Professional Master's in Artificial Intelligence
Fundamentals for Applied Data Science (DTI 5126)

Subject: Assignment 2 (Classification & Clustering)

By

Mohamed Sayed Abdelwahab Hussein

(300273145)

Mhuss073@uottawa.ca

Under Supervision
Dr. Olubisi Runsewe

Part A (Classification):

i. Ensure data is in the correct format for downstream processes and address missing data

```
Console    Terminal ×    Jobs ×
R 4.1.0 · /cloud/project/
> # Check the format of the data
> unique(df$MultipleLines)
[1] "No phone service" "No"               "Yes"
> unique(df$OnlineSecurity)
[1] "No"                  "Yes"                  "No internet service"
> unique(df$OnlineBackup)
[1] "Yes"                 "No"                   "No internet service"
> unique(df$DeviceProtection)
[1] "No"                  "Yes"                  "No internet service"
> unique(df$StreamingMovies)
[1] "No"                  "Yes"                  "No internet service"
> unique(df$StreamingTV)
[1] "No"                  "Yes"                  "No internet service"
> unique(df$TechSupport)
[1] "No"                  "Yes"                  "No internet service"
>
```

```
#Transform the data
df$MultipleLines <- revalue(df$MultipleLines, replace= c("No phone service" = "No"))
df$OnlineSecurity <- revalue(df$OnlineSecurity, replace= c("No internet service" = "No"))
df$OnlineBackup <- revalue(df$OnlineBackup, replace= c("No internet service" = "No"))
df$DeviceProtection<- revalue(df$DeviceProtection, replace= c("No internet service" = "No"))
df$StreamingMovies <- revalue(df$StreamingMovies, replace= c("No internet service" = "No"))
df$StreamingTV <- revalue(df$StreamingTV, replace= c("No internet service" = "No"))
df$TechSupport <- revalue(df$TechSupport, replace= c("No internet service" = "No"))
df$SeniorCitizen <- as.factor(df$SeniorCitizen)
```

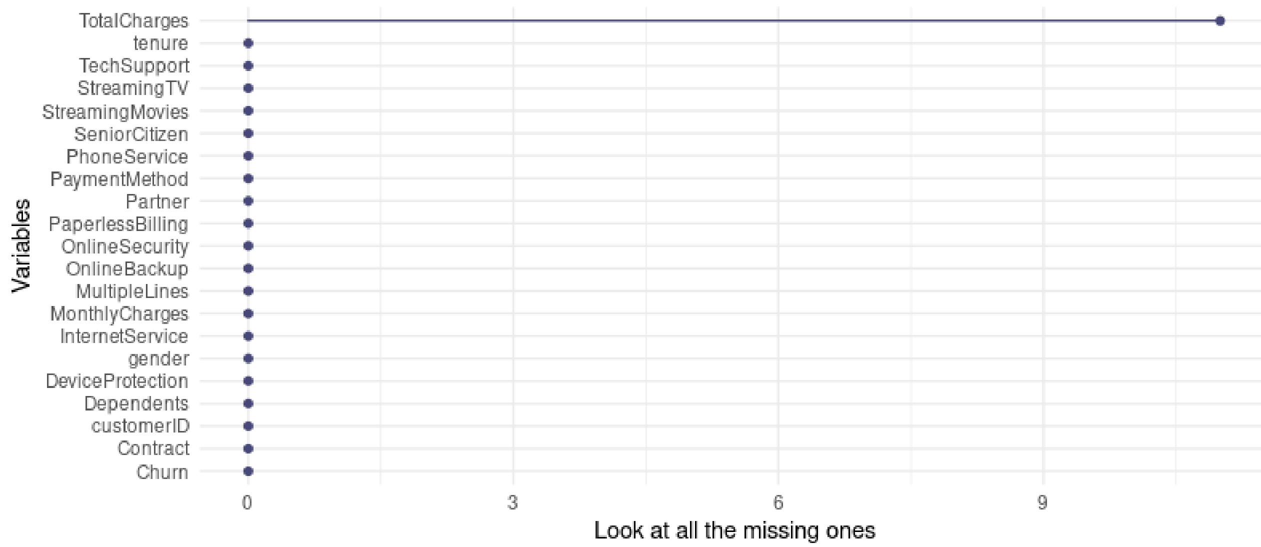
After investigating the data, we have an assumption here: the values of “No Internet service” has equal weight to the value of “No”, so we will convert them to “No” so the feature values will be binary “Yes” and “No”.

Then we will drop the customerID column: -

(df<- select(df, -customerID)).

Let`s check and address the missing values

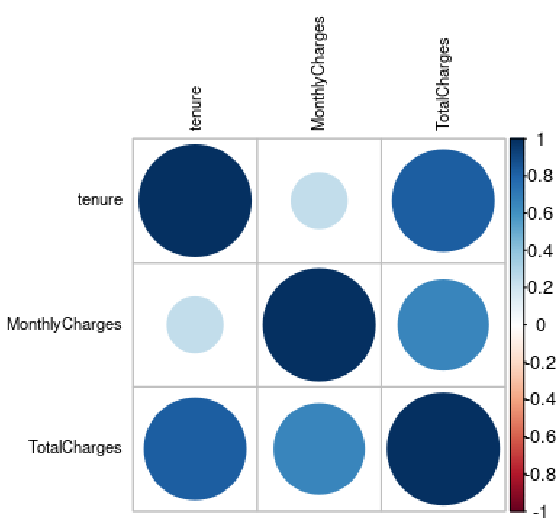
```
# Check missing values
sum(is.na(df))
gg_miss_var(df) + labs(y = "Look at all the missing ones")
# handle missing values by replacing NA with mean value of TotalCharges
df$TotalCharges <- replace(df$TotalCharges,is.na(df$TotalCharges),mean(df$TotalCharges,na.rm=TRUE))
sum(is.na(df))
```

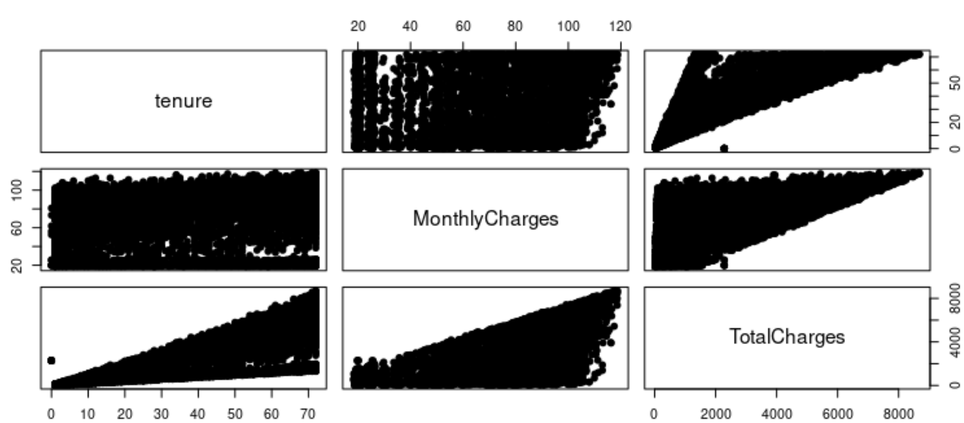


We noticed that "TotalChanges" column has 11 missing values, so we replaced them by the mean of the total charges column.

```
# handle missing values by replacing NA with mean value of TotalCharges
df$TotalCharges <- replace(df$TotalCharges,is.na(df$TotalCharges),mean(df$TotalCharges,na.rm=TRUE))
```

ii. Generate a scatterplot matrix to show the relationships between the variables and a correlation matrix to determine correlated attributes

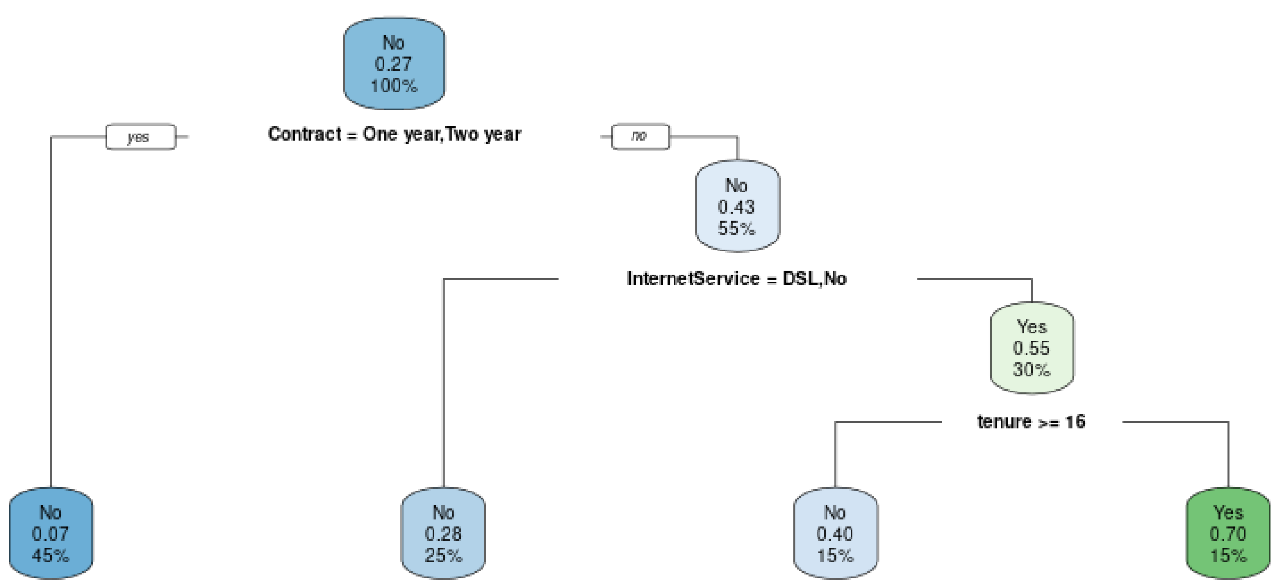




iii. Split the dataset into 80 training /20 test set and fit a decision tree to the training data. Plot the tree, and interpret the results.

iv. Describe the first few splits in the decision tree. Extract some rules.

```
# split data into 80% training and 20% testing
set.seed(42)
sample_split <- sample.split(Y = df$Churn, SplitRatio = 0.80)
train_set <- subset(x = df, sample_split == TRUE)
test_set <- subset(x = df, sample_split == FALSE)

# fit training data on the decision tree
dt_model <- rpart(Churn ~ ., data = train_set, method = "class")
dt_model

# Plot the decision tree
rpart.plot(dt_model)
rpart.rules(dt_model, roundint=FALSE, clip.facs=TRUE)
```



```
> rpart.rules(dt_model, roundint=FALSE, clip.facs=TRUE)
 Churn
  0.07 when One year or Two year
  0.28 when       Month-to-month &    DSL or No
  0.40 when       Month-to-month & Fiber optic & tenure >= 16
  0.70 when       Month-to-month & Fiber optic & tenure <  16
```

From the above figure we can tell that: -

- root node is the overall probability of Churn. It shows the proportion of customers that churned. 27% of customers churned. This node asks whether the customer's contract is one year or two-year. If yes, 45% from customers are with one or two-year contract with a churn probability of 7%. If no 55% from customers are without one or two-year contract with a churn probability of 43%.
- in the second node, you ask if the customer's internetService is DSL or No. If yes, then the chance of churned is 28%.
- in the third node, you ask if the customer's tenure greater than or equal 16. If yes, then the chance of churned is 40%.

v. Try different ways to improve your decision tree algorithm (e.g., use different splitting strategies, prune tree after splitting). Does pruning the tree improves the accuracy?

Use different splitting strategies

1. With default splitting parameter (Gini): -

```
# make predictions
preds <- predict(dt_model, newdata = test_set, type = "class")

#Print the confusion Matrix
confusionMatrix(as.factor(test_set$Churn),preds)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  965  70
       Yes 232 142

               Accuracy : 0.7857
```

2. With information gain splitting parameter: -

```
# using information gain spiting strategy instead of Gini
dt_model2 <- rpart(Churn ~ ., data = train_set, parms=list(split= "information"))
# make predictions
preds2 <- predict(pruned_model2, newdata = test_set, type = "class")
#Print the confusion Matrix
confusionMatrix(as.factor(test_set$Churn),preds2)
```

```
          Reference
Prediction  No Yes
       No  965  70
       Yes 232 142

               Accuracy : 0.7857
```

We tried Gini and Information Gain strategies for splitting, there is no different in accuracy.

Prune tree after splitting

1. With default splitting parameter (Gini): -

Printcp(dt_model)

```
        CP nsplit rel error  xerror     xstd
0.072687      0   1.00000 1.00000 0.022168
0.010000      3   0.78194 0.78462 0.020385
```

It is clear from the above, that the lowest cross-validation error (xerror in the table) occurs for alpha =0.0100. we will get this value programmatically, then we will prune the tree based on the value of cp, make predications and get accuracy.

```
#cost-complexity pruning
printcp(dt_model)
# get index of CP with lowest xerror
opt <- which.min(dt_model$cptable[,"xerror"])
#get its value
cp <- dt_model$cptable[opt, "CP"]
#prune tree
pruned_model1 <- prune(dt_model,cp)
# make predictions
preds1 <- predict(pruned_model1, newdata = test_set, type = "class")
#Print the confusion Matrix
confusionMatrix(as.factor(test_set$Churn),preds1)
```

```
          Reference
Prediction  No Yes
       No  965  70
       Yes 232 142

               Accuracy : 0.7857
```

It's the same accuracy of the default model.

2. With spiriting parameter (Information Gain): -

```
# using information gain spiting strategy instead of Gini
dt_model2 <- rpart(Churn ~ ., data = train_set, parms=list(split= "information"))
#cost-complexity pruning
printcp(dt_model2)
# get index of CP with lowest xerror
opt2 <- which.min(dt_model2$cptable[,"xerror"])
#get its value
cp2 <- dt_model2$cptable[opt2, "CP"]
#prune tree
pruned_model2 <- prune(dt_model2,cp2)
# make predictions
preds2 <- predict(pruned_model2, newdata = test_set, type = "class")
#Print the confusion Matrix
confusionMatrix(as.factor(test_set$Churn),preds2)
```

```
          Reference
Prediction  No Yes
       No  965  70
       Yes 232 142

               Accuracy : 0.7857
```

It's the same accuracy of the default model.

vi. Train an XGboost model using 10-fold cross-validation repeated 3 times and a hyperparameter grid search to train the optimal model. Evaluate the performance.

First, we will split the original data to features and labels, then convert all categorical data to numeric.

```
# XGboost model using 10-fold cross-validation
X_data = select(train_set, -Churn)
y_label = select(train_set, Churn)

# label encoding for the data
X_data$gender = as.integer(factor(X_data$gender))
X_data$Partner = as.integer(factor(X_data$Partner))
X_data$SeniorCitizen = as.integer(factor(X_data$SeniorCitizen))
X_data$Dependents = as.integer(factor(X_data$Dependents))
X_data$PhoneService = as.integer(factor(X_data$PhoneService))
X_data$MultipleLines = as.integer(factor(X_data$MultipleLines))
X_data$InternetService = as.integer(factor(X_data$InternetService))
X_data$OnlineSecurity = as.integer(factor(X_data$OnlineSecurity))
X_data$OnlineBackup = as.integer(factor(X_data$OnlineBackup))
X_data$DeviceProtection = as.integer(factor(X_data$DeviceProtection))
X_data$TechSupport = as.integer(factor(X_data$TechSupport))
X_data$StreamingTV = as.integer(factor(X_data$StreamingTV))
X_data$StreamingMovies = as.integer(factor(X_data$StreamingMovies))
X_data$Contract = as.integer(factor(X_data$Contract))
X_data$PaperlessBilling = as.integer(factor(X_data$PaperlessBilling))
X_data$PaymentMethod = as.integer(factor(X_data$PaymentMethod))
y_label$Churn = ifelse(y_label$Churn == "Yes",1,0)
```

Fit XGBoost model using cross validation and grid search

```
# train XGBoost with cross validation and grid search
xgb_params <- expand.grid(
  eta = c(0.01, 0.1, 1),
  lambda = c(0.1, 0.5, 1),
  alpha =  c(0.1, 0.5, 1))

xgb_model_cv = xgb.cv(params = as.list(xgb_params),
                   data = as.matrix(X_data),
                   label = y_label$Churn,
                   showsd = T,
                   stratified = T,
                   print_every_n = 1,
                   maximize = F,
                   prediction = TRUE,
                   nround = 3,
                   nfold = 10,
                   objective = "binary:logistic",
                   eval_metric='logloss')

print(xgb_model_cv)
xgb_model = xgboost(data = as.matrix(X_data),
                    label = y_label$Churn,
                    nround = 3,
                    objective = "binary:logistic",
                    eval_metric = 'logloss')
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 933 102
         1 186 188

               Accuracy : 0.7956
```

the accuracy increased by 0.1 in XGBoost.

```
> print(xgb_model_cv)
##### xgb.cv 10-folds
 iter train_logloss_mean train_logloss_std test_logloss_mean test_logloss_std
    1          0.6883177      3.795537e-05         0.6886398     0.0002110359
    2          0.6835898      8.039876e-05         0.6842303     0.0004198912
    3          0.6789480      1.101617e-04         0.6798992     0.0006176350
```

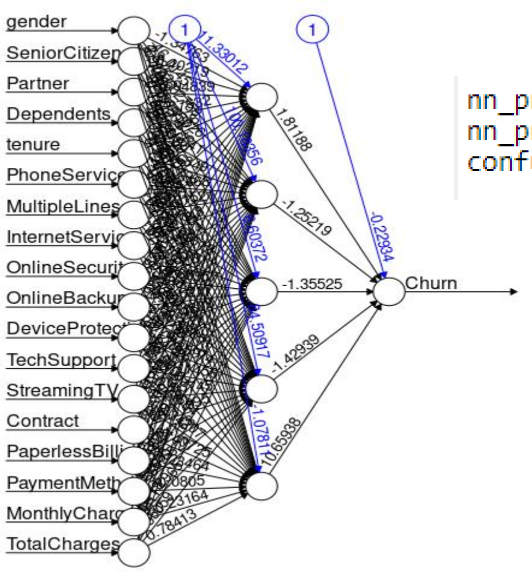
vii. Build a multilayer perceptron with 5 nodes at the hidden layer. Use a standard or normalization to scale the variables.

Here we will start by standardize the three numeric features in the dataset

```
# standardize the numeric column in the dataset
train_scaled_df = X_data
train_scaled_df$Churn = y_label$Churn
train_scaled_df$tenure = (train_scaled_df$tenure - mean(train_scaled_df$tenure)) / sd(train_scaled_df$tenure)
train_scaled_df$MonthlyCharges = (train_scaled_df$MonthlyCharges - mean(train_scaled_df$MonthlyCharges)) / sd(train_scaled_df$MonthlyChar
train_scaled_df$TotalCharges = (train_scaled_df$TotalCharges - mean(train_scaled_df$TotalCharges)) / sd(train_scaled_df$TotalCharges)
```

Fit NN model on the training data

```
NN_model = neuralnet(Churn ~ gender + SeniorCitizen + Partner + Dependents + tenure +
                     PhoneService + MultipleLines + InternetService + OnlineSecurity +
                     OnlineBackup + DeviceProtection + TechSupport + StreamingTV + Contract +
                     PaperlessBilling + PaymentMethod + MonthlyCharges + TotalCharges,
                     data= train_scaled_df, hidden = 5 , act.fct = "logistic", linear.output = F )
```



Making prediction using NN model and calculate accuracy

```
nn_preds_prop = compute(NN_model,test_scaled_df)
nn_preds = ifelse(nn_preds_prop$net.result>0.5, 1, 0)
confusionMatrix(as.factor(test_scaled_df$Churn), as.factor(nn_preds))
```

```
          Reference
Prediction   0   1
         0 928 107
         1 182 192

            Accuracy : 0.7949

         Sensitivity : 0.8360
         Specificity : 0.6421
      Pos Pred Value : 0.8966
      Neg Pred Value : 0.5134
          Prevalence : 0.7878
      Detection Rate : 0.6586
Detection Prevalence : 0.7346
   Balanced Accuracy : 0.7391
```

When we change some parameters like (activation function = “thanh”, learning rate = “0.2”, neurons=2, no_hidden layers= 2, threshold= 0.1, stepmax= 1e7)

```
# change hyperparamters of the model
NN_model_tanh = neuralnet(Churn ~ gender + SeniorCitizen + Partner + Dependents + tenure +
                          PhoneService + MultipleLines + InternetService + OnlineSecurity +
                          OnlineBackup + DeviceProtection + TechSupport + StreamingTV + Contract +
                          PaperlessBilling + PaymentMethod + MonthlyCharges + TotalCharges,
                        data= train_scaled_df, hidden = c(2, 2), act.fct = 'tanh', linear.output = F,
                        learningrate = 0.2, threshold = 0.1, stepmax=1e7)
```

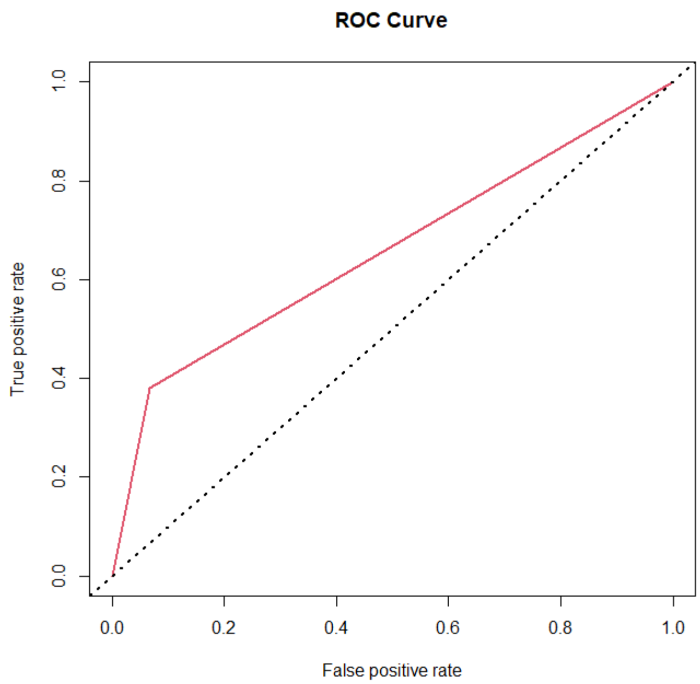
```
          Reference
Prediction   0   1
         0 937  98
         1 172 202

            Accuracy : 0.8084

         Sensitivity : 0.8449
         Specificity : 0.6733
      Pos Pred Value : 0.9053
      Neg Pred Value : 0.5401
          Prevalence : 0.7871
      Detection Rate : 0.6650
Detection Prevalence : 0.7346
   Balanced Accuracy : 0.7591
```

the accuracy increased by a little bit when we changed some parameters.

viii. Carry out a ROC analysis to compare the performance of the DT, XGboost & NN techniques. Plot the ROC graph of the models.

1. Decision Tree ROC Curve: -

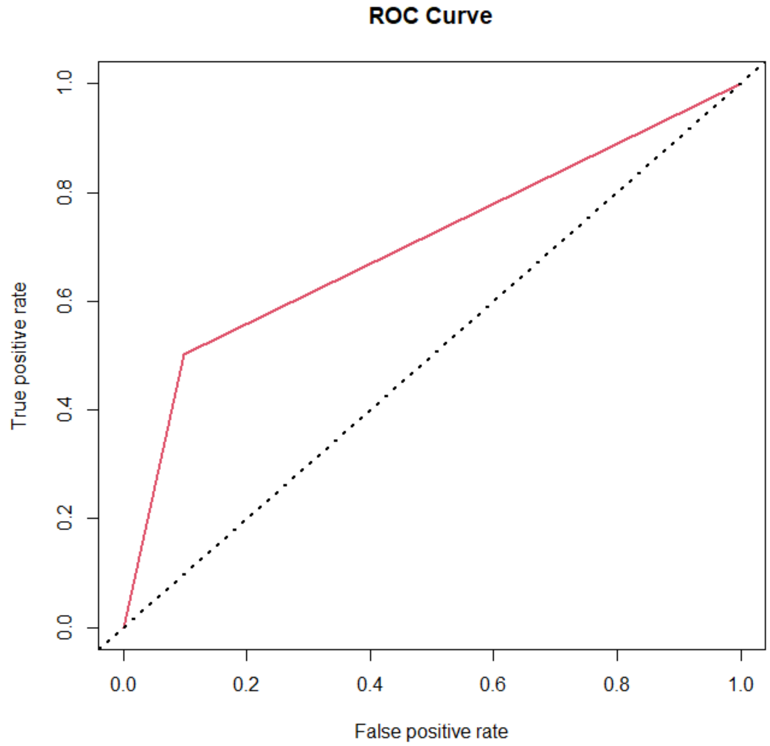
```
# plot ROC curve for the DT model
Predictions_dt <- prediction(as.numeric(preds), as.numeric(as.factor(test_set$Churn)))
pref_dt <- performance(Predictions_dt, "tpr","fpr")
plot(pref_dt,main = "ROC Curve",col = 2,lwd = 2)
abline(a = 0,b = 1,lwd = 2,lty = 3,col = "black")
auc_dt <- performance(Predictions_dt, measure = "auc")
auc_dt <- auc_dt@y.values[[1]]
print(auc_dt)
```



AUC = 0.6560231

2. XGBoost ROC Curve: -

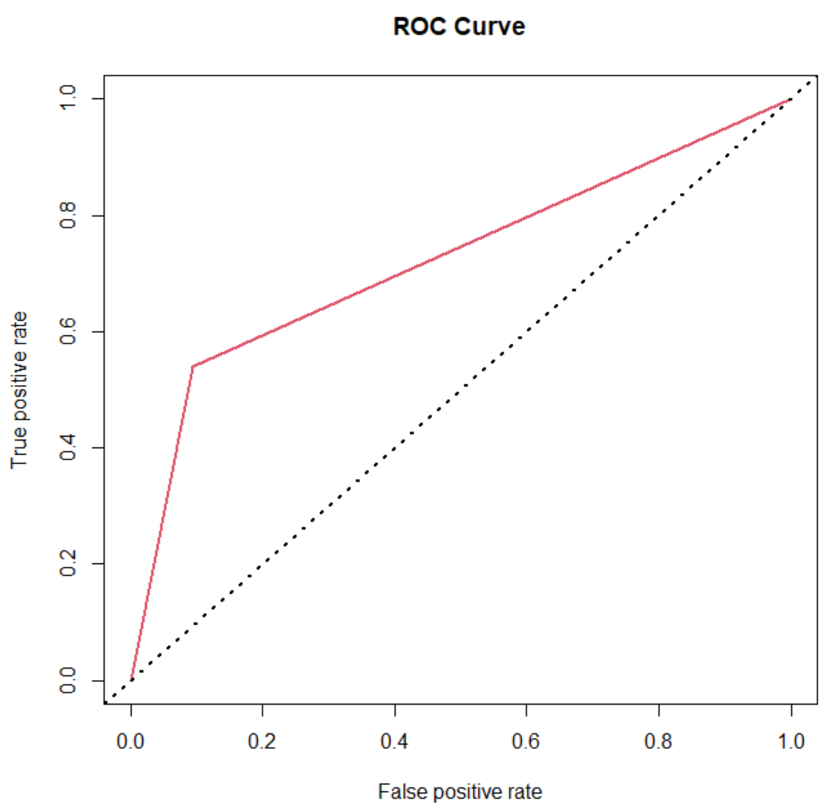
```
# plot ROC curve for the XGBoost model
Predictions_xgb <- prediction(xgb_preds, y_label_test$Churn)
pref_xgb<- performance(Predictions_xgb, "tpr","fpr")
plot(pref_xgb,main = "ROC Curve",col = 2,lwd = 2)
abline(a = 0,b = 1,lwd = 2,lty = 3,col = "black")
auc_xgb <- performance(Predictions_xgb, measure = "auc")
auc_xgb <- auc_xgb@y.values[[1]]
print(auc_xgb)
```



AUC = 0.7020615

3. NN ROC Curve: -

```
# plot ROC curve for the XGBoost model
Predictions_nn <- prediction(as.numeric(nn_preds2), as.numeric(test_scaled_df$Churn))
pref_nn<- performance(Predictions_nn, "tpr","fpr")
plot(pref_nn,main = "ROC Curve",col = 2,lwd = 2)
abline(a = 0,b = 1,lwd = 2,lty = 3,col = "black")
auc_nn <- performance(Predictions_nn, measure = "auc")
auc_nn <- auc_nn@y.values[[1]]
print(auc_nn)
```



AUC = 0.7227105

Part B (Clustering):

I. a) Perform k-means clustering, specifying k = 2 clusters and plot. Determine the attribute that is most correlated with the clusters.

First, we remove the first two columns from the original dataset

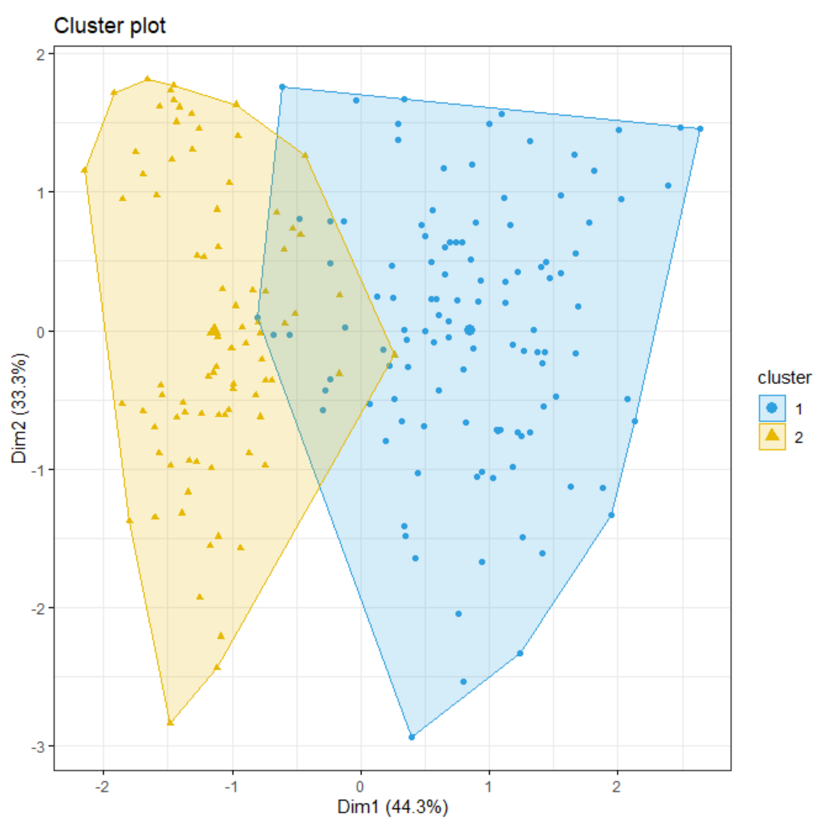
```
# we will remove the first 2 columns (CustomerId, Gender)
df<- select(df, -c(CustomerID, Gender))
```

Then we will check the missing values

```
> sum(is.na(df))
[1] 0
```

After assessing the data, we will run K-means cluster with 2 clusters.

```
# Run k-means cluster on the data set
set.seed(123)
cluster_kmean <- kmeans(df[,1:3], 2, nstart = 20)

# plot the k-means clusters
fviz_cluster(cluster_kmean, data = df,
             palette = c("#2E9FDF", "#E7B800"),
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
)
```



Add cluster labels to the original dataset and calculate correlation and determine the attributes that most correlated with the cluster.

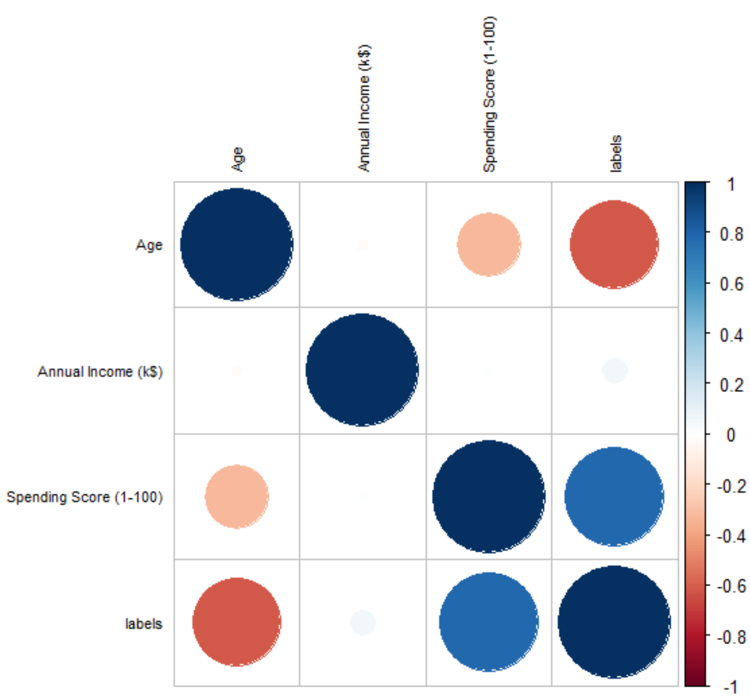
```
# add cluster labels to the original dataset
df$labels <- cluster_kmean$cluster
#calculate correlation matrix
correlationMatrix <- cor(df)
View(correlationMatrix)
# determine attributes that are highly corrected (ideally > 0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.70)
hc = sort(highlyCorrelated)
hc
reduced_Data = df[ ,-c(hc)]
View (reduced_Data)
```

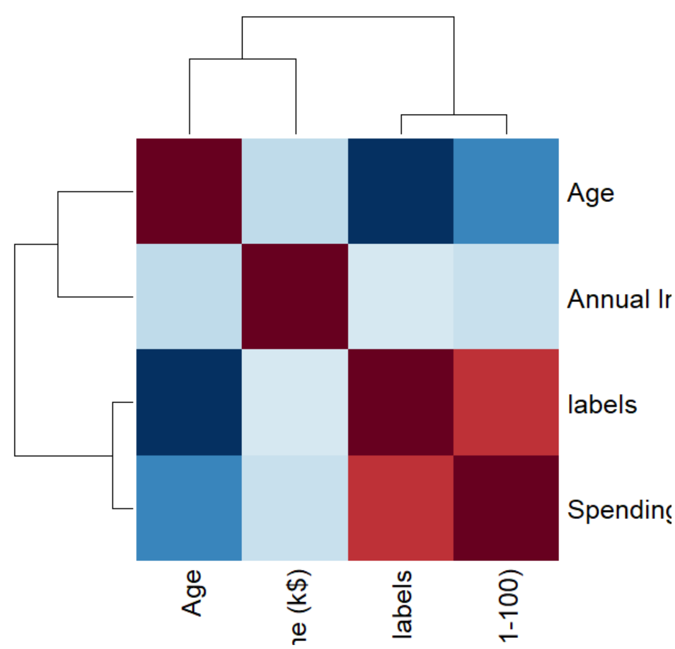
	Age	Annual Income (k$)	Spending Score (1-100)	labels
Age	1.00000000	-0.012398043	-0.327226846	-0.61064613
Annual Income (k$)	-0.01239804	1.000000000	0.009902848	0.05304495
Spending Score (1-100)	-0.32722685	0.009902848	1.000000000	0.78178236
labels	-0.61064613	0.053044946	0.781782362	1.00000000

	Age	Annual Income (k$)	Spending Score (1-100)
1	19	15	39
2	21	15	81
3	20	16	6
4	23	16	77
5	31	17	40
6	22	17	76
7	35	18	6

Plot heatmap

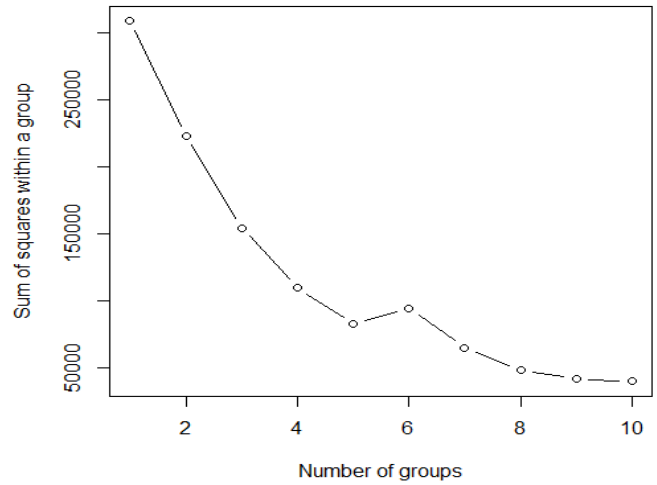
```
# plot heatmap
rquery.cormat(df)
cormat<-rquery.cormat(df, graphType="heatmap")
corrplot(cor(df),
         method = "circle",
         type = "full",
         diag = TRUE,
         tl.col = "black",
         bg = "white",
         title = "",
         col = NULL,
         tl.cex =0.7,
         cl.ratio =0.2)
```





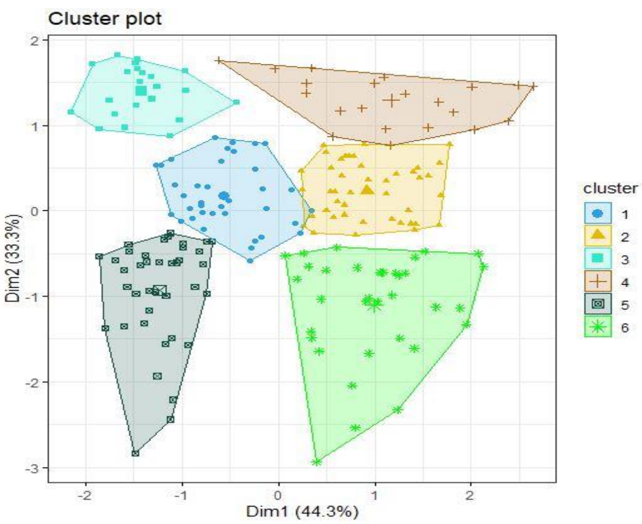
B) Apply the elbow method to determine the best k and plot.

```r
# plot elbow method to determine the best k
wssplot <- function(data, nc=15, seed=123){
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of groups",
       ylab="Sum of squares within a group")}

wssplot(df, nc = 10)
```



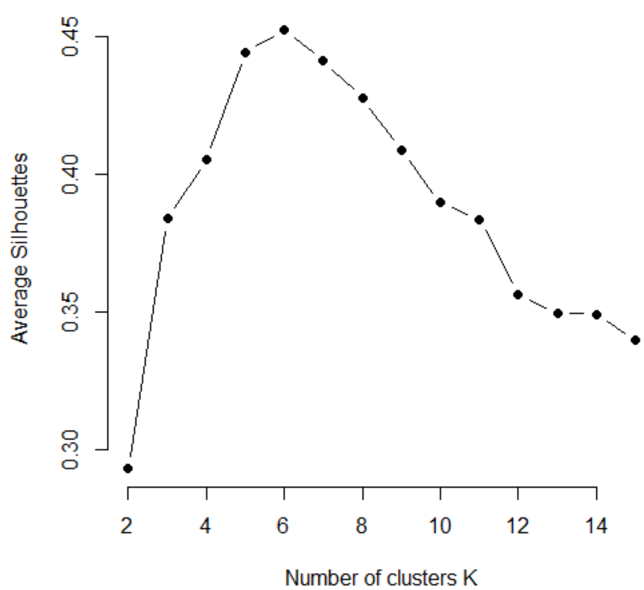
From above figure

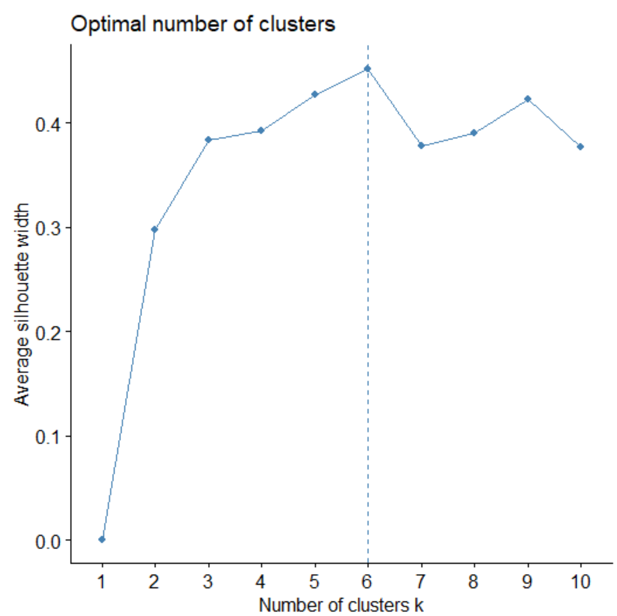
the optimal number of clusters = 6




```r
# Run k-means with k = 6
set.seed(123)
cluster_kmean2 <- kmeans(df_original[,1:3], 6, nstart = 20)

# plot the k-means clusters
fviz_cluster(cluster_kmean2, data = df_original,
             palette = c("#2E9FDF", "#E7B800", "#32e7c8", "#a75e11", "#134e41","#ffa500"),
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
)
```

c) Evaluate the quality of the clusters using Silhouette Coefficient method.

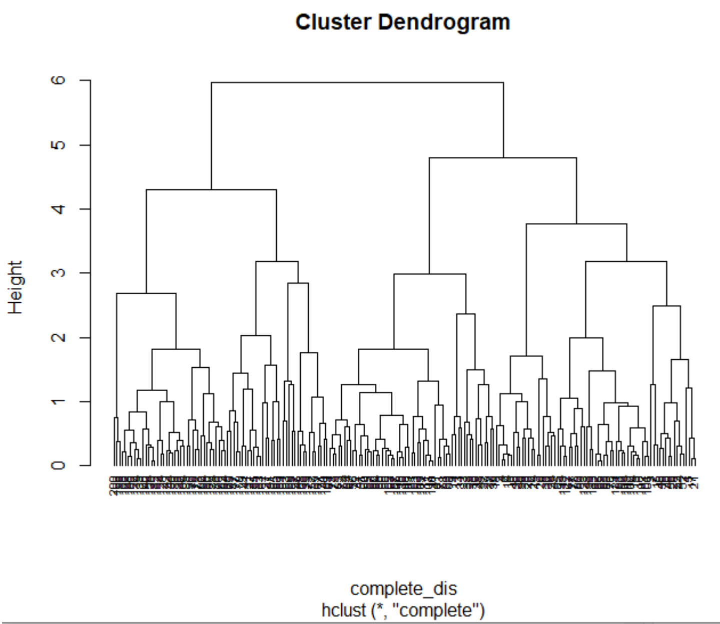


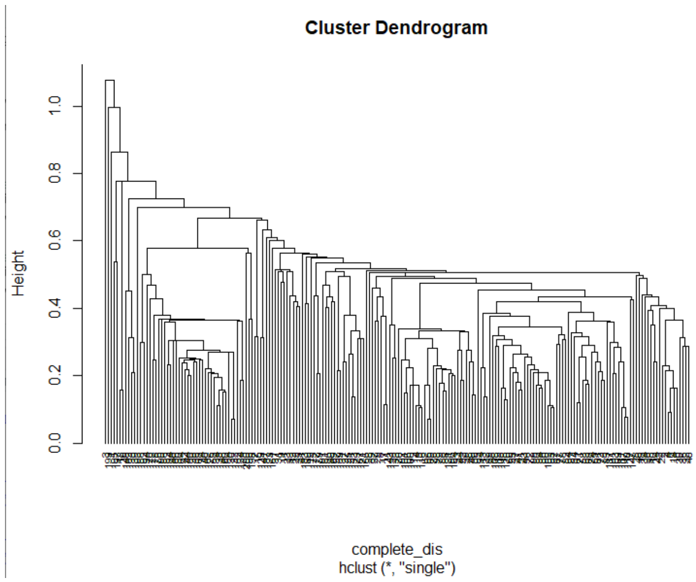


d) Apply hierarchical clustering (single & complete linkage) to the dataset using Euclidean-based distance, and plot the dendrogram. Do your results depend on the type of linkage used.

```
# Compute distances and hierarchical clustering (complete)
complete_dis <- dist(scale(df_original), method = "euclidean")
complete_hc <- hclust(complete_dis, method = "complete")
plot(complete_hc, hang = -1, cex = 0.6)
```

```
# Compute distances and hierarchical clustering (single)
complete_dis <- dist(scale(df_original), method = "euclidean")
complete_hc <- hclust(complete_dis, method = "single")
plot(complete_hc, hang = -1, cex = 0.6)
```





- Single-linkage (nearest neighbor) is the shortest distance between a pair of observations in two clusters. It can sometimes produce clusters where observations in different clusters are closer together than to observations within their own clusters.
- Complete-linkage (farthest neighbor) is where distance is measured between the farthest pair of observations in two clusters. This method usually produces tighter clusters than single-linkage, but these tight clusters can end up very close together.

II. Consider the following "data" to be clustered: **10 20 40 80 85 121 160 168 195.**

	10	20	40	80	85	121	160	168	195
10	0								
20	10	0							
40	30	20	0						
80	70	60	40	0					
85	75	65	45	5	0				
121	111	101	81	41	36	0			
160	150	140	120	80	75	39	0		
168	158	148	128	88	83	47	8	0	
195	185	175	155	115	110	74	35	27	0

	10	20	40	80, 85	121	160	168	195
10	0							
20	10	0						
40	30	20	0					
80, 85	70	60	40	0				
121	111	101	81	36	0			
160	150	140	120	75	39	0		
168	158	148	128	83	47	8	0	
195	185	175	155	110	74	35	27	0

	10	20	40	80, 85	121	160, 168	195
10	0						
20	10	0					
40	30	20	0				
80, 85	70	60	40	0			
121	111	101	81	36	0		
160, 168	150	140	120	75	39	0	
195	158	175	155	110	74	27	0

	10, 20	40	80, 85	121	160, 168	195
10, 20	0					
40	20	0				
80, 85	60	40	0			
121	101	81	36	0		
160, 168	40	120	80	39	0	
195	175	155	110	74	27	0

	10, 20, 40	80, 85	121	160, 168	195
10, 20, 40	0				
80, 85	40	0			
121	81	36	0		
160, 168	120	75	39	0	
195	155	110	74	27	0

	10, 20, 40	80, 85	121	160, 168, 195
10, 20, 40	0			
80, 85	40	0		
121	81	36	0	
160, 168, 195	120	75	39	0

	10, 20, 40	80, 85, 121	160, 168, 195
10, 20, 40	0		
80, 85, 121	40	0	
160, 168, 195	120	39	0

	10, 20, 40	80, 85, 121, 160, 168, 195
10, 20, 40	0	
80, 85, 121, 160, 168, 195	40	0

```
                          |
                        97.67
                         40
            ___________________________________
            |                                 |
            |                              134.83
            |                                39
            |                     _________________________
            |                     |                       |
          23.33                 95.33                  174.33
           20                    36                      27
     ___________          ____________           _____________
     |         |          |          |           |           |
     15        |         82.5        |          164          |
     10        |          5          |           8           |
 ________      |      ________       |       ________        |
 |      |      |      |      |       |       |      |        |
 10     20     40     80     85     121     160    168      195
```