Part 1

a)

Code

```
###### a) ######


rm(list = ls())

gc()



pre <- read.csv(file = "C:/Users/XXX/OneDrive - Aarhus universitet/Skrivebord/BI/2. semester/ML2/Exam 2022/dataLC01.csv")

options(scipen = 999)


# Make loanstatus as factor

loanstatus <- as.factor(pre$loanstatus)

pre$loanstatus <- NULL

loanstatus_num <- ifelse(loanstatus == "Charged Off", 1, 0) # convert loan status to numerical

pre <- cbind(loanstatus, pre)
```

Explanation

I just transform. I convert it to numerical as well for later purposes (for confusionmatrix etc.)

b)

Code

```
###### b) ######


set.seed(202)

options(scipen = 999)


# Split the dataset


train <- sample(nrow(pre), 5000)

pre <- pre[train, ]
```

pre$homeownershipNONE = NULL

pre$iswv = NULL

Output

```
> set.seed(202)
> options(scipen = 999)
>
> # Split the dataset
>
> train <- sample(nrow(pre), 5000)
> pre <- pre[train, ]
>
> pre$homeownershipNONE = NULL
> pre$iswv = NULL
>
```

Explanation

No relevant output

c)

Code

c)

library(caret)

index <- createDataPartition(pre$loanstatus, p = .5,

list = FALSE,

times = 1)

train <- pre[index,]

test <- pre[-index,]

Output

test	2500 obs. of 57 variables
train	2500 obs. of 57 variables

Explanation

I split using the caret library. As seen from above, there is 2500 obs in each data set.

d)

Code

load libraries

library(caret)

library(rpart)

```
library(randomForest)

# define training control
train_control<- trainControl(method = "cv", number = 3)

# train the model
rf.fit <- train(loanstatus ~ ., data = train, trControl = train_control, method = "rf", ntree = 10)

rf.fit
```

Output

```
Random Forest

2500 samples
  56 predictor
   2 classes: 'Charged Off', 'Fully Paid'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 1666, 1666, 1668
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.7696016  0.01673541
  29    0.7455997  0.12828376
  56    0.7315936  0.11516079

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```

Explanation

First I define a train_control object in order to force the caret train() function to perform a k-fold cross-validation equal to 3 (with 3 folds). Then, I train the model by stating that I want to use loanstatus as response variable, and all the other variables as predictors. I furthermore type “rf” to tell caret to perform a random forest and also type in 10 trees.

As seen from the output, the optimal mtry based on cross-validation is 2. This is because the accuracy for that model is the largest. So, when caret trains, it uses accuracy to evaluate the best model. I could also have used other metrics using the metrics argument.

Mtry is how random forest differ from bagging. Because the mtry is better in the sense, that it decorrelates the trees. So, it still builds decision trees on bootstrap training samples. But each time it splits a tree, a random sample of m predictors (a subset of predictors which in this case turns out to be 2 that is best) is chosen as split candidates from the full set of p predictors. So, it only chooses between 2 predictors every time the tree is split.

The advantage of this is that if we have a strong predictor, a normal bagging algorithm will choose that every time in the top split. And as a result, the bagged trees will look very similar, and the predictions from the bagged trees will be highly correlated. On the other hand, random forest decorrelates the different trees by not only relying on 1 single predictor for every top split.

And this is good because we will reduce variance more, in contrast to bagging that doesn't reduce variance much because the highly correlated quantities will be averaged and very similar.

Generally, we chose mtry = sqrt(p).

e)

Code

```
###### e) ######

train.param <- trainControl(method = "cv", number = 3) # we want to do CV of 3

tune.grid <- expand.grid(n.trees = seq(5,50,500), interaction.depth = 4, shrinkage = 0.1, n.minobsinnode =
10)

boost.caret.fit <- train(loanstatus ~ ., data = train,

                         method = "gbm", # boosting method

                         trControl = train.param,

                         tuneGrid = tune.grid)
```

Output

```
> boost.caret.fit <- train(loanstatus ~ ., data = train,
+                          method = "gbm", # boosting method
+                          trControl = train.param,
+                          tuneGrid = tune.grid)
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.0539             nan     0.1000    0.0073
     2        1.0389             nan     0.1000    0.0070
     3        1.0267             nan     0.1000    0.0035
     4        1.0152             nan     0.1000    0.0048
     5        1.0058             nan     0.1000    0.0032

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.0599             nan     0.1000    0.0038
     2        1.0461             nan     0.1000    0.0053
     3        1.0359             nan     0.1000    0.0028
     4        1.0265             nan     0.1000    0.0029
     5        1.0205             nan     0.1000    0.0008
```

Explanation

Not asked to explain anything.

f)

Code

```
library(xgboost)

xg.train.param <- trainControl(method = "cv", number = 3)

tune.grid.xgboost <- expand.grid(max_depth = 4:6, gamma = c(0, 1, 2), eta = c(0.03, 0.06), nrounds = 300,
                                 subsample = 0.5, colsample_bytree = 0.1, min_child_weight = 1)

model.xgboost <- train(loanstatus ~ ., train,
                       method = "xgbTree",
                       tuneGrid = tune.grid.xgboost,
                       trControl = xg.train.param,
                       metric = "Accuracy")

model.xgboost # The final values used for the model were nrounds = 300, max_depth = 4, eta = 0.03, gamma = 1, colsample_bytree = 0.1, min_child_weight =
              # 1 and subsample = 0.5.
model.xgboost$results
max(model.xgboost$results$Accuracy)
```

Output

```
> model.xgboost$results
    eta max_depth gamma colsample_bytree min_child_weight subsample nrounds  Accuracy      Kappa   AccuracySD     KappaSD
1  0.03         4     0              0.1                1       0.5     300 0.7720009 0.04648489 0.0022641816 0.013318181
2  0.03         4     1              0.1                1       0.5     300 0.7755990 0.06059516 0.0042375500 0.028063809
3  0.03         4     2              0.1                1       0.5     300 0.7739998 0.05229284 0.0030430246 0.015300746
10 0.06         4     0              0.1                1       0.5     300 0.7680017 0.11106793 0.0055645624 0.053233256
11 0.06         4     1              0.1                1       0.5     300 0.7703983 0.10499330 0.0060145939 0.024347889
12 0.06         4     2              0.1                1       0.5     300 0.7735967 0.10825450 0.0079453329 0.037096917
4  0.03         5     0              0.1                1       0.5     300 0.7703983 0.04686561 0.0051075699 0.021854546
5  0.03         5     1              0.1                1       0.5     300 0.7743985 0.06175340 0.0037353784 0.017282979
6  0.03         5     2              0.1                1       0.5     300 0.7720004 0.04663006 0.0031278133 0.008341993
13 0.06         5     0              0.1                1       0.5     300 0.7699982 0.12040124 0.0067030250 0.035696089
14 0.06         5     1              0.1                1       0.5     300 0.7627991 0.09110687 0.0019883918 0.028034801
15 0.06         5     2              0.1                1       0.5     300 0.7691969 0.10559058 0.0069754390 0.032642519
7  0.03         6     0              0.1                1       0.5     300 0.7731990 0.05413387 0.0032800141 0.019468206
8  0.03         6     1              0.1                1       0.5     300 0.7728002 0.06348044 0.0060262139 0.030783703
9  0.03         6     2              0.1                1       0.5     300 0.7748001 0.05730294 0.0006296876 0.014960675
16 0.06         6     0              0.1                1       0.5     300 0.7627986 0.09596748 0.0037915143 0.023468019
17 0.06         6     1              0.1                1       0.5     300 0.7647966 0.10180806 0.0080175047 0.018358845
18 0.06         6     2              0.1                1       0.5     300 0.7695985 0.10747486 0.0053278840 0.038808121
> max(model.xgboost$results$Accuracy)
[1] 0.775599
```

```
Tuning parameter 'nrounds' was held constant at a value of 300
Tuning parameter 'colsample_bytree' was held constant at a value of 0.1

Tuning parameter 'min_child_weight' was held constant at a value of 1
Tuning parameter 'subsample' was held constant at a value of 0.5
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were nrounds = 300, max_depth = 4, eta = 0.03, gamma = 1, colsample_bytree = 0.1, min_child_weight =
 1 and subsample = 0.5.
```

Explanation

I have tuned using a grid where max_depth was 4 to 6, gamma was 0,1,2, eta was either 0.03 or 0.06. This choice is somehow arbitrary, and I'm limited in how many values for each parameter I can perform (because of exam situation). But we have used these numbers in lecture at some point.

The accuracy of the best model based on the k-fold cross validation, is 0.7756.

The final values used for that best model were nrounds = 300, max_depth = 4, eta = 0.03, gamma = 1, colsample_bytree = 0.1, min_child_weight = 1 and subsample = 0.5.

g)

Code

```
###### g) ######


# rf


rfmetric <- rbind(max(rf.fit$results$Accuracy), rf.fit$results$Kappa[1])



# boosting


boostmetric <- rbind(boost.caret.fit$results$Accuracy, boost.caret.fit$results$Kappa)
```

```
# xgboost


xgboostmetric <- rbind(max(model.xgboost$results$Accuracy), model.xgboost$results$Kappa[2])



list <- as.data.frame(NA)

list <- cbind(rfmetric,boostmetric,xgboostmetric)


list
```

Output

```
> list
          [,1]      [,2]       [,3]
[1,] 0.76960163 0.7724001 0.77559897
[2,] 0.01673541 0.0000000 0.06059516
>
```

Explanation

I indexed the values from the models themselves and converted them into a list. The first column is random forest, the second is (gradient) boosting and the third (to the right) is the extreme gradient boosting model.

Row number 1 represents accuracy and number 2 represents kappa. Unfortunately, I didn't manage to index the kappa for the boosting model. It can probably be done by forcing R in an argument while training, but I don't have time for retraining now.

The best model according to accuracy is the extreme gradient boosting model, as the accuracy for that is 77.56%.

Normally, in terms of kappa values in statistics, we want as high kappa value as possible. So, the xgboost model has the highest kappa value of 0.06059.

h)

Code

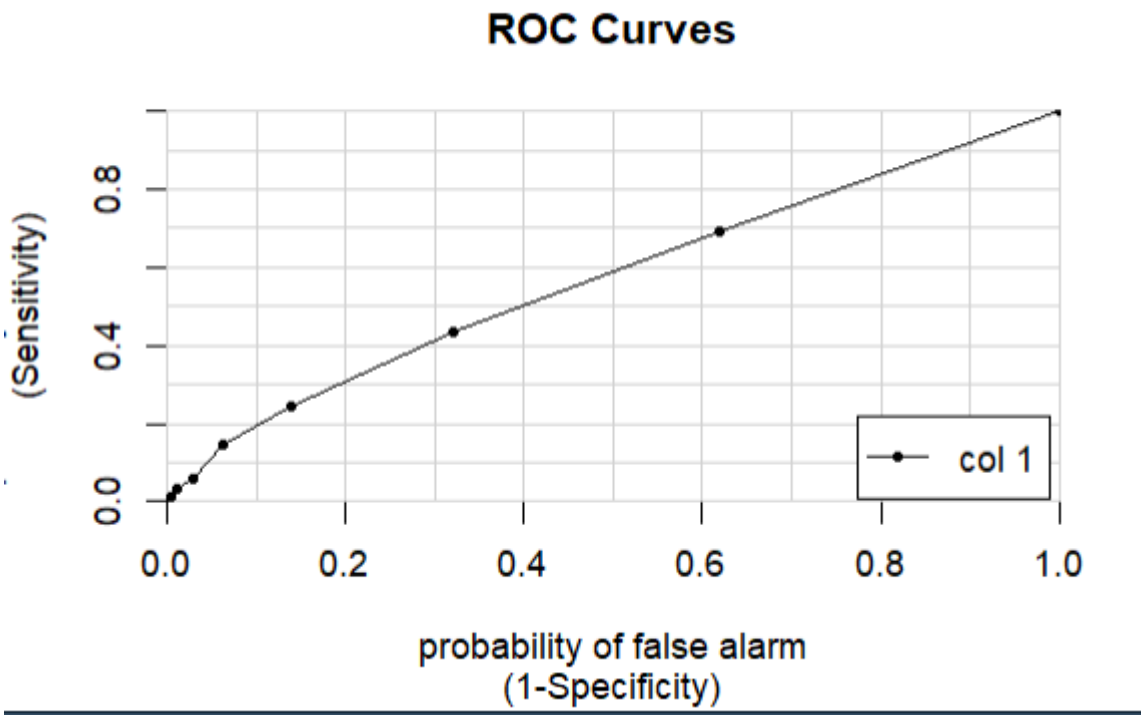
```
###### h) ######


loanstatus_num <- ifelse(test$loanstatus == "Charged Off", 1, 0)


# Random forest


rf.pred <- predict(rf.fit, test, type = "prob")

rf.prob <- ifelse(rf.pred$`Fully Paid` < 0.5, 1, 0)
```

```
confusionMatrix(as.factor(rf.prob),
                as.factor(loanstatus_num), positive = "1")


# ROC and AUC: a good way to summarize your model
library(caTools)
colAUC(rf.pred[,1], loanstatus_num, plotROC = TRUE)




# Boosting


boost.caret.pred <- predict(boost.caret.fit, test, type = "prob")
boost.caret.prob <- ifelse(boost.caret.pred[,2] < 0.5, 1, 0)


confusionMatrix(as.factor(boost.caret.prob),
                as.factor(loanstatus_num), positive = "1")

# ROC and AUC: a good way to summarize your model
library(caTools)
colAUC(boost.caret.pred[,2], loanstatus_num, plotROC = TRUE)



# Extreme gradient boosting
```

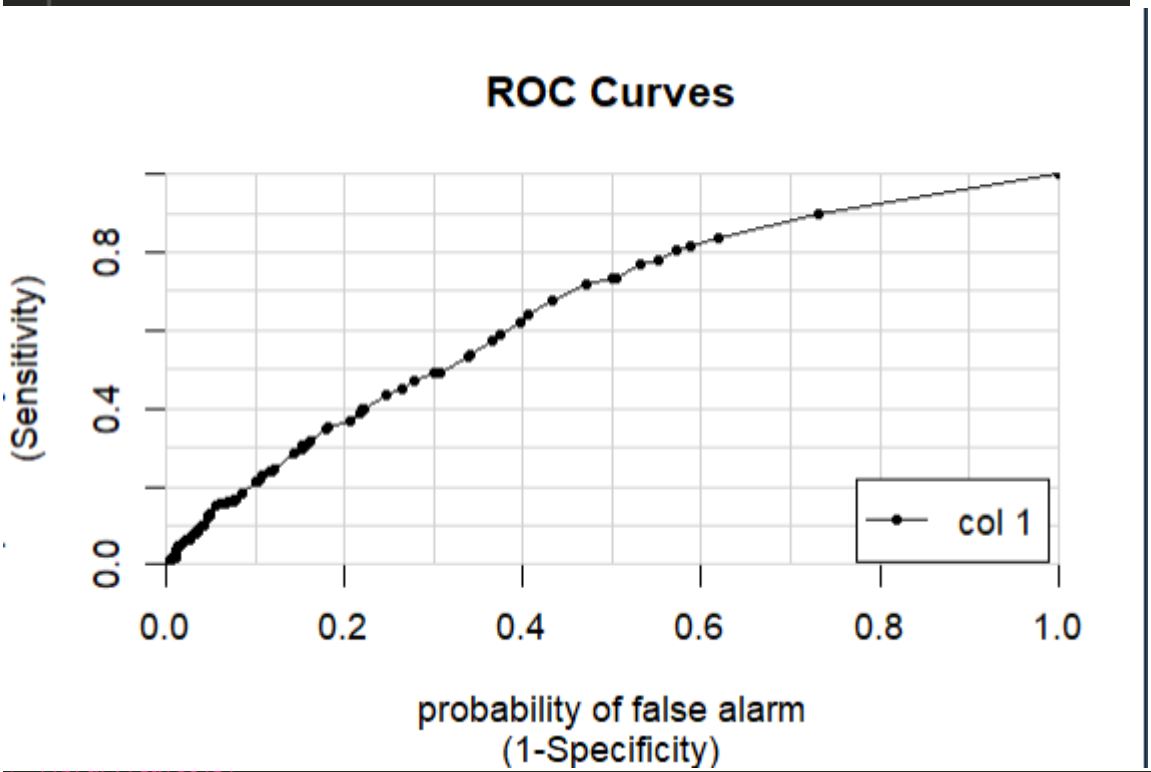
```
xgb.pred <- predict(model.xgboost, test, type = "prob")
xgb.prob <- ifelse(xgb.pred[,2] < 0.5, 1, 0)

confusionMatrix(as.factor(xgb.prob),
                as.factor(loanstatus_num), positive = "1")

# ROC and AUC: a good way to summarize your model
library(caTools)
colAUC(xgb.pred[,2] , loanstatus_num, plotROC = TRUE) # ROC is the curve, AUC the area under curve
```

Output

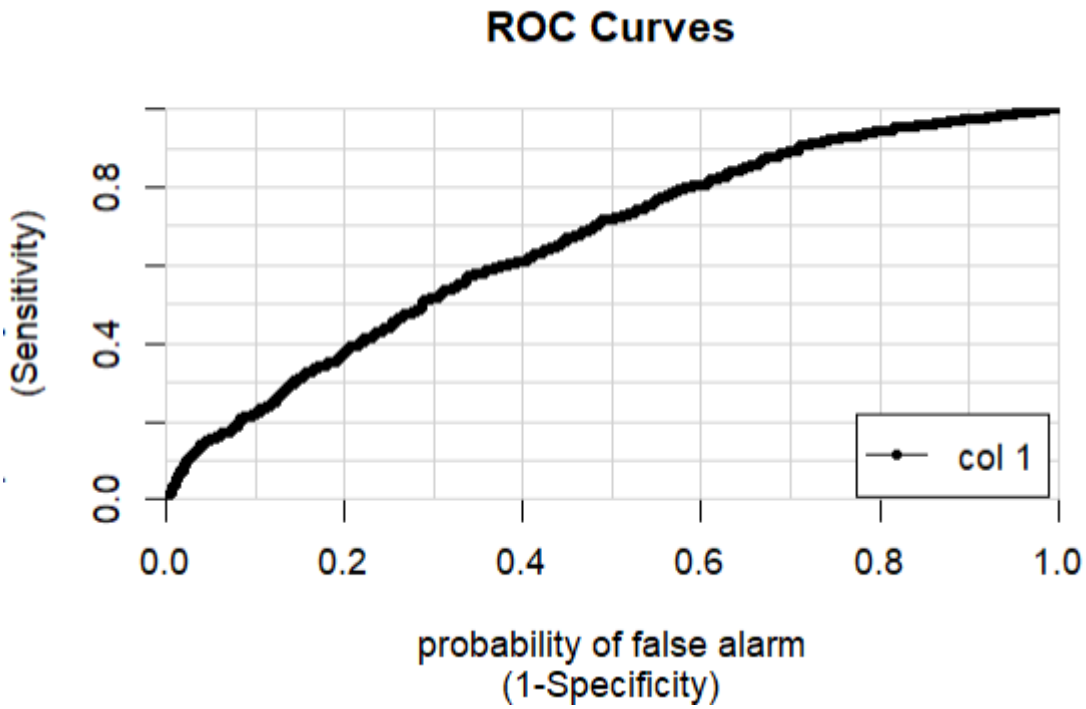
```
> colAUC(rf.pred[,1], loanstatus_num, plotROC = TRUE)
              [,1]
0 vs. 1 0.5715438
```



```
> colAUC(boost.caret.pred[,2], loanstatus_num, plotROC = TRUE)
              [,1]
0 vs. 1 0.6546041
```




```
> colAUC(xgb.pred[,2] , loanstatus_num, plotROC = TRUE) # ROC is the curve, AUC the area under curve
              [,1]
0 vs. 1 0.6609363
```



Explanation

Random forest has AUC score = 0.5715, boosting has 0.6546, and xgboost has 0.6609.

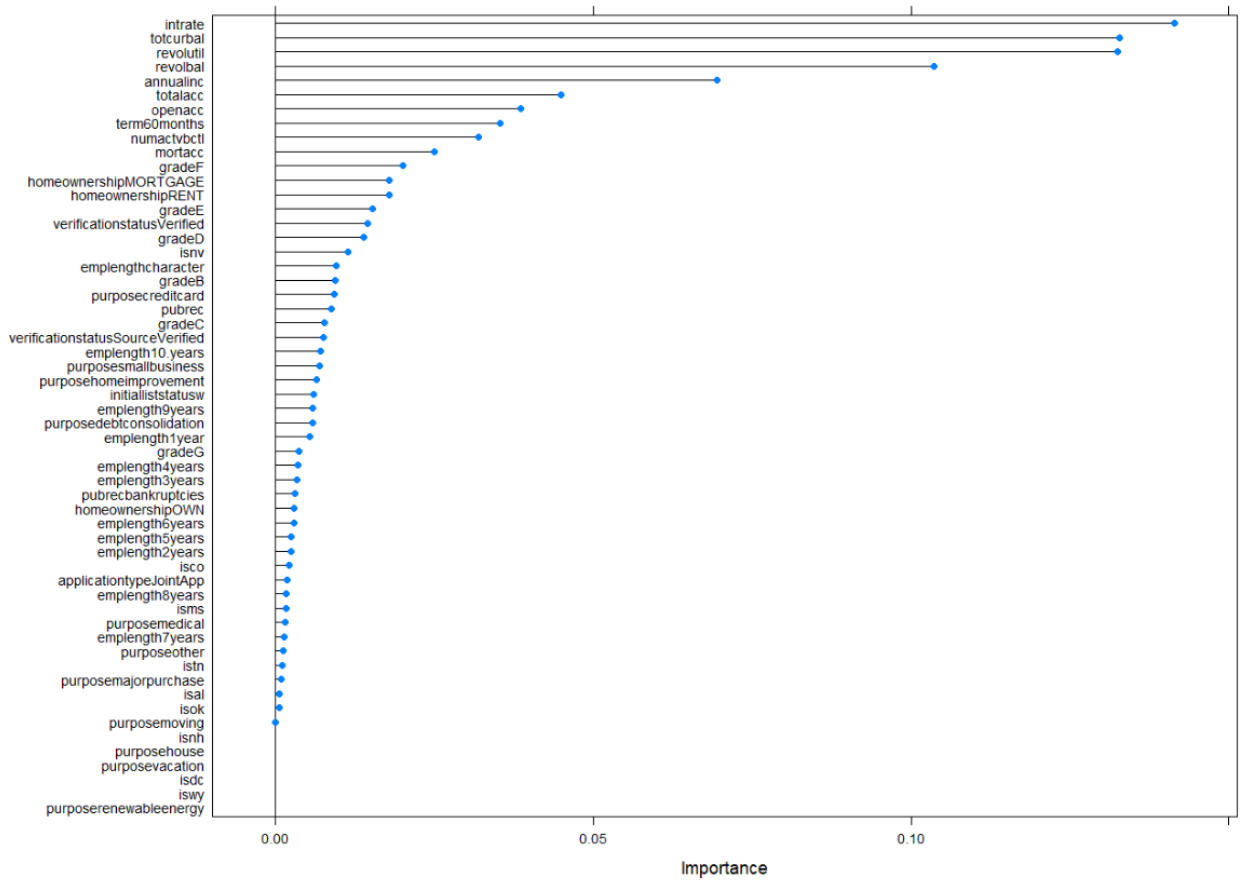
Thus, the extreme gradient boosting model is best according to the AUC.

i)

Code

```
###### i) ######


# Xgboost


var.imp <- varImp(model.xgboost, scale = FALSE)

plot(var.imp)

head(var.imp$importance)
```

Output




```
> head(var.imp$importance)
             Overall
intrate   0.14154098
totcurbal 0.13286531
revolutil 0.13256102
revolbal  0.10368163
annualinc 0.06953281
totalacc  0.04493332
```

Explanation

The xgboost model was the best in terms of AUC. Above is shown a plot of the importance values. Also, by only showing the six most important predictors, its clear to see that there is a huge gap between the importance of the first variable (interest rate) and the sixth variable (totalacc).

The six best predictors are therefore intrate, totcurbal, revolutil, revolbal, annualinc, totalacc

j)

Code

Output

Explanation

k)

Code

```
###### k) ######


prop <- prop.table(table(train$loanstatus))

baseline_acc <- prop[2]

baseline_acc




prop2 <- prop.table(table(test$loanstatus))

baseline_acc <- prop2[2]

baseline_acc
```

Output

```
> prop <- prop.table(table(train$loanstatus))
> baseline_acc <- prop[2]
> baseline_acc
Fully Paid
    0.7724
>
```

```
> confusionMatrix(as.factor(rf.prob),
+                 as.factor(loanstatus_num), positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1910  550
         1   21   19

               Accuracy : 0.7716
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1931  569
         1    0    0

               Accuracy : 0.7724
```

```
> confusionMatrix(as.factor(xgb.prob),
+                 as.factor(loanstatus_num), positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1910  542
         1   21   27

               Accuracy : 0.7748
```

Explanation

I basically just find the proportion of fully paid loans which is 77.24% for both the train and test set. Thus, on the training and test data we will get a accuracy of 77.24% if we just follow a strategy of predicting all loans as majority class. This is the same accuracy as using a boosting model.

However, the random forest has an test accuracy of 77.16%, while the xgboost has an accuracy of 77.48%.

So, in terms of accuracy, we still get better prediction using a xgboost model, but it's only slightly better than a majority class prediction.