Homework_7

February 28, 2024

Question 10.1 - Part A

> Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

To create the regression tree model I'll use the rpart package. For the random forest I'll use the randomForest package. I found this article helpful in understanding how to create the decision tree and random forest: https://quantdev.ssri.psu.edu/tutorials/introduction-classification-regression-trees.

1) Explore the data to gain a better understanding of the dataset
2) Normalize the data
3) Look at correlation values
4) Apply the PCA
5) Determine optimal number of principal components
6) Create new linear regression model in terms of the original variables
7) Compare the new model to the solution from the previous homework

Step 1 - Explore the Data For this step I am going to visually inspect the data, get summary statistics to get a sense of the properties of the values in the data, and read about the data to understand what each value represents. I'll start by loading the libraries I need for this analysis and loading the data.

[37]:
```
# Set seed so output is reproducible
set.seed(123)

# Load libraries needed for analysis
library(ggplot2) # for plots
library(dplyr) # includes pipeline operator
library(rpart) # tree model
library(rpart.plot) # visualizing the tree model
library(rattle) # another way to visualize the tree model
library(randomForest) # random forest model
library(caret) # for performing cross-validation
library(cowplot) # for creating grid of plots
library(pROC) # creating ROC curves
```

[2]:
```
# Load the data
crime_data <- read.table("uscrime.txt", header = TRUE)
```

[3]:
```
# Visually inspect the data, for this report I only printed the first few rows␣
  ↪but inspected all of them
head(crime_data[,1:10])
head(crime_data[,11:16])
```

M	So	Ed	Po1	Po2	LF	M.F	Pop	NW	U1
15.1	1	9.1	5.8	5.6	0.510	95.0	33	30.1	0.108
14.3	0	11.3	10.3	9.5	0.583	101.2	13	10.2	0.096
14.2	1	8.9	4.5	4.4	0.533	96.9	18	21.9	0.094
13.6	0	12.1	14.9	14.1	0.577	99.4	157	8.0	0.102
14.1	0	12.1	10.9	10.1	0.591	98.5	18	3.0	0.091
12.1	0	11.0	11.8	11.5	0.547	96.4	25	4.4	0.084

U2	Wealth	Ineq	Prob	Time	Crime
4.1	3940	26.1	0.084602	26.2011	791
3.6	5570	19.4	0.029599	25.2999	1635
3.3	3180	25.0	0.083401	24.3006	578
3.9	6730	16.7	0.015801	29.9012	1969
2.0	5780	17.4	0.041399	21.2998	1234
2.9	6890	12.6	0.034201	20.9995	682

I don't like the labels for some of the columns in the data as it is hard to understand what they represent just by their name. A description of the attributes taken from the website can be seen below. Next, I am going to rename my columns to match the descriptions a little better. I am also going to make a couple of tweaks to the data, I am going to turn the LF, M.F, U1, and Prob variables into percentages to match the M, NW, U2 and Ineq variables. It seems odd that the U2 variable would be as a percentage and the U1 variable is set as a ratio since they represent the same thing for different age groups. I also just want the consistency of having everything listed as a percentage. I will of course have to apply the same transformation to the values from the data point we are trying to predict or to the coefficients of the final model.

```
M:      percentage of males aged 14-24 in total state population
So:     indicator variable for Southern states (0 = No, 1 = Yes)
Ed:     mean years of schooling of the population aged 25 years or over
Po1:    per capita expenditure on police protection in 1960
Po2:    per capita expenditure on police protection in 1959
LF:     labor force participation rate of civilian urban males in the age-group 14-24
M.F:    number of males per 100 females
Pop:    state population size in hundred thousands
NW:     percentage of nonwhites in the population
U1:     unemployment rate of urban males 14-24
U2:     unemployment rate of urban males 35-39
Wealth: median value of transferable assets or family income
Ineq:   percentage of families earning below half the median income
Prob:   probability of imprisonment: ratio of number of commitments to number of offenses
Time:   average time in months served by offenders in state prisons before their first release
```

Crime: number of offenses per 100,000 population in 1960

[4]:
```
# Assign more meaningful variable names and update several values in the␣
 ↪dataset (not necessary but it makes it easier for me to interpret)
# %>% is a pipe operator that allows several operations (rename and mutate) to␣
 ↪occur
crime_data <- crime_data %>%
  rename(percent_young_male = M,
         southern_state = So,
         average_ed = Ed,
         per_cap_exp_1960 = Po1,
         per_cap_exp_1959 = Po2,
         laborforce_part = LF,
         percent_male = M.F,
         population = Pop,
         nonwhite = NW,
         unemp_youth = U1,
         unemp_adult = U2,
         median_assets = Wealth,
         low_income = Ineq,
         prob_imprisonment = Prob,
         time_served = Time,
         crimes_per_million = Crime) %>%
  mutate(laborforce_part = laborforce_part * 100,
         percent_male = (percent_male / (percent_male + 100)) * 100,
         unemp_youth = unemp_youth * 100,
         prob_imprisonment = prob_imprisonment * 100)

# print summary of the transformed data
summary(crime_data)
```

```
 percent_young_male southern_state     average_ed    per_cap_exp_1960
 Min.   :11.90      Min.   :0.0000   Min.   : 8.70   Min.   : 4.50
 1st Qu.:13.00      1st Qu.:0.0000   1st Qu.: 9.75   1st Qu.: 6.25
 Median :13.60      Median :0.0000   Median :10.80   Median : 7.80
 Mean   :13.86      Mean   :0.3404   Mean   :10.56   Mean   : 8.50
 3rd Qu.:14.60      3rd Qu.:1.0000   3rd Qu.:11.45   3rd Qu.:10.45
 Max.   :17.70      Max.   :1.0000   Max.   :12.20   Max.   :16.60
 per_cap_exp_1959 laborforce_part  percent_male     population
 Min.   : 4.100   Min.   :48.00   Min.   :48.29   Min.   :  3.00
 1st Qu.: 5.850   1st Qu.:53.05   1st Qu.:49.10   1st Qu.: 10.00
 Median : 7.300   Median :56.00   Median :49.42   Median : 25.00
 Mean   : 8.023   Mean   :56.12   Mean   :49.56   Mean   : 36.62
 3rd Qu.: 9.700   3rd Qu.:59.30   3rd Qu.:49.80   3rd Qu.: 41.50
 Max.   :15.700   Max.   :64.10   Max.   :51.71   Max.   :168.00
    nonwhite      unemp_youth      unemp_adult    median_assets
 Min.   : 0.20   Min.   : 7.000   Min.   :2.000   Min.   :2880
 1st Qu.: 2.40   1st Qu.: 8.050   1st Qu.:2.750   1st Qu.:4595
```

```
 Median : 7.60   Median : 9.200   Median :3.400   Median :5370
 Mean   :10.11   Mean   : 9.547   Mean   :3.398   Mean   :5254
 3rd Qu.:13.25   3rd Qu.:10.400   3rd Qu.:3.850   3rd Qu.:5915
 Max.   :42.30   Max.   :14.200   Max.   :5.800   Max.   :6890
   low_income    prob_imprisonment  time_served    crimes_per_million
 Min.   :12.60   Min.   : 0.690    Min.   :12.20   Min.   : 342.0
 1st Qu.:16.55   1st Qu.: 3.270    1st Qu.:21.60   1st Qu.: 658.5
 Median :17.60   Median : 4.210    Median :25.80   Median : 831.0
 Mean   :19.40   Mean   : 4.709    Mean   :26.60   Mean   : 905.1
 3rd Qu.:22.75   3rd Qu.: 5.445    3rd Qu.:30.45   3rd Qu.:1057.5
 Max.   :27.60   Max.   :11.980    Max.   :44.00   Max.   :1993.0
```

Everyone should be pretty familar with the data at this point so I won't go much into explaining it.

Step 2 - Grow a Tree Now that the data is loaded I will grow the tree! I am using the rpart() package and since I am interested in a regression tree I will set the method to "anova". I also played with some of the control paramaters including setting the minimum number of observations in a node before a split is attempted (minsplit), how much each split must decrease the overall lack of fit (cp), and the maximum depth of any node of the final tree (maxdepth - root node is counted as depth 0). It was interesting to see how these parameters affected the final model. Since I want practice pruning the tree I chose parameters that created more terminal nodes than the base model did.

Since the dataset is so small I didn't split it into training/testing data but will use cross-validation to analyze the model performance. This won't allow for an accurate errro estimation for the model but should give some insight into it's performance. More information about the rpart() function can be found here: https://www.rdocumentation.org/packages/rpart/versions/4.1.23/topics/rpart

[5]:
```
# minimum number of points for the 5% rule of thumb
min_points <- ceiling(0.05 * nrow(crime_data))
cat("The minimum number of points in each branch needs to be", min_points, "to␣
 ↪meet the 5% rule of thumb.", "\n\n")

# Grow tree without control parameters for comparison
crime_tree_default <- rpart(crimes_per_million ~ ., method = "anova", data =␣
 ↪crime_data)
cat("Model without control parameters:", "\n")
crime_tree_default

# Grow the final tree using the control parameters (only ended up using the␣
 ↪minsplit character which has a default of 20, the default for cp is 0.01)
crime_tree <- rpart(crimes_per_million ~ ., method = "anova", data =␣
 ↪crime_data, control = rpart.control(minsplit = 9))
cat("\n\n","Model with control parameters:", "\n")
crime_tree
```

```
The minimum number of points in each branch needs to be 3 to meet the 5% rule of
thumb.
```

```
Model without control parameters:

n= 47

node), split, n, deviance, yval
      * denotes terminal node

1) root 47 6880928.0  905.0851
  2) per_cap_exp_1960< 7.65 23  779243.5  669.6087
    4) population< 22.5 12  243811.0  550.5000 *
    5) population>=22.5 11  179470.7  799.5455 *
  3) per_cap_exp_1960>=7.65 24 3604162.0 1130.7500
    6) nonwhite< 7.65 10  557574.9  886.9000 *
    7) nonwhite>=7.65 14 2027225.0 1304.9290 *



 Model with control parameters:

n= 47

node), split, n, deviance, yval
      * denotes terminal node

 1) root 47 6880928.00  905.0851
   2) per_cap_exp_1960< 7.65 23  779243.50  669.6087
     4) population< 22.5 12  243811.00  550.5000
       8) average_ed< 11.65 9   58820.89  484.1111 *
       9) average_ed>=11.65 3   26320.67  749.6667 *
     5) population>=22.5 11  179470.70  799.5455 *
   3) per_cap_exp_1960>=7.65 24 3604162.00 1130.7500
     6) nonwhite< 7.65 10  557574.90  886.9000
      12) percent_young_male< 13.05 6  109289.50  727.5000 *
      13) percent_young_male>=13.05 4   67160.00 1126.0000 *
     7) nonwhite>=7.65 14 2027225.00 1304.9290
      14) median_assets< 6470 11  707252.70 1148.4550
        28) prob_imprisonment>=4.10995 4   28226.00  898.0000 *
        29) prob_imprisonment< 4.10995 7  284739.70 1291.5710 *
      15) median_assets>=6470 3   63120.67 1878.6670 *
```

Inspecting the models above, the biggest difference is that the branch for per_cap_exp_1960 >= 7.65 splits the nonwhite branches additional times, and the branch for per_cap_exp_1960 < 7.65 splits the population < 22.5 an additional time. The default value for minsplit is 20 so by reducing it to 9 it allowed the model to create several more branches. The minbucket paramater is set to a default of (minstplit/3) which means it is equal to 3. This ensures that each terminal node had at least 5% of the data in it. I suspect I will still end up pruning most of the additional branches since they have a very small number of datapoints (3, 7, 4, 4, 6, 11, 3, and 9) which likely means the model is overfitting the data, but I am going to use the model that was created with control parameters to get more experience pruning.

Step 3 - Inspect the Tree Next I am going to inspect the tree using a variety of plots and outputs.

[6]:
```
# Get a detailed summary of the model
summary(crime_tree)
```

```
Call:
rpart(formula = crimes_per_million ~ ., data = crime_data, method = "anova",
    control = rpart.control(minsplit = 9))
  n= 47

          CP nsplit rel error   xerror      xstd
1 0.36296293      0 1.0000000 1.062718 0.2641620
2 0.16540024      1 0.6370371 1.131172 0.2744131
3 0.05730143      3 0.3062366 1.292726 0.3268601
4 0.05538867      4 0.2489352 1.152379 0.3190650
5 0.05173165      5 0.1935465 1.178961 0.3179641
6 0.02305931      6 0.1418148 1.172783 0.3249076
7 0.01000000      7 0.1187555 1.146689 0.3351339

Variable importance
  per_cap_exp_1959      median_assets   per_cap_exp_1960  prob_imprisonment
                15                 14                 14                 12
        low_income percent_young_male         average_ed           nonwhite
                 8                  8                  8                  7
   laborforce_part        time_served         population     southern_state
                 4                  4                  3                  1
       unemp_adult
                 1

Node number 1: 47 observations,    complexity param=0.3629629
  mean=905.0851, MSE=146402.7
  left son=2 (23 obs) right son=3 (24 obs)
  Primary splits:
      per_cap_exp_1960  < 7.65     to the left,  improve=0.3629629, (0 missing)
      per_cap_exp_1959  < 7.2      to the left,  improve=0.3629629, (0 missing)
      prob_imprisonment < 4.18485  to the right, improve=0.3217700, (0 missing)
      median_assets     < 6470     to the left,  improve=0.2889992, (0 missing)
      nonwhite          < 7.65     to the left,  improve=0.2356621, (0 missing)
  Surrogate splits:
      per_cap_exp_1959   < 7.2      to the left,  agree=1.000, adj=1.000, (0
split)
      median_assets      < 5330     to the left,  agree=0.830, adj=0.652, (0
split)
      prob_imprisonment  < 4.3598   to the right, agree=0.809, adj=0.609, (0
split)
      percent_young_male < 13.25    to the right, agree=0.745, adj=0.478, (0
split)
```

```
      low_income         < 17.15    to the right, agree=0.745, adj=0.478, (0
split)

Node number 2: 23 observations,    complexity param=0.05173165
  mean=669.6087, MSE=33880.15
  left son=4 (12 obs) right son=5 (11 obs)
  Primary splits:
      population         < 22.5     to the left,  improve=0.4568043, (0 missing)
      percent_young_male < 14.5     to the left,  improve=0.3931567, (0 missing)
      per_cap_exp_1960   < 5.65     to the left,  improve=0.3279971, (0 missing)
      nonwhite           < 5.4      to the left,  improve=0.3184074, (0 missing)
      unemp_youth        < 9.3      to the right, improve=0.2119062, (0 missing)
  Surrogate splits:
      nonwhite           < 5.4      to the left,  agree=0.826, adj=0.636, (0
split)
      percent_young_male < 14.5     to the left,  agree=0.783, adj=0.545, (0
split)
      time_served        < 22.30055 to the left,  agree=0.783, adj=0.545, (0
split)
      southern_state     < 0.5      to the left,  agree=0.739, adj=0.455, (0
split)
      average_ed         < 10.85    to the right, agree=0.739, adj=0.455, (0
split)

Node number 3: 24 observations,    complexity param=0.1654002
  mean=1130.75, MSE=150173.4
  left son=6 (10 obs) right son=7 (14 obs)
  Primary splits:
      nonwhite           < 7.65     to the left,  improve=0.2828293, (0 missing)
      per_cap_exp_1960   < 13.85    to the left,  improve=0.2749163, (0 missing)
      per_cap_exp_1959   < 13.45    to the left,  improve=0.2749163, (0 missing)
      percent_young_male < 13.05    to the left,  improve=0.2714159, (0 missing)
      median_assets      < 6470     to the left,  improve=0.2681920, (0 missing)
  Surrogate splits:
      average_ed      < 11.45    to the right, agree=0.750, adj=0.4, (0 split)
      low_income      < 16.25    to the left,  agree=0.750, adj=0.4, (0 split)
      time_served     < 21.9001  to the left,  agree=0.750, adj=0.4, (0 split)
      population      < 30       to the left,  agree=0.708, adj=0.3, (0 split)
      laborforce_part < 58.85    to the right, agree=0.667, adj=0.2, (0 split)

Node number 4: 12 observations,    complexity param=0.02305931
  mean=550.5, MSE=20317.58
  left son=8 (9 obs) right son=9 (3 obs)
  Primary splits:
      average_ed      < 11.65    to the left,  improve=0.6507887, (0 missing)
      laborforce_part < 56.75    to the left,  improve=0.4820740, (0 missing)
      population      < 6.5      to the left,  improve=0.2836993, (0 missing)
      percent_male    < 49.1999  to the left,  improve=0.2765093, (0 missing)
```

```
      median_assets   < 5075     to the left,  improve=0.2327433, (0 missing)
  Surrogate splits:
      per_cap_exp_1960 < 7        to the left,  agree=0.917, adj=0.667, (0
split)
      median_assets    < 5075     to the left,  agree=0.917, adj=0.667, (0
split)
      per_cap_exp_1959 < 5.9      to the left,  agree=0.833, adj=0.333, (0
split)
      laborforce_part  < 56.75    to the left,  agree=0.833, adj=0.333, (0
split)
      percent_male     < 49.62215 to the left,  agree=0.833, adj=0.333, (0
split)

Node number 5: 11 observations
  mean=799.5455, MSE=16315.52

Node number 6: 10 observations,    complexity param=0.05538867
  mean=886.9, MSE=55757.49
  left son=12 (6 obs) right son=13 (4 obs)
  Primary splits:
      percent_young_male < 13.05    to the left,  improve=0.6835412, (0 missing)
      population         < 34.5     to the right, improve=0.4800125, (0 missing)
      unemp_youth        < 8.55     to the left,  improve=0.4435949, (0 missing)
      median_assets      < 5830     to the right, improve=0.2508664, (0 missing)
      time_served        < 21.14965 to the left,  improve=0.2183925, (0 missing)
  Surrogate splits:
      median_assets    < 5655     to the right, agree=0.8, adj=0.50, (0 split)
      low_income       < 17.25    to the left,  agree=0.8, adj=0.50, (0 split)
      average_ed       < 11.95    to the left,  agree=0.7, adj=0.25, (0 split)
      per_cap_exp_1960 < 9.6      to the right, agree=0.7, adj=0.25, (0 split)
      laborforce_part  < 56.05    to the left,  agree=0.7, adj=0.25, (0 split)

Node number 7: 14 observations,    complexity param=0.1654002
  mean=1304.929, MSE=144801.8
  left son=14 (11 obs) right son=15 (3 obs)
  Primary splits:
      median_assets     < 6470     to the left,  improve=0.6199862, (0 missing)
      average_ed        < 11.25    to the left,  improve=0.5922087, (0 missing)
      prob_imprisonment < 4.1899   to the right, improve=0.4574277, (0 missing)
      percent_male      < 49.73586 to the left,  improve=0.4382818, (0 missing)
      per_cap_exp_1959  < 11.55    to the left,  improve=0.3676603, (0 missing)
  Surrogate splits:
      average_ed        < 11.7     to the left,  agree=0.929, adj=0.667, (0
split)
      per_cap_exp_1959  < 11.55    to the left,  agree=0.929, adj=0.667, (0
split)
      prob_imprisonment < 1.98005  to the right, agree=0.929, adj=0.667, (0
split)
```

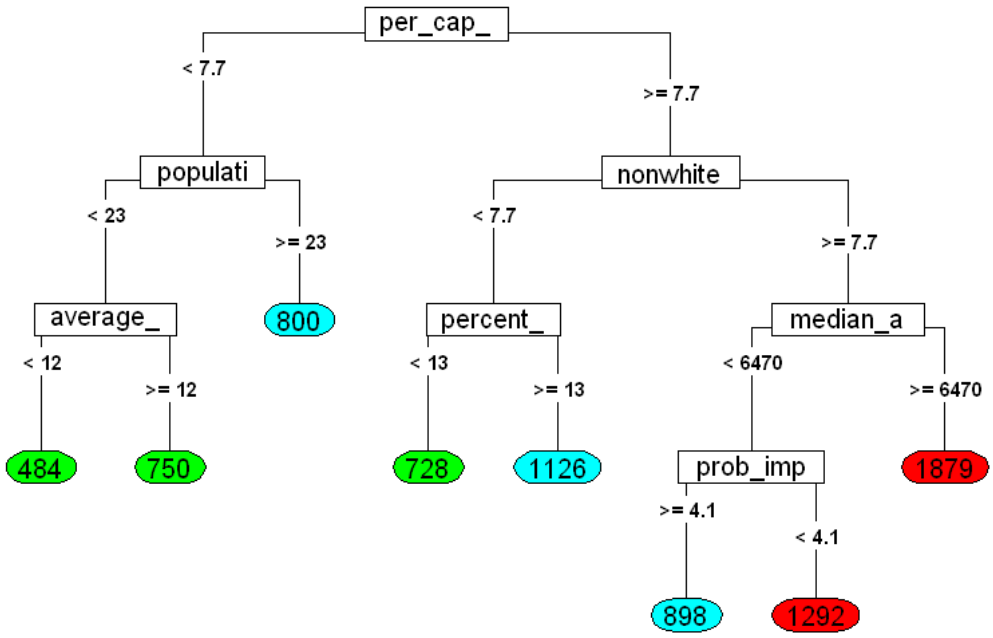
```
      per_cap_exp_1960  < 11.8     to the left,  agree=0.857, adj=0.333, (0
split)
      laborforce_part   < 57.4     to the left,  agree=0.857, adj=0.333, (0
split)

Node number 8: 9 observations
  mean=484.1111, MSE=6535.654

Node number 9: 3 observations
  mean=749.6667, MSE=8773.556

Node number 12: 6 observations
  mean=727.5, MSE=18214.92

Node number 13: 4 observations
  mean=1126, MSE=16790

Node number 14: 11 observations,    complexity param=0.05730143
  mean=1148.455, MSE=64295.7
  left son=28 (4 obs) right son=29 (7 obs)
  Primary splits:
      prob_imprisonment < 4.10995  to the right, improve=0.5574910, (0 missing)
      low_income        < 18.1     to the left,  improve=0.3556670, (0 missing)
      median_assets     < 5870     to the right, improve=0.3556670, (0 missing)
      laborforce_part   < 53.95    to the left,  improve=0.2822393, (0 missing)
      unemp_adult       < 3.7      to the right, improve=0.2767233, (0 missing)
  Surrogate splits:
      laborforce_part < 52       to the left,  agree=0.818, adj=0.50, (0 split)
      nonwhite        < 13.25    to the right, agree=0.818, adj=0.50, (0 split)
      unemp_adult     < 3.7      to the right, agree=0.818, adj=0.50, (0 split)
      time_served     < 22.64905 to the left,  agree=0.818, adj=0.50, (0 split)
      southern_state  < 0.5      to the right, agree=0.727, adj=0.25, (0 split)

Node number 15: 3 observations
  mean=1878.667, MSE=21040.22

Node number 28: 4 observations
  mean=898, MSE=7056.5

Node number 29: 7 observations
  mean=1291.571, MSE=40677.1
```

One of the interesting observations looking at the variable importance is that the model splits on per_cap_exp_1960 and doesn't include per_cap_exp_1959 despite it being listed as being more important. It also doesn't use prob_imprisonment, low_income, percent_young_male, or average_ed despite those variables having higher importance than some of the variables listed. I believe this is largely due to colinearity between those variables and the

variables that were selected. Multicollinearity doesn't impact the prediction performance of the model. This is fairly intuitive but I found a good explanation of the why here: https://medium.com/@manepriyanka48/multicollinearity-in-tree-based-models-b971292db140. "If two features are corelated, any one of the feature will be selected for splitting based on score. So if one out of two is selected, the 2nd one will not be selected for next level of tree building as the variance or impurity is already explained by the 1st feature."

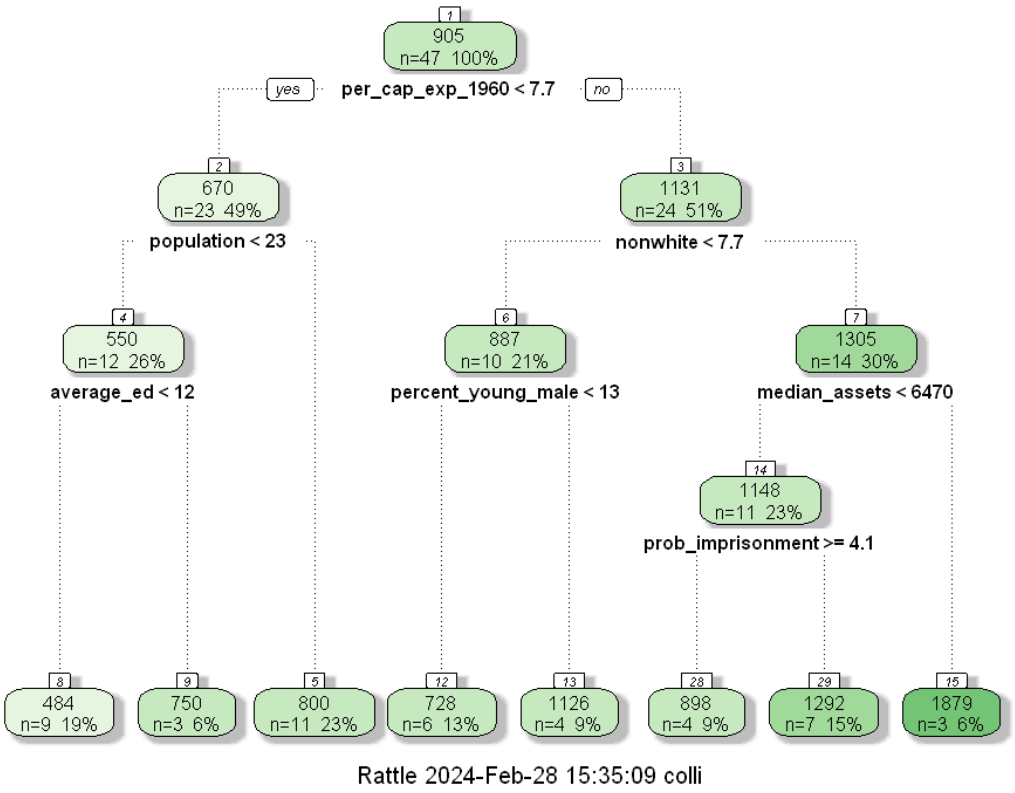
Next, I am going to plot the tree. I found multiple ways to do this including using the built in rpart.plot() function. I'm going to use two of the "fancier" options from the rpart.plot and rattle libraries to see which produces a nicer plot.

[7]:
```
# Set the plot size
options(repr.plot.width=7, repr.plot.height=5)

# Plot the tree
prp(crime_tree, type = 5, branch = 1, box.palette=c("green", "cyan", "red"),␣
 ↪split.cex = .75)
```



[8]:
```
# Set the plot size
options(repr.plot.width=8, repr.plot.height=6)

# Create the plot
```

```
fancyRpartPlot(crime_tree)
```



I'll hold off on any qualitative takeaways from the plots until the end of the document but they both worked well. prp() has a ton of different options to create the plot exactly as you want it but both get the job done.

Lastly, I am going to look at the cptable element of the crime_tree generated by rpart. The cptable provides a brief summary of the overall fit of the model. The table is printed from the smallest tree (no splits) to the largest tree. The "CP" column lists the values of the complexity parameter, the number of splits is listed under "nsplit", and the column "xerror" contains cross-validated classification error rates; the standard deviation of the cross-validation error rates are in the "xstd" column.

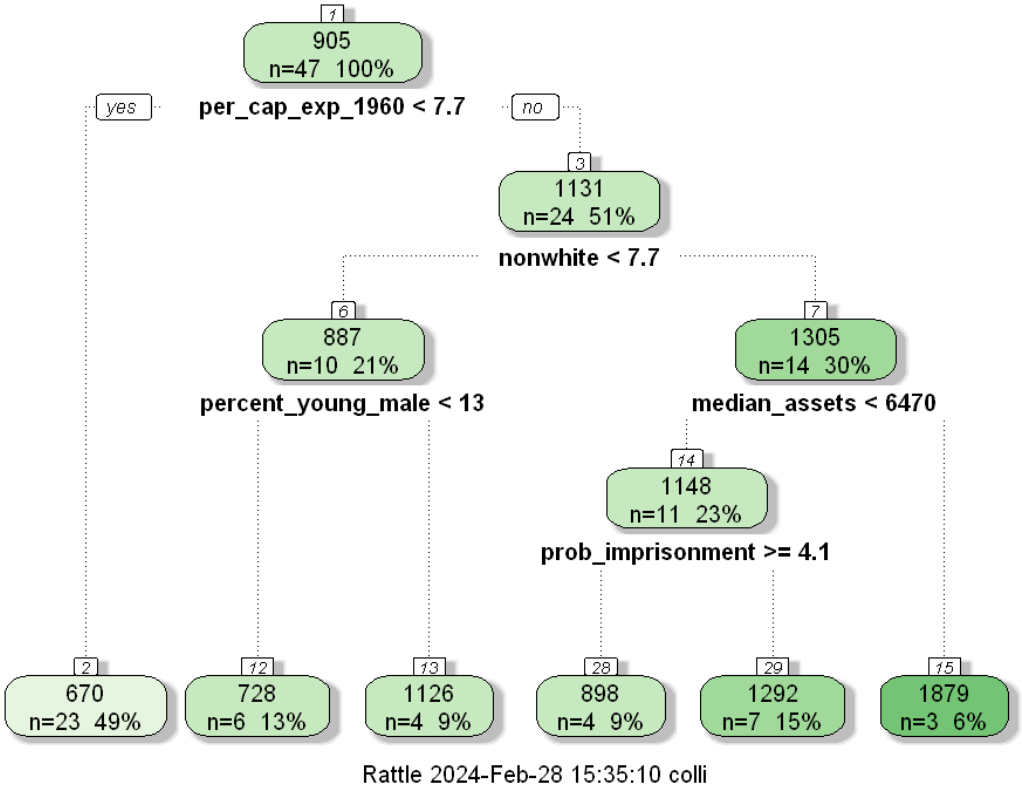
[9]:
```
crime_tree$cptable
```

CP	nsplit	rel error	xerror	xstd
0.36296293	0	1.0000000	1.062718	0.2641620
0.16540024	1	0.6370371	1.131172	0.2744131
0.05730143	3	0.3062366	1.292726	0.3268601
0.05538867	4	0.2489352	1.152379	0.3190650
0.05173165	5	0.1935465	1.178961	0.3179641
0.02305931	6	0.1418148	1.172783	0.3249076
0.01000000	7	0.1187555	1.146689	0.3351339

Normally, we select a tree size that minimizes the cross-validated error (xerror). That would actually mean pruning all of the branches and just being left with the root node… This is a pretty good indicator that the CART model isn't a good fit for this analysis.

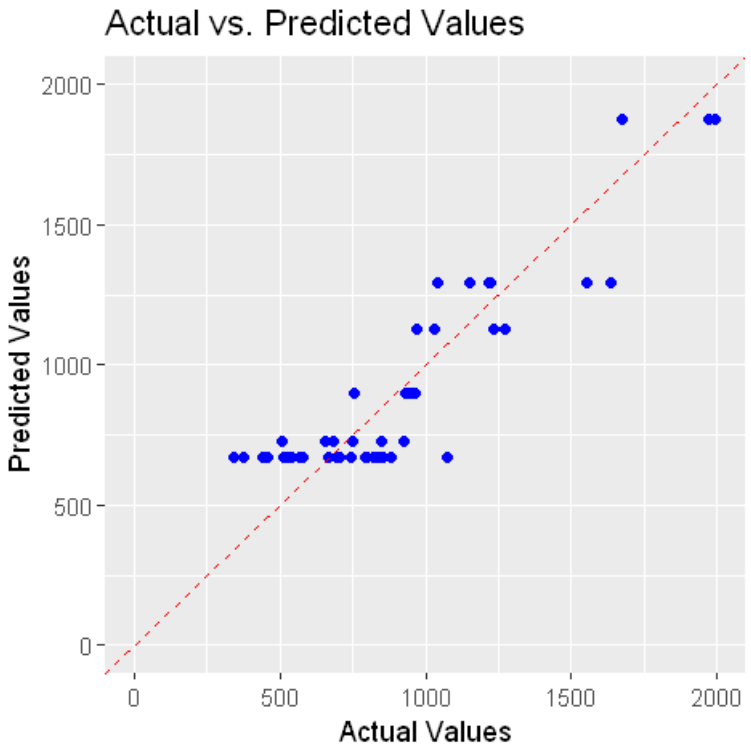
Step 4 - Prune the Tree Next I am going to prune the tree, since we used the entire set of data to train the model I can't calculate the estimation error to see if each branch increases error. Instead, I am going to prune the tree based on setting the cp threshold. To perform pruning, you normally choose a cp value that balances model complexity (number of splits) with predictive accuracy (xerror). The optimal cp value is typically chosen as the one associated with the smallest tree within one standard error of the minimum xerror, however this isn't possible since that value corresponds with the root node.

[10]:
```
pruned_crime_tree <- prune(crime_tree, cp = 0.0525)
fancyRpartPlot(pruned_crime_tree)
```



Step 5 - Interpret the Results Now that I have the final model I am going to spend some time analyzing it's overall performance. I want to look at how accurately it predicted the crime rates both graphically and using the R^2 value with and without cross-validation. I'll start by plotting the predicted values vs. the actual values.

[11]:
```
# Get predicted values from the model
pred <- predict(pruned_crime_tree)

# Plot the predicted values vs the actual values for crimes per million
plot_data <- data.frame(Actual = crime_data$crimes_per_million, Predicted =␣
↪pred)
options(repr.plot.width=4, repr.plot.height=4)

# Create a scatter plot
ggplot(plot_data, aes(x = Actual, y = Predicted)) +
    geom_point(color = "blue") +
    geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +␣
↪ # Add a line of equality
    labs(x = "Actual Values", y = "Predicted Values", title = "Actual vs.␣
↪Predicted Values") +
    xlim(0, 2000) +
    ylim(0, 2000)
```



This graph shows how the different final values are distributed among the actual values. You can distinctly see the 6 different possible predicted values and the distribution of actual values they cover. Next I'll calculate the R-Squared value using the training data.

[12]:
```
# Calculate the R-Squared and RMSE value
SSR <- sum((pred - crime_data$crimes_per_million)^2)
SST <- sum((crime_data$crimes_per_million -␣
 ↪mean(crime_data$crimes_per_million))^2)
R2 <- 1 - SSR/SST

RMSE <- sqrt((SSR)/nrow(crime_data))
cat('R-Squared value for the training data:', round(R2,3), '\n')
cat('RMSE value for the training data:', RMSE)
```

```
R-Squared value for the training data: 0.806
RMSE value for the training data: 168.3322
```

This shows a reasonable fit for the training dataset, I suspect it is overfitting so next I'll use cross validation to calculate the R-Squared value again.

[13]:
```
# 5-fold cross-validation
# Set the parameters for the model
ctrl <- trainControl(method = "cv", number = 5)
rpart_params <- list(split = "anova", minsplit = 9)

# Perform cross-validation
cv_model <- train(crimes_per_million ~ ., data = crime_data, method = "rpart",␣
 ↪trControl = ctrl, tuneLength = 10, parms = rpart_params)

# Get R^2 and RMSE values
cv_model$results
```

```
Warning message in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
trainInfo, :
"There were missing values in resampled performance measures."
```

cp	RMSE	Rsquared	MAE	RMSESD	RsquaredSD	MAESD
0.00000000	331.2935	0.30233684	270.2860	115.31931	0.27902045	95.01692
0.04032921	331.0697	0.30189365	273.3767	115.02224	0.27943322	99.56268
0.08065843	339.4182	0.27995110	278.1480	110.66839	0.30112875	97.48590
0.12098764	347.2662	0.21832683	282.2159	95.60076	0.22825680	90.24611
0.16131686	344.5430	0.22314361	273.8537	93.75588	0.22508572	85.42627
0.20164607	344.5430	0.22314361	273.8537	93.75588	0.22508572	85.42627
0.24197529	344.5430	0.22314361	273.8537	93.75588	0.22508572	85.42627
0.28230450	344.5430	0.22314361	273.8537	93.75588	0.22508572	85.42627
0.32263372	366.6472	0.13957395	290.9037	111.38682	0.14488680	92.09223
0.36296293	377.8427	0.07093795	304.8309	92.17892	0.05677036	69.12573

The first thing I noticed is the warning message that is provided. It indicates that there were missing values in the resampled performance measures during the cross-validation process. This warning typically occurs when one or more folds of the cross-validation procedure result in missing performance metrics. One possible reason would be if it created degenerate folds which happens when the data is small and one or more of the folds potentially has empty classes. With how small the dataset is this is definitely possible.

My final model pruned to cp values at or above 0.0525 which lies between the second and third row in the table. Looking at the R-Squared value it is ~ 0.30 which is much lower than the R-Squared value calculated on the training data. Comparing the RMSE values it also shows much higher values in the cross-validated model vs. the model that only used training data. **This suggest significant overfitting.**

Question 10.1 - Part B

> Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

For part B I am going to create a Random Forest model using the randomForest package and then evaluate it's performance. I plan to follow the steps below: 1) Create Model 2) Analyze Model Performance 3) Compare model to CART model from part A

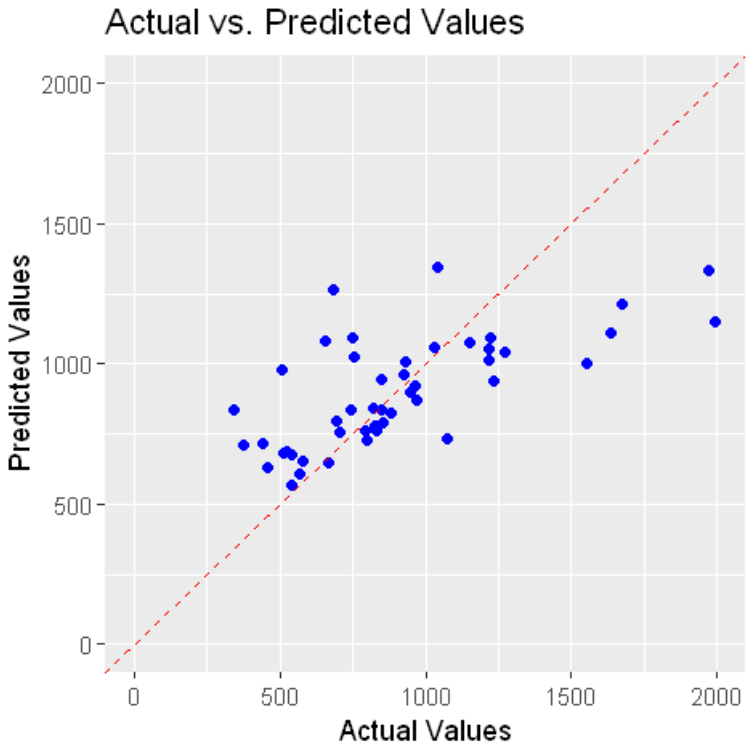
Step 1 - Create Model There are a lot of parameters that can be set for the random forest model which can be found here: https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.1/topics/randomForest. I played around with several different parameters, specifically the mtry, nodesize, and maxnodes. The mtry parameter is the number of variables randomly sampled as candidates at each split which defaults to p/3 for regression. This would be 5 for our sample since we have 15 dependent variables. The nodesize determines the minimum size of terminal nodes which defaults to 5 for regression. The maxnodes paramater determines the maximum number of terminal nodes the trees in the forest can have.

[14]:
```
crime_rforest <- randomForest(crimes_per_million ~ ., data = crime_data, mtry =␣
↪3, nodesize = 3, maxnodes = 25, importance = TRUE)
crime_rforest
```

```
Call:
 randomForest(formula = crimes_per_million ~ ., data = crime_data,      mtry =␣
↪3, nodesize = 3, maxnodes = 25, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 83197.59
                    % Var explained: 43.17
```

Step 2 - Analyze the Model Looking at the result above the model is able to describe ~ 43% of the variance in the dependent variable which is the same thing as saying the R-Squared value is 0.4527. It also has a RMSE of ~288 (square root of Mean of squared residuals). Next I am going to plot the predicted values from the model vs. the actual values to visualize the models performance.

[15]:
```
# Get predicted values from the model
rf_pred <- predict(crime_rforest)

# Create a dataframe of predicted and actual values to plot
rf_plot_data <- data.frame(Actual = crime_data$crimes_per_million, Predicted =␣
↪rf_pred)

# Plot the predicted values vs the actual values for crimes per million
ggplot(rf_plot_data, aes(x = Actual, y = Predicted)) +
    geom_point(color = "blue") +
    geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +␣
↪ # Add a line of equality
    labs(x = "Actual Values", y = "Predicted Values", title = "Actual vs.␣
↪Predicted Values") +
    xlim(0, 2000) +
    ylim(0, 2000)
```



Next I am going to calculate the R-Squared and RMSE value manually, this is simply a check on

the value output from the model above.

[16]:
```
# Calculate the R-Squared value
rf_SSR <- sum((rf_pred - crime_data$crimes_per_million)^2)
rf_R2 <- 1 - rf_SSR/SST

# Calculate the RMSE
rf_RMSE <- sqrt((rf_SSR)/nrow(crime_data))

cat('R-Squared value for the Random Forest model built with training data:',␣
 ↪round(rf_R2,3), '\n')
cat('RMSE value for the Random Forest model built with training data:', rf_RMSE)
```

```
R-Squared value for the Random Forest model built with training data: 0.432
RMSE value for the Random Forest model built with training data: 288.4399
```

These values match the output from the model above, it's nice when things check out like that :)

Next I will calculate the R-Squared and RMSE using cross-validation.

[17]:
```
# 5-fold cross-validation
# Set the parameters for the model
rf_ctrl <- trainControl(method = "cv", number = 5)
rf_params <- list(nodesize = 3, maxnodes = 25)

# Perform cross-validation
rf_cv_model <- train(crimes_per_million ~ ., data = crime_data, method = "rf",␣
 ↪tuneLength = 10, trControl = ctrl, parms = rf_params)

# Print the model to view the results
print(rf_cv_model)
```

```
Random Forest

47 samples
15 predictors

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 37, 37, 38, 38, 38
Resampling results across tuning parameters:

  mtry  RMSE      Rsquared   MAE
   2    281.9872  0.4993885  213.7193
   3    277.6091  0.4811486  208.7167
   4    285.6385  0.4386798  213.7613
   6    283.2740  0.4389022  209.4042
   7    283.9091  0.4347107  209.4308
   9    288.3659  0.4166768  211.2063
```

```
10    289.1673  0.4234507  213.7562
12    293.6504  0.4043417  215.6539
13    295.1157  0.4048620  215.7577
15    293.7088  0.4144728  214.6193

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 3.
```

These results are very suprising, when looking at the R-Squared values for mtry = 3 in the cross validated model it is actually larger than the model trained on the full training dataset. The RMSE is slightly lower as well. Looking into this I did find that it isn't totally uncommon. During cross-validation, the model is trained multiple times on different subsets of the data, which can help prevent overfitting by penalizing overly complex models. As a result, the cross-validated model may generalize better to unseen data, leading to higher performance metrics. The metrics above show that **the Random Forest did perform better than the CART model and has less of an issue with overfitting.**

Question 10.2

> Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

One example of a situation where logistic regression would be appropriate is whether or not a student will get into a particular college. Some of the predictors you might use for the model would be grades, SAT/ACT/GRE score, extracurricular activities, race/ethnicity, and parents highest education level. If a model was created for different colleges it would provide a very interesting way to compare what each college prioritizes in admittance vs. other colleges.

Question 10.3

> 1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.
> 2. Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

To answer this question I am going to use the following steps:

1) Load and explore the dataset
2) Create training and testing datasets
3) Train and optimize the logistic regression model

4) Interpret model
5) Create threshold based on part 2 of the question

Step 1 - Load and Explore the Dataset The first thing I am going to do is to load the dataset and get some summary statistics to get a better idea of the data I am working with.

[18]:
```
# Load the German credit data and print the head of the table to make sure it␣
 ↪loaded correctly
credit_data <- read.table("germancredit.txt", header = FALSE)
head(credit_data)
```

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

In examining the dataset I started by looking at the description provided at the UC Irvine ML Repository where the data was originally taken from. It included a cost matrix that showed that the cost of classifying a customer as "good" when they are "bad" was 5x the cost of classifying a customer as "bad" when they are actually "good" which is in line with part 2 of this question. It also provided a description of all of the attributes and what each of their values means. Lastly, it provides baseline model performance for various classification algorithms. For logistic regression it has a baseline accuracy of 70% - 80.8% and precision of 64.242% - 78.042%. It also shows that a Random Forest Classification model is the best option for the dataset. I have provided a copy of the Variables Table below:

Variable Name	Role	Type	Demographic	Description	Units	Missing Values
Attribute1	Feature	Categorical		Status of existing checking account		no
Attribute2	Feature	Integer		Duration	months	no
Attribute3	Feature	Categorical		Credit history		no
Attribute4	Feature	Categorical		Purpose		no
Attribute5	Feature	Integer		Credit amount		no
Attribute6	Feature	Categorical		Savings account/bonds		no
Attribute7	Feature	Categorical	Other	Present employment since		no
Attribute8	Feature	Integer		Installment rate in percentage of disposable income		no
Attribute9	Feature	Categorical	Marital Status	Personal status and sex		no
Attribute10	Feature	Categorical		Other debtors / guarantors		no
Attribute11	Feature	Integer		Present residence since		no
Attribute12	Feature	Categorical		Property		no
Attribute13	Feature	Integer	Age	Age	years	no
Attribute14	Feature	Categorical		Other installment plans		no
Attribute15	Feature	Categorical	Other	Housing		no
Attribute16	Feature	Integer		Number of existing credits at this bank		no
Attribute17	Feature	Categorical	Occupation	Job		no
Attribute18	Feature	Integer		Number of people being liable to provide maintenance for		no
Attribute19	Feature	Binary		Telephone		no
Attribute20	Feature	Binary	Other	foreign worker		no
class	Target	Binary		1 = Good, 2 = Bad		no

Since the original file didn't include headers I am going to add them to the dataset below. I also considered changing the attribute values to what they represent (ie A11 -> '<0', A40 -> 'car') but decided it isn't necassary for this analysis. However, if this was a model I was planning on using frequently it could be worth it to help with interpretting the results.

[19]:
```
colnames(credit_data) <- c("exist_acct", "duration", "credit_hist", "purpose",␣
↪"credit_amt", "savings", "employ_len", "install_rate", "status_sex",␣
↪"debts", "resid_len", "prop", "age", "oth_inst", "house", "credit_cnt",␣
↪"job", "liab_maint", "phone", "foreign", "decision")
```

Next, I want to get some summary statistics about the dataset which I'll do below. I already know that their aren't any missing values from the table above so I won't check that but I will look at the distribution of the variables.

[20]:
```
summary(credit_data)
```

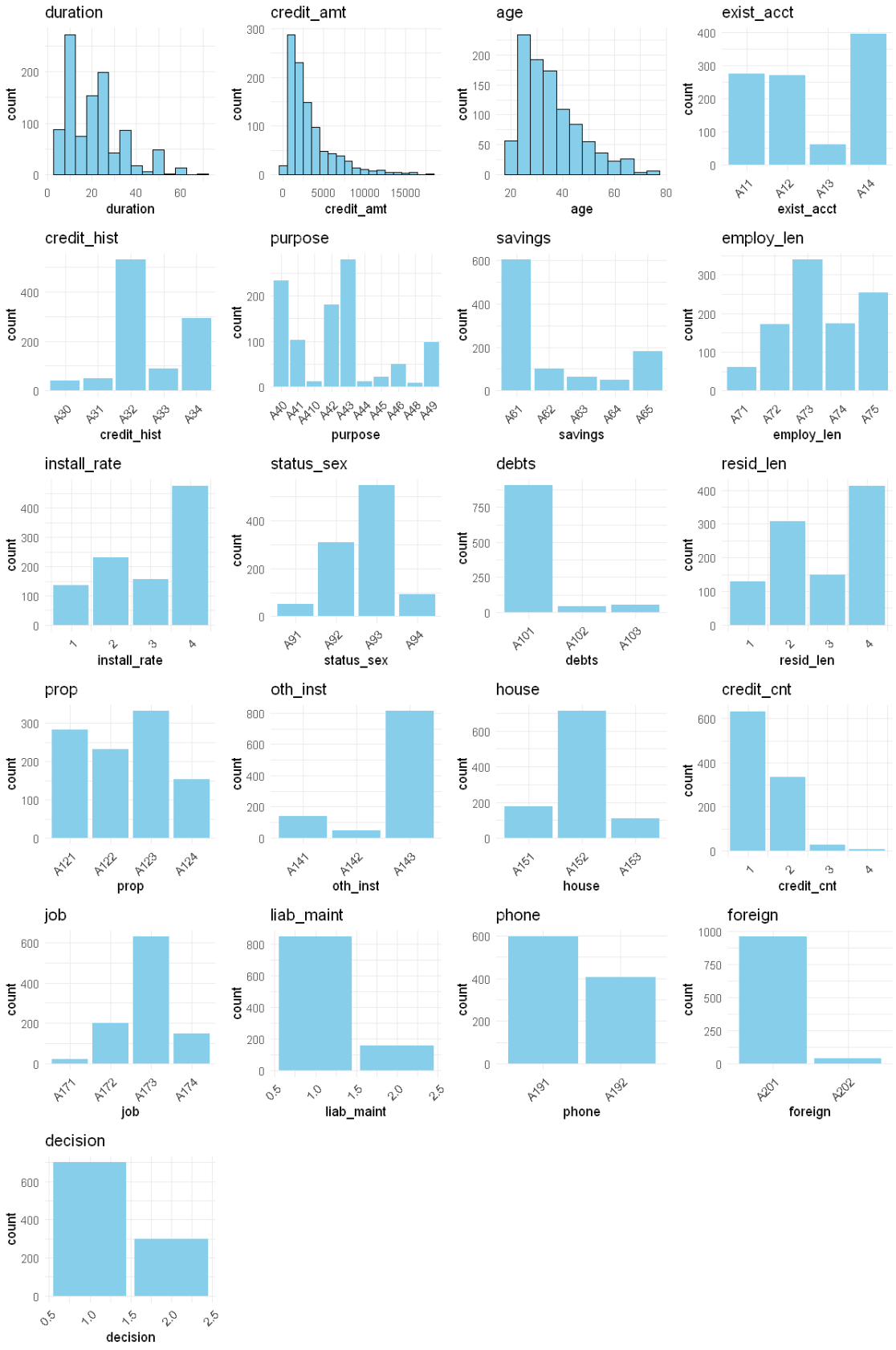
```
 exist_acct    duration     credit_hist    purpose      credit_amt    savings
 A11:274    Min.   : 4.0   A30: 40     A43    :280   Min.   :  250   A61:603
 A12:269    1st Qu.:12.0   A31: 49     A40    :234   1st Qu.: 1366   A62:103
 A13: 63    Median :18.0   A32:530     A42    :181   Median : 2320   A63: 63
 A14:394    Mean   :20.9   A33: 88     A41    :103   Mean   : 3271   A64: 48
            3rd Qu.:24.0   A34:293     A49    : 97   3rd Qu.: 3972   A65:183
            Max.   :72.0               A46    : 50   Max.   :18424
```

```
                                    (Other): 55
 employ_len  install_rate   status_sex  debts       resid_len       prop
 A71: 62    Min.   :1.000   A91: 50    A101:907   Min.   :1.000   A121:282
 A72:172    1st Qu.:2.000   A92:310    A102: 41   1st Qu.:2.000   A122:232
 A73:339    Median :3.000   A93:548    A103: 52   Median :3.000   A123:332
 A74:174    Mean   :2.973   A94: 92               Mean   :2.845   A124:154
 A75:253    3rd Qu.:4.000                         3rd Qu.:4.000
            Max.   :4.000                         Max.   :4.000

      age         oth_inst    house       credit_cnt      job
 Min.   :19.00   A141:139   A151:179   Min.   :1.000   A171: 22
 1st Qu.:27.00   A142: 47   A152:713   1st Qu.:1.000   A172:200
 Median :33.00   A143:814   A153:108   Median :1.000   A173:630
 Mean   :35.55                         Mean   :1.407   A174:148
 3rd Qu.:42.00                         3rd Qu.:2.000
 Max.   :75.00                         Max.   :4.000

   liab_maint     phone     foreign       decision
 Min.   :1.000   A191:596   A201:963   Min.   :1.0
 1st Qu.:1.000   A192:404   A202: 37   1st Qu.:1.0
 Median :1.000                         Median :1.0
 Mean   :1.155                         Mean   :1.3
 3rd Qu.:1.000                         3rd Qu.:2.0
 Max.   :2.000                         Max.   :2.0
```

The dataset includes 11 categorical variables, three binary variables, and seven integers. The high number of categorical/binary variables makes me think that a CART model would be a great option since the different classes would allow for easy interpretation. Next I am going to plot the distibution of values for each of the variables to visually inspect them.

[21]:
```r
options(repr.plot.width=10, repr.plot.height=15)

# Define the names of columns containing categorical variables
categorical_columns <- c("exist_acct", "credit_hist", "purpose", "savings",␣
 ↪"employ_len", "install_rate",
                         "status_sex", "debts", "resid_len", "prop",␣
 ↪"oth_inst", "house", "credit_cnt",
                         "job", "liab_maint", "phone", "foreign", "decision")

# Define the names of columns containing numerical variables
numerical_columns <- c("duration", "credit_amt", "age")

# Plot histograms for numerical columns
num_plots <- lapply(numerical_columns, function(col) {
  if(col == "credit_amt") {
    ggplot(credit_data, aes_string(x = col)) +
      geom_histogram(binwidth = 1000, fill = "skyblue", color = "black") +
```

```
      labs(title = col) +
      theme_minimal()
  } else {
    ggplot(credit_data, aes_string(x = col)) +
      geom_histogram(binwidth = 5, fill = "skyblue", color = "black") +
      labs(title = col) +
      theme_minimal()
  }
})

# Plot histograms for categorical columns
cat_plots <- lapply(categorical_columns, function(col) {
  ggplot(credit_data, aes_string(x = col)) +
    geom_bar(fill = "skyblue") +
    labs(title = col) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
})

# Combine plots for numerical and categorical columns
all_plots <- c(num_plots, cat_plots)

# Arrange plots in a grid with four plots in each row using cowplot
plot_grid(plotlist = all_plots, ncol = 4)
```


duration
count
duration
credit_amt
credit_amt
age
age
exist_acct
A11
A12
A13
A14
exist_acct
credit_hist
A30
A31
A32
A33
A34
credit_hist
purpose
A40
A41
A410
A42
A43
A44
A45
A46
A48
A49
purpose
savings
A61
A62
A63
A64
A65
savings
employ_len
A71
A72
A73
A74
A75
employ_len
install_rate
install_rate
status_sex
A91
A92
A93
A94
status_sex
debts
A101
A102
A103
debts
resid_len
resid_len
prop
A121
A122
A123
A124
prop
oth_inst
A141
A142
A143
oth_inst
house
A151
A152
A153
house
credit_cnt
credit_cnt
job
A171
A172
A173
A174
job
liab_maint
liab_maint
phone
A191
A192
phone
foreign
A201
A202
foreign
decision
decision

One thing worth noting is the large discrepency in the number of datapoints labeled as 'good' vs. the number labeled as 'bad' which is less than half as many. It could create bias in our model towards false positives since the model will have a larger number of 'good' classifications.

Step 2 - Create Training and Testing Sets Now that I have a good handle on what the data is, I'm going to partition it into training and testing sets. I am going to split the data into 70-30 for training-testing using the createDataPartition() function from the caret package. This function ensures that the split is stratified, meaning that the distribution of the dependent variable is preserved in both the training and testing sets.

[22]:
```
# Create an index for the split
split_index <- createDataPartition(credit_data$decision, p = 0.7, list = FALSE)

# Split the data into training and testing sets
training_data <- credit_data[split_index, ]
testing_data <- credit_data[-split_index, ]
cat("The training data has", nrow(training_data), "rows and the mean of the of␣
 ↪dependent variable 'decision' is", mean(training_data[,'decision']),"with a␣
 ↪standard deviation of", sd(training_data[,'decision']), "\n")
cat("The testing data has", nrow(testing_data), "rows and the mean of the of␣
 ↪dependent variable 'decision' is", mean(testing_data[,'decision']),"with a␣
 ↪standard deviation of", sd(testing_data[,'decision']))
```

```
The training data has 700 rows and the mean of the of dependent variable
'decision' is 1.302857 with a standard deviation of 0.4598225
The testing data has 300 rows and the mean of the of dependent variable
'decision' is 1.293333 with a standard deviation of 0.4560506
```

Step 3 - Train the Logistic Regression Model I am using the binomial family because our result 'decision' has a binomial distribution. The different options for the family to use in the regression model can be found here https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/family and the details about the inputs for the glm() function can be found here https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/glm. I had to set the dependent variable 'decision' to be a factor since the binomial family expects classification values of 0 and 1.

[23]:
```
credit_model <- glm(as.factor(decision) ~ ., family = binomial(link = "logit"),␣
 ↪data=training_data)
summary(credit_model)
```

```
Call:
glm(formula = as.factor(decision) ~ ., family = binomial(link = "logit"),
    data = training_data)
```

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2564  -0.6915  -0.3310   0.5833   2.6549

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)     7.982e-01  1.351e+00   0.591 0.554575
exist_acctA12  -2.159e-01  2.719e-01  -0.794 0.427135
exist_acctA13  -1.454e+00  4.628e-01  -3.143 0.001675 **
exist_acctA14  -1.805e+00  2.916e-01  -6.192 5.94e-10 ***
duration        3.659e-02  1.214e-02   3.015 0.002569 **
credit_histA31  3.826e-01  7.314e-01   0.523 0.600946
credit_histA32 -6.000e-01  5.616e-01  -1.068 0.285386
credit_histA33 -7.925e-01  6.102e-01  -1.299 0.194071
credit_histA34 -1.301e+00  5.651e-01  -2.302 0.021325 *
purposeA41     -1.729e+00  4.717e-01  -3.665 0.000247 ***
purposeA410    -1.534e+00  8.567e-01  -1.790 0.073401 .
purposeA42     -1.013e+00  3.320e-01  -3.049 0.002293 **
purposeA43     -1.030e+00  3.122e-01  -3.299 0.000970 ***
purposeA44     -7.406e-01  9.698e-01  -0.764 0.445081
purposeA45     -4.756e-01  6.669e-01  -0.713 0.475797
purposeA46      1.262e-01  4.844e-01   0.260 0.794518
purposeA48     -2.338e+00  1.310e+00  -1.785 0.074202 .
purposeA49     -8.586e-01  4.184e-01  -2.052 0.040162 *
credit_amt      1.362e-04  5.563e-05   2.447 0.014399 *
savingsA62     -3.859e-01  3.490e-01  -1.106 0.268883
savingsA63     -3.967e-01  4.958e-01  -0.800 0.423630
savingsA64     -3.806e+00  1.201e+00  -3.169 0.001527 **
savingsA65     -1.087e+00  3.282e-01  -3.311 0.000930 ***
employ_lenA72   2.917e-01  6.194e-01   0.471 0.637679
employ_lenA73   2.786e-01  6.050e-01   0.460 0.645245
employ_lenA74  -5.446e-01  6.465e-01  -0.842 0.399528
employ_lenA75   6.713e-02  5.975e-01   0.112 0.910551
install_rate    3.934e-01  1.160e-01   3.391 0.000697 ***
status_sexA92  -1.540e-01  4.719e-01  -0.326 0.744222
status_sexA93  -7.956e-01  4.573e-01  -1.740 0.081911 .
status_sexA94  -2.040e-01  5.481e-01  -0.372 0.709727
debtsA102       3.550e-02  4.962e-01   0.072 0.942954
debtsA103      -1.110e+00  4.828e-01  -2.298 0.021556 *
resid_len       8.683e-02  1.083e-01   0.802 0.422598
propA122        2.456e-01  3.018e-01   0.814 0.415794
propA123        2.019e-01  2.882e-01   0.701 0.483581
propA124        9.725e-01  5.698e-01   1.707 0.087869 .
age            -1.409e-02  1.159e-02  -1.216 0.224012
oth_instA142   -4.405e-01  5.574e-01  -0.790 0.429337
oth_instA143   -8.134e-01  2.989e-01  -2.721 0.006502 **
houseA152      -5.186e-01  2.973e-01  -1.744 0.081119 .
houseA153      -1.426e+00  6.421e-01  -2.221 0.026331 *
```

```
credit_cnt      -5.017e-02  2.345e-01  -0.214 0.830615
jobA172         -1.308e-01  8.272e-01  -0.158 0.874407
jobA173         -5.926e-02  8.063e-01  -0.073 0.941410
jobA174         -1.450e-01  8.070e-01  -0.180 0.857400
liab_maint       2.698e-01  3.085e-01   0.875 0.381744
phoneA192       -2.883e-01  2.668e-01  -1.081 0.279848
foreignA202     -1.224e+00  7.041e-01  -1.739 0.082066 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 858.57  on 699  degrees of freedom
Residual deviance: 588.69  on 651  degrees of freedom
AIC: 686.69

Number of Fisher Scoring iterations: 6
```

Looking at the results the first thing I notice is that all of the categorical variables with multiple levels were converted into dummy variables. For example, the 'job' variable has been divided into 'jobA172', 'jobA173', and 'jobA174' variables. I found that the glm function automatically creates dummy variables for any categorical variable with multiple levels (known as one-hot encoding). To deal with this I am going to add categorical variables for each of the dummy variables and redo the linear regression model.

[24]:
```r
# Perform One-Hot Encoding by converting the dataset to a matrix and then␣
 ↪convert it back to a dataset for the glm() function
credit_data_dummy <- model.matrix(~ ., data = credit_data)
credit_data_dummy <- as.data.frame(credit_data_dummy[,2:
 ↪ncol(credit_data_dummy)])

# Split the data into training and testing
training_data_dummy <- credit_data_dummy[split_index, ]
testing_data_dummy <- credit_data_dummy[-split_index, ]

# Create new logistic regression model
credit_model_dummy <- glm(as.factor(decision) ~ ., family = binomial(link =␣
 ↪"logit"), data=training_data_dummy)
summary(credit_model_dummy)
```

```
Call:
glm(formula = as.factor(decision) ~ ., family = binomial(link = "logit"),
    data = training_data_dummy)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
```

```
-2.2564  -0.6915  -0.3310   0.5833   2.6549

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)     7.982e-01  1.351e+00   0.591 0.554575
exist_acctA12  -2.159e-01  2.719e-01  -0.794 0.427135
exist_acctA13  -1.454e+00  4.628e-01  -3.143 0.001675 **
exist_acctA14  -1.805e+00  2.916e-01  -6.192 5.94e-10 ***
duration        3.659e-02  1.214e-02   3.015 0.002569 **
credit_histA31  3.826e-01  7.314e-01   0.523 0.600946
credit_histA32 -6.000e-01  5.616e-01  -1.068 0.285386
credit_histA33 -7.925e-01  6.102e-01  -1.299 0.194071
credit_histA34 -1.301e+00  5.651e-01  -2.302 0.021325 *
purposeA41     -1.729e+00  4.717e-01  -3.665 0.000247 ***
purposeA410    -1.534e+00  8.567e-01  -1.790 0.073401 .
purposeA42     -1.013e+00  3.320e-01  -3.049 0.002293 **
purposeA43     -1.030e+00  3.122e-01  -3.299 0.000970 ***
purposeA44     -7.406e-01  9.698e-01  -0.764 0.445081
purposeA45     -4.756e-01  6.669e-01  -0.713 0.475797
purposeA46      1.262e-01  4.844e-01   0.260 0.794518
purposeA48     -2.338e+00  1.310e+00  -1.785 0.074202 .
purposeA49     -8.586e-01  4.184e-01  -2.052 0.040162 *
credit_amt      1.362e-04  5.563e-05   2.447 0.014399 *
savingsA62     -3.859e-01  3.490e-01  -1.106 0.268883
savingsA63     -3.967e-01  4.958e-01  -0.800 0.423630
savingsA64     -3.806e+00  1.201e+00  -3.169 0.001527 **
savingsA65     -1.087e+00  3.282e-01  -3.311 0.000930 ***
employ_lenA72   2.917e-01  6.194e-01   0.471 0.637679
employ_lenA73   2.786e-01  6.050e-01   0.460 0.645245
employ_lenA74  -5.446e-01  6.465e-01  -0.842 0.399528
employ_lenA75   6.713e-02  5.975e-01   0.112 0.910551
install_rate    3.934e-01  1.160e-01   3.391 0.000697 ***
status_sexA92  -1.540e-01  4.719e-01  -0.326 0.744222
status_sexA93  -7.956e-01  4.573e-01  -1.740 0.081911 .
status_sexA94  -2.040e-01  5.481e-01  -0.372 0.709727
debtsA102       3.550e-02  4.962e-01   0.072 0.942954
debtsA103      -1.110e+00  4.828e-01  -2.298 0.021556 *
resid_len       8.683e-02  1.083e-01   0.802 0.422598
propA122        2.456e-01  3.018e-01   0.814 0.415794
propA123        2.019e-01  2.882e-01   0.701 0.483581
propA124        9.725e-01  5.698e-01   1.707 0.087869 .
age            -1.409e-02  1.159e-02  -1.216 0.224012
oth_instA142   -4.405e-01  5.574e-01  -0.790 0.429337
oth_instA143   -8.134e-01  2.989e-01  -2.721 0.006502 **
houseA152      -5.186e-01  2.973e-01  -1.744 0.081119 .
houseA153      -1.426e+00  6.421e-01  -2.221 0.026331 *
credit_cnt     -5.017e-02  2.345e-01  -0.214 0.830615
jobA172        -1.308e-01  8.272e-01  -0.158 0.874407
```

```
jobA173        -5.926e-02  8.063e-01  -0.073 0.941410
jobA174        -1.450e-01  8.070e-01  -0.180 0.857400
liab_maint      2.698e-01  3.085e-01   0.875 0.381744
phoneA192      -2.883e-01  2.668e-01  -1.081 0.279848
foreignA202    -1.224e+00  7.041e-01  -1.739 0.082066 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 858.57  on 699  degrees of freedom
Residual deviance: 588.69  on 651  degrees of freedom
AIC: 686.69

Number of Fisher Scoring iterations: 6
```

Comparing the results of the new model that utilizes the dummy variables to the original model it appears identical which is expected. Now that I am able to remove each variable I am going to use backward elimination to iteratively remove the least significant variable one at a time until all remaining variables are statistically significant. I'll do this by removing the variable with the highest p value, updating the model without that variable, and repeating until all the variables are above a threshold I set.

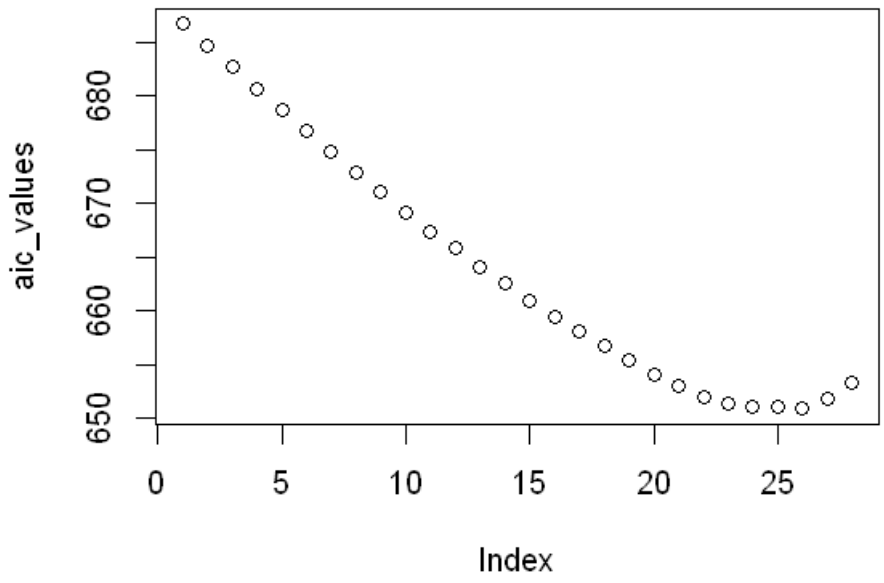
[25]:
```r
# Create vector for storing AIC values
aic_values <- c()

# Set p-value threshold
p_thre <- 0.05

while (TRUE) {
    # Get the AIC value for the current model and append it to the aic_values␣
 ↪vector
    model_aic <- AIC(credit_model_dummy)
    aic_values <- append(aic_values, model_aic)

    # Get max p-value name
    p_values <- summary(credit_model_dummy)$coefficients[, "Pr(>|z|)"][-1]
    max_p_value <- max(p_values)

    if (max_p_value > p_thre) {
        max_p_variable <- names(which(p_values == max_p_value))
        cat("Currently removing:", max_p_variable, "\n")

        # create updated formula without the variable
        formula <- as.formula(paste("as.factor(decision) ~ . -",␣
 ↪max_p_variable))
```

```
        # Update the model
        credit_model_dummy <- update(credit_model_dummy, formula)
    }

    else {
        break
    }
}
```

```
Currently removing: debtsA102
Currently removing: jobA173
Currently removing: employ_lenA75
Currently removing: credit_cnt
Currently removing: jobA174
Currently removing: jobA172
Currently removing: purposeA46
Currently removing: status_sexA92
Currently removing: status_sexA94
Currently removing: credit_histA31
Currently removing: employ_lenA72
Currently removing: employ_lenA73
Currently removing: propA123
Currently removing: propA122
Currently removing: savingsA63
Currently removing: resid_len
Currently removing: oth_instA142
Currently removing: purposeA44
Currently removing: purposeA45
Currently removing: savingsA62
Currently removing: liab_maint
Currently removing: phoneA192
Currently removing: exist_acctA12
Currently removing: propA124
Currently removing: age
Currently removing: foreignA202
Currently removing: purposeA48
```

Now that I have the model updated I'm intersted in what happened to the AIC (Akaike Information Criterion) value as each variable was removed. The AIC provides a balance between model goodness-of-fit and model complexity, aiming to select the model that best explains the data while penalizing for complexity. Lower AIC values indicate better model fit, with the model being both a good fit to the data and parsimonious (not overly complex). A more robust analysis would also include looking at the R-Square, RMSE, and BIC among others.

[26]:
```
options(repr.plot.width=5, repr.plot.height=4)
plot(aic_values)
```



Looking at the results of the AIC values it does appear that the model produced at the 24th/26th iteration has the best balance of goodness-of-fit and model complexity. When selecting the final model it's a balance between explainability and performance, so depending on the situation it might make sense to go with that model. However, for this homework I am going to move forward with the final model produced from the backward elimination procedure.

Step 4 - Interpret the Model Now that I have the model reduced to only have variables with a p-value above the significance threshold of 0.05 Iam ready to interpret the model. I'll start with a summary of the final mode.

[27]:
```
summary(credit_model_dummy)
```

```
Call:
glm(formula = as.factor(decision) ~ exist_acctA13 + exist_acctA14 +
    duration + credit_histA32 + credit_histA33 + credit_histA34 +
    purposeA41 + purposeA410 + purposeA42 + purposeA43 + purposeA49 +
    credit_amt + savingsA64 + savingsA65 + employ_lenA74 + install_rate +
    status_sexA93 + debtsA103 + oth_instA143 + houseA152 + houseA153,
    family = binomial(link = "logit"), data = training_data_dummy)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2556  -0.7047  -0.3471   0.6574   2.6301
```

```
Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)     7.713e-01  5.711e-01   1.351 0.176848
exist_acctA13  -1.285e+00  4.196e-01  -3.062 0.002197 **
exist_acctA14  -1.764e+00  2.474e-01  -7.127 1.02e-12 ***
duration        4.185e-02  1.148e-02   3.645 0.000267 ***
credit_histA32 -7.668e-01  3.642e-01  -2.105 0.035275 *
credit_histA33 -9.957e-01  4.694e-01  -2.121 0.033887 *
credit_histA34 -1.475e+00  3.940e-01  -3.744 0.000181 ***
purposeA41     -1.641e+00  4.457e-01  -3.683 0.000230 ***
purposeA410    -1.663e+00  8.153e-01  -2.040 0.041385 *
purposeA42     -7.908e-01  2.922e-01  -2.706 0.006808 **
purposeA43     -9.426e-01  2.721e-01  -3.465 0.000531 ***
purposeA49     -9.003e-01  3.857e-01  -2.335 0.019565 *
credit_amt      1.233e-04  5.098e-05   2.419 0.015567 *
savingsA64     -3.498e+00  1.106e+00  -3.163 0.001562 **
savingsA65     -1.031e+00  3.010e-01  -3.426 0.000612 ***
employ_lenA74  -7.401e-01  2.968e-01  -2.493 0.012656 *
install_rate    3.776e-01  1.095e-01   3.450 0.000562 ***
status_sexA93  -5.886e-01  2.180e-01  -2.700 0.006936 **
debtsA103      -1.140e+00  4.565e-01  -2.498 0.012482 *
oth_instA143   -7.767e-01  2.580e-01  -3.010 0.002611 **
houseA152      -6.890e-01  2.677e-01  -2.574 0.010065 *
houseA153      -9.324e-01  4.196e-01  -2.222 0.026262 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 858.57  on 699  degrees of freedom
Residual deviance: 609.27  on 678  degrees of freedom
AIC: 653.27

Number of Fisher Scoring iterations: 6
```

We can see that the model has now been trained only with variables with a significance < 0.05. The AIC value has also been decreased compared to the original model although not as far as it could have been reduced as seen in the plot above. I want to see the coefficients for my final model which I will output below.

[39]:
```
# Get model coefficients
print(credit_model_dummy$coefficients)
```

```
   (Intercept)  exist_acctA13  exist_acctA14       duration credit_histA32
  0.7713278405  -1.2849753747  -1.7635294799   0.0418524116  -0.7667749318
credit_histA33 credit_histA34     purposeA41    purposeA410     purposeA42
 -0.9957428533  -1.4748464953  -1.6414808117  -1.6630009816  -0.7907875768
```

```
   purposeA43     purposeA49     credit_amt     savingsA64     savingsA65
-0.9425797542  -0.9003451974   0.0001233166  -3.4983739271  -1.0313526820
employ_lenA74   install_rate  status_sexA93      debtsA103   oth_instA143
-0.7401182465   0.3775702623  -0.5886461045  -1.1404630191  -0.7766752938
    houseA152      houseA153
-0.6890182557  -0.9323711974
```

Next, I am going to use the testing data to see how well the model predicts the credit 'decision' variable. I found the information about evaluating regression models at https://www.geeksforgeeks.org/plotting-roc-curve-in-r-programming/ very useful for this.

[29]:
```
# Error metrix taken from geeksforgeeks
err_metric = function(GFGCM)
{
    GFGTN = GFGCM[1, 1]
    GFGRATE = GFGCM[2, 2]
    FP = GFGCM[1, 2]
    FN = GFGCM[2, 1]
    gfgPrecise = (GFGRATE)/(GFGRATE+FP)
    recall_score = (FP)/(FP+GFGTN)
    f1_score = 2*((gfgPrecise*recall_score)/(gfgPrecise+recall_score))
    accuracy_model = (GFGRATE+GFGTN)/(GFGRATE+GFGTN+FP+FN)
    False_positive_rate = (FP)/(FP+GFGTN)
    False_negative_rate = (FN)/(FN+GFGRATE)
    print(paste("GfgPrecise value of the model: ", round(gfgPrecise, 2)))
    print(paste("Accuracy of the model: ", round(accuracy_model, 2)))
    print(paste("Recall value of the model: ", round(recall_score, 2)))
    print(paste("False Positive rate of the model: ",␣
 ↪round(False_positive_rate, 2)))
    print(paste("False Negative rate of the model: ",␣
 ↪round(False_negative_rate, 2)))
    print(paste("f1 score of the model: ", round(f1_score, 2)))
}
```

I will start by creating a confusion matrix that I can use to get various model metrics from

[38]:
```
# Create predictions from the regression model on the testing data
pred_log_r <- predict(credit_model_dummy, testing_data_dummy, type = "response")
summary(pred_log_r)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.001117 0.078413 0.218238 0.314353 0.541337 0.950794
```

[31]:
```
# Convert predictions to yes-no answers (1 or 0) and output the confusion matrix
pred_int <- as.integer(pred_log_r > 0.5)
conf_matrix <- table(testing_data_dummy$decision, pred_int)
conf_matrix
```

```
   pred_int
      0   1
```

```
  1 170  42
  2  41  47
```

[32]:
```
# Get the error metrics
err_metric(conf_matrix)
```

```
[1] "GfgPrecise value of the model:  0.53"
[1] "Accuracy of the model:  0.72"
[1] "Recall value of the model:  0.2"
[1] "False Positive rate of the model:  0.2"
[1] "False Negative rate of the model:  0.47"
[1] "f1 score of the model:  0.29"
```

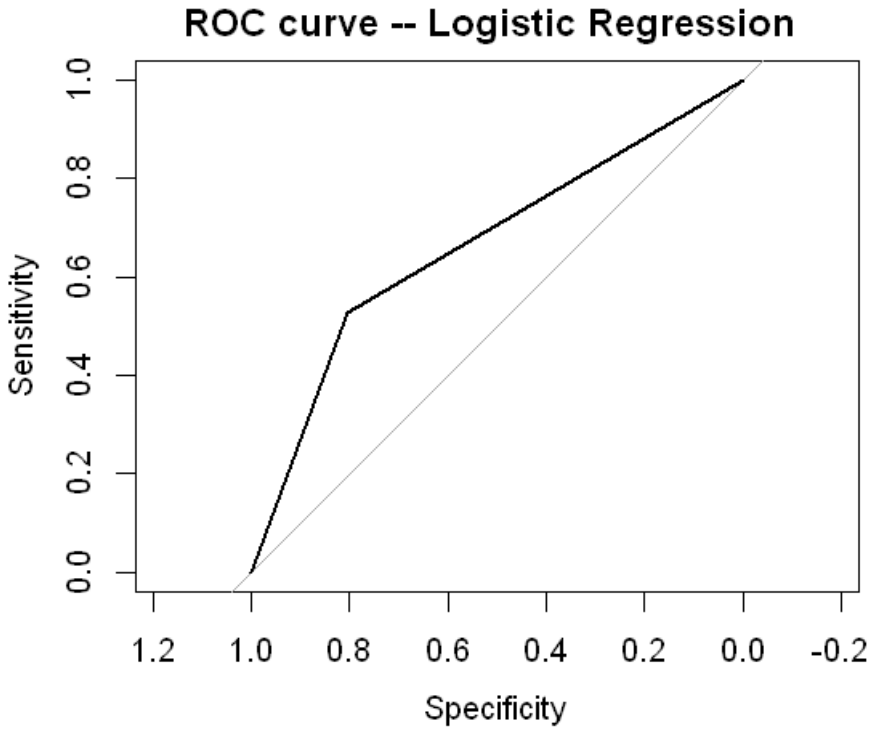
The model correctly classifies 72% of the points and has an f1 score of 0.29. The F1 score is a metric commonly used in binary classification tasks to evaluate the performance of the model. It is the harmonic mean of precision and recall, providing a single score that balances both measures. A value close to 1 is ideal. and recall:

Next I'll print out the ROC curve and get the AUC value.

[33]:
```
# Print out the ROC curve
roc_values <- roc(pred_int, testing_data_dummy$decision)
cat('The area under the curve is:', round(roc_values$auc, 3), '\n')
plot(roc_values, main="ROC curve -- Logistic Regression ")
```

```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
```

```
The area under the curve is: 0.667
```



Step 5 - Create threshold based on part 2 of the question I know that in this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Next, I am going to calculate the total cost of various threshold values to determine a good threshold probability. I am going to do that by iterating through multiple thresholds and calculating and storing the total cost for that threshold. The cost will have the formula:

$$Cost = (FalsePositives * 5) + FalseNegatives$$

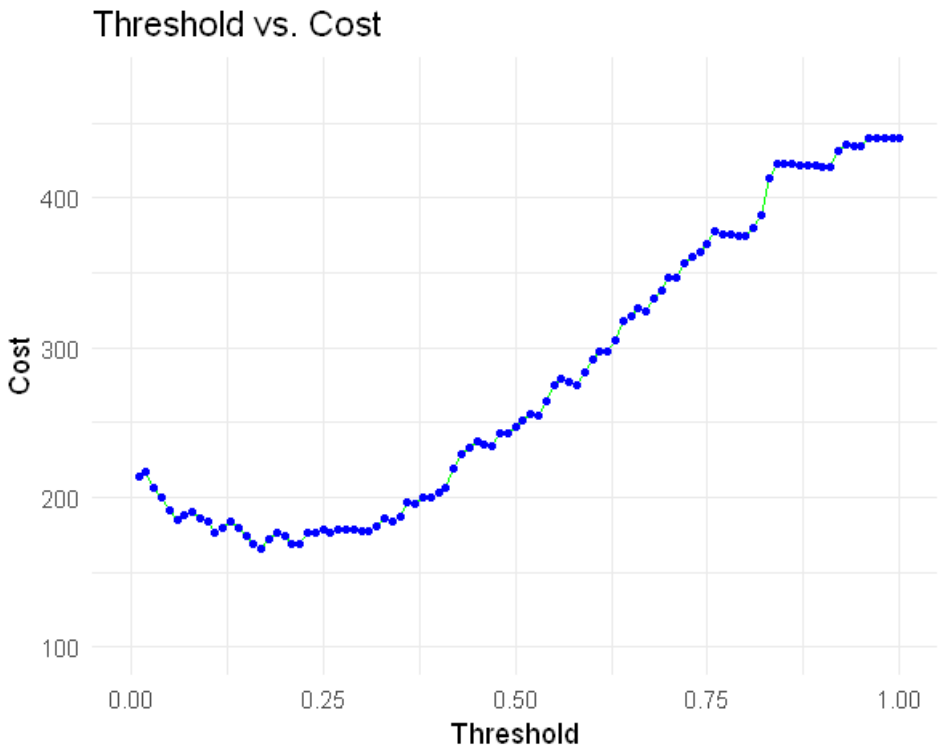
[34]:
```
# Loop through various threshold values and calculate the cost
thres_values <- seq(0.01, 1, by = 0.01)
cost_results <- c()

for (value in thres_values) {
    pred_int <- as.integer(pred_log_r > value)
    conf_matrix <- table(testing_data_dummy$decision, pred_int)

    if(ncol(conf_matrix) > 1) {
        false_neg <- conf_matrix[1,2]
        }
    else {
        false_neg <- 0
```

```
        }

    if(nrow(conf_matrix) > 1) {
        false_pos <- conf_matrix[2,1]
        }
    else {
        false_pos <- 0
        }

    cost <- (false_pos * 5) + false_neg
    cost_results <- append(cost_results, cost)
    }

min_cost <- which.min(cost_results)
cat('The threshold that produces the lowest cost is', min_cost/100, 'which has␣
 ↪a cost of', cost_results[min_cost], '\n')
cat('Compared to the threshold value of 0.5 which had a cost of',␣
 ↪cost_results[50], 'the new threshold reduced the total cost by',␣
 ↪cost_results[50] - cost_results[min_cost])
```

```
The threshold that produces the lowest cost is 0.17 which has a cost of 166
Compared to the threshold value of 0.5 which had a cost of 247 the new threshold
reduced the total cost by 81
```

Next I want to plot the cost vs. the threshold value to visualize the results

[35]:
```
# Create table of thresholds and cost
cost_table <- data.frame(Threshold = thres_values, Cost = cost_results)
ggplot(cost_table, aes(x = Threshold, y = Cost)) +
    geom_line(size = .5, color = "green") +
    geom_point(color = "blue", size = 1) +
    labs(x = "Threshold", y = "Cost", title = "Threshold vs. Cost") +
    xlim(0, 1) +
    ylim(100, 475) +
    theme_minimal()
```



The graph shows that values around 0.16 to 0.22 provide the lowest cost. Based on the information above, for my model setting a **threshold value of 0.17 provided the best model**. The confusion matrix for this final model is provided below.

[36]:
```
pred_int <- as.integer(pred_log_r > 0.2)
conf_matrix <- table(testing_data_dummy$decision, pred_int)
conf_matrix
err_metric(conf_matrix)
```

```
   pred_int
      0   1
  1 123  89
  2  17  71
```

```
[1] "GfgPrecise value of the model:  0.44"
[1] "Accuracy of the model:  0.65"
[1] "Recall value of the model:  0.42"
[1] "False Positive rate of the model:  0.42"
[1] "False Negative rate of the model:  0.19"
[1] "f1 score of the model:  0.43"
```