Question 9.1

Load in all the necessary libraries.

```
rm(list=ls(all=TRUE))
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 3.4.4
```

```
library(Rmisc)
```

```
## Warning: package 'Rmisc' was built under R version 3.4.4
## Loading required package: lattice
## Loading required package: plyr
## Warning: package 'plyr' was built under R version 3.4.4
```

```
library(car)
```

```
## Warning: package 'car' was built under R version 3.4.4
## Loading required package: carData
## Warning: package 'carData' was built under R version 3.4.4
```

Univariate analysis

Let's read in the data:

```
crime_data <- read.table("9.1uscrimeSummer2018.txt", header = T)
str(crime_data)
```

```
## 'data.frame':    47 obs. of  16 variables:
##  $ M     : num  15.1 14.3 14.2 13.6 14.1 12.1 12.7 13.1 15.7 14 ...
##  $ So    : int  1 0 1 0 0 0 1 1 1 0 ...
##  $ Ed    : num  9.1 11.3 8.9 12.1 12.1 11 11.1 10.9 9 11.8 ...
##  $ Po1   : num  5.8 10.3 4.5 14.9 10.9 11.8 8.2 11.5 6.5 7.1 ...
##  $ Po2   : num  5.6 9.5 4.4 14.1 10.1 11.5 7.9 10.9 6.2 6.8 ...
##  $ LF    : num  0.51 0.583 0.533 0.577 0.591 0.547 0.519 0.542 0.553 0.632 ...
##  $ M.F   : num  95 101.2 96.9 99.4 98.5 ...
##  $ Pop   : int  33 13 18 157 18 25 4 50 39 7 ...
##  $ NW    : num  30.1 10.2 21.9 8 3 4.4 13.9 17.9 28.6 1.5 ...
##  $ U1    : num  0.108 0.096 0.094 0.102 0.091 0.084 0.097 0.079 0.081 0.1 ...
##  $ U2    : num  4.1 3.6 3.3 3.9 2 2.9 3.8 3.5 2.8 2.4 ...
##  $ Wealth: int  3940 5570 3180 6730 5780 6890 6200 4720 4210 5260 ...
##  $ Ineq  : num  26.1 19.4 25 16.7 17.4 12.6 16.8 20.6 23.9 17.4 ...
##  $ Prob  : num  0.0846 0.0296 0.0834 0.0158 0.0414 ...
##  $ Time  : num  26.2 25.3 24.3 29.9 21.3 ...
##  $ Crime : int  791 1635 578 1969 1234 682 963 1555 856 705 ...
```

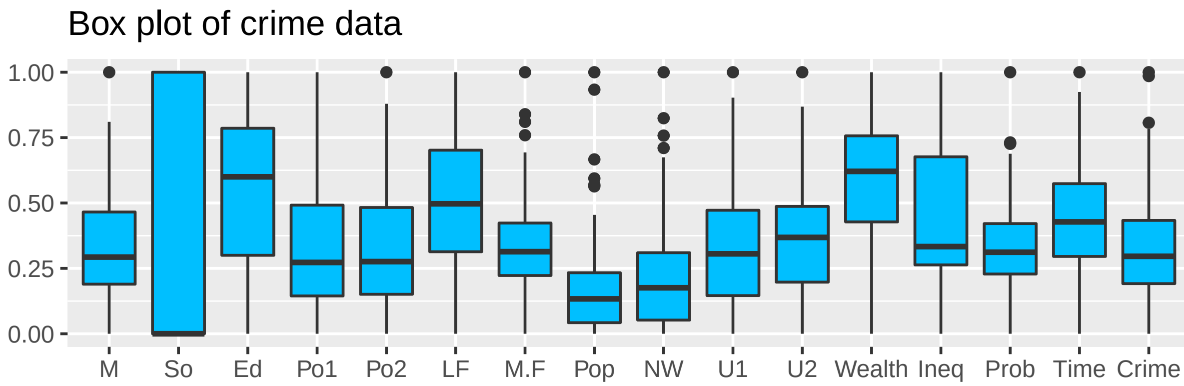
One categorical (binary) variable (So), and the rest are continuous variables. Let's get a summarized view:

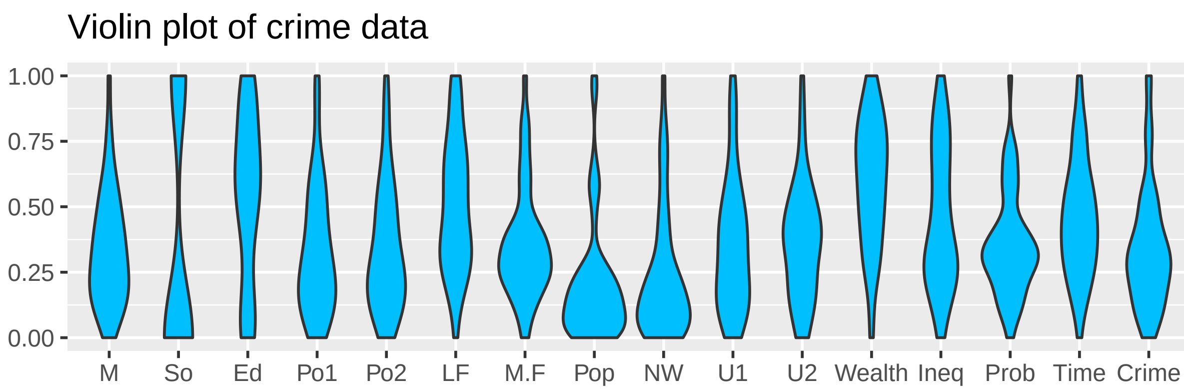
```
summary(crime_data)
```

```
##         M               So              Ed              Po1
##  Min.   :11.90   Min.   :0.0000   Min.   : 8.70   Min.   : 4.50
##  1st Qu.:13.00   1st Qu.:0.0000   1st Qu.: 9.75   1st Qu.: 6.25
##  Median :13.60   Median :0.0000   Median :10.80   Median : 7.80
##  Mean   :13.86   Mean   :0.3404   Mean   :10.56   Mean   : 8.50
##  3rd Qu.:14.60   3rd Qu.:1.0000   3rd Qu.:11.45   3rd Qu.:10.45
##  Max.   :17.70   Max.   :1.0000   Max.   :12.20   Max.   :16.60
##       Po2               LF              M.F              Pop
##  Min.   : 4.100   Min.   :0.4800   Min.   : 93.40   Min.   :  3.00
##  1st Qu.: 5.850   1st Qu.:0.5305   1st Qu.: 96.45   1st Qu.: 10.00
##  Median : 7.300   Median :0.5600   Median : 97.70   Median : 25.00
##  Mean   : 8.023   Mean   :0.5612   Mean   : 98.30   Mean   : 36.62
##  3rd Qu.: 9.700   3rd Qu.:0.5930   3rd Qu.: 99.20   3rd Qu.: 41.50
##  Max.   :15.700   Max.   :0.6410   Max.   :107.10   Max.   :168.00
##        NW              U1                U2            Wealth
##  Min.   : 0.20   Min.   :0.07000   Min.   :2.000   Min.   :2880
##  1st Qu.: 2.40   1st Qu.:0.08050   1st Qu.:2.750   1st Qu.:4595
##  Median : 7.60   Median :0.09200   Median :3.400   Median :5370
##  Mean   :10.11   Mean   :0.09547   Mean   :3.398   Mean   :5254
##  3rd Qu.:13.25   3rd Qu.:0.10400   3rd Qu.:3.850   3rd Qu.:5915
##  Max.   :42.30   Max.   :0.14200   Max.   :5.800   Max.   :6890
##       Ineq            Prob              Time           Crime
##  Min.   :12.60   Min.   :0.00690   Min.   :12.20   Min.   : 342.0
##  1st Qu.:16.55   1st Qu.:0.03270   1st Qu.:21.60   1st Qu.: 658.5
##  Median :17.60   Median :0.04210   Median :25.80   Median : 831.0
##  Mean   :19.40   Mean   :0.04709   Mean   :26.60   Mean   : 905.1
##  3rd Qu.:22.75   3rd Qu.:0.05445   3rd Qu.:30.45   3rd Qu.:1057.5
##  Max.   :27.60   Max.   :0.11980   Max.   :44.00   Max.   :1993.0
```

We see different orders of magnitude among the data, which indicates that standardization is needed for PCA. Furthermore, is looks like that the distributions deviate from the normal distribution. Let's do a temporary min-max scaling of the dataframe (so we can see all the columns on the same graph) and make a violin plot to better visualize the distributions, and a box-plot to see if there any strange data points:

```r
# Min max scale
temp <- sapply(crime_data, function (x) (x - min(x)) / (max(x) - min(x)))

p1 <- ggplot(melt(temp), aes(x = Var2, y = value)) +
geom_boxplot(fill = 'deepskyblue1') +
ylab("") +
labs(title = "Box plot of crime data") +
xlab("")

p2 <- ggplot(melt(temp), aes(x = Var2, y = value)) +
geom_violin(fill = 'deepskyblue1') +
ylab("") +
labs(title = "Violin plot of crime data") +
xlab("")

multiplot(p1, p2, cols = 1)
```





Indeed, it looks like some varaibles are highly skewed (Pop and NW for example), which indicates heteroskedasticity. Furthermore, the box-plot indicates quite a few extreme values - potential outliers. In this case, a log-transform would be beneficial, as it will make the distributions more normally distributed, it will stabilize the variances, and will also reduce the impact of the extreme values shown in the box-plot. On the other hand, it will also make our model multiplative on the raw scale instead of additive, but that is not a problem in this case.

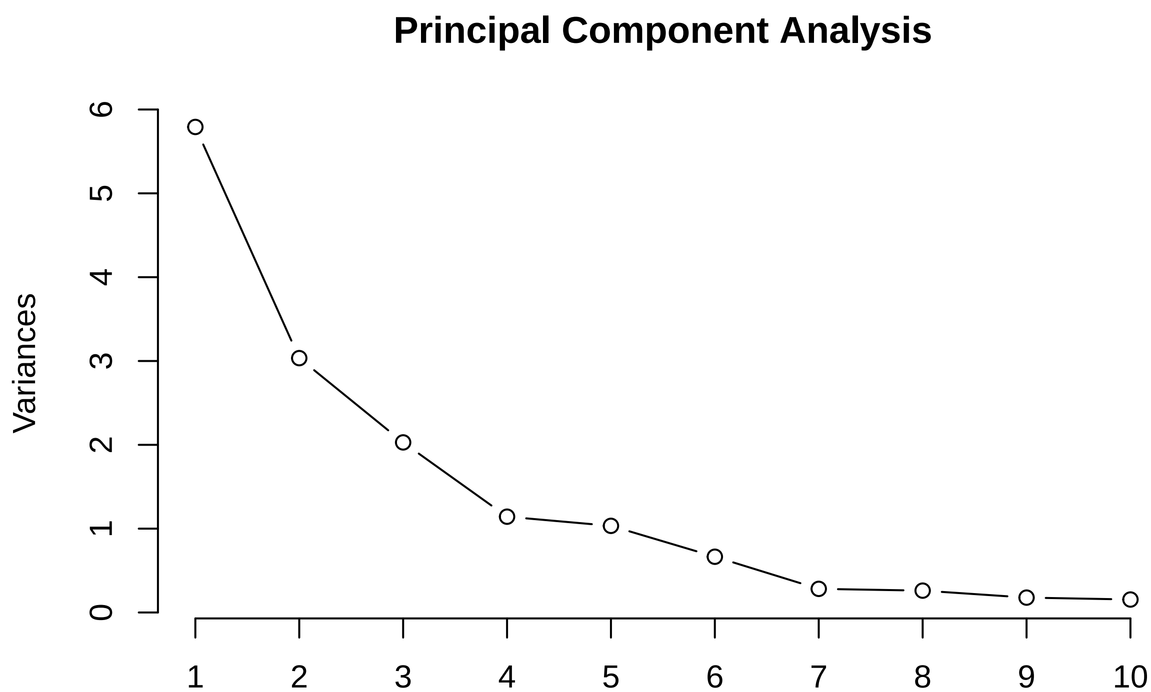
Do note that PCA does **not** make any assumption about data notmality, which means that PCA does not care if we log-transform our data. On the other hand, it does try to find the dimensions within the data that capture most of the variance. Variance, though, is a good measure for spread on *symmetric* distributions, but it can fail when we consider highly skewed or assymetric distributions. Log-transforming here will probably make the results more robust.

Pre-processing

Now, we could apply a Box-cox tranformation on each variable individually to get a more normal-looking dataset, but we will settle with a more generic log-transform on all columns, due to its ease of use in calculating the linear coefficients backwards at the end. Let's log transform the data, and standardize them:

```
# Log transform over the entire dataset
log_crime <- log(crime_data)
log_crime$So <- crime_data$So # Do not log transform the categorical variable

# Isolate the target variable
log_target <- log_crime$Crime
log_crime$Crime <- NULL
```

```r
# Apply PCA
ir.pca <- prcomp(log_crime, center = TRUE, scale. = TRUE)
plot(ir.pca, type = 'l', main='Principal Component Analysis')
```




```r
# Re-attach the target variable
log_crime$Crime <- log_target

# Get the mean and standard deviation of each column - for later ref
log_means <- colMeans(log_crime)
log_sd <- sapply(log_crime, function(x) sd(x))
```

We can see that most of the variance can be captured from the fisrt four components. There is still a slight reduction for components 5 to 7, after which the effect of additional components diminishes.

```r
# Log transform over the entire dataset
summary(ir.pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4    PC5     PC6     PC7
## Standard deviation     2.4066 1.7420 1.4244 1.06903 1.0166 0.81548 0.53085
## Proportion of Variance 0.3861 0.2023 0.1353 0.07619 0.0689 0.04433 0.01879
## Cumulative Proportion  0.3861 0.5884 0.7237 0.79986 0.8688 0.91309 0.93188
##                            PC8     PC9    PC10    PC11    PC12    PC13
## Standard deviation     0.51071 0.42137 0.39328 0.38183 0.34670 0.29621
## Proportion of Variance 0.01739 0.01184 0.01031 0.00972 0.00801 0.00585
## Cumulative Proportion  0.94927 0.96111 0.97142 0.98114 0.98915 0.99500
##                           PC14    PC15
```

```
## Standard deviation     0.26395 0.07303
## Proportion of Variance 0.00464 0.00036
## Cumulative Proportion  0.99964 1.00000
```

Indeed, the first component captures a third of the total variance, whereas the top 4 components account for 80% of the variance in the data. A modeling choice at this point is to limit ourselves on the first four components only.

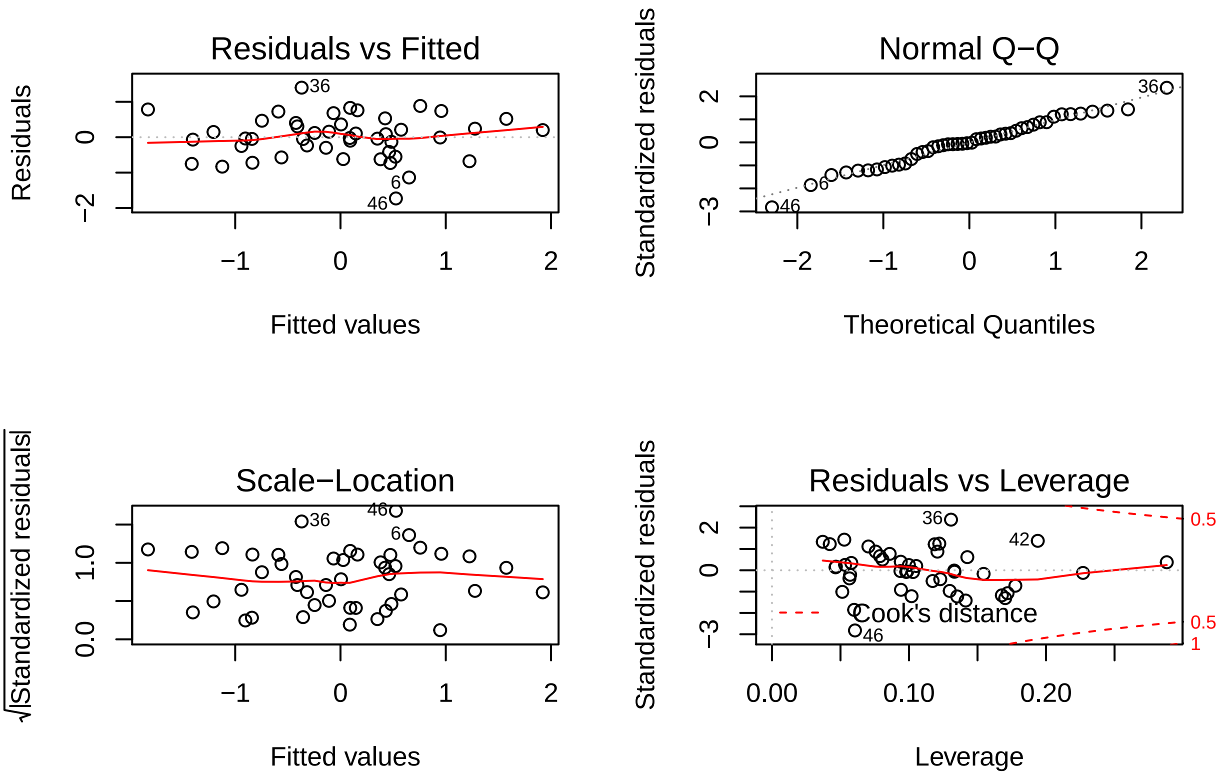
Modeling

Let's extract these principal components, and make a linear model using the training set:

```
# Get the first four principal components
crime_prc <- ir.pca$x[, 1:4]

# And append the log_crime on the matrix, after standardizing it
target <- (log_crime$Crime - log_means["Crime"]) / log_sd["Crime"]

crime_prc <- cbind(crime_prc, target)
colnames(crime_prc)[length(colnames(crime_prc))] <- 'log_crime'

# Make a linear model
temp <- data.frame(crime_prc)
rownames(temp) <-1:length(rownames(temp))
model <- lm(log_crime ~., data = temp[-18, ])

# Show results
summary(model)
```

```
##
## Call:
## lm(formula = log_crime ~ ., data = temp[-18, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.72944 -0.38670 -0.01525  0.39238  1.39888
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.02946    0.09338   0.315 0.754021
## PC1          0.14686    0.03875   3.790 0.000486 ***
## PC2         -0.23828    0.05361  -4.444 6.56e-05 ***
## PC3          0.01001    0.06556   0.153 0.879391
## PC4         -0.59376    0.09272  -6.404 1.15e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6324 on 41 degrees of freedom
## Multiple R-squared:  0.643,  Adjusted R-squared:  0.6082
## F-statistic: 18.46 on 4 and 41 DF,  p-value: 9.586e-09
```

```
par(mfrow=c(2,2))
plot(model)
```



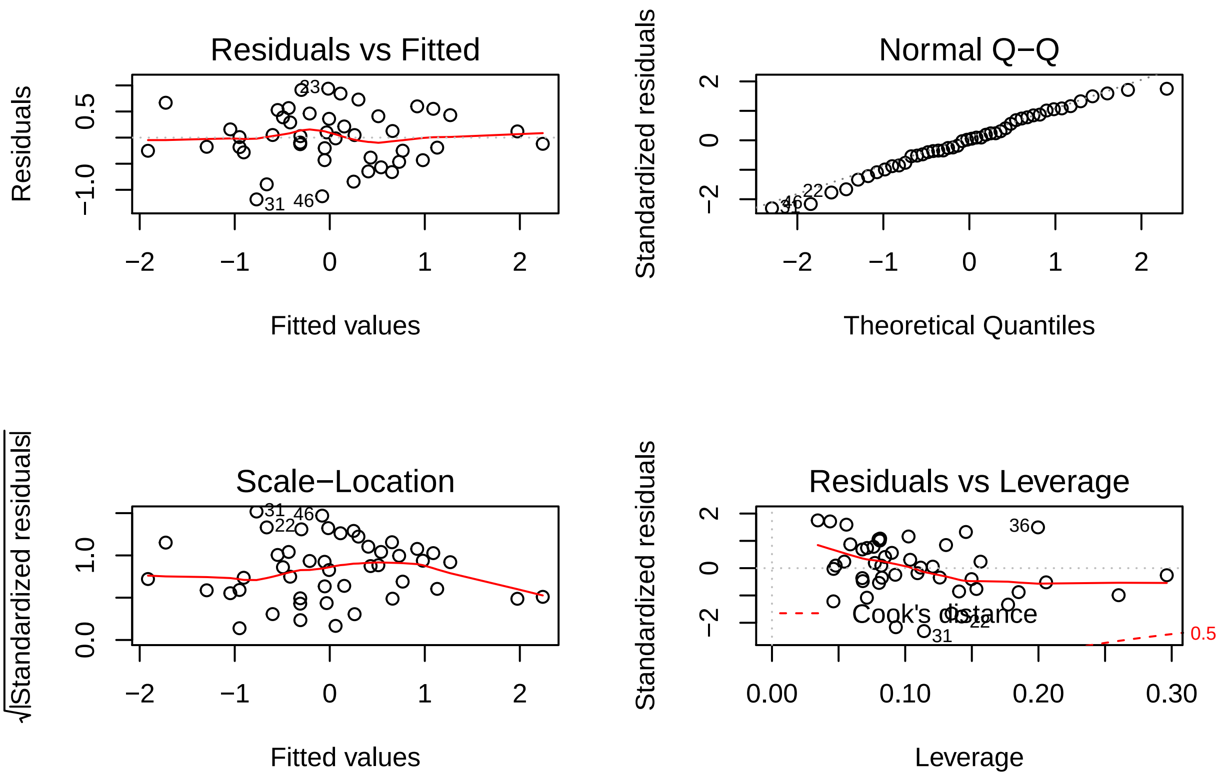
It looks like our model is not doing so well. Apart from the low R^2 value, the third principal component is not statistically significant, and the std. residual vs fitted plot shows a slight angle, indicating somewhat unequal variance (heteroskedasticity). Let's skp the third component, and add the fifth:

```
# Get the principal components
p_comps_to_use <- c(1, 2, 4, 5)
crime_prc <- ir.pca$x[, p_comps_to_use]

# And append the log_crime on the matrix
crime_prc <- cbind(crime_prc, target)
colnames(crime_prc)[length(p_comps_to_use) + 1] <- 'log_crime'

# Make a linear model
temp <- data.frame(crime_prc)
rownames(temp) <-1:length(rownames(temp))
model <- lm(log_crime ~., data = temp[-42, ])

# Show results
summary(model)
```

```
##
## Call:
## lm(formula = log_crime ~ ., data = temp[-42, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.18339 -0.27524  0.01895  0.40418  0.93693
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.03867    0.08068  -0.479 0.634241
## PC1          0.14606    0.03340   4.373 8.19e-05 ***
## PC2         -0.27690    0.04877  -5.677 1.24e-06 ***
## PC4         -0.58877    0.07801  -7.547 2.81e-09 ***
## PC5          0.36232    0.08529   4.248 0.000121 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5451 on 41 degrees of freedom
## Multiple R-squared:  0.7286, Adjusted R-squared:  0.7021
## F-statistic: 27.51 on 4 and 41 DF,  p-value: 3.913e-11
```

```r
par(mfrow=c(2,2))
plot(model)
```



The model is performing significantly better when excluding the third component, which does verify its redundancy. Moreover, the diagnostic plots do not indicate any problem with the model at all.

Coefficients of original variables

To get the coefficients in terms of the original variables, we need to multiply the coefficients of the linear model above, with the eigenvectors determined by PCA (for the components used in the model):

```r
# Make a matrix that holds the product of each linear coeff with its
# principal component for all original variables
temp <- ir.pca$rotation[, p_comps_to_use] * model$coefficients[-1]

# and sum each row
orig_coeffs <- apply(temp, 1, function(x) sum(x))
orig_coeffs
```

```
##           M          So          Ed         Po1         Po2          LF
## -0.01711888  0.05736021 -0.27484973  0.35944216  0.17643819 -0.30734096
##         M.F         Pop          NW          U1          U2      Wealth
## -0.07610203  0.26834560  0.13423669 -0.10653177  0.09913251  0.20390639
##        Ineq        Prob        Time
## -0.03428879  0.26786368  0.36112830
```

Keep in mind that these coefficients correspond to the *log-transformed and standardized* variables.

Prediction

The final step is to make a prediction for the same city as in Question 8.2. After we get the city's data, we need to calculate the log value of each predictor, subtract from that the mean and divide by the standard deviation (of the same predictor) from the training data before we use our model. The reverse process has to be applied to get the predicter crime: Since the model predicts a log-transformed, standardized crime value, the output we get from the model must be multiplied by its standard deviation, its mean must be added, and then its exponential value needs to be calculated:

```r
new_data <- data.frame('M' = 14.0,
                       'So' = 0,
                       'Ed' = 10.0,
                       'Po1' = 12.0,
                       'Po2' = 15.5,
                       'LF' = 0.64,
                       'M.F' = 94.0,
                       'Pop' = 150.0,
                       'NW' = 1.1,
                       'U1' = 0.12,
                       'U2' = 3.6,
                       'Wealth' = 3200.0,
                       'Ineq' = 20.1,
                       'Prob' = 0.04,
                       'Time' = 39.0)

# Log transform on all (except the categorical) vars
temp <- new_data$So
log_new <- log(new_data)
log_new$So <- temp

# Apply the PCA from the training set (which will also do the standardization)
test_prc <- predict(ir.pca, log_new)
test_prc <- test_prc[, p_comps_to_use]

# Get the result
y_hat <- predict(model, data.frame(t(test_prc)), interval = 'prediction', level = 0.95)
```

```r
# Un-standardize
y_hat <- y_hat * log_sd["Crime"] + log_means["Crime"]

# And reverse the log transformation
y_hat <- exp(y_hat)

y_hat
```

```
##        fit     lwr      upr
## 1 1004.472 627.578 1607.711
```

Conclusions

Principal components 1, 2, 4, 5 were used for the log-log model, which has comparable performance (very similar adj. R^2 alues) as the model in Question 8.2. The crime rate prediction for the city with the given data is 1004.472, or between 627.578 - 1607.711, (prediction interval) at a 95% confidence level.

Question 10.1

At first, let's clear the workspace and load the necessary libraries.

```r
# Remove all variables from te previous analysis
rm(list=ls(all=TRUE))

# Load the necesary libraries
library(ggplot2)
library(reshape2)
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.4.4
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.4.4
```

```r
library(rfUtilities)
```

```
## Warning: package 'rfUtilities' was built under R version 3.4.4
## rfUtilities 2.1-3
## Type rfu.news() to see new features/changes/bug fixes.
```

```r
# Read in the data
df <- read.table("9.1uscrimeSummer2018.txt", header = T)
```

Regression tree

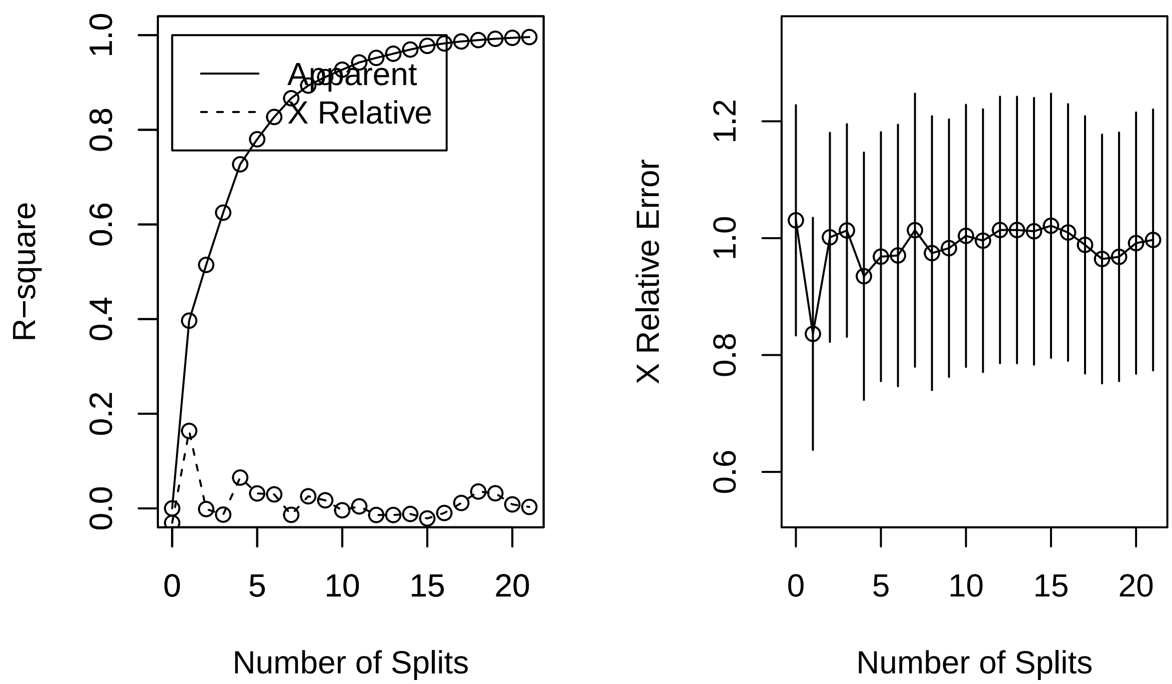
In general, tree algorithms are not affected by the order of magnitude of each predictor (especially if they don't present extreme differences within the dataset), and they do not make any assumptions about the underlying structure of the data - which means that they are quite robust to heteroskedasticity. However, since we are going to apply a CART here, a log-transformation would be beneficial: CART models use analysis of variance to perform splits, and variance is indeed very sensible to outliers and skewed data, which is the reason why transforming the variables can potentially improve model accuracy. As in the previous question, we could apply a Box-cox transformation to each variable individually, to get each distribution as close to normal as possible. But due to ease of use, a general log transform will be used here as well.

Do note that we do not need to split the data and perform cross validation manually, as rpart will do that for us:

```r
# Log transform the data (not the categorical variable)
temp <- df$So
log.df <- log(df)
log.df$So <- temp

set.seed(42)

# Set the control settings
control <- rpart.control(minsplit = floor( .05 * nrow(log.df)),
                         # min no of points in a node to attempt a split
                         # 5% of total points here
                         cp = 0.001, # Do not attempt
                         # splits that don't decrease overall
                         # lack of fit by a factor of cp
                         xval = 10) # No. of folds in cv



# Grow a regresion tree
model <- rpart(Crime ~., log.df, method = 'anova', control = control)

# plot approximate R-squared and relative error for different splits (2 plots)
par(mfrow=c(1,2))
rsq.rpart(model)
```

```
##
## Regression tree:
## rpart(formula = Crime ~ ., data = log.df, method = "anova", control = control)
##
## Variables actually used in tree construction:
## [1] Ed     LF     M      NW     Po1    Pop    Prob   U2     Wealth
##
## Root node error: 7.7726/47 = 0.16537
##
## n= 47
##
##           CP nsplit rel error  xerror    xstd
## 1  0.3966662      0 1.0000000 1.03057 0.19715
```

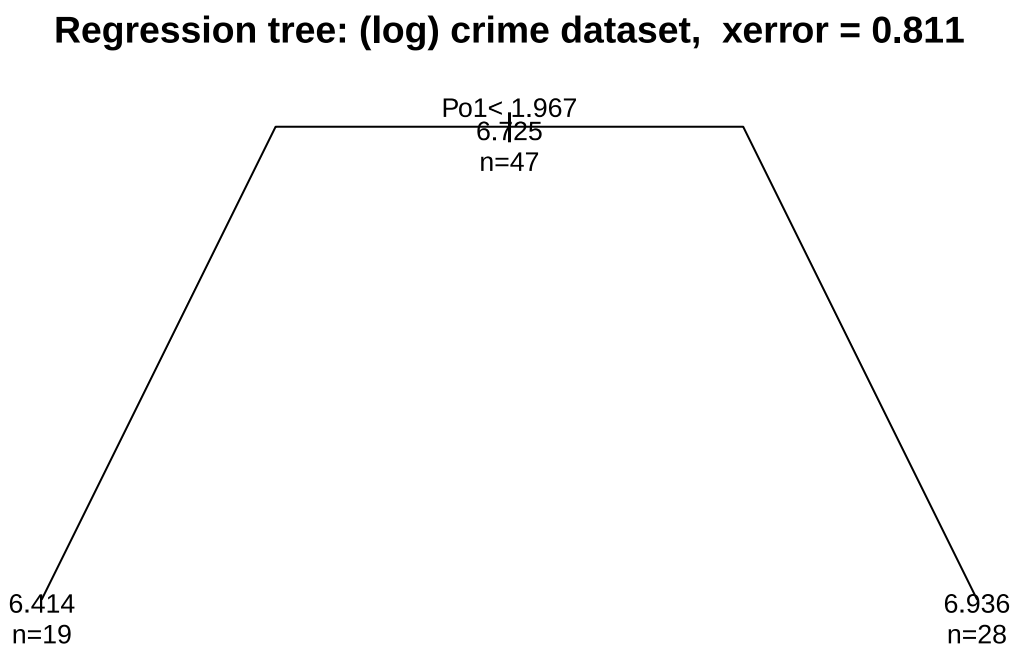
```
## 2  0.1178609      1 0.6033338 0.83626 0.19868
## 3  0.1104223      2 0.4854729 1.00145 0.17903
## 4  0.1021444      3 0.3750506 1.01315 0.18206
## 5  0.0529546      4 0.2729062 0.93487 0.21174
## 6  0.0470756      5 0.2199516 0.96839 0.21308
## 7  0.0395380      6 0.1728760 0.97042 0.22387
## 8  0.0267925      7 0.1333380 1.01366 0.23376
## 9  0.0180113      8 0.1065456 0.97442 0.23439
## 10 0.0155331      9 0.0885342 0.98292 0.22045
## 11 0.0153411     10 0.0730011 1.00395 0.22443
## 12 0.0096125     11 0.0576600 0.99574 0.22487
## 13 0.0088267     12 0.0480475 1.01395 0.22817
## 14 0.0087767     13 0.0392208 1.01395 0.22817
## 15 0.0078107     14 0.0304441 1.01181 0.22833
## 16 0.0051681     15 0.0226333 1.02125 0.22615
## 17 0.0041756     16 0.0174653 1.00974 0.21971
## 18 0.0028847     17 0.0132896 0.98855 0.22027
## 19 0.0026783     18 0.0104049 0.96436 0.21296
## 20 0.0020630     19 0.0077266 0.96803 0.21279
## 21 0.0015969     20 0.0056636 0.99154 0.22365
## 22 0.0010000     21 0.0040668 0.99700 0.22336
```



We want to select a tree size that minimizes the cross validation error (the xerror column printed by printcp()). We must, therefore, selct the complexity parameter associated with the minimum error, and put that to the prune() function of rpart, to prune the tree to the right size to reduce overfitting.

We see from the output that this value occurs for a cp equal to 0.117861, for which the cross validation error reaches a minimum of 0.83626 with a standard deviation of 0.19868. Let's prune our tree, and plot it:

```
model <- prune(model, cp = 0.117861)

# plot the pruned decision tree
par(mfrow = c(1,1), xpd = NA)
plot(model, uniform = T, branch = 0.5,
    main = "Regression tree: (log) crime dataset,  xerror = 0.811")
text(model, use.n = T, all = T, cex = 0.8)
```



That is quite interesting. Apparently, the best strategy is to decide on the target variable (crime rate) based on Po1 only (!)

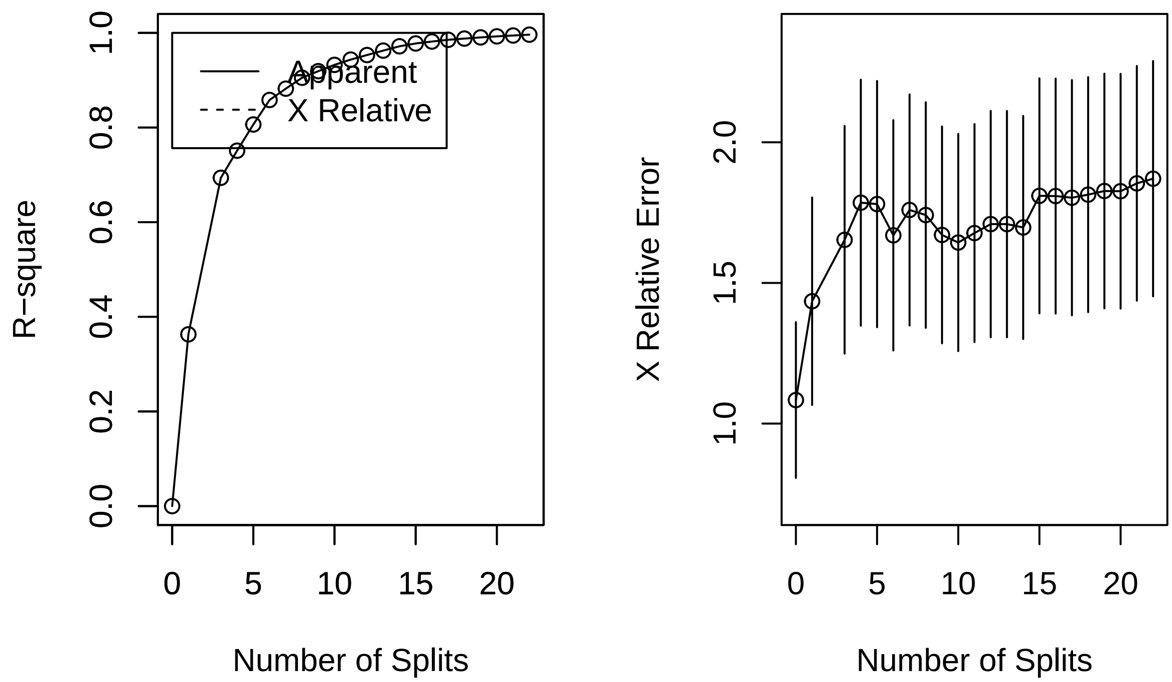
Regression tree on the original dataset

For the sake of curiosity, let's make another tree for the non-log (original) dataset, and see what is the minimum error we can achieve on the cross validation. Doing so, it will show us if the log tranformation on the entire dataset was indeed a good idea, or just meaningless pre-processing:

```
# Grow a regresion tree using the sae settings as before, on the original data
model_orig <- rpart(Crime ~., df, method = 'anova', control = control)

# plot approximate R-squared and relative error for different splits (2 plots)
par(mfrow=c(1,2))
rsq.rpart(model_orig)
```

```
##
```

```
## Regression tree:
## rpart(formula = Crime ~ ., data = df, method = "anova", control = control)
##
## Variables actually used in tree construction:
##  [1] Ed     LF     M      NW     Po1    Pop    Prob   U1     U2     Wealth
##
## Root node error: 6880928/47 = 146403
##
## n= 47
##
##           CP nsplit rel error xerror    xstd
## 1  0.3629629      0 1.0000000 1.0836 0.27695
## 2  0.1654002      1 0.6370371 1.4349 0.36883
## 3  0.0573014      3 0.3062366 1.6532 0.40469
## 4  0.0553887      4 0.2489352 1.7852 0.43736
## 5  0.0517317      5 0.1935465 1.7803 0.43778
## 6  0.0239291      6 0.1418148 1.6692 0.40946
## 7  0.0230593      7 0.1178857 1.7595 0.41081
## 8  0.0139168      8 0.0948264 1.7413 0.40091
## 9  0.0133099      9 0.0809096 1.6706 0.38551
## 10 0.0111671     10 0.0675997 1.6436 0.38611
## 11 0.0095822     11 0.0564326 1.6773 0.38740
## 12 0.0093761     12 0.0468504 1.7091 0.40233
## 13 0.0091314     13 0.0374744 1.7091 0.40233
## 14 0.0059906     14 0.0283429 1.6971 0.39666
## 15 0.0040181     15 0.0223523 1.8096 0.41774
## 16 0.0037030     16 0.0183343 1.8088 0.41781
## 17 0.0025962     17 0.0146312 1.8030 0.41829
## 18 0.0024648     18 0.0120351 1.8139 0.41781
## 19 0.0021855     19 0.0095703 1.8268 0.41732
## 20 0.0018140     20 0.0073848 1.8260 0.41736
## 21 0.0017920     21 0.0055708 1.8540 0.41711
## 22 0.0010000     22 0.0037788 1.8705 0.41838
```



Interesting. Minimum cross validation error 1.08 with zero splits (!)

The log transfrormation has definitely helped. It is possible that with the variable-specific Box-cox transformation we might achieve even better results.

PCA'ed, log-dataset (the same as Question 9.1)

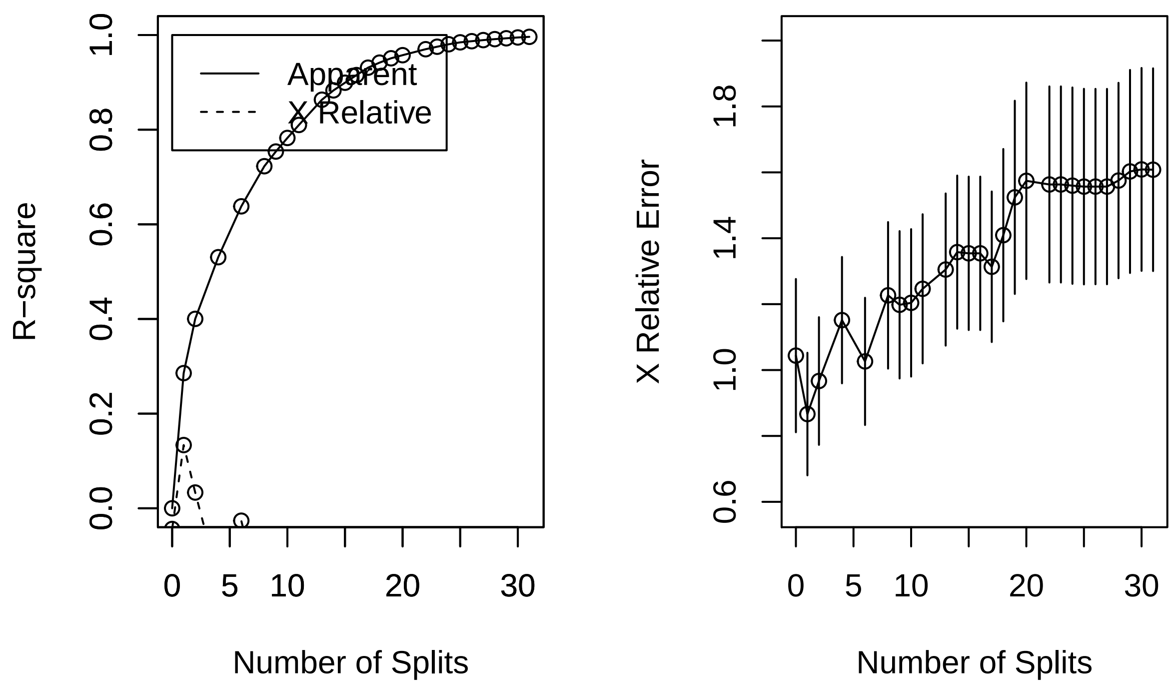
Again for the sake of curiosity, let's apply PCA on the log transformed dataset, and then create a regression tree:

```
# Log transform the data (not the categorical variable)
temp <- df$So
log.df <- log(df)
log.df$So <- temp

# Isolate the target variable
log_target <- log.df$Crime
log.df$Crime <- NULL

# Apply PCA
ir.pca <- prcomp(log.df, center = TRUE, scale. = TRUE)

# Standardize the target variable
log_target <- (log_target - mean(log_target)) / sd(log_target)

# Get the first three principal components
```

```r
p_comps_to_use <- 1:4
crime_prc <- ir.pca$x[, p_comps_to_use]

# And append the log_crime on the matrix
crime_prc <- cbind(crime_prc, log_target)
colnames(crime_prc)[4] <- 'log_crime'

# Grow a regresion tree
model_PCA <- rpart(log_crime ~., data.frame(crime_prc),
                   method = 'anova', control = control)

# plot approximate R-squared and relative error for different splits (2 plots)
par(mfrow=c(1,2))
rsq.rpart(model_PCA)
```

```
##
## Regression tree:
## rpart(formula = log_crime ~ ., data = data.frame(crime_prc),
##     method = "anova", control = control)
##
## Variables actually used in tree construction:
## [1] log_target PC1        PC2        PC3
##
## Root node error: 52.57/47 = 1.1185
##
## n= 47
##
##           CP nsplit rel error  xerror    xstd
## 1  0.2856265      0 1.0000000 1.04379 0.23256
## 2  0.1145939      1 0.7143735 0.86642 0.18598
## 3  0.0651809      2 0.5997796 0.96667 0.19372
## 4  0.0537247      4 0.4694178 1.15150 0.19177
## 5  0.0423857      6 0.3619684 1.02625 0.19307
## 6  0.0310337      8 0.2771969 1.22672 0.22237
## 7  0.0286719      9 0.2461632 1.19803 0.22361
## 8  0.0274811     10 0.2174913 1.20390 0.22375
## 9  0.0266682     11 0.1900102 1.24650 0.22627
## 10 0.0197584     13 0.1366739 1.30505 0.23086
## 11 0.0161465     14 0.1169155 1.35788 0.23240
## 12 0.0161216     15 0.1007690 1.35422 0.23276
## 13 0.0156905     16 0.0846474 1.35422 0.23276
## 14 0.0106856     17 0.0689569 1.31346 0.22847
## 15 0.0091592     18 0.0582713 1.40933 0.26150
## 16 0.0065391     19 0.0491121 1.52417 0.29318
## 17 0.0063166     20 0.0425730 1.57432 0.29804
## 18 0.0052659     22 0.0299398 1.56324 0.29765
## 19 0.0051670     23 0.0246739 1.56324 0.29765
## 20 0.0040619     24 0.0195069 1.55966 0.29799
## 21 0.0024179     25 0.0154450 1.55675 0.29685
## 22 0.0023296     26 0.0130272 1.55685 0.29650
## 23 0.0020365     27 0.0106976 1.55685 0.29650
## 24 0.0017962     28 0.0086611 1.57514 0.29657
## 25 0.0017243     29 0.0068650 1.60270 0.30810
## 26 0.0013200     30 0.0051407 1.60890 0.30785
```

```
## 27 0.0010000     31 0.0038207 1.60807 0.30764
```

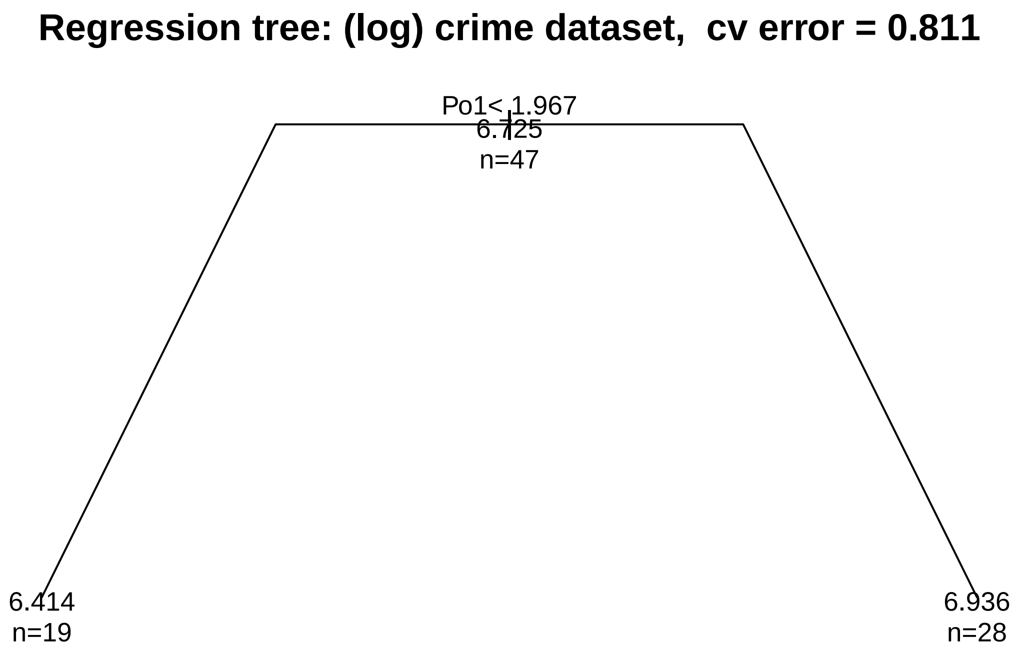


Almost identical results with the first log-transformed model. A minimum cross validation error of 0.86934 with a standard deviation of 0.18.

Final tree

For completeness, the final tree is shown again below. The only preprocessing step needed is a log transformation on the original dataset, excluding the one categorical variable (So).

```
par(mfrow = c(1,1), xpd = NA)
plot(model, uniform = T, branch = 0.5,
    main = "Regression tree: (log) crime dataset,  cv error = 0.811")
text(model, use.n = T, all = T, cex = 0.8)
```



Apparently, the best strategy is to split the data into two buckets, depending on the value of Po1. *It is extremely possible that with more observations the resulting tree would radically change.*

Random Forest

Now let's generate a large number of bootstrapped trees based on random samples of the variables, and decide a final predicted outcome by combining the results across all of the trees (averaging).

Since the log-transformed dataset is showing somewhat improved results across all models used in this homework, we will firstly try with that:

```
# Log transform the data (not the categorical variable)
temp <- df$So
log.df <- log(df)
log.df$So <- temp

# Reproducibility
set.seed(40)

# Grow the forest
model <- randomForest(Crime ~., log.df, ntree = 10)

# See results
print(model) # view results
```

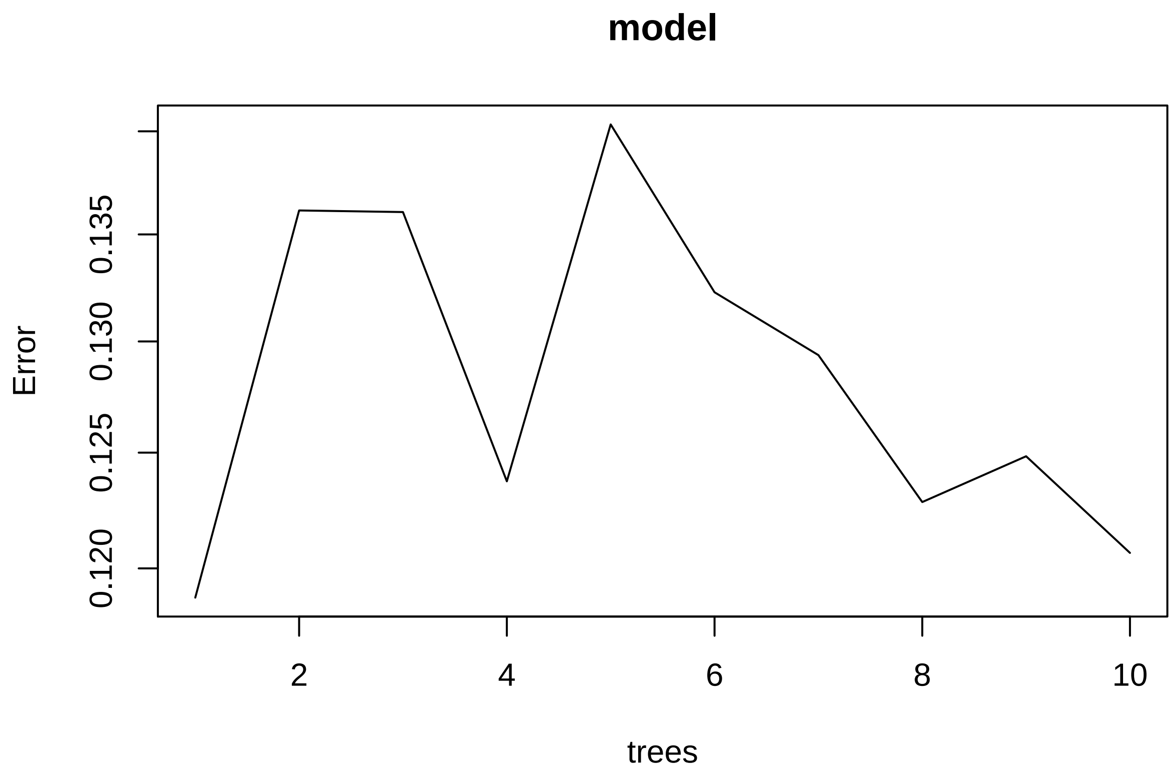
```
##
## Call:
```

```
##  randomForest(formula = Crime ~ ., data = log.df, ntree = 10)
##                Type of random forest: regression
##                      Number of trees: 10
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 0.1206586
##                     % Var explained: 27.04
```

```r
importance(model) # importance of each predictor
```

```
##        IncNodePurity
## M         0.10978954
## So        0.04830768
## Ed        0.28326439
## Po1       1.44402663
## Po2       1.09126605
## LF        0.48574330
## M.F       0.09236308
## Pop       0.73433757
## NW        0.88797264
## U1        0.13257128
## U2        0.21741114
## Wealth    0.36645110
## Ineq      0.45447885
## Prob      0.59069052
## Time      0.28901777
```

```r
plot(model, log="y")
```




```r
cv <- rf.crossValidation(model, log.df,
                         p = 0.15, # Proportion data withhold
                         n = 10, # Number of folds
                         seed = 43, # Set random seed
                         normalize = T, # Min-max normalizate mse, mbe and mae
                         bootstrap = F) # Apply bootstrap sampling
```

```
## running: regression cross-validation with 10 iterations
```

```r
summary(cv)
```

```
## Fit MSE = 0.1271075
## Fit percent variance explained = 27.04
## Median permuted MSE = 0.04187178
## Median permuted percent variance explained = 75.865
## Median permuted percent variance explained = 75.865
## Median cross-validation RMSE = 0.1403668
## Median cross-validation MBE = -0.006093352
## Median cross-validation MAE = 0.1115596
## RMSE cross-validation error variance = 0.000938081
## MBE cross-validation error variance = 0.002515258
## MAE cross-validation error variance = 0.0005046772
```

Once again, Po1 seems to be the most important variable, followed by Po2 and NW. This observation agrees with past analyses we have done with different kinds of models. Furthermore, the cross-validation results indicate a median cross-validation RMSE of 0.158, with a 77.3% variance explained, which indicates that the model does provide a relatively good fit (not necessarily better than a multi linear regression though). On the other hand, comparable errors can be achieved with just one tree, as the plot shows. This is understandanble, as we have too few observations to train a random forest model.

Question 10.2

A logistic regression model would be appropriate to identify the probability of rain at any given day. Predictors that could be used include day of the year, mean expected temperature, humidity, location and so on.

Question 10.3

At first, let's have a look at the description of the data:

- Attribute 1: (qualitative) Status of existing checking account
- Attribute 2: (numerical) Duration in month
- Attribute 3: (qualitative) Credit history
- Attribute 4: (qualitative) Purpose
- Attribute 5: (numerical) Credit amount
- Attribute 6: (qualitative) Savings account/bonds
- Attribute 7: (qualitative) Present employment since
- Attribute 8: (numerical) Installment rate in percentage of disposable income
- Attribute 9: (qualitative) Personal status and sex
- Attribute 10: (qualitative) Other debtors / guarantors
- Attribute 11: (numerical) Present residence since
- Attribute 12: (qualitative) Property
- Attribute 13: (numerical) Age in years
- Attribute 14: (qualitative) Other installment plans
- Attribute 15: (qualitative) Housing
- Attribute 16: (numerical) Number of existing credits at this bank
- Attribute 17: (qualitative) Job
- Attribute 18: (numerical) Number of people being liable to provide maintenance
- Attribute 19: (qualitative) Telephone
- Attribute 20: (qualitative) foreign worker
- Target (attribute 21): 1 = Good creditor, 2 = Bad creditor

Let's clean our workspace and load the necessary libraries:

```
# Remove all variables from te previous analysis
rm(list=ls(all=TRUE))

# Load the necesary libraries
library(ggplot2)
library(Rmisc)
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.4.4
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

```
library(gmodels)
```

```
## Warning: package 'gmodels' was built under R version 3.4.4
```

```r
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
##
## Attaching package: 'tidyr'
## The following object is masked from 'package:reshape2':
##
##     smiths
```

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.4.4
```

```r
# Read in the data
credit <- read.table("10.3germancreditSummer2018.txt", header = F)
colnames(credit) <- c("acc_status",
                      "duration",
                      "history",
                      "purpose",
                      "amount",
                      "savings_accs",
                      "employed_since",
                      "instal_income_ratio",
                      "pers_status",
                      "other_debtors",
                      "resid_since",
                      "property_type",
                      "age",
                      "other_plans",
                      "housing",
                      "no_credits",
                      "job",
                      "ppl_liable",
                      "phone",
                      "foreigner",
                      "creditor_type")

credit$creditor_type <- as.factor(credit$creditor_type - 1) # 0: good, 1: bad
levels(credit$creditor_type) <- c("Good", "Bad")
# Show the structure of the data
str(credit)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ acc_status         : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...
##  $ duration           : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ history            : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...
##  $ purpose            : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...
##  $ amount             : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ savings_accs       : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...
##  $ employed_since     : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...
##  $ instal_income_ratio: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ pers_status        : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...
##  $ other_debtors      : Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...
##  $ resid_since        : int  4 2 3 4 4 4 4 2 4 2 ...
```

```
##  $ property_type      : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...
##  $ age                : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ other_plans        : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ housing            : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...
##  $ no_credits         : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ job                : Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...
##  $ ppl_liable         : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ phone              : Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreigner          : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
##  $ creditor_type      : Factor w/ 2 levels "Good","Bad": 1 2 1 1 2 1 1 1 1 2 ...
```

```r
summary(credit)
```

```
##  acc_status    duration    history      purpose        amount
##  A11:274    Min.   : 4.0   A30: 40   A43    :280   Min.   :  250
##  A12:269    1st Qu.:12.0   A31: 49   A40    :234   1st Qu.: 1366
##  A13: 63    Median :18.0   A32:530   A42    :181   Median : 2320
##  A14:394    Mean   :20.9   A33: 88   A41    :103   Mean   : 3271
##             3rd Qu.:24.0   A34:293   A49    : 97   3rd Qu.: 3972
##             Max.   :72.0             A46    : 50   Max.   :18424
##                                      (Other): 55
##  savings_accs employed_since instal_income_ratio pers_status other_debtors
##  A61:603      A71: 62        Min.   :1.000       A91: 50     A101:907
##  A62:103      A72:172        1st Qu.:2.000       A92:310     A102: 41
##  A63: 63      A73:339        Median :3.000       A93:548     A103: 52
##  A64: 48      A74:174        Mean   :2.973       A94: 92
##  A65:183      A75:253        3rd Qu.:4.000
##                              Max.   :4.000
##
##   resid_since    property_type      age        other_plans housing
##  Min.   :1.000   A121:282      Min.   :19.00   A141:139    A151:179
##  1st Qu.:2.000   A122:232      1st Qu.:27.00   A142: 47    A152:713
##  Median :3.000   A123:332      Median :33.00   A143:814    A153:108
##  Mean   :2.845   A124:154      Mean   :35.55
##  3rd Qu.:4.000                 3rd Qu.:42.00
##  Max.   :4.000                 Max.   :75.00
##
##    no_credits      job        ppl_liable     phone     foreigner
##  Min.   :1.000   A171: 22   Min.   :1.000   A191:596   A201:963
##  1st Qu.:1.000   A172:200   1st Qu.:1.000   A192:404   A202: 37
##  Median :1.000   A173:630   Median :1.000
##  Mean   :1.407   A174:148   Mean   :1.155
##  3rd Qu.:2.000              3rd Qu.:1.000
##  Max.   :4.000              Max.   :2.000
##
##  creditor_type
##  Good:700
##  Bad :300
##
##
##
##
##
```

One thousand observations of 21 variables, being a mix of categorical and numerical data. Some extremes values seem to appear (differences between 3rd quantile and max) for some variables. Also, there are heavy imbalances in most of the categorical data.

More specifically, the column named 'ppl_liable' (refers to the Number of people being liable to provide maintenance') has a minimum of one and a maximum of two, and is an integer (understandable, since it refers to a number of people). Furthermore, 'no_credits' (Number of existing credits at this bank) is between one and four and also an integer, as is instal_income_ratio. All these variables can be treated as numeric (continuous) variables, but I believe that treating them as factors might be better in this case. Also, from the values of 'resid_since', it looks like this could also be treated as a factor. Let's get their unique values:

```
# See unique values
print(paste("ppl_liable unique values:", length(unique(credit$ppl_liable))))
```

```
## [1] "ppl_liable unique values: 2"
```

```
print(paste("no_credits unique values", length(unique(credit$no_credits))))
```

```
## [1] "no_credits unique values 4"
```

```
print(paste("instal_income_ratio unique values", length(unique(credit$instal_income_ratio))))
```

```
## [1] "instal_income_ratio unique values 4"
```

```
print(paste("resid_since unique values", length(unique(credit$resid_since))))
```

```
## [1] "resid_since unique values 4"
```

Indeed, two or four different variables. Let's treat these values as factors:

```
# Convert some numeric variables to factors
credit$ppl_liable <- as.factor(credit$ppl_liable)
credit$no_credits <- as.factor(credit$no_credits)
credit$instal_income_ratio <- as.factor(credit$instal_income_ratio)
credit$resid_since <- as.factor(credit$resid_since)
```

A bit of cross-tabulation and independence tests for some varaibles:

```
with(credit,CrossTable(creditor_type, savings_accs, digits=1, prop.r=F, prop.t=F, prop.chisq=F, chisq=T
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1000
##
##
##              | savings_accs
## creditor_type |       A61 |       A62 |       A63 |       A64 |       A65 | Row Total |
## --------------|-----------|-----------|-----------|-----------|-----------|-----------|
##          Good |       386 |        69 |        52 |        42 |       151 |       700 |
##               |       0.6 |       0.7 |       0.8 |       0.9 |       0.8 |           |
## --------------|-----------|-----------|-----------|-----------|-----------|-----------|
##           Bad |       217 |        34 |        11 |         6 |        32 |       300 |
##               |       0.4 |       0.3 |       0.2 |       0.1 |       0.2 |           |
```

```
## --------------|-----------|-----------|-----------|-----------|-----------|-----------|
##  Column Total |       603 |       103 |        63 |        48 |       183 |      1000 |
##               |       0.6 |       0.1 |       0.1 |       0.0 |       0.2 |           |
## --------------|-----------|-----------|-----------|-----------|-----------|-----------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  36.09893     d.f. =  4     p =  2.761214e-07
##
##
##
```

```
with(credit,CrossTable(creditor_type, pers_status, digits=1, prop.r=F, prop.t=F, prop.chisq=F, chisq=T))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1000
##
##
##               | pers_status
## creditor_type |       A91 |       A92 |       A93 |       A94 | Row Total |
## --------------|-----------|-----------|-----------|-----------|-----------|
##          Good |        30 |       201 |       402 |        67 |       700 |
##               |       0.6 |       0.6 |       0.7 |       0.7 |           |
## --------------|-----------|-----------|-----------|-----------|-----------|
##           Bad |        20 |       109 |       146 |        25 |       300 |
##               |       0.4 |       0.4 |       0.3 |       0.3 |           |
## --------------|-----------|-----------|-----------|-----------|-----------|
##  Column Total |        50 |       310 |       548 |        92 |      1000 |
##               |       0.0 |       0.3 |       0.5 |       0.1 |           |
## --------------|-----------|-----------|-----------|-----------|-----------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  9.605214     d.f. =  3     p =  0.02223801
##
##
##
```

```r
with(credit,CrossTable(creditor_type, other_debtors, digits=1, prop.r=F, prop.t=F, prop.chisq=F, chisq=
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1000
##
##
##               | other_debtors
## creditor_type |      A101 |      A102 |      A103 | Row Total |
## --------------|-----------|-----------|-----------|-----------|
##          Good |       635 |        23 |        42 |       700 |
##               |       0.7 |       0.6 |       0.8 |           |
## --------------|-----------|-----------|-----------|-----------|
##           Bad |       272 |        18 |        10 |       300 |
##               |       0.3 |       0.4 |       0.2 |           |
## --------------|-----------|-----------|-----------|-----------|
##  Column Total |       907 |        41 |        52 |      1000 |
##               |       0.9 |       0.0 |       0.1 |           |
## --------------|-----------|-----------|-----------|-----------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  6.645367     d.f. =  2     p =  0.03605595
##
##
##
```

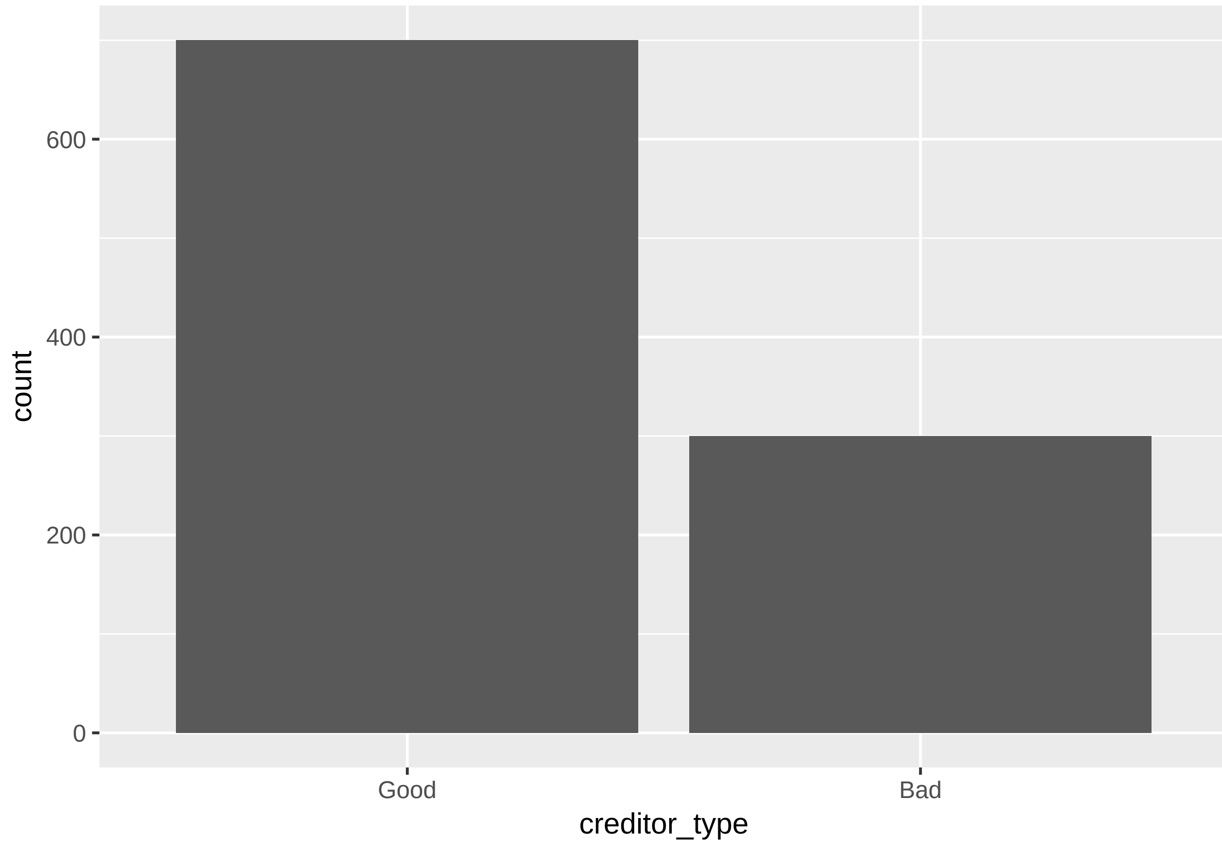
There seems to be a dependence of savings and personal status on the credit rating. Also, the number of dependents does not seem to have any bearing on the credit rating. Perhaps its fair to say that people who are intent on having a good credit rating continue to maintain their status irrespective of the number of dependents (other debtors).

Let's visualize:

Exploratory Analysis

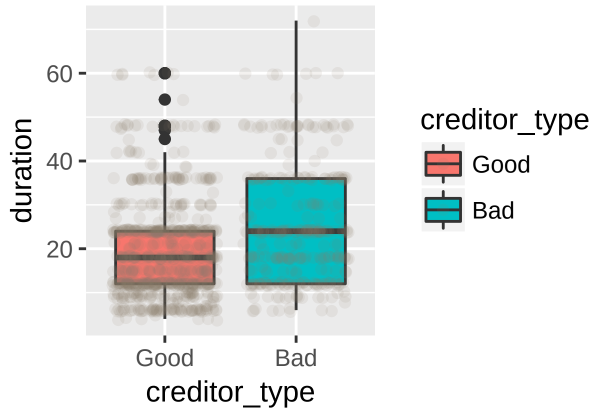
At first let's have a look at the proportions of the target variable:

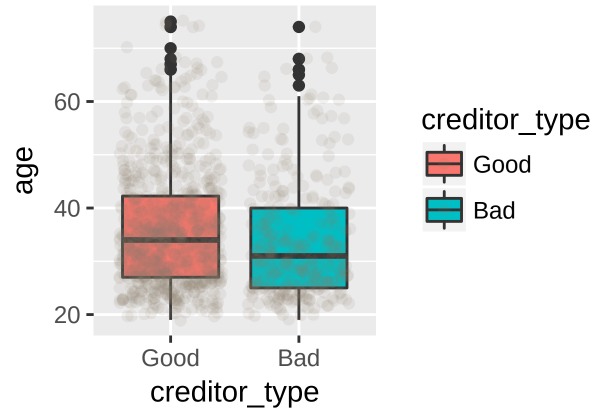
```r
g <- ggplot(credit, aes(creditor_type))
# Number of cars in each class:
g + geom_bar()
```

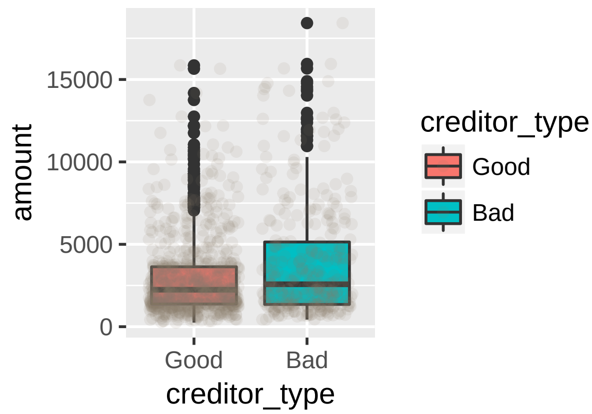


About two thirds of the sample are goog creditors, and one third is bad creditors. Let's make some box-plots for the numerical variables:

```
# Boxplots

p1 <- ggplot(credit, aes(x = creditor_type, y = duration, fill = creditor_type)) +
geom_boxplot(alpha = 1) +
geom_jitter(alpha = 0.1, color = "bisque4")

p2 <- ggplot(credit, aes(x = creditor_type, y = amount, fill = creditor_type)) +
geom_boxplot(alpha = 1) +
geom_jitter(alpha = 0.1, color = "bisque4")

p3 <- ggplot(credit, aes(x = creditor_type, y = age, fill = creditor_type)) +
geom_boxplot(alpha = 1) +
geom_jitter(alpha = 0.1, color = "bisque4")

# Plot them in one plot
multiplot(p1, p2, p3, cols = 2)
```







No significant differences between the distributions for good or bad creditors. Most of the credit amounts are less than 5000, with some higher credit amounts. The largest amount disbursed is as high as 18000. The middle 50% of the population seems to lie between 1300 to 3900, and as the credit amount increases, more and more creditors are defaulting, with amounts higher than 15000 to be mostly defaulting. Age does not seem to play a vital role in default rate, whereas duration for good creditors is on average significantly lower.

Let's have a look at the categorical variables:

```
# Plot the categorical variables only
numerical <- c(2, 5, 13)

df1 <- credit[which(credit$creditor_type == 'Good'),-numerical]
df2 <- credit[which(credit$creditor_type != 'Good'),-numerical]

df1$creditor_type <- NULL
df2$creditor_type <- NULL

df1 <- gather(df1)
```

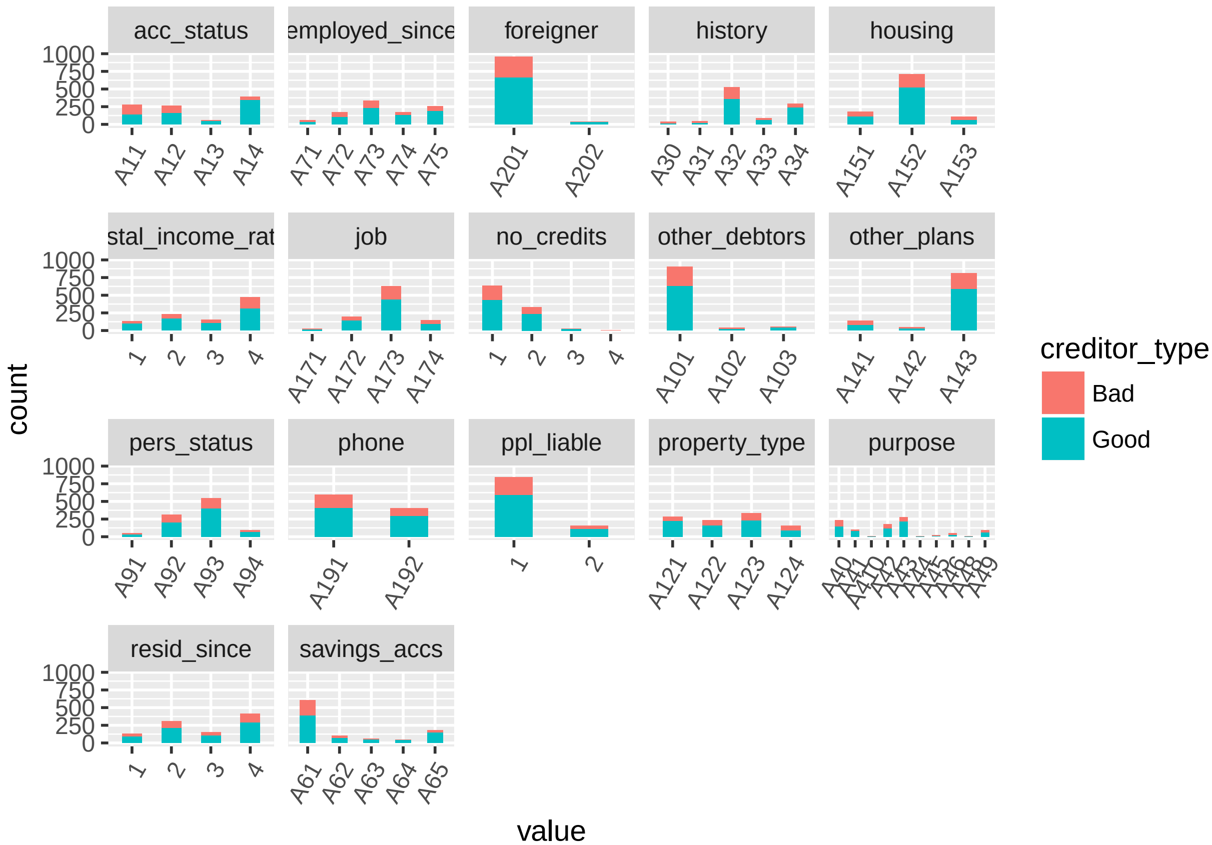
```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
df2 <- gather(df2)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
df1$creditor_type <- 'Good'
df2$creditor_type <- 'Bad'
```

```r
df <- rbind(df1, df2)

ggplot(df, aes(x = value)) +
  stat_count(width = 0.5, aes(fill = creditor_type)) +
  facet_wrap(~key, scales = 'free_x') +
  theme(axis.text.x = element_text(angle = 60, hjust = 1))
```



The following observations can be made:

- Job: A-173 (skilled employees / officials) present a high number of defaults
- other_debtors: Very few observations in categories A102, A103
- housing: A-152 (Own house) has a high chance of defaulting
- acc_status: A-14 presents the biggest spread
- history: A-32, A-34 acount for most bad creditors
- property_type: A-121, A-123 have a high number of bad creditors
- purpose: A-40, A-42 show the highest number of good creditors
- savings_accs: A61 has a high rate of defaulting *instal_income_ratio: 4% of disposable income has a high number of defaults

Modeling

At first let's split the data into a train and a test set (70 / 30):

```r
#Reproducibility
set.seed(1)
```

```r
# Split data
inTraining <- createDataPartition(credit$creditor_type, p = .7, list = FALSE)
train <- credit[ inTraining,]
test  <- credit[-inTraining,]
```

To build the model, we'll perform stepise logistic regression (although it is considered a naive method by the statistics community). We'll start with a full model that includes all predictors, and we'll start removing them one by one, using the Bayesian Information Criterion (BIC), which, as we know, tends to penalize the use of an increased number of predictors more than the AIC:

```r
model <- step(glm(creditor_type ~ ., data=train, family = binomial(link="logit")),
     direction = 'backward', k = log(nrow(train)))
```

```
## Start:  AIC=951.87
## creditor_type ~ acc_status + duration + history + purpose + amount +
##     savings_accs + employed_since + instal_income_ratio + pers_status +
##     other_debtors + resid_since + property_type + age + other_plans +
##     housing + no_credits + job + ppl_liable + phone + foreigner
##
##                       Df Deviance    AIC
## - purpose              9   616.51 917.86
## - employed_since       4   596.88 930.99
## - property_type        3   594.28 934.94
## - job                  3   595.69 936.35
## - resid_since          3   596.12 936.78
## - no_credits           3   597.32 937.98
## - instal_income_ratio  3   598.70 939.36
## - housing              2   592.62 939.83
## - other_plans          2   593.03 940.24
## - other_debtors        2   595.28 942.49
## - pers_status          3   603.88 944.54
## - savings_accs         4   610.51 944.61
## - age                  1   591.58 945.33
## - ppl_liable           1   592.81 946.57
## - phone                1   593.84 947.60
## - amount               1   594.23 947.99
## - foreigner            1   596.62 950.38
## <none>                     591.56 951.87
## - history              4   618.04 952.14
## - duration             1   600.28 954.04
## - acc_status           3   654.40 995.05
##
## Step:  AIC=917.86
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     employed_since + instal_income_ratio + pers_status + other_debtors +
##     resid_since + property_type + age + other_plans + housing +
##     no_credits + job + ppl_liable + phone + foreigner
##
##                       Df Deviance    AIC
## - employed_since       4   621.65 896.79
## - job                  3   619.38 901.08
## - property_type        3   619.69 901.38
## - no_credits           3   622.26 903.96
## - resid_since          3   623.15 904.85
```

```
## - housing             2   617.42 905.66
## - other_plans         2   617.53 905.78
## - instal_income_ratio  3   624.28 905.98
## - other_debtors       2   621.21 909.45
## - pers_status         3   628.18 909.88
## - age                 1   616.61 911.41
## - savings_accs        4   636.73 911.88
## - ppl_liable          1   617.98 912.77
## - phone               1   618.50 913.29
## - amount              1   619.38 914.18
## - foreigner           1   620.27 915.06
## - duration            1   621.84 916.63
## - history             4   641.50 916.65
## <none>                    616.51 917.86
## - acc_status          3   683.77 965.47
##
## Step:  AIC=896.79
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     instal_income_ratio + pers_status + other_debtors + resid_since +
##     property_type + age + other_plans + housing + no_credits +
##     job + ppl_liable + phone + foreigner
##
##                      Df Deviance    AIC
## - job                 3   624.63 880.12
## - property_type       3   624.88 880.37
## - no_credits          3   626.58 882.07
## - resid_since         3   628.84 884.33
## - housing             2   622.85 884.89
## - other_plans         2   622.93 884.97
## - instal_income_ratio  3   629.62 885.11
## - other_debtors       2   626.26 888.30
## - age                 1   621.66 890.25
## - ppl_liable          1   622.96 891.55
## - savings_accs        4   642.96 891.90
## - pers_status         3   636.66 892.15
## - phone               1   623.85 892.45
## - amount              1   624.71 893.31
## - foreigner           1   625.27 893.86
## - duration            1   626.63 895.23
## - history             4   647.21 896.15
## <none>                    621.65 896.79
## - acc_status          3   689.39 944.89
##
## Step:  AIC=880.12
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     instal_income_ratio + pers_status + other_debtors + resid_since +
##     property_type + age + other_plans + housing + no_credits +
##     ppl_liable + phone + foreigner
##
##                      Df Deviance    AIC
## - property_type       3   627.95 863.79
## - no_credits          3   629.21 865.05
## - resid_since         3   631.20 867.04
## - housing             2   625.49 867.88
```

```
## - other_plans          2   626.23 868.62
## - instal_income_ratio  3   633.21 869.05
## - other_debtors        2   628.94 871.33
## - age                  1   624.71 873.65
## - pers_status          3   638.80 874.64
## - ppl_liable           1   625.85 874.79
## - savings_accs         4   646.45 875.74
## - phone                1   626.88 875.82
## - amount               1   627.98 876.92
## - foreigner            1   628.29 877.23
## - history              4   648.82 878.11
## - duration             1   629.88 878.83
## <none>                     624.63 880.12
## - acc_status           3   690.71 926.55
##
## Step:  AIC=863.79
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     instal_income_ratio + pers_status + other_debtors + resid_since +
##     age + other_plans + housing + no_credits + ppl_liable + phone +
##     foreigner
##
##                       Df Deviance    AIC
## - no_credits           3   632.30 848.48
## - resid_since          3   634.47 850.65
## - housing              2   628.91 851.64
## - other_plans          2   629.84 852.58
## - instal_income_ratio  3   637.29 853.48
## - other_debtors        2   632.75 855.48
## - age                  1   628.07 857.35
## - pers_status          3   641.65 857.83
## - ppl_liable           1   629.13 858.42
## - savings_accs         4   648.93 858.57
## - phone                1   629.66 858.94
## - foreigner            1   631.26 860.54
## - amount               1   631.50 860.79
## - history              4   651.66 861.30
## - duration             1   633.74 863.03
## <none>                     627.95 863.79
## - acc_status           3   698.37 914.55
##
## Step:  AIC=848.48
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     instal_income_ratio + pers_status + other_debtors + resid_since +
##     age + other_plans + housing + ppl_liable + phone + foreigner
##
##                       Df Deviance    AIC
## - resid_since          3   638.48 835.02
## - housing              2   633.12 836.21
## - other_plans          2   634.46 837.54
## - instal_income_ratio  3   641.65 838.18
## - other_debtors        2   636.60 839.69
## - pers_status          3   645.42 841.95
## - age                  1   632.38 842.02
## - history              4   652.15 842.13
```

```
## - phone              1   633.74 843.38
## - savings_accs       4   653.41 843.40
## - ppl_liable         1   633.86 843.49
## - foreigner          1   635.85 845.48
## - amount             1   636.03 845.66
## - duration           1   637.80 847.44
## <none>                   632.30 848.48
## - acc_status         3   700.42 896.96
##
## Step:  AIC=835.02
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     instal_income_ratio + pers_status + other_debtors + age +
##     other_plans + housing + ppl_liable + phone + foreigner
##
##                       Df Deviance    AIC
## - housing              2   638.78 822.21
## - other_plans          2   640.45 823.88
## - instal_income_ratio  3   648.74 825.62
## - other_debtors        2   642.44 825.87
## - history              4   657.47 827.80
## - pers_status          3   650.94 827.82
## - age                  1   639.00 828.98
## - savings_accs         4   659.05 829.37
## - ppl_liable           1   640.14 830.12
## - phone                1   640.19 830.17
## - foreigner            1   641.78 831.76
## - amount               1   643.37 833.35
## - duration             1   643.44 833.42
## <none>                     638.48 835.02
## - acc_status           3   703.25 880.13
##
## Step:  AIC=822.21
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     instal_income_ratio + pers_status + other_debtors + age +
##     other_plans + ppl_liable + phone + foreigner
##
##                       Df Deviance    AIC
## - other_plans          2   640.64 810.97
## - instal_income_ratio  3   648.96 812.73
## - other_debtors        2   642.73 813.05
## - history              4   658.17 815.40
## - pers_status          3   652.33 816.10
## - age                  1   639.46 816.34
## - savings_accs         4   659.24 816.47
## - phone                1   640.46 817.34
## - ppl_liable           1   640.47 817.35
## - foreigner            1   642.05 818.93
## - duration             1   643.65 820.53
## - amount               1   643.78 820.66
## <none>                     638.78 822.21
## - acc_status           3   704.81 868.58
##
## Step:  AIC=810.97
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
```

```
##     instal_income_ratio + pers_status + other_debtors + age +
##     ppl_liable + phone + foreigner
##
##                        Df Deviance    AIC
## - instal_income_ratio  3   650.67 801.35
## - other_debtors        2   644.56 801.79
## - pers_status          3   653.81 804.49
## - age                  1   641.19 804.97
## - savings_accs         4   661.12 805.25
## - phone                1   642.24 806.01
## - ppl_liable           1   642.50 806.28
## - foreigner            1   643.73 807.51
## - history              4   664.89 809.01
## - amount               1   645.50 809.27
## - duration             1   645.60 809.38
## <none>                     640.64 810.97
## - acc_status           3   707.27 857.95
##
## Step:  AIC=801.35
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     pers_status + other_debtors + age + ppl_liable + phone +
##     foreigner
##
##                  Df Deviance    AIC
## - pers_status    3   660.19 791.21
## - other_debtors  2   654.90 792.47
## - savings_accs   4   670.38 794.85
## - age            1   650.96 795.09
## - ppl_liable     1   651.81 795.94
## - amount         1   651.96 796.08
## - phone          1   652.16 796.28
## - foreigner      1   653.98 798.11
## - history        4   674.09 798.56
## <none>               650.67 801.35
## - duration       1   661.34 805.46
## - acc_status     3   715.93 846.95
##
## Step:  AIC=791.21
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     other_debtors + age + ppl_liable + phone + foreigner
##
##                  Df Deviance    AIC
## - other_debtors  2   664.96 782.88
## - ppl_liable     1   660.25 784.72
## - savings_accs   4   680.01 784.82
## - amount         1   661.04 785.52
## - age            1   661.38 785.85
## - phone          1   661.49 785.96
## - foreigner      1   663.25 787.72
## - history        4   684.30 789.12
## <none>               660.19 791.21
## - duration       1   670.12 794.59
## - acc_status     3   724.40 835.77
##
```

```
## Step:  AIC=782.88
## creditor_type ~ acc_status + duration + history + amount + savings_accs +
##     age + ppl_liable + phone + foreigner
##
##                 Df Deviance    AIC
## - savings_accs  4   682.82 774.54
## - ppl_liable    1   665.02 776.39
## - amount        1   665.96 777.33
## - phone         1   666.06 777.43
## - age           1   666.11 777.48
## - history       4   688.78 780.49
## - foreigner     1   669.52 780.89
## <none>              664.96 782.88
## - duration      1   674.30 785.67
## - acc_status    3   726.81 825.08
##
## Step:  AIC=774.54
## creditor_type ~ acc_status + duration + history + amount + age +
##     ppl_liable + phone + foreigner
##
##               Df Deviance    AIC
## - ppl_liable  1   682.87 768.03
## - amount      1   683.58 768.74
## - phone       1   684.39 769.55
## - age         1   684.51 769.67
## - history     4   705.88 771.39
## - foreigner   1   687.87 773.03
## <none>            682.82 774.54
## - duration    1   691.78 776.94
## - acc_status  3   764.72 836.79
##
## Step:  AIC=768.03
## creditor_type ~ acc_status + duration + history + amount + age +
##     phone + foreigner
##
##               Df Deviance    AIC
## - amount      1   683.64 762.25
## - phone       1   684.44 763.06
## - age         1   684.51 763.12
## - history     4   705.98 764.94
## - foreigner   1   687.90 766.52
## <none>            682.87 768.03
## - duration    1   691.78 770.39
## - acc_status  3   764.75 830.26
##
## Step:  AIC=762.25
## creditor_type ~ acc_status + duration + history + age + phone +
##     foreigner
##
##               Df Deviance    AIC
## - phone       1   684.74 756.80
## - age         1   685.23 757.29
## - history     4   707.42 759.83
## - foreigner   1   688.30 760.36
```

```
## <none>            683.64 762.25
## - duration    1   704.53 776.59
## - acc_status  3   765.29 824.25
##
## Step:  AIC=756.8
## creditor_type ~ acc_status + duration + history + age + foreigner
##
##              Df Deviance    AIC
## - age         1   686.85 752.36
## - history     4   708.47 754.32
## - foreigner   1   689.21 754.72
## <none>            684.74 756.80
## - duration    1   704.95 770.46
## - acc_status  3   768.78 821.19
##
## Step:  AIC=752.36
## creditor_type ~ acc_status + duration + history + foreigner
##
##              Df Deviance    AIC
## - foreigner   1   691.05 750.01
## - history     4   711.34 750.64
## <none>            686.85 752.36
## - duration    1   707.69 766.65
## - acc_status  3   772.77 818.63
##
## Step:  AIC=750.01
## creditor_type ~ acc_status + duration + history
##
##              Df Deviance    AIC
## - history     4   715.42 748.18
## <none>            691.05 750.01
## - duration    1   713.27 765.68
## - acc_status  3   776.33 815.64
##
## Step:  AIC=748.18
## creditor_type ~ acc_status + duration
##
##              Df Deviance    AIC
## <none>            715.42 748.18
## - duration    1   744.18 770.39
## - acc_status  3   819.88 832.98
```

```r
# k=log(n), n:# of observations >- BIC
# k = 2 gives the original AIC
```

We can see that, according to BIC the predictors to include are only the dummy variables for acc_status and duration. Let's have a look at the models summary:

```r
summary(model)
```

```
##
## Call:
## glm(formula = creditor_type ~ acc_status + duration, family = binomial(link = "logit"),
##     data = train)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6723  -0.8769  -0.4170   0.9641   2.3713
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.879150   0.212638  -4.134 3.56e-05 ***
## acc_statusA12 -0.346512   0.213554  -1.623   0.1047
## acc_statusA13 -1.051245   0.392515  -2.678   0.0074 **
## acc_statusA14 -2.221363   0.255226  -8.704  < 2e-16 ***
## duration       0.039004   0.007467   5.223 1.76e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 855.21  on 699  degrees of freedom
## Residual deviance: 715.42  on 695  degrees of freedom
## AIC: 725.42
##
## Number of Fisher Scoring iterations: 5
```

The dummy variable indicating if account status is equal to A12 is somehat redundant. This makes sense, as for this value, there is a 50-50 chance that a creditor will default as shown in one of the plots in the exploratory data analysis.

Now that we have our model, let's get its performance on the test set through a confusion matrix, *by assuming a 0.5 probability threshold*:

```r
#Predict
y_hat <- predict(model, newdata = test)

# Get the probability from the logit function
y_hat <- exp(y_hat) / (exp(y_hat) + 1)

# Apply the threshold
y_hat <- ifelse(y_hat > 0.5, 1, 0)

# Covert the factor from the test set to binary
y <- ifelse(test$creditor_type == 'Good', 0, 1)

# Create Confusion Matrix
confusionMatrix(data=as.factor(y_hat),
                reference=as.factor(y),
                positive='1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 190  60
##          1  20  30
##
##                Accuracy : 0.7333
##                  95% CI : (0.6795, 0.7825)
##     No Information Rate : 0.7
```

```
##     P-Value [Acc > NIR] : 0.1149
##
##                   Kappa : 0.2727
##  Mcnemar's Test P-Value : 1.299e-05
##
##             Sensitivity : 0.3333
##             Specificity : 0.9048
##          Pos Pred Value : 0.6000
##          Neg Pred Value : 0.7600
##              Prevalence : 0.3000
##          Detection Rate : 0.1000
##    Detection Prevalence : 0.1667
##       Balanced Accuracy : 0.6190
##
##        'Positive' Class : 1
##
```

This is a very nice result, as results on the same dataset [elsewhere] (http://bayesian-intelligence.com/publications/TR2010_1_zonneveldt__korb__nicholson__bn__class__credit__data.pdf), report slightly higher results, but with more complex models.

Now, we have to change the cut-off probability: if we increase the threshold value opportunity cost (a good customer rejected by our model) goes up but default risk (when a bad customer is given a credit facility and person defaults) goes down. When we reduce the cut-off probability value (probability of finding a good customer), default risk increases. The last step is to balance these two costs. We know that the bank estimates that incorrectly identifying a bad customer as good is 5 times worse than incorrectly classifying a good customer as bad.

This means that every false positive is worth 5 false negative predictions. As such, a threshfold value for which $FP/FN \approx 5$. Let's try different threshold values and observe the confusion matrix:

```r
#Predict
y_hat <- predict(model, newdata = test)

# Get the probability from the logit function
y_hat <- exp(y_hat) / (exp(y_hat) + 1)

# Apply the threshold
y_hat <- ifelse(y_hat >= 0.22, 1, 0)

# Create Confusion Matrix
confusionMatrix(data=as.factor(y_hat),
                reference=as.factor(y),
                positive='1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 106  19
##          1 104  71
##
##                Accuracy : 0.59
##                  95% CI : (0.532, 0.6462)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 1
```

```
##
##                   Kappa : 0.2312
##  Mcnemar's Test P-Value : 3.618e-14
##
##             Sensitivity : 0.7889
##             Specificity : 0.5048
##          Pos Pred Value : 0.4057
##          Neg Pred Value : 0.8480
##              Prevalence : 0.3000
##          Detection Rate : 0.2367
##    Detection Prevalence : 0.5833
##       Balanced Accuracy : 0.6468
##
##        'Positive' Class : 1
##
```

Conclusions

We've applied a steprwise logistic regression using the Bayesian Information Criterion (BIC), which performs just as good as [more complex models on the same dataset] (http://bayesian-intelligence.com/publications/TR2010_1_zonneveldt_korb_nicholson_bn_class_credit_data.pdf). To balance the opportunity cost for the company, we should apply a threshold value of 0.22.