Intro to Data Science - Lab 9

Week 9 - Supervised Data Mining

```
# Enter your name here: Hendi Kushta
```

Please include nice comments.

Instructions:

Run the necessary code on your own instance of R-Studio.

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

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

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:

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

```
# install.packages("kernlab")
# install.packages("caret")
library(kernlab)
library(caret)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':
##
##     alpha

## Loading required package: lattice
```

```
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()** command to learn more about the **GermanCredit** data set. Summarize what you see in a comment.

```
# from what we see, all the data are numeric,
# NumberExistingCredits, NumberPeopleMaintainance, Telephone, ForeignWorked and Class are
# classified data in only 2 groups.
# from what we can see, InstallmentRatePercentage, ResidenceDuratio, might also be grouped
# in 4 different groups.
help(GermanCredit)
```

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)`

```
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 6 8 11 13 16 17 18 20 22 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "Resample1"
```

```
# numeric with 400 case numbers.
```

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

```
# it's a portion of our actual dataset that is fed into the machine learning model
# to discover and learn patterns. They represent part of our dataset chosen randomly.
# Class is a balanced attribute.
```

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,]
str(trainSet)
```

```
## 'data.frame':    400 obs. of  10 variables:
##  $ Duration                 : int  6 36 36 12 12 24 24 30 24 6 ...
##  $ Amount                   : int  1169 9055 6948 1295 1567 1282 2424 8072 3430 2647 ...
##  $ InstallmentRatePercentage: int  4 2 2 3 1 4 4 2 3 2 ...
##  $ ResidenceDuration        : int  4 4 2 1 1 2 4 3 2 3 ...
##  $ Age                      : int  67 35 35 25 22 32 53 25 31 44 ...
##  $ NumberExistingCredits    : int  2 1 1 1 1 1 2 3 1 1 ...
##  $ NumberPeopleMaintenance  : int  1 2 1 1 1 1 1 1 2 2 ...
##  $ Telephone                : num  0 0 0 1 0 1 1 1 0 1 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 2 2 2 1 2 1 2 2 2 2 ...
```

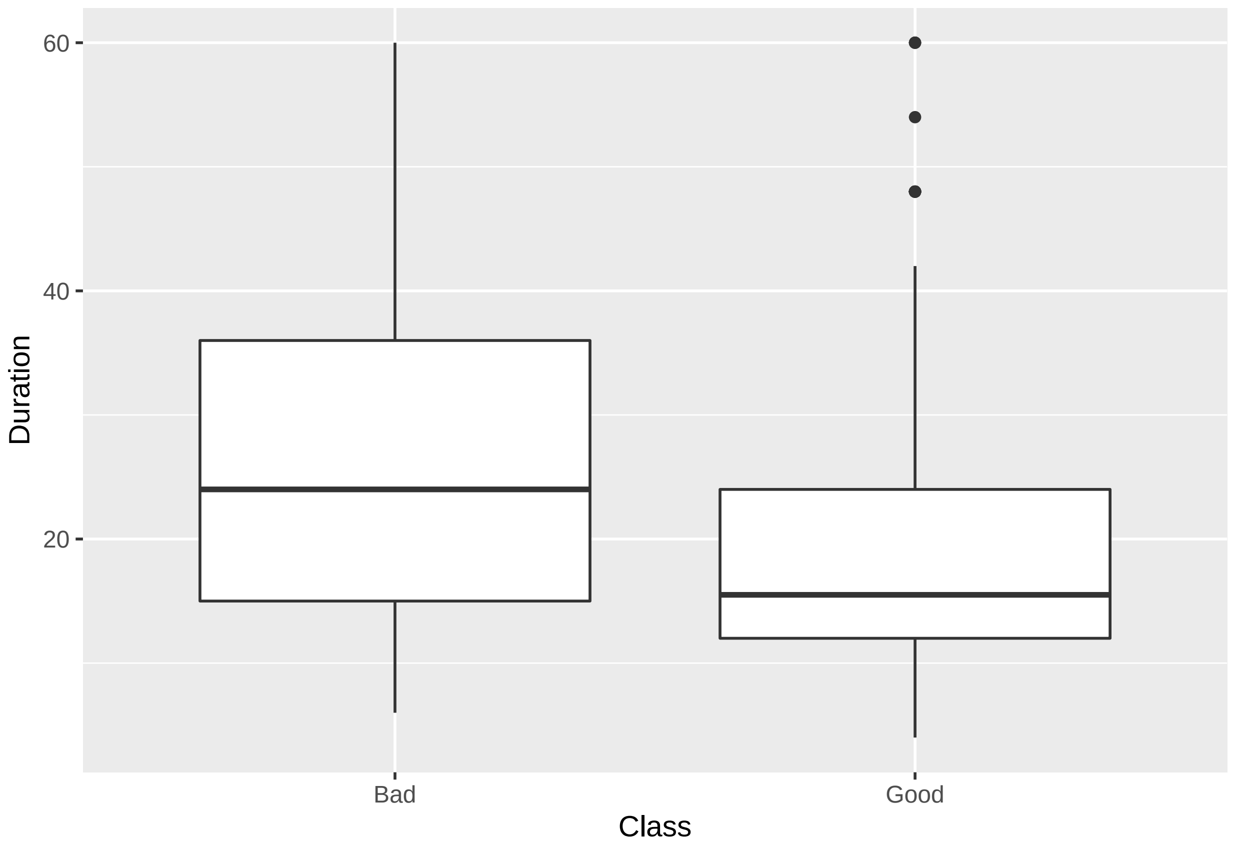
6. Use **trainList** and the square brackets notation to create a testing data set called ** testSet ** from the subCredit data frame. The **testSet** structure should be a data frame with **600 rows and 10 columns** and should be a completely different set of cases than **trainSet**.

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

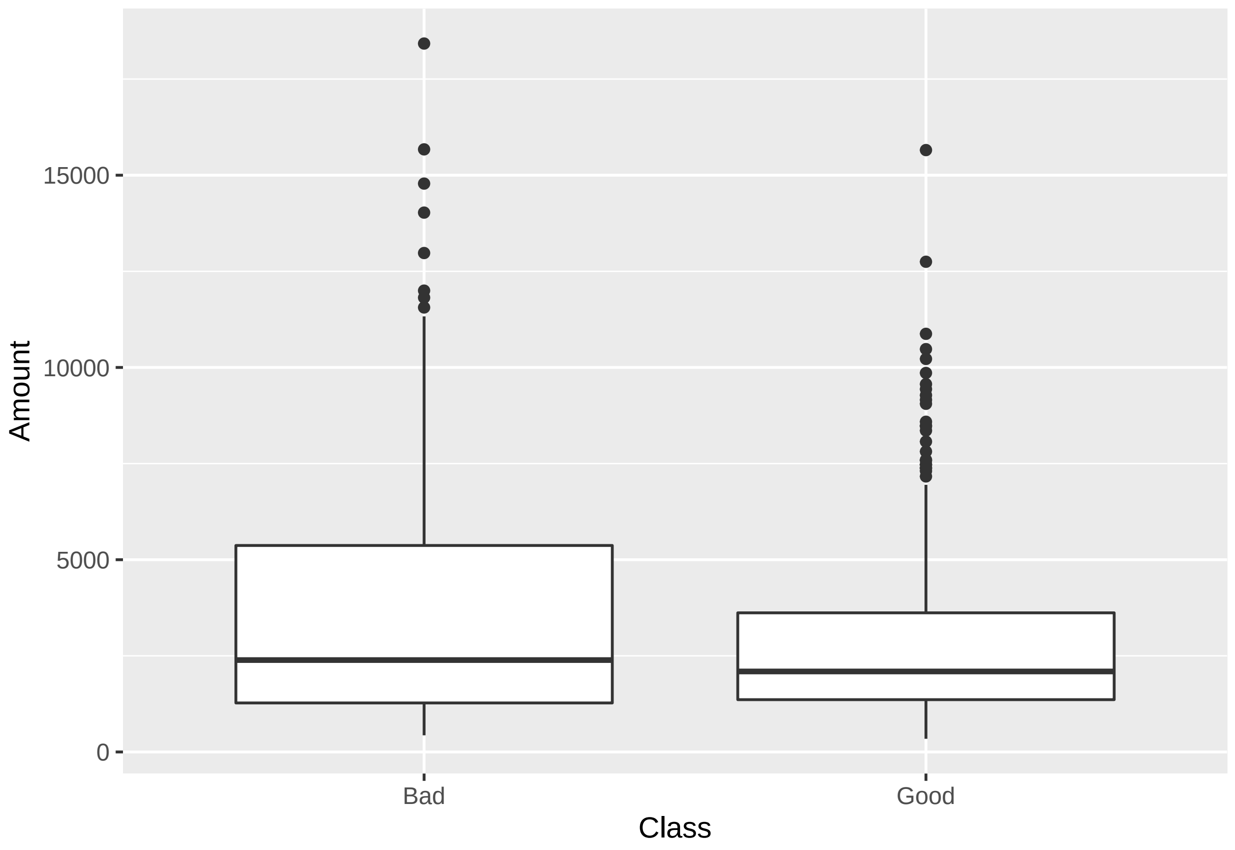
```
## 'data.frame':    600 obs. of  10 variables:
##  $ Duration                 : int  48 12 42 24 24 12 30 48 24 15 ...
##  $ Amount                   : int  5951 2096 7882 4870 2835 3059 5234 4308 1199 1403 ...
##  $ InstallmentRatePercentage: int  2 2 2 3 3 2 4 3 4 2 ...
##  $ ResidenceDuration        : int  2 3 4 4 4 4 2 4 4 4 ...
##  $ Age                      : int  22 49 45 53 53 61 28 24 60 28 ...
##  $ NumberExistingCredits    : int  1 1 1 2 1 1 2 1 2 1 ...
##  $ NumberPeopleMaintenance  : int  1 2 2 2 1 1 1 1 1 1 ...
##  $ Telephone                : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 1 2 2 1 2 2 1 1 1 2 ...
```

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

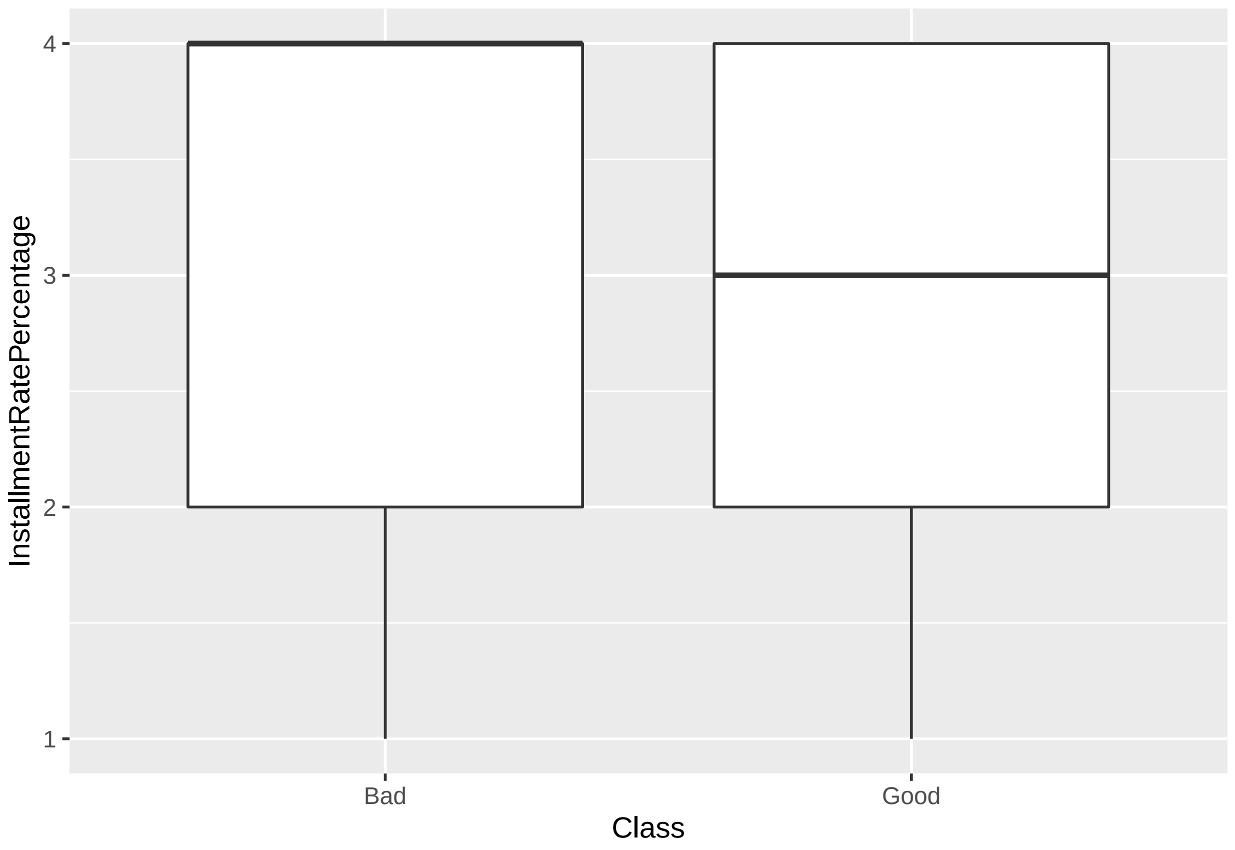
```
# Duration
ggplot(trainSet,aes(Class,Duration))+geom_boxplot()
```



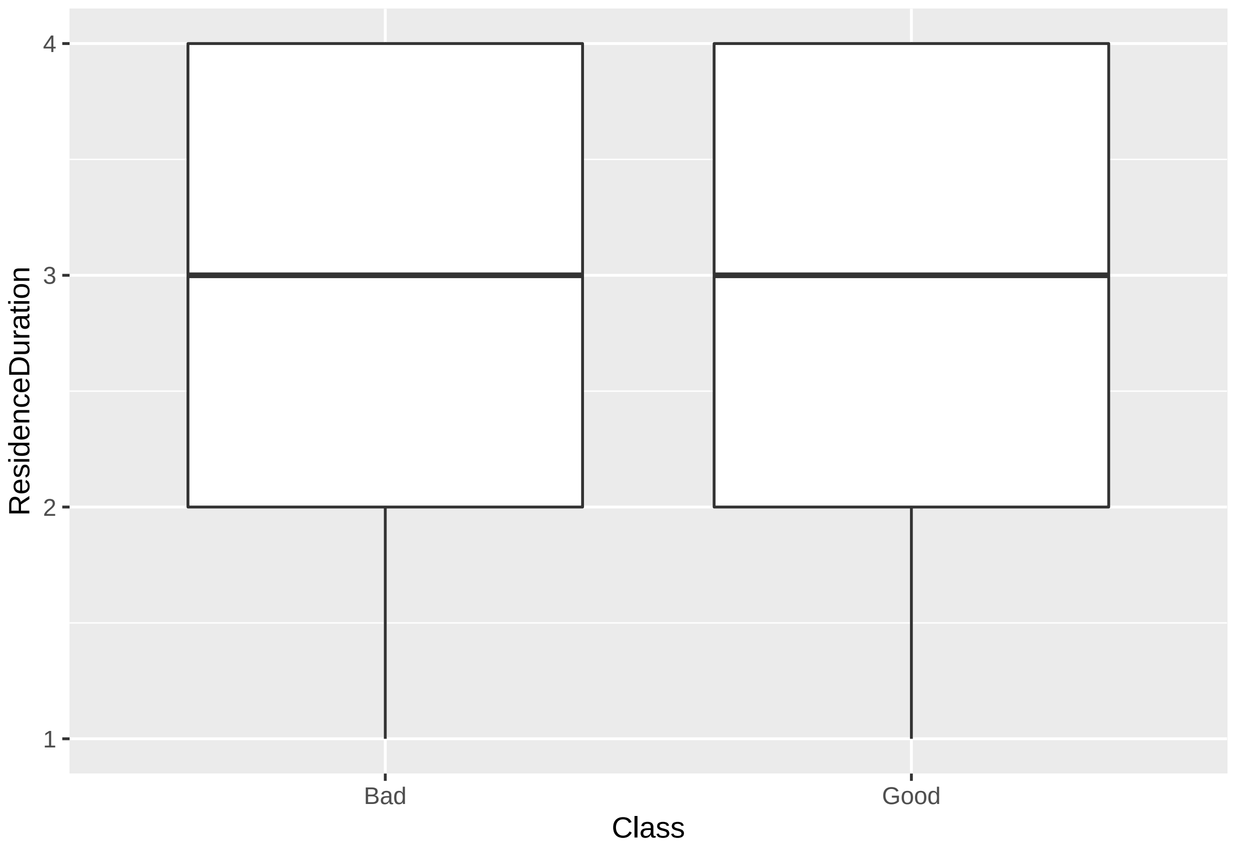
```
#  Amount
ggplot(trainSet,aes(Class,Amount))+geom_boxplot()
```



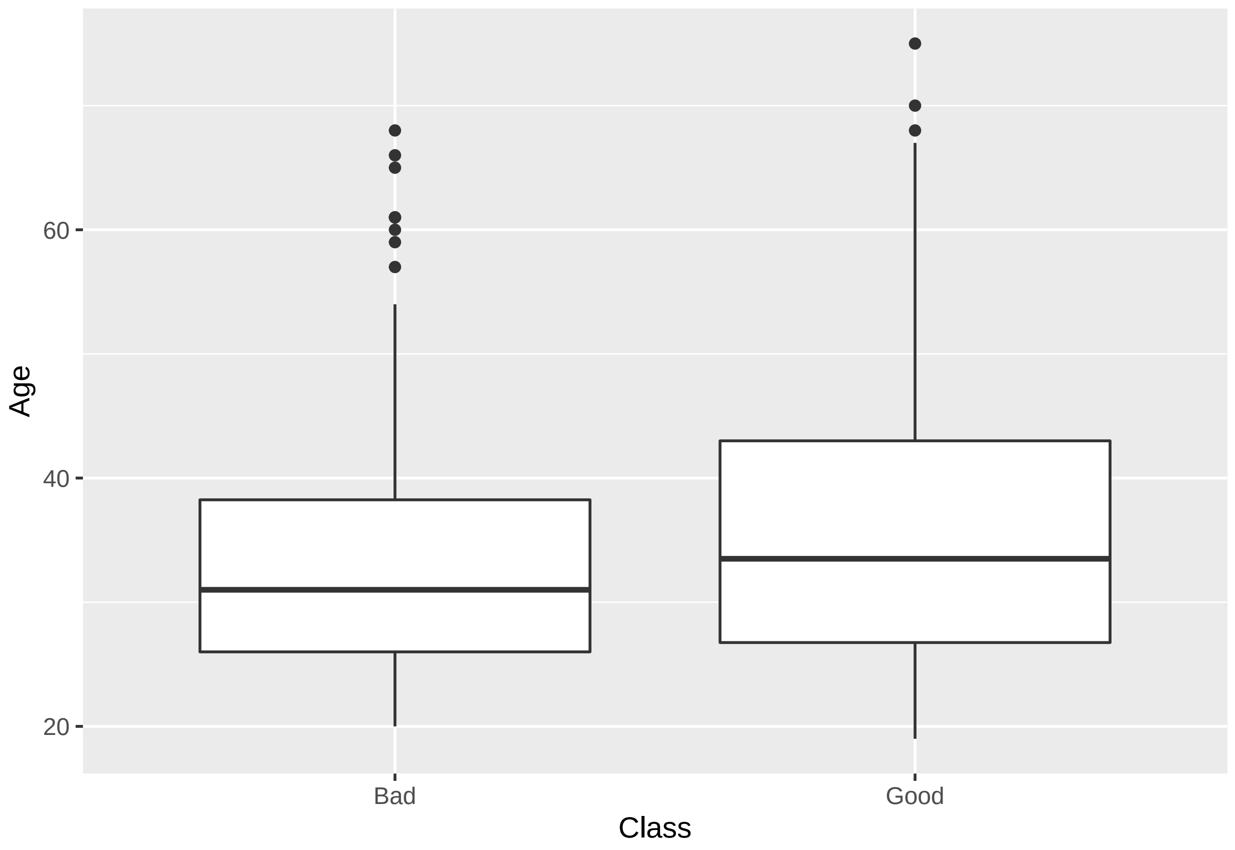
```
#  InstallmentRatePercentage
ggplot(trainSet,aes(Class,InstallmentRatePercentage))+geom_boxplot()
```



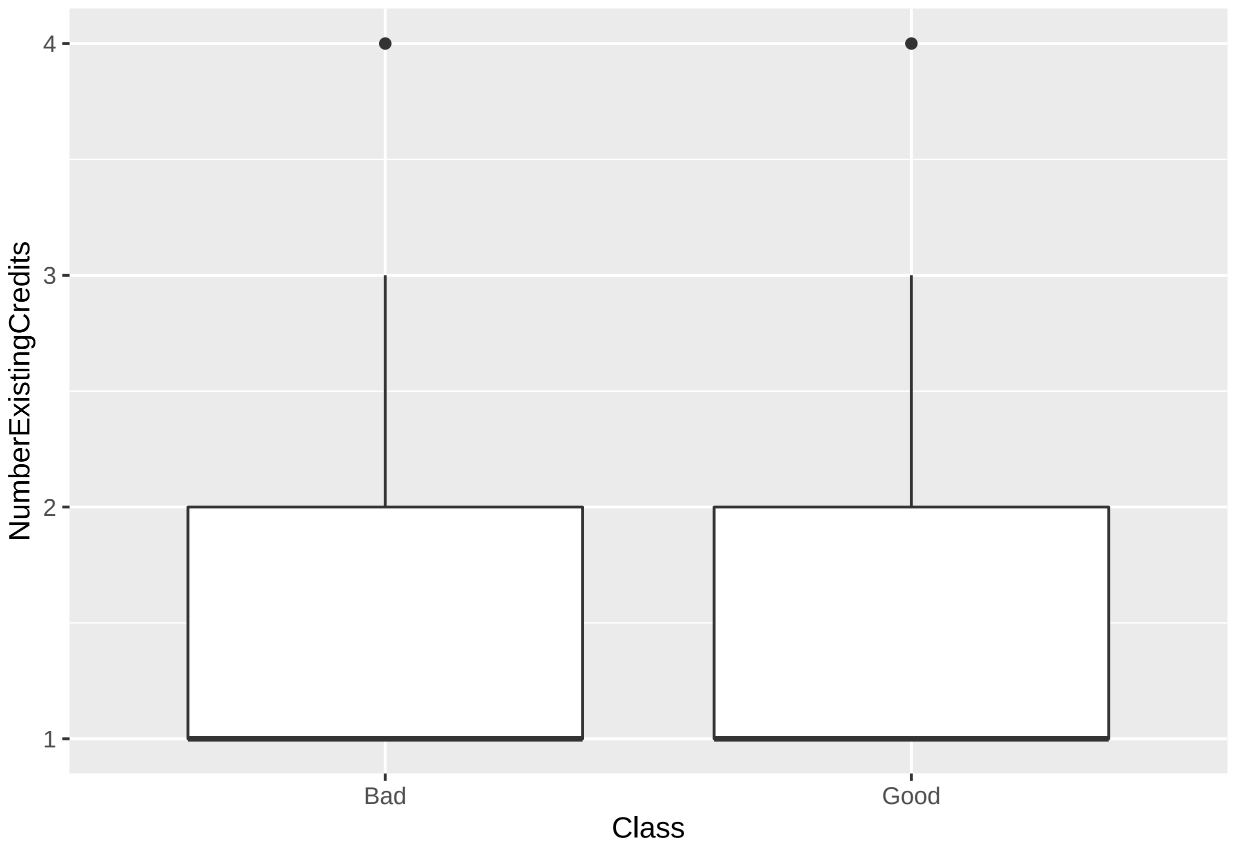
```
#  ResidenceDuration
ggplot(trainSet,aes(Class,ResidenceDuration))+geom_boxplot()
```



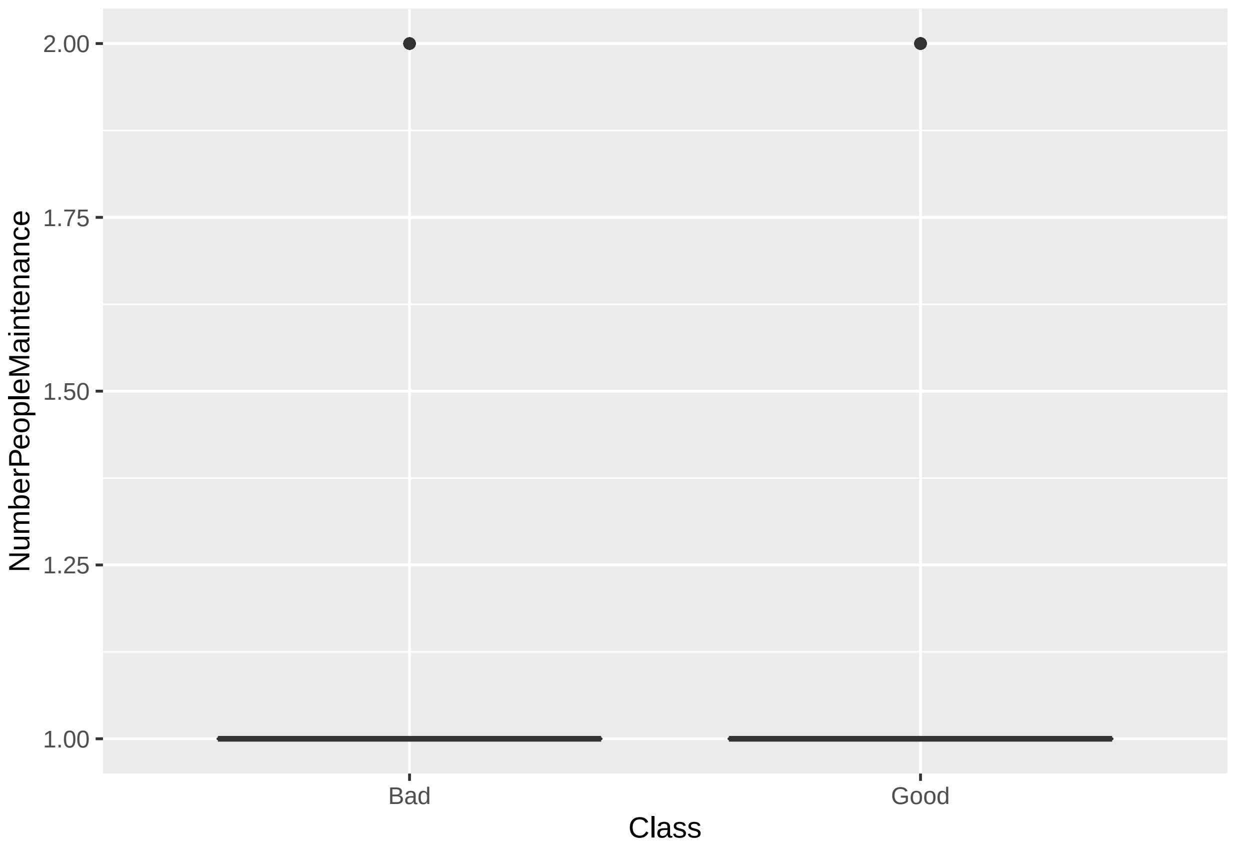
```r
#  Age
ggplot(trainSet,aes(Class,Age))+geom_boxplot()
```



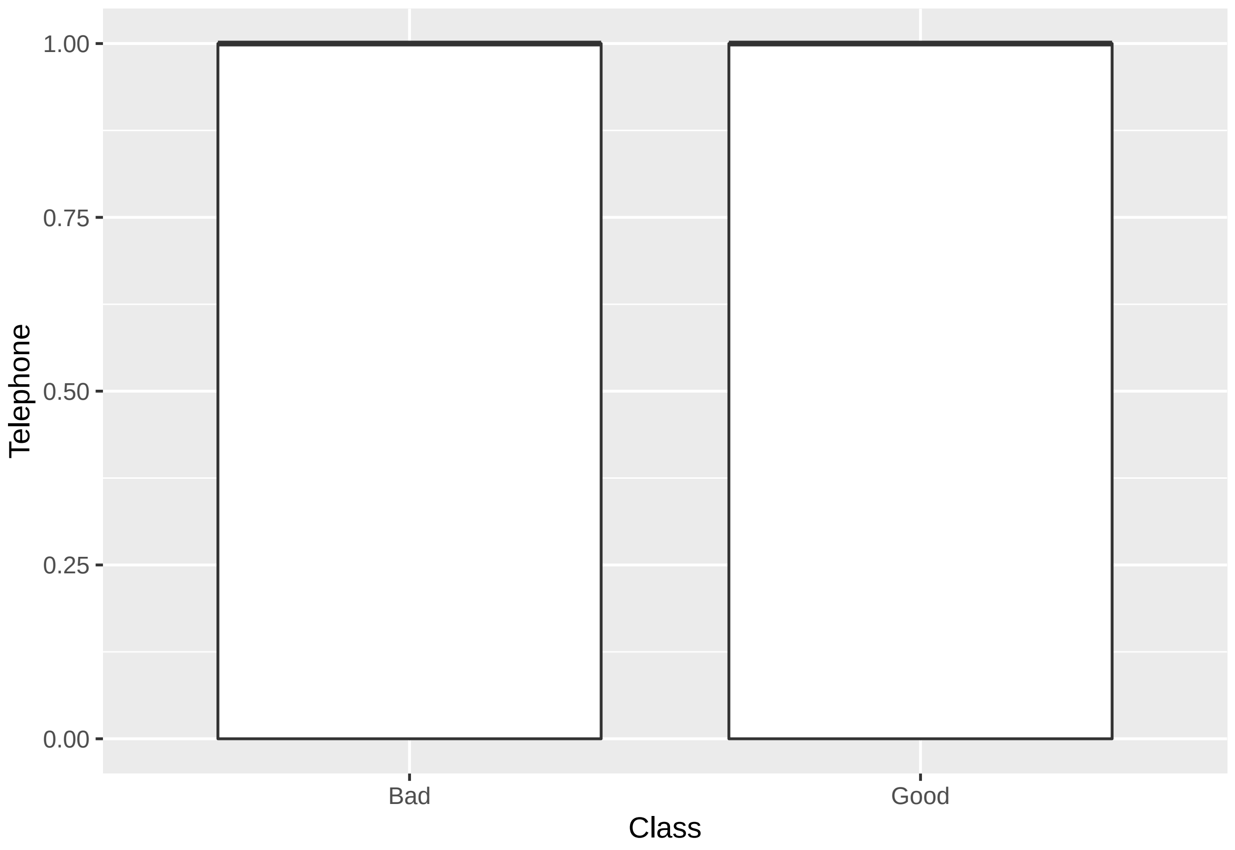
```
#  NumberExistingCredits
ggplot(trainSet,aes(Class,NumberExistingCredits))+geom_boxplot()
```



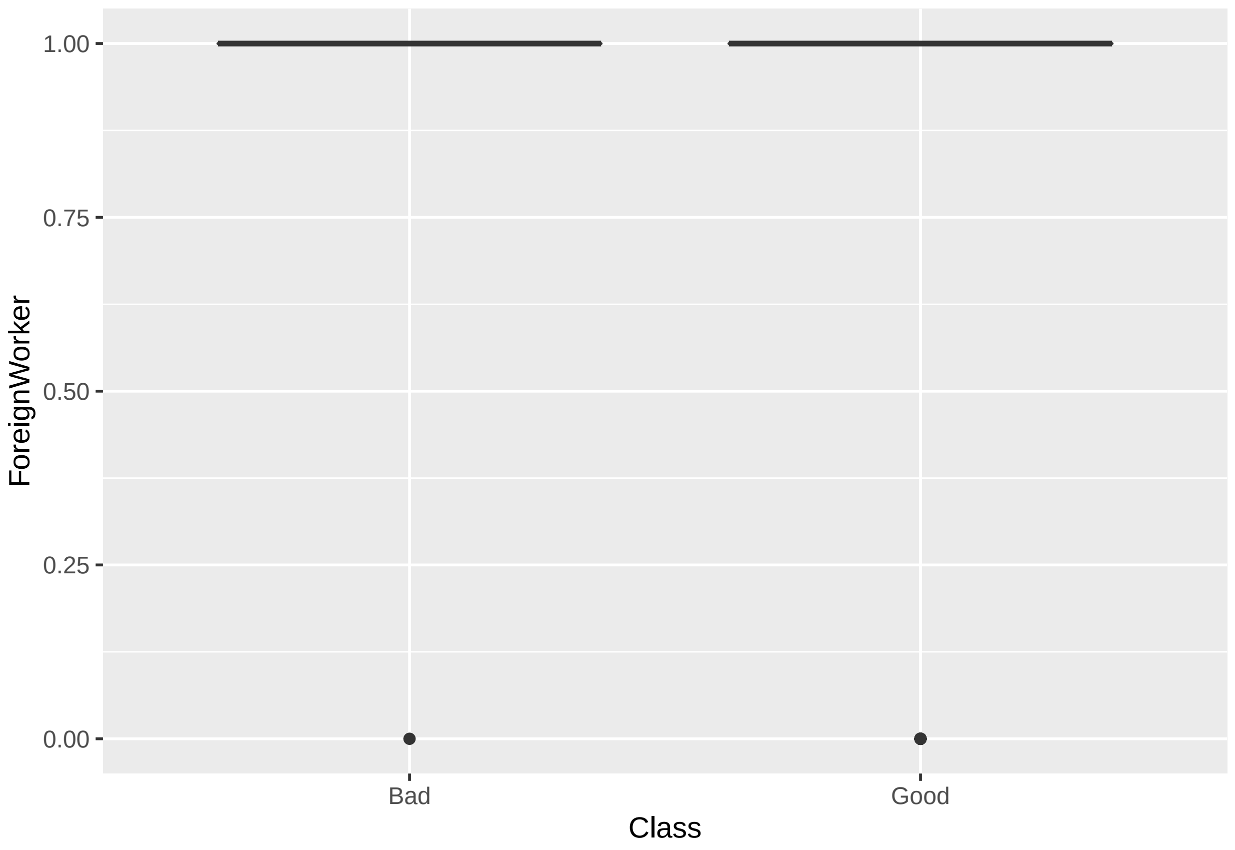
```
#  NumberPeopleMaintenance
ggplot(trainSet,aes(Class,NumberPeopleMaintenance))+geom_boxplot()
```



```
#  Telephone
ggplot(trainSet,aes(Class,Telephone))+geom_boxplot()
```



```
#  ForeignWorker
ggplot(trainSet,aes(Class,ForeignWorker))+geom_boxplot()
```



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`)

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

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.

```
model
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.092219395402324
##
## Number of Support Vectors : 270
##
## Objective Function Value : -952.3245
## Training error : 0.2125
## Probability model included.
```

```
# 79% accurate model. We can not decide if it is a good or bad model.
```

10. Predict the training cases using the **predict()** command

```r
predOut <- predict(model, newdata=testSet)
```

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

```r
str(predOut)
```

```
##  Factor w/ 2 levels "Bad","Good": 2 2 2 1 2 2 2 1 2 2 ...
```

```r
confMatrix <- table(predOut,testSet$Class)
```

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.

```r
errorRate <- (sum(confMatrix)-sum(diag(confMatrix)))/sum(confMatrix)
errorRate
```

```
## [1] 0.2733333
```

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

```r
confusionMatrix(predOut, testSet$Class)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction Bad Good
##       Bad   39   23
##       Good 141  397
## 
##                Accuracy : 0.7267
##                  95% CI : (0.6891, 0.762)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.08278
## 
##                   Kappa : 0.1992
## 
##  Mcnemar's Test P-Value : < 2e-16
## 
##             Sensitivity : 0.2167
##             Specificity : 0.9452
##          Pos Pred Value : 0.6290
##          Neg Pred Value : 0.7379
##              Prevalence : 0.3000
##          Detection Rate : 0.0650
##    Detection Prevalence : 0.1033
##       Balanced Accuracy : 0.5810
## 
##        'Positive' Class : Bad
## 
```

```r
# accuracy is the same 71%, but the model is not significant.
# p-value is high and no information rate is almost the same as Accuracy.
```