Intro to Data Science Lab 9

```
# Enter your name here: Hrishikesh Telang
```

Attribution statement: (choose only one and delete the rest)

```
# 1. I did this homework by myself, with help from the book and the professor.
```

#Instructions: Supervised data mining/machine learning is the most prevalent form of data mining as it allows for the prediction of new cases in the future. For example, when credit card companies are trying to detect fraud, they will create a supervised model by training it on fraud data that they already have. Then they will deploy the model into the field: As new input data arrives the model predicts whether it seems fraudulent and flags those transactions where that probability is high.In these exercises we will work with a built-in data set called GermanCredit. This data set is in the "caret" package so we will need that and the "kernlab" package to be installed and "libraried" before running the following:

```
#install.packages('caret')
#install.packages('kernlab')
#install.packages('libraried')
library('caret')
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library('kernlab')
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
data("GermanCredit")
subCredit <- GermanCredit[,1:10]
str(subCredit)
```

```
## 'data.frame':    1000 obs. of  10 variables:
##  $ Duration                 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                   : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration        : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                      : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits    : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance  : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone                : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
```

#1. Examine the data structure that str() reveals. Also use the help() function to learn more about the GermanCredit data set. Summarize what you see in a comment.

```
help('GermanCredit')
head(subCredit)
```

```
##   Duration Amount InstallmentRatePercentage ResidenceDuration Age
## 1        6   1169                         4                 4  67
## 2       48   5951                         2                 2  22
## 3       12   2096                         2                 3  49
## 4       42   7882                         2                 4  45
## 5       24   4870                         3                 4  53
## 6       36   9055                         2                 4  35
##   NumberExistingCredits NumberPeopleMaintenance Telephone ForeignWorker Class
## 1                     2                       1         0             1  Good
## 2                     1                       1         1             1   Bad
## 3                     1                       2         1             1  Good
## 4                     1                       2         1             1  Good
## 5                     2                       2         1             1   Bad
## 6                     1                       2         0             1  Good
```

#2. Use the createDataPartition() function to generate a list of cases to include in the training data. This function is conveniently provided by caret and allows one to directly control the number of training cases. It also ensures that the training cases are balanced with respect to the outcome variable. Try this:

```
trainList <- createDataPartition(y=subCredit$Class,p=.40,list=FALSE)
```

#3. Examine the contents of trainList to make sure that it is a list of case numbers. With p=0.40, it should have 400 case numbers in it.

```
str(trainList)
```

```
##  int [1:400, 1] 1 2 3 14 15 22 23 24 29 30 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "Resample1"
```

```
#trainList
```

#4. What is trainList? What do the elements in trainList represent? Which attribute is balanced in the trainList dataset?

```
#trainList is used to train our model with the available model to implement
#it on the test data.
#The train list consists the 40 percent of the data
```

#5. Use trainList and the square brackets notation to create a training data set called "trainSet" from the subCredit data frame. Look at the structure of trainSet to make sure it has all of the same variables as subCredit. The trainSet structure should be a data frame with 400 rows and 10 columns.

```
trainSet <- subCredit[trainList,]
head(trainSet)
```

```
##    Duration Amount InstallmentRatePercentage ResidenceDuration Age
## 1         6   1169                         4                 4  67
## 2        48   5951                         2                 2  22
## 3        12   2096                         2                 3  49
## 14       24   1199                         4                 4  60
## 15       15   1403                         2                 4  28
## 22        6   2647                         2                 3  44
##    NumberExistingCredits NumberPeopleMaintenance Telephone ForeignWorker Class
## 1                      2                       1         0             1  Good
## 2                      1                       1         1             1   Bad
## 3                      1                       2         1             1  Good
## 14                     2                       1         1             1   Bad
## 15                     1                       1         1             1  Good
## 22                     1                       2         1             1  Good
```

```
str(trainSet)
```

```
## 'data.frame':    400 obs. of  10 variables:
##  $ Duration                 : int  6 48 12 24 15 6 10 12 7 60 ...
##  $ Amount                   : int  1169 5951 2096 1199 1403 2647 2241 1804 2415 6836 ...
##  $ InstallmentRatePercentage: int  4 2 2 4 2 2 1 3 3 3 ...
##  $ ResidenceDuration        : int  4 2 3 4 4 3 3 4 2 4 ...
##  $ Age                      : int  67 22 49 60 28 44 48 44 34 63 ...
##  $ NumberExistingCredits    : int  2 1 1 2 1 1 2 1 1 2 ...
##  $ NumberPeopleMaintenance  : int  1 1 2 1 1 2 2 1 1 1 ...
##  $ Telephone                : num  0 1 1 1 1 1 1 1 1 0 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 0 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 1 2 2 2 2 2 1 ...
```

#6. Use trainList and the square brackets notation to create a testing data set called "testSet" from the subCredit data frame. The testSet structure should be a data frame with 600 rows and 10 columns and should be a completely different set of cases than trainSet.

```
testSet <- subCredit[-trainList,]
head(testSet)
```

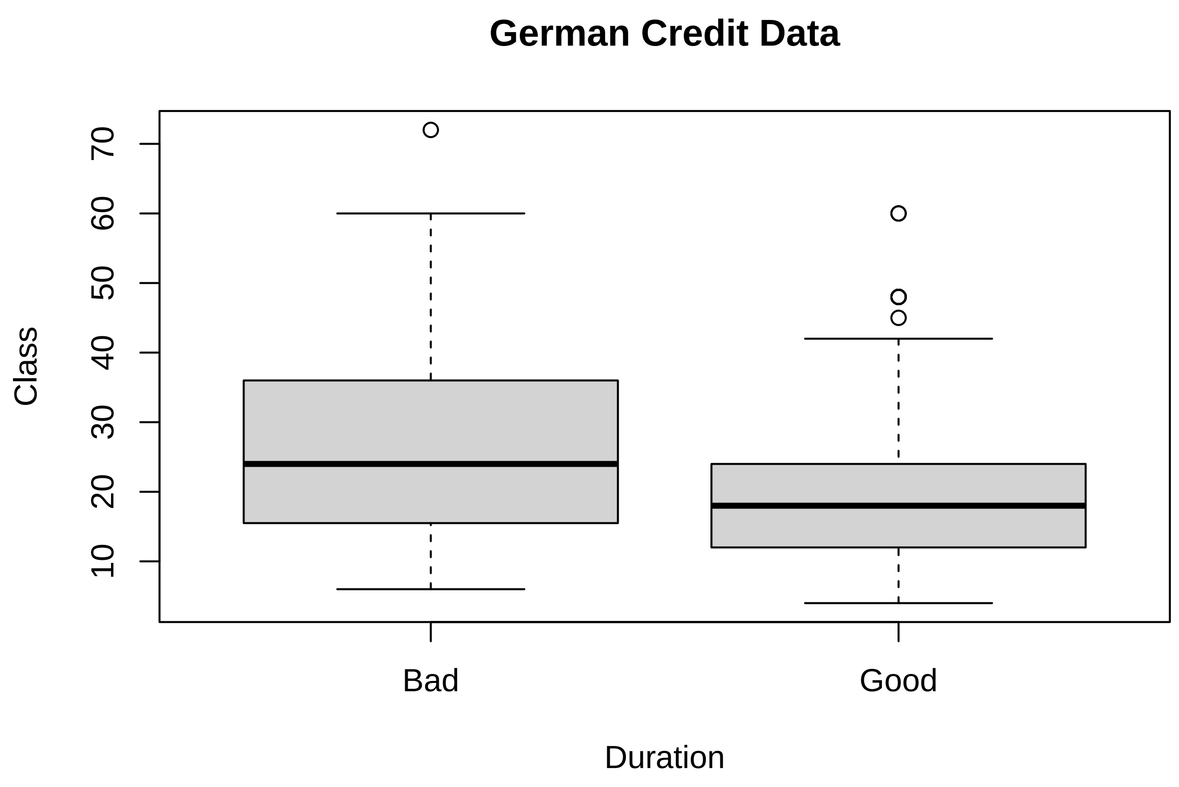
```
##   Duration Amount InstallmentRatePercentage ResidenceDuration Age
## 4       42   7882                         2                 4  45
## 5       24   4870                         3                 4  53
## 6       36   9055                         2                 4  35
## 7       24   2835                         3                 4  53
## 8       36   6948                         2                 2  35
## 9       12   3059                         2                 4  61
##   NumberExistingCredits NumberPeopleMaintenance Telephone ForeignWorker Class
## 4                     1                       2         1             1  Good
## 5                        2                        2         1             1   Bad
## 6                        1                        2         0             1  Good
## 7                        1                        1         1             1  Good
## 8                        1                        1         0             1  Good
## 9                        1                        1         1             1  Good
```

```
str(testSet)
```

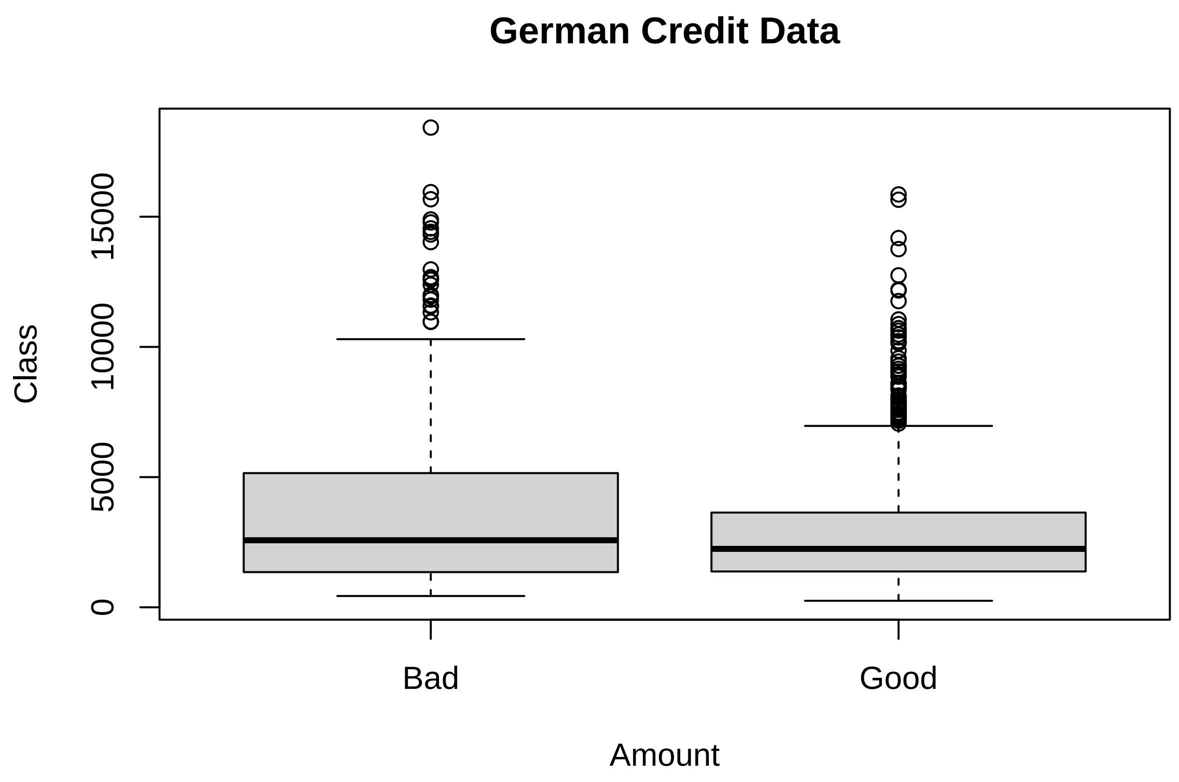
```
## 'data.frame':    600 obs. of  10 variables:
##  $ Duration                 : int  42 24 36 24 36 12 30 12 48 12 ...
##  $ Amount                   : int  7882 4870 9055 2835 6948 3059 5234 1295 4308 1567 ...
##  $ InstallmentRatePercentage: int  2 3 2 3 2 2 4 3 3 1 ...
##  $ ResidenceDuration        : int  4 4 4 4 2 4 2 1 4 1 ...
##  $ Age                      : int  45 53 35 53 35 61 28 25 24 22 ...
##  $ NumberExistingCredits    : int  1 2 1 1 1 1 2 1 1 1 ...
##  $ NumberPeopleMaintenance  : int  2 2 2 1 1 1 1 1 1 1 ...
##  $ Telephone                : num  1 1 0 1 0 1 1 1 1 0 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 2 2 2 1 1 1 2 ...
```

#7. Create and interpret boxplots of all the predictor variables in relation to the outcome variable (Class).

```
boxplot(Duration~Class,data=trainSet, main="German Credit Data",
   xlab="Duration", ylab="Class")
```



```
boxplot(Amount~Class,data=subCredit, main="German Credit Data",
   xlab="Amount", ylab="Class")
```



```
#boxplot(InstallmentRatePercentage~Class,data=subCredit, main="German Credit Data",
#   xlab="Installment Rate Percentage", ylab="Class")
#boxplot(ResidenceDuration~Class,data=subCredit, main="German Credit Data",
#   xlab="Residence Duration", ylab="Class")
#boxplot(Age~Class,data=subCredit, main="German Credit Data",
#   xlab="Age", ylab="Class")
#boxplot(NumberExistingCredits~Class,data=subCredit, main="German Credit Data",
#   xlab="Number of existing credits", ylab="Class")
#boxplot(NumberPeopleMaintenance~Class,data=subCredit, main="German Credit Data",
#   xlab="Number of people being liable to provide maintenance for", ylab="Class")
#boxplot(Telephone~Class,data=subCredit, main="German Credit Data",
#   xlab="Telephone", ylab="Class")
#boxplot(ForeignWorker~Class,data=subCredit, main="German Credit Data",
#   xlab="Foreign Worker Status", ylab="Class")
```

#8. Train a support vector machine with the ksvm() function from the kernlab package. Make sure that you have installed and libraried the kernlab package. Have the cost be 5, and have ksvm do 3 cross validations (hint: try prob.model = TRUE)

```
svm <- ksvm(Class ~ .,data = trainSet, C=5, cross=3, prob.model = TRUE)
svm
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0885704216237498
##
## Number of Support Vectors : 269
##
## Objective Function Value : -947.3436
## Training error : 0.1975
## Cross validation error : 0.307635
## Probability model included.
```

#9. Examine the ksvm output object. In particular, look at the cross-validation error for an initial indication of model quality. Add a comment that gives your opinion on whether this is a good model.

```
#The cross-validation error is 0.297591
#This error is close to the training error, and thus can be deemed as a good model
```

#10. Predict the training cases using the predict command

```
pred <- predict(svm, testSet)
pred
```

```
##   [1] Good Good Good Good Good Good Good Good Good Good Good Good Bad  Good Bad
##  [16] Good Bad  Good Good Good Good Good Good Good Bad  Good Good Good Good Good
##  [31] Bad  Good Good Good Good Good Good Good Good Good Good Bad  Good Good Bad
##  [46] Good Good Good Good Good Good Good Good Good Good Good Good Good Good Good
##  [61] Good Good Good Good Good Good Bad  Good Good Good Good Good Good Bad  Good
##  [76] Good Good Good Good Bad  Good Bad  Bad  Good Good Good Good Good Good Good
##  [91] Good Good Good Bad  Good Good Good Good Good Good Good Good Good Good Good
## [106] Good Good Good Good Good Good Good Good Good Good Good Good Good Good Good
## [121] Good Good Good Good Good Bad  Good Good Bad  Good Good Good Good Good Good
## [136] Good Good Good Good Good Good Good Good Good Good Good Good Good Good Good
## [151] Bad  Good Good Bad  Bad  Good Good Good Good Good Good Good Good Good Good
## [166] Good Bad  Bad  Good Good Good Bad  Good Good Good Good Good Good Bad  Good
## [181] Good Bad  Good Bad  Bad  Good Good Good Good Good Good Good Good Good Good
## [196] Good Good Good Good Good Good Good Good Good Good Good Good Good Bad  Good
## [211] Good Good Good Good Good Good Good Good Good Good Good Good Good Good Good
## [226] Good Good Bad  Good Good Good Good Good Good Good Good Good Good Good Good
## [241] Good Good Bad  Good Good Good Good Good Good Good Good Good Good Good Good
## [256] Good Good Good Good Bad  Good Good Good Good Good Good Good Good Good Good
## [271] Good Good Good Good Good Good Good Good Bad  Good Good Good Good Good Good
## [286] Good Good Good Good Good Good Good Good Good Good Good Bad  Good Good Good
## [301] Good Good Good Bad  Good Good Good Good Good Good Good Good Good Good Good
## [316] Good Bad  Good Good Good Good Good Bad  Good Good Good Good Good Good Good
## [331] Good Good Good Good Good Good Good Good Good Good Good Good Good Good Good
## [346] Good Good Good Good Good Good Good Good Good Good Good Good Good Bad  Bad
## [361] Good Good Good Good Good Good Good Good Good Good Good Good Good Good Good
## [376] Good Bad  Good Good Bad  Good Good Bad  Good Good Good Good Good Good Bad
## [391] Good Good Good Good Bad  Bad  Good Good Good Good Good Good Good Good Good
## [406] Good Good Bad  Good Good Good Good Bad  Good Good Good Good Good Good Good
## [421] Good Good Good Good Bad  Good Good Good Good Good Good Bad  Good Good Good
## [436] Bad  Good Bad  Good Good Good Bad  Good Good Good Good Good Good Good Good
## [451] Good Good Good Good Bad  Good Good Good Bad  Bad  Good Good Good Good Good
## [466] Good Good Good Good Good Good Bad  Good Good Good Good Bad  Good Good Good
## [481] Bad  Good Good Good Good Good Good Bad  Good Bad  Bad  Good Good Good Good
## [496] Good Good Good Good Good Good Bad  Good Good Good Good Good Good Good Good
## [511] Good Good Good Bad  Good Good Good Good Good Good Good Bad  Good Good Good
## [526] Good Good Good Good Good Good Good Good Bad  Good Good Good Good Bad  Good
## [541] Good Good Good Bad  Good Good Good Good Good Good Bad  Good Good Good Good
## [556] Good Good Good Good Good Bad  Good Good Good Good Good Good Good Good Good
## [571] Bad  Good Good Good Good Good Good Good Good Good Good Good Good Good Good
## [586] Good Bad  Good Good Good Good Good Good Good Good Good Good Good Good Good
## Levels: Bad Good
```

#11. Examine the predicted out object with str( ). Then, calculate a confusion matrix using the table function.

```
summary(pred)
```

```
##  Bad Good
##   68  532
```

```
table(pred,testSet$Class)
```

```
##
## pred   Bad Good
##   Bad   29   39
##   Good 151  381
```

#12. Interpret the confusion matrix and in particular calculate the overall accuracy of the model. The diag( ) command can be applied to the results of the table command you ran in the previous step. You can also use sum() to get the total of all four cells.

```
#The accuracy of the model is 69.33%
sum(diag(table(pred,testSet$Class)))/sum(table(pred,testSet$Class))
```

```
## [1] 0.6833333
```

#13. Check you calculation with confusionMatrix() function in the caret package.

```
#We are using Confusion Matrix using the function
library(caret)
confusionMatrix(pred,testSet$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad   29   39
##       Good 151  381
```

```
##
##                Accuracy : 0.6833
##                  95% CI : (0.6444, 0.7204)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.8254
##
##                   Kappa : 0.083
##
##  Mcnemar's Test P-Value : 8.093e-16
##
##             Sensitivity : 0.16111
##             Specificity : 0.90714
##          Pos Pred Value : 0.42647
##          Neg Pred Value : 0.71617
##              Prevalence : 0.30000
##          Detection Rate : 0.04833
##    Detection Prevalence : 0.11333
##       Balanced Accuracy : 0.53413
##
##        'Positive' Class : Bad
##
```