uOttawa

DTI5126 – FUNDAMENTALS/APPLIED DATA SCIENCE

Assignment 2

Name: Ali Amin El-Sayed Mahmoud El-Sherif ID: 300327246

Table of Contents

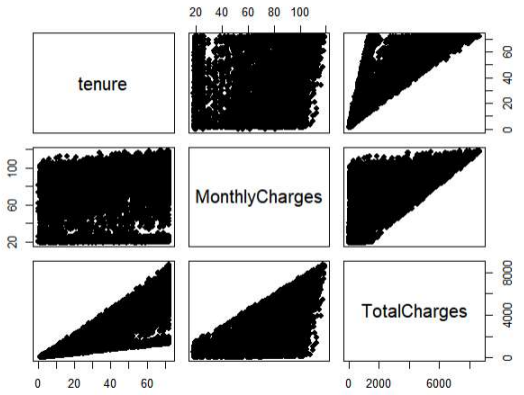
Part A 2
1 2
2 3
3 3
4 4
5 5
6 6
7 7
8 7
Part B 9
A) 9
B) 10
C) 11

Part A

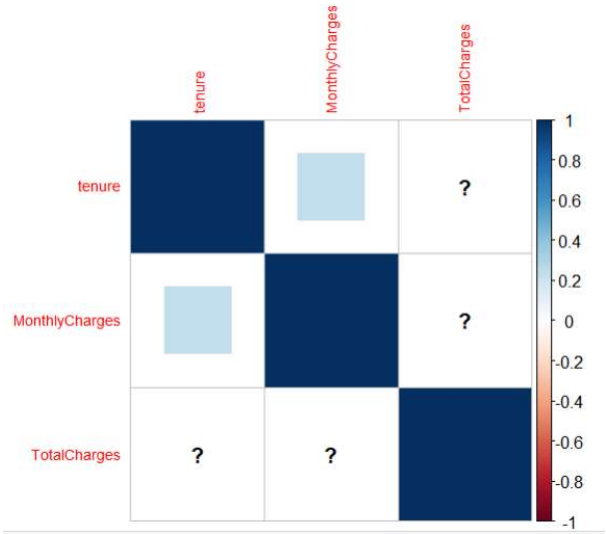
1

Scatterplot:

```
data = subset(dataset, select = c("tenure","MonthlyCharges","TotalCharges"))
pairs(data, pch = 19)

correlationMatrix <- cor(data)
print(correlationMatrix)

corrplot(cor(data), method = "square", type = "full", diag = TRUE, tl.col = "red", bg = "white", title = "",
         col = NULL,
         tl.cex =0.7,
         cl.ratio =0.2)
```

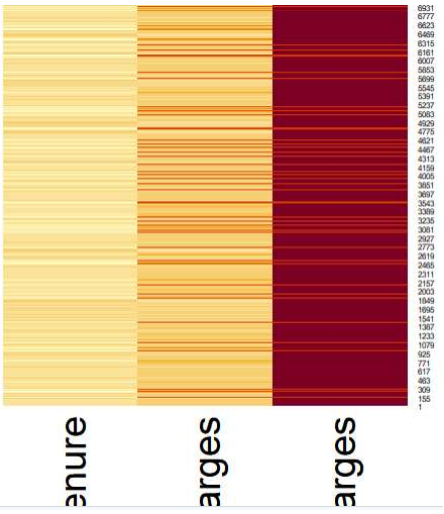



```
                tenure MonthlyCharges TotalCharges
tenure       1.0000000      0.2478999           NA
MonthlyCharges 0.2478999    1.0000000           NA
TotalCharges        NA             NA            1
```



Heat map:

```
correlationmatrix <- round(x = cor(data), digits = 2)
head(correlationmatrix)

heat_dataset <- as.matrix(data)
heatmap(heat_dataset, Rowv = NA, Colv = NA)
```



2

Data Cleaning:

```
# 2.
anyNA(dataset)

#find the columns with missing values(NA)
missing <- colnames(dataset)[apply(dataset, 2, anyNA) ]
missing
#Remove missing values
dataset_N <- na.omit(dataset)
#Drop "CustomerID"
dataset_N <- dataset_N[,!(names(dataset_N) %in% c("customerID"))]
dataset_N
#check again if any missing values exits
anyNA(dataset_N)
sum(is.na(dataset_N))

# Convert categorical data into numerical ones
md.pattern(dataset_N, plot = FALSE)
```

```
 /\_/\
{ `---' }
{  O O  }
==>  V <==  No need for mice. This data set is completely observed.
 \  \|/  /
  `-----'

     gender SeniorCitizen Partner Dependents tenure PhoneService MultipleLines InternetService OnlineSecurity
7032      1             1       1          1      1            1             1              1              1
          0             0       0          0      0            0             0              0              0
     OnlineBackup DeviceProtection TechSupport StreamingTV StreamingMovies Contract PaperlessBilling PaymentMethod
7032            1                1           1           1               1        1                1             1
                0                0           0           0               0        0                0             0
     MonthlyCharges TotalCharges Churn
7032              1            1     1 0
                  0            0     0 0
```

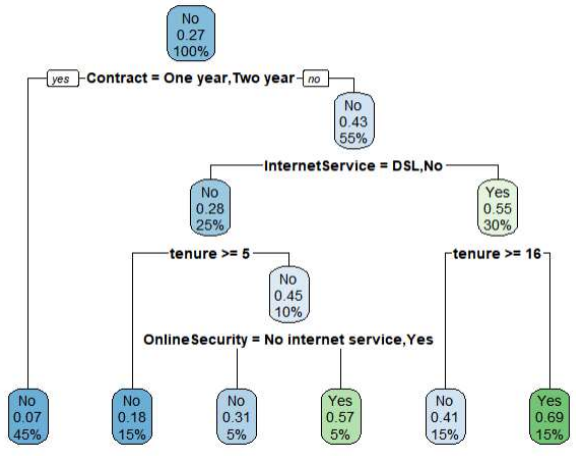
```
> anyNA(dataset_N)
[1] FALSE
> sum(is.na(dataset_N))
[1] 0
```

3

Decision Tree gini index:

```
set.seed(123)
state <- sample.split(Y = dataset_N$Churn, SplitRatio = 0.8)
training_Set <- subset(x = dataset_N, state == TRUE)
testing_Set <- subset(x = dataset_N, state == FALSE)
dim(training_Set)
dim(testing_Set)
#Decision tree
DecisionTree <- rpart(Churn ~ ., data = training_Set, method = "class")
rpart.plot(DecisionTree)

#Prediction
y_pred <- predict(DecisionTree, newdata = testing_Set , type = "class")

#Confusion Matrix
confmatrix1 <- confusionMatrix(as.factor(testing_Set$Churn), factor(y_pred),
                               mode = "prec_recall", dnn = c("Actual", "Prediction"))
confmatrix1
```

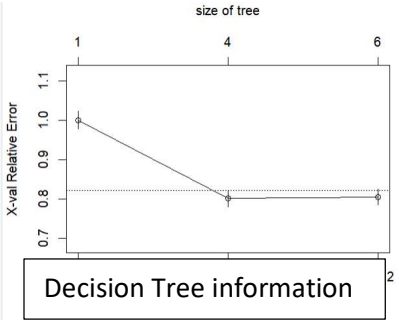
```
> dim(training_Set)
[1] 5625   20
> dim(testing_Set)
[1] 1407   20
```




```
> confmatrix1
Confusion Matrix and Statistics

          Prediction
Actual     No Yes
   No     928 105
   Yes    199 175

               Accuracy : 0.7839
                 95% CI : (0.7615, 0.8052)
    No Information Rate : 0.801
    P-Value [Acc > NIR] : 0.9479

                  Kappa : 0.3982

 Mcnemar's Test P-Value : 9.612e-08

              Precision : 0.8984
                 Recall : 0.8234
                     F1 : 0.8593
             Prevalence : 0.8010
         Detection Rate : 0.6596
   Detection Prevalence : 0.7342
      Balanced Accuracy : 0.7242

       'Positive' Class : No
```

4

```
decisionTree_Information <- rpart(Churn ~ ., data = training_Set, method = "class",
                                  parms = list(split = "information"))
decisionTree_Information
plotcp(decisionTree_Information)

y_ipred <- predict(decisionTree_Information, newdata = testing_Set , type = "class")

infoCM <- confusionMatrix(as.factor(testing_Set$Churn), factor(y_ipred),
                          mode = "prec_recall", dnn = c("Actual", "Prediction"))
infoCM

plot_confusionMatrix(infoCM)
ROSE::roc.curve(testing_Set$Churn, y_ipred)


#Prune
decisionTree_Prune <- rpart(Churn ~ ., data = training_Set, method = "class",
                            control = rpart.control(cp = 0.0082,
                                                    maxdepth = 3,
                                                    minsplit = 2))
rpart.plot(decisionTree_Prune)

prune_pred <- predict(decisionTree_Prune, newdata = testing_Set , type = "class")

CM_prune <- confusionMatrix(as.factor(testing_Set$Churn), factor(prune_pred), mode = "prec_recall",
                            dnn = c("Actual", "Prediction"))
CM_prune

plot_confusionMatrix(CM_prune)

ROSE::roc.curve(testing_Set$Churn, prune_pred)
```



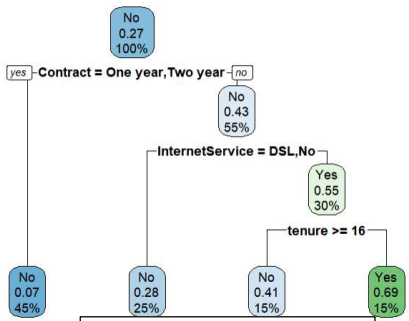
Decision Tree information

Decision tree information gain:

```
> decisionTree_Information
n= 5625

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 5625 1495 No (0.73422222 0.26577778)
   2) Contract=One year,Two year 2532  167 No (0.93404423 0.06595577) *
   3) Contract=Month-to-month 3093 1328 No (0.57064339 0.42935661)
     6) InternetService=DSL,No 1402  399 No (0.71540656 0.28459344)
      12) tenure>=4.5 863  158 No (0.81691773 0.18308227) *
      13) tenure< 4.5 539  241 No (0.55287570 0.44712430)
        26) OnlineSecurity=No internet service,Yes 256   79 No (0.69140625 0.30859375) *
        27) OnlineSecurity=No 283  121 Yes (0.42756184 0.57243816) *
     7) InternetService=Fiber optic 1691  762 Yes (0.45062093 0.54937907)
      14) tenure>=14.5 902  377 No (0.58203991 0.41796009) *
      15) tenure< 14.5 789  237 Yes (0.30038023 0.69961977) *
```

```
> infoCM
Confusion Matrix and Statistics

      Prediction
Actual  No Yes
   No  932 101
   Yes 205 169

               Accuracy : 0.7825
                 95% CI : (0.76, 0.8038)
    No Information Rate : 0.8081
    P-Value [Acc > NIR] : 0.9926

                  Kappa : 0.3886

 Mcnemar's Test P-Value : 3.906e-09

              Precision : 0.9022
                 Recall : 0.8197
                     F1 : 0.8590
             Prevalence : 0.8081
         Detection Rate : 0.6624
   Detection Prevalence : 0.7342
      Balanced Accuracy : 0.7228

       'Positive' Class : No
```

Decision tree pruning:



Decision Tree Prune

```
> CM_prune
Confusion Matrix and Statistics

      Prediction
Actual  No Yes
   No  966  67
   Yes 228 146

               Accuracy : 0.7903
                 95% CI : (0.7681, 0.8113)
    No Information Rate : 0.8486
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3773

 Mcnemar's Test P-Value : <2e-16

              Precision : 0.9351
                 Recall : 0.8090
                     F1 : 0.8675
             Prevalence : 0.8486
         Detection Rate : 0.6866
   Detection Prevalence : 0.7342
      Balanced Accuracy : 0.7472

       'Positive' Class : No
```

According to the previous confusion matrices, the accuracy of pruning is 79.03% which is higher than both of Information decision tree at 78.25% and That of confusion matrix in question 3 at 78.39%.

5

```
  dataset_N$Churn = factor(dataset_N$Churn, level = c("Yes", "No"),
                                    labels = c(0,1))
set.seed(42)
status <- sample.split(Y = dataset_N$Churn, SplitRatio = 0.8)
train <- subset(x = dataset_N, status == TRUE)
test <- subset(x = dataset_N, status == FALSE)

X_train = data.matrix(train[,-20])
y_train = train[,20]

X_test = data.matrix(test[,-20])
y_test = test[,20]

# XGboost
xgboost_train = xgb.DMatrix(data=X_train, label=y_train)
xgboost_test = xgb.DMatrix(data=X_test, label=y_test)

XGBoost_model <- xgboost(data = xgboost_train,
                         max.depth=3, ,
                         nrounds=70)
summary(XGBoost_model)


xgb_pred_test = predict(XGBoost_model, newdata= X_test)
xgb_pred = as.factor((levels(y_test))[round(xgb_pred_test)])
xgb_pred
```

```
xgb_accuracy <- mean(xgb_pred == test$Churn)
print(paste('Accuracy for XGB test is ', xgb_accuracy))
xgb_precision <- posPredValue(xgb_pred, test$Churn, positive="1")
print(paste('precision for XGB test is ',xgb_precision))
xgb_recall <- sensitivity(xgb_pred, test$Churn, positive="1")
print(paste('Recall for XGB test is ',xgb_recall))
xgb_F1 <- (2 * xgb_precision * xgb_recall) / (xgb_precision + xgb_recall)
print(paste('F1-score for XGB test is ',xgb_F1))


xgb_cm = confusionMatrix(y_test, xgb_pred)
print(xgb_cm)


xgb_cm <- confusionMatrix(factor(xgb_pred), factor(y_test), dnn = c("Prediction", "Reference"))
print(xgb_cm)
plot <- as.data.frame(xgb_cm$table)
plot$Prediction <- factor(plot$Prediction, levels=rev(levels(plot$Prediction)))
ggplot(plot, aes(Prediction,Reference, fill= Freq)) +
  geom_tile() + geom_text(aes(label=Freq)) +
  scale_fill_gradient(low="white", high="#009194") +
  labs(x = "Reference",y = "Prediction") +
  scale_x_discrete(labels=c("Class_1","Class_2")) +
  scale_y_discrete(labels=c("Class_1","Class_2"))


ROSE::roc.curve(y_test, xgb_pred_test)
```

XGboost:

```
> summary(XGBoost_model)
                Length Class              Mode
handle               1 xgb.Booster.handle externalptr
raw              85293 -none-             raw
niter                1 -none-             numeric
evaluation_log       2 data.table         list
call                14 -none-             call
params               2 -none-             list
callbacks            2 -none-             list
feature_names       19 -none-             character
nfeatures            1 -none-             numeric
```

```
> xgb_pred
  [1] 0 1 1 1 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1
 [56] 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1
[111] 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[166] 1 1 1 0 1 1 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1
[221] 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0
[276] 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0
[331] 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0
[386] 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 1 1
[441] 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 1
[496] 1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1
[551] 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1
[606] 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0
[661] 0 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 0 1 0 0 1 1 1 1 1 1 1 1
[716] 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1
[771] 0 1 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1
[826] 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 0 0 1
[881] 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1
[936] 0 1 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0
[991] 1 1 1 1 1 0 1 1 1 0
[ reached getOption("max.print") -- omitted 407 entries ]
Levels: 0 1
```

```
> print(xgb_cm)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 193 104
         1 181 929

               Accuracy : 0.7974
                 95% CI : (0.7755, 0.8182)
    No Information Rate : 0.7342
    P-Value [Acc > NIR] : 2.010e-08

                  Kappa : 0.4446

 Mcnemar's Test P-Value : 6.736e-06

            Sensitivity : 0.5160
            Specificity : 0.8993
         Pos Pred Value : 0.6498
         Neg Pred Value : 0.8369
             Prevalence : 0.2658
         Detection Rate : 0.1372
   Detection Prevalence : 0.2111
      Balanced Accuracy : 0.7077

       'Positive' Class : 0
```

```
> xgb_accuracy <- mean(xgb_pred == test$Churn)
> print(paste('Accuracy for XGB test is ', xgb_accuracy))
[1] "Accuracy for XGB test is  0.797441364605544"
> xgb_precision <- posPredValue(xgb_pred, test$Churn, positive="1")
> print(paste('precision for XGB test is ',xgb_precision))
[1] "precision for XGB test is  0.836936936936937"
> xgb_recall <- sensitivity(xgb_pred, test$Churn, positive="1")
> print(paste('Recall for XGB test is ',xgb_recall))
[1] "Recall for XGB test is  0.89932362052275"
> xgb_F1 <- (2 * xgb_precision * xgb_recall) / (xgb_precision + xgb_recall)
> print(paste('F1-score for XGB test is ',xgb_F1))
[1] "F1-score for XGB test is  0.867008866075595"
```

N.B: The model worked just fine with an accuracy of 79.74% which is a reasonable percentage. As a result of that, there's no overfitting signs shown.

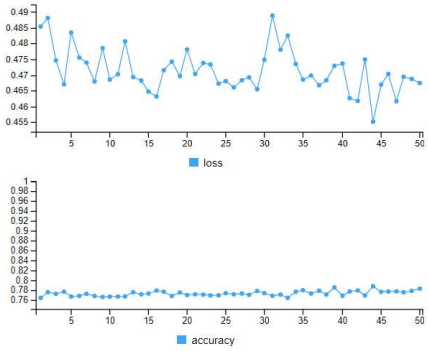
6

```
library(keras)
library(magrittr)
library(reticulate)
library(caTools)
library(tensorflow)

set.seed(123)
DNN_split<- sample.split(Y = dataset_N$Churn, SplitRatio = 0.8)
DNN_train <- subset(x = dataset_N, DNN_split == TRUE)
DNN_test <- subset(x = dataset_N, DNN_split == FALSE)

DNN_X_train = data.matrix(DNN_train[,-20])
DNN_y_train = DNN_train[,20]

DNN_X_test = data.matrix(DNN_test[,-20])
DNN_y_test = DNN_test[,20]

model <- keras_model_sequential()

x_train_keras <- array(DNN_X_train, dim = c(dim(DNN_X_train)[1], prod(dim(DNN_X_train)[-1])))
x_test_keras <- array(DNN_X_test, dim = c(dim(DNN_X_test)[1], prod(dim(DNN_X_test)[-1])))

#One hot encoding
y_train_keras<-to_categorical(DNN_y_train,2)
y_test_keras<-to_categorical(DNN_y_test,2)

model %>%
  layer_dense(units = 128, input_shape = 19) %>%
  layer_dropout(rate=0.3)%>%
  layer_activation(activation = 'tanh') %>%
  layer_dense(units = 64)%>%
  layer_activation(activation = 'tanh')%>%
  layer_dropout(rate=0.3)%>%
  layer_dense(units = 2) %>%
  layer_activation(activation = 'sigmoid')

model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = 'adam',
  metrics = c('accuracy')
)
```

```
#fitting
model %>% fit(x_train_keras, y_train_keras, epochs = 50, batch_size = 128)


#Evaluating model
loss_and_metrics <- model %>% evaluate(x_test_keras, y_test_keras, batch_size = 128)
pred1 <- model %>% predict(x_test_keras)
pred1
pred1 = predict(model,data.matrix(x_test_keras), type = "response")
pred1 <-as.factor(as.numeric(pred1>0.5))
cm_model1 <- confusionMatrix(as.factor(y_test_keras), factor(pred1),
                             mode = "prec_recall", dnn = c("Actual", "Prediction"))
cm_model1

model2 <- keras_model_sequential()
model2 %>%
  layer_dense(units = 128, input_shape = 19) %>%
  layer_dropout(rate=0.3)%>%
  layer_activation(activation = 'relu') %>%
  layer_dense(units = 64)%>%
  layer_activation(activation = 'relu')%>%
  layer_dropout(rate=0.3)%>%
  layer_dense(units = 2) %>%
  layer_activation(activation = 'sigmoid')

#cross entropy
model2 %>% compile(
  loss = 'categorical_crossentropy', optimizer = 'adam',
  metrics = c('accuracy')
)

#fitting
model2 %>% fit(x_train_keras, y_train_keras, epochs = 50, batch_size = 128)
#Evaluation
loss_and_metrics <- model2 %>% evaluate(x_test_keras, y_test_keras, batch_size = 128)


pred2 = predict(model2,data.matrix(x_test_keras), type = "response")
pred2 <-as.factor(as.numeric(pred2>0.5))
```

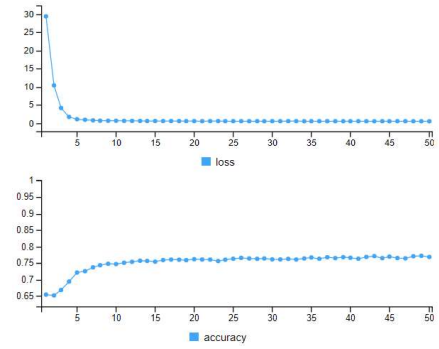
```
# Confusion Matrix
cm_model2 <- confusionMatrix(as.factor(y_test_keras), factor(pred2),
                             mode = "prec_recall", dnn = c("Actual", "Prediction"))
cm_model2
plot_confusionMatrix(cm_model2)


ROSE::roc.curve(y_test_keras, pred2)
```

Model 1 (tanh, tanh, sigmoid):




```
> cm_model1
Confusion Matrix and Statistics

      Prediction
Actual    0    1
     0 1132  275
     1  377 1030

               Accuracy : 0.7683
                 95% CI : (0.7523, 0.7838)
    No Information Rate : 0.5362
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5366

 Mcnemar's Test P-Value : 7.639e-05

              Precision : 0.8045
                 Recall : 0.7502
                     F1 : 0.7764
             Prevalence : 0.5362
         Detection Rate : 0.4023
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.7697

       'Positive' Class : 0
```

Model 2 (RelU, RelU, sigmoid):




```
> cm_model2
Confusion Matrix and Statistics

      Prediction
Actual    0    1
     0  919  488
     1  244 1163

               Accuracy : 0.7399
                 95% CI : (0.7232, 0.756)
    No Information Rate : 0.5867
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4797

 Mcnemar's Test P-Value : < 2.2e-16

              Precision : 0.6532
                 Recall : 0.7902
                     F1 : 0.7152
             Prevalence : 0.4133
         Detection Rate : 0.3266
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.7473

       'Positive' Class : 0
```

After changing the activation function in model 2 the accuracy has decreased to 73.99% from 76.83% in model 1. Also, the precision had a huge decrease from 80.45% to 65.32%.

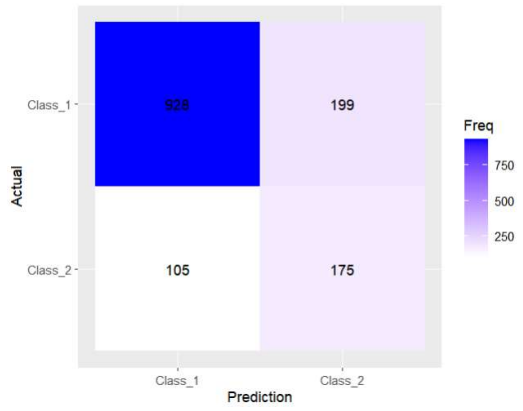
7

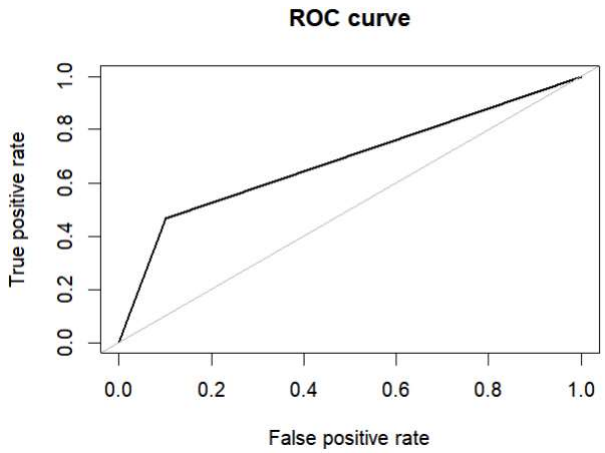
	Precision	Recall	Accuracy	F-Score
Decision Tree Gini	0.8984	0.8234	0.7839	0.8593
Decision Tree Information gain	0.9022	0.8197	0.7825	0.8590
Decision Tree Pruning	0.9351	0.8090	0.7903	0.8675
XGboost	0.8369	0.8993	0.7974	0.8670
DNN – Model 1	0.8045	0.7502	0.7683	0.7764
DNN – Model 2	0.6532	0.7902	0.7399	0.7152
Best Model →	Decision Tree Pruning	XGboost	XGboost	Decision Tree Pruning
Worst Model →	DNN – Model 2	DNN – Model 1	DNN – Model 2	DNN – Model 2

8

Decision Tree gini:

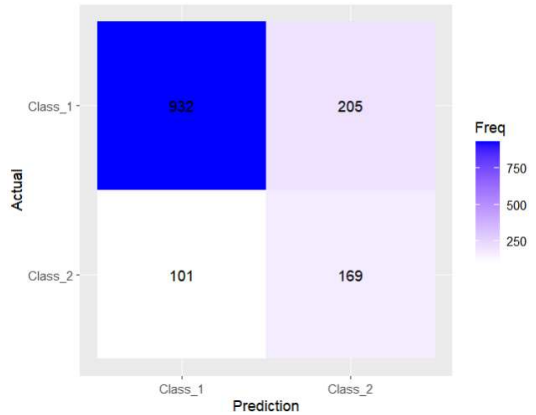
Area under the curve (AUC): 0.683

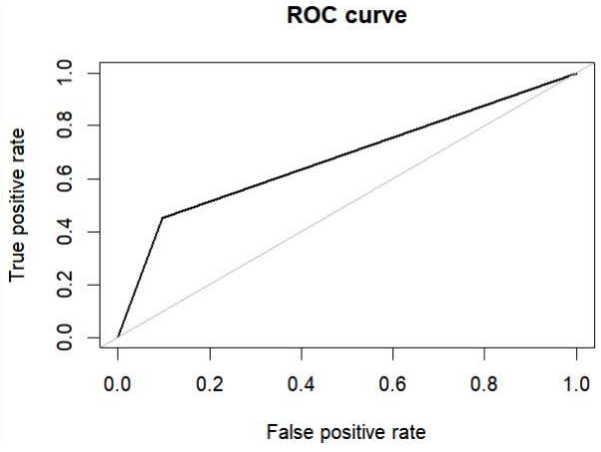




Decision Tree Information gain:

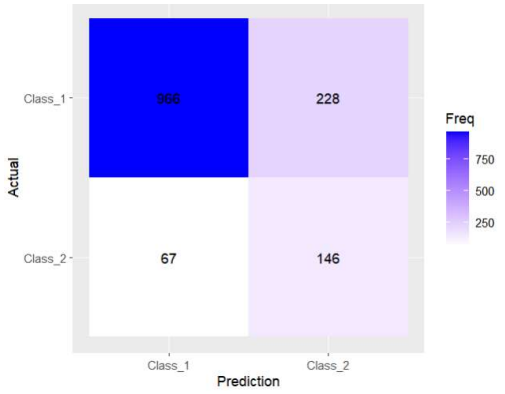
Area under the curve (AUC): 0.677

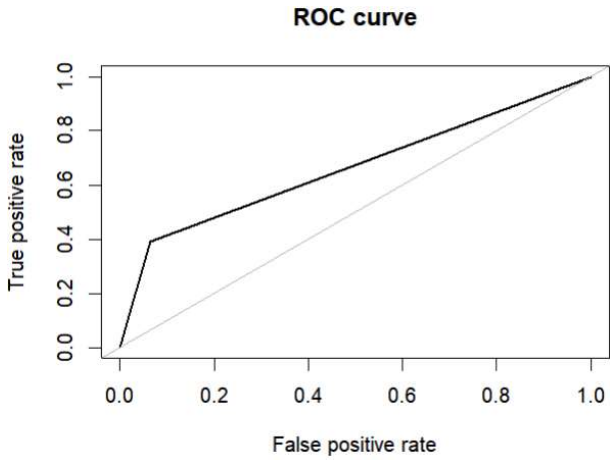




Decision Tree Pruning:

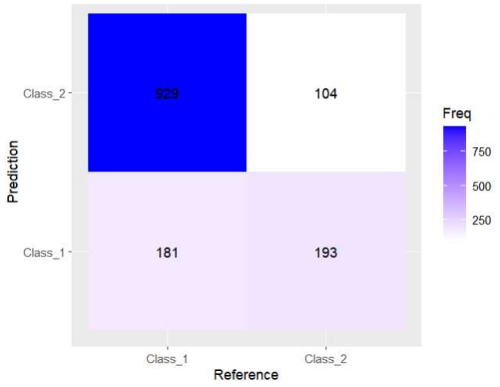
Area under the curve (AUC): 0.663

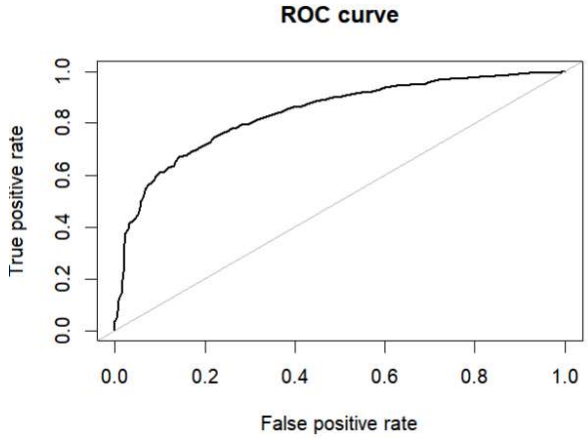




XGboost:

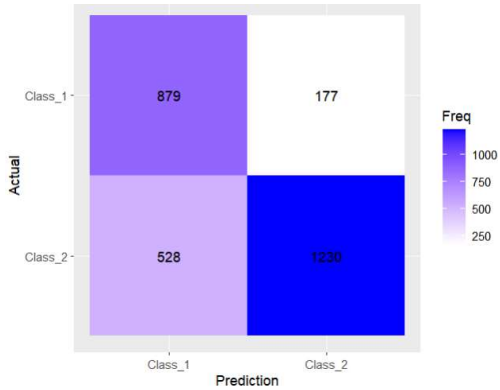
Area under the curve (AUC): 0.839

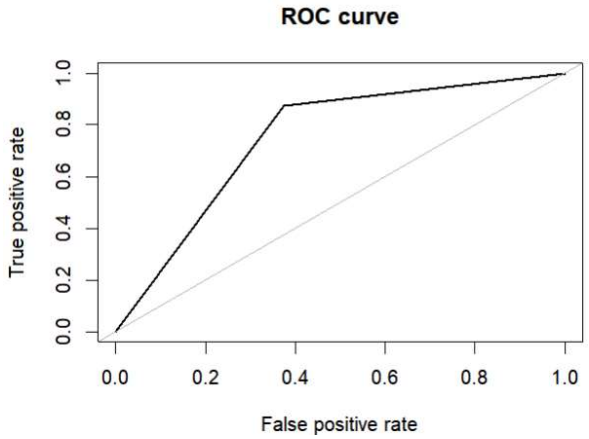




DNN – Model 1:

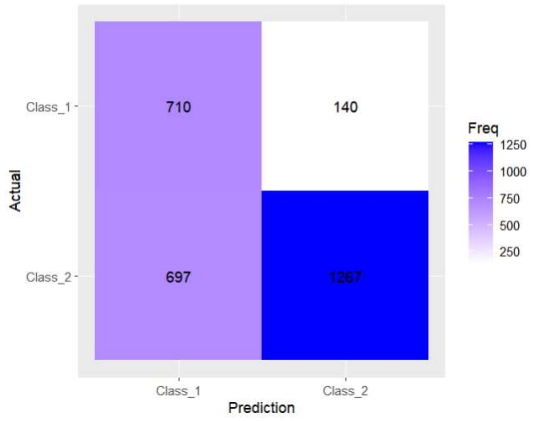
Area under the curve (AUC): 0.747

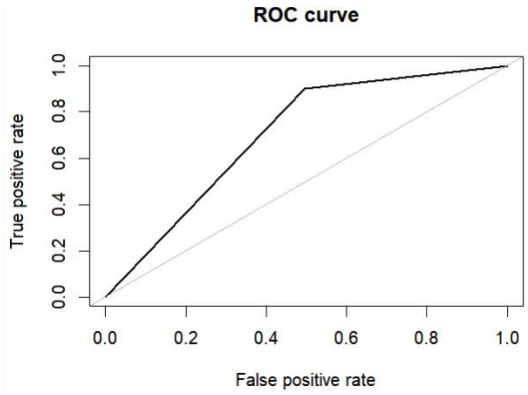




DNN – Model 2:

Area under the curve (AUC): 0.73





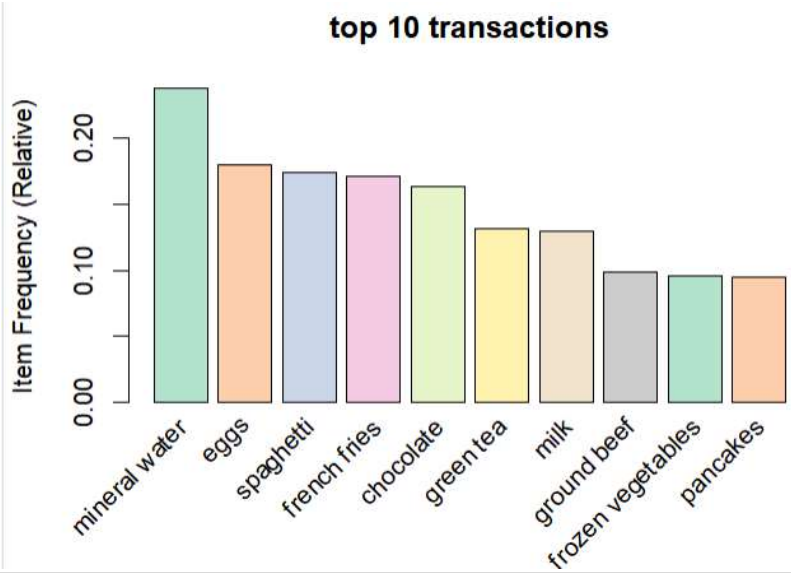
According to question 7 and 8, the XGboost is the best model since it has the best recall, accuracy and AUC.

Part B

A)

```
itemFrequencyPlot(items(dataset), topN = 10,
                  col = brewer.pal(8, 'Pastel2'),
                  main = 'top 10 transactions',
                  type = "relative",
                  ylab = "Item Frequency (Relative)"
                  )
```

Top 10 transactions:



B)

Generating association rule with support of 0.002, minimum confidence of 0.20, and maximum length of 3, also, descending sort by lift.

```
arule_l3 <- sort (apriori(dataset, parameter = list(maxlen=3,support = 0.002, confidence = 0.20)), by="lift")
inspect(arule_l3)
```

```
> inspect(arule_l3)
     lhs                                  rhs                     support     confidence coverage    lift      count
[1]  {escalope, mushroom cream sauce}  => {pasta}                 0.002533333 0.4418605  0.005733333 [1]  28.084352 19
[2]  {escalope, pasta}                 => {mushroom cream sauce} 0.002533333 0.4318182  0.005866667 [2]  22.647807 19
[3]  {mushroom cream sauce, pasta}     => {escalope}             0.002533333 0.9500000  0.002666667 [3]  11.974790 19
[4]  {parmesan cheese, tomatoes}       => {frozen vegetables}    0.002133333 0.6666667  0.003200000 [4]   6.993007 16
[5]  {mineral water, whole wheat pasta}=> {olive oil}            0.003866667 0.4027778  0.009600000 [5]   6.127451 29
[6]  {frozen vegetables, parmesan cheese} => {tomatoes}          0.002133333 0.3902439  0.005466667 [6]   5.705320 16
[7]  {burgers, herb & pepper}          => {ground beef}          0.002266667 0.5483871  0.004133333 [7]   5.580601 17
[8]  {light cream, mineral water}      => {chicken}              0.002400000 0.3272727  0.007333333 [8]   5.454545 18
[9]  {french fries, mushroom cream sauce} => {escalope}          0.002000000 0.4285714  0.004666667 [9]   5.402161 15
[10] {fromage blanc}                   => {honey}                0.003333333 0.2450980  0.013600000 [10]  5.178128 25
[11] {ground beef, shrimp}             => {herb & pepper}        0.002933333 0.2558140  0.011466667 [11]  5.171441 22
[12] {ground beef, low fat yogurt}     => {herb & pepper}        0.002400000 0.2500000  0.009600000 [12]  5.053908 18
[13] {spaghetti, tomato sauce}         => {ground beef}          0.003066667 0.4893617  0.006266667 [13]  4.979936 23
[14] {chocolate, parmesan cheese}      => {frozen vegetables}    0.002000000 0.4687500  0.004266667 [14]  4.916958 15
[15] {meatballs, spaghetti}            => {tomatoes}             0.002133333 0.3333333  0.006400000 [15]  4.873294 16
[16] {chocolate, whole wheat pasta}    => {olive oil}            0.002000000 0.3191489  0.006266667 [16]  4.855207 15
[17] {light cream}                     => {chicken}              0.004533333 0.2905983  0.015600000 [17]  4.843305 34
[18] {frozen vegetables, herb & pepper}=> {ground beef}          0.002800000 0.4666667  0.006000000 [18]  4.748982 21
[19] {mineral water, tomato sauce}     => {ground beef}          0.002666667 0.4651163  0.005733333 [19]  4.733205 20
[20] {pasta}                           => {escalope}             0.005866667 0.3728814  0.015733333 [20]  4.700185 44
[21] {french fries, herb & pepper}     => {ground beef}          0.003200000 0.4615385  0.006933333 [21]  4.696796 24
[22] {cereals, spaghetti}              => {ground beef}          0.003066667 0.4600000  0.006666667 [22]  4.681140 23
[23] {french fries, ground beef}       => {herb & pepper}        0.003200000 0.2307692  0.013866667 [23]  4.665146 24
[24] {cereals, spaghetti}              => {olive oil}            0.002000000 0.3000000  0.006666667 [24]  4.563895 15
[25] {chicken, ground beef}            => {herb & pepper}        0.002133333 0.2253521  0.009466667 [25]  4.555636 16
[26] {grated cheese, ground beef}      => {herb & pepper}        0.002533333 0.2235294  0.011333333 [26]  4.518789 19
[27] {pasta}                           => {shrimp}               0.005066667 0.3220339  0.015733333 [27]  4.514494 38
[28] {chocolate, herb & pepper}        => {ground beef}          0.004000000 0.4411765  0.009066667 [28]  4.489584 30
[29] {chicken, herb & pepper}          => {ground beef}          0.002133333 0.4324324  0.004933333 [29]  4.400601 16
[30] {cake, frozen vegetables}         => {tomatoes}             0.003066667 0.2987013  0.010266667 [30]  4.366978 23
[31] {milk, tomatoes}                  => {soup}                 0.003066667 0.2190476  0.014000000 [31]  4.334715 23
[32] {soup, whole wheat rice}          => {milk}                 0.002000000 0.5555556  0.003600000 [32]  4.286694 15
[33] {herb & pepper, shrimp}           => {ground beef}          0.002933333 0.4150943  0.007066667 [33]  4.224162 22
[34] {eggs, ground beef}               => {herb & pepper}        0.004133333 0.2066667  0.020000000 [34]  4.177898 31
[35] {milk, olive oil}                 => {soup}                 0.003600000 0.2109375  0.017066667 [35]  4.174225 27
[36] {herb & pepper, low fat yogurt}   => {ground beef}          0.002400000 0.4090909  0.005866667 [36]  4.163069 18
[37] {whole wheat pasta}               => {olive oil}            0.008000000 0.2714932  0.029466667 [37]  4.130221 60
[38] {french fries, ham}               => {burgers}              0.002000000 0.3571429  0.005600000 [38]  4.095675 15
[39] {frozen smoothie, shrimp}         => {frozen vegetables}    0.002800000 0.3888889  0.007200000 [39]  4.079254 21
[40] {frozen vegetables, soup}         => {olive oil}            0.002133333 0.2666667  0.008000000 [40]  4.056795 16
```

These are the first 40 combinations of inspection with (escalope, mushroom cream sauce) at the top of the list at “28.084352” .

C)

Generating association rule with support of 0.002, minimum confidence of 0.20, and maximum length of 2, also, descending sort by lift.

```
arule_l2 <- sort (apriori(dataset, parameter = list(maxlen=2,support = 0.002, confidence = 0.20)), by="lift")
inspect(arule_l2)

inspect(arule_l3[1])

inspect(arule_l2[1])
```

```
> inspect(arule_l2)
     lhs                     rhs                support     confidence coverage    lift     count
[1]  {fromage blanc}      => {honey}            0.003333333 0.2450980  0.013600000 5.178128  25
[2]  {light cream}        => {chicken}          0.004533333 0.2905983  0.015600000 4.843305  34
[3]  {pasta}              => {escalope}         0.005866667 0.3728814  0.015733333 4.700185  44
[4]  {pasta}              => {shrimp}           0.005066667 0.3220339  0.015733333 4.514494  38
[5]  {whole wheat pasta}  => {olive oil}        0.008000000 0.2714932  0.029466667 4.130221  60
[6]  {extra dark chocolate} => {chicken}        0.002800000 0.2333333  0.012000000 3.888889  21
[7]  {tomato sauce}       => {ground beef}      0.005333333 0.3773585  0.014133333 3.840147  40
[8]  {mushroom cream sauce} => {escalope}       0.005733333 0.3006993  0.019066667 3.790327  43
[9]  {barbecue sauce}     => {turkey}           0.002533333 0.2345679  0.010800000 3.751086  19
[10] {extra dark chocolate} => {olive oil}      0.002666667 0.2222222  0.012000000 3.380663  20
[11] {herb & pepper}      => {ground beef}      0.016000000 0.3234501  0.049466667 3.291555 120
[12] {gluten free bar}    => {pancakes}         0.002133333 0.3076923  0.006933333 3.236595  16
[13] {shallot}            => {cookies}          0.002000000 0.2586207  0.007733333 3.216675  15
[14] {light cream}        => {olive oil}        0.003200000 0.2051282  0.015600000 3.120612  24
[15] {almonds}            => {burgers}          0.005200000 0.2565789  0.020266667 2.942419  39
[16] {parmesan cheese}    => {frozen vegetables} 0.005466667 0.2751678  0.019866667 2.886375  41
[17] {strong cheese}      => {spaghetti}        0.003733333 0.4827586  0.007733333 2.772350  28
[18] {blueberries}        => {ground beef}      0.002400000 0.2608696  0.009200000 2.654711  18
[19] {bacon}              => {burgers}          0.002000000 0.2307692  0.008666667 2.646436  15
[20] {whole weat flour}   => {pancakes}         0.002266667 0.2463768  0.009200000 2.591621  17
[21] {bacon}              => {pancakes}         0.002133333 0.2461538  0.008666667 2.589276  16
[22] {whole wheat pasta}  => {milk}             0.009866667 0.3348416  0.029466667 2.583655  74
[23] {flax seed}          => {green tea}        0.003066667 0.3382353  0.009066667 2.562389  23
[24] {tomato sauce}       => {spaghetti}        0.006266667 0.4433962  0.014133333 2.546303  47
[25] {pepper}             => {ground beef}      0.006533333 0.2462312  0.026533333 2.505744  49
[26] {green grapes}       => {frozen vegetables} 0.002133333 0.2388060  0.008933333 2.504958  16
[27] {bacon}              => {ground beef}      0.002133333 0.2461538  0.008666667 2.504958  16
[28] {tomatoes}           => {frozen vegetables} 0.016133333 0.2358674  0.068400000 2.474134 121
[29] {shrimp}             => {frozen vegetables} 0.016666667 0.2336449  0.071333333 2.450820 125
[30] {cider}              => {milk}             0.003333333 0.3164557  0.010533333 2.441788  25
[31] {ham}                => {burgers}          0.005600000 0.2110553  0.026533333 2.420359  42
[32] {yams}               => {ground beef}      0.002666667 0.2352941  0.011333333 2.394445  20
[33] {fresh tuna}         => {pancakes}         0.005066667 0.2275449  0.022266667 2.393530  38
[34] {rice}               => {ground beef}      0.004400000 0.2340426  0.018800000 2.381708  33
[35] {burger sauce}       => {spaghetti}        0.002400000 0.4090909  0.005866667 2.349297  18
[36] {light cream}        => {pancakes}         0.003466667 0.2222222  0.015600000 2.337541  26
[37] {cider}              => {burgers}          0.002133333 0.2025316  0.010533333 2.322611  16
[38] {soup}               => {milk}             0.015200000 0.3007916  0.050533333 2.320923 114
[39] {black tea}          => {milk}             0.004266667 0.2990654  0.014266667 2.307604  32
[40] {green beans}        => {spaghetti}        0.003466667 0.4000000  0.008666667 2.297090  26
```

These are the first 40 combinations of inspection with (fromage blanc) at the top of the list at “5.178128”.

```
> inspect(arule_l3[1])
    lhs                                rhs     support     confidence coverage    lift     count
[1] {escalope, mushroom cream sauce} => {pasta} 0.002533333 0.4418605  0.005733333 28.08435 19
>
> inspect(arule_l2[1])
    lhs                rhs     support     confidence coverage lift     count
[1] {fromage blanc} => {honey} 0.003333333 0.245098   0.0136   5.178128 25
> |
```

I)

The greater lift is: association rule (length of 3)

The greater support is: association rule (length of 2)

II)

The greater confidence is: association rule (length of 3)

I would personally prefer association rule with length of 3 since it has higher lift and confidence. Generally, the greater the confidence, the higher rate of customer’s return to buy the same item, nevertheless, the lift illustrates the association between the products in the rule, in other words, the higher the lift, the great the link or correlation between two products.