Classification

Divyang Vinubhai Hirpara

07/04/2023

1. Preliminary and Exploratory

1. Rename all variables with your initials appended

```
data_DVH <- read.table("PROG8430_Assign_dataset.txt",sep=",",header = TRUE)

#Rename Variables of column name
#Append data_DVH initials to column names
colnames(data_DVH) <- paste(colnames(data_DVH), "DVH", sep = "_")
head(data_DVH)
```

```
##   DL_DVH VN_DVH PG_DVH CS_DVH ML_DVH DM_DVH HZ_DVH   CR_DVH WT_DVH
## 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
## 6    9.4    397      2      8   1002      I      N  Sup Del     47
```

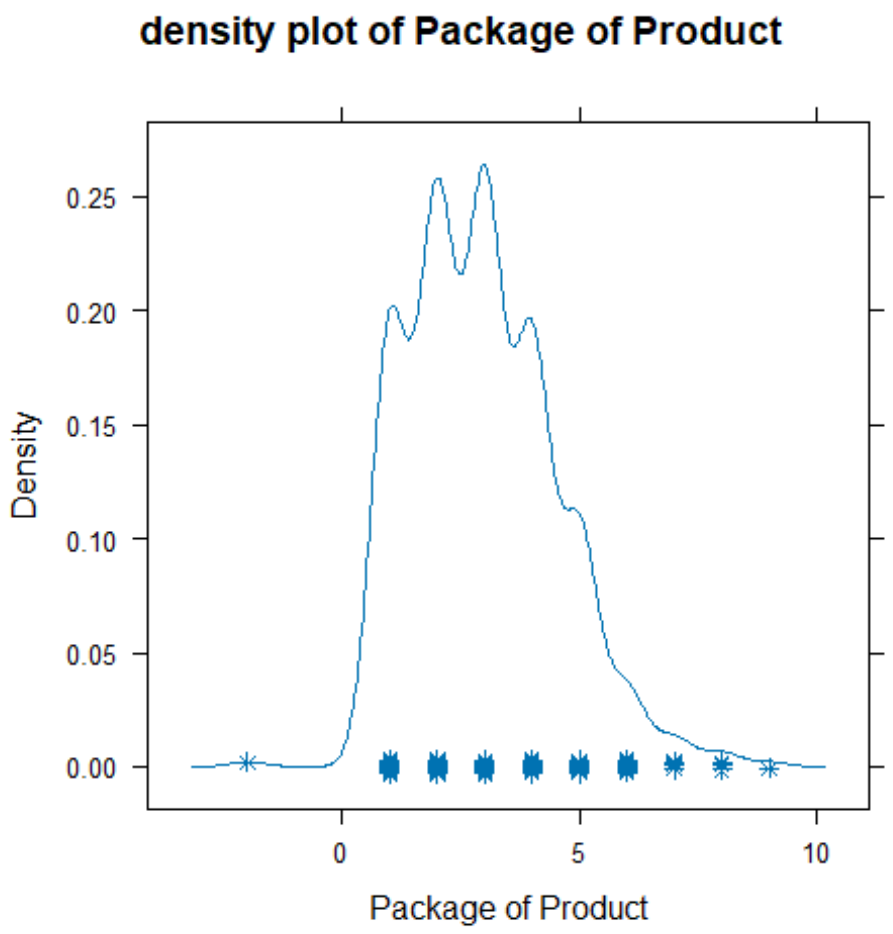
Interpretation

Every column are replaced with initials.
DL –> DL_DVH
VN –> VN_DVH
PG –> PG_DVH
CS –> CS_DVH
ML –> ML_DVH
DM –> DM_DH
HZ –> HZ_DH
CR –> CR_DVH
WT –> WT_DVH

2. Delete the observation with PG <0 using the following code:

Before deleting outlier from package of Product

```
densityplot( ~ data_DVH$PG_DVH, pch=8,main="density plot of Package of
Product",xlab="Package of Product")
```



NOTE: Deleting PG_DVH < 0, based on the descriptive analysis done previously.

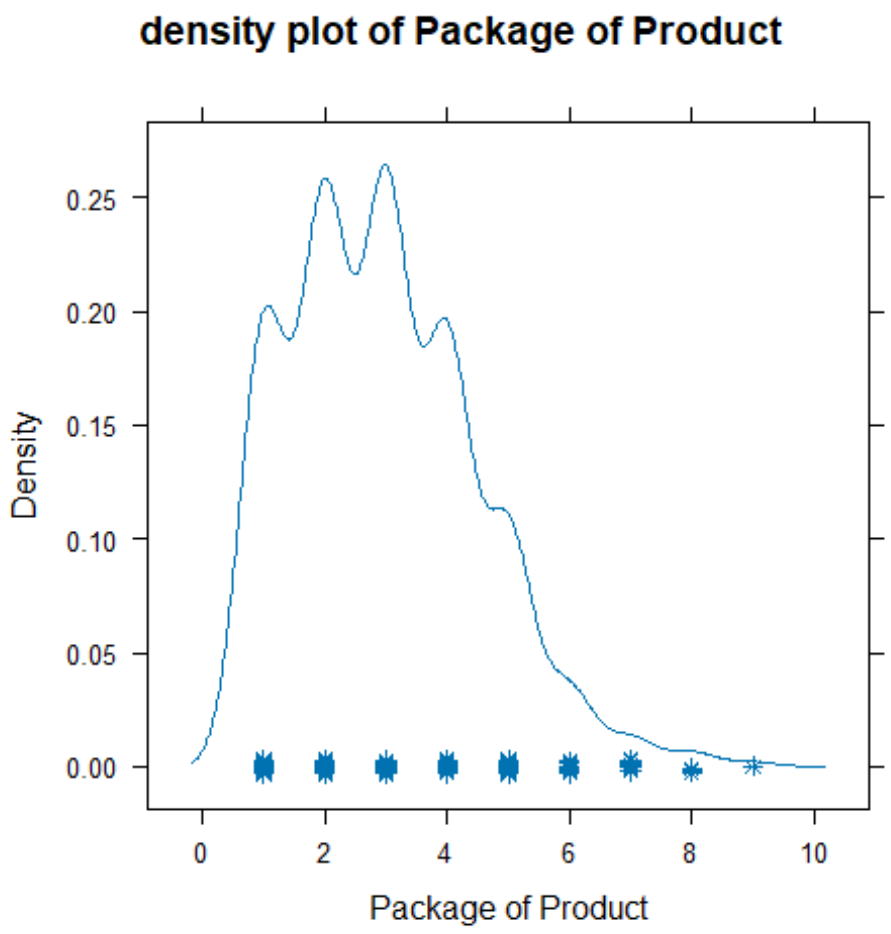
```
data_DVH <- data_DVH[!data_DVH$PG < 0,]
```

Interpretation

To remove any negative values from a given dataset, I am performing a filtering operation by creating a same data variable that contains only the values greater than or equal to zero. This approach effectively excludes any values less than zero, resulting in a refined dataset that only comprises of values that are zero or greater.

After deleting outlier from package of Product

```
densityplot( ~ data_DVH$PG_DVH, pch=8,main="density plot of Package of
Product",xlab="Package of Product")
```



Interpretation
As you can see from above density plot, outlier is successfully removed from Package of Product column.

3. Create a new variable in the dataset called OT_[Intials] which will have a value of 1 if DL ≤ 8.5 and 0 otherwise.

```
data_DVH$OT_DVH <- as.factor(ifelse(data_DVH$DL_DVH <= 8.5, 1,0))

data_DVH <- data_DVH[, !colnames(data_DVH) %in% c("DL_DVH")]
```

Interpretation
Here, as.factor function is used to convert new column into categorical variable with two level(1,0).

Above first line of code creates a new categorical variable “OT_DVH” in a data frame, where 1 is assigned if the corresponding value in the “DL_DVH” column is less than or equal to 8.5, and 0 is assigned if it is greater than 8.5.

Second line of code remove the ‘DL_DVH’ column from table.

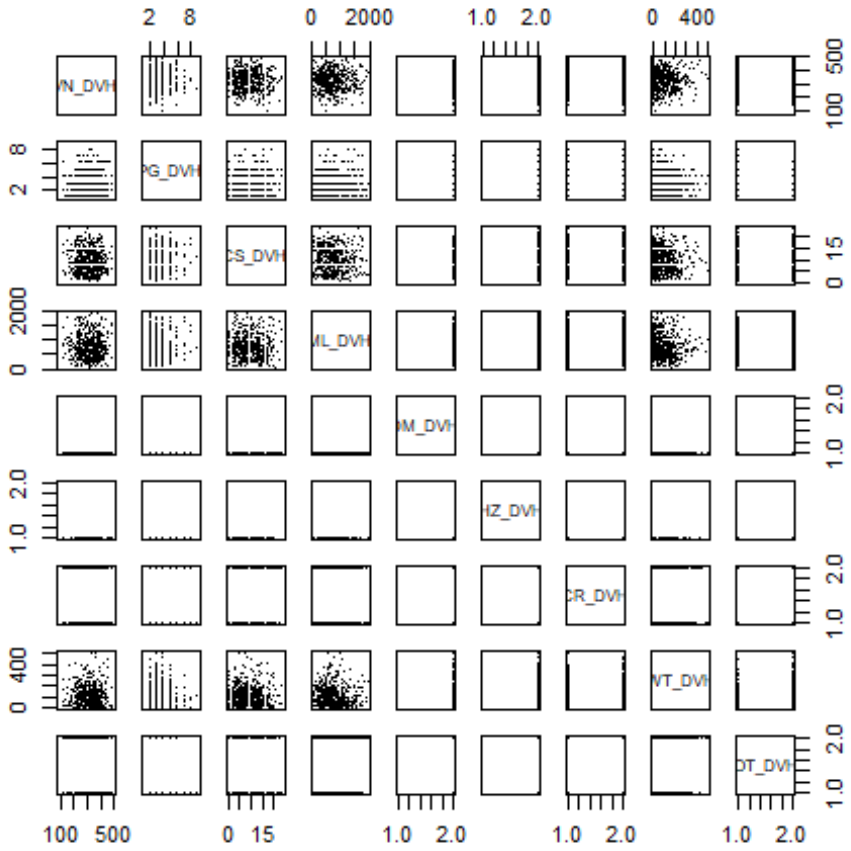
2. Exploratory Analysis

1. Correlations: Create correlations (as demonstrated) and comment on what you see. Are there co-linear variables?

```
# Change all character variable to factor variables.
data_DVH <- as.data.frame(unclass(data_DVH), stringsAsFactors = TRUE)



pairs(data_DVH, pch=46)
```




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

##         VN_DVH PG_DVH CS_DVH ML_DVH DM_DVH HZ_DVH CR_DVH WT_DVH OT_DVH
## VN_DVH    1.00   0.01  -0.02  -0.01   0.05   0.04  -0.07   0.00   0.00
## PG_DVH    0.01   1.00   0.08   0.06   0.03   0.08  -0.04  -0.01  -0.50
## CS_DVH   -0.02   0.08   1.00  -0.03   0.10  -0.03   0.02  -0.02  -0.04
## ML_DVH   -0.01   0.06  -0.03   1.00  -0.06  -0.05   0.09  -0.04  -0.20
## DM_DVH    0.05   0.03   0.10  -0.06   1.00   0.05   0.03   0.00  -0.11
## HZ_DVH    0.04   0.08  -0.03  -0.05   0.05   1.00   0.14   0.11   0.19
```

```
## CR_DVH  -0.07  -0.04   0.02   0.09   0.03   0.14   1.00  -0.10  -0.39
## WT_DVH   0.00  -0.01  -0.02  -0.04   0.00   0.11  -0.10   1.00   0.37
## OT_DVH   0.00  -0.50  -0.04  -0.20  -0.11   0.19  -0.39   0.37   1.00
```

Interpretation

Before, performing a correlation between variable, I have converted all character variables into factor variables here.

Looking at the correlation matrix, there doesn't seem to be any evidence of perfect collinearity between variables.

However, there is a moderate correlations between one pair of variables, which is PG_DVH and OT_DVH (-0.50).

Overall, there does not seem to be any evidence of collinearity between the variables in the dataset.

3. Model Developement

Splitting data_DVH into train and test variable.

```
# Set the seed using the last four digits of your student number
set.seed(6337)

# Split the dataframe into a training and test set using an 80/20 split
train_index_DVH <- sample(1:nrow(data_DVH), floor(0.8*nrow(data_DVH)),
replace = FALSE)
train_data_DVH <- data_DVH[train_index_DVH, ]
test_data_DVH <- data_DVH[-train_index_DVH, ]
```

Interpretation

The function set.seed(6337) is used to set a fixed seed value for the random number generator, which ensures that the random split is reproducible.

The sample() function is then used to randomly sample 80% of the rows of data_DVH for the training set.

The floor() function is used to round down to the nearest integer value, as the sample() function requires the sample size to be an integer.

The replace = FALSE argument specifies that sampling should be done without replacement.

1. A full model using all of the variables.

NOTE: I am calculating a time for full model here, because it will be needed for upcoming task in part B.

```
# start time for full model
start_full_model_time = Sys.time()
glm_fit_DVH <- glm(OT_DVH ~ . ,
data=train_data_DVH,family='binomial',na.action=na.omit)
#end time for full model
end_full_model_time = Sys.time()
```

```r
# total time taken to fit full model
glm_full_model_time = end_full_model_time - start_full_model_time
glm_full_model_time
```

```
## Time difference of 0.011935 secs
```

```r
summary(glm_fit_DVH)
```

```
##
## Call:
## glm(formula = OT_DVH ~ ., family = "binomial", data = train_data_DVH,
##     na.action = na.omit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3249  -0.8311   0.2664   0.8152   2.5945
##
## Coefficients:
##                   Estimate  Std. Error z value          Pr(>|z|)
## (Intercept)    2.02339548  0.77076991   2.625           0.00866 **
## VN_DVH         0.00001209  0.00165471   0.007           0.99417
## PG_DVH        -0.78449493  0.10270523  -7.638 0.000000000000022 ***
## CS_DVH         0.00459032  0.02464506   0.186           0.85224
## ML_DVH        -0.00066452  0.00031240  -2.127           0.03341 *
## DM_DVHI       -0.34160552  0.27459566  -1.244           0.21349
## HZ_DVHN        1.11551554  0.38543405   2.894           0.00380 **
## CR_DVHSup Del -1.44679727  0.27326084  -5.295 0.000000119300114 ***
## WT_DVH         0.00786818  0.00155259   5.068 0.000000402507269 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 537.51  on 387  degrees of freedom
## Residual deviance: 390.74  on 379  degrees of freedom
## AIC: 408.74
##
## Number of Fisher Scoring iterations: 5
```

2. An additional model using backward selection.

```r
back_model_DVH <- step(glm_fit_DVH,
direction="backward",details=TRUE,trace=FALSE)
summary(back_model_DVH)
```

```
##
## Call:
## glm(formula = OT_DVH ~ PG_DVH + ML_DVH + HZ_DVH + CR_DVH + WT_DVH,
##     family = "binomial", data = train_data_DVH, na.action = na.omit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.2806  -0.8253   0.2812   0.8271   2.6357
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    1.9659637  0.5321245   3.695  0.00022 ***
## PG_DVH        -0.7822152  0.1019491  -7.673 1.69e-14 ***
## ML_DVH        -0.0006353  0.0003097  -2.051  0.04026 *
## HZ_DVHN        1.0877408  0.3838175   2.834  0.00460 **
## CR_DVHSup Del -1.4407850  0.2721585  -5.294 1.20e-07 ***
## WT_DVH         0.0078059  0.0015486   5.041 4.64e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 537.51  on 387  degrees of freedom
## Residual deviance: 392.33  on 382  degrees of freedom
## AIC: 404.33
##
## Number of Fisher Scoring iterations: 5
```

For each model, interpret and comment on the main measures we discussed in class:

Fisher iterations : In both the full model and the backward selection model, the algorithm performed 5 iterations. So, we can say that it is conversed.

AIC: From above result, the backward selection model has a lower AIC (404.33) than the full model (408.74), indicating that the backward selection model is a better fit for the data.

Residual Deviance: Residual Deviance is less than the Null Deviance, which are 390.74/537.51 for Full Model and 392.33/537.51 for Backword Selection Model. Lower residual deviance is considered as better.

Residual symmetry: In both the full model(median: 0.2664) and the backward selection model(median: 0.2812), we can say from both models, that values are near to zero. Hence, residual symmetry is passed for both models.

z-values: 5/8 variables passed the z-test in the Full Model, and all 5/5 variables passed the z-test in the Backward Selection Model.

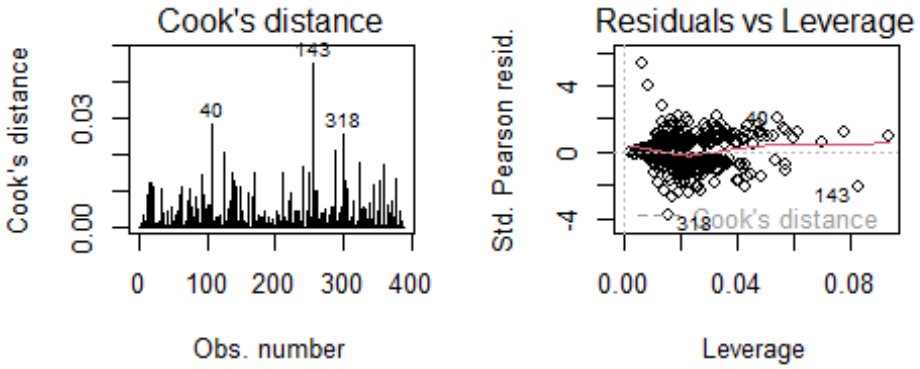
Parameter Co-Efficient: In the full model, except CS_DVH all passed, which is 4/5. whereas, 3/3 passed in backward model.

3. As demonstrated in class, analyze the output for any significantly influential datapoints. Are there any?

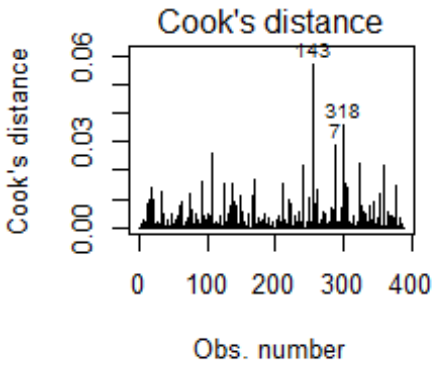
```
#glm_fit_DVH
# Full Model
par(mfrow = c(2, 2))
plot(glm_fit_DVH, which=4)
plot(glm_fit_DVH, which=5)
```

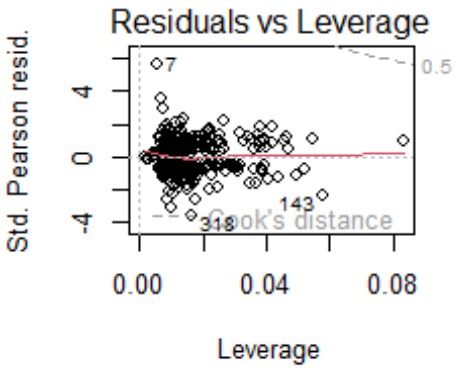
```r
#back_model_DVH

# Backword Model
par(mfrow=c(2,2))
```




```r
plot(back_model_DVH, which=4)
plot(back_model_DVH, which=5)
```





Interpretation
From above result, No significant influential datapoints from the given output.

4. Based on your preceding analysis, recommend which model should be selected and explain why.

Answer: The backward selection model is a better choice for selection process on new data. Because, All parameter coefficient passed(3/3). AIC for backword selection is lower than full model. Additionally, the backward selection model has fewer predictors(5/5 ~ p-value at z test passed), which can help prevent overfitting and simplify the model.

Part B:

Logistic Regression – stepwise

1. use the step option in the glm function to fit the model (using stepwise selection).

```
# Start time for step wise model
start_time_DVH <- Sys.time()
```

```
step_glm_DVH <- step(glm_fit_DVH, trace=FALSE)

# End time for step wise model
end_time_DVH <- Sys.time()

# Total time taken for fitting of step wise model
GLM_Time_DVH <- end_time_DVH - start_time_DVH

summary(step_glm_DVH)

##
## Call:
## glm(formula = OT_DVH ~ PG_DVH + ML_DVH + HZ_DVH + CR_DVH + WT_DVH,
##     family = "binomial", data = train_data_DVH, na.action = na.omit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2806  -0.8253   0.2812   0.8271   2.6357
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    1.9659637  0.5321245   3.695  0.00022 ***
## PG_DVH        -0.7822152  0.1019491  -7.673 1.69e-14 ***
## ML_DVH        -0.0006353  0.0003097  -2.051  0.04026 *
## HZ_DVHN        1.0877408  0.3838175   2.834  0.00460 **
## CR_DVHSup Del -1.4407850  0.2721585  -5.294 1.20e-07 ***
## WT_DVH         0.0078059  0.0015486   5.041 4.64e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 537.51  on 387  degrees of freedom
## Residual deviance: 392.33  on 382  degrees of freedom
## AIC: 404.33
##
## Number of Fisher Scoring iterations: 5
```

Interpretation

Analysis for step wise Logistic regression model.
Fisher Scoring : 5
AIC : 404.33
Res. Deviance : 392.33
Balanced Mean 0 at Residuals : pass
p-value at z test : pass 5/5

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

```
# Predict on train set
resp_glm_train_DVH <- predict(step_glm_DVH, newdata=train_data_DVH,
```

```
type="response")
Class_glm_train_DVH <- ifelse(resp_glm_train_DVH > 0.5,1,0)

# Generate confusion matrix for train set
CF_GLM_train_DVH <- table(train_data_DVH$OT_DVH, Class_glm_train_DVH,
                dnn=list("Actual DL","Predicted") )



# Predict on test set
resp_glm_test_DVH <- predict(step_glm_DVH, newdata=test_data_DVH,
type="response")
Class_glm_test_DVH <- ifelse(resp_glm_test_DVH > 0.5,1,0)

# Generate confusion matrix for train set
CF_GLM_test_DVH <- table(test_data_DVH$OT_DVH, Class_glm_test_DVH,
                dnn=list("Actual DL","Predicted") )


# Confusion Matrices of for Train set
CF_GLM_train_DVH

##          Predicted
## Actual DL   0   1
##         0 140  48
##         1  45 155

# Confusion Matrices of for Test set
CF_GLM_test_DVH

##          Predicted
## Actual DL  0  1
##         0 33 12
##         1 10 43
```

Interpretation

It shows the number of correct and incorrect predictions made by the model on the training and test data.

`For the training data:`

- True negative (TN) = 140
- False positive (FP) = 48
- False negative (FN) = 45
- True positive (TP) = 155

`For the test data:`

- True negative (TN) = 33
- False positive (FP) = 12
- False negative (FN) = 10
- True positive (TP) = 43

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary

```
# Time calculated only for step wise selection process
GLM_Time_DVH

## Time difference of 0.064533 secs

# Time calculated for step wise selection and full model
Total_GLM_Time_DVH = glm_full_model_time + GLM_Time_DVH
Total_GLM_Time_DVH

## Time difference of 0.07646799 secs
```

Interpretation
Time taken to fit only step wise process is 0.07787609 secs.

Total Time taken to fit Step wise model is 0.09813023 secs.

Note: Full Model time is calculated in task 3 (model development) in part A.

2. Naïve-Bayes Classification

1. Use all the variables in the dataset to fit a Naïve-Bayesian classification model.

```
start_time_NB_DVH <- Sys.time()

NB_mod_DVH <- NaiveBayes(OT_DVH ~ . , data = train_data_DVH,
na.action=na.omit)

end_time_NB_DVH <- Sys.time()

NB_Time_DVH <- end_time_NB_DVH - start_time_NB_DVH


NB_Time_DVH

## Time difference of 0.006281137 secs
```

Interpretation
In this Naive-Bayes Classification model, I have added start time and end time to find time taken to fit model.

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

```
# Predict on train set
pred_NB_train_DVH <- predict(NB_mod_DVH,newdata=train_data_DVH)

# Generate confusion matrix for train set
CF_NB_train_DVH <- table(Actual=train_data_DVH$OT_DVH,
Predicted=pred_NB_train_DVH$class)



# Predict on Test set
pred_NB_test_DVH <- predict(NB_mod_DVH,newdata=test_data_DVH)

# Generate confusion matrix for Test set
CF_NB_test_DVH <- table(Actual=test_data_DVH$OT_DVH,
Predicted=pred_NB_test_DVH$class)




# Confusion Matrices of for Train set
CF_NB_train_DVH
```

```
##       Predicted
## Actual   0   1
##      0 135  53
##      1  50 150
```

```
# Confusion Matrices of for Test set
CF_NB_test_DVH
```

```
##       Predicted
## Actual  0  1
##      0 33 12
##      1  9 44
```

Interpretation

It shows the number of correct and incorrect predictions made by the model on the training and test data.

For the training data:

- True negative (TN) = 135
- False positive (FP) = 53
- False negative (FN) = 50
- True positive (TP) = 150

For the test data:

- True negative (TN) = 33
- False positive (FP) = 12
- False negative (FN) = 9
- True positive (TP) = 44

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary

```
# Time calculated to fit Naïve-Bayes Classification
NB_Time_DVH
```

```
## Time difference of 0.006281137 secs
```

Interpretation

Total Time taken to fit Naive-Bayes Classification model is 0.0392981 secs.

3. Recursive Partitioning Analysis

1. Use all the variables in the dataset to fit an recursive partitioning classification model.

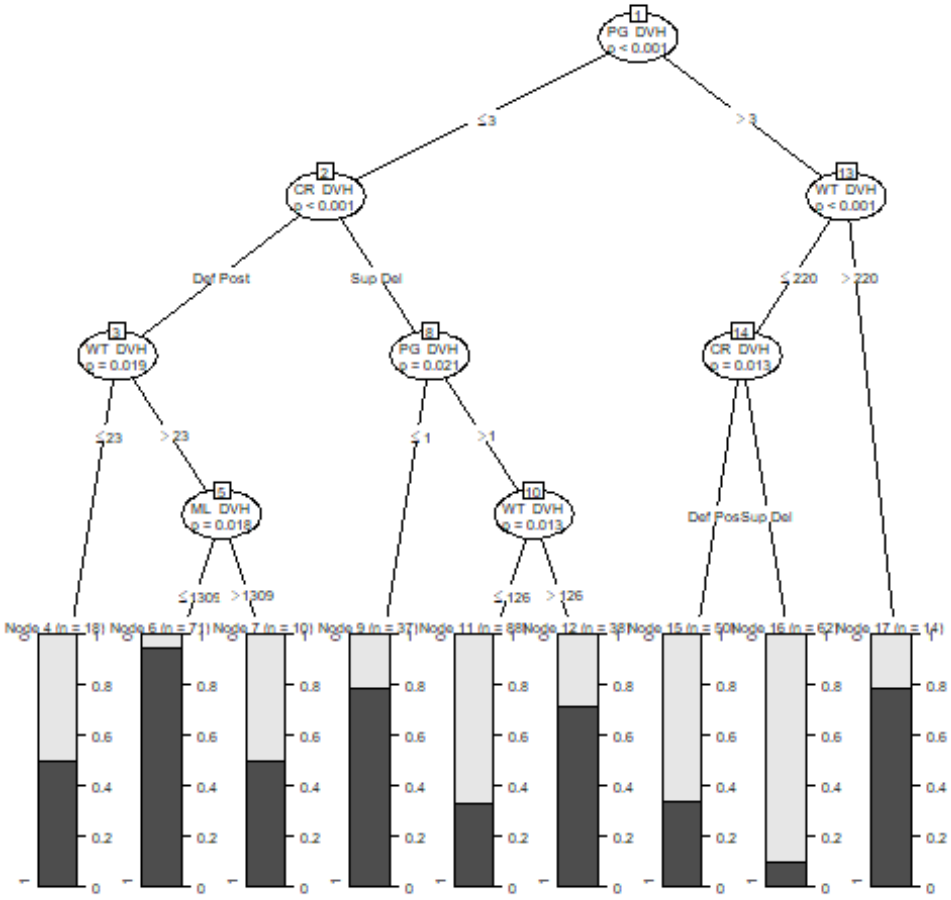
```
start_time_DVH <- Sys.time()

RP_mod_DVH <- ctree(OT_DVH ~ ., data=train_data_DVH)

end_time_DVH <- Sys.time()

RP_Time_DVH <- end_time_DVH - start_time_DVH

plot(RP_mod_DVH, gp=gpar(fontsize=5))
```



Interpretation

The model consists of 8 decision nodes and 9 terminal nodes.
From output, The left branch of the tree is split based on the "CR_DVH" variable.

If "CR_DVH" is in the category "Def Post", the subset is further split based on the "WT_DVH" variable. If "WT_DVH" is less than or equal to 23, the predicted value is 0.

If "WT_DVH" is greater than 23 but less than or equal to 1309, the predicted value is 1. If "WT_DVH" is greater than 1309, the predicted value is 0.

Like that it is splitted further.

The Lowest one is Node 16 with pair of 1,13 an 14.

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

```
# Predict on train set
pred_RP_train_DVH <- predict(RP_mod_DVH, newdata=train_data_DVH)

# Generate confusion matrix for train set
CF_RP_tain_DVH <- table(Actual=train_data_DVH$OT_DVH,
Predicted=pred_RP_train_DVH)
```

```r
# Predict on test set
pred_RP_test_DVH <- predict(RP_mod_DVH, newdata=test_data_DVH)

# Generate confusion matrix for test set
CF_RP_test_DVH <- table(Actual=test_data_DVH$OT_DVH,
Predicted=pred_RP_test_DVH)



# Confusion Matrices of for Train set
CF_RP_tain_DVH
```

```
##       Predicted
## Actual   0   1
##      0 162  26
##      1  66 134
```

```r
# Confusion Matrices of for Test set
CF_RP_test_DVH
```

```
##       Predicted
## Actual  0  1
##      0 35 10
##      1 16 37
```

Interpretation

It shows the number of correct and incorrect predictions made by the model on the training and test data.

`For the training data:`

- True negative (TN) = 162
- False positive (FP) = 26
- False negative (FN) = 66
- True positive (TP) = 134

`For the test data:`

- True negative (TN) = 35
- False positive (FP) = 10
- False negative (FN) = 16
- True positive (TP) = 37

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary.

```
# Time calculated to fit Recursive Partitioning Analysis
RP_Time_DVH

## Time difference of 0.06176496 secs
```

Interpretation

Total Time taken to fit Recursive Partitioning Classification model is 0.04825306 secs.

3a. BONUS SECTION

1. Use all the variables in the dataset to fit a Neural Network classification model. Set the seed to 8430, the size to 3, rang=0.1 and maxit=1200.

```
start_time_DVH <- Sys.time()

set.seed(8430)
nn_mod_DVH <- nnet(OT_DVH ~ .,
          data=train_data_DVH,
          size=3,
          rang=0.1,
          maxit=1200,
          trace=FALSE)

end_time_DVH <- Sys.time()

NN_Time_DVH <- end_time_DVH - start_time_DVH
```

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

```
# Predict on train set
pred_nn_train_DVH <- predict(nn_mod_DVH, newdata=train_data_DVH,
type="class")

# Generate confusion matrix for train set
CF_NN_train_DVH <- table(Actual=train_data_DVH$OT_DVH,
Predicted=pred_nn_train_DVH)

# Predict on train set
pred_nn_test_DVH <- predict(nn_mod_DVH, newdata=test_data_DVH, type="class")

# Generate confusion matrix for train set
CF_NN_test_DVH <- table(Actual=test_data_DVH$OT_DVH,
Predicted=pred_nn_test_DVH)




# Confusion Matrices of for Train set
CF_NN_train_DVH
```

```
##       Predicted
## Actual   0   1
##      0  16 172
##      1   7 193
```

```
# Confusion Matrices of for Test set
CF_NN_test_DVH
```

```
##       Predicted
## Actual  0  1
##      0  3 42
##      1  3 50
```

Interpretation
It shows the number of correct and incorrect predictions made by the model on the training and test data.

For the training data:

- True negative (TN) = 16
- False positive (FP) = 172
- False negative (FN) = 7
- True positive (TP) = 193

For the test data:

- True negative (TN) = 3
- False positive (FP) = 42
- False negative (FN) = 3
- True positive (TP) = 50

3. As demonstrated in class, calculate the time (in seconds) it took to fit the model and include this in your summary.

```
# Time calculated to fit Neural Network Model
NN_Time_DVH
```

```
## Time difference of 0.0160141 secs
```

Interpretation
Total Time taken to fit Recursive Partitioning Classification model is 0.06588602 secs.

4. Compare All Classifiers

1. Which classifier is most accurate?

From test data confusion matrices, Accuracy = (TP+TN)/Total

- Logistic Regression :76/98 (77.55% accuracy).
- Naïve-Bayes correctly :77/98 (78.57% accuracy).
- Recursive Partitioning Analysis :72/98 (73.47% accuracy).
- Neural Network :53/98 (54.08% accuracy).

Based on the test data results, the Naïve-Bayes Classification model appears to be the most accurate classifier as it has the highest number of correct predictions for both actual 0 and actual 1.

2. Which classifier seems most consistent (think train and test)?

Comparing train and test accuracy of models,
Test accuracy is already calculated of all models in Question 1.

`Accuracy of Train data:`

- Logistic Regression :295/388 (76.03% accuracy).
- Naïve-Bayes correctly :285/388 (73.45% accuracy).
- Recursive Partitioning Analysis :296/388 (76.28% accuracy).
- Neural Network :209/388 (53.86% accuracy).

From the result, Logistic Regression model has the highest consistency between the train (76.45%) and test(77.55%) accuracy.
The difference between the train and test accuracy is relatively low, indicating that the model is not overfitting or underfitting.

3. Which classifier is most suitable when processing speed is most important?

The Naïve-Bayes Classification model seems to be the most suitable when processing speed is most important, as it has the lowest time taken to fit the data of all the classifiers.

4. Which classifier minimizes false positives?

From confusion matrices, the false positive(FP) values for each classifier's test data are as follows:

- Logistic Regression: 12
- Naïve-Bayes Classification: 12
- Recursive Partitioning Analysis: 10
- Neural Network classification model: 42

Therefore, the classifier that minimizes false positives is Recursive Partitioning Analysis with a value of 10.

5. In your opinion, classifier is best overall?

By considering 5 factors which are `Highest Accuracy`, `Min False Positive`, `Min False Negative`, `Highest Precision` and `Highest Sensitivity`, I have sorted models below.

Highest Accuracy :Logistic Regression
Min false Positive :Recursive Partitioning
Min False Negative :Neural Network
Highest Precision :Recursive Partitioning
Highest Sensitivity :Neural Network

Now, it is completely depend on the specific requirements and priorities of the problem at hand to choose best one.

Hence, If the goal is to get highest accuracy i would go with Logistic Regression. If the priority is to minimize false negatives, then Neural Network would be the best choice. If the priority is to have high precision, then Recursive Partitioning Analysis would be the best choice.