Regression Tree and Risk Assessment Models

Alena Fedash

2022-10-06

Contents

Please find below my step-by-step solution in R with explanations and comments for each step.

Part 1.A Regression Trees

Step 0: Load the libraries

```
library(dplyr)
library(tidyverse)
library(dslabs)
library(data.table)
library(ggplot2)
library(plotly)
library(outliers)
library(qcc)
library(mctest)
library(ppcor)
library(car)
library(psych)
library(ggthemes)
library(corrplot)
library(DAAG)
library(GGally)
library(caret)
library(psych)
library(ggpubr)
library(tree)
library(randomForest)
library(vip)
library(rsample)
library(ROCR)
library(gridExtra)
```

Step 1: Load the dataset

```
data <- read.table("uscrime.txt",
                   header = TRUE,
                   stringsAsFactors = FALSE,
                   sep = "",
                   dec = ".")
head(data)
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

```r
test_data <- data.frame(M = 14.0,
                        So = 0,
                        Ed = 10.0,
                        Po1 = 12.0,
                        Po2 = 15.5,
                        LF = 0.640,
                        M.F = 94.0,
                        Pop = 150,
                        NW = 1.1,
                        U1 = 0.120,
                        U2 = 3.6,
                        Wealth = 3200,
                        Ineq = 20.1,
                        Prob = 0.040,
                        Time = 39.0)
test_data
```

```
##    M So Ed Po1  Po2   LF M.F Pop  NW   U1  U2 Wealth Ineq Prob Time
## 1 14  0 10  12 15.5 0.64  94 150 1.1 0.12 3.6   3200 20.1 0.04   39
```

Step 2: Basic Explorations

Plots and basic data description to refer to:

```r
describe.by(data)
```

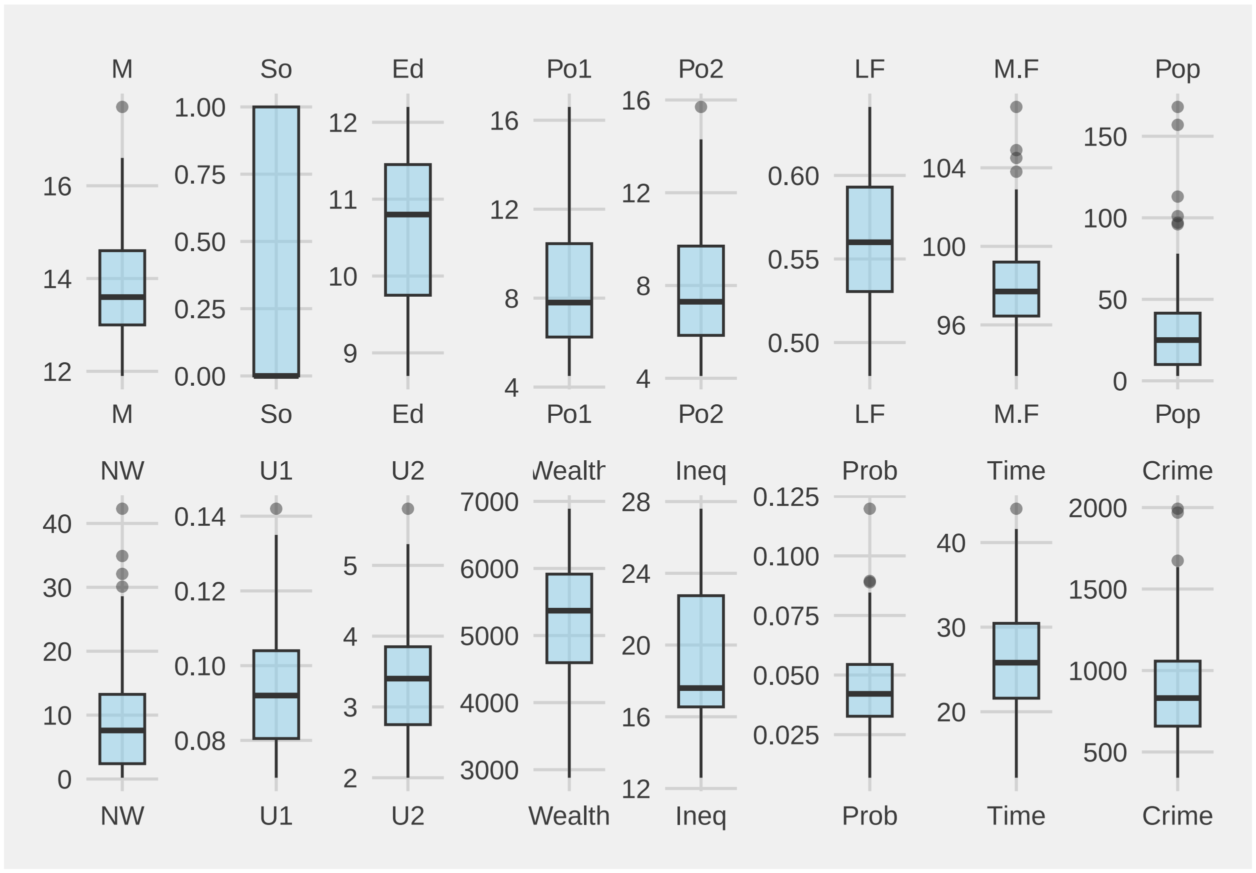
```
## Warning: describe.by is deprecated.  Please use the describeBy function

## Warning in describeBy(x = x, group = group, mat = mat, type = type, ...): no
## grouping variable requested

##        vars  n    mean     sd  median trimmed     mad     min     max   range
## M         1 47   13.86   1.26   13.60   13.75    1.19   11.90   17.70    5.80
## So        2 47    0.34   0.48    0.00    0.31    0.00    0.00    1.00    1.00
## Ed        3 47   10.56   1.12   10.80   10.59    1.19    8.70   12.20    3.50
## Po1       4 47    8.50   2.97    7.80    8.21    2.82    4.50   16.60   12.10
## Po2       5 47    8.02   2.80    7.30    7.76    2.82    4.10   15.70   11.60
## LF        6 47    0.56   0.04    0.56    0.56    0.05    0.48    0.64    0.16
## M.F       7 47   98.30   2.95   97.70   98.02    1.93   93.40  107.10   13.70
## Pop       8 47   36.62  38.07   25.00   29.95   22.24    3.00  168.00  165.00
## NW        9 47   10.11  10.28    7.60    8.56    7.71    0.20   42.30   42.10
## U1       10 47    0.10   0.02    0.09    0.09    0.02    0.07    0.14    0.07
## U2       11 47    3.40   0.84    3.40    3.35    0.89    2.00    5.80    3.80
## Wealth   12 47 5253.83 964.91 5370.00 5286.67 1111.95 2880.00 6890.00 4010.00
## Ineq     13 47   19.40   3.99   17.60   19.28    3.56   12.60   27.60   15.00
## Prob     14 47    0.05   0.02    0.04    0.05    0.02    0.01    0.12    0.11
## Time     15 47   26.60   7.09   25.80   26.35    6.37   12.20   44.00   31.80
## Crime    16 47  905.09 386.76  831.00  863.05  314.31  342.00 1993.00 1651.00
##         skew kurtosis    se
## M       0.82     0.38  0.18
## So      0.65    -1.61  0.07
## Ed     -0.32    -1.15  0.16
## Po1     0.89     0.16  0.43
## Po2     0.84     0.01  0.41
## LF      0.27    -0.89  0.01
## M.F     0.99     0.65  0.43
## Pop     1.85     3.08   5.55
## NW      1.38     1.08   1.50
## U1      0.77    -0.13   0.00
## U2      0.54     0.17   0.12
## Wealth -0.38    -0.61 140.75
## Ineq    0.37    -1.14   0.58
## Prob    0.88     0.75   0.00
## Time    0.37    -0.41   1.03
## Crime   1.05     0.78  56.42
```

```
#melt data for easier visualization
melted<-melt(data)
```

```
## Warning in melt(data): The melt generic in data.table has been passed a
## data.frame and will attempt to redirect to the relevant reshape2 method; please
## note that reshape2 is deprecated, and this redirection is now deprecated as
## well. To continue using melt methods from reshape2 while both libraries are
## attached, e.g. melt.list, you can prepend the namespace like
## reshape2::melt(data). In the next version, this warning will become an error.
```

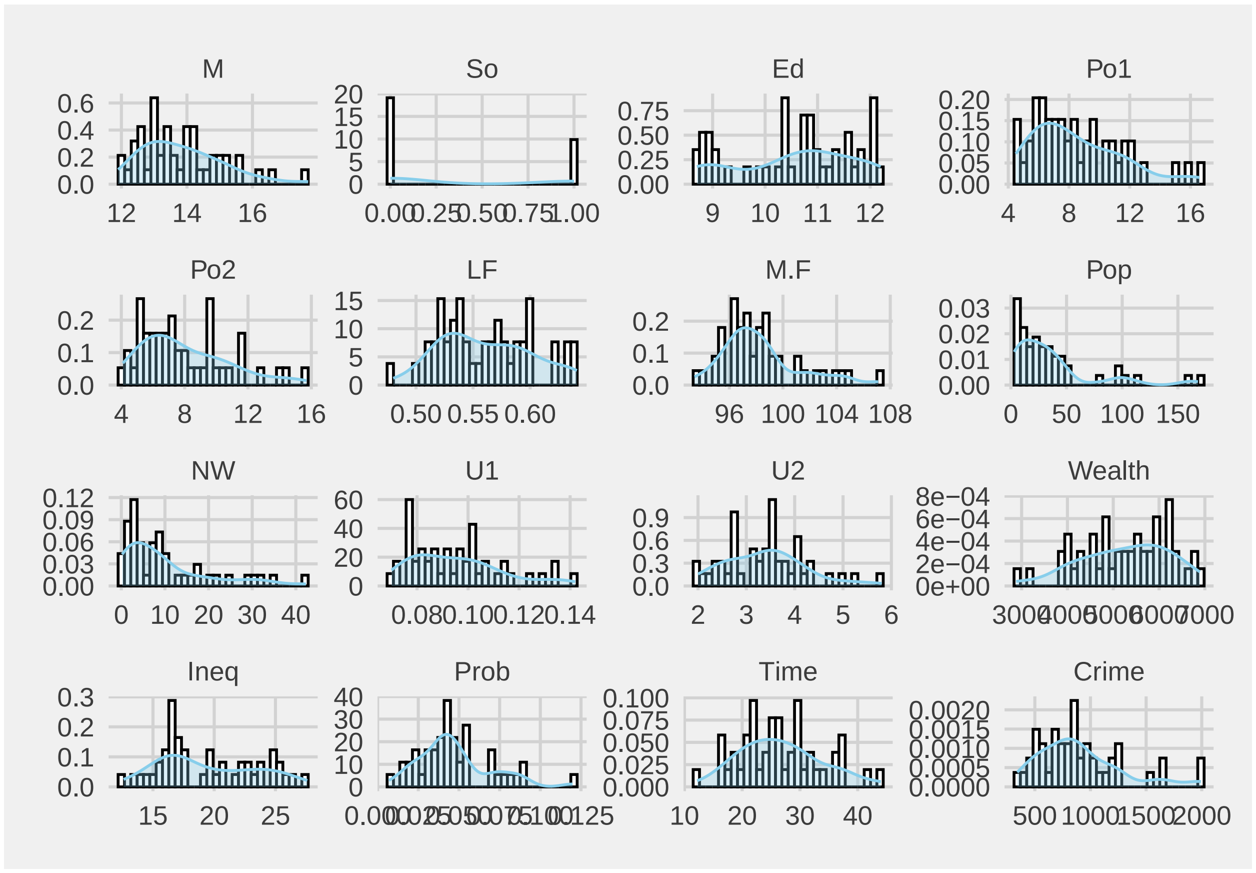
```
## No id variables; using all as measure variables
```

```
#boxplots
box_plots <- ggplot(melted,
                    aes(x=factor(variable), y=value))+
              geom_boxplot(alpha=.5, fill="skyblue")+
              facet_wrap(~variable, ncol=8, scale="free")+
              theme_fivethirtyeight()


box_plots
```

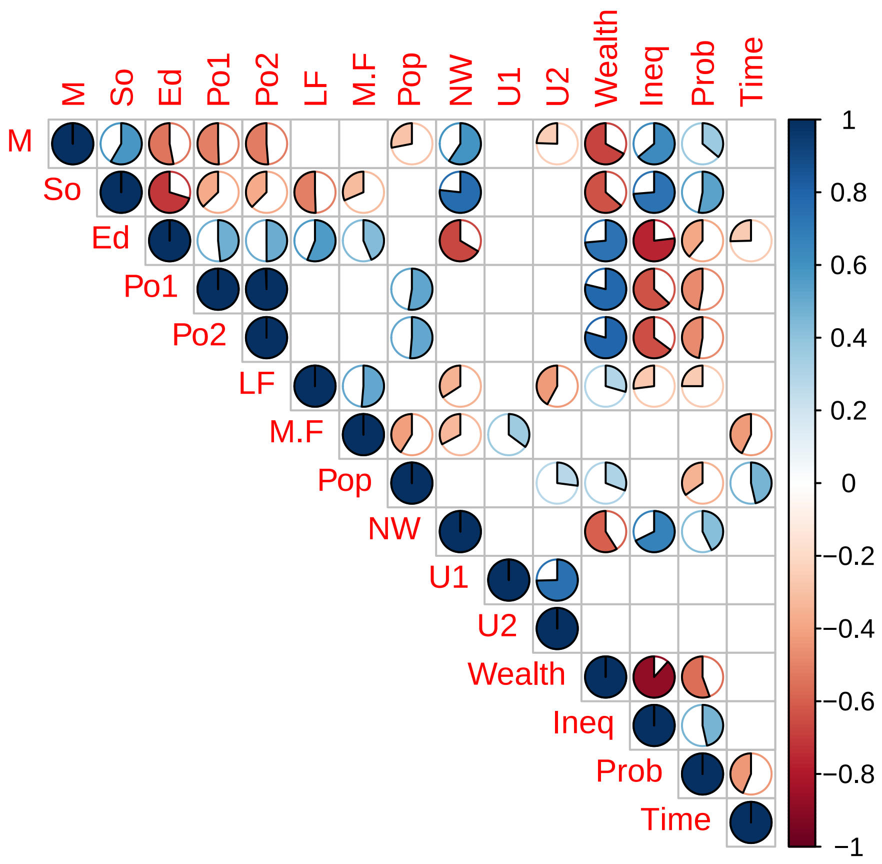


```r
#distribution plots with density
hist_plots <- ggplot(melted,
                aes(x=value))+
              geom_histogram(aes(y=..density..), colour="black", fill="white")+
              geom_density(alpha=.3, color="skyblue", fill="skyblue")+
              facet_wrap(~variable, scale="free")+
              theme_fivethirtyeight()
hist_plots
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
cor.mtest <- function(mat, ...) {
    mat <- as.matrix(data[,-16])
    n <- ncol(mat)
    p.mat<- matrix(NA, n, n)
    diag(p.mat) <- 0
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
            tmp <- cor.test(mat[, i], mat[, j], ...)
            p.mat[i, j] <- p.mat[j, i] <- tmp$p.value
        }
    }
  colnames(p.mat) <- rownames(p.mat) <- colnames(mat)
  p.mat
}
# matrix of the p-value of the correlation
p.mat <- cor.mtest(data)
corrplot(cor(data[,-16]),
         method='pie',
         type="upper",
         #order="hclust",
         p.mat = p.mat,
         sig.level = 0.1,
         insig = "blank")
```



Step 3: Fitting an unpruned regression tree to data

To start, we build a basic tree model, which we can then cross-validate and prune. I decided to grow two trees - one with default parameters of the tree() function, and another one with manually set control parameters. I will then compare both trees to check which one has better performance, so that we can improve it to find the best model.

Tree n.1 with default settings:

```r
set.seed(111)

tree1 <- tree(Crime~.,
              data=data)
summary(tree1)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = data)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100
```

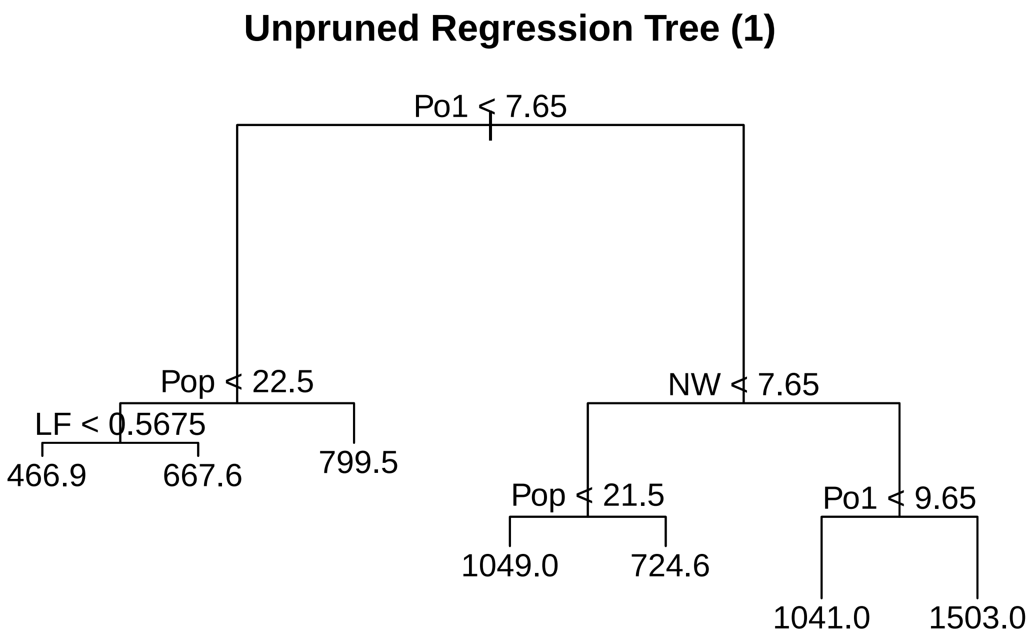
The first tree used only four predictors:

- Po1, expenditure on police protection in 1960
- Pop, population of state
- LF, labour force participation rate
- NW, number of non-whites per 1000 people

The tree has **7 terminal nodes** (leaves). **The residual mean deviance (how well the model can predict the response variable)** is 47390. **The lower this value, the better**, so we will compare it to the second model.

Let's visualize the first tree - we can see that Po1 is the major branching factor:

```
plot(tree1)
text(tree1)
title(main = "Unpruned Regression Tree (1)")
```



```
tree1$frame
```

```
##       var  n        dev      yval splits.cutleft splits.cutright
## 1     Po1 47 6880927.66  905.0851          <7.65           >7.65
## 2     Pop 23  779243.48  669.6087          <22.5           >22.5
## 4      LF 12  243811.00  550.5000        <0.5675         >0.5675
## 8  <leaf>  7   48518.86  466.8571
## 9  <leaf>  5   77757.20  667.6000
## 5  <leaf> 11  179470.73  799.5455
## 3      NW 24 3604162.50 1130.7500          <7.65           >7.65
## 6     Pop 10  557574.90  886.9000          <21.5           >21.5
## 12 <leaf>  5  146390.80 1049.2000
## 13 <leaf>  5  147771.20  724.6000
## 7     Po1 14 2027224.93 1304.9286          <9.65           >9.65
## 14 <leaf>  6  170828.00 1041.0000
## 15 <leaf>  8 1124984.88 1502.8750
```

Check the split information of the first tree:

We can see that **each of the terminal nodes has at least 10 data points**, which is a good ~20% share from the data.

This is what I would like to experiment with for the second tree. By the rule of thumb, **each leaf should have at least 5% of the data points**. Of course, with such a small data set, lowering this limit to 5% (~3 data points) could lead to overfitting. However, I would still like to see the difference this would make to the tree. I will set the minimal number of data points in each terminal node to 3 and set withing-node deviance to 1%, meaning that to split the root node, we need at least a 1% change in the model due to the split:

```r
set.seed(111)
tree_ctrl <- tree.control(nobs=nrow(data),
                          mincut=3,
                          mindev=0.01)
tree2 <- tree(Crime~.,
                   data=data,
                   control=tree_ctrl
                   )
summary(tree2)
```

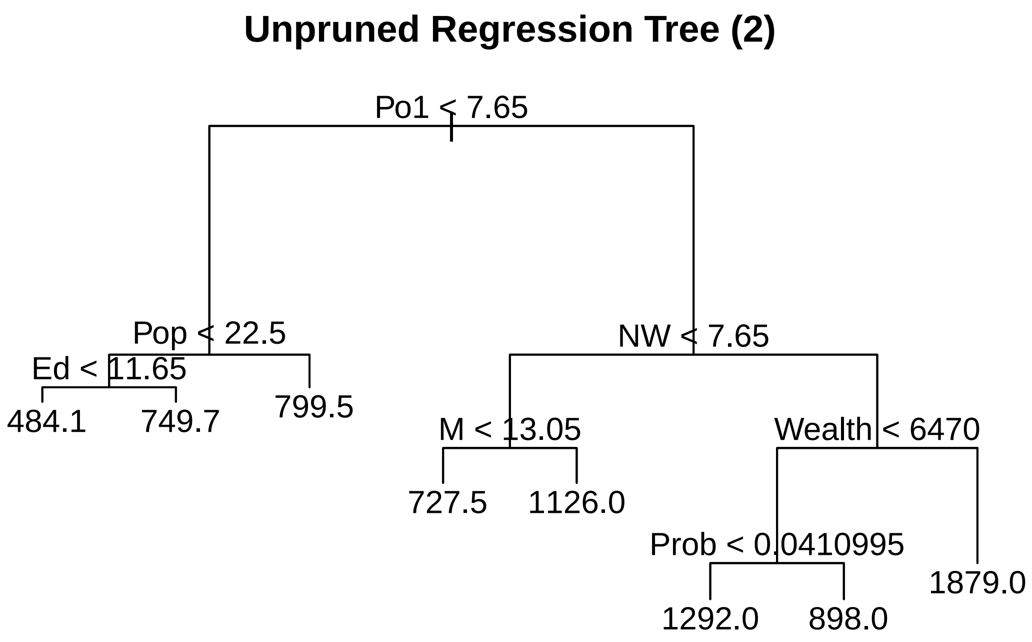
```
##
## Regression tree:
## tree(formula = Crime ~ ., data = data, control = tree_ctrl)
## Variables actually used in tree construction:
## [1] "Po1"    "Pop"    "Ed"     "NW"     "M"      "Wealth" "Prob"
## Number of terminal nodes:  8
## Residual mean deviance:  20950 = 817100 / 39
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -288.50  -80.62   22.50    0.00   73.44  343.40
```

This time, there are 8 terminal nodes and 7 predictors used:

- Po1, expenditure on police protection in 1960
- Pop, population of state
- Ed, mean years of schooling
- NW, number of non-whites per 1000 people
- M, percentage of males 14-24 y.o.
- Wealth, median of assets or household income
- Prob, probability of imprisonment

With manually set control parameters, the **residual mean deviance** has reduced to 20950, compared to 47390 with the first model, suggesting that this model predicts response values better/

```r
plot(tree2)
text(tree2)
title(main = "Unpruned Regression Tree (2)")
```



Compared to the previous tree, this one uses other factors on the third level following Pop and Wealth. Let's check how many data points there are for each node:

```
tree2$frame
```

```
##       var  n        dev      yval splits.cutleft splits.cutright
## 1     Po1 47 6880927.66  905.0851          <7.65           >7.65
## 2     Pop 23  779243.48  669.6087          <22.5           >22.5
## 4      Ed 12  243811.00  550.5000         <11.65          >11.65
## 8  <leaf>  9   58820.89  484.1111
## 9  <leaf>  3   26320.67  749.6667
## 5  <leaf> 11  179470.73  799.5455
## 3      NW 24 3604162.50 1130.7500          <7.65           >7.65
## 6       M 10  557574.90  886.9000         <13.05          >13.05
## 12 <leaf>  6  109289.50  727.5000
## 13 <leaf>  4   67160.00 1126.0000
## 7  Wealth 14 2027224.93 1304.9286          <6470           >6470
## 14   Prob 11  707252.73 1148.4545     <0.0410995      >0.0410995
## 28 <leaf>  7  284739.71 1291.5714
## 29 <leaf>  4   28226.00  898.0000
## 15 <leaf>  3   63120.67 1878.6667
```

Due to the manually set parameters, this tree has 3-4 points on some of the leaves. This could be a sign of overfitting - we would have to check whether this tree explains the data better due to it in the next steps.

Step 4: Cross-validation

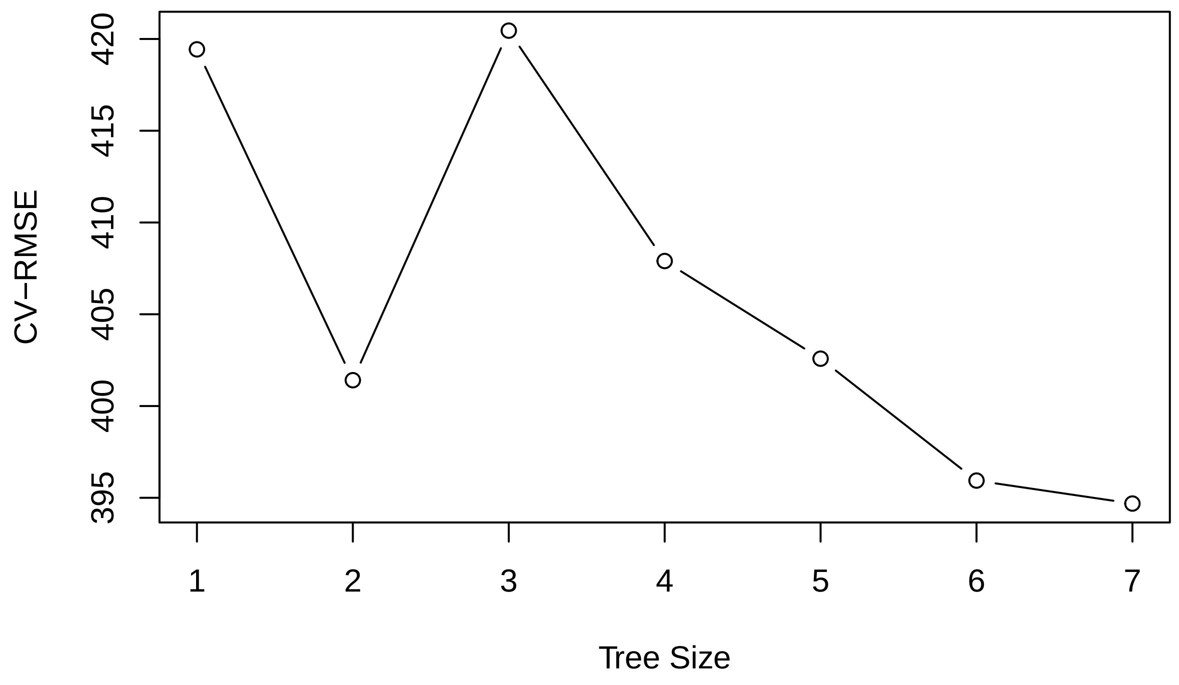
To understand whether we should prune the tree to improve the model and define which pruning would be good for our tree, we need to cross-validate the model:

```
set.seed(11)
tree1_cv <- cv.tree(tree1)
tree1_cv
```

```
## $size
## [1] 7 6 5 4 3 2 1
##
## $dev
## [1] 7321535 7368052 7617290 7819974 8308663 7572993 8268483
##
## $k
## [1]      -Inf  117534.9  263412.9  355961.8  731412.1 1019362.7 2497521.7
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

We can use the results above to plot RMSE (root mean square error) after cross-validation with tree size:

```
plot(tree1_cv$size, sqrt(tree1_cv$dev / nrow(data)), type = "b",
     xlab = "Tree Size", ylab = "CV-RMSE")
```



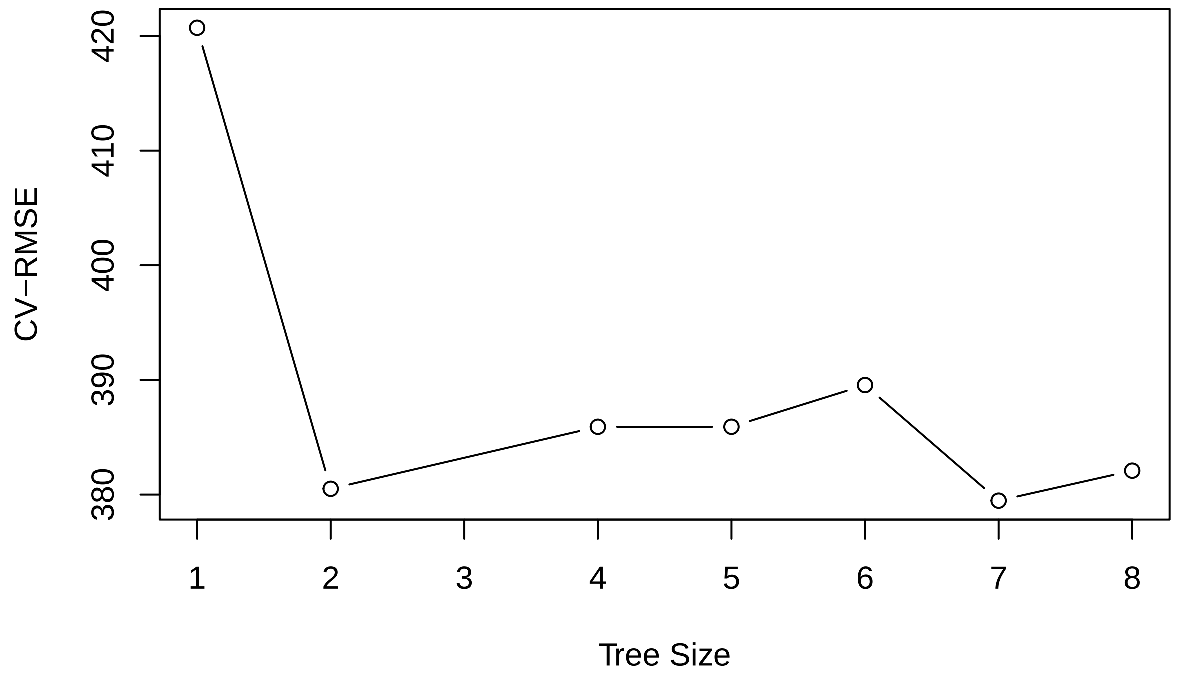
As we can see on the plot, the tree of size 7 seems to have the lowest RMSE, however the one with size 6 is not very different. We can **either leave the tree as it is, or prune it to size 6**. **The pruned tree would be smaller and easier to interpret**, so perhaps at such a small RMSE cost we should prune it.

Before making a decision about pruning, let's check CV results for the second tree:

```r
set.seed(111)
tree2_cv <- cv.tree(tree2)
tree2_cv
```

```
## $size
## [1] 8 7 6 5 4 2 1
##
## $dev
## [1] 6861535 6767877 7132280 6999703 6999703 6804821 8319169
##
## $k
## [1]      -Inf  158669.4  355961.8  381125.4  394287.0 1138107.1 2497521.7
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
plot(tree2_cv$size, sqrt(tree2_cv$dev / nrow(data)), type = "b",
     xlab = "Tree Size", ylab = "CV-RMSE")
```



As for the second tree, the lowest RMSE is achieved at tree size 7 - we need to prune it to this size.

Plan for the next steps:

- First, we prune both trees (Tree 1 to size 6, Tree 2 to size 7)
- Estimate model quality on training data and using CV, and compare the models

Then, if the results are bad, we should consider **building linear regression models on leaves' ends**. However, since our data set is so small, it would be useless to build linear regression on 5-10 data points of each leaf. So:

- Prune one of the trees to size 2 (split based on Po1 only) - both trees have relatively low RMSE for size two, so in such a case we might benefit from pruning to it.
- Build linear regressions for each leaf, cross-validate, estimate the quality
- If only few factors are significant - reduce models to those
- If such a problem persists - use PCA, then cross-validate and estimate quality

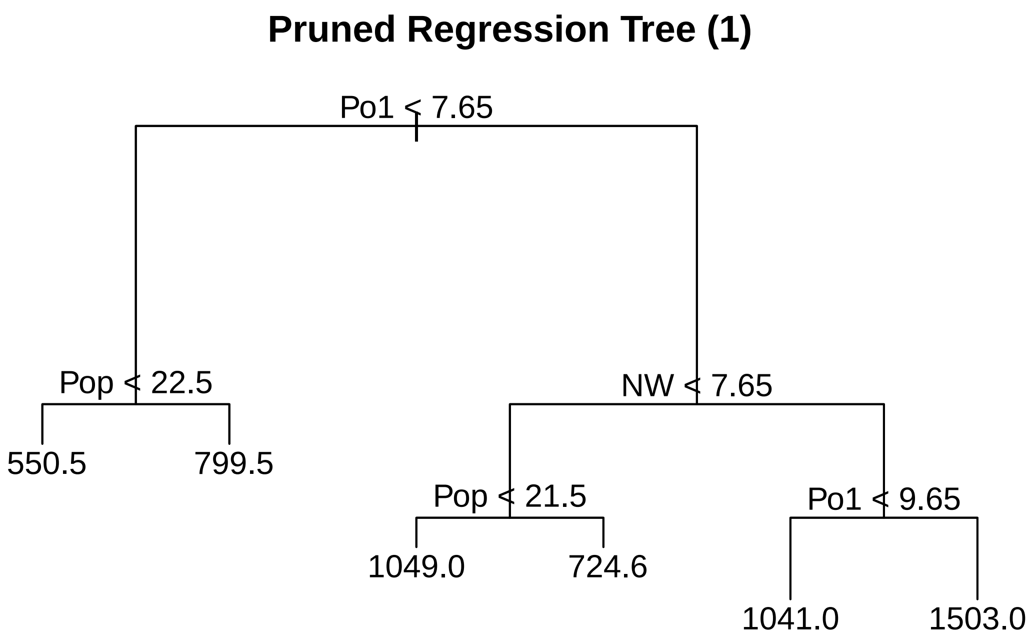
Step 5: Pruning

Lets's prune the trees:

```
#prune
tree1_pruned <- prune.tree(tree1, best=6)
summary(tree1_pruned)
```

```
##
## Regression tree:
## snip.tree(tree = tree1, nodes = 4L)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "NW"
## Number of terminal nodes:  6
## Residual mean deviance:  49100 = 2013000 / 41
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -99.520   -1.545    0.000  122.800  490.100
```

```
#plot
plot(tree1_pruned)
text(tree1_pruned)
title(main = "Pruned Regression Tree (1)")
```



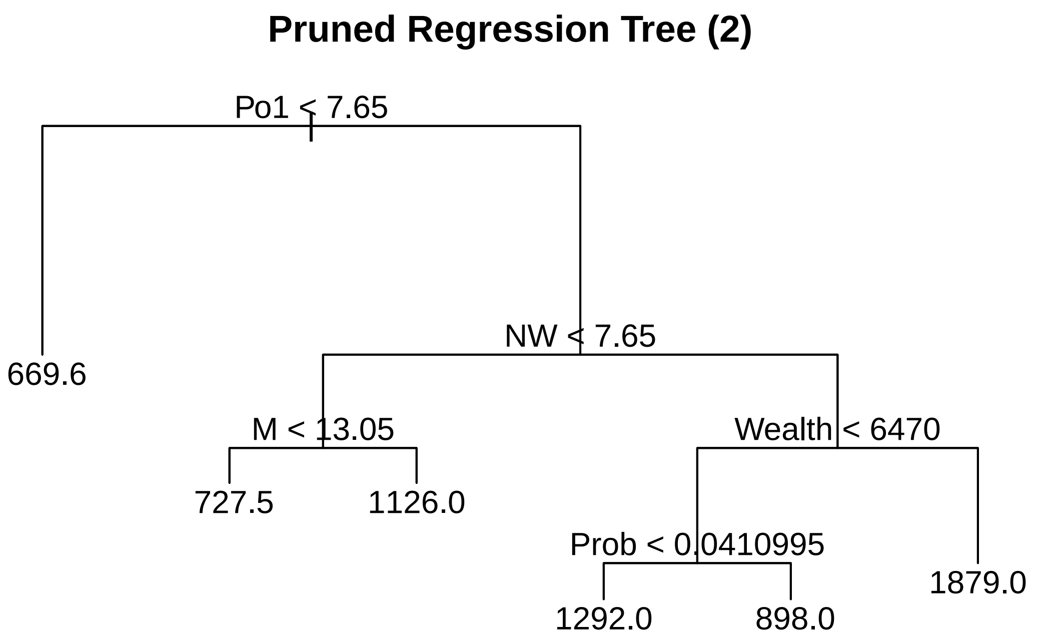
We can see that after pruning the LF split on the left is gone.

Prune the second tree:

```
#prune
tree2_pruned <- prune.tree(tree2, best=6)
summary(tree2_pruned)
```

```
##
## Regression tree:
## snip.tree(tree = tree2, nodes = 2L)
## Variables actually used in tree construction:
## [1] "Po1"    "NW"     "M"      "Wealth" "Prob"
## Number of terminal nodes:  6
## Residual mean deviance:  32480 = 1332000 / 41
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -327.6  -135.6    22.5     0.0   124.9   402.4
```

```
#plot
plot(tree2_pruned)
text(tree2_pruned)
title(main = "Pruned Regression Tree (2)")
```



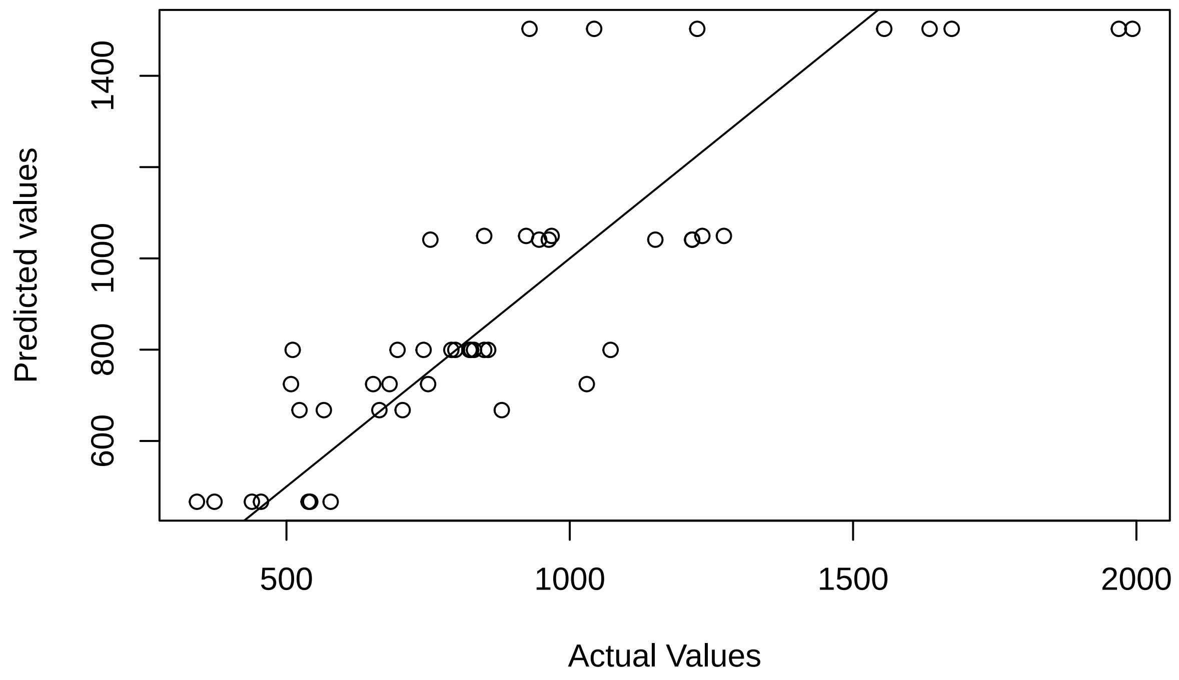
And here, on the other hand, only the right side of the Po1 has further splits.

Step 6: Estimating quality

Now we have 4 trees - two unpruned (tree 1 and tree 2), and two pruned (by 1 split each). Let's estimate the quality of each and compare to see how pruning has affected the result.

Tree 1 (unpruned):

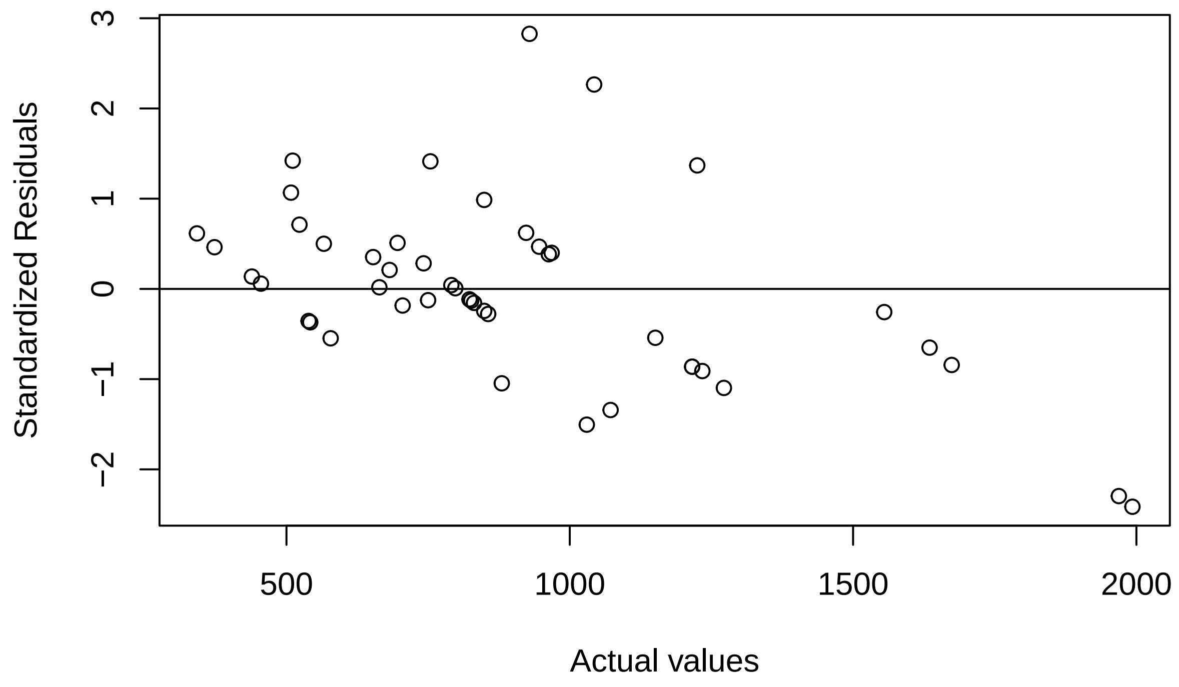
```
set.seed(11)
#predictions
tree1pred <- predict(tree1)

#plot predicted vs actual Crime values
plot(data$Crime, tree1pred, xlab="Actual Values", ylab="Predicted values")
abline(0,1)
```



As we can see on the plot, although some points are close to the line, in general predictions are not very good, and even the range of predicted values is significantly smaller compare to the actual ones (it ends at ~1550, while some actual values are at 2000).

Check the residuals:

```
plot(data$Crime, scale(tree1pred - data$Crime),  xlab="Actual values", ylab="Standardized Residuals")
abline(0,0)
```



The x and y axis are balanced and there doesn't seem to be a non-linear trend, although some residuals are further from the line than the others.

Find R-Squared:

```
#Calculate SSres
tree1_ssr <- sum((tree1pred-data$Crime)^2)
tree1_ssr
```

```
## [1] 1895722
```

```
#SSTot
tree1_SStot <- sum((data$Crime - mean(data$Crime))^2)
tree1_SStot
```

```
## [1] 6880928
```

```
#R2
tree1_r2 <- 1 - tree1_ssr/tree1_SStot
tree1_r2
```

```
## [1] 0.7244962
```

For the first tree, we have **R-squared of 72% on training data**. Can it be due to overfitting? We can compare each of the nodes with a cross-validated model by using the Prune function on the tree. We can then compare the values with Sum of Squares Total - If a cross validated model has values of deviation bigger than that, the model is probably overfitting:

```
#sst for comparison
tree1_SStot
```

```
## [1] 6880928
```

```
#values for each node of tree without CV
prune.tree(tree1)$dev
```

```
## [1] 1895722 2013257 2276670 2632631 3364043 4383406 6880928
```

```
#and for the CV model
tree1_cv$dev
```

```
## [1] 7321535 7368052 7617290 7819974 8308663 7572993 8268483
```

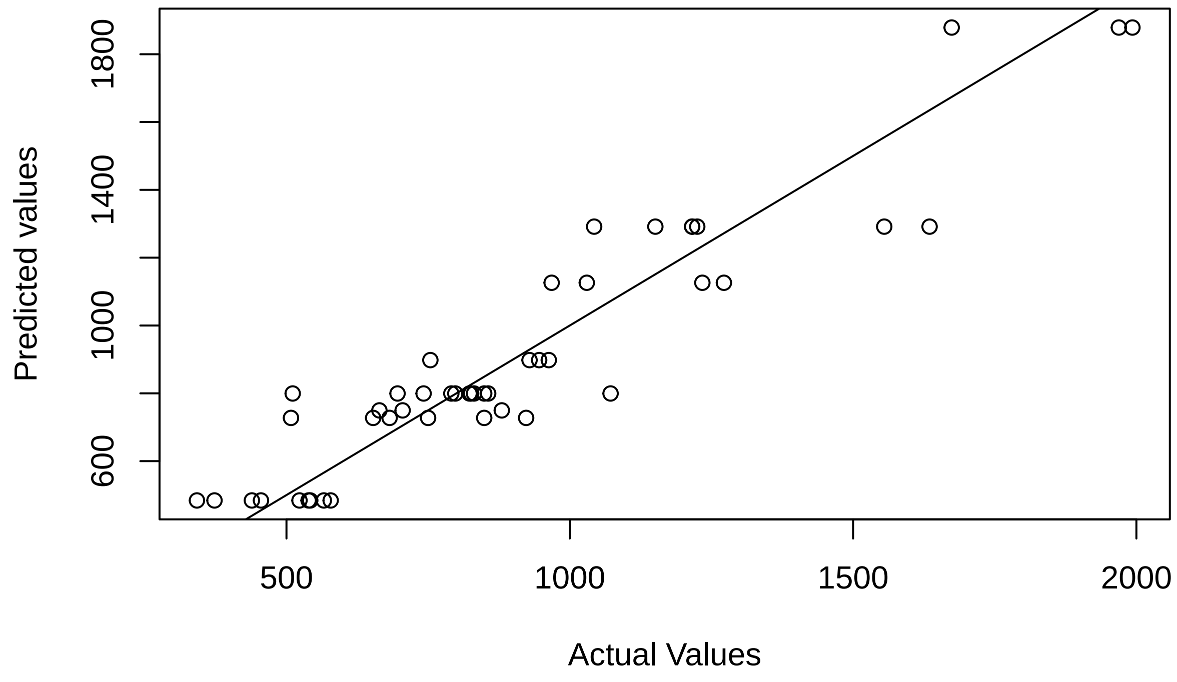
As we can see, the deviations for the CV Models are extremely large compared to the one fitted on training data -so **R-squared of 72% is just a sign of huge overfitting, not a good model**, and even SST is lower than those deviations.

Now we repeat the same procedure for Tree2 and the pruned trees.

Tree 2 (unpruned)

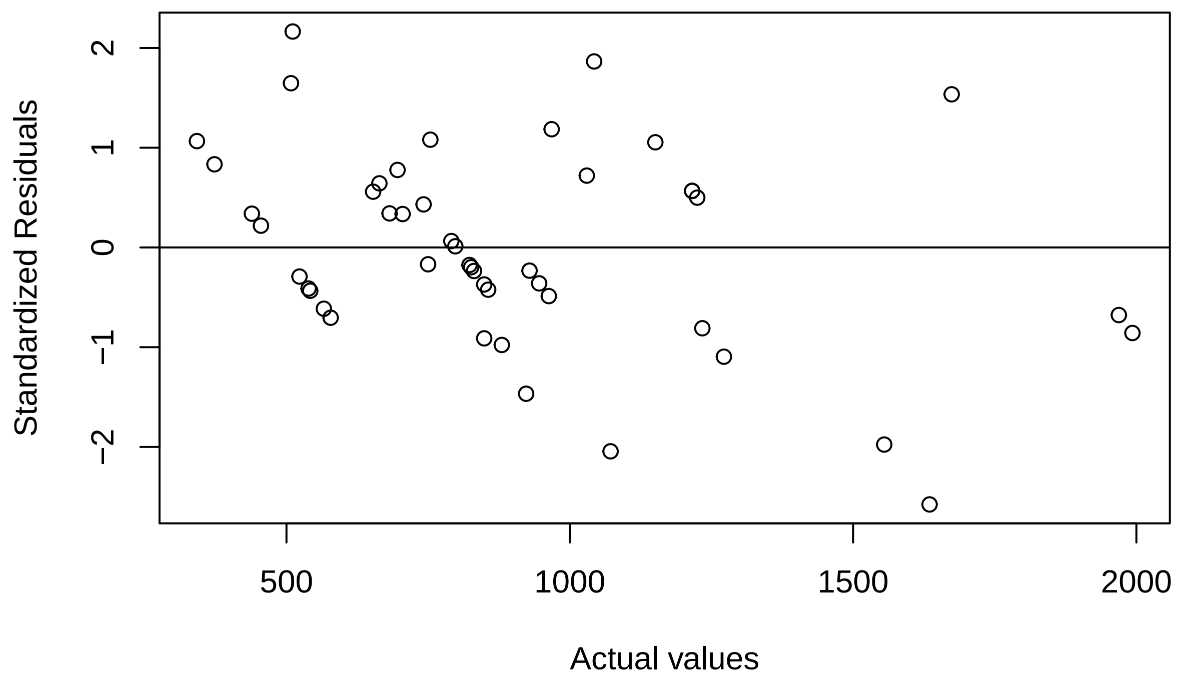
Compared to unpruned tree 1, the predicted values here are much closer to the actual ones:

```
set.seed(11)
#predictions
tree2pred <- predict(tree2)

#plot predicted vs actual Crime values
plot(data$Crime, tree2pred, xlab="Actual Values", ylab="Predicted values")
abline(0,1)
```



The residuals also look slightly better, the range is a bit smaller (-2 to 2 compared to -2 to 3 with Tree 1)

```r
plot(data$Crime, scale(tree2pred - data$Crime),  xlab="Actual values", ylab="Standardized Residuals")
abline(0,0)
```



SSTotal is the same, but **R2 is 88% on training data**.

```r
#Calculate SSres
tree2_ssr <- sum((tree2pred-data$Crime)^2)
tree2_ssr
```

```
## [1] 817148.2
```

```r
#SSTot
tree2_SStot <- sum((data$Crime - mean(data$Crime))^2)
tree2_SStot
```

```
## [1] 6880928
```

```r
#R2
tree2_r2 <- 1 - tree2_ssr/tree2_SStot
tree2_r2
```

```
## [1] 0.8812445
```

Is the overfitting as bad here as with Tree1?

```r
#sst for comparison
tree2_SStot
```

```
## [1] 6880928
```

```
#values for each node of tree without CV
prune.tree(tree2)$dev
```

```
## [1]  817148.2  975817.6 1331779.4 1712904.8 2107191.8 4383406.0 6880927.7
```

```
#and for the CV model
tree2_cv$dev
```

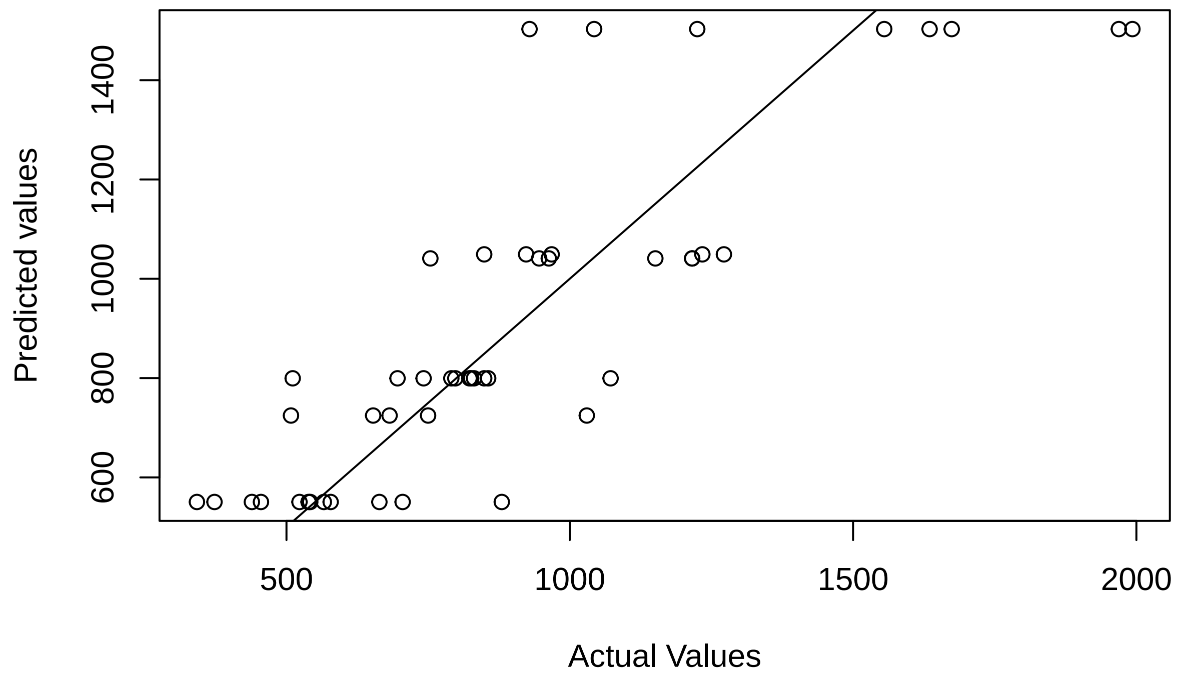
```
## [1] 6861535 6767877 7132280 6999703 6999703 6804821 8319169
```

Yes, we also have a lot of overfitting - this model is no good, just like the first tree.

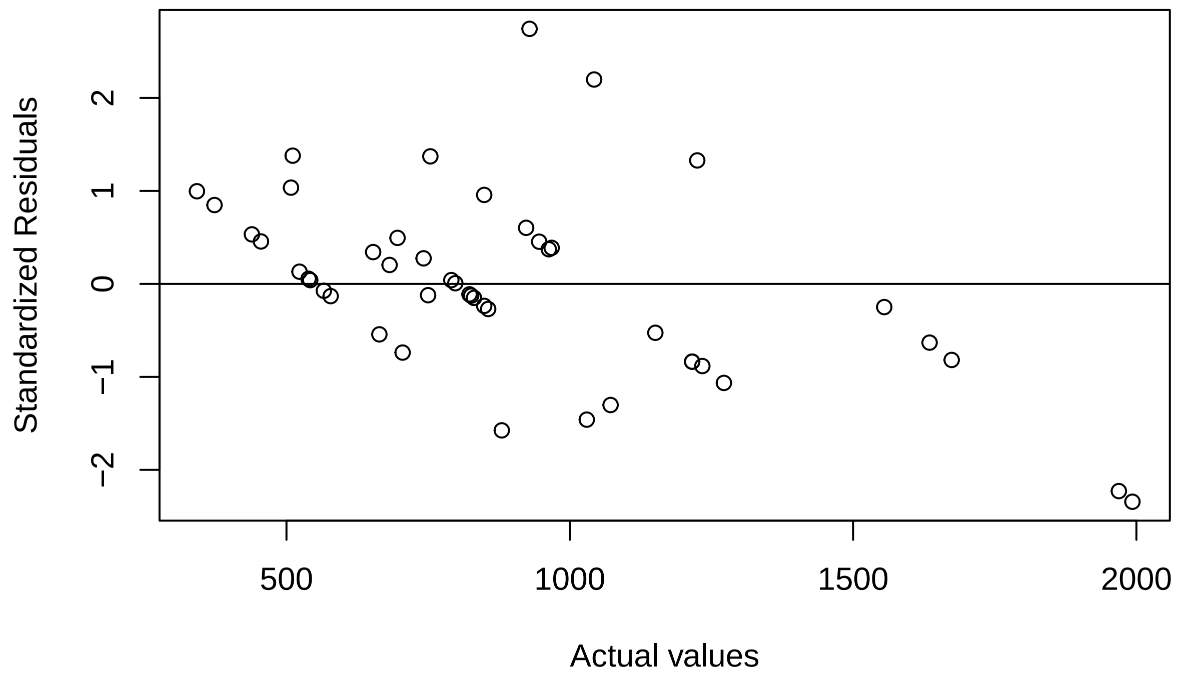
Check if something would be different for pruned trees:

Tree 1 (pruned)

```
set.seed(11)
#predictions
tree1pruned_pred <- predict(tree1_pruned)

#plot predicted vs actual Crime values
plot(data$Crime, tree1pruned_pred, xlab="Actual Values", ylab="Predicted values")
abline(0,1)
```



```
#residuals plot
plot(data$Crime, scale(tree1pruned_pred - data$Crime),  xlab="Actual values", ylab="Standardized Residu
abline(0,0)
```



```
#Calculate SSres
tree1pruned_ssr <- sum((tree1pruned_pred-data$Crime)^2)
tree1pruned_ssr
```

```
## [1] 2013257
```

```
#SSTot
tree1pruned_SStot <- sum((data$Crime - mean(data$Crime))^2)
tree1pruned_SStot
```

```
## [1] 6880928
```

```
#R2
tree1pruned_r2 <- 1 - tree1pruned_ssr/tree1pruned_SStot
tree1pruned_r2
```

```
## [1] 0.7074149
```

```
#sst for comparison
tree1pruned_SStot
```

```
## [1] 6880928
```

```
#values for each node of tree without CV
prune.tree(tree1_pruned)$dev
```

```
## [1] 2013257 2276670 2632631 3364043 4383406 6880928
```

```
#cross-validate:
tree1pruned_cv <- cv.tree(tree1_pruned)
```

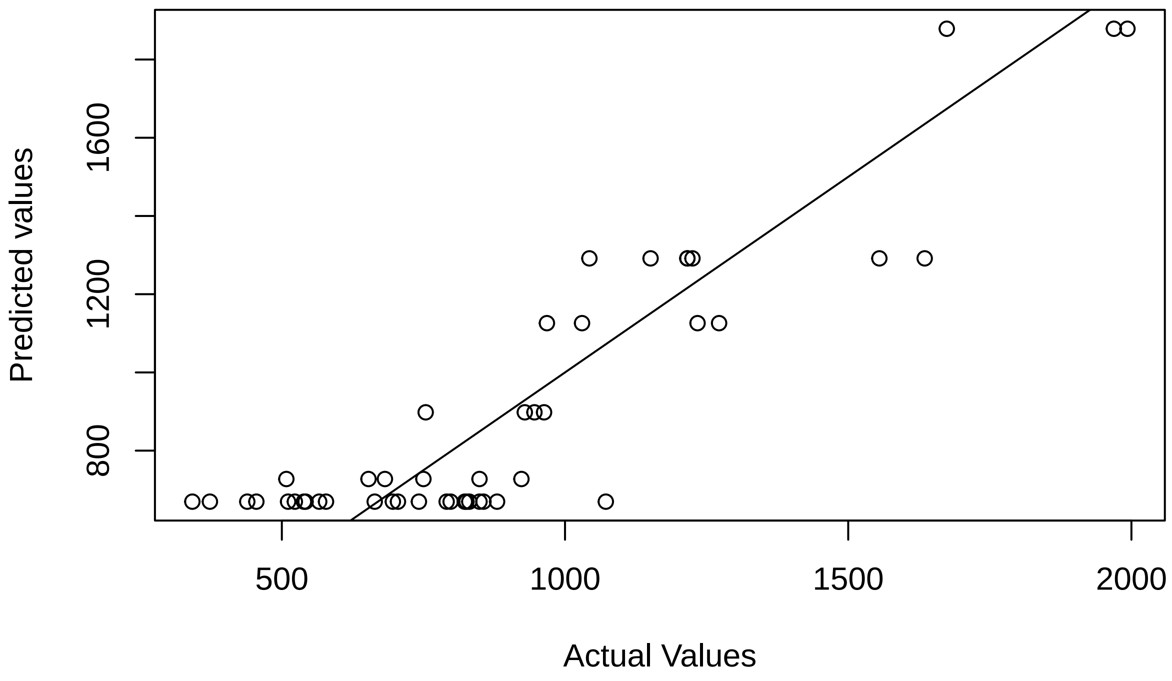
```
tree1pruned_cv$dev
```

```
## [1] 7321535 7617290 7819974 8308663 7572993 8268483
```

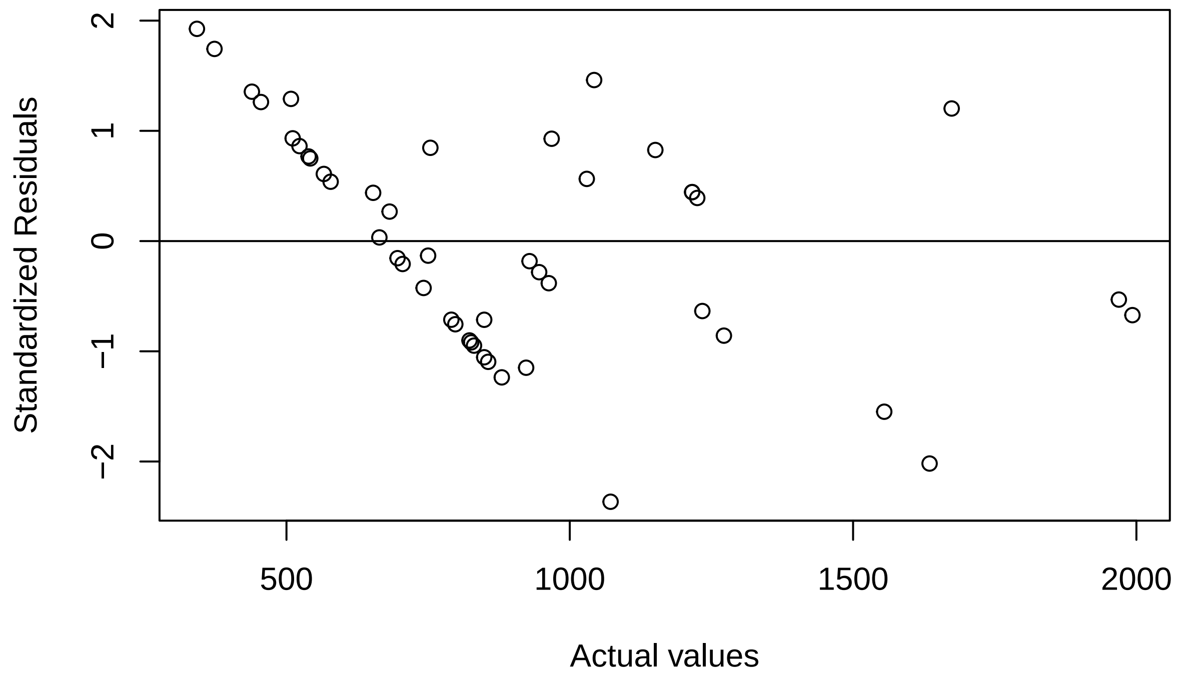
Unfortunately, for the pruned tree the situation is not better - deviations of the cross-validated model are huge and even larger than the total error - so a 70% R-squared is due to overfitting.

Tree 2 (pruned)

```
set.seed(11)
#predictions
tree2pruned_pred <- predict(tree2_pruned)

#plot predicted vs actual Crime values
plot(data$Crime, tree2pruned_pred, xlab="Actual Values", ylab="Predicted values")
abline(0,1)
```



```
#residuals plot
plot(data$Crime, scale(tree2pruned_pred - data$Crime),  xlab="Actual values", ylab="Standardized Residua
abline(0,0)
```



```
#Calculate SSres
tree2pruned_ssr <- sum((tree2pruned_pred-data$Crime)^2)
tree2pruned_ssr
```

```
## [1] 1331779
```

```
#SSTot
tree2pruned_SStot <- sum((data$Crime - mean(data$Crime))^2)
tree2pruned_SStot
```

```
## [1] 6880928
```

```
#R2
tree2pruned_r2 <- 1 - tree2pruned_ssr/tree2pruned_SStot
tree2pruned_r2
```

```
## [1] 0.8064535
```

```
#sst for comparison
tree2pruned_SStot
```

```
## [1] 6880928
```

```
#values for each node of tree without CV
prune.tree(tree2_pruned)$dev
```

```
## [1] 1331779 1712905 2107192 4383406 6880928
```

```
#cross-validate:
tree2pruned_cv <- cv.tree(tree2_pruned)
```

```
tree2pruned_cv$dev
```

```
## [1] 7321535 7819674 7721697 6199812 8268483
```

No luck with this tree either - an R-squared is 80% on training data, but the deviations are larger than the SSTotal.

As we have seen, overfitting is once again the problem for our model - mostly, it is due to the size of our data set. Using less factors in a tree would ruin the accuracy of predictions, and increasing them would lead to overfitting.

What are the next steps?

We could stop here and choose one of the 4 trees we have - perhaps the first one that is unpruned, although all of the are hugely overfitted and bad models (no big differences to consider when choosing).

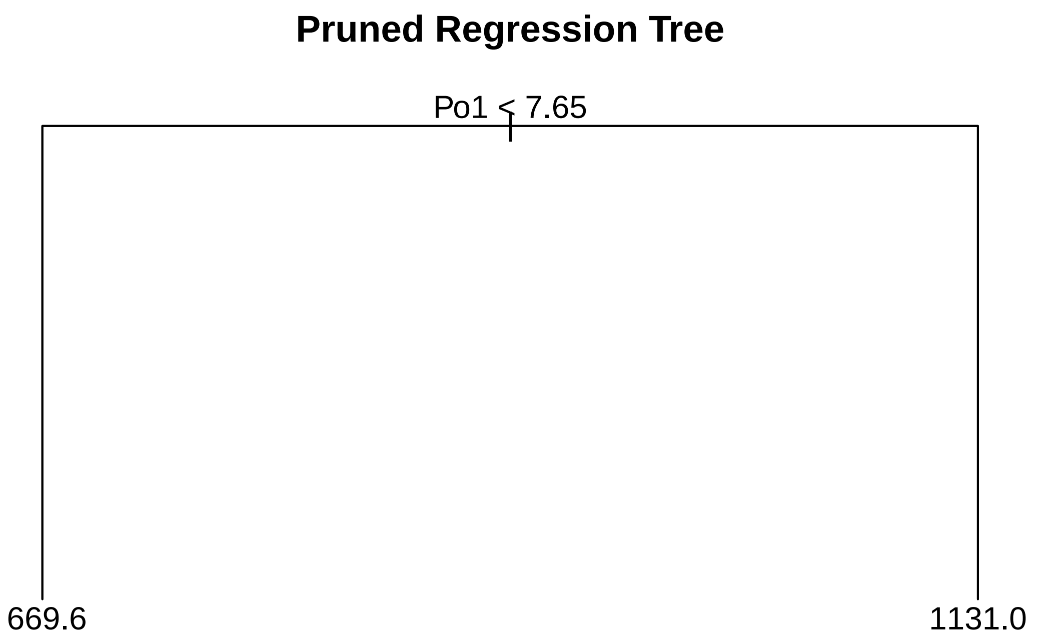
To improve the model, if we had more data, we could build linear regression models for each leaf. However, we only have 3-10 data points on each end - so building a regression would be useless - we would have too many predictors and very few data points, which would cause the same overfitting problem.

Therefore, we switch to plan B mentioned in the previous step - **Pruning one of the trees to size two (1-branch tree with 2 leaves)**.

Once we do that, we **build linear regression for each step, cross-validate, estimate the quality and use PCA, if needed**.

Step 7: Pruning to 1-branch tree with 2 leaves

```
#prune
final_tree <- prune.tree(tree1,best=2)
#plot
plot(final_tree)
text(final_tree)
title(main = "Pruned Regression Tree")
```



Now our tree braches only on Po1, police expenditure in 1960.

Let's estimate the quality - we do not expect it to be any better than the previous model, since we are going to add regressions to each leaf anyway:

```r
set.seed(11)
#predictions
tree_pred <- predict(final_tree)

#Calculate SSres
tree_ssr <- sum((tree_pred-data$Crime)^2)

#SSTot
tree_SStot <- sum((data$Crime - mean(data$Crime))^2)

#R2
tree_r2 <- 1 - tree_ssr/tree_SStot
tree_r2
```

```
## [1] 0.3629629
```

```r
#sst for comparison
tree_SStot
```

```
## [1] 6880928
```

```r
#values for each node of tree without CV
prune.tree(final_tree)$dev
```

```
## [1] 4383406 6880928
```

```
#cross-validate:
tree_cv <- cv.tree(final_tree)
tree_cv$dev
```

```
## [1] 7321535 8268483
```

On training data, we have an R-squared on 36%. And deviations after cross-validation are still bigger than SST. However, we have 23 and 24 values in each leaf, and **can now build a linear regression model**:

```
final_tree$frame
```

```
##      var  n       dev      yval splits.cutleft splits.cutright
## 1    Po1 47 6880927.7  905.0851          <7.65           >7.65
## 2 <leaf> 23  779243.5  669.6087
## 3 <leaf> 24 3604162.5 1130.7500
```

Step 8: Linear Regression for Leaf no.1

Checking for multicollinearity

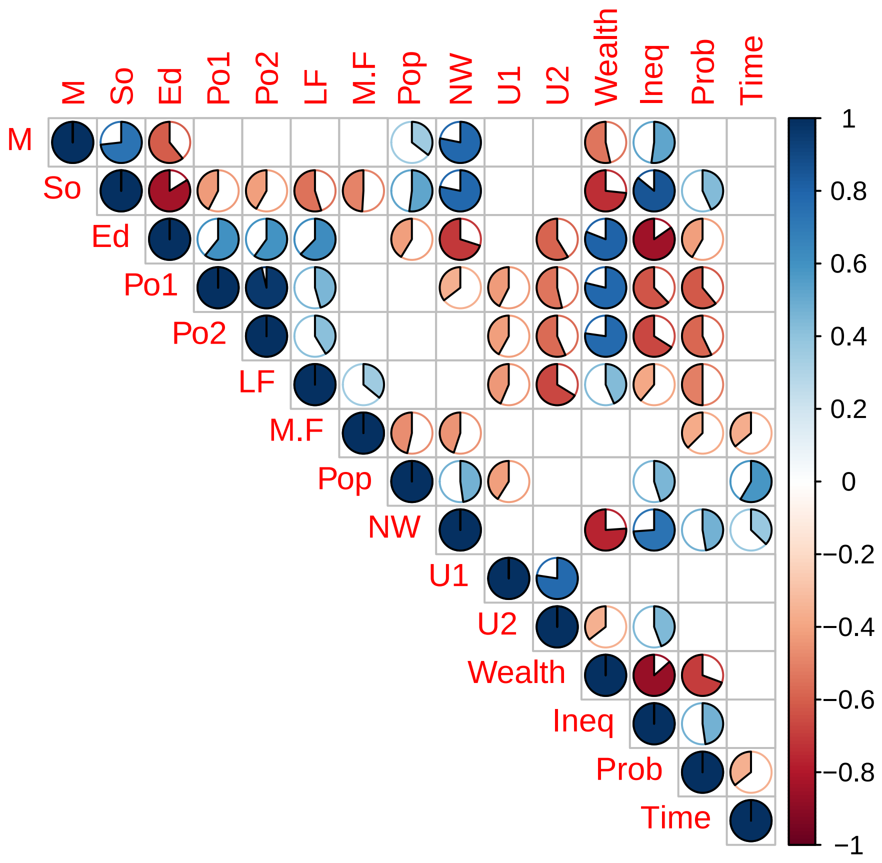
Before performing LR, we as usual check correlation of variables to detect possible issues in the model.

```
#get the data for leaf 1
d1 <- data[which(final_tree$where==2),]
head(d1)
```

```
##       M So   Ed Po1 Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1  15.1  1  9.1 5.8 5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 3  14.2  1  8.9 4.5 4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 9  15.7  1  9.0 6.5 6.2 0.553  95.5  39 28.6 0.081 2.8   4210 23.9 0.071697
## 10 14.0  0 11.8 7.1 6.8 0.632 102.9   7  1.5 0.100 2.4   5260 17.4 0.044498
## 12 13.4  0 10.8 7.5 7.1 0.595  97.2  47  5.9 0.083 3.1   5800 17.2 0.031201
## 13 12.8  0 11.3 6.7 6.0 0.624  97.2  28  1.0 0.077 2.5   5070 20.6 0.045302
##       Time Crime
## 1  26.2011   791
## 3  24.3006   578
## 9  29.4001   856
## 10 19.5994   705
## 12 34.2984   849
## 13 36.2993   511
```

I will check this with a correlation plot combined with significance of correlation (via p values):

```
#p values
cor.mtest <- function(mat, ...) {
    mat <- as.matrix(d1[,-16])
    n <- ncol(mat)
    p.mat<- matrix(NA, n, n)
    diag(p.mat) <- 0
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
            tmp <- cor.test(mat[, i], mat[, j], ...)
            p.mat[i, j] <- p.mat[j, i] <- tmp$p.value
        }
    }
  colnames(p.mat) <- rownames(p.mat) <- colnames(mat)
  p.mat
}
# matrix of the p-value of the correlation
p.mat <- cor.mtest(mtcars)

#corrplot
corrplot(cor(d1[,-16]),
         method='pie',
         type="upper",
         #order="hclust",
         p.mat = p.mat,
         sig.level = 0.1,
         insig = "blank")
```



As before in the full data set, we have quite a few pairs of predictors with correlation greater than 0.5 - we will definitely have to remove some of the variables to avoid multicollinearity in the model

Linear Regression

We have 23 data points, and according to the **1:10 ration**, it's better for us to have ~2-3 predictors in the model. To quickly figure out which predictors should be used, we build a model with all parameters first:

```
#l1m1 - leaf 1 model 1
l1m1 <- lm(Crime~., data=d1)
summary(l1m1)
```

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

```
## lm(formula = Crime ~ ., data = d1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -109.147  -52.803   -6.495   53.784  127.196
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -48.5477  2044.9766  -0.024   0.9817
## M              45.8622    58.6256   0.782   0.4597
## So            380.4815   223.1072   1.705   0.1319
## Ed            187.9074    89.5799   2.098   0.0741 .
## Po1            -3.5138   157.7513  -0.022   0.9829
## Po2            44.6382   148.5528   0.300   0.7725
## LF           1059.3652  1187.9722   0.892   0.4021
## M.F           -22.5521    21.4677  -1.051   0.3284
## Pop            10.6413     5.0929   2.089   0.0750 .
## NW              0.1010     7.9019   0.013   0.9902
## U1           4878.2802  4874.8165   1.001   0.3503
## U2             -5.5126   133.5094  -0.041   0.9682
## Wealth         -0.1022     0.1752  -0.583   0.5779
## Ineq            4.7779    35.5290   0.134   0.8968
## Prob        -7317.4407  3280.7511  -2.230   0.0609 .
## Time          -20.0603     7.7287  -2.596   0.0357 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 115.9 on 7 degrees of freedom
## Multiple R-squared:  0.8794, Adjusted R-squared:  0.6209
## F-statistic: 3.403 on 15 and 7 DF,  p-value: 0.0541
```

Only **four factors are significant, and R2=88% is probably due to overfitting**. Let's reduce factors to significant-only and repeat the process until all factors of the model are significant:

```
l1m2 <- lm(Crime~Ed+Pop+Prob+Time, data=d1)
summary(l1m2)
```

```
##
## Call:
## lm(formula = Crime ~ Ed + Pop + Prob + Time, data = d1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -206.35  -90.22   -7.59   59.64  357.11
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   819.960    515.112   1.592   0.1288
## Ed              9.499     34.869   0.272   0.7884
## Pop            11.395      3.229   3.529   0.0024 **
## Prob        -3164.075   2095.755  -1.510   0.1485
## Time          -12.130      6.830  -1.776   0.0927 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 154.5 on 18 degrees of freedom
## Multiple R-squared:  0.4485, Adjusted R-squared:  0.3259
## F-statistic: 3.659 on 4 and 18 DF,  p-value: 0.02379
```

Now only 2 factors are significant, reduce again:

```r
l1m3 <- lm(Crime~Pop+Time, data=d1)
summary(l1m3)
```

```
##
## Call:
## lm(formula = Crime ~ Pop + Time, data = d1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -185.42 -117.40  -23.00   36.78  391.19
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  638.392    142.637   4.476 0.000232 ***
## Pop            9.571      3.114   3.074 0.005993 **
## Time          -6.969      6.255  -1.114 0.278412
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 160.7 on 20 degrees of freedom
## Multiple R-squared:  0.3369, Adjusted R-squared:  0.2706
## F-statistic: 5.081 on 2 and 20 DF,  p-value: 0.01644
```

And now only 1 - so we leave just one factor, Pop, population of state:

```r
l1m4 <- lm(Crime~Pop, data=d1)
summary(l1m4)
```

```
##
## Call:
## lm(formula = Crime ~ Pop, data = d1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -224.71 -114.53    0.29   60.43  400.75
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  497.836     66.949   7.436  2.6e-07 ***
## Pop            7.540      2.539   2.970  0.00731 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 161.7 on 21 degrees of freedom
## Multiple R-squared:  0.2957, Adjusted R-squared:  0.2622
## F-statistic: 8.818 on 1 and 21 DF,  p-value: 0.007313
```

R2=29.6% on training data. To reduce overfitting, lets run cross validation and compare. LOO CV is more appropriate for this case, because we have very few data points - more samples will be used in each repetition with LOO:

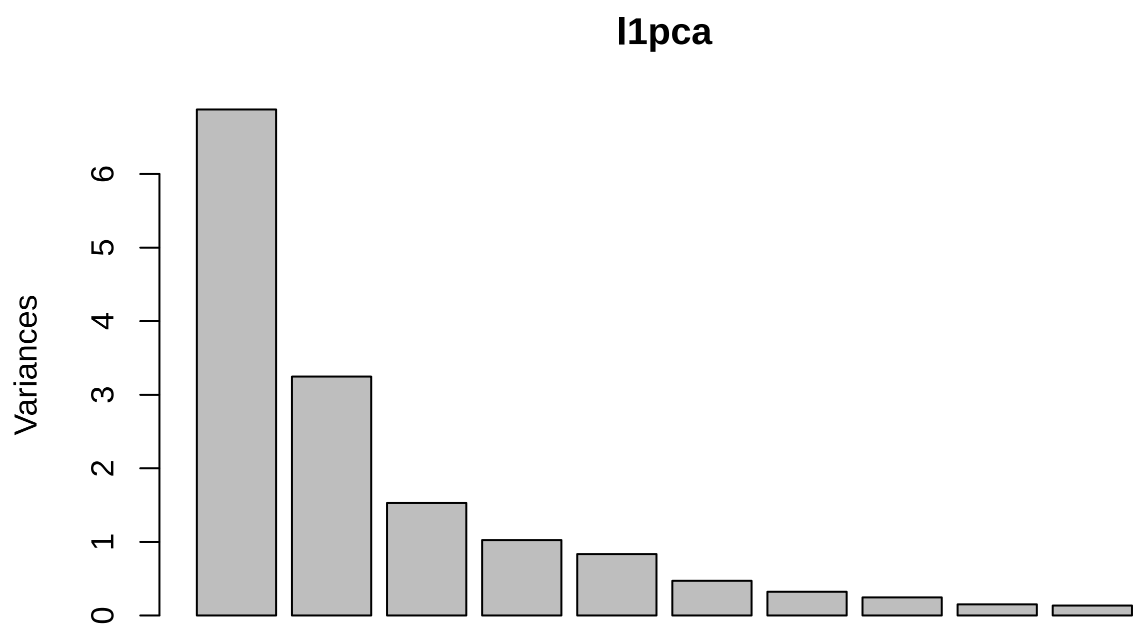
```r
l1m4cv <- cv.lm(d1, l1m4, m=nrow(d1), plotit=F, printit=F)
#R2
1 - attr(l1m4cv,"ms")*nrow(d1)/sum((d1$Crime - mean(d1$Crime))^2)
```

```
## [1] 0.1805179
```

For a Cross-Validated Linear Regression Using 1 Factor Population for Lead no.1, we have R-squared of 18%. This is a bad quality, and I will try to improve it using PCA:

PCA

```r
l1pca <- prcomp(~., data=d1[,-16], scale=T)
screeplot(l1pca)
```



After the 4th component values are lower than 1. So we can build a model using 4 PCs and then remove them one by one if not all will be significant:

```r
pca1m1 <- lm(Crime~., data=as.data.frame(cbind(Crime=d1[,16], l1pca$x[,1:4])))
summary(pca1m1)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = as.data.frame(cbind(Crime = d1[,
##     16], l1pca$x[, 1:4])))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -340.31  -62.35   10.72   84.83  328.11
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  669.609     34.014  19.686 1.27e-13 ***
## PC1           -1.808     13.261  -0.136  0.89308
## PC2           64.629     19.300   3.349  0.00358 **
## PC3            1.786     28.113   0.064  0.95005
## PC4            7.507     34.346   0.219  0.82944
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 163.1 on 18 degrees of freedom
## Multiple R-squared:  0.3853, Adjusted R-squared:  0.2487
## F-statistic: 2.821 on 4 and 18 DF,  p-value: 0.05597
```

Only one component is significant, so we leave PC2 only:

```
pca1m2 <- lm(Crime~., data=as.data.frame(cbind(Crime=d1[,16], l1pca$x[,2])))
summary(pca1m2,2)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = as.data.frame(cbind(Crime = d1[,
##     16], l1pca$x[, 2])))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -355.63  -54.42   12.81   77.87  327.27
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   669.61      31.55   21.22 1.14e-15 ***
## V2             64.63      17.90    3.61  0.00164 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 151.3 on 21 degrees of freedom
## Multiple R-squared:  0.3829, Adjusted R-squared:  0.3535
## F-statistic: 13.03 on 1 and 21 DF,  p-value: 0.001645
##
## Correlation of Coefficients:
##    (Intercept)
## V2 0.00
```

With PCA, we have R2 of 38% on training data. Let's check if the cross validated model would have a better result compared to cross-validated trees with 6 or 7 branches:

```
pca1cv <- cv.lm(as.data.frame(cbind(Crime=d1[,16], l1pca$x[,2])),pca1m2,m=nrow(d1), plotit=F, printit =

#calculate R2

1 - attr(pca1cv,"ms")*nrow(d1)/sum((d1$Crime - mean(d1$Crime))^2)
```

```
## [1] 0.3041442
```

Cross-validated model using PCA is shows a significant improvement in R2 - 30.4%, although

its R2 was not so high on training data. This suggests that this model has less overfitting, and 30% is more of a real result.

To sum up, for the first leaf of the model with 1 branch and 2 leaves, we should use linear regression with pca, using PC2 as the factor.

Unscale to get model equation:

```
#getting rid of 'e'values in output
options("scipen"=100, "digits"=4)
#getting coefs
int_sc <- summary(pca1m2)$coefficients[1]
coefs1_sc <- summary(pca1m2)$coefficients[2]
a_sc <- l1pca$rotation[,2]
#unscaling
unsc_coefs <- a_sc/l1pca$scale
unsc_coefs
```

```
##            M           So           Ed          Po1          Po2           LF
##   0.18868401   0.36704294  -0.02343830   0.33884999   0.32519728   2.81189850
##          M.F          Pop           NW           U1           U2       Wealth
##  -0.09516939   0.03012692   0.01942132 -23.72116465  -0.34948738   0.00005869
##         Ineq         Prob         Time
##   0.01750349  -7.06468939   0.05300165
```

```
unsc_int <-int_sc - sum(a_sc*l1pca$center/l1pca$scale)
unsc_int
```

```
## [1] 671.4
```

Step 9: Linear Regression for Leaf no.2

Checking for multicollinearity

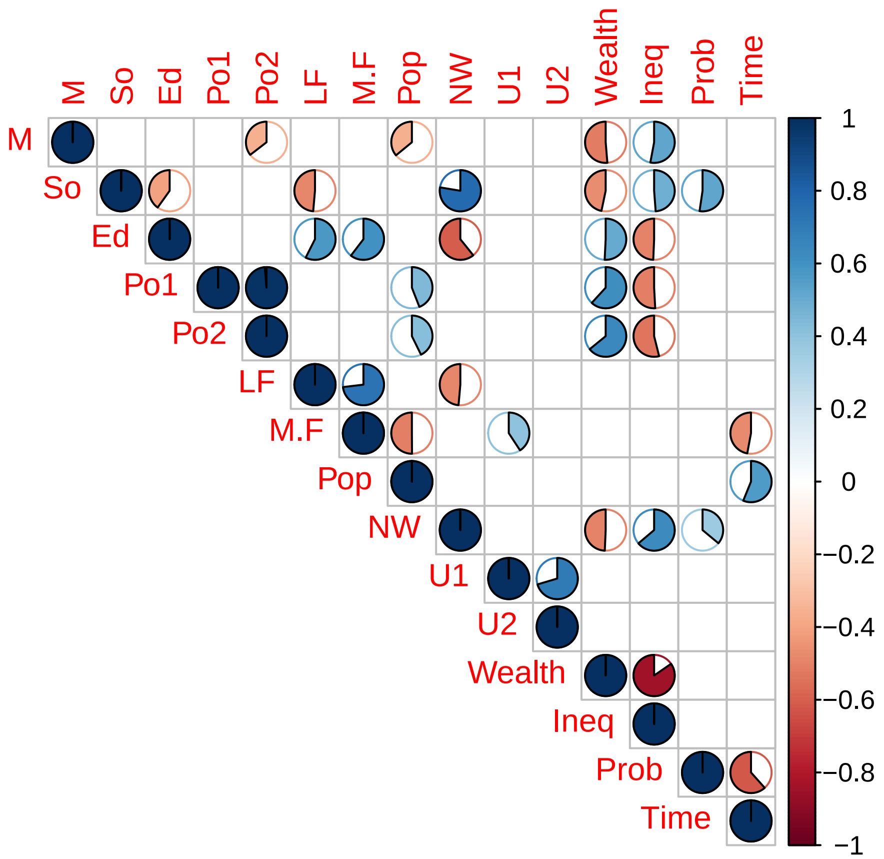
As for leaf 2, let's start with multicollinearity check using corrplot with p-value for correlation to exclude insignificant pairs of predictors.

```
#get the data for leaf 2
d2 <- data[which(final_tree$where==3),]
head(d2)
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq   Prob Time
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.0296 25.3
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.0158 29.9
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.0414 21.3
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.0342 21.0
## 7 12.7  1 11.1  8.2  7.9 0.519  98.2   4 13.9 0.097 3.8   6200 16.8 0.0421 20.7
## 8 13.1  1 10.9 11.5 10.9 0.542  96.9  50 17.9 0.079 3.5   4720 20.6 0.0401 24.6
##   Crime
## 2  1635
## 4  1969
## 5  1234
## 6   682
## 7   963
## 8  1555
```

Corrplot:

```
#p values
cor.mtest <- function(mat, ...) {
    mat <- as.matrix(d2[,-16])
    n <- ncol(mat)
    p.mat<- matrix(NA, n, n)
    diag(p.mat) <- 0
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
            tmp <- cor.test(mat[, i], mat[, j], ...)
            p.mat[i, j] <- p.mat[j, i] <- tmp$p.value
        }
    }
  colnames(p.mat) <- rownames(p.mat) <- colnames(mat)
  p.mat
}
# matrix of the p-value of the correlation
p.mat <- cor.mtest(mtcars)

#corrplot
corrplot(cor(d2[,-16]),
         method='pie',
         type="upper",
         #order="hclust",
         p.mat = p.mat,
         sig.level = 0.1,
         insig = "blank")
```



Although for leaf two we have fewer correlated pairs, there still are some - especially among variable that describe similar things (inequality and wealth, Po1 and Po2 - police protection expenditure, U1 and U2). Let's see if we would have a better linear regression

Linear Regression

Once again, although predictors-data points ration is far from the perfect 1:10, let's start with a linear regression using all parameters:

```
#l1m1 - leaf 1 model 1
l2m1 <- lm(Crime~., data=d2)
summary(l2m1)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = d2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -206.80 -120.41   -9.49  103.98  248.23
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8634.170   2366.404   -3.65   0.0065 **
## M               5.603     96.162    0.06   0.9550
## So            179.627    409.521    0.44   0.6725
## Ed            263.085    146.423    1.80   0.1101
## Po1           235.235    166.129    1.42   0.1945
## Po2          -140.702    193.876   -0.73   0.4887
## LF           1442.421   4832.446    0.30   0.7729
## M.F            -1.238     54.816   -0.02   0.9825
## Pop            -3.769      2.883   -1.31   0.2275
## NW             -0.540     24.504   -0.02   0.9830
## U1          -3779.984  10923.343   -0.35   0.7382
## U2            163.711    150.536    1.09   0.3085
## Wealth          0.302      0.205    1.47   0.1795
## Ineq          155.375     65.508    2.37   0.0451 *
## Prob        -3624.071   4381.472   -0.83   0.4321
## Time           21.933     14.675    1.49   0.1734
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 230 on 8 degrees of freedom
## Multiple R-squared:  0.883,  Adjusted R-squared:  0.663
## F-statistic: 4.01 on 15 and 8 DF,  p-value: 0.0267
```

Only **one factor is significant, Inequality, and R2=88% on training data is definitely because of overfitting**. One factor would probably be unhelpful just like in the ifrst leaf's case, so I will leave 2 predictors with the smallest Pr>|t| value, Inequality and Po1, since it has the second smallest value:

```
l2m2 <- lm(Crime~Ineq+Po1, data=d2)
summary(l2m2)
```

```
##
## Call:
## lm(formula = Crime ~ Ineq + Po1, data = d2)
```

```
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -612.1 -215.1   61.3  175.9  424.5
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1402.4      686.8   -2.04   0.0539 .
## Ineq            73.3       25.7    2.85   0.0096 **
## Po1            116.2       31.1    3.74   0.0012 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 316 on 21 degrees of freedom
## Multiple R-squared:  0.42,   Adjusted R-squared:  0.364
## F-statistic: 7.59 on 2 and 21 DF,  p-value: 0.0033
```

This time both factors are significant, and **R2 on training data is 42%**. Let's cross-validate the model to see if it's actually bad and this value is due to overfitting only:

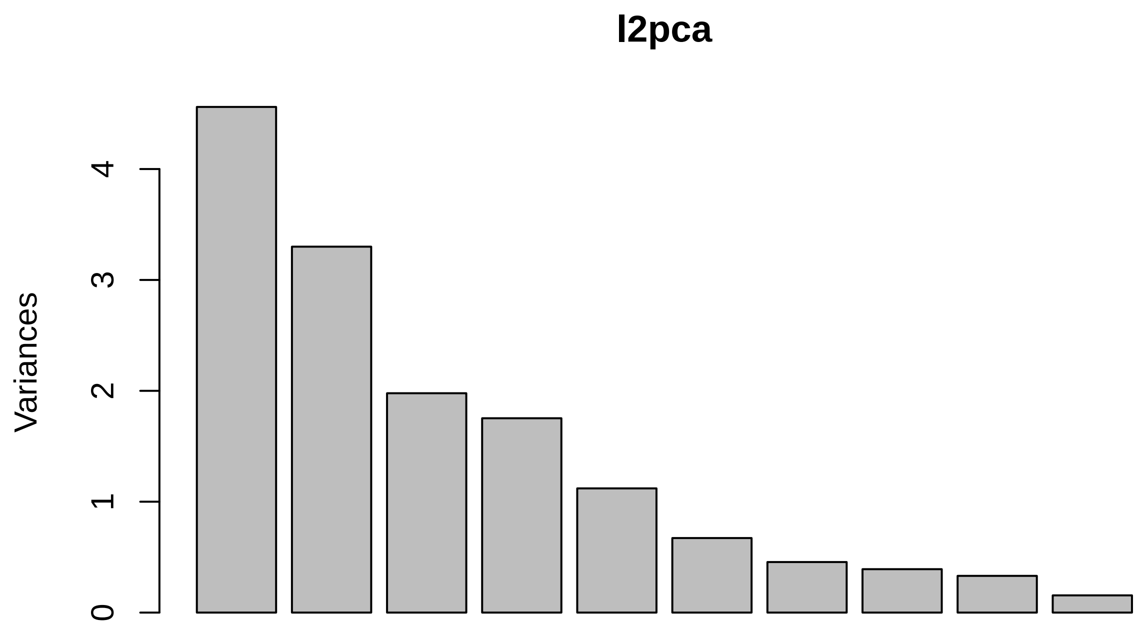
```
l2m2cv <- cv.lm(d2, l2m2, m=nrow(d2), plotit=F, printit=F)
#R2
1 - attr(l2m2cv,"ms")*nrow(d2)/sum((d2$Crime - mean(d2$Crime))^2)
```

```
## [1] 0.1469
```

Yes, R2 **reduced to 15% after CV, so the model was overfitting**. We cannot use it for the second leaf. Let's try a remedy - PCA:

PCA

```
l2pca <- prcomp(~., data=d2[,-16], scale=T)
screeplot(l2pca)
```



The first 5 components have values over 1 on screeplot, so we start with them and reduce their number depending on significance:

```
pca2m1 <- lm(Crime~., data=as.data.frame(cbind(Crime=d2[,16], l2pca$x[,1:5])))
summary(pca2m1)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = as.data.frame(cbind(Crime = d2[,
##     16], l2pca$x[, 1:5])))
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -357.9 -166.9   34.3  131.1  364.9
##
## Coefficients:
##             Estimate Std. Error t value           Pr(>|t|)
## (Intercept)  1130.75      48.49   23.32 0.0000000000000067 ***
## PC1            32.06      23.20    1.38               0.18
## PC2             2.88      27.27    0.11               0.92
## PC3            15.34      35.22    0.44               0.67
## PC4           -31.45      37.42   -0.84               0.41
## PC5           307.05      46.80    6.56 0.0000036492324846 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 238 on 18 degrees of freedom
## Multiple R-squared:  0.718,  Adjusted R-squared:  0.64
## F-statistic: 9.17 on 5 and 18 DF,  p-value: 0.000179
```

Only the last, 5th component, is significant, so we rebuild the model using only PC5:

```
pca2m2 <- lm(Crime~., data=as.data.frame(cbind(Crime=d2[,16], l2pca$x[,5])))
summary(pca2m2,2)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = as.data.frame(cbind(Crime = d2[,
##     16], l2pca$x[, 5])))
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -443.1 -170.9   18.7  181.3  438.1
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept)   1130.8       47.2   23.97 < 0.0000000000000002 ***
## V2             307.1       45.5    6.74           0.00000089 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 231 on 22 degrees of freedom
## Multiple R-squared:  0.674,  Adjusted R-squared:  0.659
## F-statistic: 45.5 on 1 and 22 DF,  p-value: 0.000000888
##
## Correlation of Coefficients:
##    (Intercept)
## V2 0.00
```

With PCA, we have R2 of 67% on training data. Let's cross-validate the model to find the real R2:

```
pca2cv <- cv.lm(as.data.frame(cbind(Crime=d2[,16], l2pca$x[,5])),pca2m2,m=nrow(d2), plotit=F, printit =

#calculate R2

1 - attr(pca2cv,"ms")*nrow(d2)/sum((d2$Crime - mean(d2$Crime))^2)
```

```
## [1] 0.6139
```

Cross-validated model using PCA has only a slight change in R2 - it reduced to 61%. This is the best quality we've had so far, so we leave this model as the final one for leaf 2.

Let's get the equation for leaf 2:

```
#getting rid of 'e'values in output
options("scipen"=100, "digits"=4)
#getting coefs
int_sc <- summary(pca2m2)$coefficients[1]
coefs1_sc <- summary(pca2m2)$coefficients[2]
a_sc <- l2pca$rotation[,5] #fifth PC
#unscaling
unsc_coefs <- a_sc/l2pca$scale
unsc_coefs
```

```
##          M         So          Ed         Po1         Po2          LF         M.F
##   0.4898621  0.1878390  0.1436230  0.1532393  0.1383011  8.7824136  0.1084171
##         Pop         NW         U1         U2     Wealth       Ineq       Prob
##   0.0042952  0.0438051  3.7497660  0.1465392 -0.0001574  0.1124151 -3.5505393
##        Time
##   0.0263755
```

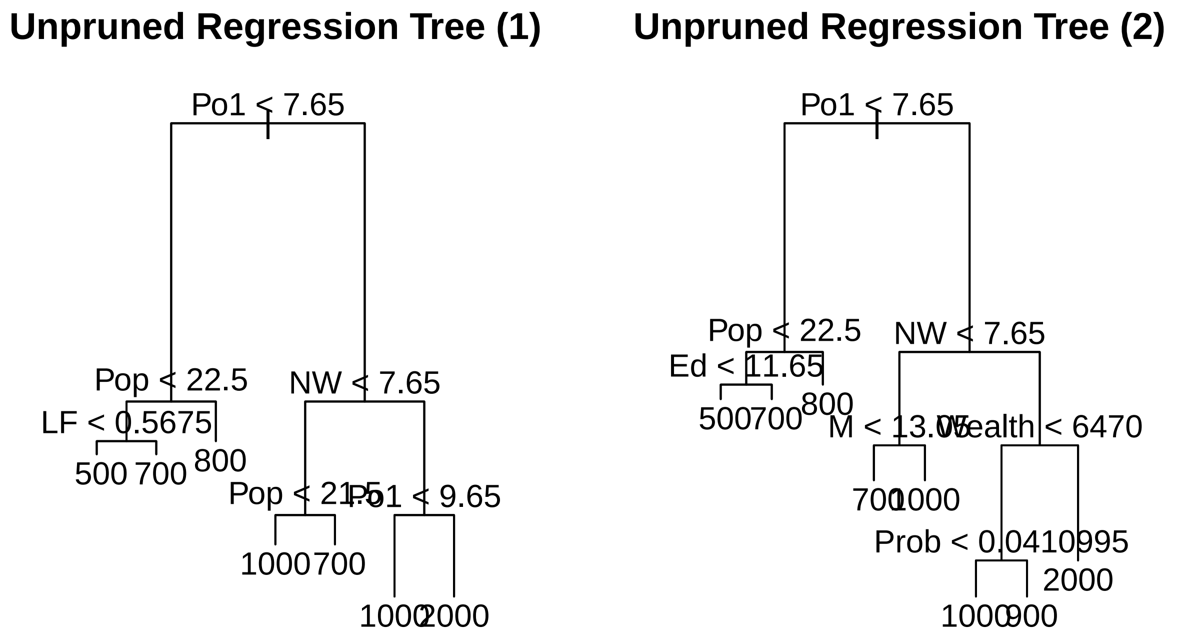
```
unsc_int <-int_sc - sum(a_sc*l2pca$center/l2pca$scale)
unsc_int
```

```
## [1] 1101
```

Step 10: Conclusion: Choosing the best model

We first started by growing **two regression trees** - one with default tree() function settings, and another with manual setting of minimum 3 data points in each leaf (5% rule), which produced two slightly different trees with 7 and 8 nodes:

```
par(mfrow=c(1,2))
plot(tree1)
text(tree1)
title(main = "Unpruned Regression Tree (1)")
plot(tree2)
text(tree2)
title(main = "Unpruned Regression Tree (2)")
```

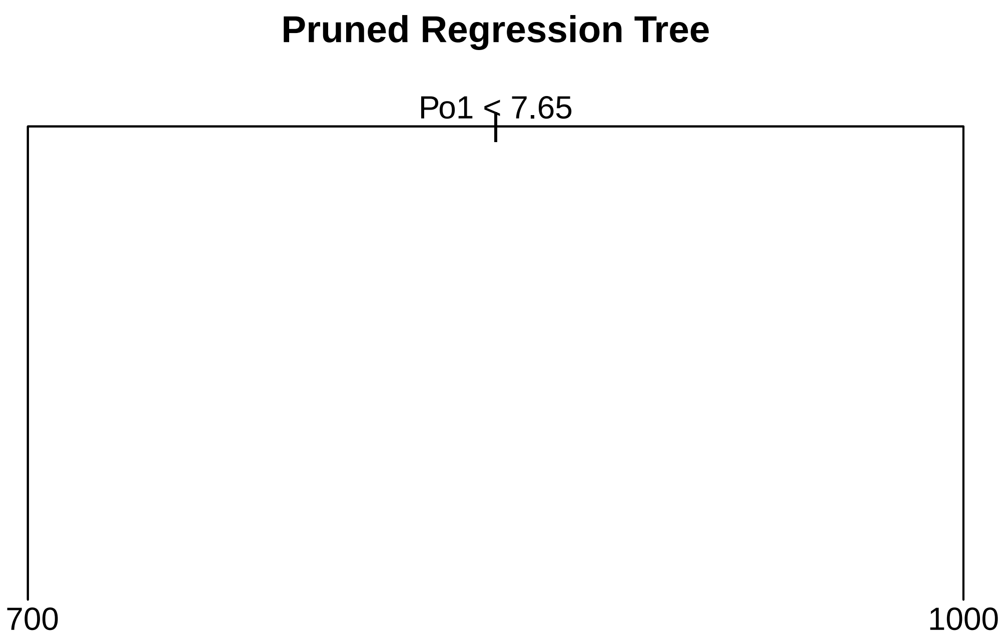


Based on **RMSE vs Tree size plot** we saw that pruning was not necessary for tree 1, as the number of nodes we already had provided the best possible explanation. However, we still pruned tree 1 by 1 node - the loss of explanation was minimal, but a smaller tree meant easier interpretation. Tree 2 was pruned to 7 nodes, since that size had the lowest RMSE.

We then estimated the quality for each tree of the four we had (pruned and unpruned) on training data, and after that compared the result to cross-validated models. Although all 4 trees (Tree 1, 2, pruned and unpruned) had R-squared values of 70-88% on training data, after cross validation deviation values for each of the tree nodes were extremely high, higher than even the SSTotal. That meant that there was huge overfitting and we couldn't use such trees.

As a remedy, the only option was to build linear regressions on leaves. However, due to only 47 data points in the set, we had 5-10 point on each leaf, so linear regression would have been useless with such a small set and would have led to overfitting. This is why we pruned to a **1-branch tree with 2 leaves**, where the main split was based on police expenditure in 1960 - this tree is part of our final model:

```r
plot(final_tree)
text(final_tree)
title(main = "Pruned Regression Tree")
```



On each of the leaves, we built linear regressions, they ended up having only one and two predictors correspondingly for leaf 1 and 2, and after cross-validation they had low R2 of 18% and 15%.

To fix that, we performed PCA and built a regression using that. For leaf 1, we started with the first four PCs (based on Screeplot, for PC 1-4 value was >1), but the final model used only the second PC, as other PCs were insignificant for our linear regression. After cross-validation, **R2 was estimated at 30%**, which was a good result compared to what we had before that, and not a significant decline from 38% for R2 on training data. Therefore, for leaf 1 (Po1 <7.65 == YES) the equation for linear regression (unscaled) is:

Crime ~ 671 + 0.189M + 0.368So - 0.023Ed + 0.339Po1 + 0.326Po2 + 2.812LF - 0.0952MF + 0.0301Pop + 0.0194NW- 23.721U1 - 0.349U2 + 0.000059Wealth + 0.0175Ineq - 7.065Prob + 0.053Time

On the second leaf we first built a linear regression using the first 5 Pricnipal Components (based on Screeplot, the first 5 PCs had values >1), but as with leaf one, only one component was significant (PC5), so we used it for the final model. On training data we had R2 of 67%, and **after cross-validation R2 was 61.4%** - a good result, considering how few data points we had. After unscaling, the equation for linear equation on leaf 2 is:

Crime ~ 1101 + 0.4899M + 0.1879So + 0.144Ed + 0.153Po1 + 0.138Po2 + 8.782Lf + 0.108MF + 0.004Pop + 0.044NW + 3.7498U1 + 0.147U2 -0.0016Wealth + 0.112Ineq - 3.551Prob + 0.027Time.

Although the accuracy of the model is not ideal, this is the best it can get, due to the low number of data points and a big number of predictors (47 vs 15, when the ideal data-predictor ration is 10:1). However, the benefit of our model is that it is easier to explain it. Interpreting the model, we can say that the main predictor of Crime rate is Per capita police protection expenditure in 1960 - it is the first parameter by which the initial split is made. If the police expenditure in a state is lower than 7.65 per capita, then we can make a prediction using the first (left) leaf, which has a rough prediction of 670 for states in such a category. To make the prediction more accurate, we then use the linear regression model, where the initial predicted Crime value (if other factors = 0) is 671 crimes. The number of Crimes in a state increases by the following parameters (the bigger the value of parameters for the state, the more the crime rate increases: share of males aged 14-24, Southern state (=yes), Police expenditure in 1960 and in 1959 as well, participation of civilian urban males 14-24 y.o. in labour force, state population in 1960, percentage of nonwhites in population, median assets or family income, income inequality, average time served before release in prisons. On the other had, the number of crimes in a state in 1960 is decreased the more mean years of schooling 25+y.o. population has in the state and the more males there are per 100 females; surprisingly, unemployment rate of urban males (both age groups 14-24 and 35-39) also decreases Crime - the more unemployed, the less crime there is, and the crime is also lower, the higher the probability of imprisonment is.

If at the initial split we have a state with a value of police expenditure in 1960 higher than 7.65, then follow the second (right) leaf. Here, based on linear regression, we can see that crime amount is increased the more males aged 14-24 there are in the state, the fact that the state is southern also increases number of crimes, as well as mean years of schooling for 25+y.o. citizens. Bigger police expenditure also signalizes that there is more crime, as well as male participation in labour force, bigger amount of males per females, bigger population, larger share of non-whites in population, larger unemployment rate, inequality and time served in prisons, while more wealth and more probability of imprisonment is related to a lower amount of crime.

This is not a prescriptive model, so we wouldn't be able to make a conclusion of what should have been done back then to decrease crime. However, the tree helps us split states into two categories with slightly different effect of factors on the response, and we can also notice some logical relation, for example that more crime is associated with more inequality, more population and bigger police expenditures, as well as larger number of males and southern state. And there is a negative relationship between the amount of crime and probability of imprisonment in both leaves. We cannot say that one factor causes another, since linear regression is not supposed to explain causality, but with the model we can see the type (positive/negative) of relation between predictors and response.

Part 1.B Random Forest

Using the same data set, let's move to a Random Forest model.

Step 1: Load the data

Same Crime data:

```r
head(data)
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq   Prob Time
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.0846 26.2
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.0296 25.3
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.0834 24.3
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.0158 29.9
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.0414 21.3
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.0342 21.0
##   Crime
## 1   791
## 2  1635
## 3   578
## 4  1969
## 5  1234
## 6   682
```

Here we can transform So to a factor, since it is a binary variable:

```
# Since So is a binary variable, set it as factor for better model building
data <- transform(data, So=as.factor(data[,2]))
summary(data)
```

```
##        M          So          Ed              Po1              Po2
##  Min.   :11.9   0:31   Min.   : 8.70   Min.   : 4.50   Min.   : 4.10
##  1st Qu.:13.0   1:16   1st Qu.: 9.75   1st Qu.: 6.25   1st Qu.: 5.85
##  Median :13.6          Median :10.80   Median : 7.80   Median : 7.30
##  Mean   :13.9          Mean   :10.56   Mean   : 8.50   Mean   : 8.02
##  3rd Qu.:14.6          3rd Qu.:11.45   3rd Qu.:10.45   3rd Qu.: 9.70
##  Max.   :17.7          Max.   :12.20   Max.   :16.60   Max.   :15.70
##        LF             M.F              Pop              NW
##  Min.   :0.480   Min.   : 93.4   Min.   :  3.0   Min.   : 0.2
##  1st Qu.:0.530   1st Qu.: 96.5   1st Qu.: 10.0   1st Qu.: 2.4
##  Median :0.560   Median : 97.7   Median : 25.0   Median : 7.6
##  Mean   :0.561   Mean   : 98.3   Mean   : 36.6   Mean   :10.1
##  3rd Qu.:0.593   3rd Qu.: 99.2   3rd Qu.: 41.5   3rd Qu.:13.2
##  Max.   :0.641   Max.   :107.1   Max.   :168.0   Max.   :42.3
##        U1               U2            Wealth          Ineq           Prob
##  Min.   :0.0700   Min.   :2.00   Min.   :2880   Min.   :12.6   Min.   :0.0069
##  1st Qu.:0.0805   1st Qu.:2.75   1st Qu.:4595   1st Qu.:16.6   1st Qu.:0.0327
##  Median :0.0920   Median :3.40   Median :5370   Median :17.6   Median :0.0421
##  Mean   :0.0955   Mean   :3.40   Mean   :5254   Mean   :19.4   Mean   :0.0471
##  3rd Qu.:0.1040   3rd Qu.:3.85   3rd Qu.:5915   3rd Qu.:22.8   3rd Qu.:0.0544
##  Max.   :0.1420   Max.   :5.80   Max.   :6890   Max.   :27.6   Max.   :0.1198
##       Time          Crime
##  Min.   :12.2   Min.   : 342
##  1st Qu.:21.6   1st Qu.: 658
##  Median :25.8   Median : 831
##  Mean   :26.6   Mean   : 905
##  3rd Qu.:30.4   3rd Qu.:1058
##  Max.   :44.0   Max.   :1993
```

There is no median now for binary So now, so we can continue with the analysis.

Step 2: Grow random trees

Let's start by growing a default RF with default setting and then improving it.

I will use the caret package, since it is more powerful and gives lots of options for tuning random forests. We will start with default parameters and then try to compare different tuning options to find the best model. Each time within the caret package we will **cross-validate the model** using LOOCV leave-one-out cross-validation (due to a small data set it will provide more realistic results than k-fold, as it gets more points for the training set).

For each model we will make predictions using the built model (y-hat values), plot predictions against actual Crime values, calculate Residual Sum of the Squares SSres, Total Sum of the Squares SStotal, R squared, and then recalculate everything described above to compare the quality of the model with quality on training set.

Default RF model

Let's start with basic settings. We use LOOCV for cross-validation, RMSE as a model metric, since we have a regression problem (for classifcation we would've used Accuracy). We set mtry, number of variables randomly used at each split, as a default value of 5, based on the formula p/3, where p is the number of predictors (15). We use this mtry value as our tune grid parameter for the model (setting which values the main parameter will take).

```
set.seed(1)
ctrl <- trainControl(method='LOOCV')

#RMSE quality metric since its a regression, not classification
metric <- 'RMSE'

#p/3 for regression models
mtry <- (ncol(data)/3)

tgrd <- expand.grid(.mtry=mtry)

rf_basic <- train(Crime~.,
                    data=data,
                    method='rf',
                    metric=metric,
                    tuneGrid=tgrd,
                    trControl=ctrl,
                    #verbose to get more text description on the model
                    verbose=T)

print(rf_basic)
```

```
## Random Forest
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results:
##
##   RMSE  Rsquared  MAE
##   287   0.4549    206.7
##
## Tuning parameter 'mtry' was held constant at a value of 5.333
```

```
rf_basic$finalModel
```

```
##
```

```
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, verbose = ..1)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 86439
##                     % Var explained: 40.96
```

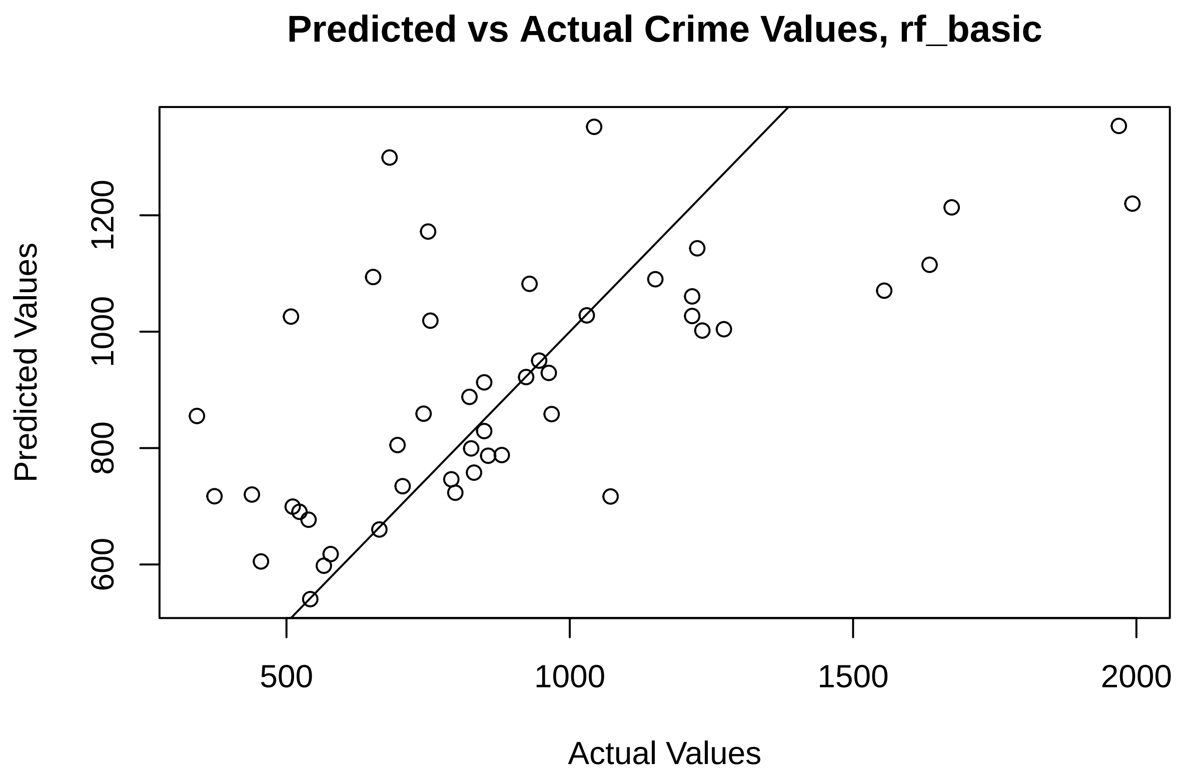
```
varImp(rf_basic)
```

```
## rf variable importance
##
##        Overall
## Po1     100.00
## Po2      92.63
## Prob     61.47
## Wealth   49.78
## NW       45.79
## Pop      28.27
## M.F      25.70
## LF       23.30
## Ineq     16.41
## Ed       15.96
## Time     14.98
## M        14.69
## U2        9.01
## U1        8.07
## So1       0.00
```

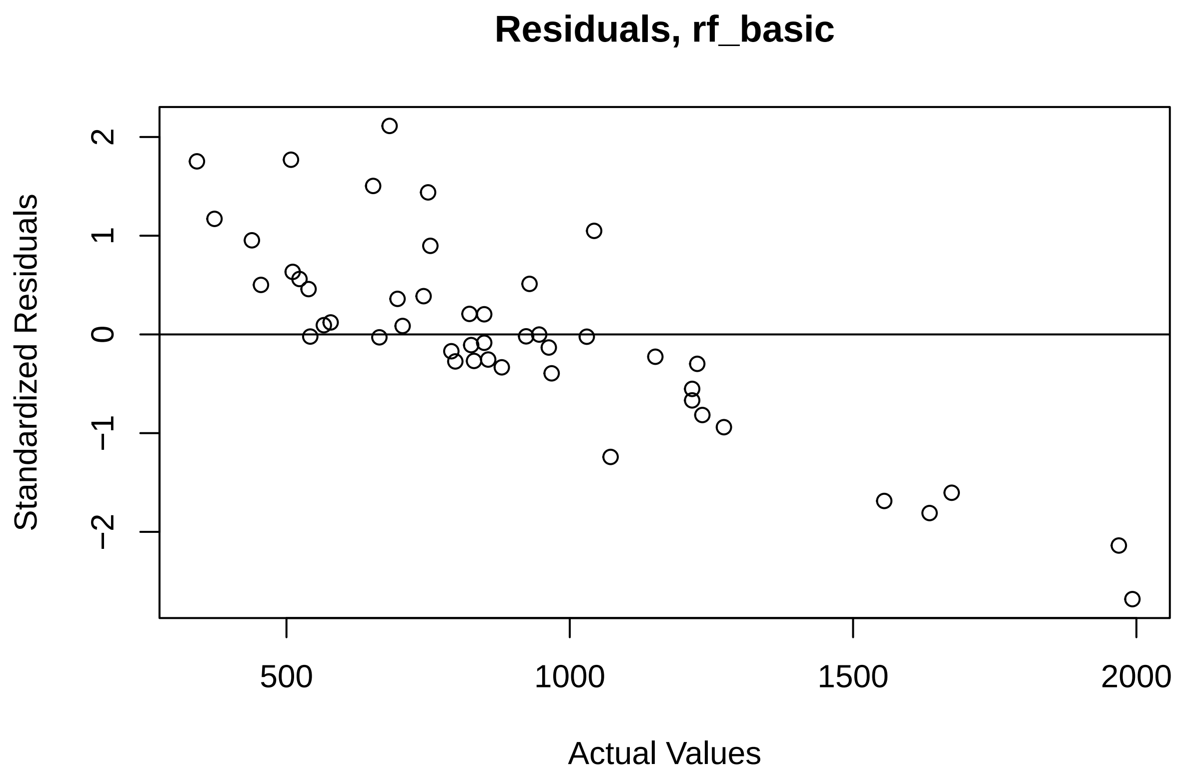
Cross-validated Random Forest with 5 variables used at each split has an R2 of 45.5%, RMSE of 287 and MAE of 207; 40.96% of variance is is explained with the model. This is a better result compared to regression trees, but we will still try to tune the random forest to improve them. As expected, the most important variable used is Po1 - police protection expenditure in 1960, just as in previous models, followed by expenditure in 1959, probability of imprisonment and wealth.

Let's make some visualizations of predictions, residuals, variable importance:

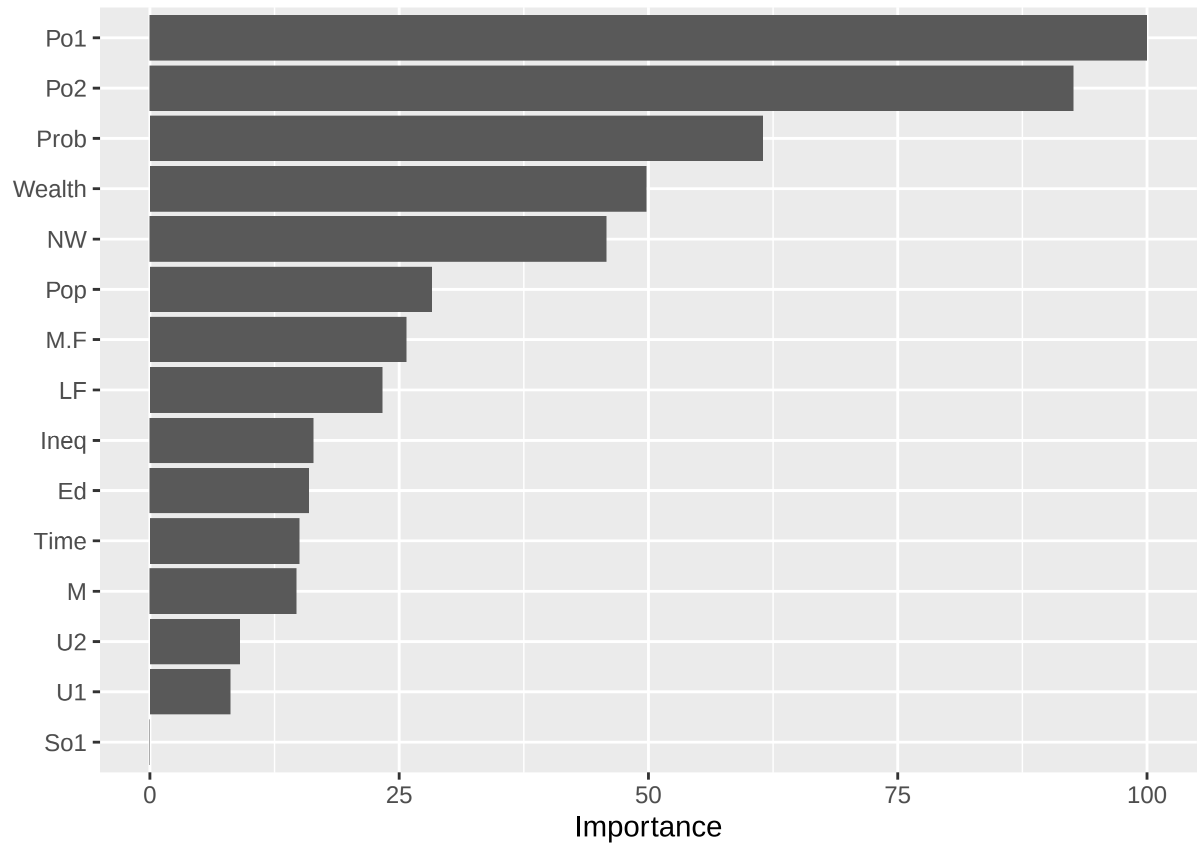
```
#Residual sum of squares:
yhat_rf_basic <- rf_basic$pred[,1]
ssres <- sum((yhat_rf_basic-data$Crime)^2)

#Total sum of squares:
sstot <- sum((data$Crime - mean(data$Crime))^2)

#Plot predicted vs actual:
plot(data$Crime, yhat_rf_basic, xlab='Actual Values', ylab='Predicted Values', main='Predicted vs Actua
abline(0,1)
```



```
#Plot residuals
plot(data$Crime, scale(yhat_rf_basic - data$Crime), xlab='Actual Values', ylab='Standardized Residuals'
abline(0,0)
```



```
#Plot Variable improtance
vip::vip(rf_basic, num_features = 15)
```



As we can see, although many predictions and residuals are close to the actual values, some are quite far from the line, especially for the higher crime rates. As for importance, the top 5 predictors are police protection expenditures (both), probability of going to prison for a crime, wealth, non-whites share.

Let's try different tuning options to see if we can improve the model

RF model with Random Search

Here we are experimenting with the mtry parameter, number of variables randomly used at each split, but instead of setting it manually, we will let the model randomly choose its values

```
set.seed(11)
control <- trainControl(method='LOOCV',
                        search='random')

#grow 500 trees
ntree <- 500

rf_randsearch <- train(Crime ~ .,
                   data = data,
                   method = 'rf',
                   metric = metric,
                   tuneLength  = 15,
                   trControl = control,
                   verbose=T)
print(rf_randsearch)
```

```
## Random Forest
```

```
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE   Rsquared  MAE
##    1    305.7  0.4535    228.4
##    3    286.3  0.4735    208.5
##    5    287.1  0.4522    205.0
##    6    287.9  0.4456    207.5
##    7    290.1  0.4330    208.2
##    8    292.8  0.4207    208.6
##    9    294.1  0.4136    209.4
##   12    299.0  0.3926    213.4
##   14    298.8  0.3919    214.9
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 3.
```

```
rf_randsearch$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, verbose = ..1)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 83302
##                     % Var explained: 43.1
```

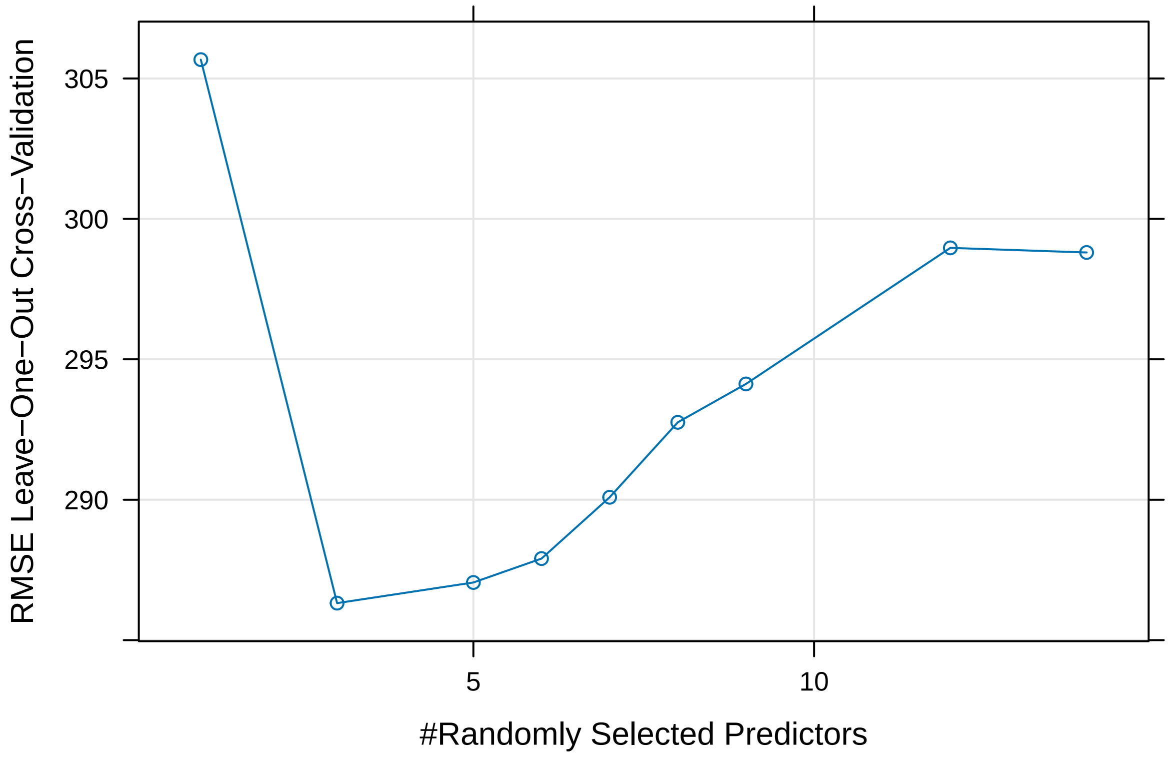
```
varImp(rf_randsearch)
```

```
## rf variable importance
##
##        Overall
## Po2      100.0
## Po1       93.7
## Prob      72.7
## Wealth    55.5
## NW        50.6
## Pop       42.1
## LF        34.5
## M.F       30.7
## Ed        30.3
## Ineq      24.9
## U2        22.1
## Time      21.0
## M         19.3
## U1        15.1
## So1        0.0
```

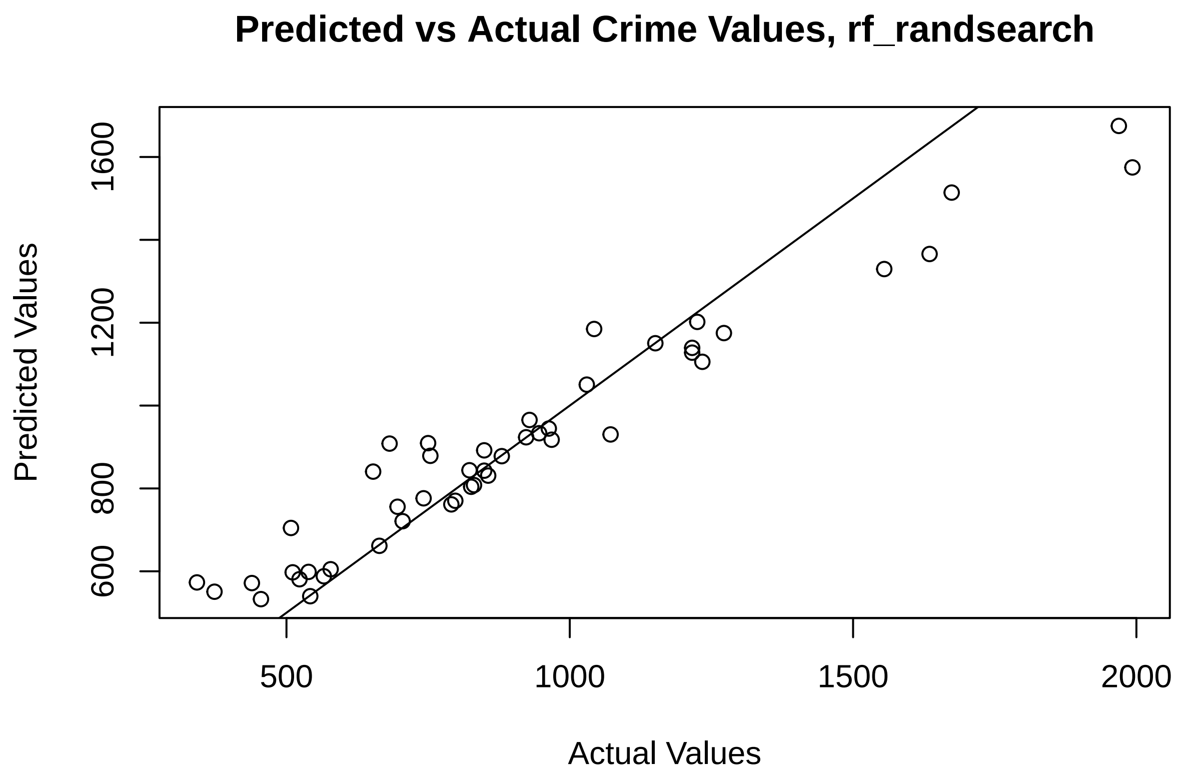
With several mtry values generated, the best model now appears to be **with mtry=3**. It has **R2 of 47% after LOO Cross-validation, MAE of 208 and explains 43% of variance (+3% to mtry=5 model).**

Visualize the results:

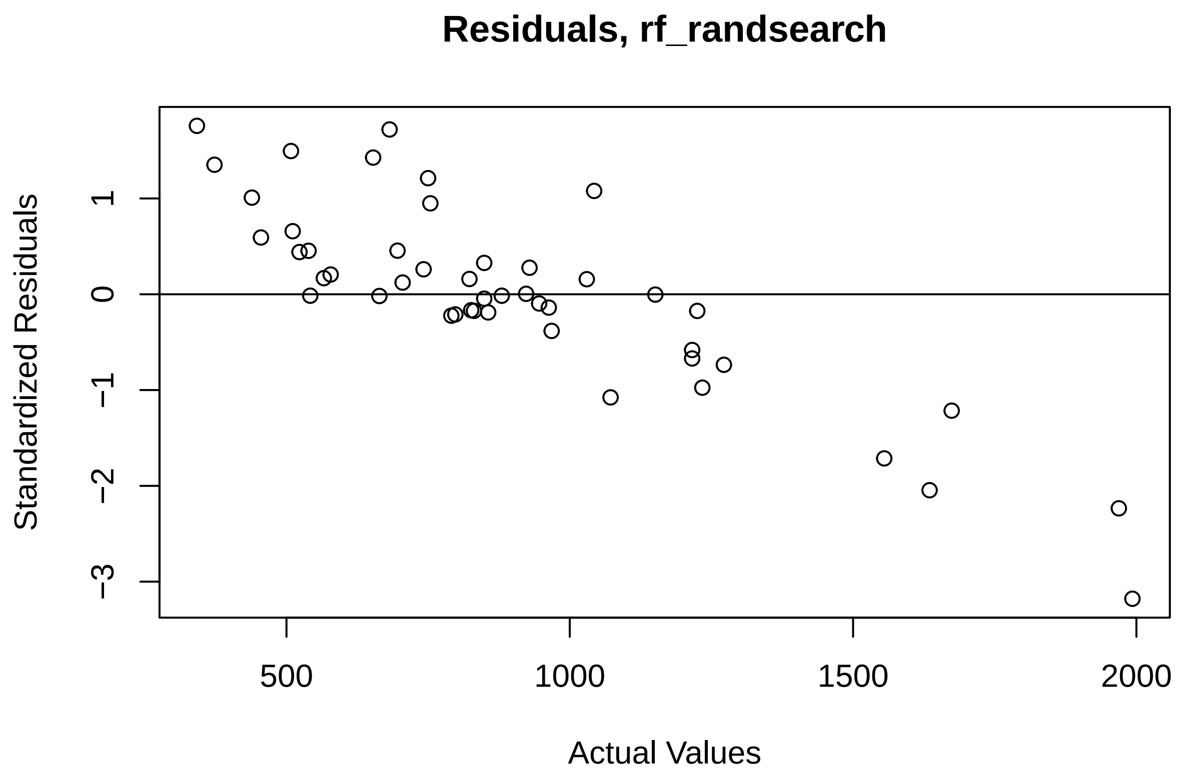
```
set.seed(1)
#Residual sum of squares:
yhat_rf_randsearch <- predict(rf_randsearch)
ssres <- sum((yhat_rf_randsearch-data$Crime)^2)

#plot the model
plot(rf_randsearch)
```



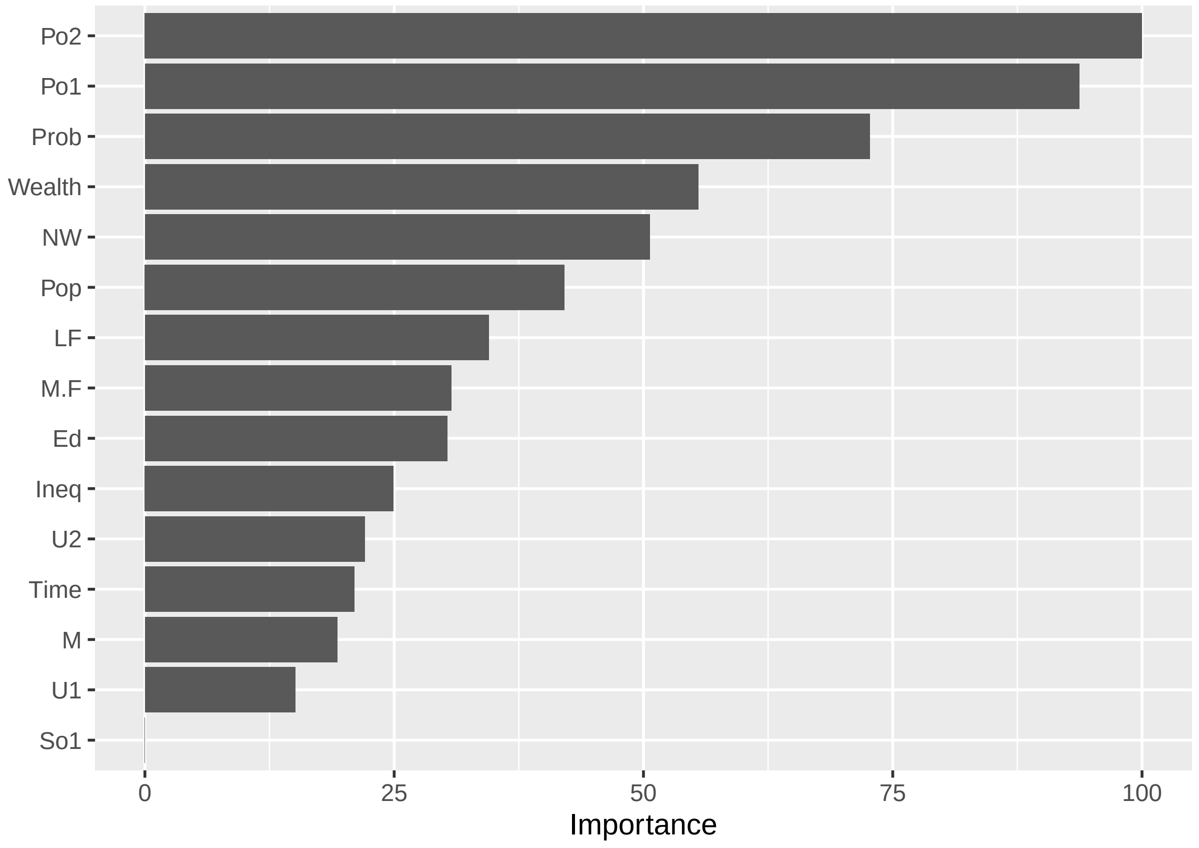
```
#Plot predicted vs actual:
plot(data$Crime, yhat_rf_randsearch, xlab='Actual Values', ylab='Predicted Values', main='Predicted vs 
abline(0,1)
```



```
#Plot residuals
plot(data$Crime, scale(yhat_rf_randsearch - data$Crime), xlab='Actual Values', ylab='Standardized Residu
abline(0,0)
```



```r
#Plot Variable improtance
vip::vip(rf_randsearch, num_features = 15)
```



The first plot shows how with mtry=3 we get the lowest RMSE, and lower compared to mtry=5 as in our previous model. We can also see the improvement in the model on the plots - predictions are a lot closer to reality, and the range of residuals on the upper side has reduced by 1 stdev. The top 5 importance predictors are the same with the previous model, but now Po2 is the most important, not Po1, and the general importance of the first few predictors is a bit higher - for example, the 5th important predictor NW had a 45.7 overall importance in the default model, whereas now it is 50.6.

This model could be chosen as the best one, but I will still do some more tuning to see if it can be better.

RF model with Grid Search

In this model, instead of a random search (=choice of mtry), we will use a tune grid to try all the possible mtry values and see how model accuracy changes with increasing mtry:

```r
set.seed(1)
control <- trainControl(method='LOOCV',
                        search='grid')

tunegrid <- expand.grid(.mtry=(1:15))

rf_grid <- train(Crime ~ .,
                 data = data,
                 method = 'rf',
                 metric = metric,
                 tuneGrid = tunegrid,
                 verbose=T)
print(rf_grid)
```

```
## Random Forest
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 47, 47, 47, 47, 47, 47, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE   Rsquared  MAE
##    1    294.7  0.5094    228.0
##    2    281.7  0.5251    216.8
##    3    278.7  0.5165    214.3
##    4    277.3  0.5156    211.8
##    5    279.0  0.5016    213.0
##    6    280.3  0.4912    214.2
##    7    280.0  0.4900    213.6
##    8    282.6  0.4791    215.2
##    9    285.7  0.4676    218.3
##   10    287.8  0.4599    218.5
##   11    287.8  0.4621    219.2
##   12    290.3  0.4492    220.9
##   13    290.9  0.4480    221.6
##   14    291.4  0.4442    221.4
##   15    292.7  0.4421    222.5
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 4.
```

```r
rf_grid$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, verbose = ..1)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 82417
##                     % Var explained: 43.71
```

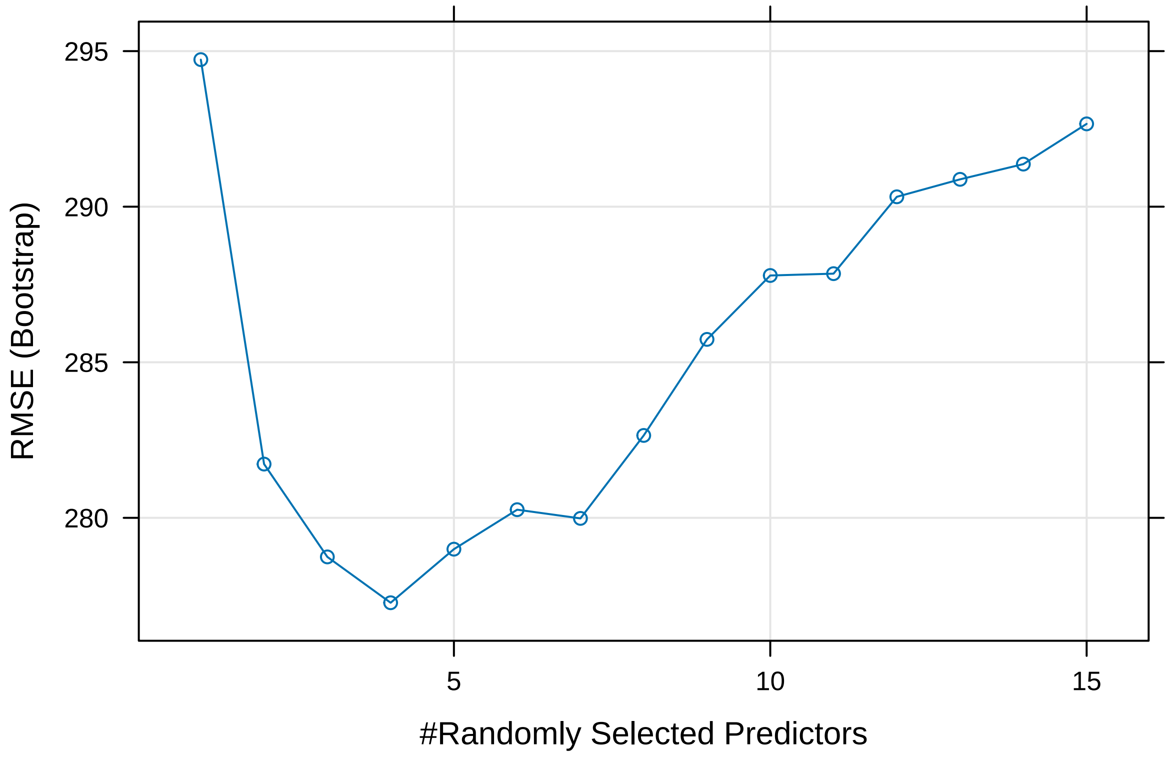
```r
varImp(rf_grid)
```

```
## rf variable importance
##
##        Overall
## Po1      100.0
## Po2       96.3
## Prob      69.1
## Wealth    64.3
## NW        46.0
## Pop       31.2
## LF        27.1
## M.F       26.4
## Ed        23.6
## Ineq      18.5
## Time      17.3
## M         16.9
## U2        14.4
## U1        11.0
## So1        0.0
```

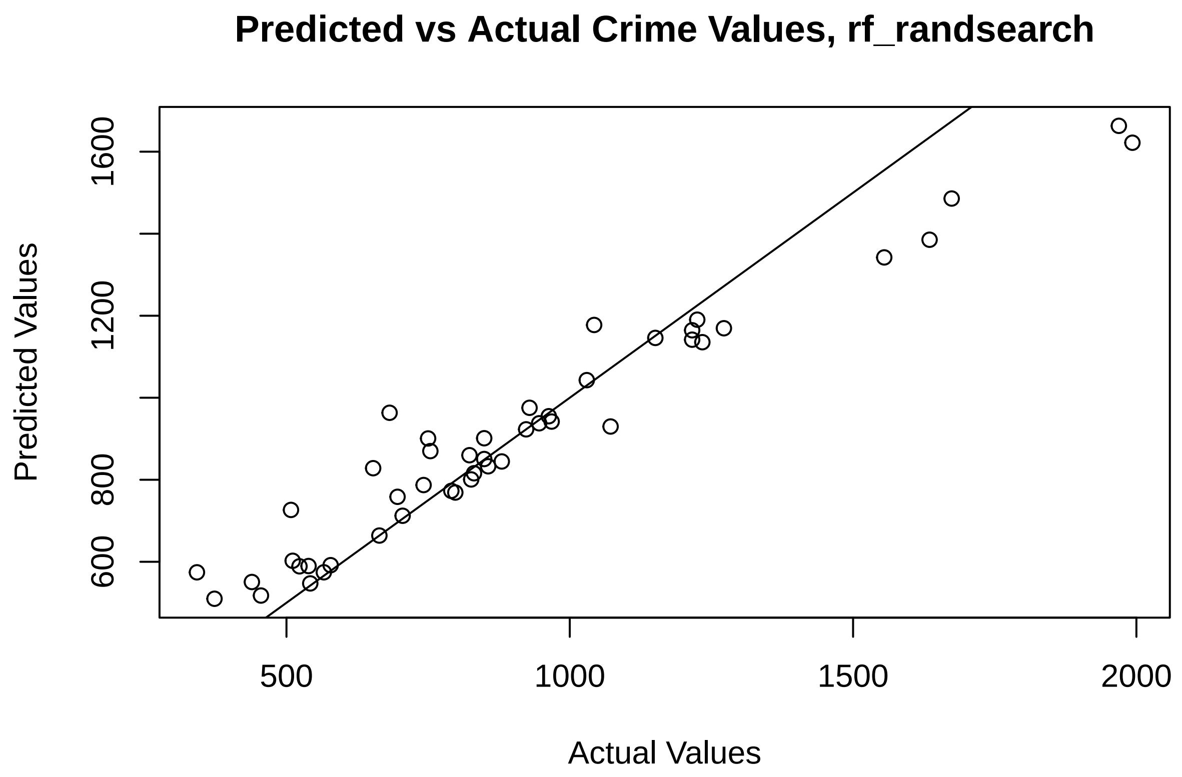
This time, the cross-validated model shows a better result once again, with **R2 of 51.6% achieved with mtry=4, with MAE of 211.8 and 43.7% of variance explained**. So far, this is the best model we've seen, and a 4% improvement in R2 is a good one, although it comes with a slight increase in MAE as well (211 instead of 208 with random search).

Check the plots:

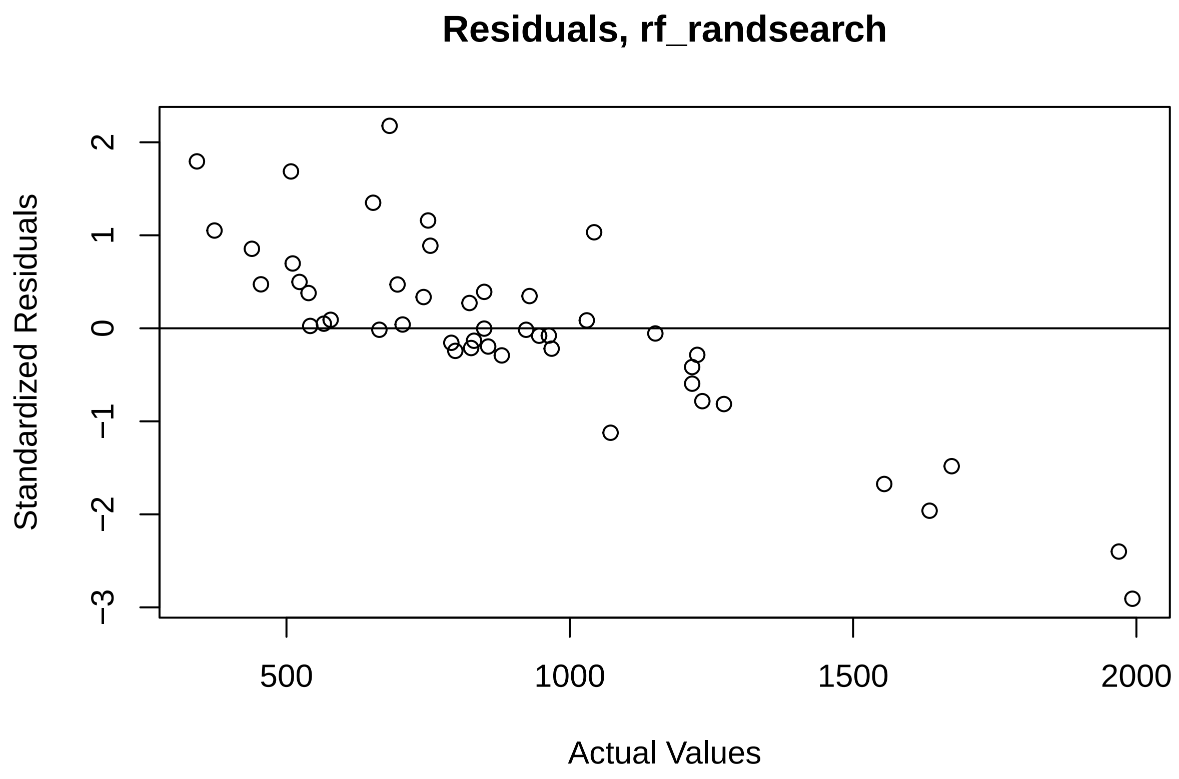
```
set.seed(1)
#Residual sum of squares:
yhat_rf_grid <- predict(rf_grid)
ssres <- sum((yhat_rf_grid-data$Crime)^2)

#plot the model
plot(rf_grid)
```



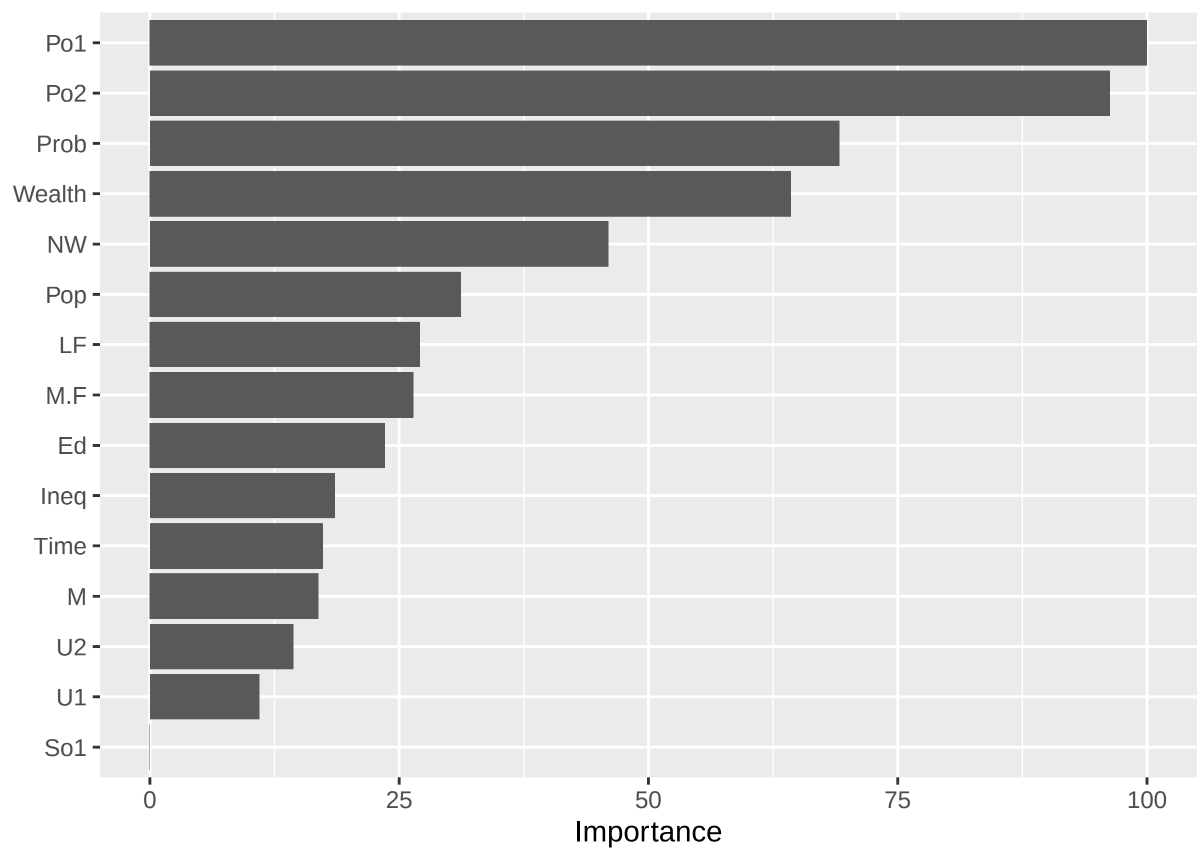
```
#Plot predicted vs actual:
plot(data$Crime, yhat_rf_grid, xlab='Actual Values', ylab='Predicted Values', main='Predicted vs Actual
abline(0,1)
```



```
#Plot residuals
plot(data$Crime, scale(yhat_rf_grid - data$Crime), xlab='Actual Values', ylab='Standardized Residuals',
abline(0,0)
```



```
#Plot Variable improtance
vip::vip(rf_grid, num_features = 15)
```



On the plot of the grid search model we can see how the lowest RMSE is achieved with 3-7 used variable at each split, and the lowest error is with mtry=4. Predicted values and residuals show the same situation as random search model, where many values are close to the line, but the prediction for states with higher crime is less accurate. As for variable importance, the top-5 is again the same, and Po1 is the most important one, as in the same model. The difference we can notice is that the second half of predictors seems to have reduced importance compare to two previous model, so only top 4 predictors have importance of 50+%, the next 4 are withing 25-50%, and the last 7 have importance below 25%. So the prediction is mostly (50+%) made with trees using variables such as police protection expenditure, probability of imprisonment and wealth.

Bagged RF

We will also try Bagging, which is a random forest model where mtry=15, all variables. This method considers all features when splitting the node and uses bootstrap replications. Bagging is powerful for variance reduction and overfitting prevention, so perhaps this model would perform well on our data set. I will use LOO cross-validation and 'treebag' method in caret:

```
set.seed(1)
ctrl <- trainControl(method='LOOCV')

bag <- train(
  Crime~.,
  data=data,
  method='treebag',
  trControl=ctrl,
  metric='RMSE', importance=T
)
print(bag)
```

```
## Bagged CART
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results:
##
##   RMSE   Rsquared  MAE
##   316.2  0.3205    225.2
```

```
bag$finalModel
```

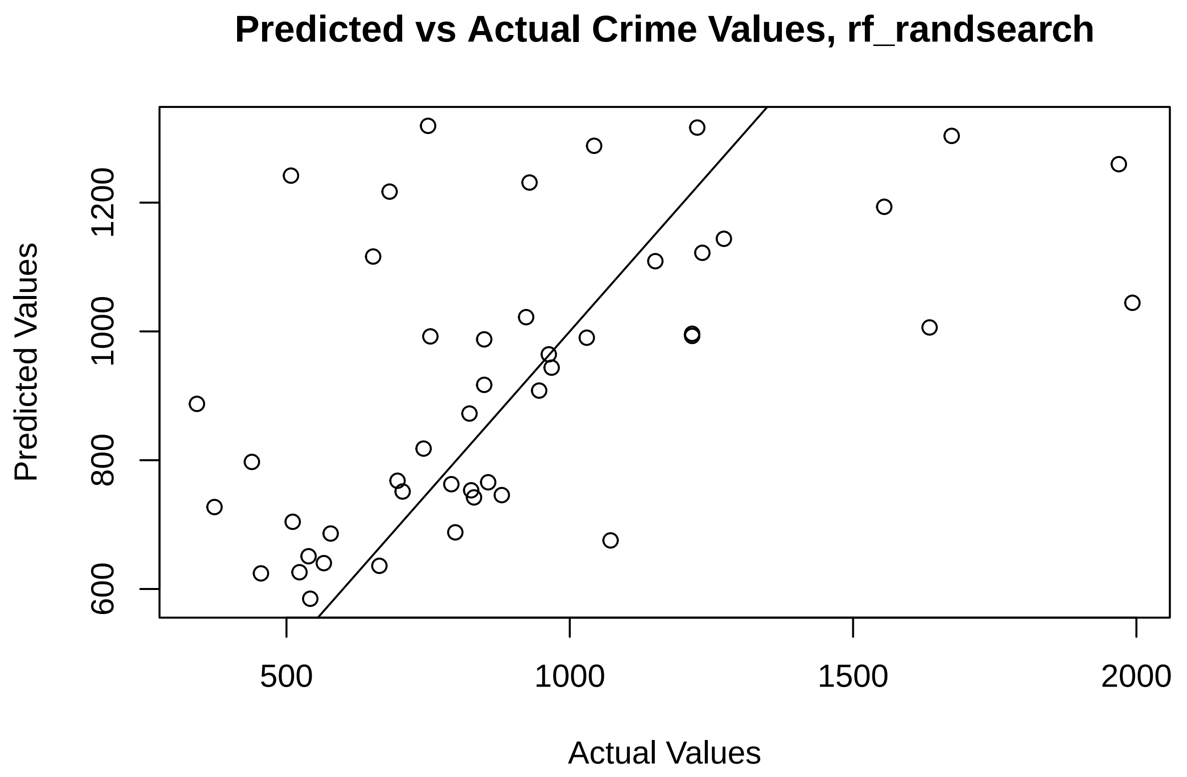
```
##
## Bagging regression trees with 25 bootstrap replications
```

```
varImp(bag)
```

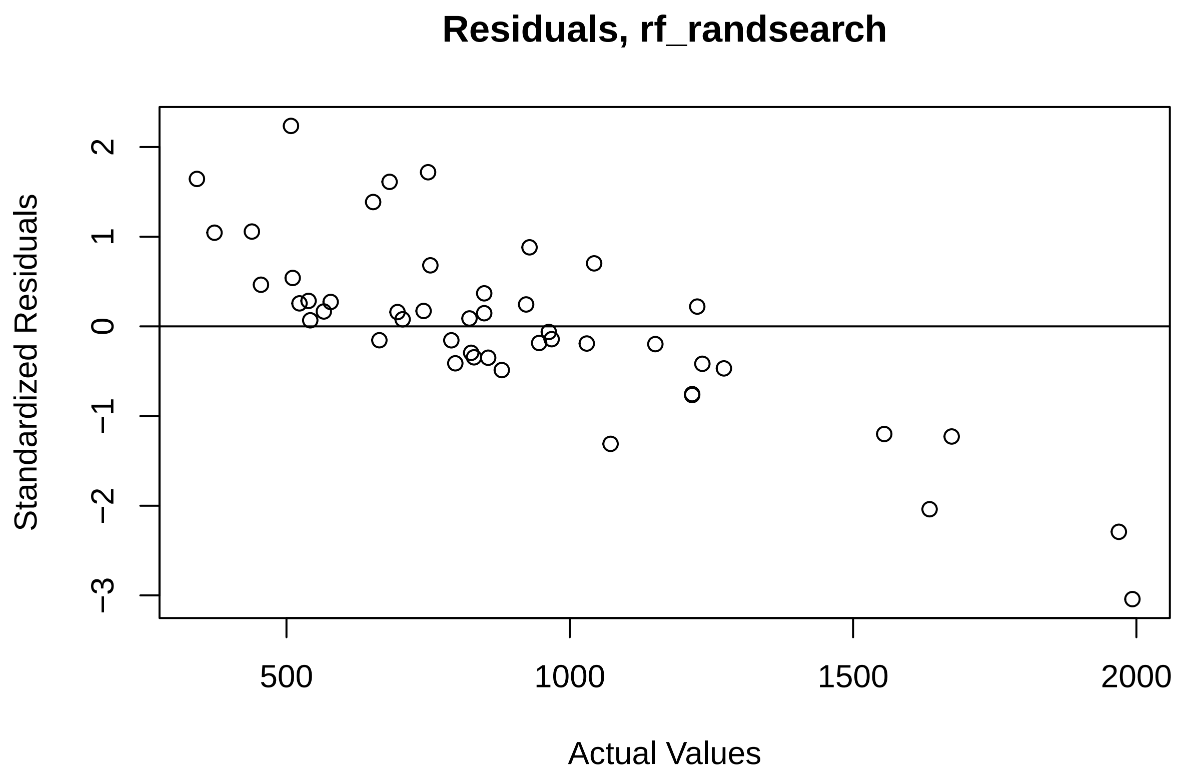
```
## treebag variable importance
##
##        Overall
## Po1     100.00
## Po2      98.67
## NW       73.43
## Prob     72.35
## Pop      43.86
## Wealth   41.69
## LF       28.87
## Ed       23.64
## Ineq     20.13
## M        20.09
## Time     10.08
## So1       9.09
## U2        5.23
## M.F       5.18
## U1        0.00
```

Unfortunately baggind does not seem to go well with our data, as it provides R2 of 32% after cross-validation.

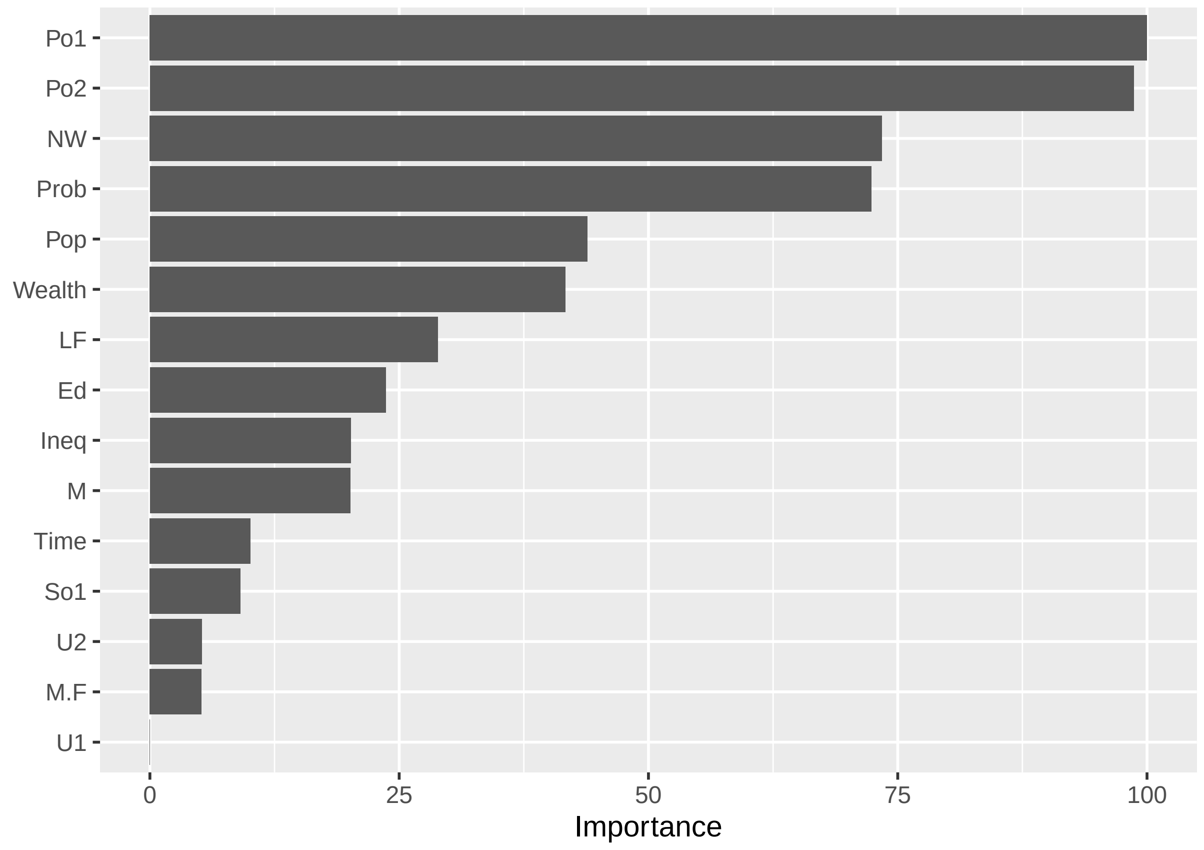
```
#Residual sum of squares:
yhat_bag <- bag$pred[,1]
ssres <- sum((yhat_bag-data$Crime)^2)


#Plot predicted vs actual:
plot(data$Crime, yhat_bag, xlab='Actual Values', ylab='Predicted Values', main='Predicted vs Actual Cri
abline(0,1)
```



```
#Plot residuals
plot(data$Crime, scale(yhat_bag - data$Crime), xlab='Actual Values', ylab='Standardized Residuals', mai
abline(0,0)
```



```
#Plot Variable improtance
vip::vip(bag, num_features = 15)
```



From the plots we can tell that although the first 2 predictors are the same, the order of the following ones by importance is a bit different - for example wealth which was the 5th important one is replaced here with population. The predictions are far from being accurate, so we will not use this model

Choosing the final model

Based on all the plots and metric we have analysed, the best model is the one with mtry=4 (4 random variables used at each split), which we got using the grid search parameter:

```
rf_grid$results[4,]
```

```
##   mtry  RMSE Rsquared   MAE RMSESD RsquaredSD MAESD
## 4    4 277.3   0.5156 211.8  53.77      0.152  47.9
```

R2 for our model is 51.6%, which is a better result compared to our previous models on the set. The model is also cross-validated using leave-one-out cross-validation, which helped us reduce overfitting .

Now we can try and apply the random forest model on the two leaves we have from Part 1.A

Step 3: Use chosen model on 2 leaves from Part 1.A

Now we can estimate model's quality on each leaf. I will use the split data set based on Po1 < 7.65 yes/no condition and calculate R2 that the model gives for each split.

```
#split based on Po1
leaves<-split(data, data$Po1<7.65)
#df for each leaf
#leaf 1 - TRUE vals from split
df1<- as.data.frame(leaves[2])
```

```r
#leaf 2 - FALSE vals from split
df2<- as.data.frame(leaves[1])


#yhat from the model for each leaf
#leaf 1 prediction
yhat1<-yhat_rf_grid[data$Po1<7.65]
#leaf 2 prediction
yhat2<-yhat_rf_grid[data$Po1>=7.65]

#ssres and sstot for each leaf
#leaf 1
ssres1 <- sum((yhat1-df1[,16])^2)
sstot1 <- sum((df1[,16]-mean(df1[,16]))^2)
#leaf 2
ssres2 <- sum((yhat2-df2[,16])^2)
sstot2 <- sum((df2[,16]-mean(df2[,16]))^2)

#R2 for each leaf
#leaf 1
r2_leaf1 <- 1 - ssres1/sstot1
r2_leaf1
```

```
## [1] 0.8254
```

```r
#leaf 2
r2_leaf2 <- 1 - ssres2/sstot2
r2_leaf2
```

```
## [1] 0.8275
```

We can see that our cross-validated model performs well on both leaves with R2 values over 80%. The result is much better than with linear regression on each leaf, however we sacrifice explainability, since random forest cannot be explained easily with words.

Step 4: Conclusion

Having compared different random forest models with different tuning parameters, we chose the model with 4 random variables being used at each split, as it showed the lowest Root-mean-square error RMSE and highest R-squared value. We made predictions using that model and saw on the graph that they have good accuracy, the only problem-causing points are states with high Crime rate - there aren't many of those, and considering that we only have 47 records in the set, this is acceptable.

Based on the variable importance, where the top 5 predictors are Po1 (police protection expenditure in 1960), Po2 (expenditure in 1959), Prob (probability of imprisonment), Wealth (household wealth), NW (share of non-whites in population), we can say that the highest increment in tree leaves' purity, meaning better quality of prediction and better quality of splits, is due to police protection expenditure variable, which is no surprise, since we saw this variable as the primary one for use in regression trees. Other features with high importance also improve the quality of the splits in the random forest model. So, for example, in our model better Crime predictions are made using first of all police protection expenditure values for the state, then probability of imprisonment (so high probability of punishment has strong relation with crime amount in a state), then wealth (so how much money household has in the state on average is also related to crime), and the amount of non-whites in population. In the future information on feature importance could be used for feature selection - perhaps randomly selecting variables for each split from a narrowed predictor set would result in higher accuracy.

We also tried to fit the random forest model on each leaves and saw a good R2 value, suggesting that we might want to combine those methods and use random forest instead of regression on each of 2 leaves of regression tree, if we are willing to sacrifice explainability. After all, if we could see the coefficients for each variable in linear regression and the exact tree in regression tree, with Random Forest there is not much to explain visually or verbally, as we only know quality metrics and feature importance.

10.2 Logistic Regression Example

I work in a food delivery company - we deliver from restaurants, supermarkets, pharmacies and many other shops. On of the possible uses of logistic regression for us would be in **predicting customer churn**.

Data needed: - we can use data describing user behaviour in the app, no extra survey would be necessary, as we always monitor these features.

Target feature - whether the customer churned or not - a binary response variable (0/1, as in Yes/No)

Predictors:

- Number of days the user has been on the app (tenure)
- Whether the customer is a subscriber or not. We have a ~$4 monthly 'Premium' subscription, with which users can get special deals only for premium customers and have a fixed, lower than average delivery fee (so that lunch hour surge would not affect delivery cost for them). This is a binary variable
- Delivery zone, where the customer is located. Customers can set their address in the app, which falls into one of our delivery zones, and defines what restaurants/shops they will see as available. This is a categorical variable, since it is an identifier of a zone.
- Share of orders the customer has reviewed. After each order there is a chance to review the order, and some customers can be more involved in this process than others.
- Gender, which customers also specify in the app

There are other interesting predictors that we can use. For example, if we take order reviews, there are several categories a customer can rate - how satisfied they are with delivery time, order packaging, delivery person politeness, app navigation, etc. Average of each of these averages for a customer can be also used as predictors (for example, we can check a theory whether those with lower average review features on the last few orders are likely to churn).

The app has about 10 millions of users in total, and about 5 million are active in a 3-month period, each making about 6 orders throughout 3 months. So, there would be lots of user data which can be used for the model (of course, some NA values will require treatment).

Part 2.A Logistic Regression

Step 1: Load the data

```r
data <- read.table("germancredit.txt",
                   header = FALSE,
                   stringsAsFactors = FALSE,
                   sep = "",
                   dec = ".")
head(data)
```

```
##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172   2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173   2
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173   2
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172   2
##    V19  V20 V21
## 1 A192 A201   1
## 2 A191 A201   2
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   2
## 6 A192 A201   1
```

Before exploring the data, I will transform all the attributes based on description - set numeric variables as numeric, categorical as factors. Also, since column names don't exist, I will try to rename them according to data description

Also, the response variable has values of 1 and 2, but we need 0 and 1 for the regression, so I will also convert the variables of Credit_Risk (V21) category:

```r
#1 and 2 in last column to 0 and 1
data$V21[data$V21==1] <- 0
data$V21[data$V21==2] <- 1

data <- transform(data,
                  V1=as.factor(data$V1),
                  V2=as.integer(data$V2),
                  V3=as.factor(data$V3),
                  V4=as.factor(data$V4),
                  V5=as.integer(data$V5),
                  V6=as.factor(data$V6),
                  V7=as.factor(data$V7),
                  V8=as.integer(data$V8),
                  V9=as.factor(data$V9),
                  V10=as.factor(data$V10),
                  V11=as.integer(data$V11),
                  V12=as.factor(data$V12),
                  V13=as.integer(data$V13),
                  V14=as.factor(data$V14),
                  V15=as.factor(data$V15),
                  V16=as.integer(data$V16),
                  V17=as.factor(data$V17),
                  V18=as.integer(data$V18),
                  V19=as.factor(data$V19),
                  V20=as.factor(data$V20),
                  #response is a factor! 0-good risk 1-bad
                  V21=as.factor(data$V21))

colnames(data) <- c('Checking_Account', 'Duration_months', 'Credit_History', 'Credit_Purpose', 'Credit_
summary(data)
```

```
##  Checking_Account Duration_months Credit_History Credit_Purpose Credit_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   Employment_Present Installment_to_Income Status_and_Sex
##  A61:603   A71: 62            Min.   :1.00          A91: 50
##  A62:103   A72:172            1st Qu.:2.00          A92:310
##  A63: 63   A73:339            Median :3.00          A93:548
##  A64: 48   A74:174            Mean   :2.97          A94: 92
##  A65:183   A75:253            3rd Qu.:4.00
##                               Max.   :4.00
##
##  Debtors_Guarantors Residence_Since Property        Age
##  A101:907           Min.   :1.00    A121:282   Min.   :19.0
##  A102: 41           1st Qu.:2.00    A122:232   1st Qu.:27.0
##  A103: 52           Median :3.00    A123:332   Median :33.0
##                     Mean   :2.85    A124:154   Mean   :35.5
##                     3rd Qu.:4.00               3rd Qu.:42.0
##                     Max.   :4.00               Max.   :75.0
##
##  Other_Installments Housing    Existing_Credits_at_Bank   Job
##  A141:139           A151:179   Min.   :1.00             A171: 22
##  A142: 47           A152:713   1st Qu.:1.00             A172:200
##  A143:814           A153:108   Median :1.00             A173:630
##                                Mean   :1.41             A174:148
##                                3rd Qu.:2.00
##                                Max.   :4.00
##
##  People_Liable  Telephone  Foreign_Worker Credit_Risk
##  Min.   :1.00   A191:596   A201:963       0:700
##  1st Qu.:1.00   A192:404   A202: 37       1:300
##  Median :1.00
##  Mean   :1.16
##  3rd Qu.:1.00
##  Max.   :2.00
##
```

```r
dim(data)
```

```
## [1] 1000   21
```

This looks much better. Now we can explore the data.

Step 2: Data Exploration

First, check if there are NA values in the set:

```r
is.null(data)
```

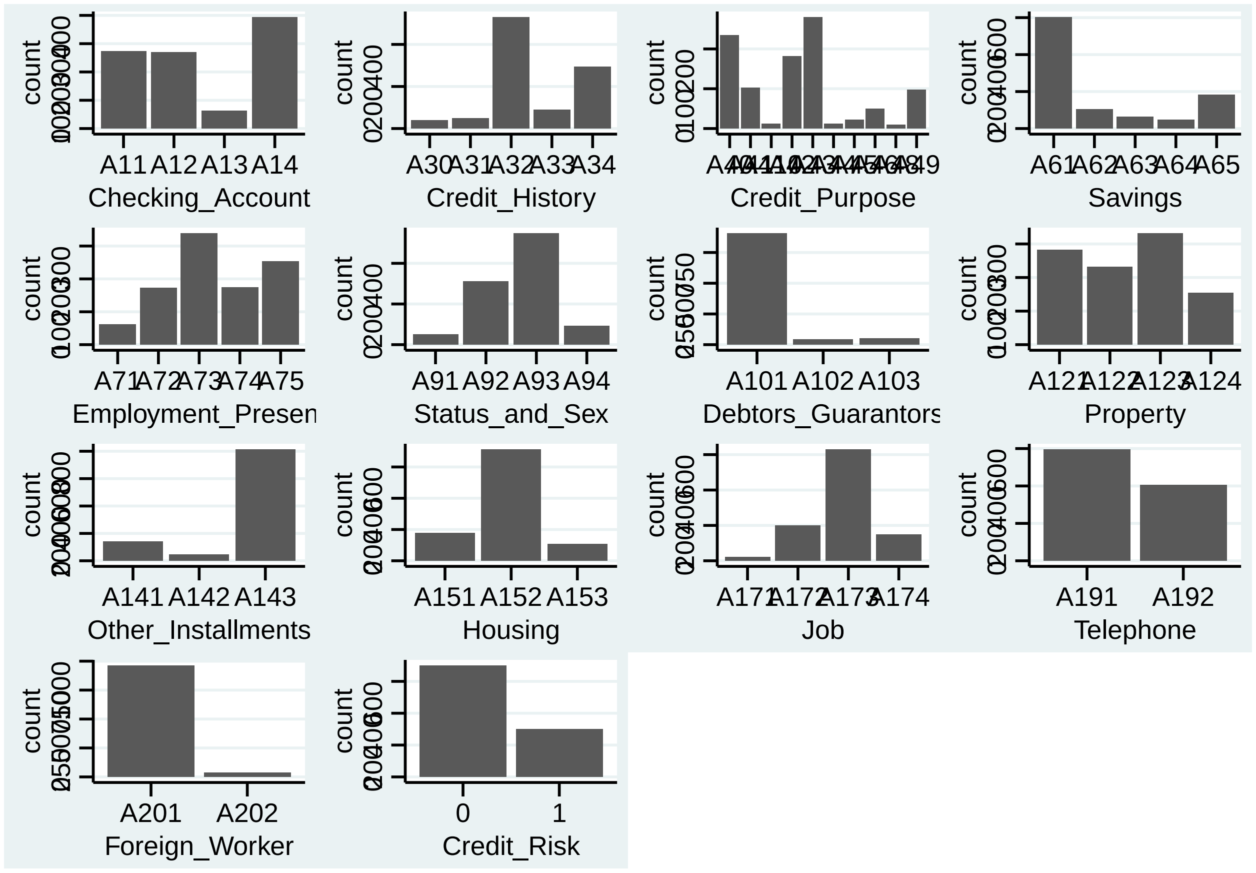
```
## [1] FALSE
```

No N/As.

Next, I'll plot all the variables. First, I plot barcharts for all the categorical variables:

```r
p1=ggplot(data, aes(x=Checking_Account))+geom_bar()+theme_stata()
p2=ggplot(data, aes(x=Credit_History))+geom_bar()+theme_stata()
p3=ggplot(data, aes(x=Credit_Purpose))+geom_bar()+theme_stata()
p4=ggplot(data, aes(x=Savings))+geom_bar()+theme_stata()
p5=ggplot(data, aes(x=Employment_Present))+geom_bar()+theme_stata()
p6=ggplot(data, aes(x=Status_and_Sex))+geom_bar()+theme_stata()
```

```
p7=ggplot(data, aes(x=Debtors_Guarantors))+geom_bar()+theme_stata()
p8=ggplot(data, aes(x=Property))+geom_bar()+theme_stata()
p9=ggplot(data, aes(x=Other_Installments))+geom_bar()+theme_stata()
p10=ggplot(data, aes(x=Housing))+geom_bar()+theme_stata()
p11=ggplot(data, aes(x=Job))+geom_bar()+theme_stata()
p12=ggplot(data, aes(x=Telephone))+geom_bar()+theme_stata()
p13=ggplot(data, aes(x=Foreign_Worker))+geom_bar()+theme_stata()
p14=ggplot(data, aes(x=Credit_Risk))+geom_bar()+theme_stata()

grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14)
```



Take a closer look at response variable - Credit Risk: there is imbalance in responses towards 0 - Good risks (70%).

```
t<-table(data$Credit_Risk)
t
```

```
##
##   0   1
## 700 300
```

```
round(t[1]/(t[1]+t[2])*100,2)
```

```
##  0
## 70
```

```
round(t[2]/(t[1]+t[2])*100,2)
```

```
##  1
## 30
```

There are more than twice as many 'good' applicants in terms of credit risk.

Next, let's check histogram plots for numeric variables:

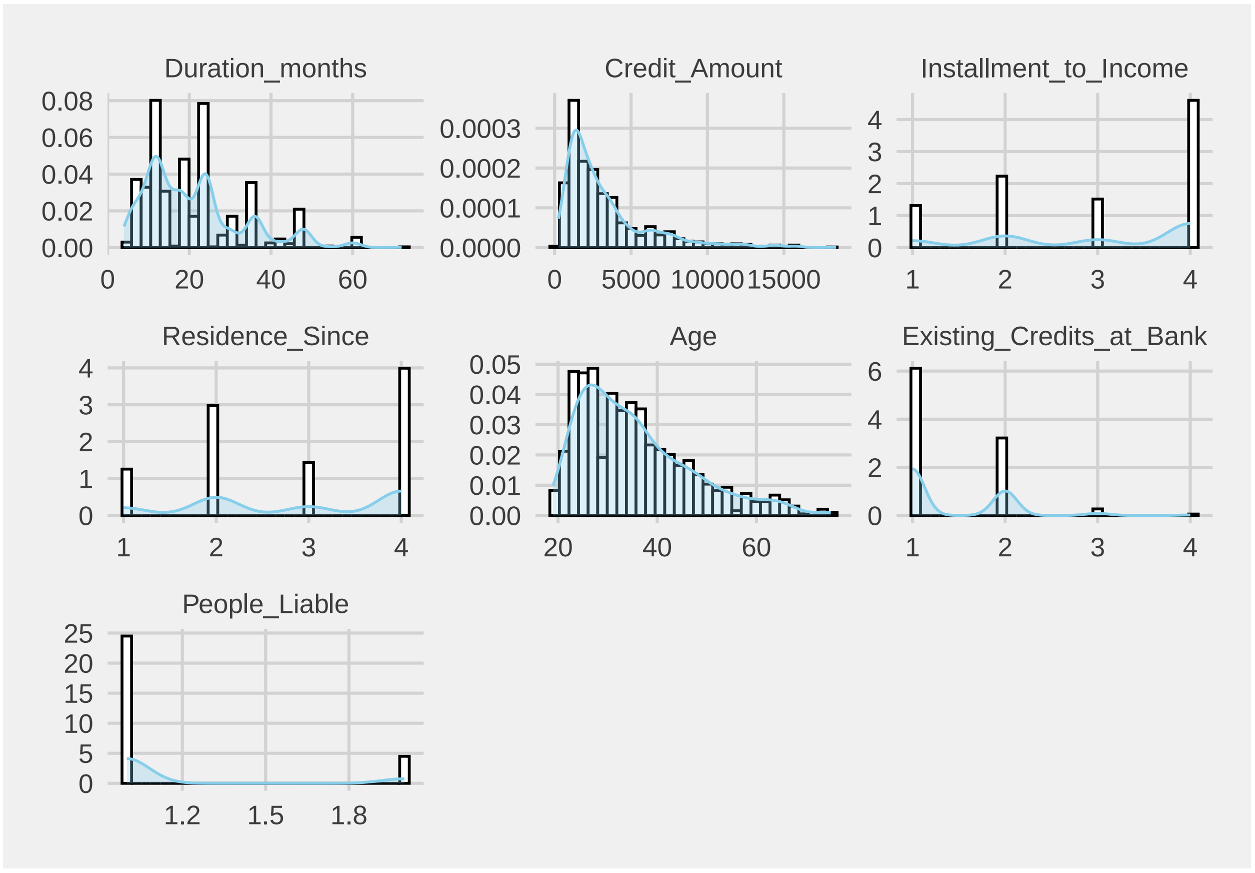
```
melted <- melt(data)[,15:16]
```

```
## Warning in melt(data): The melt generic in data.table has been passed a
## data.frame and will attempt to redirect to the relevant reshape2 method; please
## note that reshape2 is deprecated, and this redirection is now deprecated as
## well. To continue using melt methods from reshape2 while both libraries are
## attached, e.g. melt.list, you can prepend the namespace like
## reshape2::melt(data). In the next version, this warning will become an error.
```

```
## Using Checking_Account, Credit_History, Credit_Purpose, Savings, Employment_Present, Status_and_Sex,
```

```
hist_plots <- ggplot(melted,
                   aes(x=value))+
             geom_histogram(aes(y=..density..), colour="black", fill="white")+
             geom_density(alpha=.3, color="skyblue", fill="skyblue")+
             facet_wrap(~variable, scale="free")+
             theme_fivethirtyeight()
hist_plots
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Here, we can see that most variables are right-skewed. Duration of credit is usually under 40 months, it's amount is mostly under ~7000, and installment rates are more often as high as 4% of income; credit applicants are on the younger side by age, most of them are under 50 (working age?). Most people tend to have ~1-2 credits at bank, so 3-4 are more rare, and most of them also have ~1 dependent, less often 2.

Step 3: Split data into train/test sets

To train our model on one set and validate on a different one, I will split the data into training and test sets with 70/30 ratio:

```
set.seed(1)

#split
data_split <- initial_split(data, prop = .7, strata = "Credit_Risk")
data_train <- training(data_split)
data_test  <- testing(data_split)

#check proportions
t_train <- table(data_train$Credit_Risk)
round(t_train[1]/(t_train[1]+t_train[2])*100,2)
```

```
##     0
## 69.96
```

```
round(t_train[2]/(t_train[1]+t_train[2])*100,2)
```

```
##     1
## 30.04
```

```
t_test <- table(data_test$Credit_Risk)
round(t_test[1]/(t_test[1]+t_test[2])*100,2)
```

```
##    0
## 70.1
```

```
round(t_test[2]/(t_test[1]+t_test[2])*100,2)
```

```
##    1
## 29.9
```

The share of response variable withing the split is very close to the initial set, so we keep this split.

Step 4: Logistic Regression with all predictors

Confusion Matrix

To start, I will build logistic Regression on the training set using all the predictions. I will be using the caret package to also perform repeated cross-validation on the model with standard 10 folds and 10 repeats:

```
set.seed(1)

logreg1 <- train(Credit_Risk~.,
                   data=data_train,
                   method='glm',
                 #use logit since we have 0/1 as response - natural log link
                   family=binomial(link = "logit"),
                   trControl = trainControl(method='repeatedcv', number=10, repeats=10))

summary(logreg1)
```

```
##
```

```
## Call:
## NULL
##
## Coefficients:
##                            Estimate  Std. Error z value     Pr(>|z|)
## (Intercept)              -0.4662739   1.3065053   -0.36      0.72118
## Checking_AccountA12      -0.2033072   0.2649704   -0.77      0.44291
## Checking_AccountA13      -0.8580515   0.4315140   -1.99      0.04676 *
## Checking_AccountA14      -1.6776106   0.2889165   -5.81 0.0000000064 ***
## Duration_months           0.0293948   0.0116099    2.53      0.01135 *
## Credit_HistoryA31        -0.0917074   0.7005147   -0.13      0.89584
## Credit_HistoryA32        -0.5804996   0.5401056   -1.07      0.28247
## Credit_HistoryA33        -0.9460115   0.5911029   -1.60      0.10951
## Credit_HistoryA34        -1.3933785   0.5379551   -2.59      0.00959 **
## Credit_PurposeA41        -1.7643128   0.4574044   -3.86      0.00011 ***
## Credit_PurposeA410       -1.6494976   1.2042294   -1.37      0.17076
## Credit_PurposeA42        -0.9285309   0.3207350   -2.90      0.00379 **
## Credit_PurposeA43        -0.9546496   0.2981228   -3.20      0.00136 **
## Credit_PurposeA44         0.0586439   0.8268497    0.07      0.94346
## Credit_PurposeA45        -0.3939206   0.6861474   -0.57      0.56590
## Credit_PurposeA46         0.2758350   0.4803899    0.57      0.56584
## Credit_PurposeA48       -14.9555004 527.1633529   -0.03      0.97737
## Credit_PurposeA49        -0.9339561   0.4180338   -2.23      0.02547 *
## Credit_Amount             0.0001623   0.0000575    2.82      0.00479 **
## SavingsA62               -0.2336438   0.3419253   -0.68      0.49441
## SavingsA63                0.3249315   0.4237662    0.77      0.44322
## SavingsA64               -1.5014721   0.6149753   -2.44      0.01463 *
## SavingsA65               -0.7864661   0.3251580   -2.42      0.01558 *
## Employment_PresentA72    -0.6191504   0.5033714   -1.23      0.21869
## Employment_PresentA73    -0.5649724   0.4890025   -1.16      0.24794
## Employment_PresentA74    -1.2407425   0.5245215   -2.37      0.01801 *
## Employment_PresentA75    -0.5251832   0.4908864   -1.07      0.28468
## Installment_to_Income     0.4058162   0.1072831    3.78      0.00016 ***
## Status_and_SexA92        -0.0322119   0.4609995   -0.07      0.94429
## Status_and_SexA93        -0.7592802   0.4565406   -1.66      0.09629 .
## Status_and_SexA94        -0.0223771   0.5431685   -0.04      0.96714
## Debtors_GuarantorsA102    0.4079591   0.5130338    0.80      0.42650
## Debtors_GuarantorsA103   -0.8346407   0.4701371   -1.78      0.07585 .
## Residence_Since           0.0532894   0.1034605    0.52      0.60650
## PropertyA122              0.4952964   0.3032765    1.63      0.10244
## PropertyA123              0.3116775   0.2960705    1.05      0.29247
## PropertyA124              1.0738974   0.5453384    1.97      0.04893 *
## Age                      -0.0192729   0.0111858   -1.72      0.08489 .
## Other_InstallmentsA142    0.0686161   0.5374848    0.13      0.89842
## Other_InstallmentsA143   -0.9238187   0.2758119   -3.35      0.00081 ***
## HousingA152              -0.3689250   0.2851958   -1.29      0.19581
## HousingA153              -0.9474680   0.6020175   -1.57      0.11553
## Existing_Credits_at_Bank  0.2858097   0.2236786    1.28      0.20133
## JobA172                   1.0365025   0.9004829    1.15      0.24971
## JobA173                   0.9745664   0.8681392    1.12      0.26161
## JobA174                   0.6289279   0.8743842    0.72      0.47197
## People_Liable             0.5764097   0.3022389    1.91      0.05650 .
## TelephoneA192            -0.4834610   0.2476395   -1.95      0.05091 .
## Foreign_WorkerA202       -1.7289793   0.8563551   -2.02      0.04349 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 854.50  on 698  degrees of freedom
## Residual deviance: 619.41  on 650  degrees of freedom
## AIC: 717.4
##
## Number of Fisher Scoring iterations: 14
```

```
logreg1$results
```

```
##   parameter Accuracy Kappa AccuracySD KappaSD
## 1      none   0.7465 0.363    0.04552  0.1111
```

As we can see, not all the factors are significant. **Accuracy for the model with all factors is 74.6% and Kappa is 36.3%**. Kappa shows the prediction performance of variables - the bigger Kappa is, the more agreement there is between agreement. Our Kappa here falls into 'fair' agreement category. Accuracy standard deviation is 4.5%, which is not a bad value. **AIC is 717.41**, let's remember this value to see if it gets lower for the following models - this is a relative measure of model fit, and its lower value suggests that the model is better than the one with a higher AIC.

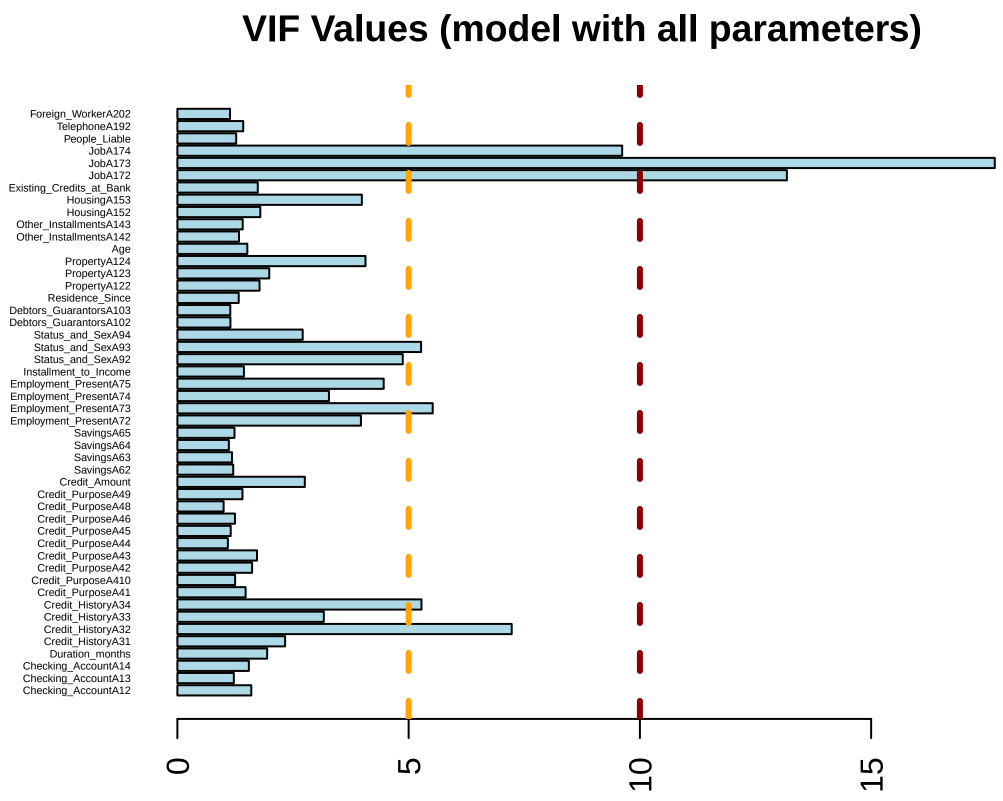
Multicollinearity Check

We can go ahead and leave only significant predictors in the model. However, I would also like to check the model for multicollinearity using the VIF values - variance inflation factor. It measure the effect which multicollinearity has on the model, and VIF >10 means that we have a serious multicollinearity problem. Sometimes high VIF value of one predictor causes other vifs to rise, and removing this variable might help make model's performance better. So I would like to try removing one of the highest VIF variables and checking how the model would change:

```
par(mar = c(3,8,3,3))
vif1<-car::vif(logreg1$finalModel)
as.table(vif1)
```

```
##      Checking_AccountA12      Checking_AccountA13      Checking_AccountA14
##                    1.598                    1.220                    1.545
##          Duration_months        Credit_HistoryA31        Credit_HistoryA32
##                    1.941                    2.329                    7.228
##        Credit_HistoryA33        Credit_HistoryA34        Credit_PurposeA41
##                    3.166                    5.278                    1.475
##       Credit_PurposeA410        Credit_PurposeA42        Credit_PurposeA43
##                    1.248                    1.615                    1.722
##        Credit_PurposeA44        Credit_PurposeA45        Credit_PurposeA46
##                    1.091                    1.153                    1.245
##        Credit_PurposeA48        Credit_PurposeA49            Credit_Amount
##                    1.000                    1.407                    2.755
##               SavingsA62               SavingsA63               SavingsA64
##                    1.207                    1.181                    1.113
##               SavingsA65    Employment_PresentA72    Employment_PresentA73
##                    1.234                    3.969                    5.520
##    Employment_PresentA74    Employment_PresentA75    Installment_to_Income
##                    3.277                    4.460                    1.439
##        Status_and_SexA92        Status_and_SexA93        Status_and_SexA94
##                    4.873                    5.269                    2.709
##    Debtors_GuarantorsA102   Debtors_GuarantorsA103          Residence_Since
##                     1.147                    1.144                    1.326
##               PropertyA122             PropertyA123             PropertyA124
##                     1.777                    1.986                    4.067
##                        Age  Other_InstallmentsA142   Other_InstallmentsA143
##                     1.512                    1.332                    1.412
##               HousingA152              HousingA153 Existing_Credits_at_Bank
##                     1.793                    3.988                    1.738
##                   JobA172                  JobA173                  JobA174
##                    13.177                   17.673                    9.617
##              People_Liable             TelephoneA192       Foreign_WorkerA202
##                     1.271                    1.423                    1.138
```

```
barplot(vif1,
        main="VIF Values (model with all parameters)",
        horiz=TRUE,
        col="lightblue",
        las=2,
        cex.names=0.3,
        )
abline(v=5, lwd=3, lty=2, col="orange")
abline(v=10, lwd=3, lty=2, col="darkred")
```



From VIF, it seems like all the categories of Job have very high VIFs, including the highest one over 12. In the model, no categories of Job are significant their p values are at 0.36, 0.43, and even 0.99. Perhaps, if we remove this variable, the model will improve and we will have more significant variables and smaller VIFs?

Step 5: Logistic Regression without 'Job' (based on VIF)

Let's repeat logistic regression without Job and compare results:

```
set.seed(1)

logreg2 <- train(Credit_Risk~. -Job,
                 data=data_train,
                 method='glm',
                 family=binomial(link = "logit"),
                 trControl = trainControl(method='repeatedcv', number=10, repeats=10))

summary(logreg2)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                              Estimate Std. Error z value     Pr(>|z|)
## (Intercept)                  0.186281   1.147549    0.16      0.87105
## Checking_AccountA12         -0.223383   0.263743   -0.85      0.39701
## Checking_AccountA13         -0.879698   0.430706   -2.04      0.04111 *
## Checking_AccountA14         -1.665568   0.287848   -5.79 0.0000000072 ***
## Duration_months              0.030368   0.011466    2.65      0.00808 **
## Credit_HistoryA31           -0.108258   0.697132   -0.16      0.87659
## Credit_HistoryA32           -0.584028   0.536146   -1.09      0.27602
## Credit_HistoryA33           -0.918468   0.585463   -1.57      0.11670
## Credit_HistoryA34           -1.394604   0.534007   -2.61      0.00901 **
## Credit_PurposeA41           -1.749149   0.450939   -3.88      0.00010 ***
## Credit_PurposeA410          -1.739353   1.184557   -1.47      0.14201
## Credit_PurposeA42           -0.893934   0.318177   -2.81      0.00496 **
## Credit_PurposeA43           -0.941883   0.297281   -3.17      0.00153 **
## Credit_PurposeA44            0.113579   0.820651    0.14      0.88992
## Credit_PurposeA45           -0.432670   0.689700   -0.63      0.53044
## Credit_PurposeA46            0.283633   0.478428    0.59      0.55329
## Credit_PurposeA48          -14.880601 527.147895   -0.03      0.97748
## Credit_PurposeA49           -0.907874   0.415723   -2.18      0.02897 *
## Credit_Amount                0.000152   0.000056    2.71      0.00665 **
## SavingsA62                  -0.191531   0.338476   -0.57      0.57149
## SavingsA63                   0.280578   0.420893    0.67      0.50501
## SavingsA64                  -1.463871   0.605335   -2.42      0.01559 *
## SavingsA65                  -0.810751   0.323358   -2.51      0.01217 *
## Employment_PresentA72       -0.312099   0.450036   -0.69      0.48800
## Employment_PresentA73       -0.222514   0.420068   -0.53      0.59631
## Employment_PresentA74       -0.920141   0.468418   -1.96      0.04949 *
## Employment_PresentA75       -0.220089   0.436346   -0.50      0.61399
## Installment_to_Income        0.397839   0.105577    3.77      0.00016 ***
## Status_and_SexA92           -0.052577   0.458215   -0.11      0.90865
## Status_and_SexA93           -0.772726   0.453695   -1.70      0.08853 .
## Status_and_SexA94           -0.022538   0.541671   -0.04      0.96681
## Debtors_GuarantorsA102       0.369995   0.514946    0.72      0.47244
## Debtors_GuarantorsA103      -0.827922   0.469942   -1.76      0.07811 .
## Residence_Since              0.068799   0.102581    0.67      0.50242
## PropertyA122                 0.498992   0.299970    1.66      0.09622 .
## PropertyA123                0.281659   0.289763    0.97      0.33103
## PropertyA124                0.980807   0.545602    1.80      0.07223 .
## Age                        -0.019642   0.011052   -1.78      0.07552 .
## Other_InstallmentsA142      0.099757   0.537823    0.19      0.85285
## Other_InstallmentsA143     -0.921086   0.274204   -3.36      0.00078 ***
## HousingA152                -0.335759   0.282147   -1.19      0.23404
## HousingA153                -0.863007   0.601534   -1.43      0.15138
## Existing_Credits_at_Bank    0.262557   0.223426    1.18      0.23994
## People_Liable               0.596083   0.296960    2.01      0.04472 *
## TelephoneA192              -0.543770   0.229366   -2.37      0.01775 *
## Foreign_WorkerA202         -1.695445   0.853171   -1.99      0.04690 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 854.50  on 698  degrees of freedom
## Residual deviance: 621.36  on 653  degrees of freedom
## AIC: 713.4
##
## Number of Fisher Scoring iterations: 14
```

```
logreg2$results
```

```
##   parameter Accuracy Kappa AccuracySD KappaSD
## 1      none   0.7489  0.37    0.04497  0.1094
```

We get a slightly higher accuracy and Kappa, but the improvement is not significant. And still not all coefficients are significant, so it is unlikely that further removal of predictors based on VIF one-by-one would make all other variables significant. We can therefore move on and use significant variables only. AIC is 713, a bit lower than AIC of the all-factor model, so we are on the right way with removing insignificant variables.

Step 6: Logistic Regression with significant predictors only (keeping full category sets)

Logistic Regression

As with linear regression, our next step to improve the model is to only leave significant variables in the model.

We saw that for our categorical variables nor all categories were significant for the model. However, we will not remove specifica categories just yet - let's keep categorical variables if at least 1 category was found significant.

With this condition, we are getting rid of: Residence_Since, Housing, Existing_Credits_at_Bank, Job

```
set.seed(1)

logreg3 <- train(Credit_Risk~. - Residence_Since - Housing - Existing_Credits_at_Bank - Job,
                   data=data_train,
                   method='glm',
                   family=binomial(link = "logit"),
                   trControl = trainControl(method='repeatedcv', number=10, repeats=10))

summary(logreg3)
```

```
##
```

```
## Call:
## NULL
##
## Coefficients:
##                           Estimate  Std. Error z value     Pr(>|z|)
## (Intercept)              0.6675201   1.0312409    0.65      0.51744
## Checking_AccountA12     -0.2752312   0.2599134   -1.06      0.28963
## Checking_AccountA13     -1.0066866   0.4268814   -2.36      0.01836 *
## Checking_AccountA14     -1.6935717   0.2862040   -5.92 0.0000000033 ***
## Duration_months          0.0290310   0.0113248    2.56      0.01036 *
## Credit_HistoryA31       -0.3218927   0.6740510   -0.48      0.63297
## Credit_HistoryA32       -0.8187144   0.5091494   -1.61      0.10783
## Credit_HistoryA33       -0.9993265   0.5810391   -1.72      0.08545 .
## Credit_HistoryA34       -1.4354397   0.5309904   -2.70      0.00686 **
## Credit_PurposeA41       -1.7177079   0.4445158   -3.86      0.00011 ***
## Credit_PurposeA410      -1.7087975   1.1656683   -1.47      0.14267
## Credit_PurposeA42       -0.9093469   0.3161496   -2.88      0.00402 **
## Credit_PurposeA43       -0.9507494   0.2941255   -3.23      0.00123 **
## Credit_PurposeA44        0.0670158   0.8248246    0.08      0.93524
## Credit_PurposeA45       -0.4282982   0.6896198   -0.62      0.53456
## Credit_PurposeA46        0.2900448   0.4732144    0.61      0.53993
## Credit_PurposeA48      -14.9185059 529.0212367   -0.03      0.97750
## Credit_PurposeA49       -0.8700048   0.4108346   -2.12      0.03420 *
## Credit_Amount            0.0001471   0.0000549    2.68      0.00739 **
## SavingsA62              -0.1254263   0.3327090   -0.38      0.70618
## SavingsA63               0.3186560   0.4190367    0.76      0.44699
## SavingsA64              -1.4385190   0.6023303   -2.39      0.01693 *
## SavingsA65              -0.7724480   0.3197565   -2.42      0.01570 *
## Employment_PresentA72   -0.2738628   0.4468298   -0.61      0.53994
## Employment_PresentA73   -0.1969752   0.4181565   -0.47      0.63760
## Employment_PresentA74   -0.8505952   0.4648934   -1.83      0.06730 .
## Employment_PresentA75   -0.1335518   0.4294611   -0.31      0.75582
## Installment_to_Income    0.3807670   0.1037823    3.67      0.00024 ***
## Status_and_SexA92        0.0581423   0.4481593    0.13      0.89678
## Status_and_SexA93       -0.7466250   0.4463461   -1.67      0.09438 .
## Status_and_SexA94        0.0073014   0.5378777    0.01      0.98917
## Debtors_GuarantorsA102   0.4685266   0.5071061    0.92      0.35553
## Debtors_GuarantorsA103  -0.8087562   0.4663061   -1.73      0.08285 .
## PropertyA122             0.4692158   0.2984541    1.57      0.11591
## PropertyA123             0.2907215   0.2877679    1.01      0.31237
## PropertyA124             0.5874758   0.3836878    1.53      0.12574
## Age                     -0.0211351   0.0107517   -1.97      0.04933 *
## Other_InstallmentsA142   0.1495599   0.5300190    0.28      0.77781
## Other_InstallmentsA143  -0.9171780   0.2705707   -3.39      0.00070 ***
## People_Liable            0.6037745   0.2945320    2.05      0.04037 *
## TelephoneA192           -0.5042610   0.2268004   -2.22      0.02619 *
## Foreign_WorkerA202      -1.5448745   0.8277620   -1.87      0.06200 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 854.50  on 698  degrees of freedom
## Residual deviance: 626.58  on 657  degrees of freedom
## AIC: 710.6
##
## Number of Fisher Scoring iterations: 14
```

```
logreg3$results
```

```
##   parameter Accuracy  Kappa AccuracySD KappaSD
## 1      none    0.747 0.3649    0.04793  0.1186
```

After exclusion of some predictors, the Accuracy has slightly improved compared to model with all factors, by less than 0.1% to **74.7%**, and Kappa is almost the same. AIC = 710.6, lower than the previous 2 models, so this model has a better fit.

It is important to notice that this time we have one insignificant category - Property. And other categories have significant values. Let's try and remove Property - maybe other significant predictors will appear:

```
set.seed(1)

logreg4 <- train(Credit_Risk~. - Residence_Since - Housing - Existing_Credits_at_Bank - Job - Property,
                   data=data_train,
                   method='glm',
                   family=binomial(link = "logit"),
                   trControl = trainControl(method='repeatedcv', number=10, repeats=10))

summary(logreg4)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                         Estimate  Std. Error z value     Pr(>|z|)
## (Intercept)            0.8864324   1.0101660    0.88      0.38021
## Checking_AccountA12   -0.3020300   0.2580133   -1.17      0.24176
## Checking_AccountA13   -1.0166499   0.4226741   -2.41      0.01616 *
## Checking_AccountA14   -1.7330375   0.2846701   -6.09 0.0000000011 ***
## Duration_months        0.0305189   0.0111355    2.74      0.00613 **
## Credit_HistoryA31     -0.3322854   0.6687322   -0.50      0.61927
## Credit_HistoryA32     -0.8702355   0.5095585   -1.71      0.08767 .
## Credit_HistoryA33     -1.0588159   0.5800378   -1.83      0.06794 .
## Credit_HistoryA34     -1.4955664   0.5316692   -2.81      0.00491 **
## Credit_PurposeA41     -1.6635971   0.4397416   -3.78      0.00015 ***
## Credit_PurposeA410    -1.7339933   1.1859945   -1.46      0.14373
## Credit_PurposeA42     -0.8400842   0.3102130   -2.71      0.00677 **
## Credit_PurposeA43     -0.9739534   0.2927746   -3.33      0.00088 ***
## Credit_PurposeA44      0.0351905   0.8300833    0.04      0.96618
## Credit_PurposeA45     -0.4594151   0.6910401   -0.66      0.50617
## Credit_PurposeA46      0.4511427   0.4626113    0.98      0.32946
## Credit_PurposeA48    -15.0106243 512.9613154   -0.03      0.97666
## Credit_PurposeA49     -0.9336180   0.4061597   -2.30      0.02152 *
## Credit_Amount          0.0001555   0.0000544    2.86      0.00428 **
## SavingsA62            -0.0455653   0.3277505   -0.14      0.88943
## SavingsA63             0.3603257   0.4151577    0.87      0.38544
## SavingsA64            -1.4515990   0.5969662   -2.43      0.01503 *
## SavingsA65            -0.7652398   0.3163935   -2.42      0.01558 *
## Employment_PresentA72 -0.2993534   0.4442902   -0.67      0.50045
## Employment_PresentA73   -0.2912257  0.4132498  -0.70      0.48098
## Employment_PresentA74   -0.9121564  0.4613006  -1.98      0.04800 *
## Employment_PresentA75   -0.1447182  0.4277336  -0.34      0.73511
## Installment_to_Income    0.3950004  0.1028795   3.84      0.00012 ***
## Status_and_SexA92        0.0893207  0.4443843   0.20      0.84070
## Status_and_SexA93       -0.6931079  0.4402726  -1.57      0.11543
## Status_and_SexA94       -0.0203117  0.5312090  -0.04      0.96950
## Debtors_GuarantorsA102   0.4889461  0.5062783   0.97      0.33416
## Debtors_GuarantorsA103  -0.8423774  0.4545696  -1.85      0.06386 .
## Age                     -0.0195981  0.0104836  -1.87      0.06157 .
## Other_InstallmentsA142   0.1126620  0.5276030   0.21      0.83091
## Other_InstallmentsA143  -0.9396181  0.2689443  -3.49      0.00048 ***
## People_Liable            0.5994218  0.2926680   2.05      0.04055 *
## TelephoneA192           -0.4602015  0.2246358  -2.05      0.04050 *
## Foreign_WorkerA202      -1.5032454  0.8124015  -1.85      0.06426 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 854.50  on 698  degrees of freedom
## Residual deviance: 629.92  on 660  degrees of freedom
## AIC: 707.9
##
## Number of Fisher Scoring iterations: 14
```

```
logreg4$results
```

```
##   parameter Accuracy  Kappa AccuracySD KappaSD
## 1      none   0.7485 0.3659    0.04634  0.1141
```

The Accuracy has improved to 74.8%, and Kappa to 36.6%, however now Status and sex is not significant - we remove it:

```
set.seed(1)

logreg5 <- train(Credit_Risk~. - Residence_Since - Housing - Existing_Credits_at_Bank - Job - Property
                   data=data_train,
                   method='glm',
                   family=binomial(link = "logit"),
                   trControl = trainControl(method='repeatedcv', number=10, repeats=10))

summary(logreg5)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                          Estimate Std. Error z value      Pr(>|z|)
## (Intercept)              1.149760   0.908463    1.27       0.20565
## Checking_AccountA12     -0.289636   0.255009   -1.14       0.25605
## Checking_AccountA13     -1.007285   0.419445   -2.40       0.01633 *
## Checking_AccountA14     -1.728482   0.282055   -6.13 0.00000000089 ***
## Duration_months          0.030523   0.010962    2.78       0.00536 **
## Credit_HistoryA31       -0.359681   0.665342   -0.54       0.58879
## Credit_HistoryA32       -0.864383   0.504546   -1.71       0.08668 .
## Credit_HistoryA33       -1.143520   0.575399   -1.99       0.04688 *
## Credit_HistoryA34       -1.549616   0.526515   -2.94       0.00325 **
## Credit_PurposeA41       -1.593667   0.426421   -3.74       0.00019 ***
## Credit_PurposeA410      -1.683154   1.116267   -1.51       0.13160
## Credit_PurposeA42       -0.781891   0.305183   -2.56       0.01041 *
## Credit_PurposeA43       -0.921661   0.288555   -3.19       0.00140 **
## Credit_PurposeA44        0.074400   0.800415    0.09       0.92594
## Credit_PurposeA45       -0.505457   0.696824   -0.73       0.46822
## Credit_PurposeA46        0.531016   0.462498    1.15       0.25091
## Credit_PurposeA48      -14.768983 515.859349   -0.03       0.97716
## Credit_PurposeA49       -0.882706   0.395425   -2.23       0.02560 *
## Credit_Amount            0.000134   0.000053    2.53       0.01151 *
## SavingsA62              -0.040321   0.323241   -0.12       0.90073
## SavingsA63               0.307127   0.413243    0.74       0.45735
## SavingsA64              -1.333009   0.592885   -2.25       0.02455 *
## SavingsA65              -0.734388   0.312575   -2.35       0.01880 *
## Employment_PresentA72   -0.190446   0.439147   -0.43       0.66453
## Employment_PresentA73   -0.276150   0.410044   -0.67       0.50065
## Employment_PresentA74   -0.921296   0.459775   -2.00       0.04509 *
## Employment_PresentA75   -0.163076   0.425112   -0.38       0.70127
## Installment_to_Income    0.340821   0.100505    3.39       0.00070 ***
## Debtors_GuarantorsA102   0.480148   0.506197    0.95       0.34285
## Debtors_GuarantorsA103  -0.838599   0.443675   -1.89       0.05874 .
## Age                     -0.022861   0.010379   -2.20       0.02762 *
## Other_InstallmentsA142   0.029757   0.524922    0.06       0.95479
## Other_InstallmentsA143  -0.875145   0.265578   -3.30       0.00098 ***
## People_Liable            0.305144   0.277518    1.10       0.27153
## TelephoneA192           -0.437323   0.221635   -1.97       0.04848 *
## Foreign_WorkerA202      -1.538964   0.804903   -1.91       0.05588 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 854.50  on 698  degrees of freedom
## Residual deviance: 641.49  on 663  degrees of freedom
## AIC: 713.5
##
## Number of Fisher Scoring iterations: 14
```

```
logreg5$results
```

```
##   parameter Accuracy  Kappa AccuracySD KappaSD
## 1      none   0.7476 0.3581    0.04572   0.113
```

Metrics are not improving, but we have another insignificant parameter - People_liable. Redo the model again:

```
set.seed(1)

logreg6 <- train(Credit_Risk~. - Residence_Since - Housing - Existing_Credits_at_Bank - Job - Property
                   data=data_train,
                   method='glm',
                   family=binomial(link = "logit"),
                   trControl = trainControl(method='repeatedcv', number=10, repeats=10))
```

```r
summary(logreg6)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                            Estimate  Std. Error z value      Pr(>|z|)
## (Intercept)               1.5056588   0.8463646    1.78       0.07524 .
## Checking_AccountA12      -0.3154170   0.2539625   -1.24       0.21424
## Checking_AccountA13      -1.0301774   0.4188777   -2.46       0.01392 *
## Checking_AccountA14      -1.7414725   0.2812382   -6.19 0.00000000059 ***
## Duration_months           0.0299040   0.0109364    2.73       0.00625 **
## Credit_HistoryA31        -0.3001966   0.6605089   -0.45       0.64947
## Credit_HistoryA32        -0.8541269   0.5004737   -1.71       0.08789 .
## Credit_HistoryA33        -1.1044920   0.5707869   -1.94       0.05299 .
## Credit_HistoryA34        -1.5383644   0.5223549   -2.95       0.00323 **
## Credit_PurposeA41        -1.5979160   0.4244099   -3.77       0.00017 ***
## Credit_PurposeA410       -1.6446505   1.1357225   -1.45       0.14759
## Credit_PurposeA42        -0.8131215   0.3033266   -2.68       0.00735 **
## Credit_PurposeA43        -0.9497012   0.2871552   -3.31       0.00094 ***
## Credit_PurposeA44         0.0222261   0.7983807    0.03       0.97779
## Credit_PurposeA45        -0.5068980   0.6995250   -0.72       0.46868
## Credit_PurposeA46         0.5330289   0.4623932    1.15       0.24901
## Credit_PurposeA48       -14.7448031 519.9775142   -0.03       0.97738
## Credit_PurposeA49        -0.8923729   0.3943658   -2.26       0.02365 *
## Credit_Amount             0.0001347   0.0000529    2.55       0.01084 *
## SavingsA62               -0.0445715   0.3226559   -0.14       0.89013
## SavingsA63                0.2891374   0.4115652    0.70       0.48235
## SavingsA64               -1.2910038   0.5858136   -2.20       0.02754 *
## SavingsA65               -0.7432357   0.3130848   -2.37       0.01760 *
## Employment_PresentA72    -0.1641876   0.4375501   -0.38       0.70748
## Employment_PresentA73    -0.2425952   0.4074326   -0.60       0.55156
## Employment_PresentA74    -0.8801648   0.4560162   -1.93       0.05359 .
## Employment_PresentA75    -0.1081430   0.4204732   -0.26       0.79703
## Installment_to_Income     0.3322514   0.1001271    3.32       0.00091 ***
## Debtors_GuarantorsA102    0.4634981   0.5078131    0.91       0.36138
## Debtors_GuarantorsA103   -0.8259083   0.4449475   -1.86       0.06343 .
## Age                      -0.0222451   0.0103125   -2.16       0.03100 *
## Other_InstallmentsA142    0.0399653   0.5228371    0.08       0.93907
## Other_InstallmentsA143   -0.8839296   0.2651250   -3.33       0.00086 ***
## TelephoneA192            -0.4421652   0.2210902   -2.00       0.04551 *
## Foreign_WorkerA202       -1.5191761   0.8101505   -1.88       0.06077 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 854.50  on 698  degrees of freedom
## Residual deviance: 642.69  on 664  degrees of freedom
## AIC: 712.7
##
## Number of Fisher Scoring iterations: 14
```

```
logreg6$results
```

```
##   parameter Accuracy  Kappa AccuracySD KappaSD
## 1      none   0.7473 0.3578    0.04338  0.1052
```

The model is not improving, and we once again get 21 significant variables - so let's keep only them for the new model

Step 7: Logistic Regression with significant values only

This time, we will build logistic regression using only significant variables without keeping full category sets - so if only a few categories from one predictor are significant, we will keep those values only.

Creating binary variables from significant predictors

Since we do not need all the categories of each predictor, we can create new binary data columns for each of the significant variables. We will add values of '1' for rows that contain this significant value and 0 for rows that do not

```
#Checking Account A13, A14
data_train$Checking_AccountA13[data_train$Checking_Account == 'A13'] <-1
data_train$Checking_AccountA13[data_train$Checking_Account != 'A13'] <-0
data_train$Checking_AccountA14[data_train$Checking_Account == 'A14'] <-1
data_train$Checking_AccountA14[data_train$Checking_Account != 'A14'] <-0

#Credit History A32, A33, A34
data_train$Credit_HistoryA32[data_train$Credit_History == 'A32'] <-1
data_train$Credit_HistoryA32[data_train$Credit_History != 'A32'] <-0
data_train$Credit_HistoryA33[data_train$Credit_History == 'A33'] <-1
data_train$Credit_HistoryA33[data_train$Credit_History != 'A33'] <-0
data_train$Credit_HistoryA34[data_train$Credit_History == 'A34'] <-1
data_train$Credit_HistoryA34[data_train$Credit_History != 'A34'] <-0

#Credit Purpose A41, A42, A43, A49
data_train$Credit_PurposeA41[data_train$Credit_Purpose == 'A41'] <-1
data_train$Credit_PurposeA41[data_train$Credit_Purpose != 'A41'] <-0
data_train$Credit_PurposeA42[data_train$Credit_Purpose == 'A42'] <-1
data_train$Credit_PurposeA42[data_train$Credit_Purpose != 'A42'] <-0
data_train$Credit_PurposeA43[data_train$Credit_Purpose == 'A43'] <-1
data_train$Credit_PurposeA43[data_train$Credit_Purpose != 'A43'] <-0
data_train$Credit_PurposeA49[data_train$Credit_Purpose == 'A49'] <-1
data_train$Credit_PurposeA49[data_train$Credit_Purpose != 'A49'] <-0

#Savings A64, A65
data_train$SavingsA64[data_train$Savings == 'A64'] <-1
data_train$SavingsA64[data_train$Savings != 'A64'] <-0
data_train$SavingsA65[data_train$Savings == 'A65'] <-1
data_train$SavingsA65[data_train$Savings != 'A65'] <-0

#Employment A74
data_train$Employment_PresentA74[data_train$Employment_Present == 'A74'] <-1
data_train$Employment_PresentA74[data_train$Employment_Present != 'A74'] <-0

#Debtors_Guarantors A103
data_train$Debtors_GuarantorsA103[data_train$Debtors_Guarantors == 'A103'] <-1
```

```r
data_train$Debtors_GuarantorsA103[data_train$Debtors_Guarantors != 'A103'] <-0

#Other Installments A143
data_train$Other_InstallmentsA143[data_train$Other_Installments == 'A143'] <-1
data_train$Other_InstallmentsA143[data_train$Other_Installments != 'A143'] <-0

#Telephone A192
data_train$TelephoneA192[data_train$Telephone == 'A192'] <-1
data_train$TelephoneA192[data_train$Telephone != 'A192'] <-0

#Foreign_Worker A202
data_train$Foreign_WorkerA202[data_train$Foreign_Worker == 'A202'] <-1
data_train$Foreign_WorkerA202[data_train$Foreign_Worker != 'A202'] <-0
```

Logistic Regression

Using significant variables, build a new model:

```r
set.seed(1)

final <- train(Credit_Risk~ Checking_AccountA13 + Checking_AccountA14 + Duration_months + Credit_Histor
                   data=data_train,
                   method='glm',
                   family=binomial(link = "logit"),
                   trControl = trainControl(method='repeatedcv', number=10, repeats=10))

summary(final)
```

```
##
## Call:
## NULL
##
## Coefficients:
##                          Estimate Std. Error z value       Pr(>|z|)
## (Intercept)             0.7958851  0.5908700    1.35        0.17799
## Checking_AccountA13    -0.8423551  0.3946040   -2.13        0.03279 *
## Checking_AccountA14    -1.5693550  0.2362056   -6.64 0.000000000031 ***
## Duration_months         0.0318988  0.0107170    2.98        0.00292 **
## Credit_HistoryA32      -0.6010404  0.3442000   -1.75        0.08078 .
## Credit_HistoryA33      -0.9294740  0.4409108   -2.11        0.03502 *
## Credit_HistoryA34      -1.2685477  0.3726099   -3.40        0.00066 ***
## Credit_PurposeA41      -1.5377085  0.4031448   -3.81        0.00014 ***
## Credit_PurposeA42      -0.7161428  0.2799619   -2.56        0.01053 *
## Credit_PurposeA43      -0.9114103  0.2594662   -3.51        0.00044 ***
## Credit_PurposeA49      -0.8768048  0.3713409   -2.36        0.01822 *
## Credit_Amount           0.0001241  0.0000505    2.46        0.01403 *
## SavingsA64             -1.3350442  0.5634469   -2.37        0.01782 *
## SavingsA65             -0.7515197  0.2941245   -2.56        0.01062 *
## Employment_PresentA74  -0.6940545  0.2899408   -2.39        0.01668 *
## Installment_to_Income   0.3251179  0.0977653    3.33        0.00088 ***
## Debtors_GuarantorsA103 -0.9146153  0.4386524   -2.09        0.03706 *
## Age                    -0.0184810  0.0094271   -1.96        0.04995 *
## Other_InstallmentsA143 -0.8647828  0.2378630   -3.64        0.00028 ***
## TelephoneA192          -0.4449015  0.2163102   -2.06        0.03971 *
## Foreign_WorkerA202     -1.4422695  0.7964003   -1.81        0.07014 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 854.50  on 698  degrees of freedom
## Residual deviance: 655.52  on 678  degrees of freedom
## AIC: 697.5
##
## Number of Fisher Scoring iterations: 5
```

```
final$results
```

```
##   parameter Accuracy  Kappa AccuracySD KappaSD
## 1      none   0.7598 0.3849    0.04722  0.1197
```

Now, after training and cross-validating the model using only significant values, we have **75.98% Accuracy and AIC of 697.5, which is lower than what we had before, meaning that this model a better fit**. Kappa also increased to **38.5**, showing that our final model makes better predictions.

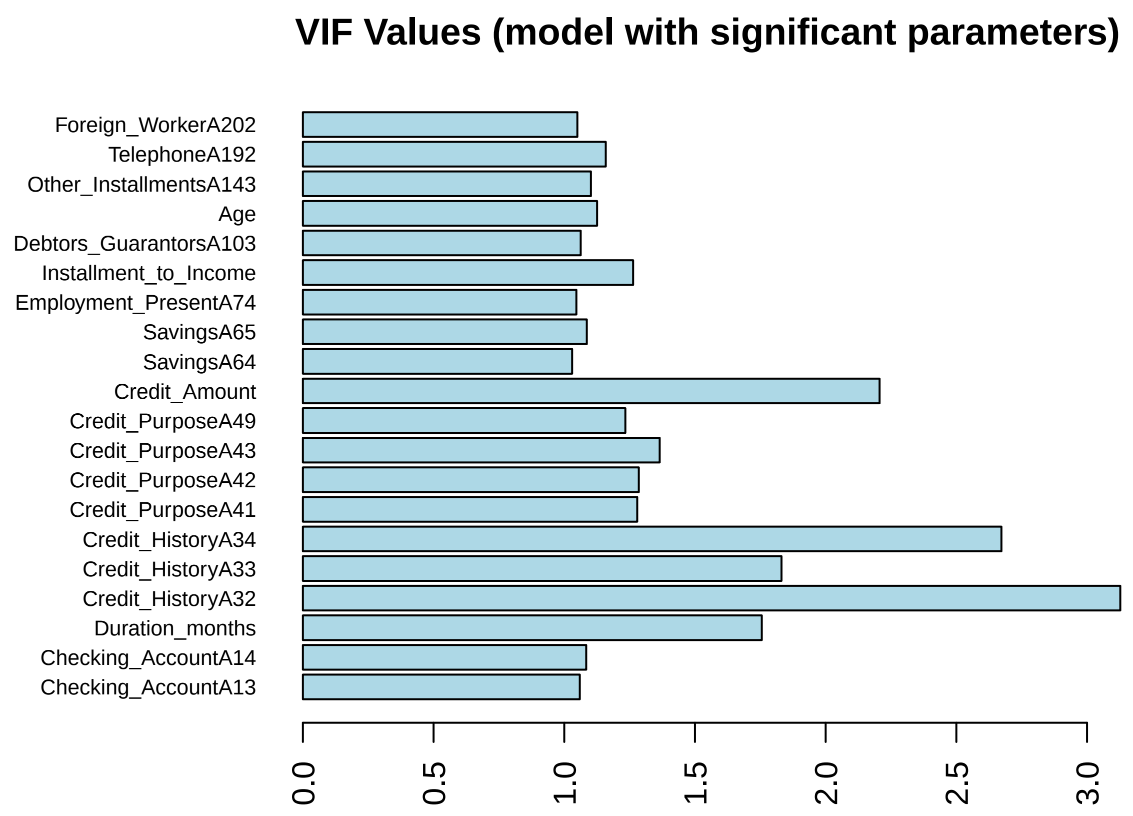
Multicollinearity

Before making a prediction on the test set, let's check VIF values. We remember than we had issues with multicollinearity before, and several predictors exceeded VIF of 5, which meant we could suspect multicollinearity.

```
par(mar = c(3,8,3,3))
vif<-car::vif(final$finalModel)
as.table(vif)
```

```
##    Checking_AccountA13    Checking_AccountA14        Duration_months
##                  1.059                  1.084                  1.756
##      Credit_HistoryA32      Credit_HistoryA33      Credit_HistoryA34
##                  3.127                  1.831                  2.672
##      Credit_PurposeA41      Credit_PurposeA42      Credit_PurposeA43
##                  1.279                  1.285                  1.365
##      Credit_PurposeA49          Credit_Amount              SavingsA64
##                  1.234                  2.206                  1.030
##             SavingsA65  Employment_PresentA74  Installment_to_Income
##                  1.086                  1.046                  1.263
## Debtors_GuarantorsA103                    Age Other_InstallmentsA143
##                  1.063                  1.126                  1.102
##          TelephoneA192     Foreign_WorkerA202
##                  1.159                  1.050
```

```
barplot(vif,
        main="VIF Values (model with significant parameters)",
        horiz=TRUE,
        col="lightblue",
        las=2,
        cex.names=0.7,
        )
abline(v=5, lwd=3, lty=2, col="orange")
abline(v=10, lwd=3, lty=2, col="darkred")
```



As we can see, this time all values are lower than 3, so we have no issues with multicollinearity and our model is good.

Testing the final model

Our final model should now be tested on the test set.

Before validating, we need to add the new value columns to our test set, same as we did with training:

```
#Checking Account A13, A14
data_test$Checking_AccountA13[data_test$Checking_Account == 'A13'] <-1
data_test$Checking_AccountA13[data_test$Checking_Account != 'A13'] <-0
data_test$Checking_AccountA14[data_test$Checking_Account == 'A14'] <-1
data_test$Checking_AccountA14[data_test$Checking_Account != 'A14'] <-0

#Credit History A32, A33, A34
data_test$Credit_HistoryA32[data_test$Credit_History == 'A32'] <-1
data_test$Credit_HistoryA32[data_test$Credit_History != 'A32'] <-0
data_test$Credit_HistoryA33[data_test$Credit_History == 'A33'] <-1
data_test$Credit_HistoryA33[data_test$Credit_History != 'A33'] <-0
data_test$Credit_HistoryA34[data_test$Credit_History == 'A34'] <-1
data_test$Credit_HistoryA34[data_test$Credit_History != 'A34'] <-0

#Credit Purpose A41, A42, A43, A49
data_test$Credit_PurposeA41[data_test$Credit_Purpose == 'A41'] <-1
data_test$Credit_PurposeA41[data_test$Credit_Purpose != 'A41'] <-0
data_test$Credit_PurposeA42[data_test$Credit_Purpose == 'A42'] <-1
```

```r
data_test$Credit_PurposeA42[data_test$Credit_Purpose != 'A42'] <-0
data_test$Credit_PurposeA43[data_test$Credit_Purpose == 'A43'] <-1
data_test$Credit_PurposeA43[data_test$Credit_Purpose != 'A43'] <-0
data_test$Credit_PurposeA49[data_test$Credit_Purpose == 'A49'] <-1
data_test$Credit_PurposeA49[data_test$Credit_Purpose != 'A49'] <-0

#Savings A64, A65
data_test$SavingsA64[data_test$Savings == 'A64'] <-1
data_test$SavingsA64[data_test$Savings != 'A64'] <-0
data_test$SavingsA65[data_test$Savings == 'A65'] <-1
data_test$SavingsA65[data_test$Savings != 'A65'] <-0

#Employment A74
data_test$Employment_PresentA74[data_test$Employment_Present == 'A74'] <-1
data_test$Employment_PresentA74[data_test$Employment_Present != 'A74'] <-0

#Debtors_Guarantors A103
data_test$Debtors_GuarantorsA103[data_test$Debtors_Guarantors == 'A103'] <-1
data_test$Debtors_GuarantorsA103[data_test$Debtors_Guarantors != 'A103'] <-0

#Other Installments A143
data_test$Other_InstallmentsA143[data_test$Other_Installments == 'A143'] <-1
data_test$Other_InstallmentsA143[data_test$Other_Installments != 'A143'] <-0

#Telephone A192
data_test$TelephoneA192[data_test$Telephone == 'A192'] <-1
data_test$TelephoneA192[data_test$Telephone != 'A192'] <-0

#Foreign_Worker A202
data_test$Foreign_WorkerA202[data_test$Foreign_Worker == 'A202'] <-1
data_test$Foreign_WorkerA202[data_test$Foreign_Worker != 'A202'] <-0
```

Now we can make a prediction - round with a 0.5 threshold for now:

```r
set.seed(1)

prediction <- predict(final, newdata=data_test, type='prob')


prediction_round <- factor(as.integer(prediction[,2]>0.5))
prediction_round
```

```
##   [1] 1 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0
##  [38] 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1
##  [75] 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0
## [112] 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
## [149] 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 0 1 1
## [186] 0 1 1 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1
## [223] 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0
## [260] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 1
## [297] 0 0 0 0 1
## Levels: 0 1
```

Assessing Quality - Confusion Matrix

As another measure of quality, let's check the confusion matrix

We first predict probabilities of data points being in group 0 or group 1 - we get a data set with two columns (0 and 1) and probability of data point being in each.

We can round those values. **For now, let's use a threshold of 0.5** - we will try others in part 2. Values higher or equal to 0.5 will be categorizes as 1:

```r
set.seed(1)
cm <- confusionMatrix(
  data = prediction_round,
  reference = relevel(data_test$Credit_Risk, ref = "0")
)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 181  39
##          1  30  51
##
##                Accuracy : 0.771
##                  95% CI : (0.719, 0.817)
##     No Information Rate : 0.701
##     P-Value [Acc > NIR] : 0.00419
##
##                   Kappa : 0.437
##
##  Mcnemar's Test P-Value : 0.33550
##
##             Sensitivity : 0.858
##             Specificity : 0.567
##          Pos Pred Value : 0.823
##          Neg Pred Value : 0.630
##              Prevalence : 0.701
##          Detection Rate : 0.601
##    Detection Prevalence : 0.731
##       Balanced Accuracy : 0.712
##
##        'Positive' Class : 0
##
```

'No Information Rate' in the output means that we have a ratio of 0/1 attributes in training set - so if we predicted '0' for each data point, we would still have an accuracy of 70%. Therefore, **a good model would have an accuracy that exceeds this value**.

Our cross validated model performs on the test set with a **77.1%** accuracy based on confusion matrix (the function calculates it as (TP+TN)/(TP+TN+FN+FP)), which is a good accuracy thanks to cross-validation and two sets (we minimized overfitting)

Our model has sensitivity of 85.8% and specificity of 56.7%

Misclassification error is 22.9%:

```r
#misclassification error
round((cm$table[1,2]+cm$table[2,1])/sum(cm$table)*100,2)
```

```
## [1] 22.92
```

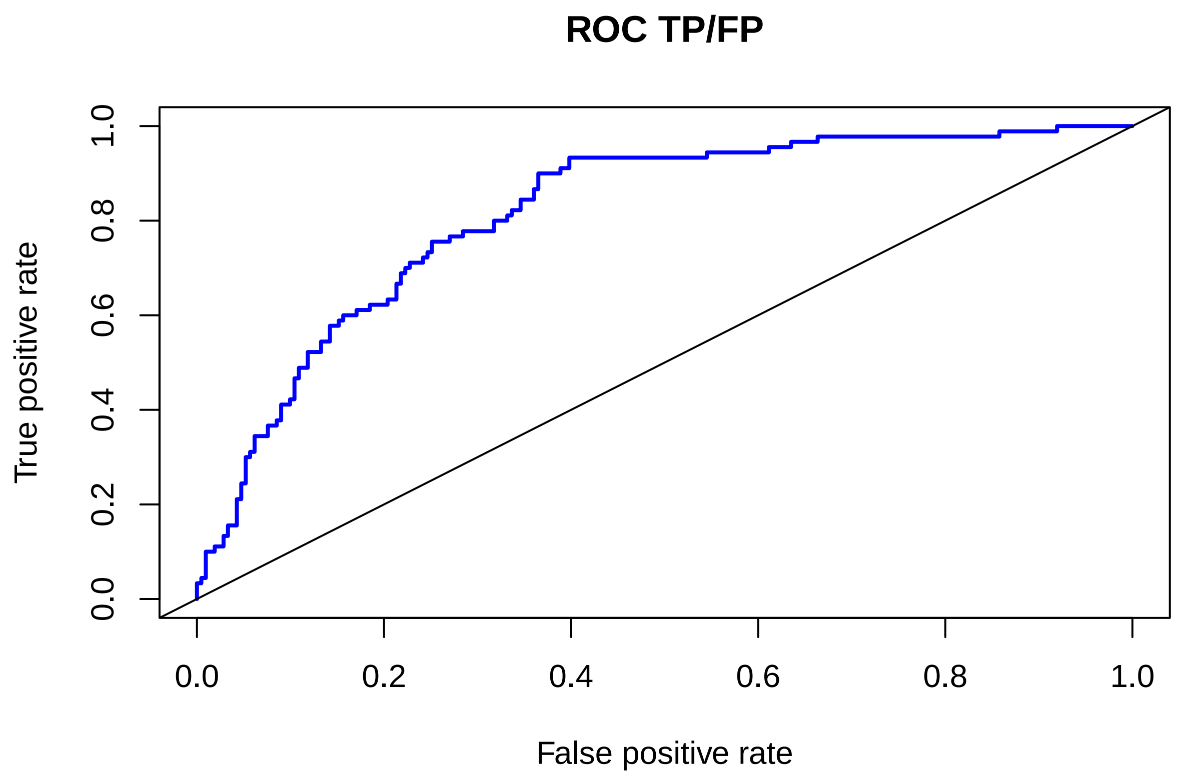
AS expected, the accuracy on test set is slightly lower than on training set due to less overfitting and fewer random patterns, but it is still a good result. Let's keep exploring other quality parameters of the model.

Assessing Quality - ROC Curve

Let's plot ROC curve for our prediction.

First, we build a plot using probabilities to compare true positive and true negative values - the result looks good:

```
library(ROCR)

# predicted probabilities

prob_final <- predict(final, data_test, type = "prob")[,2]

# AUC metrics
perffinal <- prediction(prob_final, data_test$Credit_Risk) %>%
  performance(measure = "tpr", x.measure = "fpr")

# ROC curve
plot(perffinal, col = "blue", main='ROC TP/FP', lwd=2)
abline(0,1)
```



Next, let's plot ROC with rounded values - we can see that aur Area Under the Curve is 71.2%, and our confidence intervals with alpha of 0.9 are 65.6-76.9%. **This is the ability of our model to put data**

points into Good/Bad credit risk categories. By rule of thumb AUC of 0.5 is a bad classifier, and the higher above that AUC is, the better - this value is not excellent or outstanding as AUC >0.8 or 0.9 would be, but we can say that it is 'good' and beeter than just 'acceptable'.

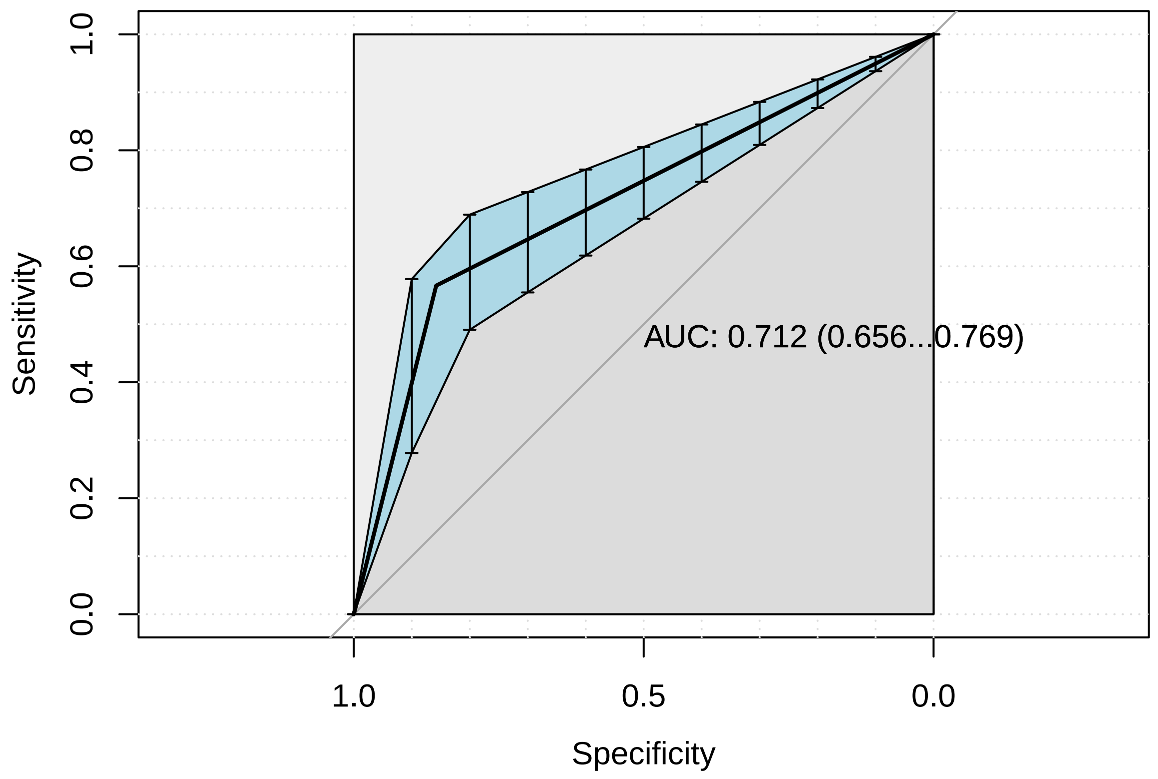
```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
roccurve <- roc(response=data_test$Credit_Risk,predictor=as.integer(prediction[,2]>0.5),
                levels = c(0, 1),
                direction = "<",
                ci=TRUE, ci.alpha=0.9, stratified=FALSE,
                plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
                print.auc=TRUE, show.thres=TRUE)

sens.ci <- ci.se(roccurve)
plot(sens.ci, type="shape", col="lightblue")
```

```
## Warning in plot.ci.se(sens.ci, type = "shape", col = "lightblue"): Low
## definition shape.
```

```r
plot(sens.ci, type="bars")
```



Step 8: Showing the final model

Having run 7 iterations of cross-validated logistic regression model, we found the best one with **lowest AIC among other models we've built of 697**, suggesting that this model fits the data better than others, and **Accuracy of 77.1% on test data (calculated with confusion matrix) using threshold of 0.5**. We cross-validated model using repeated cross-validation with 10 folds and 10 repeats, so considering that we have a good size of both training and validation sets (700/300), random patterns (overfitting) should be minimized, and we can trust the Accuracy we got.

Let's show the coefficients of the final model - remember that we renamed the columns to get more understanding of what's inside, so I will name all the categories we use as variables as well:

Variable (Initial Name)	Value	Description	Coefficient
Intercept	-	-	-
Checking_Account (V1)	Checking_AccountA13	>= 200 DM / salary assignments for at least 1 year	-0.8423551
Checking_Account (V1)	Checking_AccountA14	no checking account	-1.5693549
Duration_Months (V2)	Duration_months	Duration of credit	0.0318988
Credit_History (V3)	Credit_HistoryA32	existing credits paid back duly till now	-0.6010404
Credit_History (V3)	Credit_HistoryA33	delay in paying off in the past	-0.9294740
Credit_History (V3)	Credit_HistoryA34	A34 : critical account/ other credits existing (not at this bank)	-1.2685477
Credit_Purpose (V4)	Credit_PurposeA41	car (used)	-1.5377085
Credit_Purpose (V4)	Credit_PurposeA42	furniture/equipment	-0.7161428
Credit_Purpose (V4)	Credit_PurposeA43	radio/television	-0.9114103
Credit_Purpose (V4)	Credit_PurposeA49	business	-0.8768048
Credit_Amount (V5)	Credit_Amount	Amount of credit	0.0001241
Savings (V6)	SavingsA64	>= 1000 DM	-1.3350442
Savings (V6)	SavingsA65	unknown/ no savings account	-0.7515197
Employment_Present (V7)	Employment_PresentA74	Present employment since 4 <= ... < 7 years	-0.6940545
Installment_to_Income (V8)	Installment_to_Income	Installment rate in percentage of disposable income	0.3251178
Debtors_Guarantors (V10)	Debtors_GuarantorsA103	guarantor	-0.9146153
Age (V13)	Age	Age in years	-0.0184810
Other_Installments (V14)	Other_InstallmentsA143	None	-0.8647828
Telephone (V19)	TelephoneA192	yes, registered under the customers name	-0.4449015
Foreign_Worker (V20)	Foreign_WorkerA202	no	-1.4422695

(Extra) Step 9: Test other thresholds

Let's check how accuracy and AUC would change if we use other thresholds. I will use values from 0 to 1 with a 0.01 step

```r
auc <- list()
acc <- list()
thresh <- seq(0,1, by=0.01)
for (i in thresh){
  acci=confusionMatrix(data = factor(as.integer(prediction[,2]>i)),
                      reference = relevel(data_test$Credit_Risk, ref = "0"))$overall[1]
  auci=roc(response=data_test$Credit_Risk,predictor=as.integer(prediction[,2]>i),
           levels = c(0, 1),
           direction = "<")$auc
  acc<-append(acc,as.numeric(acci))
  auc<-append(auc,as.numeric(auci))
}
```

```
## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

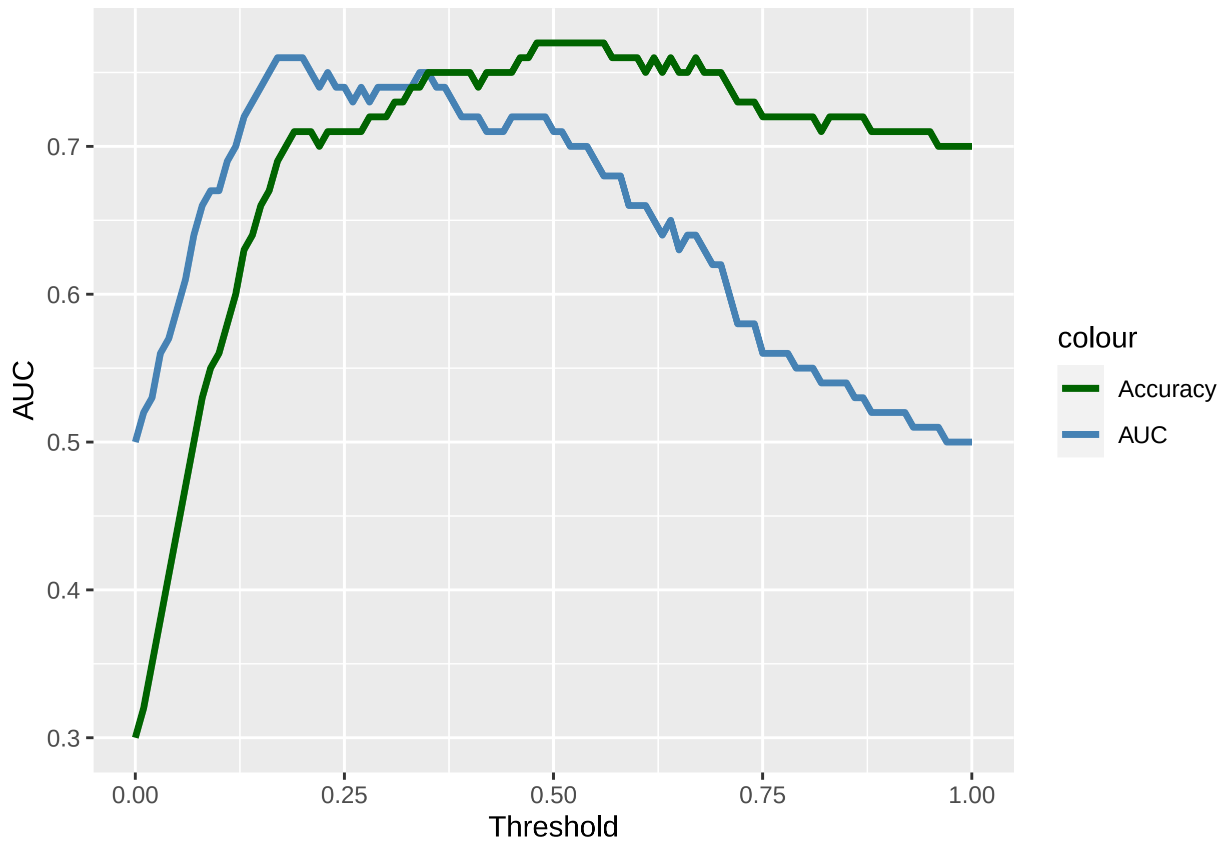
```r
thresholds<-data.frame(unlist(thresh), round(unlist(auc),2), round(unlist(acc),2))
colnames(thresholds)<-c('Threshold', 'AUC', 'Accuracy')
head(thresholds)
```

```
##   Threshold  AUC Accuracy
## 1      0.00 0.50     0.30
## 2      0.01 0.52     0.32
## 3      0.02 0.53     0.35
## 4      0.03 0.56     0.38
## 5      0.04 0.57     0.41
## 6      0.05 0.59     0.44
```

Visualize the results:

```r
colors<-c('AUC'='steelblue', 'Accuracy'='darkgreen')
ggplot(thresholds, aes(x=Threshold))+
  geom_line(aes(y=AUC, color='AUC'), size=1.2)+
  geom_line(aes(y=Accuracy, color='Accuracy'), size=1.2)+
  scale_color_manual(values=colors)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```



As we can see, the compromise where AUC value 'meets' Accuracy is at **threshold of 0.33**. After that, AUC starts to fall, and Accuracy increases.

So our threshold choice would depend on our goal:

The **'golden mean'**, where AUC and Accuracy meet each other is threshold **0.33 and 0.35**:

```r
thresholds[thresholds$AUC==thresholds$Accuracy,]
```

```
##    Threshold  AUC Accuracy
## 34      0.33 0.74     0.74
## 36      0.35 0.75     0.75
```

If maximizing **AUC** is the priority - **threshold 0.17-0.20** is best:

```r
thresholds[thresholds$AUC==max(thresholds$AUC),]
```

```
##    Threshold  AUC Accuracy
## 18      0.17 0.76     0.69
## 19      0.18 0.76     0.70
## 20      0.19 0.76     0.71
## 21      0.20 0.76     0.71
```

If maximizing **Accuracy** is the priority - **threshold 0.48 - 0.56** is best:

```r
thresholds[thresholds$Accuracy==max(thresholds$Accuracy),]
```

```
##    Threshold  AUC Accuracy
## 49      0.48 0.72     0.77
## 50      0.49 0.72     0.77
## 51      0.50 0.71     0.77
## 52      0.51 0.71     0.77
## 53      0.52 0.70     0.77
## 54      0.53 0.70     0.77
## 55      0.54 0.70     0.77
## 56      0.55 0.69     0.77
## 57      0.56 0.68     0.77
```

But what if we take cost into consideration? Let's check in the next section

Part 2.B Threshold with Cost

Now we know that incorrectly identifying a customer with bad risk as a good one (0), False Positive, is 5 times more expensive than misclassifying a good risk customer as a bad one , False Negative.

We can calculate loss for each of the thresholds.

Step 10: Calculate Loss

Knowing the cost, we build confusion matrix and count each False Positive as 5 False Negatives:

```
losses<-list()
thresh <- seq(0,1, by=0.01)
for (i in thresh){
  confm = confusionMatrix(data = factor(as.integer(prediction[,2]>i)),
                          reference = relevel(data_test$Credit_Risk, ref = "0"))
  losses <- append(losses, confm$table[1,2]*5+confm$table[2,1])
}
```

```
## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(data = factor(as.integer(prediction[, :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

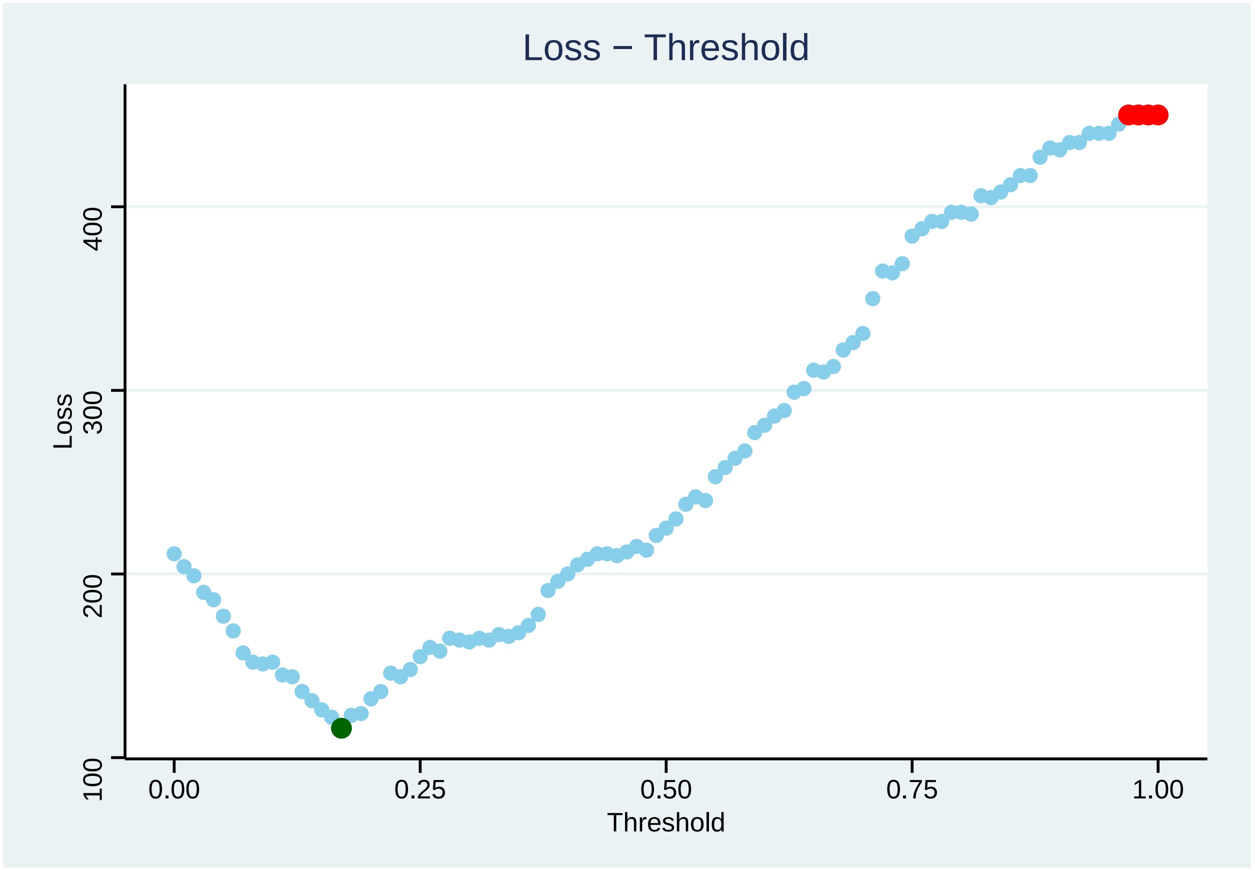
```
losses <- data.frame(unlist(thresh), unlist(losses))
colnames(losses) <- c('Threshold', 'Loss')
head(losses)
```

```
##   Threshold Loss
## 1      0.00  211
## 2       0.01  204
## 3       0.02  199
## 4       0.03  190
## 5       0.04  186
## 6       0.05  177
```

Step 11: Find best threshold

Now we know the losses for each of our thresholds. Let's visualize the result:

```
ggplot(losses, aes(x=Threshold, y=Loss))+
  geom_point(color='skyblue', size=2)+
  geom_point(data= losses[losses$Loss==min(losses$Loss),], color='darkgreen', size=3)+
  geom_point(data= losses[losses$Loss==max(losses$Loss),], color='red', size=3)+
  labs(title='Loss - Threshold')+
  theme_stata()
```



As we can see, **to minimize loss, we should keep the threshold at about 15-19%**, with **minimal loss of 116 achieved at 0.17 threshold**:

```
losses[losses$Loss==min(losses$Loss),]
```

```
##    Threshold Loss
## 18      0.17  116
```

So, with the given cost of one false positive equal to 5 false negatives, we should use **threshold of 0.17**, which will give us **AUC of 76% an Accuracy of 69%**. So in this case, we are sacrificing the accuracy to avoid costly false positives, but we get a 'good' AUC, or degree of separability, of 0.76:

```r
thresholds[thresholds$Threshold==0.17,]
```

```
##    Threshold  AUC Accuracy
## 18      0.17 0.76     0.69
```