Data Analysis - Classification

Varun Bansal

2023-04-01

Setting the work directory.

Loading and attaching all the necessary packages.

```
#Load packages

if(!require(tinytex)){install.packages("tinytex")
  library("tinytex")
}
```

```
## Loading required package: tinytex
```

```
if (!require(lattice)) {install.packages("lattice")
  library(lattice)
}
```

```
## Loading required package: lattice
```

```
if (!require(gridExtra)) {install.packages("gridExtra")
  library(gridExtra)
}
```

```
## Loading required package: gridExtra
```

```
if(!require(corrgram)){install.packages("corrgram")
  library("corrgram")
}
```

```
## Loading required package: corrgram

##
## Attaching package: 'corrgram'

## The following object is masked from 'package:lattice':
##
##     panel.fill
```

```r
if (!require(corrplot)) {install.packages("corrplot")
  library(corrplot)
}
```

```
## Loading required package: corrplot
```

```
## corrplot 0.92 loaded
```

```r
if(!require(pastecs)){install.packages("pastecs")
  library("pastecs")
}
```

```
## Loading required package: pastecs
```

```r
if(!require(vcd)){install.packages("vcd")
  library("vcd")
}
```

```
## Loading required package: vcd
```

```
## Loading required package: grid
```

```r
if(!require(HSAUR)){install.packages("HSAUR")
  library("HSAUR")
}
```

```
## Loading required package: HSAUR
```

```
## Loading required package: tools
```

```r
if(!require(rmarkdown)){install.packages("rmarkdown")
  library("rmarkdown")
}
```

```
## Loading required package: rmarkdown
```

```r
if(!require(ggplot2)){install.packages("ggplot2")
  library("ggplot2")
}
```

```
## Loading required package: ggplot2
```

```r
if(!require(polycor)){install.packages("polycor")
  library("polycor")
}
```

```
## Loading required package: polycor
```

```
if(!require(klaR)){install.packages("klaR")
  library("klaR")
}
```

```
## Loading required package: klaR
```

```
## Loading required package: MASS
```

```
if(!require(MASS)){install.packages("MASS")
  library("MASS")
}

if(!require(partykit)){install.packages("partykit")
  library("partykit")
}
```

```
## Loading required package: partykit
```

```
## Loading required package: libcoin
```

```
## Loading required package: mvtnorm
```

```
if(!require(nnet)){install.packages("nnet")
  library("nnet")
}
```

```
## Loading required package: nnet
```

```
if(!require(caret)){install.packages("caret")
  library(caret)
}
```

```
## Loading required package: caret
```

Reading file

```
# Reading File

df <- read.csv("Mail_Order_Delivery_Time.txt", header = TRUE, sep = ",")
head(df, 5)
```

```
##     DL  VN PG CS   ML DM HZ       CR  WT
## 1  8.1 324  5 13  313  C  N  Sup Del 216
## 2  8.4 135  2 13  830  I  N  Sup Del 160
## 3  8.6 391  3 12  304  C  N  Sup Del  25
## 4 11.3 245  6  7 1258  C  N  Sup Del  67
## 5  5.4 321  1  2  221  C  N Def Post  14
```

Preliminary Data Preparation

Renaming all variables with my initials appended.

```
# Appending initials to all variables in the data frame

df_VB <- df
colnames(df_VB) <- paste(colnames(df_VB), "VB", sep = "_")
head(df_VB, 5)
```

```
##   DL_VB VN_VB PG_VB CS_VB ML_VB DM_VB HZ_VB    CR_VB WT_VB
## 1   8.1   324     5    13   313     C     N  Sup Del   216
## 2   8.4   135     2    13   830     I     N  Sup Del   160
## 3   8.6   391     3    12   304     C     N  Sup Del    25
## 4  11.3   245     6     7  1258     C     N  Sup Del    67
## 5   5.4   321     1     2   221     C     N Def Post    14
```

As the dataset is same as that we used building Murltivariate Linear Rgression model, we will not do the regular descriptive analysis and outlier detection. We will just delete the observation with PG <0.

Deleting the observation with PG <0

```
df_VB <- df_VB[!df_VB$PG_VB < 0,]
```

Creating a new variable in the dataset called OT_VB which will have a value of 1 if DL 8.5 and 0 otherwise.

Creating a new variable OT_VB.

```
df_VB$OT_VB <- as.factor(ifelse(df_VB$DL_VB <= 8.5, 1,0))
```

Removing the first Column, DL.

```
df_VB <- df_VB[,-1]
```

Changing each character variable to factor variable

```
df_VB <- as.data.frame(unclass(df_VB), stringsAsFactors = TRUE)
```

```
str(df_VB)
```

```
## 'data.frame':    486 obs. of  9 variables:
##  $ VN_VB: int  324 135 391 245 321 397 390 252 355 159 ...
##  $ PG_VB: int  5 2 3 6 1 2 6 2 4 1 ...
##  $ CS_VB: int  13 13 12 7 2 8 13 8 2 12 ...
##  $ ML_VB: int  313 830 304 1258 221 1002 655 1367 675 888 ...
##  $ DM_VB: Factor w/ 2 levels "C","I": 1 2 1 1 1 2 1 2 1 1 ...
```

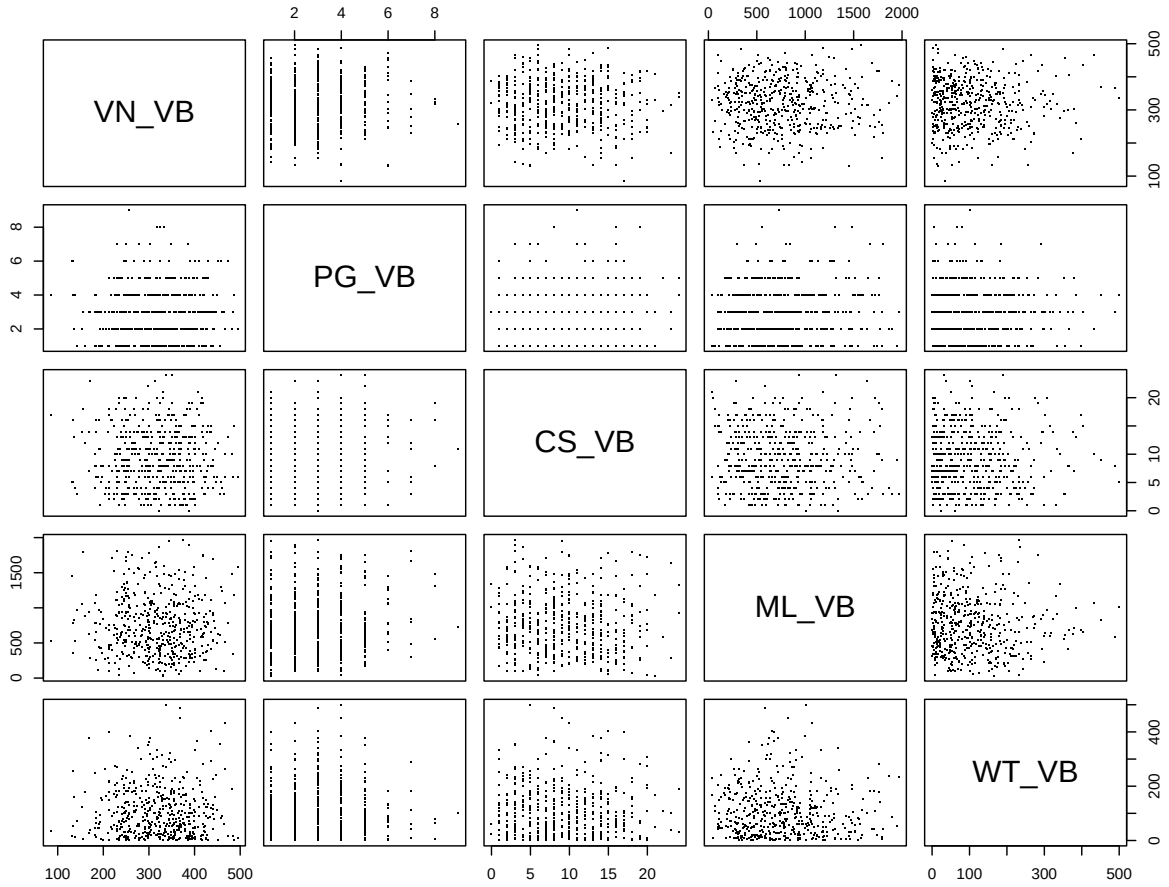
```
##  $ HZ_VB: Factor w/ 2 levels "H","N": 2 2 2 2 2 2 2 2 2 2 ...
##  $ CR_VB: Factor w/ 2 levels "Def Post","Sup Del": 2 2 2 2 1 2 2 2 2 2 ...
##  $ WT_VB: num  216 160 25 67 14 47 7 6 30 177 ...
##  $ OT_VB: Factor w/ 2 levels "0","1": 2 2 1 1 2 1 2 1 1 1 ...
```

Exploratory Analysis

Checking Correlations

Looking for correlations using pairs function.

```
pairs(df_VB[sapply(df_VB,is.numeric)], pch=46)
```



There doesn't seem to be any co-linear variables. But we will check it using hetcor function.

```
ht <- hetcor(df_VB)  #from polycor library
round(ht$correlations,2)
```

```
##       VN_VB PG_VB CS_VB ML_VB DM_VB HZ_VB CR_VB WT_VB OT_VB
## VN_VB  1.00  0.01 -0.02 -0.01  0.05  0.04 -0.07  0.00  0.00
## PG_VB  0.01  1.00  0.08  0.06  0.03  0.08 -0.04 -0.01 -0.50
```

```
## CS_VB -0.02  0.08  1.00 -0.03  0.10 -0.03  0.02 -0.02 -0.04
## ML_VB -0.01  0.06 -0.03  1.00 -0.06 -0.05  0.09 -0.04 -0.20
## DM_VB  0.05  0.03  0.10 -0.06  1.00  0.05  0.03  0.00 -0.11
## HZ_VB  0.04  0.08 -0.03 -0.05  0.05  1.00  0.14  0.11  0.19
## CR_VB -0.07 -0.04  0.02  0.09  0.03  0.14  1.00 -0.10 -0.39
## WT_VB  0.00 -0.01 -0.02 -0.04  0.00  0.11 -0.10  1.00  0.37
## OT_VB  0.00 -0.50 -0.04 -0.20 -0.11  0.19 -0.39  0.37  1.00
```

The correlation coefficient between OT_VB and PG_VB is -0.50, which indicates a moderate negative correlation between these two variables. Generally, a correlation coefficient of 0.7 or higher is considered to indicate high collinearity between two variables. So we say there are no co-linear variables.

Model Development

Spittling the data in train and test

```
# Choosing the sampling rate
sr <- 0.8

# Find the number of rows of data
n.row <- nrow(df_VB)

#Choose the rows for the traning sample

set.seed(5021)
training.rows <- sample(1:n.row, sr*n.row, replace=FALSE)

#Assign to the training sample

train_VB <- subset(df_VB[training.rows,])

# Assign the balance to the Test Sample

test_VB <- subset(df_VB[-c(training.rows),])
```

Creating a full model using all of the variables.

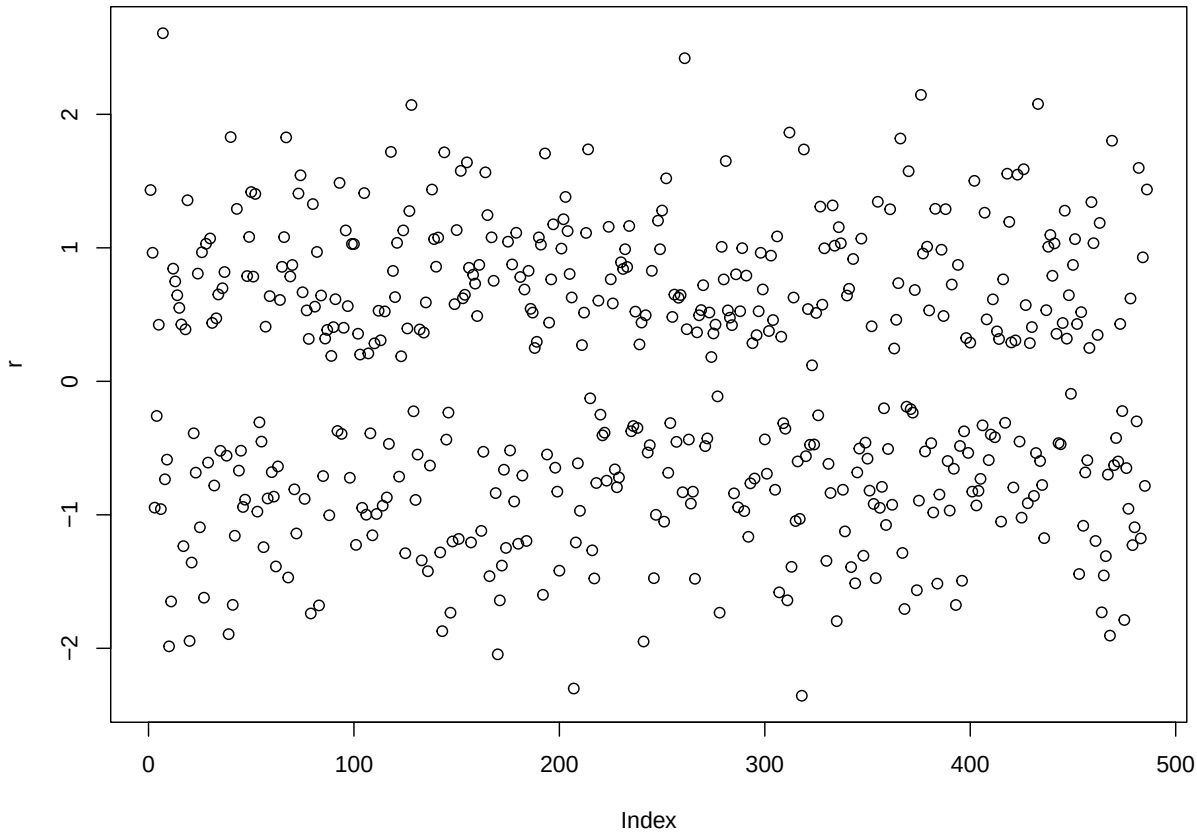
```
del_glm_VB = glm(OT_VB ~ . ,
                 family="binomial", data=df_VB, na.action=na.omit)
```

```
summary(del_glm_VB)
```

```
##
## Call:
## glm(formula = OT_VB ~ ., family = "binomial", data = df_VB, na.action = na.omit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.3553  -0.8172   0.2738   0.8007   2.6077
##
## Coefficients:
##                Estimate Std. Error z value      Pr(>|z|)
## (Intercept)   2.0856895  0.7039793   2.963       0.00305 **
## VN_VB         0.0001194  0.0015298   0.078       0.93777
## PG_VB        -0.7961927  0.0928126  -8.579       < 2e-16 ***
## CS_VB         0.0065113  0.0221318   0.294       0.76860
## ML_VB        -0.0007476  0.0002786  -2.683       0.00729 **
## DM_VBI       -0.4027813  0.2497650  -1.613       0.10682
## HZ_VBN        1.0726491  0.3309553   3.241       0.00119 **
## CR_VBSup Del -1.4471191  0.2427734  -5.961 0.00000000251 ***
## WT_VB         0.0083202  0.0013916   5.979 0.00000000224 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 672.92  on 485  degrees of freedom
## Residual deviance: 481.40  on 477  degrees of freedom
## AIC: 499.4
##
## Number of Fisher Scoring iterations: 5
```

```
r <- residuals(del_glm_VB)
plot(r)
```



MODEL SUMMARY

(1) Fisher iterations: The output indicates that the algorithm converged after 5 iterations.

(2) AIC: A lower AIC value indicates a better fit of the model to the data. In this case, the AIC value is 499.4, which is relatively low, suggesting a good fit of the model to the data.

(3) Residual Deviance: A lower residual deviance indicates a better fit of the model to the data. In this case, the residual deviance is 481.40, which is lower than the null deviance of 672.92, suggesting that the model explains a significant proportion of the variation in the data.

(4) Residual symmetry: Observing the residual plot, we see that residuals are symmetrically distributed around zero.

(5) z-values: Some variables such as PG_VB, HZ_VBN, and CR_VBSup Del have high absolute z-values, indicating that they are significant predictors of the outcome variable.

(6) Parameter Coefficients: Variables such as PG_VB, ML_VB, HZ_VBN, CR_VBSup Del, and WT_VB have statistically significant coefficients. They are likely to be important predictors of the outcome variable OT_VB.

Creating an additional model using backward selection.

```
stp_del_glm_VB <- step(del_glm_VB)
```

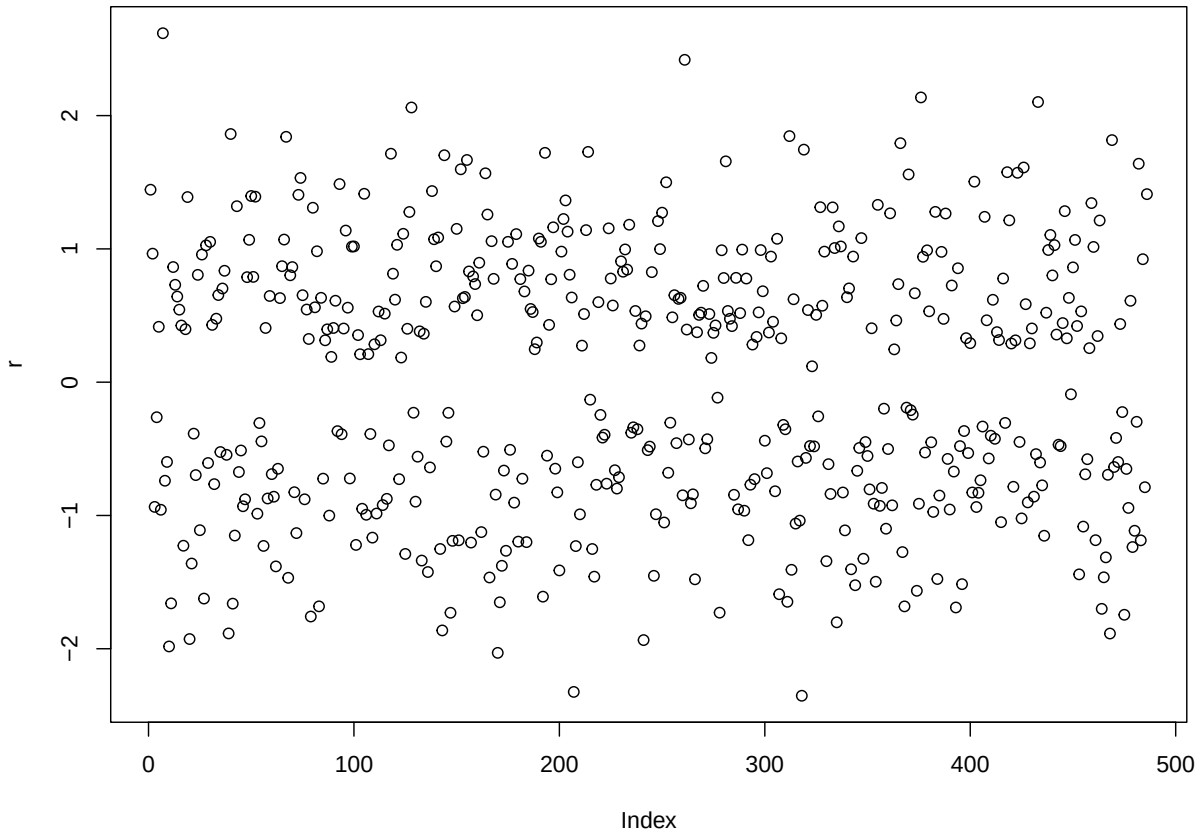
```
## Start:  AIC=499.4
## OT_VB ~ VN_VB + PG_VB + CS_VB + ML_VB + DM_VB + HZ_VB + CR_VB +
##     WT_VB
##
##         Df Deviance    AIC
## - VN_VB  1   481.40 497.40
## - CS_VB  1   481.48 497.48
## <none>       481.40 499.40
## - DM_VB  1   484.01 500.01
## - ML_VB  1   488.74 504.74
## - HZ_VB  1   492.32 508.32
## - CR_VB  1   520.59 536.59
## - WT_VB  1   525.15 541.15
## - PG_VB  1   582.88 598.88
##
## Step:  AIC=497.4
## OT_VB ~ PG_VB + CS_VB + ML_VB + DM_VB + HZ_VB + CR_VB + WT_VB
##
##         Df Deviance    AIC
## - CS_VB  1   481.49 495.49
## <none>       481.40 497.40
## - DM_VB  1   484.02 498.02
## - ML_VB  1   488.75 502.75
## - HZ_VB  1   492.33 506.33
## - CR_VB  1   520.68 534.68
## - WT_VB  1   525.16 539.16
## - PG_VB  1   582.93 596.93
##
## Step:  AIC=495.49
## OT_VB ~ PG_VB + ML_VB + DM_VB + HZ_VB + CR_VB + WT_VB
##
##         Df Deviance    AIC
## <none>       481.49 495.49
## - DM_VB  1   484.06 496.06
## - ML_VB  1   488.87 500.87
## - HZ_VB  1   492.34 504.34
## - CR_VB  1   520.71 532.71
## - WT_VB  1   525.19 537.19
## - PG_VB  1   583.17 595.17
```

```
summary(stp_del_glm_VB)
```

```
##
## Call:
## glm(formula = OT_VB ~ PG_VB + ML_VB + DM_VB + HZ_VB + CR_VB +
##     WT_VB, family = "binomial", data = df_VB, na.action = na.omit)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -2.3525  -0.8263   0.2746   0.8017   2.6182
##
## Coefficients:
##                Estimate Std. Error z value      Pr(>|z|)
## (Intercept)   2.1828335  0.4730099   4.615 0.00000393525 ***
## PG_VB        -0.7942781  0.0925623  -8.581       < 2e-16 ***
## ML_VB        -0.0007487  0.0002784  -2.690       0.00715 **
## DM_VBI       -0.3980684  0.2488869  -1.599       0.10973
## HZ_VBN        1.0666705  0.3300125   3.232       0.00123 **
## CR_VBSup Del -1.4459345  0.2424911  -5.963 0.00000000248 ***
## WT_VB         0.0083073  0.0013890   5.981 0.00000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 672.92  on 485  degrees of freedom
## Residual deviance: 481.49  on 479  degrees of freedom
## AIC: 495.49
##
## Number of Fisher Scoring iterations: 5
```

```
r <- residuals(stp_del_glm_VB)
plot(r)
```



MODEL SUMMARY

(1) Fisher iterations: The output indicates that the algorithm converged after 5 iterations.

(2) AIC: A lower AIC value indicates a better fit of the model to the data. In this case, the AIC value is 495.49, which is relatively low, suggesting a good fit of the model to the data.

(3) Residual Deviance: A lower residual deviance indicates a better fit of the model to the data. In this case, the residual deviance is 481.49, which is lower than the null deviance of 672.92, suggesting that the model explains a significant proportion of the variation in the data.

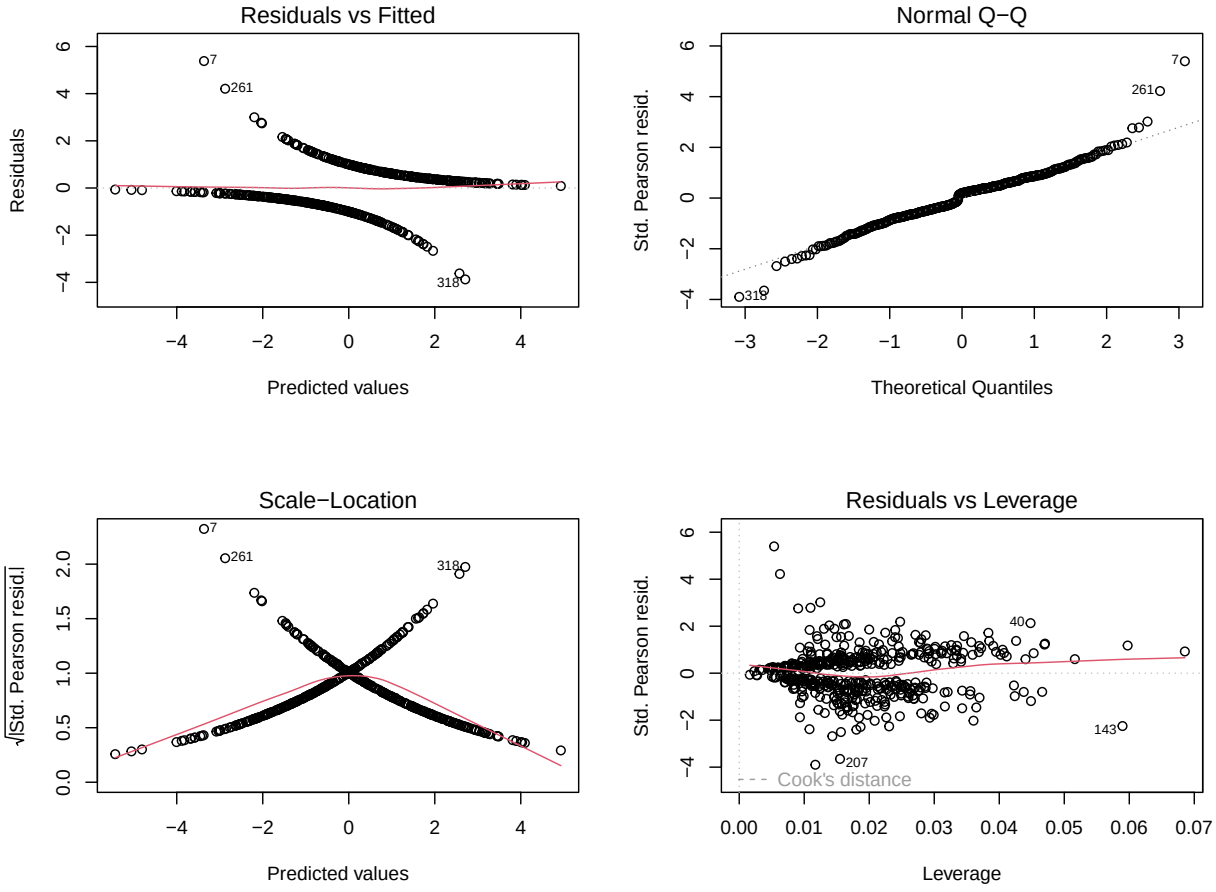
(4) Residual symmetry: Observing the residual plot, we see that residuals are symmetrically distributed around zero.

(5) z-values: Some variables such as PG_VB, HZ_VBN, and CR_VBSup Del have high absolute z-values, indicating that they are significant predictors of the outcome variable.

(6) Parameter Coefficients: Variables such as PG_VB, ML_VB, HZ_VBN, CR_VBSup Del, and WT_VB have statistically significant coefficients. They are likely to be important predictors of the outcome variable OT_VB.

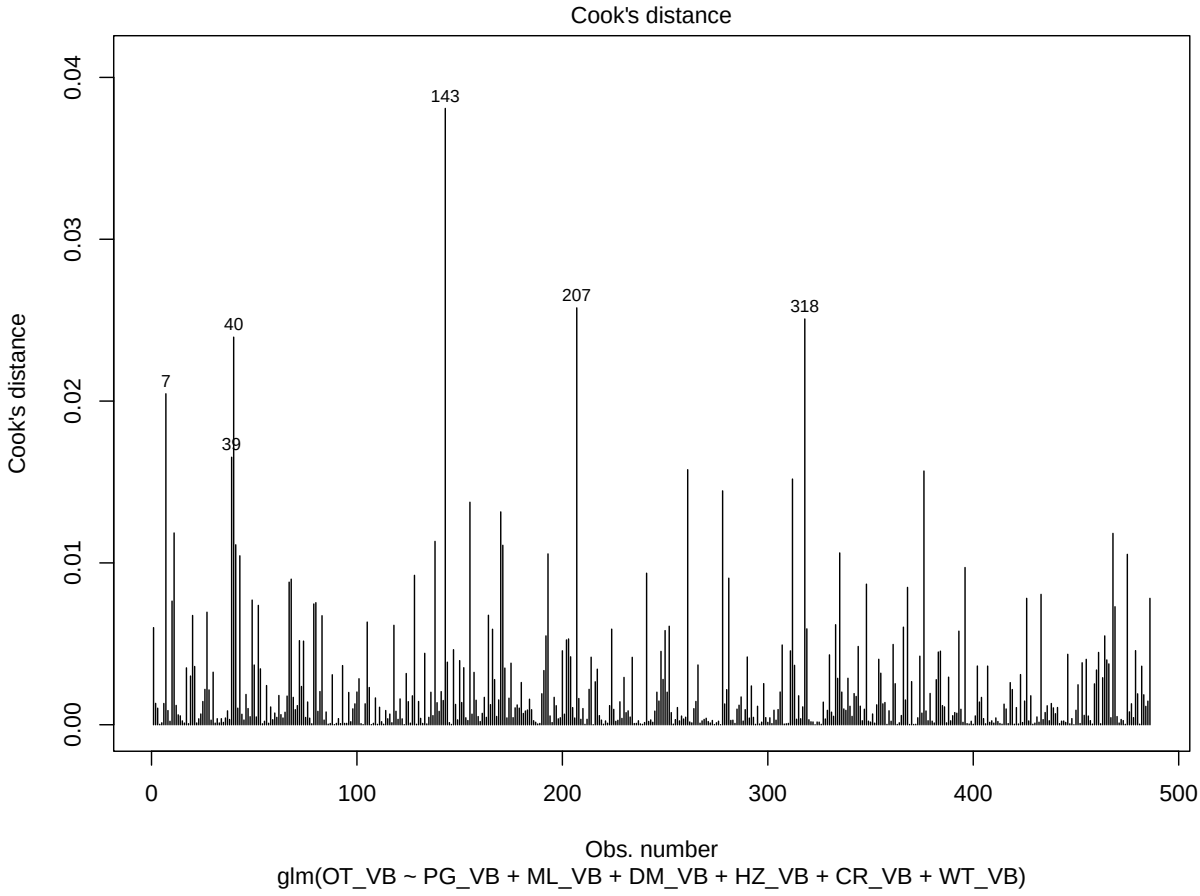
Analyzing the output for any significantly influential datapoints.

```
par(mfrow = c(2, 2))
plot(del_glm_VB)
```




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

```
plot(stp_del_glm_VB,which=4, id.n=6)
```



Some of the influential points include :7, 39, 40, 143, 208, 319. These indicate that specific observations in our dataset that have a significant impact on the model fit. These observations have high leverage and/or large standardized residuals, which means they have a large influence on the estimated regression coefficients and model fit.

Recommending the model that should be selected.

The AIC of Model 2 (495.49) is lower than that of Model 1 (499.4) The residual deviance of Model 1 and 2 are almost same. The higher z values indicate a more significant predictor varaibles. The residuals of both the models are symmetrically distributed around zero. So we conclude that Model 2 is slightly better than Model 1.

Creating more classifiers

1. Logistic Regression – stepwise

Using the step option in the glm function to fit the model.

```
# Fitting the logistic regression model with stepwise selection
start_time <- Sys.time() # starting the timer
logit_VB <- glm(OT_VB ~ ., data = train_VB, family = "binomial", na.action=na.omit)
logit_stepwise_VB <- step(logit_VB, direction = "both", trace=FALSE)
end_time <- Sys.time() # ending the timer
```

Summarizing the results in Confusion Matrices for both train and test sets.

```
# Predicting the classes for the training set and the test set
train_pred <- predict(logit_stepwise_VB, train_VB, type = "response")
test_pred <- predict(logit_stepwise_VB, test_VB, type = "response")

# Create confusion matrices for both the training set and the test set
train_cm <- table(train_VB$OT_VB, train_pred > 0.5)
test_cm <- table(test_VB$OT_VB, test_pred > 0.5)


# Calculate the time it took to fit the model
fit_time <- end_time - start_time

cat("Train Confusion Matrix:\n")
```

```
## Train Confusion Matrix:
```

```
print(train_cm)
```

```
##
##     FALSE TRUE
##   0   130   51
##   1    43  164
```

```
cat("Test Confusion Matrix:\n")
```

```
## Test Confusion Matrix:
```

```
print(test_cm)
```

```
##
##     FALSE TRUE
##   0    33   19
##   1    13   33
```

```
cat("\nModel fit time (in seconds):", fit_time, "\n")
```

```
##
## Model fit time (in seconds): 0.0539391
```

Train set:

Accuracy: 0.757
Misclassification Rate: 0.242
Precision: 0.763
Sensitivity: 0.792
Specificity: 0.718 Balanced Accuracy: 0.755

Test set:

Accuracy: 0.673
Misclassification Rate: 0.326
Precision: 0.635
Sensitivity: 0.717
Specificity: 0.635
Balanced Accuracy: 0.676

2. Naïve-Bayes Classification

```
# Fitting the Naïve-Bayes Classification model
start_time <- Sys.time() # starting the timer
nb_VB <- NaiveBayes(OT_VB ~ ., data = train_VB, na.action=na.omit)
end_time <- Sys.time() # ending the timer
```

Summarizing the results in Confusion Matrices for both train and test sets.

```
# Predicting the classes for the training set and the test set
train_pred <- predict(nb_VB, train_VB)
test_pred <- predict(nb_VB, test_VB)

# Creating confusion matrices for both the training set and the test set
train_cm <- table(Actual=train_VB$OT_VB, Predicted=train_pred$class)
test_cm <- table(Actual=test_VB$OT_VB, Predicted=test_pred$class)


# Calculating the time it took to fit the model
fit_time <- end_time - start_time

cat("Train Confusion Matrix:\n\n")
```

```
## Train Confusion Matrix:
```

```
print(train_cm)
```

```
##       Predicted
## Actual   0   1
##      0 129  52
##      1  41 166
```

```
cat("\n\nTest Confusion Matrix:\n\n")
```

```
##
##
## Test Confusion Matrix:
```

```
print(test_cm)
```

```
##       Predicted
## Actual  0  1
##      0 35 17
##      1 13 33
```

```
cat("\nModel fit time (in seconds):", fit_time, "\n")
```

```
##
## Model fit time (in seconds): 0.002997875
```

Train set:

Accuracy: 0.76
Misclassification Rate: 0.24
Precision: 0.761
Sensitivity: 0.802
Specificity: 0.713
Balanced Accuracy: 0.757

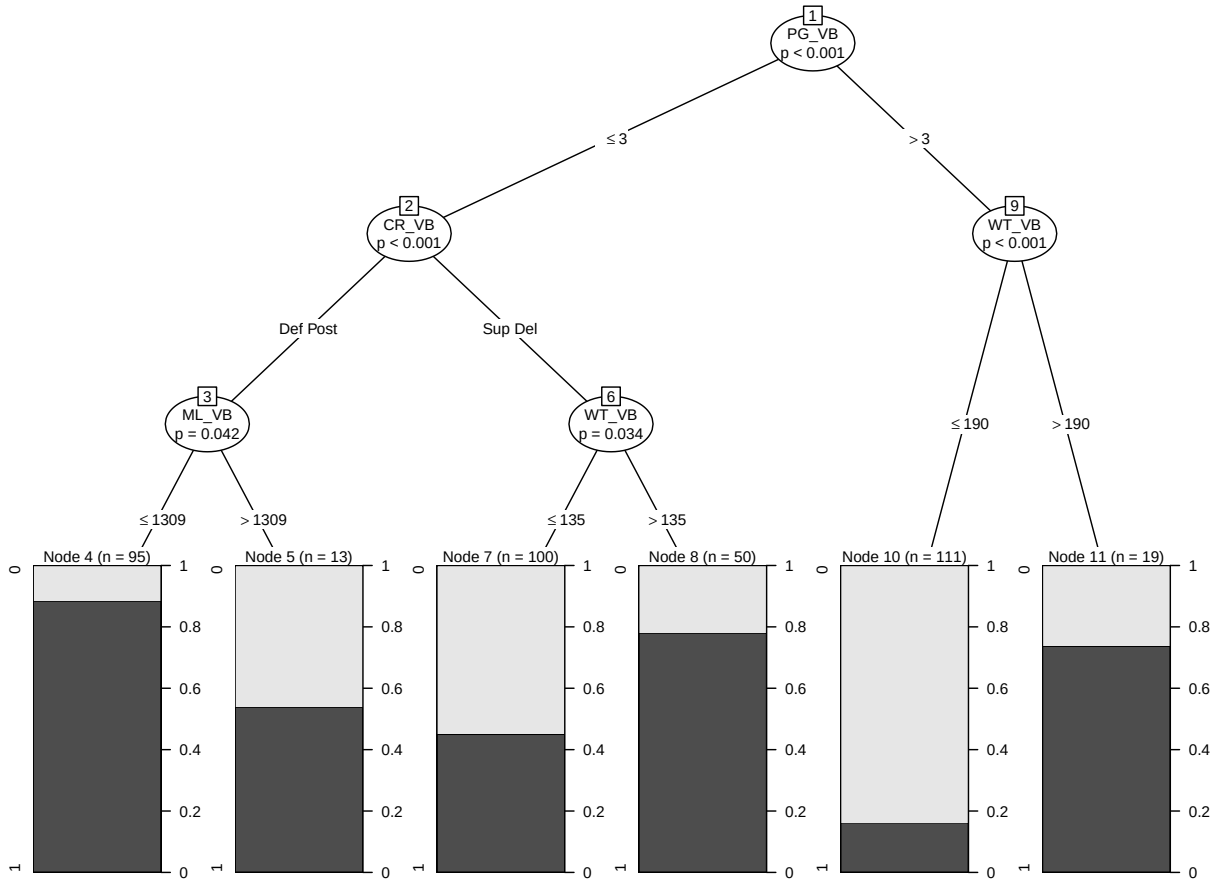
Test set:

Accuracy: 0.693
Misclassification Rate: 0.306
Precision: 0.660
Sensitivity: 0.717
Specificity: 0.673
Balanced Accuracy: 0.695

3. Recursive Partitioning Analysis

```
# Fitting the model using Recursive Partitioning Analysis
start_time <- Sys.time() # start the timer
rpa_VB <- ctree(OT_VB ~ ., data = train_VB)
end_time <- Sys.time() # end the timer
```

```
plot(rpa_VB, gp=gpar(fontsize=8))
```



Summarizing the results in Confusion Matrices for both train and test sets.

```
# Predicting the classes for the training set and the test set
train_pred <- predict(rpa_VB, train_VB)
test_pred <- predict(rpa_VB, test_VB)

# Creating confusion matrices for both the training set and the test set
train_cm <- table(Actual=train_VB$OT_VB, Predicted=train_pred)
test_cm <- table(Actual=test_VB$OT_VB, Predicted=test_pred)


# Calculating the time it took to fit the model
fit_time <- end_time - start_time

cat("Train Confusion Matrix:\n\n")
```

```
## Train Confusion Matrix:
```

```
print(train_cm)
```

```
##       Predicted
## Actual   0   1
##      0 148  33
##      1  63 144
```

```
cat("\n\nTest Confusion Matrix:\n\n")
```

```
##
##
## Test Confusion Matrix:
```

```
print(test_cm)
```

```
##       Predicted
## Actual  0  1
##      0 38 14
##      1 18 28
```

```
cat("\nModel fit time (in seconds):", fit_time, "\n")
```

```
##
## Model fit time (in seconds): 0.0240612
```

Train set:

Accuracy: 0.752
Misclassification Rate: 0.247
Precision: 0.813
Sensitivity: 0.696
Specificity: 0.818
Balanced Accuracy: 0.757

Test set:

Accuracy: 0.673
Misclassification Rate: 0.326
Precision: 0.667
Sensitivity: 0.609
Specificity: 0.731
Balanced Accuracy: 0.670

4. Neural Network classification model

```
# Fit the model using Neural Network
start_time <- Sys.time() # start the timer
set.seed(8430)
fit <- nnet(OT_VB ~ .,
          data=train_VB,
```

```
          size=3,
          rang=0.1,
          maxit=1200,
          trace=FALSE)
end_time <- Sys.time() # end the timer
```

Summarizing the results in Confusion Matrices for both train and test sets.

```
# Predicting the classes for the training set and the test set
train_pred <- predict(fit, newdata=train_VB, type="class")
test_pred <- predict(fit, newdata=test_VB, type="class")

# Creating confusion matrices for both the training set and the test set
train_cm <- table(Actual=train_VB$OT_VB, Predicted=train_pred)
test_cm <- table(Actual=test_VB$OT_VB, Predicted=test_pred)


# Calculating the time it took to fit the model
fit_time <- end_time - start_time

cat("Train Confusion Matrix:\n\n")
```

```
## Train Confusion Matrix:
```

```
print(train_cm)
```

```
##       Predicted
## Actual   0   1
##      0 134  47
##      1  48 159
```

```
cat("\n\nTest Confusion Matrix:\n\n")
```

```
##
##
## Test Confusion Matrix:
```

```
print(test_cm)
```

```
##       Predicted
## Actual  0  1
##      0 35 17
##      1 11 35
```

```
cat("\nModel fit time (in seconds):", fit_time, "\n")
```

```
##
## Model fit time (in seconds): 0.05269098
```

Train set:

Accuracy: 0.755
Misclassification Rate: 0.245
Precision: 0.736
Sensitivity: 0.74
Specificity: 0.768
Balanced Accuracy: 0.754

Test set:

Accuracy: 0.714
Misclassification Rate: 0.286
Precision: 0.673
Sensitivity: 0.761
Specificity: 0.673
Balanced Accuracy: 0.714

Comparing All Classifiers

Comparing Accuracy

To determine which classifier is most accurate, we will compare the test accuracy of each model. Looking at the test confusion matrix, the accuracy of models are: Model 1 Logistic Regression - stepwise = 67%. Model 2 Naïve-Bayes Classification has an accuracy of = 69%. Model 3 Recursive Partitioning Analysis = 67%. Model 4 Neural Network = 71%.

Therefore, Model 4 (Neural Network) is the most accurate classifier.

Comparing consistancy

To determine which classifier is most consistent, we need to compare the train accuracy and test accuracy of each model. If the difference between the train accuracy and test accuracy is small, it indicates that the model is not overfitting the training data and is generalizing well to new data.

Looking at the train and test confusion matrices for each model, we can see that Model 4 (Neural Network) has the smallest difference between train accuracy (75%) and test accuracy (71%). Therefore, Model 4 is the most consistent classifier.

Comparing processing speed

To determine which classifier is most suitable when processing speed is most important, we need to compare the model fit time for each model. Looking at the model fit times provided, we can see that Model 2 (Naïve-Bayes Classification) has the shortest model fit time of 0.03721499 seconds. Therefore, Model 2 is the most suitable classifier when processing speed is most important.

Overall best classifier

To determine the best overall classifier, we need to consider all the factors discussed above: accuracy, consistency, processing speed, and minimizing false positives. Based on these factors, we can conclude that Model 4 (Neural Network) is the best overall classifier. It has the highest accuracy and consistency. Although the processesing time is high, but looking at our dataset,the processing time is not relatively very high. While

it does not minimizes false positives better than the other models, its accuracy is still competitive with the other models.