lab_09

Question 1

Reading in the data ans scaling the predictors

```
fraud_data <- read.csv('creditcard.csv')
scaled_data <- as.data.frame(scale(fraud_data[,-ncol(fraud_data)]))
scaled_data$Fraud <- fraud_data$Fraud
```

Question 2

Explore some visualizations to investigate what predictors may be relevant for detecting fraud.

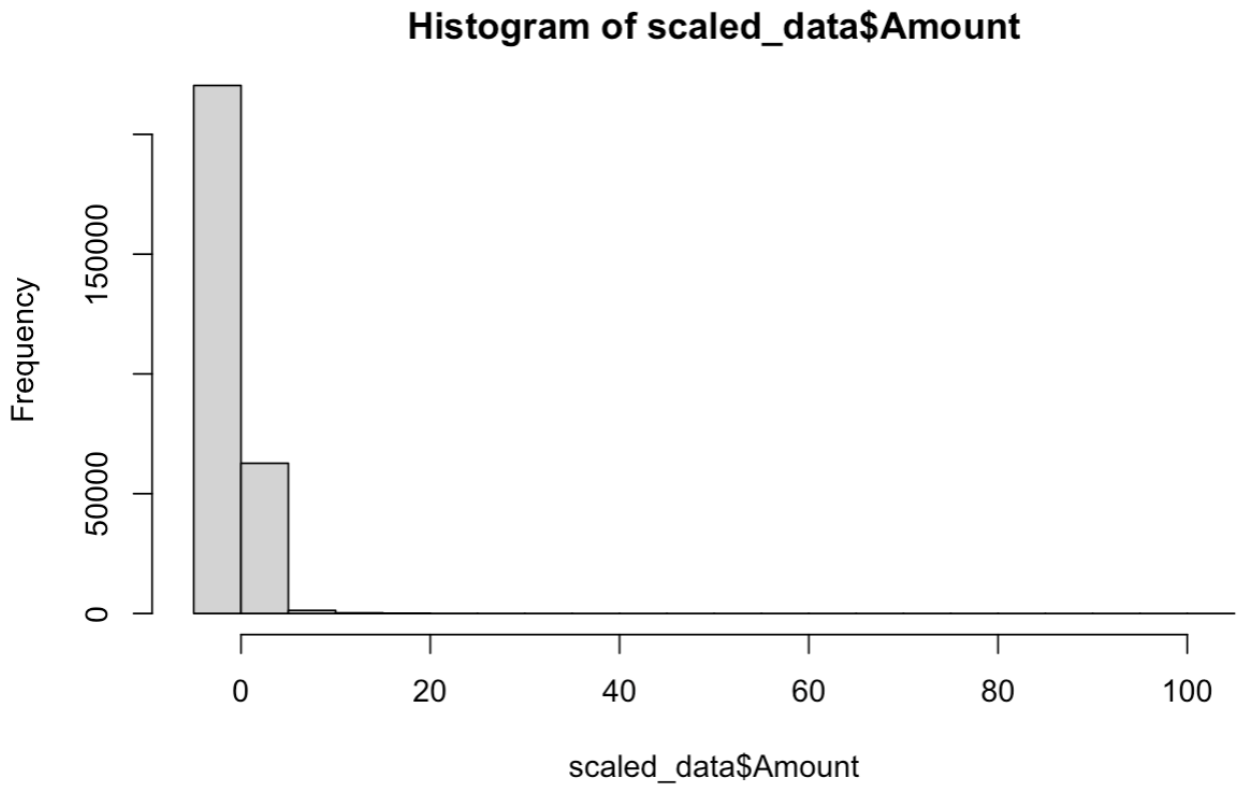
We first explore the balance between Fraud and Not Fraud.

```
table(fraud_data$Fraud)
```

```
     0      1
284315    492
```

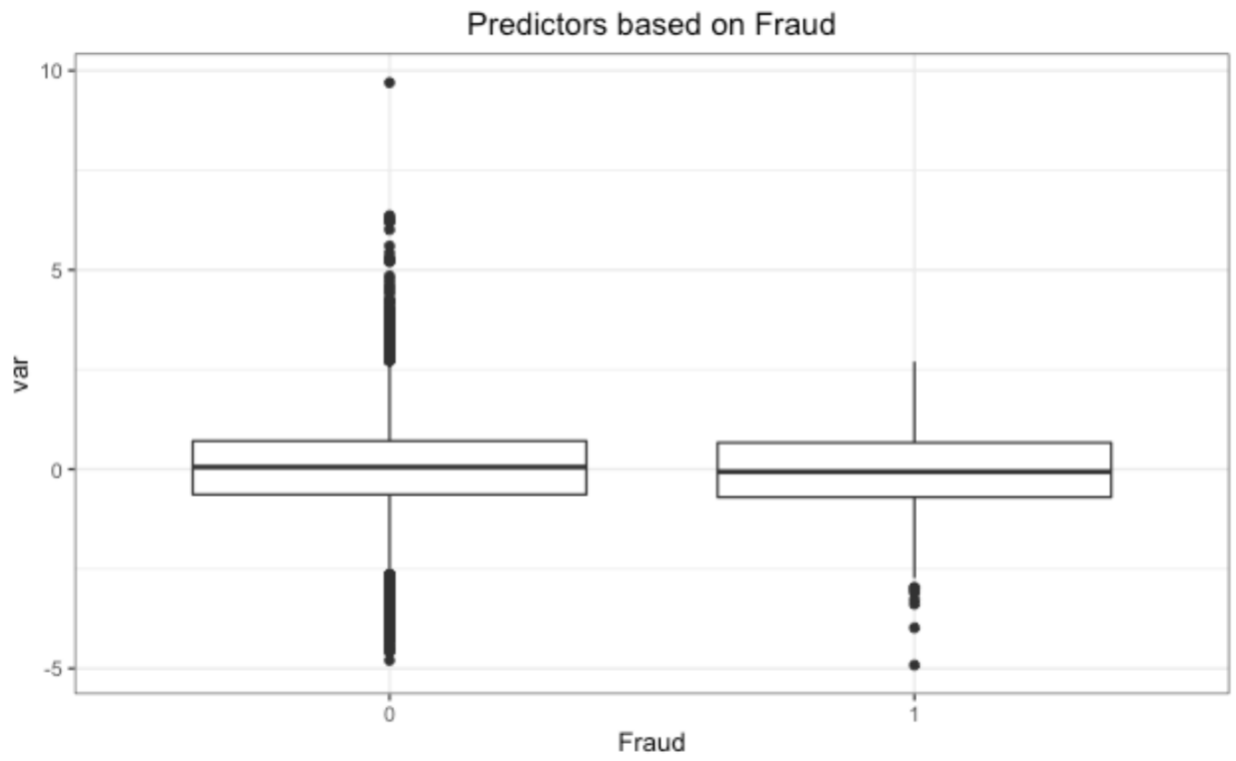
It seems like there are a lot of 0s compared to 1s. We have to keep in mind about this imbalance. We create a histogram of `Amount` because amount of money is typically skewed because of extreme outliers.

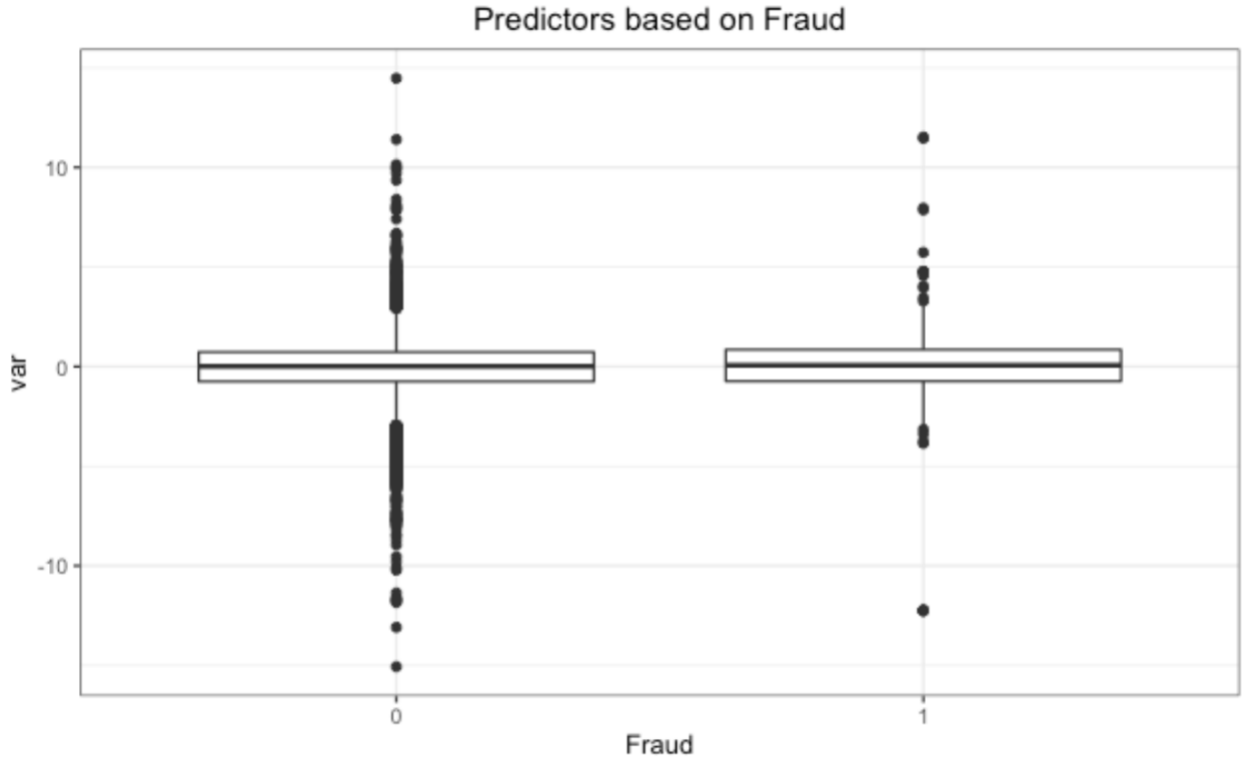
```
hist(scaled_data$Amount)
```

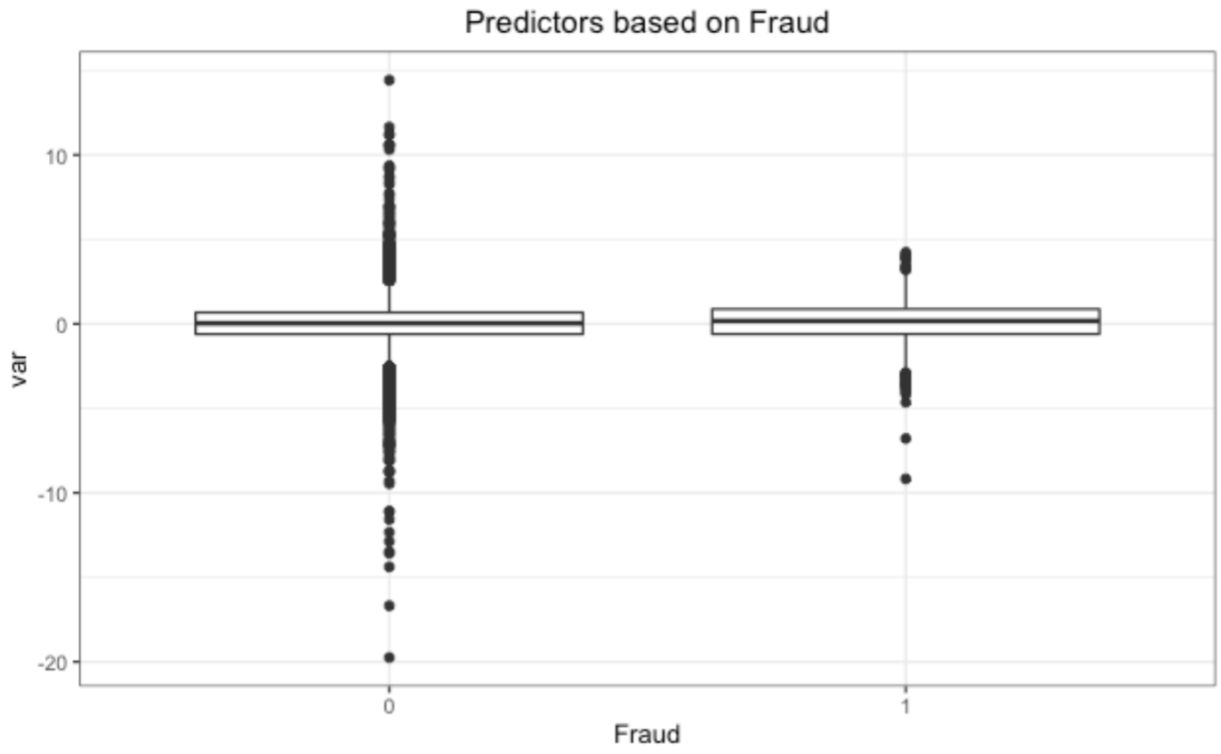


`Amount` is truly skewed as we hypothesize. Therefore, we will make sure to remove these extreme outliers in our training set. We continue to explore the data set by creating 29 boxplots to show the relationship between `Fraud` and other variables to see how other variables are good predictors.

```
explore_data <- function(var) {
  ggplot(scaled_data, aes(as.factor(Fraud), var)) +
  geom_boxplot() +
  labs(title = "Predictors based on Fraud", x = "Fraud") +
  theme_bw() +
  theme(plot.title = element_text(hjust=0.5))
}

for (i in 1:29){
  print(explore_data(scaled_data[,i]))
}
```


Predictors based on Fraud
10
5
0
-5
var
0
1
Fraud


Predictors based on Fraud
10
0
-10
var
0
1
Fraud



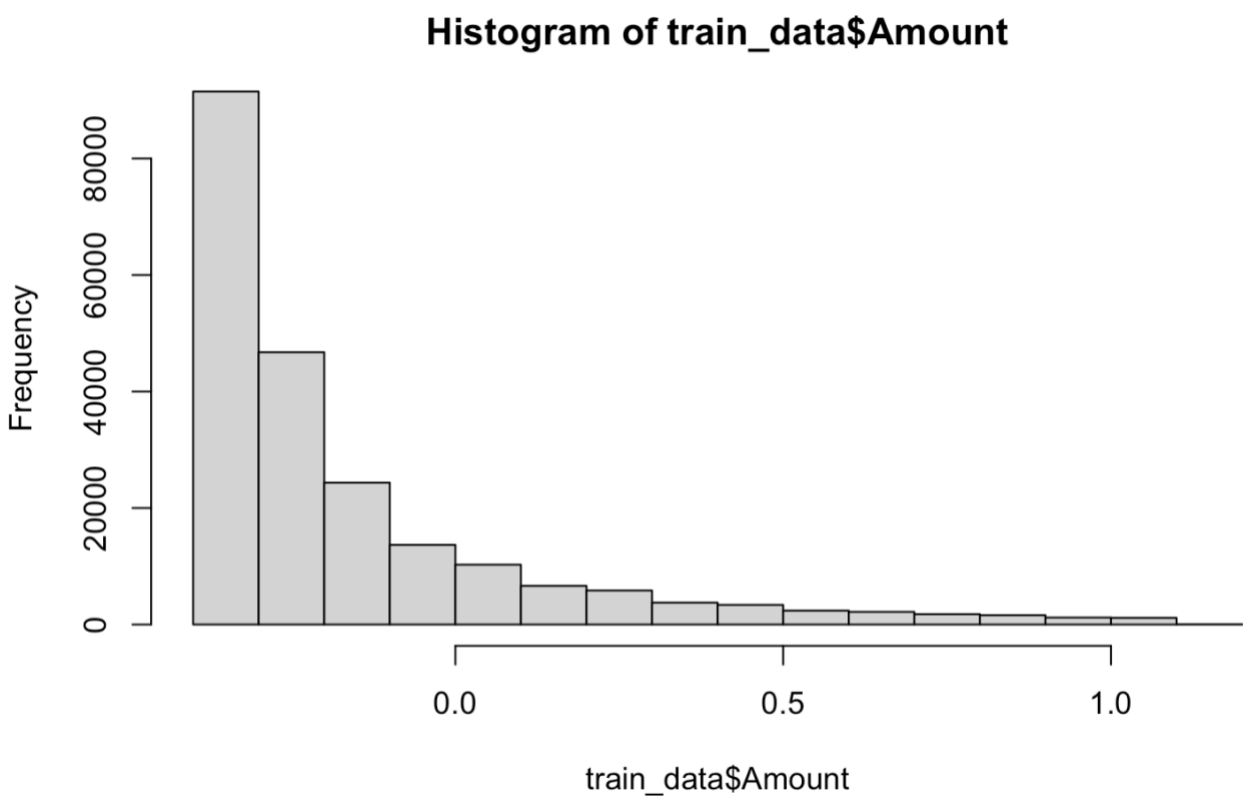
Based on the box plots above, there are many predictors that are not very useful in predicting as there are no differences between Fraud and Not Fraud using those predictors, such as V15, V22, V25. For some other variables, Not Fraud has such a large range compared to Fraud (because of the imbalance) that it's hard to see the relationship clearly. However, we choose not to remove any variables because neural network needs a large amount of data and we only have 29 predictors.

Question 3

Split the data 70/30 intro training and test sets

We also take only 95% of the data based on `Amount` to remove some outliers. The histogram below shows the distribution of `Amount` after we remove those. It is still not very good but it is clearly better than before.

```
set.seed(1)
idx <- createDataPartition(y = scaled_data$Amount, p = 0.8, list = FALSE)
train_data <- scaled_data[idx,]
test_data <- scaled_data[-idx,]

train_data = subset(train_data, Amount < quantile(train_data$Amount,.95))
hist(train_data$Amount)
```



Train a neural network with 1 hidden layer with 5 nodes and linear activation functions

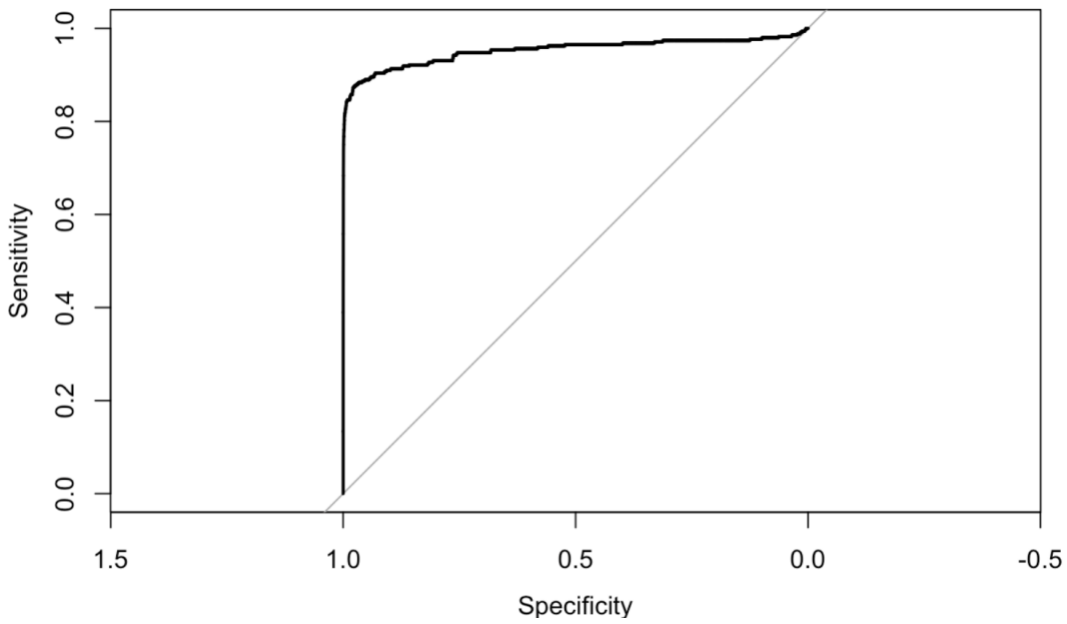
```
model <- keras_model_sequential() %>%  #Initiates model, no need to change
  layer_flatten(input_shape = c(29)) %>%  # Layer flatten specifies the input layer.  The '2' is the number of predictor variables provided
  layer_dense(units = 5, activation = "linear") %>% #Units is how many nodes in this layer
  layer_dense(units = 1, activation = "sigmoid") #Last layer is always treated as output layer

summary(model)
```

```
model %>%
  compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = "accuracy")

model %>%
  fit(
    x = as.matrix(train_data[,-ncol(train_data)]), y = train_data[,"Fraud"],
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )
```

Generate the valid / fraudulent predictions on the training data with a cutoff Δ=0.5 and use the confusionMatrix function to compare the predictions to the true outcomes

```
report <- function(data, model, delta = 0.5) {
  model_pred <- model %>%
    predict(as.matrix(data[,-ncol(data)]))

  defaultROC = roc(data$Fraud, model_pred)
  print(plot.roc(defaultROC))
  #print(paste('AUC is',defaultROC$auc))
  model_pred <- ifelse(model_pred >= delta,1,0)
  print(confusionMatrix(as.factor(model_pred), as.factor(data$Fraud), mode =
'everything', positive = '1'))
}

model <- load_model_tf("model")

report(train_data, model)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 344 cases (data$Fraud 1).
Area under the curve: 0.9523
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 216095    209
         1     15    135

               Accuracy : 0.999
                 95% CI : (0.9988, 0.9991)
    No Information Rate : 0.9984
    P-Value [Acc > NIR] : 3.021e-12

                  Kappa : 0.5461

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


            Sensitivity : 0.3924419
            Specificity : 0.9999306
         Pos Pred Value : 0.9000000
         Neg Pred Value : 0.9990338
              Precision : 0.9000000
                 Recall : 0.3924419
                     F1 : 0.5465587
             Prevalence : 0.0015893
         Detection Rate : 0.0006237
   Detection Prevalence : 0.0006930
      Balanced Accuracy : 0.6961862

       'Positive' Class : 1
```



The Accuracy is 0.999, which is high. According to the confusion matrix, TP = 216095, FP = 15, FN = 209, TN = 135. There are too many FNs. This makes Sensitivity low (0.392). Because the model mostly predicts 0, the Kappa score is not very good (0.5461). In general, AUC = 0.9523. This is great but there are obviously rooms for improvement.

Question 4

Balance the input data by implementing some sampling methods: downsampling, upsampling, ROSE and SMOTE

```
down_train <- downSample(x = train_data[,-ncol(train_data)], y = as.factor(tr
ain_data$Fraud))
down_train <- rename(down_train, Fraud = Class)
down_train <- down_train[sample(1:nrow(down_train)),]      #shuffle the data
again

up_train <- upSample(x = train_data[,-ncol(train_data)], y = as.factor(train_
data$Fraud))
up_train <- rename(up_train, Fraud = Class)
up_train <- up_train[sample(1:nrow(up_train)),]            #shuffle the data
again


rose_train <- ROSE(Fraud ~., data = train_data)$data
rose_train <- rose_train[sample(1:nrow(rose_train)),]      #shuffle the data
again

smote_train <- SMOTE(train_data[,-ncol(train_data)],train_data$Fraud)$data
smote_train <- rename(smote_train, Fraud = class)
smote_train <- smote_train[sample(1:nrow(smote_train)),]   #shuffle the data
again
```

```
table(down_train$Fraud)
table(up_train$Fraud)
table(rose_train$Fraud)
table(smote_train$Fraud)
```

```
  0   1 
344 344 
```

```
     0      1 
216110 216110 
```

```
     0      1 
108468 107986 
```

```
     0      1 
216110 216032 
```

Experiment with choice of parameters for training the neural network.

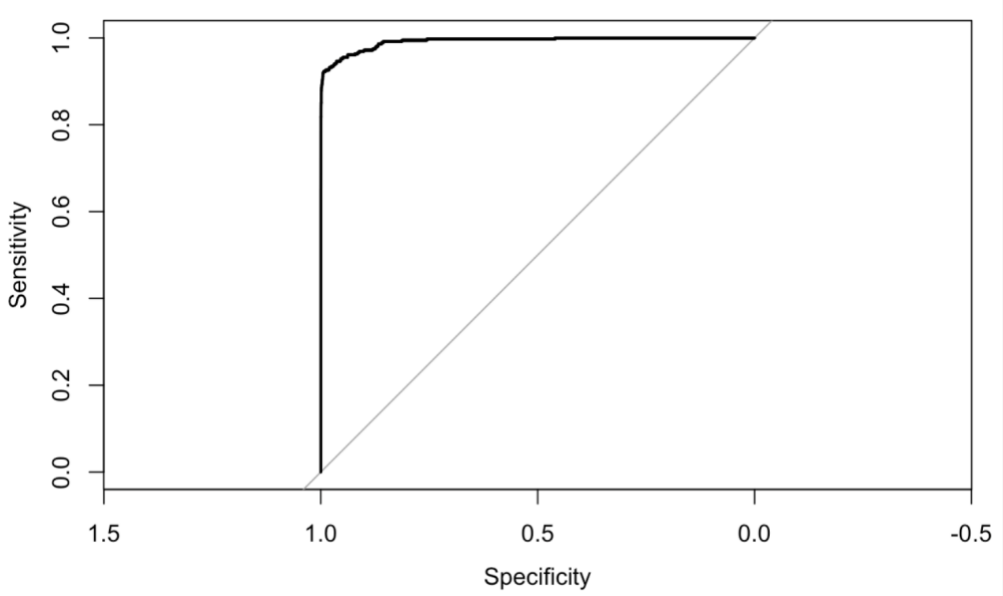
We first try training the neural network with different choices of sampling dataset to see which one works better.

```
up_train$Fraud <- as.integer(as.character(up_train$Fraud))
model_up <- keras_model_sequential() %>%  #Initiates model, no need to change
  layer_flatten(input_shape = c(29)) %>%  # Layer flatten specifies the input layer.  The '2' is the number of predictor variables provided
  layer_dense(units = 5, activation = "linear") %>% #Units is how many nodes in this layer
  layer_dense(units = 1, activation = "sigmoid") #Last layer is always treated as output layer

model_up %>%
  compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = "accuracy")

model_up %>%
  fit(
    x = as.matrix(up_train[,-ncol(up_train)]), y = up_train[,"Fraud"],
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )
```

```
model_up <- load_model_tf("model_up")
report(up_train,model_up)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 216110 cases (data$Fraud 1).
Area under the curve: 0.9921
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 212624  16016
         1   3486 200094
```

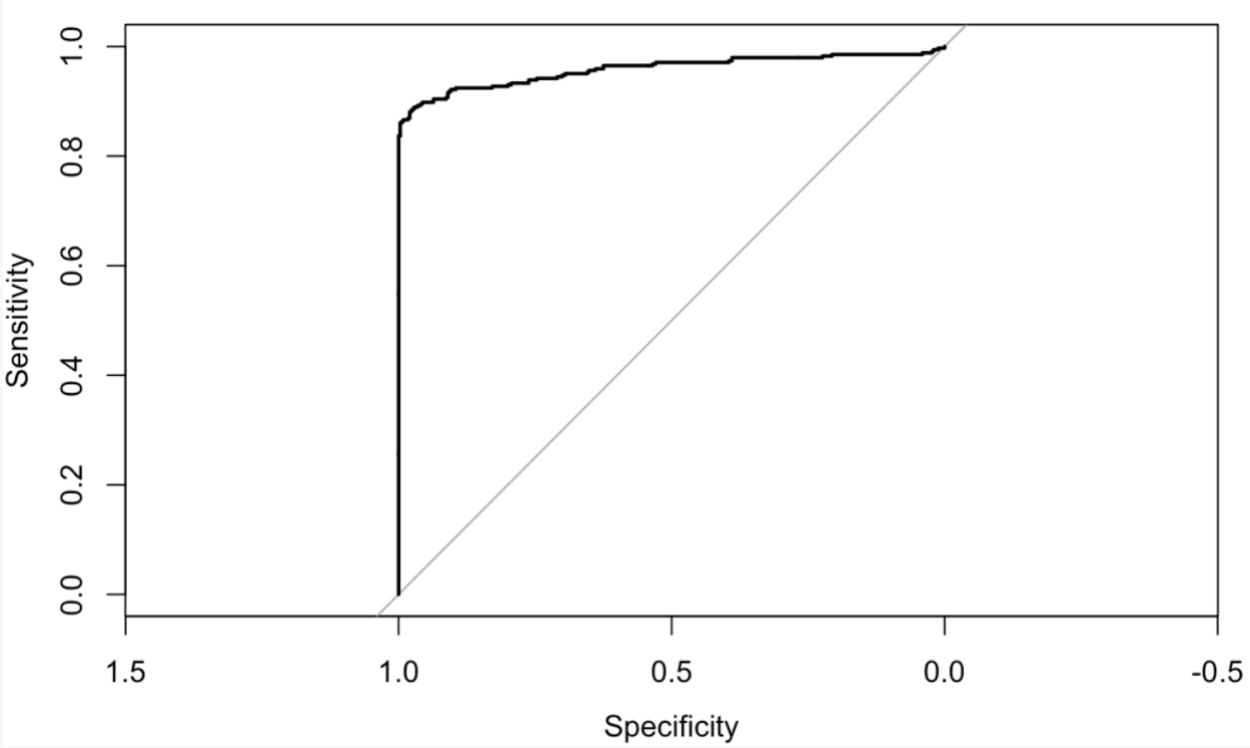
```
               Accuracy : 0.9549
                 95% CI : (0.9543, 0.9555)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9098

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

            Sensitivity : 0.9259
            Specificity : 0.9839
         Pos Pred Value : 0.9829
         Neg Pred Value : 0.9300
              Precision : 0.9829
                 Recall : 0.9259
                     F1 : 0.9535
             Prevalence : 0.5000
         Detection Rate : 0.4629
   Detection Prevalence : 0.4710
      Balanced Accuracy : 0.9549

       'Positive' Class : 1
```




```
set.seed(1)
down_train$Fraud <- as.integer(as.character(down_train$Fraud))
model_down <- keras_model_sequential() %>%  #Initiates model, no need to chan
ge
  layer_flatten(input_shape = c(29)) %>%  # Layer flatten specifies the input
layer.  The '2' is the number of predictor variables provided
  layer_dense(units = 5, activation = "linear") %>% #Units is how many nodes
in this layer
  layer_dense(units = 1, activation = "sigmoid") #Last layer is always treate
d as output layer

model_down %>%
  compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = "accuracy")

model_down %>%
  fit(
    x = as.matrix(down_train[,-ncol(down_train)]), y = down_train[,"Fraud"],
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )

model_down <- load_model_tf("model_down")
report(down_train, model_down)
```

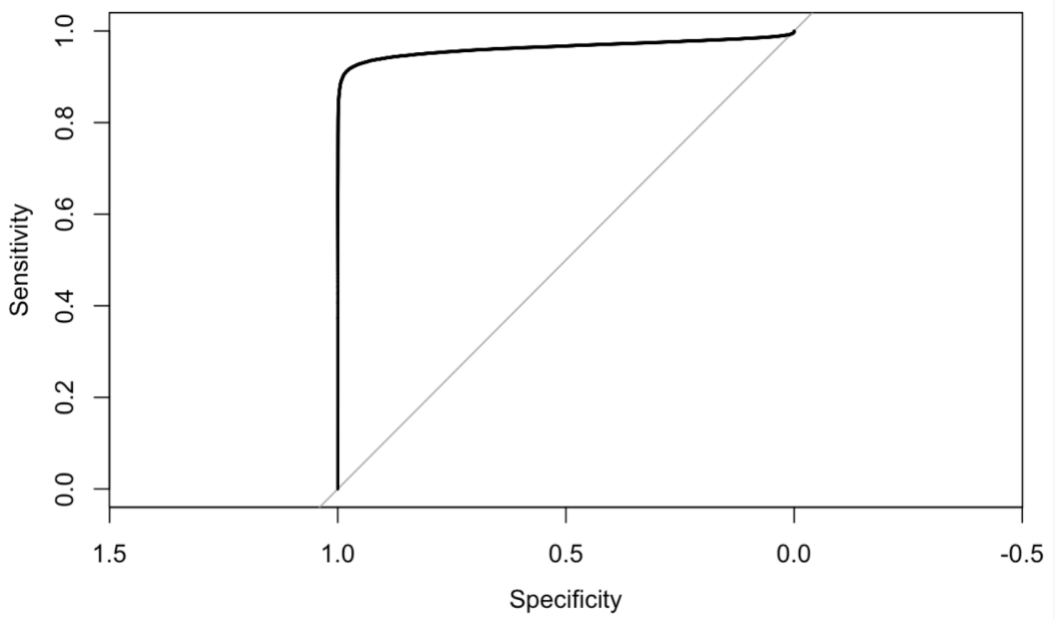
```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 344 controls (data$Fraud 0) < 344 cases (data$Fraud 1).
Area under the curve: 0.9583
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 290  26
         1  54 318

               Accuracy : 0.8837
                 95% CI : (0.8574, 0.9067)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7674
```

```
 Mcnemar's Test P-Value : 0.002539

            Sensitivity : 0.9244
            Specificity : 0.8430
         Pos Pred Value : 0.8548
         Neg Pred Value : 0.9177
              Precision : 0.8548
                 Recall : 0.9244
                     F1 : 0.8883
             Prevalence : 0.5000
         Detection Rate : 0.4622
   Detection Prevalence : 0.5407
      Balanced Accuracy : 0.8837

       'Positive' Class : 1
```




```
rose_train$Fraud <- as.integer(as.character(rose_train$Fraud))
model_rose <- keras_model_sequential() %>%  #Initiates model, no need to chan
ge
  layer_flatten(input_shape = c(29)) %>%  # Layer flatten specifies the input
layer.  The '2' is the number of predictor variables provided
  layer_dense(units = 5, activation = "linear") %>% #Units is how many nodes
in this layer
  layer_dense(units = 1, activation = "sigmoid") #Last layer is always treate
d as output layer

model_rose %>%
  compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = "accuracy")

model_rose %>%
  fit(
    x = as.matrix(rose_train[,-ncol(rose_train)]), y = rose_train[,"Fraud"],
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )


model_rose <- load_model_tf("model_rose")
report(rose_train,model_rose)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 108468 controls (data$Fraud 0) < 107986 cases (data$Fraud 1).
Area under the curve: 0.9638
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 106871  10111
         1   1597  97875

               Accuracy : 0.9459
                 95% CI : (0.9449, 0.9469)
    No Information Rate : 0.5011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8918

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

            Sensitivity : 0.9064
            Specificity : 0.9853
         Pos Pred Value : 0.9839
         Neg Pred Value : 0.9136
              Precision : 0.9839
                 Recall : 0.9064
                     F1 : 0.9436
             Prevalence : 0.4989
         Detection Rate : 0.4522
   Detection Prevalence : 0.4596
      Balanced Accuracy : 0.9458

       'Positive' Class : 1
```




```
smote_train$Fraud <- as.integer(as.character(smote_train$Fraud))
model_smote <- keras_model_sequential() %>%  #Initiates model, no need to cha
nge
  layer_flatten(input_shape = c(29)) %>%  # Layer flatten specifies the input
layer.  The '2' is the number of predictor variables provided
  layer_dense(units = 5, activation = "linear") %>% #Units is how many nodes
in this layer
  layer_dense(units = 1, activation = "sigmoid") #Last layer is always treate
d as output layer

model_smote %>%
  compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = "accuracy")

model_smote %>%
  fit(
    x = as.matrix(smote_train[,-ncol(smote_train)]), y = smote_train[,"Fraud"
],
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )

model_smote <- load_model_tf("model_smote")
report(smote_train,model_smote)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 216032 cases (data$Fraud 1).
Area under the curve: 0.9924
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 212357  16072
         1   3753 199960

               Accuracy : 0.9541
                 95% CI : (0.9535, 0.9547)
    No Information Rate : 0.5001
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9082
```

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

            Sensitivity : 0.9256
            Specificity : 0.9826
         Pos Pred Value : 0.9816
         Neg Pred Value : 0.9296
              Precision : 0.9816
                 Recall : 0.9256
                     F1 : 0.9528
             Prevalence : 0.4999
         Detection Rate : 0.4627
   Detection Prevalence : 0.4714
      Balanced Accuracy : 0.9541

       'Positive' Class : 1
```

```
dt = data.frame(Model = c('model_up', 'model_down', 'model_rose', 'model_smot
e'), Accuracy = c(0.9549, 0.8837, 0.9459, 0.9541), Kappa = c(0.9098, 0.7674,
0.8918, 0.9082), Sens = c(0.9259, 0.9244, 0.9064, 0.9256), Specs = c(0.9839,
0.8430, 0.9853, 0.9826), AUC = c(0.9921, 0.9583, 0.9638, 0.9924))
kable(dt)
```

Model	Accuracy	Kappa	Sens	Specs	AUC
model_up	0.9549	0.9098	0.9259	0.9839	0.9921
model_down	0.8837	0.7674	0.9244	0.8430	0.9583
model_rose	0.9459	0.8918	0.9064	0.9853	0.9638
model_smote	0.9541	0.9082	0.9256	0.9826	0.9924

Upsampling and SMOTE data sets clearly perform better with higher Accuracy (0.9549 and 0.9541), Kappa (0.9098 and 0.9082), AUC (0.9921 and 0.9924) and so on. Therefore, we will move forward with Upsampling and SMOTE data sets.

We will now train the neural network with more hidden layers. Instead of the linear function we used previously, we choose a nonlineear function in order to capture more complexity in the data. Particularly, we choose ReLU over two other non-linear activation function (sigmoid and tan-h). This is because sigmoid and tan-h activation functions often suffer from the vanishing gradient problem which causes slow convergence, and the final model generated by these two activation functions works not very well for too high or too low X. We have `Amount` as our variable which has many too high and too low observations so using sigmoid and tan-h is clearly not a good choice.

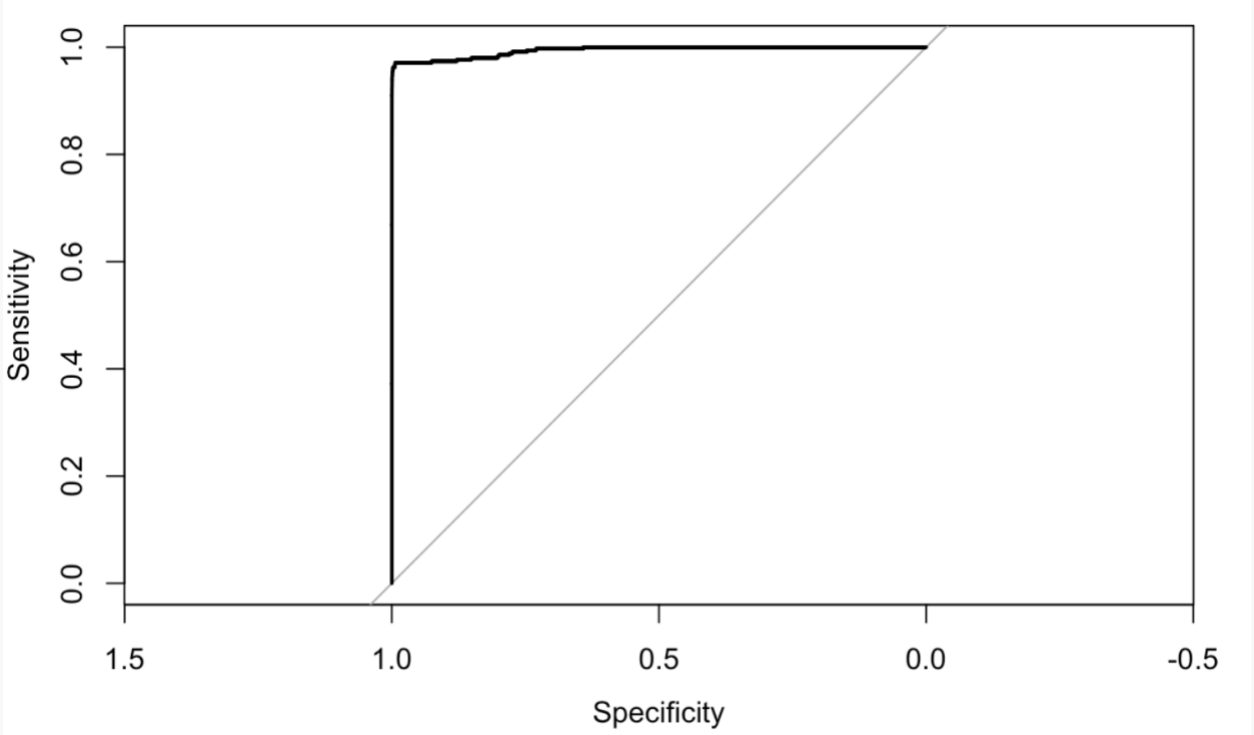
We also use 50 nodes and two hidden layers instead of 5 nodes and 1 hidden layer because more nodes and hidden layers result in better models. However, too many nodes and hidden layers for a simple data set with only 29 variables and around 200,000 observations are also not good. 50 nodes and 2 hidden layers are decent numbers.

```
model2 <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(29)) %>%
  layer_dense(units = 50, activation = 'relu') %>%
  layer_dense(units = 50, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')

summary(model2)

model2 %>%
  compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = 'accuracy'
)

model2 %>%
  fit(
    x = as.matrix(train_data[,-ncol(train_data)]), y = train_data$Fraud,
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )
```

```
model2 <- load_model_tf("model2")
report(train_data, model2)
```

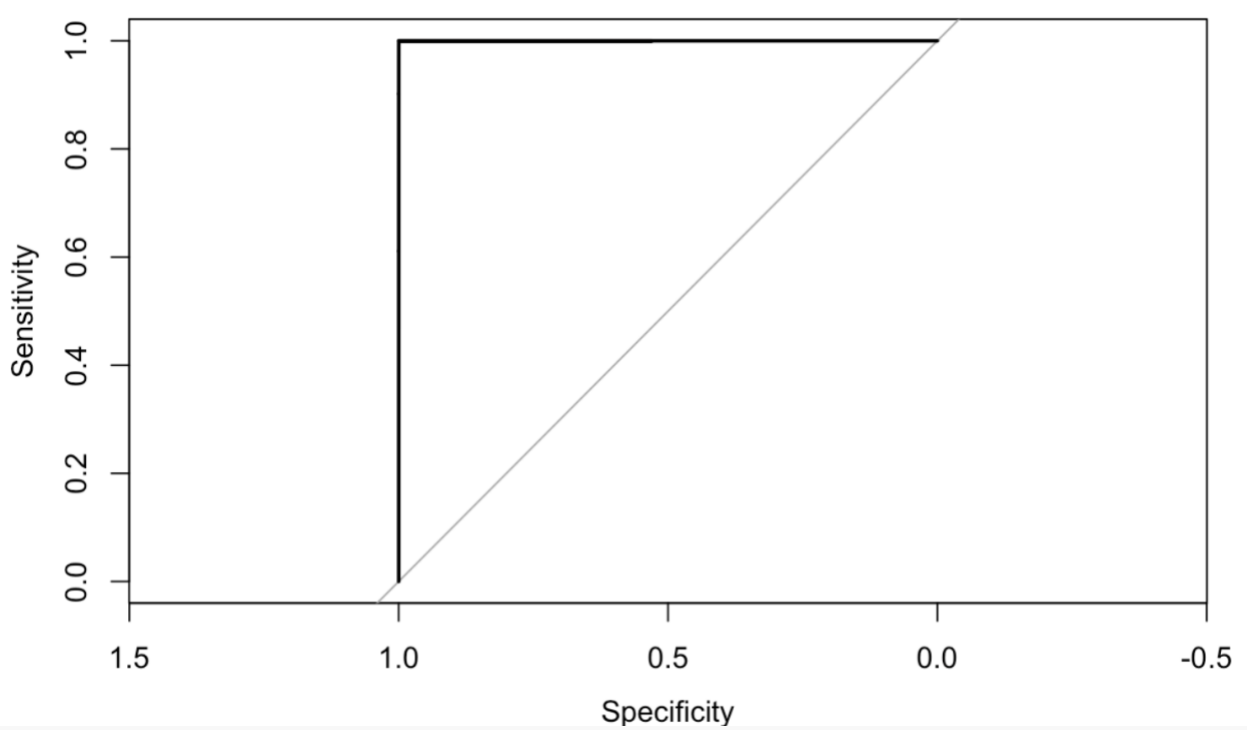
```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 344 cases (data$Fraud 1).
Area under the curve: 0.9939
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 216107     50
         1      3    294

               Accuracy : 0.9998
                 95% CI : (0.9997, 0.9998)
    No Information Rate : 0.9984
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9172
```

```
                  Kappa : 0.9172

Mcnemar's Test P-Value : 2.64e-10

            Sensitivity : 0.854651
            Specificity : 0.999986
         Pos Pred Value : 0.989899
         Neg Pred Value : 0.999769
              Precision : 0.989899
                 Recall : 0.854651
                     F1 : 0.917317
             Prevalence : 0.001589
         Detection Rate : 0.001358
   Detection Prevalence : 0.001372
      Balanced Accuracy : 0.927319

       'Positive' Class : 1
```



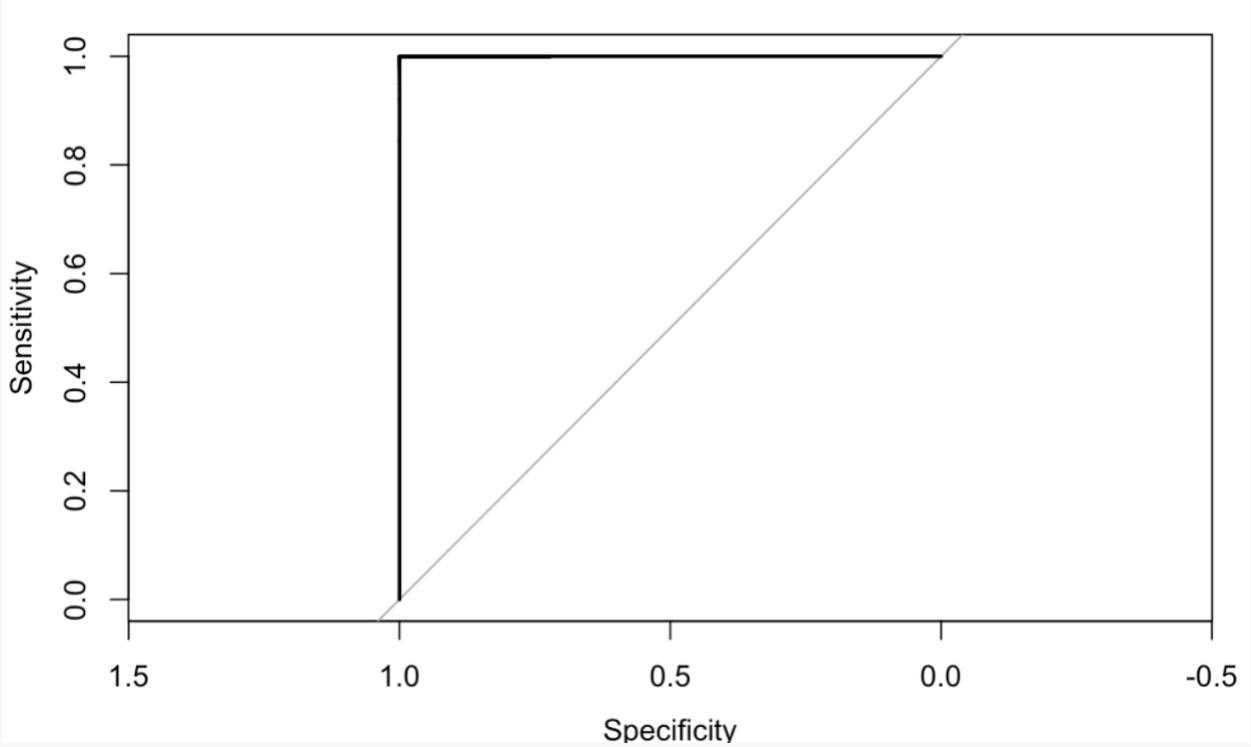

```
model2_up <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(29)) %>%
  layer_dense(units = 50, activation = 'relu') %>%
  layer_dense(units = 50, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')

summary(model2_up)

model2_up %>%
  compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = 'accuracy'
)

model2_up %>%
  fit(
    x = as.matrix(up_train[,-ncol(up_train)]), y = up_train$Fraud,
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )

model2_up <- load_model_tf("model2_up")
report(up_train, model2_up)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 216110 cases (data$Fraud 1).
Area under the curve: 1
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 216057      0
         1     53 216110

               Accuracy : 0.9999
                 95% CI : (0.9998, 0.9999)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9998

 Mcnemar's Test P-Value : 9.148e-13

            Sensitivity : 1.0000
            Specificity : 0.9998
         Pos Pred Value : 0.9998
         Neg Pred Value : 1.0000
              Precision : 0.9998
                 Recall : 1.0000
                     F1 : 0.9999
             Prevalence : 0.5000
         Detection Rate : 0.5000
   Detection Prevalence : 0.5001
      Balanced Accuracy : 0.9999

       'Positive' Class : 1
```




```
model2_smote <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(29)) %>%
  layer_dense(units = 50, activation = 'relu') %>%
  layer_dense(units = 50, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')

model2_smote %>%
  compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = 'accuracy'
)

model2_smote %>%
  fit(
    x = as.matrix(smote_train[,-ncol(smote_train)]), y = smote_train$Fraud,
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )

model2_smote <- load_model_tf("model2_smote")
report(smote_train, model2_smote)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 216032 cases (data$Fraud 1).
Area under the curve: 1
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 216013      0
         1     97 216032

               Accuracy : 0.9998
                 95% CI : (0.9997, 0.9998)
    No Information Rate : 0.5001
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9996

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

            Sensitivity : 1.0000
            Specificity : 0.9996
         Pos Pred Value : 0.9996
         Neg Pred Value : 1.0000
              Precision : 0.9996
                 Recall : 1.0000
                     F1 : 0.9998
             Prevalence : 0.4999
         Detection Rate : 0.4999
   Detection Prevalence : 0.5001
      Balanced Accuracy : 0.9998

       'Positive' Class : 1
```




```
dt2 = data.frame(Model = c('model2', 'model2_up', 'model2_smote'), Accuracy =
c(0.9998, 0.9999, 0.9998), Kappa = c(0.9172, 0.9998, 0.9996), Sens = c(0.8546
51, 1.0000, 1.0000), Specs = c(0.999986, 0.9998, 0.9996), AUC = c(0.9939, 1.0
000, 1.0000))

dt2 <- rbind(dt,dt2)
kable(dt2)
```

Model	Accuracy	Kappa	Sens	Specs	AUC
model_up	0.9549	0.9098	0.925900	0.983900	0.9921
model_down	0.8837	0.7674	0.924400	0.843000	0.9583
model_rose	0.9459	0.8918	0.906400	0.985300	0.9638
model_smote	0.9541	0.9082	0.925600	0.982600	0.9924
model2	0.9998	0.9172	0.854651	0.999986	0.9939
model2_up	0.9999	0.9998	1.000000	0.999800	1.0000
model2_smote	0.9998	0.9996	1.000000	0.999600	1.0000

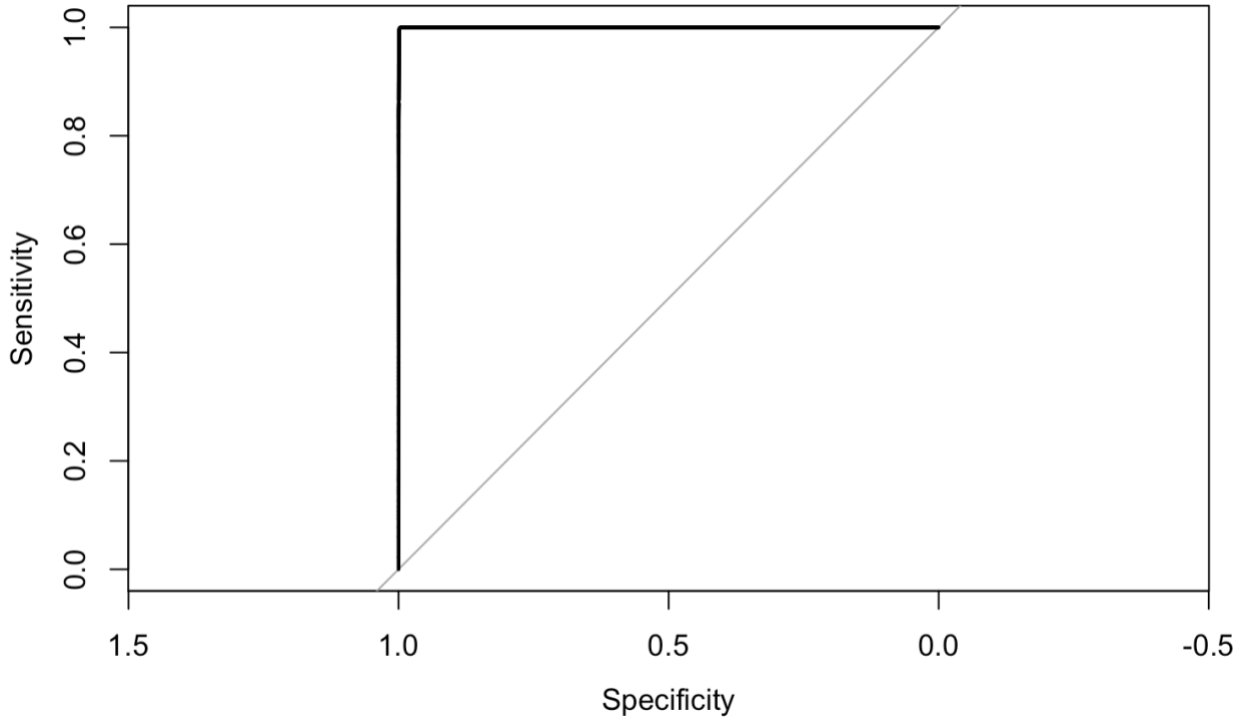
The metrics are definitely better with Accuracy increasing from 0.95 to 0.99, Kappa from 0.9 to 0.99, Sensitivity from 0.92 to 1, Specs from 0.98 to 0.99, and AUC from 0.992 to 1. Kappa, Accuracy decrease but AUC 0.9834) increases a little bit. However, to avoid overfitting as neural network is a complex method and our data set is quite small compared

to other large data sets that people train a neural network on, we will try using regularization.

```
model3_smote <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(29)) %>%
  layer_dense(units = 50, activation = 'relu', kernel_regularizer = regulariz
er_l2(0.01)) %>%
  layer_dense(units = 50, activation = 'relu', kernel_regularizer = regulariz
er_l2(0.01)) %>%
  layer_dense(units = 1, activation = 'sigmoid')

model3_smote %>%
  compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = 'accuracy'
)

model3_smote %>%
  fit(
    x = as.matrix(smote_train[,-ncol(smote_train)]), y = smote_train$Fraud,
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )

model3_smote <- load_model_tf('model3_smote')
report(smote_train, model3_smote)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 216032 cases (data$Fraud 1).
Area under the curve: 0.9998
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 215643    324
         1    467 215708

               Accuracy : 0.9982
                 95% CI : (0.998, 0.9983)
    No Information Rate : 0.5001
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9963
```

```
                Kappa : 0.9963

 Mcnemar's Test P-Value : 4.443e-07

            Sensitivity : 0.9985
            Specificity : 0.9978
         Pos Pred Value : 0.9978
         Neg Pred Value : 0.9985
              Precision : 0.9978
                 Recall : 0.9985
                     F1 : 0.9982
             Prevalence : 0.4999
         Detection Rate : 0.4992
   Detection Prevalence : 0.5002
      Balanced Accuracy : 0.9982

       'Positive' Class : 1
```

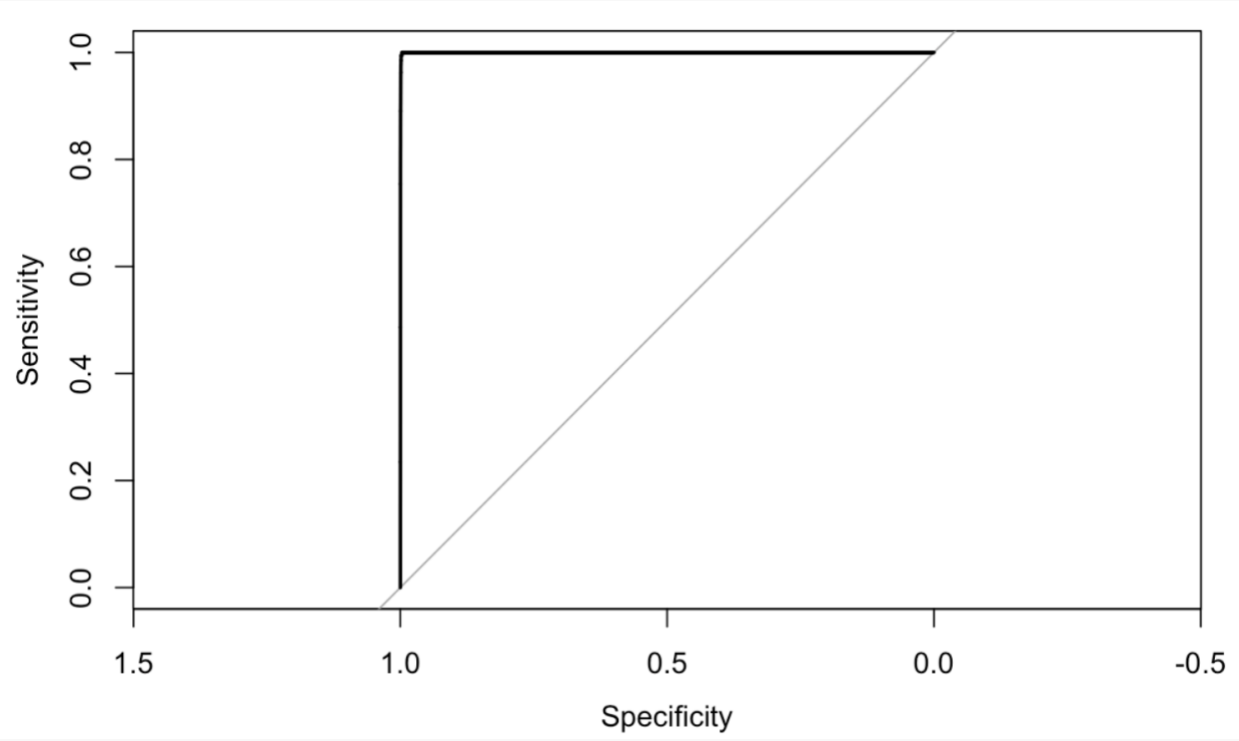



```
model3_up <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(29)) %>%
  layer_dense(units = 50, activation = 'relu', kernel_regularizer = regulariz
er_l2(0.01)) %>%
  layer_dense(units = 50, activation = 'relu', kernel_regularizer = regulariz
er_l2(0.01)) %>%
  layer_dense(units = 1, activation = 'sigmoid')

model3_up %>%
  compile(
    loss = 'binary_crossentropy',
```

```
    optimizer = 'adam',
    metrics = 'accuracy'
)

model3_up %>%
  fit(
    x = as.matrix(up_train[,-ncol(up_train)]), y = up_train$Fraud,
    epochs = 20,
    validation_split = 0.3,
    verbose = 2
  )

model3_up <- load_model_tf('model3_up')
report(up_train, model3_up)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 216110 controls (data$Fraud 0) < 216110 cases (data$Fraud 1).
Area under the curve: 0.9998
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 215015      0
         1   1095 216110

               Accuracy : 0.9975
                 95% CI : (0.9973, 0.9976)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9949

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

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

            Sensitivity : 1.0000
            Specificity : 0.9949
         Pos Pred Value : 0.9950
         Neg Pred Value : 1.0000
              Precision : 0.9950
                 Recall : 1.0000
                     F1 : 0.9975
             Prevalence : 0.5000
         Detection Rate : 0.5000
   Detection Prevalence : 0.5025
      Balanced Accuracy : 0.9975

       'Positive' Class : 1
```




```
dt3 <- data.frame(Model = c('model3_smote', 'model3_up'), Accuracy = c(0.9982
, 0.9975), Kappa = c(0.9963, 0.9949), Sens = c(0.9985, 1.0000), Specs = c(0.9
978, 0.9949), AUC = c(0.9998, 0.9998))
dt3 <- rbind(dt2,dt3)
kable(dt3)
```

Model	Accuracy	Kappa	Sens	Specs	AUC
model_up	0.9549	0.9098	0.925900	0.983900	0.9921
model_down	0.8837	0.7674	0.924400	0.843000	0.9583
model_rose	0.9459	0.8918	0.906400	0.985300	0.9638
model_smote	0.9541	0.9082	0.925600	0.982600	0.9924
model2	0.9998	0.9172	0.854651	0.999986	0.9939
model2_up	0.9999	0.9998	1.000000	0.999800	1.0000
model2_smote	0.9998	0.9996	1.000000	0.999600	1.0000
model3_smote	0.9982	0.9963	0.998500	0.997800	0.9998
model3_up	0.9975	0.9949	1.000000	0.994900	0.9998

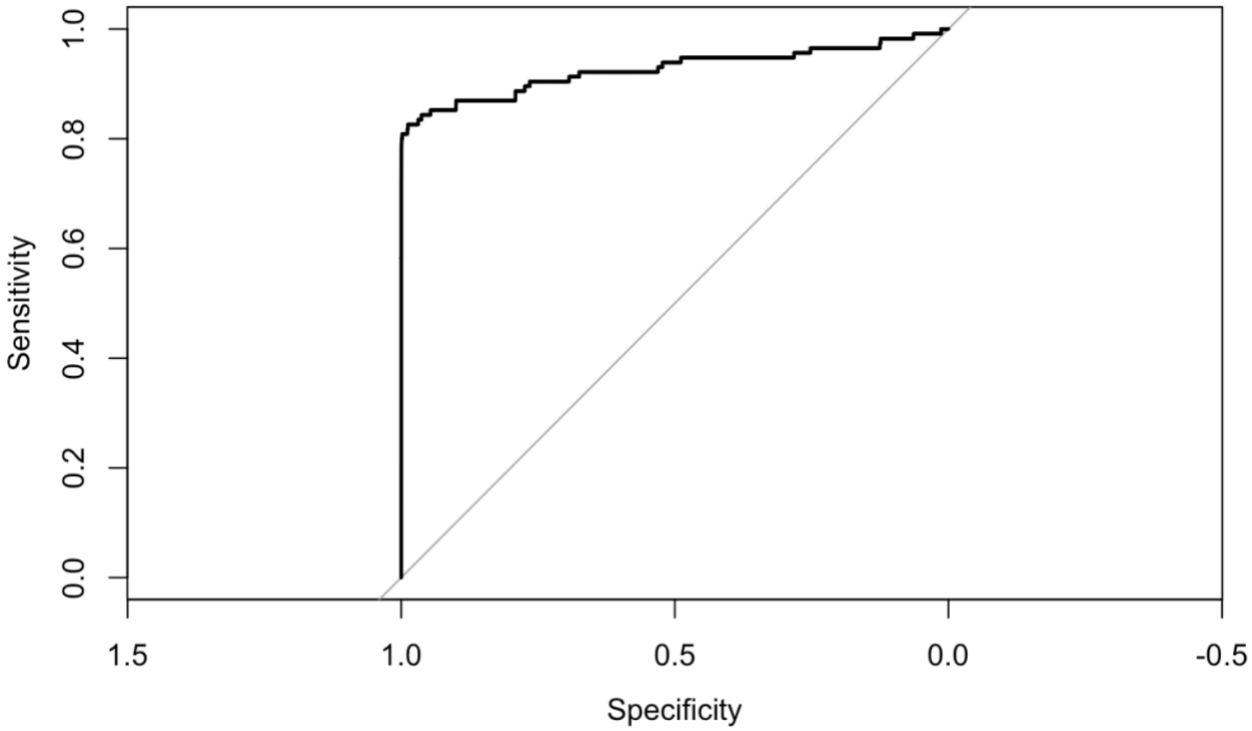
The performance gets a little bit worse. However, it's just about 0.001 and it's better to avoid overfitting than to be better for just about 0.001. Between `model3_smote` and `model3_up`, we decide to choose `model3_smote` because SMOTE creates fictious data that resembles the real data better than upsampling which just duplicates the observations.

Question 5

Use the final model to generate valid / fraudulent predictions on the (imbalanced) test set. Show the confusion matrix. Discuss the accuracy, sensitivity, specificity, and kappa values. We choose the test data that has been scaled because it is easy to scale the new data point to get a better result. In the original data before scaling, `Amount` has too much weight on the result.

```
report(test_data, model3_smote)
```

```
Call:
roc.default(response = data$Fraud, predictor = model_pred)

Data: model_pred in 56845 controls (data$Fraud 0) < 115 cases (data$Fraud 1).
Area under the curve: 0.9264
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 56718    22
         1   127    93

               Accuracy : 0.9974
                 95% CI : (0.9969, 0.9978)
    No Information Rate : 0.998
    P-Value [Acc > NIR] : 0.999

                  Kappa : 0.554

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

            Sensitivity : 0.808696
            Specificity : 0.997766
         Pos Pred Value : 0.422727
         Neg Pred Value : 0.999612
              Precision : 0.422727
                 Recall : 0.808696
                     F1 : 0.555224
             Prevalence : 0.002019
         Detection Rate : 0.001633
   Detection Prevalence : 0.003862
      Balanced Accuracy : 0.903231

       'Positive' Class : 1
```



According to the confusion matrix above, the Accuracy score is 0.9974 which seems to be good. However, the Kappa score is only 0.554. This means that the model is better than random guessing but not as good as we would expect. This may be because the model predicts too many '0' compared to '1'. Because of the class imbalance, the model can just predict all 0s and still get a really good accuracy. Kappa helps us point out that this may be the case and we may want to lower our threshold. The Sensitivity score is 0.81 and Specificity is 0.99. These scores indicate that there are many FPs compared to FNs. This is totally fine. In this case, it is better to be too careful than to lose money from credit card fraud. The loss of FPs is much lower than FNs.

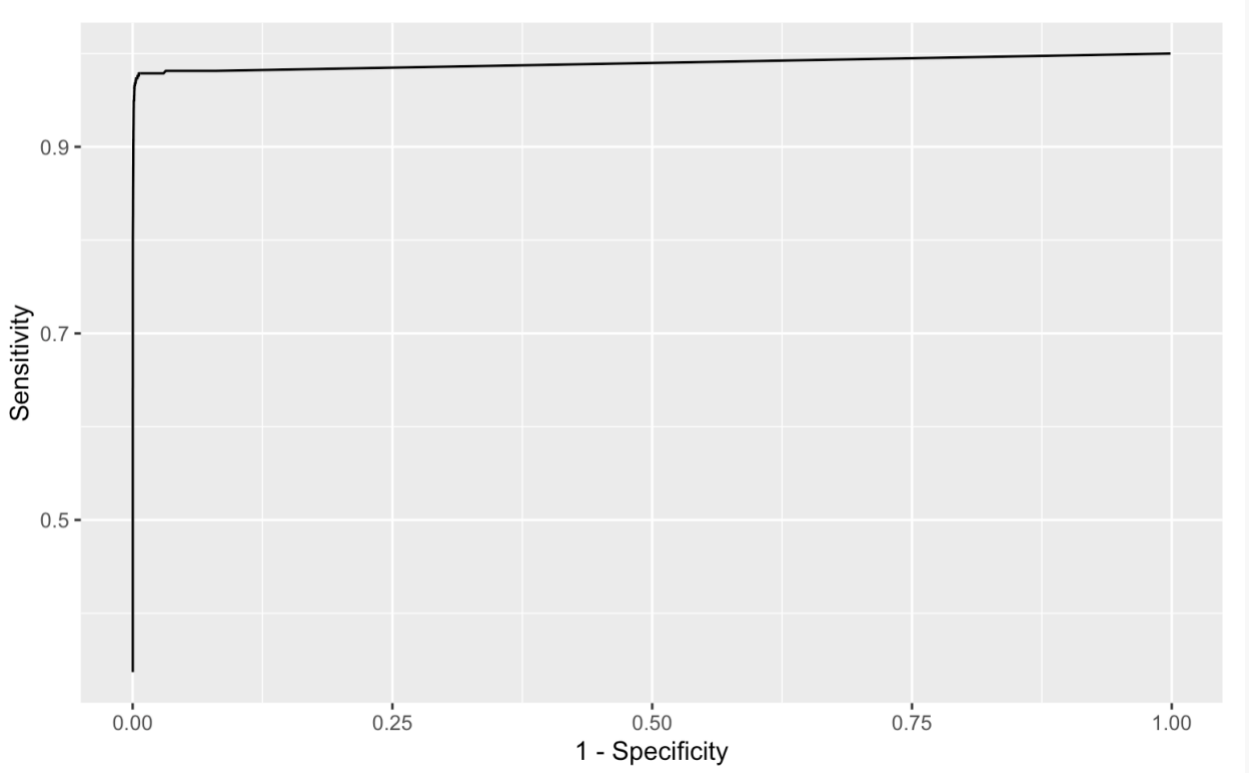
Question 6

Assume you get a job with a major national bank - in a practical business sense, how might we estimate the costs associated with a false positive and a false negative when detecting credit card fraud?

The costs of FNs can be estimated using average costs of credit card fraud in the past taken into account the inflation rates each year. The costs of FPs are more of an implicit cost (costs that are hard to quantify because they do not directly show up on our expense but affect the business in a long run due to reduced customer satisfaction). Therefore, we may want to first get the customer satisfaction data by asking customers to rate their satisfaction after the transaction on a scale of 1-5. We may then consult the sales team to quantify each rating. For example, 5 corresponds to $0 loss, 4 corresponds to $5 loss and so on.

Generate the probability predictions from your model on each observation of the original, unmodified training data. As we did in Lab 5, consider a range of probability cutoffs and create a table with the loss associated with each cutoff.

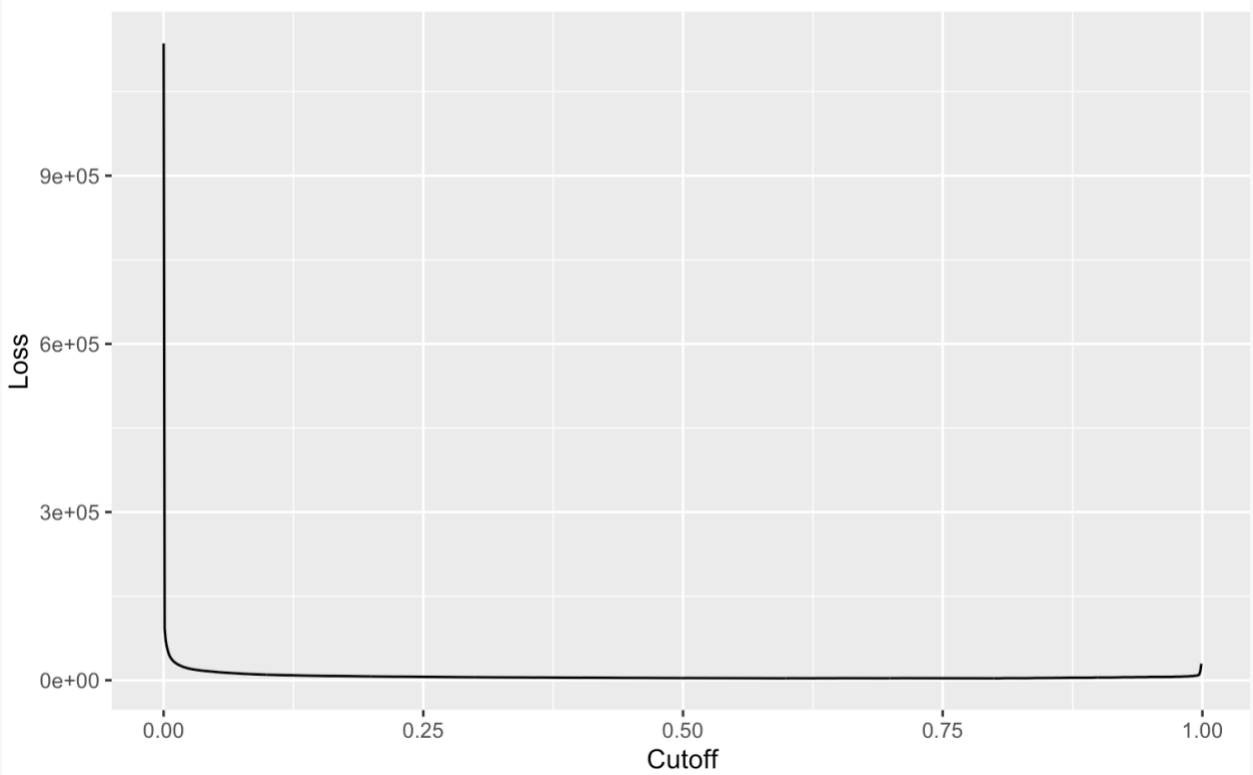
```
fraud_data[idx,] %>%
  filter(Fraud == 1) %>%
  summarise(mean = mean(Amount))

lossFN <- 118.1702
lossFP <- 5


unmodified_train_data <- scaled_data[idx,]

predicted_prob <- model3_smote %>%
    predict(as.matrix(unmodified_train_data[,-ncol(unmodified_train_data)]),
type = 'prob')
default_prob = as.data.frame(predicted_prob)$V1

cutoff <- seq(min(default_prob), max(default_prob),0.001)
performance = setNames(data.frame(matrix(ncol = 8, nrow = length(cutoff))), c
("Cutoff", "TN", "FN", "TP", "FP", "Sensitivity", "Specificity", "Accuracy"))
performance$Cutoff = cutoff
for (i in 1:length(cutoff)) {
  temp = table(default_prob > performance$Cutoff[i], unmodified_train_data$Fr
aud)
  TN = temp[1,1]
  FN = temp[1,2]
  FP = temp[2,1]
  TP = temp[2,2]
  performance$TN[i] = TN
  performance$TP[i] = TP
  performance$FN[i] = FN
  performance$FP[i] = FP
  performance$Sensitivity[i] = TP/(FN+TP)
  performance$Specificity[i] = TN/(TN+FP)
  performance$Accuracy[i] = (TP+TN)/(FP+FN+TP+TN)
}

ggplot(performance,aes(1-Specificity,Sensitivity))+
  geom_line()
```




```
table(default_prob > performance$Cutoff[i], unmodified_train_data$Fraud)
```

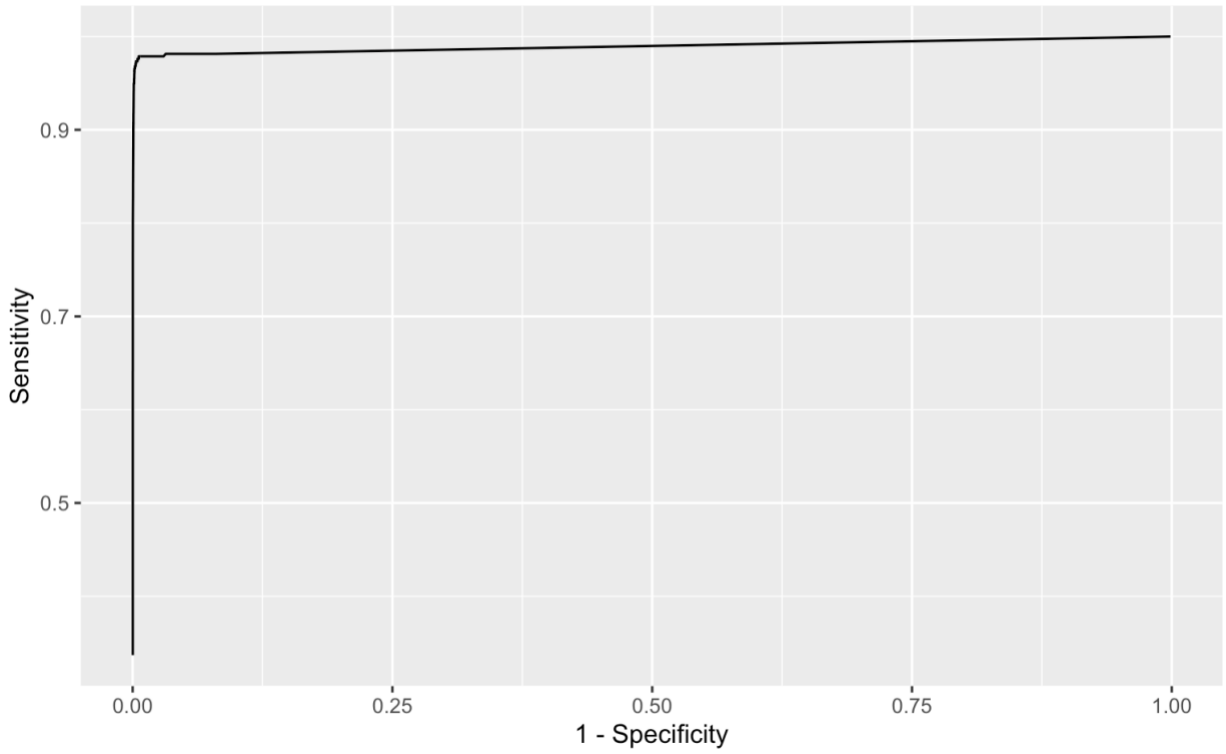
```
             0      1
FALSE 227446    250
TRUE      24    127
```

```
#Define a loss function
performance$Loss = performance$FP*lossFP + performance$FN*lossFN

ggplot(performance, aes(Cutoff, Loss)) +
  geom_line()
```




```
performance[which.min(performance$Loss),]
```

	Cutoff <dbl>	TN <int>	FN <int>	TP <int>	FP <int>	Sensitivity <dbl>	Specificity <dbl>	Accuracy <dbl>	Loss <dbl>
803	0.802	227230	19	358	240	0.9496021	0.9989449	0.9988633	3445.234



Question 7

Show the confusion matrix.

```
predicted_prob_test <- model3_smote %>%
    predict(as.matrix(test_data[,-ncol(test_data)]), type = 'prob')
default_prob_test = as.data.frame(predicted_prob_test)$V1

delta = performance[which.min(performance$Loss),]$Cutoff
predicted_default = ifelse(default_prob_test >= delta, 1, 0)
confusionMatrix(as.factor(predicted_default),as.factor(test_data$Fraud), mode
= 'everything', positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 56795    24
         1    50    91

               Accuracy : 0.9987
                 95% CI : (0.9984, 0.999)
    No Information Rate : 0.998
    P-Value [Acc > NIR] : 2.836e-05

                  Kappa : 0.7103

 Mcnemar's Test P-Value : 0.003659

            Sensitivity : 0.791304
            Specificity : 0.999120
         Pos Pred Value : 0.645390
         Neg Pred Value : 0.999578
              Precision : 0.645390
                 Recall : 0.791304
                     F1 : 0.710938
             Prevalence : 0.002019
         Detection Rate : 0.001598
   Detection Prevalence : 0.002475
      Balanced Accuracy : 0.895212

       'Positive' Class : 1
```

a. Per one million transactions, how many fraudulent transactions do we detect and stop before they go through? $\frac{91 \cdot 1000000}{56960} = 1598$ transactions

b. If we process twenty million transactions in a year, how much money do we lose to undetected fraudulent transactions per month? $\frac{24 \cdot 20000000}{56960} \cdot 118.1702 = \$995{,}816$

c. Per one million transactions, how many customers have their transactions unnecessarily put on hold for a confirmation? $\frac{50 \cdot 1000000}{56960} = 878$ transactions

d. How statistically confident are you in your answers to these questions? We are not very confident in our answers because the estimated lossFP of $5 may be far from what it actually is. However, as our AUC is 0.9264, we are confident that this is a good model.