. There are other real-world examples, such as spam email analysis, medical condition diagnosis, or scam recognition. In both of these cases, a false negative (a case missing) is worse or more costly than a false positive.

The misclassification is calculated for the best model i.e. the random forest after applying alternate tuning and the misclassification is 0.1038168

```
# Misclassification

(tab <- table(test.model$Eng_Class,rf_alternate_predict))

1-sum(diag(tab))/sum(tab)

#the misclassification is 0.1038
```

My best model is the random forest with alternate tuning, so I will be considering that model.

I will redo the model by considering the given h value. Here I can take the ratio here of

misclassification ‘costs’ as 1:h, where h = {8, 16, 24}, i.e. h can take a value of 8, 16 or 24.

i.e. h= {8,16,24}. So, I will be considering h=24.

I have given range to the mtry from 1 to 16 i.e. c(1:24) as h=24.After building the model the best tune is for the mtry 6.

The result is as follows

ROC was used to select the optimal model using the largest value.

The final value used for the model was mtry = 6.

I have given mtry=6 and build the model the result for the model is as follows

Tuning parameter 'mtry' was held constant at a value of 6

And the accuracy for the model is 89.47%.

This accuracy is same as the accuracy of random forest with basic tuning.

#**********

QUESTION F

Using the scoring data set provided predict whether these radio masts will be okay or under engineered using your best model to part d) and comment.

SOLUTION

In this task I have to Use the scoring data set provided and to predict whether these radio masts will be okay or under engineered by using your best model.

The best model is random forest with alternate tuning so I will be using that model to predict the data.

Here also I performed data reduction by following the same process that I followed in question(a).

Iread the file and stored in the variable score.And then I converted in to dataframe , and converted all categorical data to factors. After that I deleted "Antennafilename1", "Antennafilename2", "RFDBid" these three columns and applied zero and near zero variance predictors ,dummy variables concept and converted the whole data in to numerical data. After that I used random forest model with alternate tuning to predict the scoring dataset.

From the Result the radio masts for okay is 383 and for under is 553

#**********

QUESTION G

Explain in your own words and using appropriate referencing ANY TWO of the following

(400 – 750 words per topic), use diagrams where appropriate. Reference your work.

QUESTION G(i)

How does convolution layers and max pooling actually work in CNNs for an image classifier with 3 dimensions, i.e. height, length and color channels? Give also a broad overview understanding of what both layers do in deep CNN for image classification.

SOLUTION

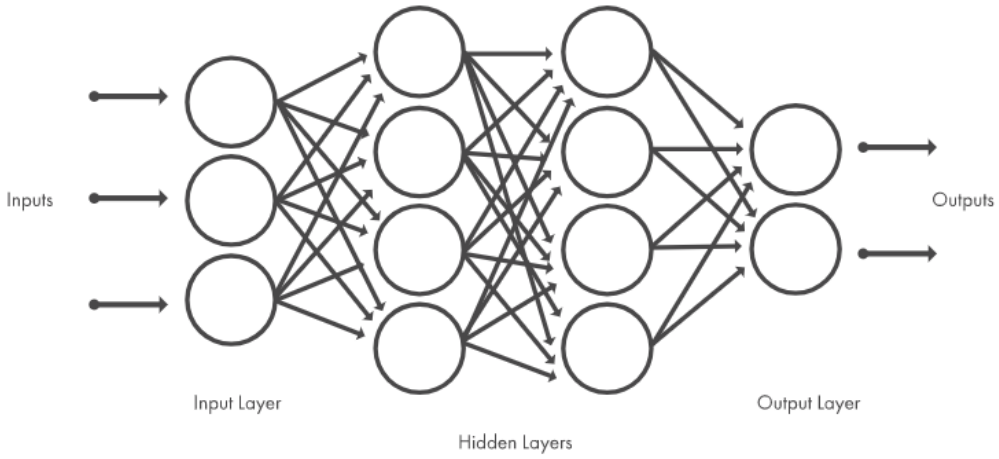
In neural networks, Convolutional Neural Network[1] is one of the main categories to do image recognition, image classification. Detections of objects, face recognition, etc., are some of the fields of which CNNs are commonly used.
CNN image recognition takes, handles, and classifies an input image in some categories (e.g., Dog, Cat, Lion). Computers interpret an input image as a pixel array and this depends on the resolution of the image. This should see h x w x d (h = Height, w = Distance, d = Dimension) depending on the image size. Technically, each input image can move through a sequence of convolution layers with filters, pooling, completely connected layers.

Layers in CNN:

Unlike most neural networks, CNN [2]consists of an input layer, an output layer, and several hidden layers within them. These layers execute operations that modify data for context-specific better learning.

Three of the commonest layers are:

convolution, activation or ReLU, and pooling.



1. Convolution:

It takes the input images into a series of convolutionary filters, each detecting certain characteristics from the pictures.

2. Rectified linear unit (ReLU):

It is made for easier and more efficient training by converting negative values to zero and holding positive values. This is sometimes referred to as activation because only the activated characteristics are transported to the next layer.

3. Pooling:

By performing nonlinear downsampling, pooling simplifies output, reducing the number of parameters the network needs to learn.

Spatial Pooling can be of different types:

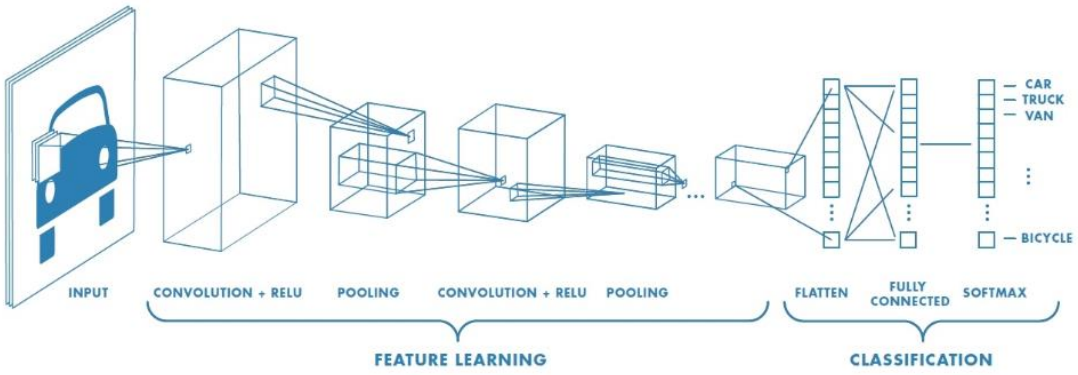
Max Pooling

Average Pooling

Sum Pooling

These operations are replicated over dozens or hundreds of layers, learning to identify various features on each layer.

Max pooling is taken from the rectified feature map with the largest dimension. The average pooling could also take on the largest item. Amount of all function map elements label as sum pooling.



Example of a network with many convolutional layers. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer.

Classification Layers:

The architecture of a CNN moves to classification, after learning features in several layers. The next-to-last layer is a completely connected layer that outputs a K dimensions matrix, where K is the number of groups that the network will predict. This vector contains the probability of each image being scored for every degree. To provide the classification output, the final layer of the CNN architecture uses a classification layer such as softmax.

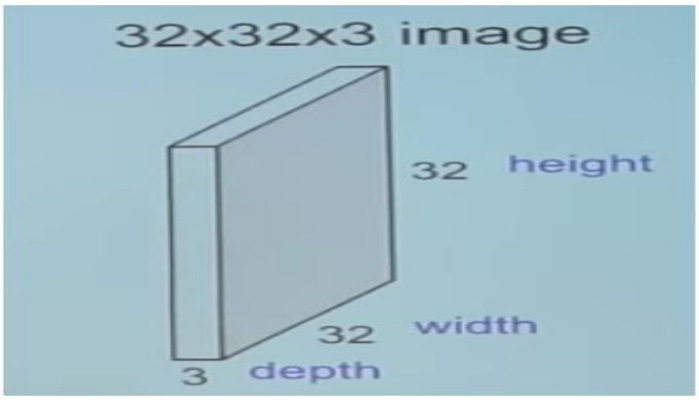
Dimensions (n× n ×c) are combined with different kernels and the outputs obtained from these kernels are stacked to form an image, and the Relu activation unit is applied to each pixel of this image, and after this process, a Maxpool layer may be added. The weights of the kernel are learned from back propagation. The hyperparameters should be carefully

chosen such as Padding, Stride, Kernels (p, s, k) The Picture and Kernel channel width should be identical. That is the underlying mechanism that happens on CNN.

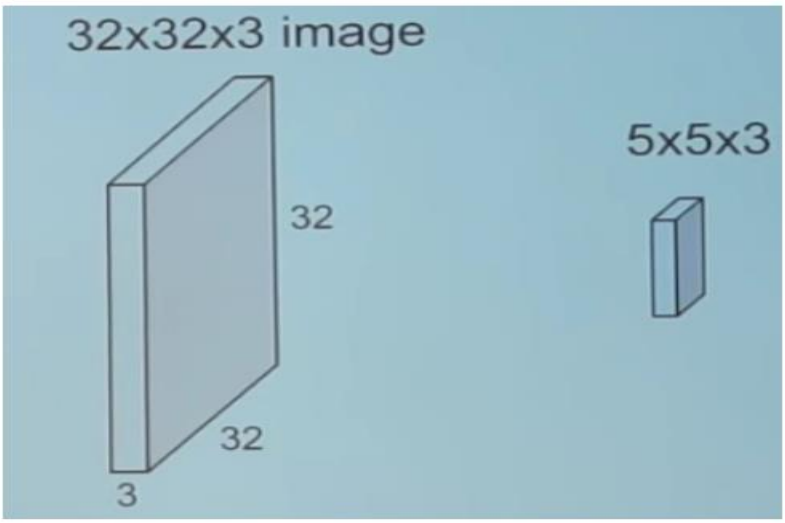
The example is as follows:

Consider an image with dimensions (32 ×32×3) with a size kernel (5 ×5×3). The picture is associated with the kernel. Convolution is performed on matrices where they multiply and add up the corresponding cells. The resulting output represents a scalar. After the process of convolution, the image element is (28×28). In this, 6 kernels are generated when the input image is combined with all the 6 kernels 6-dimensional images (28×28) that are stacked to form (28×28×6) image.

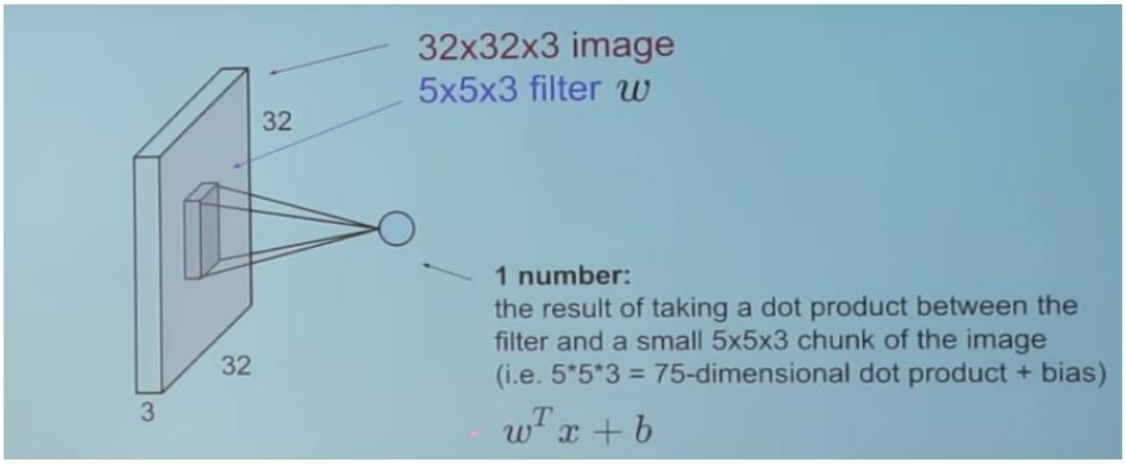
NOTE: THE EXAMPLE AND THE FIGURES FOR THIS EXAMPLE IS TAKEN FROM[3]



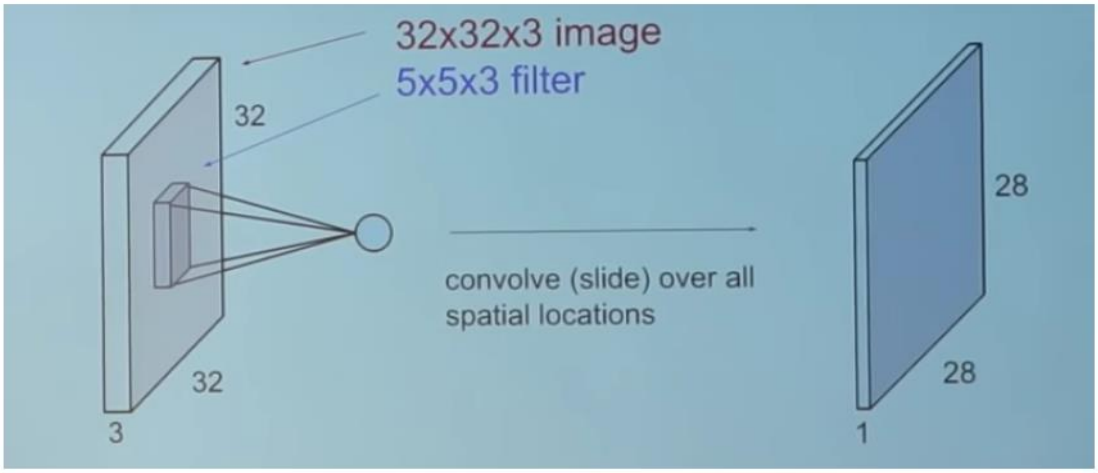
1. Example of a RGB image (let's call it 'input image')



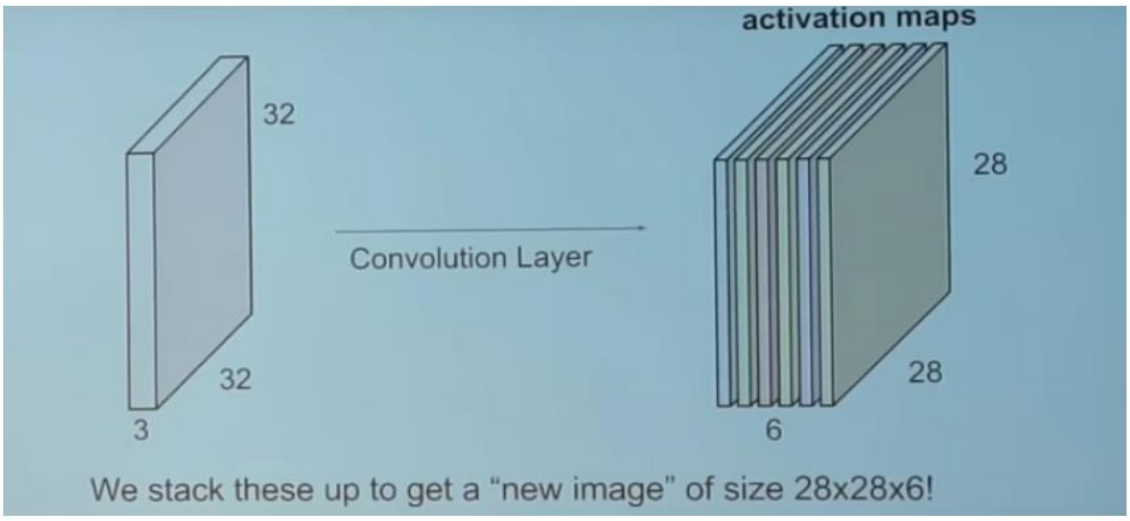
2. Convolving an image with a filter



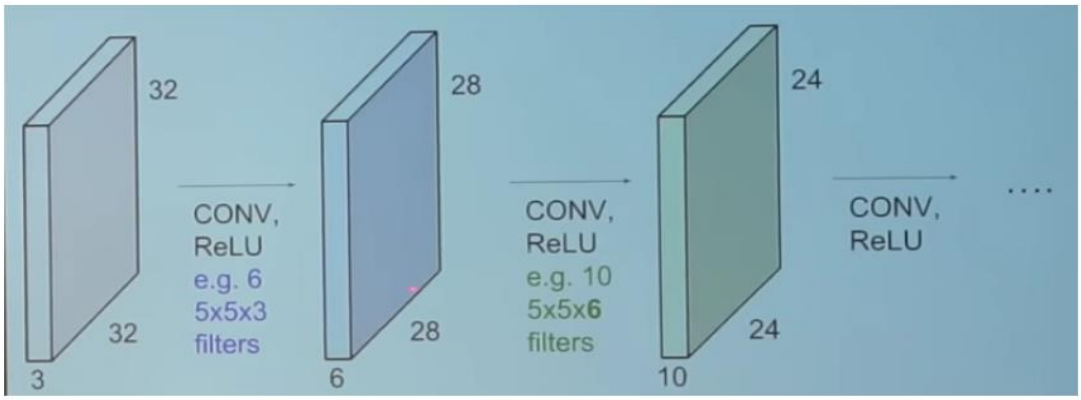
3. This is how it looks



4. This!



5. Convolution Layer

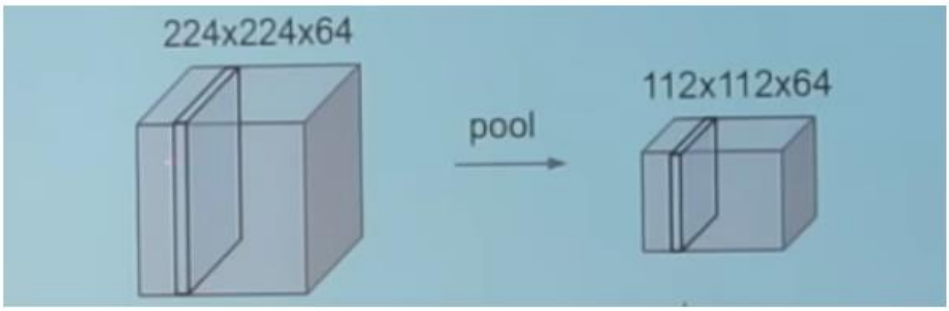


6. Convolution Layers in sequence

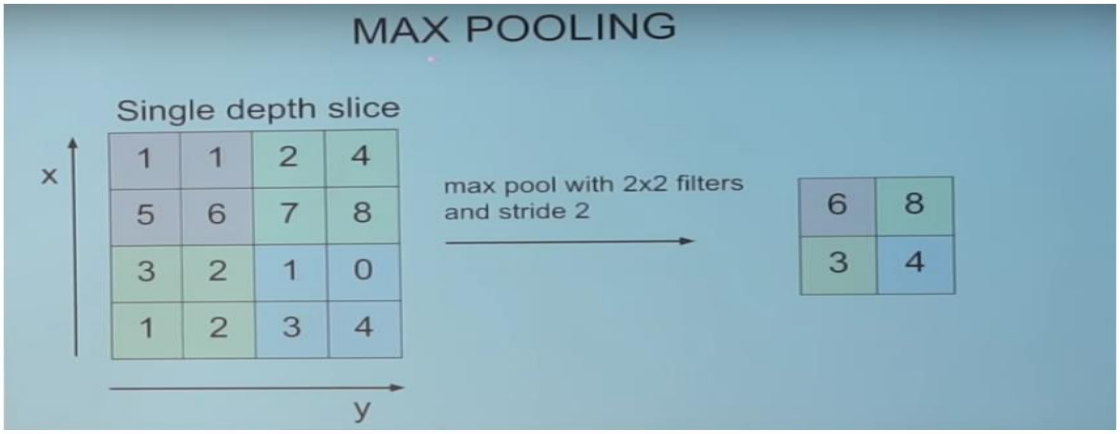
It's a Convolutionary layer. Relu activation applies to every stacked pixel of images. There would be several convolution & Relu layers in a deep-layered CNN with different kernel dimensions but with the same number of channels as Image and kernel, weights are initialized randomly that will be modified through the Back propagation process. In the deep layered CNN, called the pooling layer, there is one more layer that is sometimes included. There are several levels of pooling available, such as Min pooling, Max pooling, Average pooling, and so on. Even in deep layered CNN, the Max pooling layer is widely used.

CNN should be able to detect the features even if the scale invariance, the rotational invariance in the image, is present. This Pooling layer is used to prevent this. The pooling layer is often used in the images to minimize dimensionality while removing unnecessary elements. Also, Stride layers can be used for dimensional reduction as well, but it is sluggish because it is required to execute a convolution process. Max pooling works by choosing a maximum value (hyper parameter) from the pixel matrix while preserving the

sharp features from the image. A standard CNN requires several layers of the Convolution, Relu, Pooling.



Pooling



Max Pooling

QUESTION G(iii)

Describe in detail a deep learning methodology that are used in either (you may not usethe same methodology as you use in another assignment, e.g. Research Methods):

(1) text analytics or (2) time series analysis

SOLUTION

Text Analysis [4]is about analyzing text to derive from its machine-readable data. Text Analysis aims at making organized data out of free text material. The method can be seen as slicing and dicing heaps of unstructured, heterogeneous records into easy-to-manage data points and translating them. The basic and advanced methods in Text Analysis are:

Basic methods:

Word Frequency, Collocation, Concordance

Advanced methods:

Text Classification, Sentiment Analysis, Topic Analysis, Intent Detection, Text Extraction, Keyword Extraction, Entity Recognition, Word Sense Disambiguation

Deep Learning is a set of algorithms and techniques that uses huge amounts of training data to generate rich representations of texts which can then be fed into machine learning-based models of different kinds that will make much more accurate predictions than traditional machine learning models.

Methodology:

This section explains the pre-processing methods, the feature extraction processes, and the deep learning models used in this analysis.

Pre-processing

1. Tokenization:

Words in every sentence must be segmented and transformed into vectors

2.Part-of-speech Tagging:

Refers to the process of assigning a grammatical category, such as noun, verb, etc. to the tokens that have been detected.

3.Parsing:

This refers to the process of determining the syntactic structure of a text.

4.Dependency Parsing: Dependency parsing is the process of using dependency grammar to determine the syntactic structure of a sentence.

5.Constituency Parsing:

Constituency parsing refers to the process of using constituency grammar to determine the syntactic structure of a sentence.

6.Lemmatization and Stemming:

Lemmatization makes use of dictionaries and a much more complex morphological analysis and stemming is usually based on rules that trim word beginnings and endings

7. Stopword Removal :

To provide accurate automated analysis of the text, it is important that we remove from all the words that are very frequent but provide very little semantic information. These words are also known as stopwords.

Convolutional Neural Network(CNN):

CNN is a neural network that has a vital component, a convolutional layer, that can analyze the relationship between neighboring elements by using several filters and a sliding filter that is based on stride size.

STEPS in programming CNN:

Step 1: Getting the Data: Loading the dataset.

Step 2: Initialize parameters: We define methods to initialize both the filters for the convolutional layers and the weights for dense layers.

Stage 3: Determines the operations on backpropagation

To calculate the gradients that will force the network to update its weights and optimize its objective, we need to define methods through the convolutionary and max-pooling layers that propagate gradients.

Step 4: Network Building

We now describe in spirit abstraction a system that incorporates the forward and backward operations of a convolutionary neural network. It takes the parameters and hyperparameters of the network as inputs and spits the gradients out

Step 5: Training the network: Training the model

Step 6: Results

Once you fit a deep learning neural network model, you must evaluate its performance by using Accuracy, Precision, Recall and F1 Score.

Accuracy is the number of correct predictions the classifier has made divided by the total number of predictions.

Precision states how many texts were predicted correctly out of the ones that were predicted as belonging to a given tag.

The **recall** states how many texts were predicted correctly out of the ones that should have been predicted as belonging to a given tag.

The **F1 score** is the harmonic means of precision and recall. It tells how the classifier performs if the same importance is given to precision and recall.

Cross-validation:

Cross-validation is quite frequently used to evaluate the performance of text classifiers. First of all, the training dataset is randomly split into a number of equal-length subsets Then, all the subsets except for one are used to train a classifier and this classifier is used to predict the texts in the remaining subset. Next, all the performance metrics are computed.

Finally, the process is repeated with a new testing fold until all the folds have been used for testing purposes. Once all folds have been used, the average performance metrics are computed and the evaluation process is finished.

Text Extraction: It refers to the process of recognizing structured data of information from unstructured text.

Regular Expressions: a regular expression defines a pattern of characters that will be associated with a tag.

Conditional Random Fields:

This approach learns the patterns to be extracted by weighing a set of features of the sequences of words that appear in a text.

Evaluation:

Extractors are sometimes evaluated by calculating the same standard performance metrics we have explained above for text classification, namely, accuracy, precision, recall, and f1 score. However, these metrics do not account for partial matches of patterns.

REFERENCES:

CITATIONS BY USING IEEE STYLE:

[1] Prabhu, "Understanding of Convolutional Neural Network (CNN) — Deep Learning," *Medium*, Nov. 21, 2019. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (accessed May 16, 2020).

[2] "Convolutional Neural Network." https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html (accessed May 15, 2020).

[3] H. Pokharna, "The best explanation of Convolutional Neural Networks on the Internet!," *Medium*, Jul. 28, 2016. https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8 (accessed May 16, 2020).

[4] "Text Analysis," *MonkeyLearn*. https://monkeylearn.com/text-analysis (accessed May 16, 2020).