Week 4 Homework

Question 9.1 Using the same crime data set `uscrime.txt` as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2. You can use the R function `prcomp` for PCA. (Note that to first scale the data, you can include `scale. = TRUE` to scale as part of the PCA function. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

We will use the `prcomp` function in the `stats` library for PCA, and the library `factoextra` for visualization.

```
library(stats)
library(factoextra)
```

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

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
set.seed(42)
```

```
# load data
crime <- read.delim("../week 4 data-summer/data 9.1/uscrime.txt")
head(crime)
```

```
##      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
```

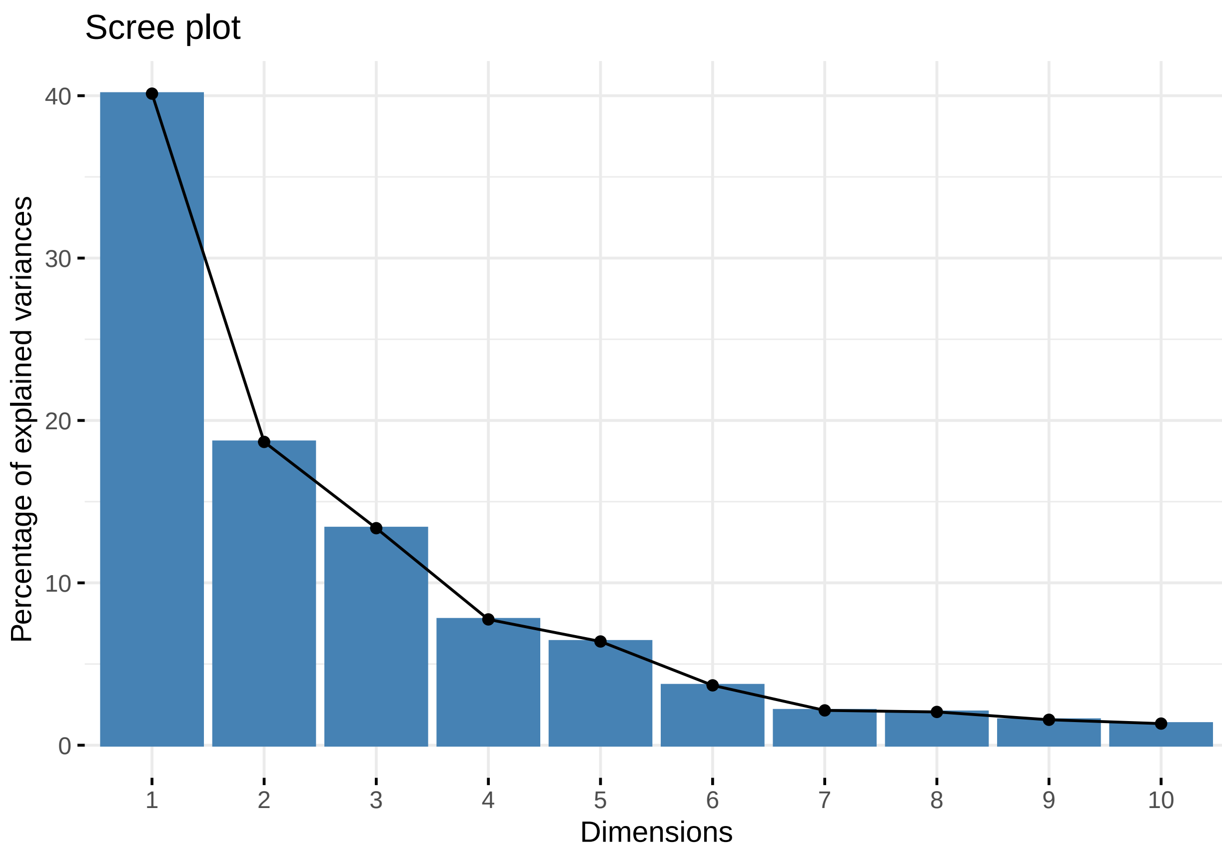
Using `prcomp`, apply PCA to the predictors of our `uscrime.txt` dataset.

```
# apply pca
crime_pca <- prcomp(crime[,1:15], center=T, scale = T)
crime_pca
```

```
## Standard deviations (1, .., p=15):
##  [1] 2.45335539 1.67387187 1.41596057 1.07805742 0.97892746 0.74377006
##  [7] 0.56729065 0.55443780 0.48492813 0.44708045 0.41914843 0.35803646
## [13] 0.26332811 0.24180109 0.06792764
##
## Rotation (n x k) = (15 x 15):
##                PC1         PC2           PC3         PC4         PC5
## M      -0.30371194  0.06280357  0.1724199946 -0.02035537 -0.35832737
## So     -0.33088129 -0.15837219  0.0155433104  0.29247181 -0.12061130
## Ed      0.33962148  0.21461152  0.0677396249  0.07974375 -0.02442839
## Po1     0.30863412 -0.26981761  0.0506458161  0.33325059 -0.23527680
## Po2     0.31099285 -0.26396300  0.0530651173  0.35192809 -0.20473383
## LF      0.17617757  0.31943042  0.2715301768 -0.14326529 -0.39407588
## M.F     0.11638221  0.39434428 -0.2031621598  0.01048029 -0.57877443
## Pop     0.11307836 -0.46723456  0.0770210971 -0.03210513 -0.08317034
## NW     -0.29358647 -0.22801119  0.0788156621  0.23925971 -0.36079387
## U1      0.04050137  0.00807439 -0.6590290980 -0.18279096 -0.13136873
## U2      0.01812228 -0.27971336 -0.5785006293 -0.06889312 -0.13499487
## Wealth  0.37970331 -0.07718862  0.0100647664  0.11781752  0.01167683
## Ineq   -0.36579778 -0.02752240 -0.0002944563 -0.08066612 -0.21672823
## Prob   -0.25888661  0.15831708 -0.1176726436  0.49303389  0.16562829
## Time   -0.02062867 -0.38014836  0.2235664632 -0.54059002 -0.14764767
##                 PC6         PC7         PC8         PC9        PC10        PC11
## M      -0.449132706 -0.15707378 -0.55367691  0.15474793 -0.01443093  0.39446657
## So     -0.100500743  0.19649727  0.22734157 -0.65599872  0.06141452  0.23397868
## Ed     -0.008571367 -0.23943629 -0.14644678 -0.44326978  0.51887452 -0.11821954
## Po1    -0.095776709  0.08011735  0.04613156  0.19425472 -0.14320978 -0.13042001
## Po2    -0.119524780  0.09518288  0.03168720  0.19512072 -0.05929780 -0.13885912
## LF      0.504234275 -0.15931612  0.25513777  0.14393498  0.03077073  0.38532827
## M.F    -0.074501901  0.15548197 -0.05507254 -0.24378252 -0.35323357 -0.28029732
## Pop     0.547098563  0.09046187 -0.59078221 -0.20244830 -0.03970718  0.05849643
## NW      0.051219538 -0.31154195  0.20432828  0.18984178  0.49201966 -0.20695666
## U1      0.017385981 -0.17354115 -0.20206312  0.02069349  0.22765278 -0.17857891
## U2      0.048155286 -0.07526787  0.24369650  0.05576010 -0.04750100  0.47021842
## Wealth -0.154683104 -0.14859424  0.08630649 -0.23196695 -0.11219383  0.31955631
## Ineq    0.272027031  0.37483032  0.07184018 -0.02494384 -0.01390576 -0.18278697
## Prob    0.283535996 -0.56159383 -0.08598908 -0.05306898 -0.42530006 -0.08978385
## Time   -0.148203050 -0.44199877  0.19507812 -0.23551363 -0.29264326 -0.26363121
##               PC12        PC13        PC14          PC15
## M       0.16580189  0.05142365  0.04901705 -0.0051398012
## So     -0.05753357  0.29368483 -0.29364512 -0.0084369230
## Ed      0.47786536 -0.19441949  0.03964277  0.0280052040
## Po1     0.22611207  0.18592255 -0.09490151  0.6894155129
## Po2     0.19088461  0.13454940 -0.08259642 -0.7200270100
## LF      0.02705134  0.27742957 -0.15385625 -0.0336823193
## M.F    -0.23925913 -0.31624667 -0.04125321 -0.0097922075
## Pop    -0.18350385 -0.12651689 -0.05326383 -0.0001496323
## NW     -0.36671707 -0.22901695  0.13227774  0.0370783671
## U1     -0.09314897  0.59039450 -0.02335942 -0.0111359325
## U2      0.28440496 -0.43292853 -0.03985736 -0.0073618948
## Wealth -0.32172821  0.14077972  0.70031840  0.0025685109
## Ineq    0.43762828  0.12181090  0.59279037 -0.0177570357
## Prob    0.15567100  0.03547596  0.04761011 -0.0293376260
## Time    0.13536989  0.05738113 -0.04488401 -0.0376754405
```

We can visualize the eigenvalues in a scree plot against the number of principal components used. This plot shows us the percentage of variances explained by each principal component.

```r
# scree plot
fviz_eig(crime_pca)
```



From the scree plot, we can see that after 4-5 principal components, the increase in variance explained becomes relatively small around 5 components. There is less and less of an advantage in using more components after the 5th component. We will proceed to build linear regression model with these 5 components.

```r
# create new crime dataframe with first 5 components
top5_components <- crime_pca$x[,1:5] # get first 5 cols of PCA components
PCAcrime <- as.data.frame(cbind(top5_components, crime[,16])) # create df with response column
PCAcrime
```

```
##           PC1         PC2         PC3         PC4          PC5   V6
## 1  -4.1992835 -1.09383120 -1.11907395  0.67178115  0.055283376  791
## 2   1.1726630  0.67701360 -0.05244634 -0.08350709 -1.173199821 1635
## 3  -4.1737248  0.27677501 -0.37107658  0.37793995  0.541345246  578
## 4   3.8349617 -2.57690596  0.22793998  0.38262331 -1.644746496 1969
## 5   1.8392999  1.33098564  1.27882805  0.71814305  0.041590320 1234
## 6   2.9072336 -0.33054213  0.53288181  1.22140635  1.374360960  682
## 7   0.2457752 -0.07362562 -0.90742064  1.13685873  0.718644387  963
## 8  -0.1301330 -1.35985577  0.59753132  1.44045387 -0.222781388 1555
## 9  -3.6103169 -0.68621008  1.28372246  0.55171150 -0.324292990  856
## 10  1.1672376  3.03207033  0.37984502 -0.28887026 -0.646056610  705
```

```
## 11  2.5384879 -2.66771358  1.54424656 -0.87671210 -0.324083561 1674
## 12  1.0065920 -0.06044849  1.18861346 -1.31261964  0.358087724  849
## 13  0.5161143  0.97485189  1.83351610 -1.59117618  0.599881946  511
## 14  0.4265556  1.85044812  1.02893477 -0.07789173  0.741887592  664
## 15 -3.3435299  0.05182823 -1.01358113  0.08840211  0.002969448  798
## 16 -3.0310689 -2.10295524 -1.82993161  0.52347187 -0.387454246  946
## 17 -0.2262961  1.44939774 -1.37565975  0.28960865  1.337784608  539
## 18 -0.1127499 -0.39407030 -0.38836278  3.97985093  0.410914404  929
## 19  2.9195668 -1.58646124  0.97612613  0.78629766  1.356288600  750
## 20  2.2998485 -1.73396487 -2.82423222 -0.23281758 -0.653038858 1225
## 21  1.1501667  0.13531015  0.28506743 -2.19770548  0.084621572  742
## 22 -5.6594827 -1.09730404  0.10043541 -0.05245484 -0.689327990  439
## 23 -0.1011749 -0.57911362  0.71128354 -0.44394773  0.689939865 1216
## 24  1.3836281  1.95052341 -2.98485490 -0.35942784 -0.744371276  968
## 25  0.2727756  2.63013778  1.83189535  0.05207518  0.803692524  523
## 26  4.0565577  1.17534729 -0.81690756  1.66990720 -2.895110075 1993
## 27  0.8929694  0.79236692  1.26822542 -0.57575615  1.830793964  342
## 28  0.1514495  1.44873320  0.10857670 -0.51040146 -1.023229895 1216
## 29  3.5592481 -4.76202163  0.75080576  0.64692974  0.309946510 1043
## 30 -4.1184576 -0.38073981  1.43463965  0.63330834 -0.254715638  696
## 31 -0.6811731  1.66926027 -2.88645794 -1.30977099 -0.470913997  373
## 32  1.7157269 -1.30836339 -0.55971313 -0.70557980  0.331277622  754
## 33 -1.8860627  0.59058174  1.43570145  0.18239089  0.291863659 1072
## 34  1.9526349  0.52395429 -0.75642216  0.44289927  0.723474420  923
## 35  1.5888864 -3.12998571 -1.73107199 -1.68604766  0.665406182  653
## 36  1.0709414 -1.65628271  0.79436888 -1.85172698  0.020031154 1272
## 37 -4.1101715  0.15766712  2.36296974 -0.56868399 -2.469679496  831
## 38 -0.7254706  2.89263339 -0.36348376 -0.50612576  0.028157162  566
## 39 -3.3451254 -0.95045293  0.19551398 -0.27716645  0.487259213  826
## 40 -1.0644466 -1.05265304  0.82886286 -0.12042931 -0.645884788 1151
## 41  1.4933989  1.86712106  1.81853582 -1.06112429  0.009855774  880
## 42 -0.6789284  1.83156328 -1.65435992  0.95121379  2.115630145  542
## 43 -2.4164258 -0.46701087  1.42808323  0.41149015 -0.867397522  823
## 44  2.2978729  0.41865689 -0.64422929 -0.63462770 -0.703116983 1030
## 45 -2.9245282 -1.19488555 -3.35139309 -1.48966984  0.806659622  455
## 46  1.7654525  0.95655926  0.98576138  1.05683769  0.542466034  508
## 47  2.3125056  2.56161119 -1.58223354  0.59863946 -1.140712406  849
```

```r
# build pca linear regression model
model1 <- lm(V6~., data=PCAcrime)
summary(model1)
```

```
##
## Call:
## lm(formula = V6 ~ ., data = PCAcrime)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -420.79 -185.01   12.21  146.24  447.86
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      35.59  25.428  < 2e-16 ***
## PC1            65.22      14.67   4.447 6.51e-05 ***
```

```
## PC2           -70.08      21.49  -3.261  0.00224 **
## PC3            25.19      25.41   0.992  0.32725
## PC4            69.45      33.37   2.081  0.04374 *
## PC5          -229.04      36.75  -6.232 2.02e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 244 on 41 degrees of freedom
## Multiple R-squared:  0.6452, Adjusted R-squared:  0.6019
## F-statistic: 14.91 on 5 and 41 DF,  p-value: 2.446e-08
```

Now we have to unscale our coefficients, expressing our principal components in terms of the original variables.

Let's see our PCA coefficients and intercept first:

```
# lr coefficients using pca components
scaled_coefficients <- model1$coefficients[2:6]
scaled_coefficients
```

```
##       PC1        PC2        PC3        PC4        PC5
##   65.21593  -70.08312   25.19408   69.44603 -229.04282
```

```
# lr intercept using pca components
scaled_intercept <- model1$coefficients[1]
scaled_intercept
```

```
## (Intercept)
##    905.0851
```

We also need to obtain the eigenvectors for each variable.

```
# implied regression coefficients for x_j from pca
eigenvectors <- crime_pca$rotation[,1:5]%*%scaled_coefficients
eigenvectors
```

```
##              [,1]
## M       60.794349
## So      37.848243
## Ed      19.947757
## Po1    117.344887
## Po2    111.450787
## LF      76.254902
## M.F    108.126558
## Pop     58.880237
## NW      98.071790
## U1       2.866783
## U2      32.345508
## Wealth  35.933362
## Ineq    22.103697
## Prob   -34.640264
## Time    27.205022
```

Then, we can transform our scaled coefficients to obtain the unscaled coefficients and intercept.

```r
# unscaled coefficients
unscaled_coefficients <- eigenvectors/crime_pca$scale
unscaled_coefficients
```

```
##                 [,1]
## M        4.837374e+01
## So       7.901922e+01
## Ed       1.783120e+01
## Po1      3.948484e+01
## Po2      3.985892e+01
## LF       1.886946e+03
## M.F      3.669366e+01
## Pop      1.546583e+00
## NW       9.537384e+00
## U1       1.590115e+02
## U2       3.829933e+01
## Wealth   3.724014e-02
## Ineq     5.540321e+00
## Prob    -1.523521e+03
## Time     3.838779e+00
```

```r
# unscaled intercept
unscaled_intercept <- scaled_intercept - sum(eigenvectors
                                             *crime_pca$center
                                             /crime_pca$scale)
unscaled_intercept
```

```
## (Intercept)
##   -5933.837
```

Next, let's compare our PCA model to our base linear regression model from 8.2.

As a refresher, here is the model and the metrics from 8.2:

```r
# build base linear regression model
model2 <- lm(Crime~., data=crime)
model2
```

```
##
## Call:
## lm(formula = Crime ~ ., data = crime)
##
## Coefficients:
## (Intercept)            M           So           Ed          Po1          Po2
##  -5.984e+03    8.783e+01   -3.803e+00    1.883e+02    1.928e+02   -1.094e+02
##          LF          M.F          Pop           NW           U1           U2
##  -6.638e+02    1.741e+01   -7.330e-01    4.204e+00   -5.827e+03    1.678e+02
##      Wealth         Ineq         Prob         Time
##   9.617e-02    7.067e+01   -4.855e+03   -3.479e+00
```

```r
# r-squared of base linear regression model
print(sprintf("base lm r-squared: %0.3f", summary(model2)$r.squared))
```

```
## [1] "base lm r-squared: 0.803"
```

```r
# rmse of base linear regression model
print(sprintf("base lm rmse: %0.3f", sigma(model2)))
```

```
## [1] "base lm rmse: 209.064"
```

And the model and metrics of the PCA model for this week.

```r
# unscaled pca linear regression model
y_pred <- unscaled_intercept + as.matrix(crime[,1:15])%*%unscaled_coefficients

# rsquared of pca linear regression model
rss <- sum((y_pred-crime[,16])^2)
tss <- sum((crime[,16]-mean(crime[,16]))^2)
rsquared <- 1-rss/tss
print(sprintf("pca lm r-squared: %0.3f", rsquared))
```

```
## [1] "pca lm r-squared: 0.645"
```

```r
# rmse of pca linear regression model
rmse <- sqrt(mean((crime[,16]-y_pred)^2))
print(sprintf("pca lm rmse: %0.3f", rmse))
```

```
## [1] "pca lm rmse: 227.913"
```

Using our PCA model to make a prediction on the sample data (given in 8.2):

```r
# create dataframe from sample input from 8.2
sample <- 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 = 0.04,
                     Time = 39.0)
```

```r
# make prediction with pca model
pred_pca <- unscaled_intercept + as.matrix(sample)%*%unscaled_coefficients
pred_pca
```

```
##          [,1]
## [1,] 1388.926
```

We can see that comparatively, the PCA linear model has a lower R^2 value than the base model (PCA: 0.645, Base: 0.803), and a higher RMSE compared to the base model (PCA: 227.913, Base: 209.064). However, remember that the base model was extremely over-fitted on the training data and resulted in a highly inaccurate prediction of 155. Then we removed predictors that we deemed insignificant and created a second model that ended up with a prediction of 1304. Our PCA model is more in line with this latter model with a prediction of 1388. Thus although our PCA model explains less of the data (lower R^2) and has a higher error (higher RMSE) compared to the base model, it is less over-fitted and generates a more accurate prediction.

Question 10.1 (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).

We are using the same dataset as the previous question so no need to re-import it.

```r
# load packages
library(rpart)
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

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

```r
# review the crime dataset
head(crime)
```

```
##      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
```

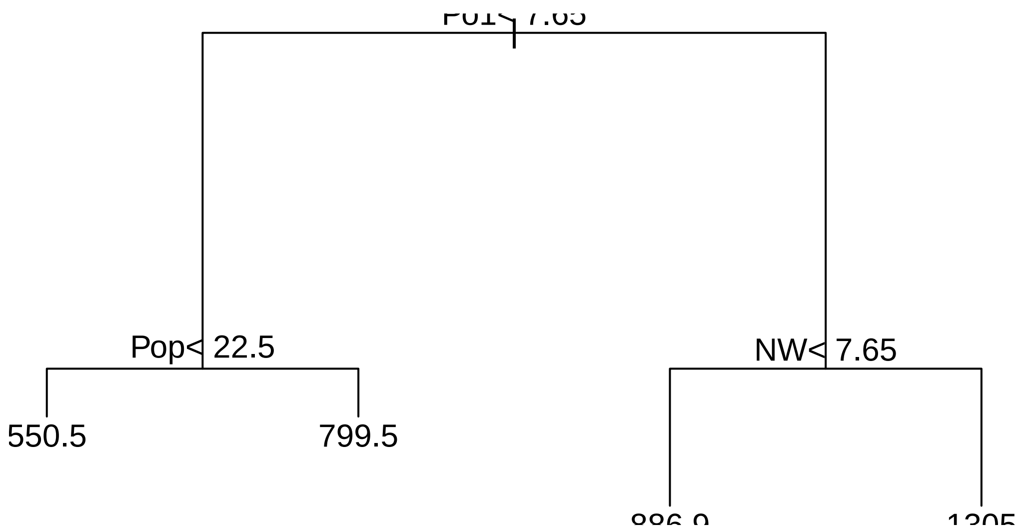
```
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

```
# build regression tree
set.seed(42)
tree_model <- rpart(Crime~., data=crime)
summary(tree_model)
```

```
## Call:
## rpart(formula = Crime ~ ., data = crime)
##   n= 47
##
##           CP nsplit rel error   xerror      xstd
## 1 0.36296293      0 1.0000000 1.044786 0.2605960
## 2 0.14814320      1 0.6370371 0.911051 0.1945477
## 3 0.05173165      2 0.4888939 1.012676 0.2303736
## 4 0.01000000      3 0.4371622 1.012402 0.2315240
##
## Variable importance
##    Po1    Po2 Wealth   Ineq   Prob      M     NW    Pop   Time     Ed     LF
##     17     17     11     11     10     10      9      5      4      4      1
##     So
##      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:
##       Po1    < 7.65      to the left,  improve=0.3629629, (0 missing)
##       Po2    < 7.2       to the left,  improve=0.3629629, (0 missing)
##       Prob   < 0.0418485 to the right, improve=0.3217700, (0 missing)
##       NW     < 7.65      to the left,  improve=0.2356621, (0 missing)
##       Wealth < 6240      to the left,  improve=0.2002403, (0 missing)
##   Surrogate splits:
##       Po2    < 7.2       to the left,  agree=1.000, adj=1.000, (0 split)
##       Wealth < 5330      to the left,  agree=0.830, adj=0.652, (0 split)
##       Prob   < 0.043598  to the right, agree=0.809, adj=0.609, (0 split)
##       M      < 13.25     to the right, agree=0.745, adj=0.478, (0 split)
##       Ineq   < 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:
##       Pop < 22.5      to the left,  improve=0.4568043, (0 missing)
##       M   < 14.5      to the left,  improve=0.3931567, (0 missing)
##       NW  < 5.4       to the left,  improve=0.3184074, (0 missing)
##       Po1 < 5.75      to the left,  improve=0.2310098, (0 missing)
##       U1  < 0.093     to the right, improve=0.2119062, (0 missing)
##   Surrogate splits:
##       NW   < 5.4       to the left,  agree=0.826, adj=0.636, (0 split)
##       M    < 14.5      to the left,  agree=0.783, adj=0.545, (0 split)
```

```
##       Time < 22.30055  to the left,  agree=0.783, adj=0.545, (0 split)
##       So   < 0.5       to the left,  agree=0.739, adj=0.455, (0 split)
##       Ed   < 10.85     to the right, agree=0.739, adj=0.455, (0 split)
##
## Node number 3: 24 observations,    complexity param=0.1481432
##   mean=1130.75, MSE=150173.4
##   left son=6 (10 obs) right son=7 (14 obs)
##   Primary splits:
##       NW   < 7.65      to the left,  improve=0.2828293, (0 missing)
##       M    < 13.05     to the left,  improve=0.2714159, (0 missing)
##       Time < 21.9001   to the left,  improve=0.2060170, (0 missing)
##       M.F  < 99.2      to the left,  improve=0.1703438, (0 missing)
##       Po1  < 10.75     to the left,  improve=0.1659433, (0 missing)
##   Surrogate splits:
##       Ed   < 11.45     to the right, agree=0.750, adj=0.4, (0 split)
##       Ineq < 16.25     to the left,  agree=0.750, adj=0.4, (0 split)
##       Time < 21.9001   to the left,  agree=0.750, adj=0.4, (0 split)
##       Pop  < 30        to the left,  agree=0.708, adj=0.3, (0 split)
##       LF   < 0.5885    to the right, agree=0.667, adj=0.2, (0 split)
##
## Node number 4: 12 observations
##   mean=550.5, MSE=20317.58
##
## Node number 5: 11 observations
##   mean=799.5455, MSE=16315.52
##
## Node number 6: 10 observations
##   mean=886.9, MSE=55757.49
##
## Node number 7: 14 observations
##   mean=1304.929, MSE=144801.8
```

```r
# plot regression tree
plot(tree_model)
text(tree_model)
```




```
# examine variable importance
tree_model$variable.importance
```

```
##       Po1       Po2    Wealth      Ineq      Prob         M        NW       Pop
## 2497521.7 2497521.7 1628818.5 1602212.0 1520230.6 1388627.8 1245883.8  661770.6
##      Time        Ed        LF        So
##  601906.0  569545.9  203872.5  161800.8
```

```
# find r-squared
ypred.tree <- predict(tree_model)
RSS <- sum((ypred.tree-crime$Crime)^2) # the residual sum of squares
TSS <- sum((mean(crime$Crime)-crime$Crime)^2)#the total sum of squares
print(sprintf("R-Squared = %0.3f",
              R.squared.tree<-1-(RSS/TSS)))
```

```
## [1] "R-Squared = 0.563"
```

Some observations:

1. Only 3 predictors were used in this regression tree: P01, Pop and NW
2. There are 4 leafs nodes and 2 branching points.
3. Each leaf contained 10-14 data points, more than 5% of the data. Structurally this tree is reasonable.

Question 10.1 (b)

For part (b), build a random forest model

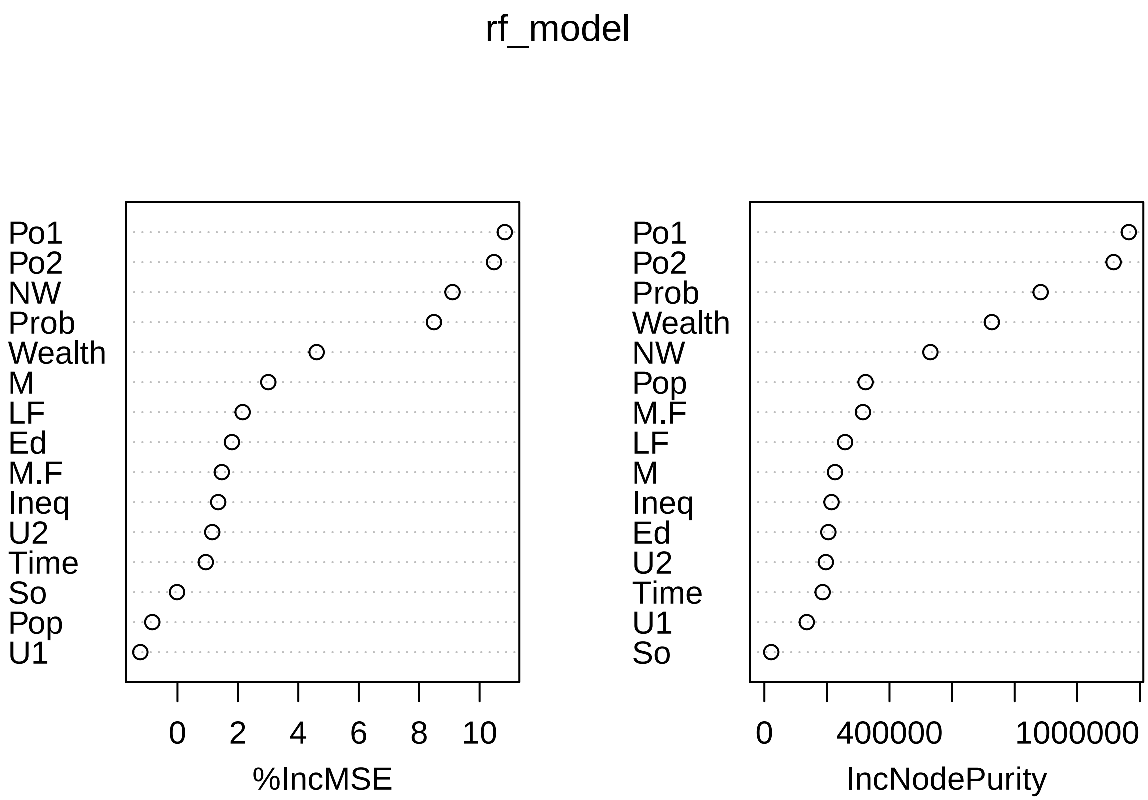
```r
# build random forest model
set.seed(42)
rf_model <- randomForest(Crime~., data=crime, keep.forest=T, importance=T)
rf_model
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = crime, keep.forest = T,      importance = T)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 87379.48
##                     % Var explained: 40.32
```

```r
# importance of each predictor
randomForest::importance(rf_model)
```

```
##            %IncMSE IncNodePurity
## M       3.00690191     225937.17
## So     -0.01356852      22096.35
## Ed      1.80268010     204621.10
## Po1    10.83366782    1164661.94
## Po2    10.47727967    1116121.11
## LF      2.15720728     258079.02
## M.F     1.46736557     315056.16
## Pop    -0.83229685     323553.45
## NW      9.10386424     530706.09
## U1     -1.22865150     135559.27
## U2      1.15212612     196488.67
## Wealth  4.60644846     726978.33
## Ineq    1.34905118     214692.60
## Prob    8.48888959     882981.35
## Time    0.93682829     186298.58
```

```r
# plot of importance predictors
varImpPlot(rf_model)
```




```
# find r-squared
ypred.RF <- predict(rf_model)
RSS_rf <- sum((ypred.RF-crime$Crime)^2) # the residual sum of squares
TSS_rf <- sum((mean(crime$Crime)-crime$Crime)^2)#the total sum of squares
print(sprintf("R-Squared of Random Forest Model = %0.3f",
              R.squared.RF <- 1-(RSS_rf/TSS_rf)))
```

```
## [1] "R-Squared of Random Forest Model = 0.403"
```

- Overall R-Squared = 0.403 which is very reasonable (all data points & all factors). The introduction of Randomness to the Random Forest model really helped in reducing over-fitting

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.

Logistic regression models can be used in identifying potential bank customers that will default on loans. Some potential predictors might include:

- Credit score
- Monthly income
- Loan amount
- Age of customer

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.

```
library(caret)
```

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

```
# load data
german <- read.table("../week 4 data-summer/data 10.3/germancredit.txt", sep=' ')
head(german)
```

```
##    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
```

```
# update response column
german$V21[german$V21==1]<-0
german$V21[german$V21==2]<-1
```

```
# train test split
rrow <- sample(1:nrow(german), as.integer(0.7*nrow(german), replace=F))
train <- german[rrow,]
test <- german[-rrow,]
```

```
# build model
set.seed(42)
logreg_model <- glm(V21~., family=binomial(link="logit"), data=train)
summary(logreg_model)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)  3.128e-01  1.287e+00   0.243 0.807990    
## V1A12       -1.707e-01  2.695e-01  -0.633 0.526520    
## V1A13       -5.591e-01  4.212e-01  -1.328 0.184316    
## V1A14       -1.490e+00  2.777e-01  -5.365 8.07e-08 ***
## V2           2.641e-02  1.135e-02   2.327 0.019971 *  
## V3A31       -1.233e-01  6.469e-01  -0.191 0.848898    
## V3A32       -6.221e-01  5.150e-01  -1.208 0.227040    
## V3A33       -7.872e-01  5.508e-01  -1.429 0.152951    
## V3A34       -1.355e+00  4.992e-01  -2.713 0.006664 ** 
## V4A41       -1.617e+00  4.654e-01  -3.475 0.000511 ***
## V4A410      -1.164e+00  9.721e-01  -1.198 0.231062    
## V4A42       -6.748e-01  3.106e-01  -2.172 0.029829 *  
## V4A43       -7.879e-01  2.993e-01  -2.633 0.008473 ** 
## V4A44        6.798e-01  9.094e-01   0.748 0.454705    
## V4A45       -2.548e-01  7.063e-01  -0.361 0.718267    
## V4A46       -2.582e-01  4.585e-01  -0.563 0.573323    
## V4A48       -1.034e+00  1.304e+00  -0.793 0.427664    
## V4A49       -6.211e-01  4.162e-01  -1.492 0.135645    
## V5           1.191e-04  5.415e-05   2.199 0.027854 *  
## V6A62       -2.606e-01  3.408e-01  -0.765 0.444501    
## V6A63       -3.082e-01  4.629e-01  -0.666 0.505527    
## V6A64       -8.784e-01  5.389e-01  -1.630 0.103083    
## V6A65       -9.555e-01  3.234e-01  -2.954 0.003132 ** 
## V7A72        2.534e-01  5.230e-01   0.485 0.627970    
## V7A73        2.249e-01  5.075e-01   0.443 0.657598    
## V7A74       -4.920e-01  5.535e-01  -0.889 0.374041    
## V7A75       -1.826e-01  5.189e-01  -0.352 0.724957    
## V8           3.207e-01  1.068e-01   3.001 0.002689 ** 
## V9A92       -1.348e-01  4.622e-01  -0.292 0.770590    
## V9A93       -6.266e-01  4.534e-01  -1.382 0.166993    
## V9A94       -2.652e-01  5.378e-01  -0.493 0.621835    
## V10A102      5.183e-01  4.807e-01   1.078 0.280982    
## V10A103     -8.914e-01  5.129e-01  -1.738 0.082228 .  
## V11         -8.127e-03  1.036e-01  -0.078 0.937500    
## V12A122      2.802e-01  2.990e-01   0.937 0.348577    
## V12A123     -2.107e-02  2.852e-01  -0.074 0.941117    
## V12A124      8.212e-01  5.172e-01   1.588 0.112349    
## V13         -1.364e-02  1.104e-02  -1.236 0.216447    
## V14A142     -5.050e-02  4.802e-01  -0.105 0.916236    
## V14A143     -8.149e-01  2.929e-01  -2.782 0.005395 ** 
## V15A152     -3.243e-01  2.847e-01  -1.139 0.254610    
## V15A153     -5.763e-01  5.804e-01  -0.993 0.320708    
## V16          2.530e-01  2.414e-01   1.048 0.294596    
## V17A172     -1.610e-01  7.531e-01  -0.214 0.830722    
## V17A173     -8.695e-03  7.233e-01  -0.012 0.990409    
## V17A174      1.018e-01  7.451e-01   0.137 0.891351    
## V18          3.050e-01  2.988e-01   1.021 0.307417    
## V19A192     -1.701e-01  2.407e-01  -0.707 0.479861    
## V20A202     -1.172e+00  6.707e-01  -1.747 0.080564 .  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 839.40  on 699  degrees of freedom
## Residual deviance: 632.14  on 651  degrees of freedom
## AIC: 730.14
##
## Number of Fisher Scoring iterations: 5
```

Just like linear regression, remove insignificant predictors and re-train the model. Thus we have our logistic regression mode and software output:

```
# re-train model with new predictors
set.seed(42)
logreg_model2 <- glm(V21~ V1+V2+V3+V4+V5+V6+V7+V8+V9+V10+V12+V14+V15+V16+V20,
                     family=binomial(link="logit"),
                     data=train)
summary(logreg_model2)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 +
##     V10 + V12 + V14 + V15 + V16 + V20, family = binomial(link = "logit"),
##     data = train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.922e-01  1.029e+00   0.187 0.851767
## V1A12       -1.738e-01  2.678e-01  -0.649 0.516437
## V1A13       -5.570e-01  4.128e-01  -1.349 0.177272
## V1A14       -1.490e+00  2.756e-01  -5.406 6.46e-08 ***
## V2           2.843e-02  1.119e-02   2.541 0.011046 *
## V3A31       -1.511e-01  6.429e-01  -0.235 0.814233
## V3A32       -6.645e-01  5.117e-01  -1.299 0.194047
## V3A33       -8.359e-01  5.478e-01  -1.526 0.127075
## V3A34       -1.413e+00  4.961e-01  -2.848 0.004406 **
## V4A41       -1.574e+00  4.581e-01  -3.437 0.000588 ***
## V4A410      -1.125e+00  9.404e-01  -1.197 0.231469
## V4A42       -6.545e-01  3.061e-01  -2.138 0.032490 *
## V4A43       -7.980e-01  2.957e-01  -2.699 0.006954 **
## V4A44        6.377e-01  9.143e-01   0.697 0.485501
## V4A45       -3.882e-01  7.033e-01  -0.552 0.580974
## V4A46       -2.689e-01  4.542e-01  -0.592 0.553770
## V4A48       -1.066e+00  1.305e+00  -0.817 0.413873
## V4A49       -6.378e-01  4.135e-01  -1.542 0.122994
## V5           1.092e-04  5.189e-05   2.104 0.035404 *
## V6A62       -2.378e-01  3.372e-01  -0.705 0.480567
## V6A63       -3.466e-01  4.553e-01  -0.761 0.446514
## V6A64       -8.794e-01  5.335e-01  -1.648 0.099288 .
## V6A65       -9.634e-01  3.205e-01  -3.006 0.002647 **
## V7A72        3.024e-01  4.527e-01   0.668 0.504106
## V7A73        2.447e-01  4.275e-01   0.572 0.567059
## V7A74       -4.659e-01  4.784e-01  -0.974 0.330179
## V7A75       -2.418e-01  4.486e-01  -0.539 0.589911
## V8           3.031e-01  1.037e-01   2.922 0.003477 **
## V9A92       -1.314e-01  4.564e-01  -0.288 0.773426
```

```
## V9A93       -5.654e-01  4.450e-01  -1.270 0.203950
## V9A94       -2.515e-01  5.318e-01  -0.473 0.636326
## V10A102      4.850e-01  4.788e-01   1.013 0.311104
## V10A103     -9.116e-01  5.099e-01  -1.788 0.073787 .
## V12A122      2.561e-01  2.919e-01   0.878 0.380210
## V12A123     -7.584e-03  2.771e-01  -0.027 0.978167
## V12A124      8.352e-01  5.049e-01   1.654 0.098089 .
## V14A142     -9.336e-02  4.768e-01  -0.196 0.844761
## V14A143     -7.920e-01  2.900e-01  -2.731 0.006316 **
## V15A152     -3.617e-01  2.724e-01  -1.328 0.184173
## V15A153     -7.102e-01  5.659e-01  -1.255 0.209477
## V16          2.347e-01  2.351e-01   0.998 0.318170
## V20A202     -1.177e+00  6.685e-01  -1.761 0.078251 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 839.40  on 699  degrees of freedom
## Residual deviance: 635.71  on 658  degrees of freedom
## AIC: 719.71
##
## Number of Fisher Scoring iterations: 5
```

```r
# generate predictions
preds <- predict(logreg_model2, test, type="response")
preds_rounded <- round(preds)
```

And the quality of fit of the logistic regression model:

```r
# confusion matrix and evaluation metrics
confusionMatrix(as.factor(preds_rounded), as.factor(test$V21))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 180  50
##          1  21  49
##
##                Accuracy : 0.7633
##                  95% CI : (0.7111, 0.8103)
##     No Information Rate : 0.67
##     P-Value [Acc > NIR] : 0.0002646
##
##                   Kappa : 0.4218
##
##  Mcnemar's Test P-Value : 0.0008906
##
##             Sensitivity : 0.8955
##             Specificity : 0.4949
##          Pos Pred Value : 0.7826
##          Neg Pred Value : 0.7000
```

```
##              Prevalence : 0.6700
##          Detection Rate : 0.6000
##    Detection Prevalence : 0.7667
##       Balanced Accuracy : 0.6952
##
##        'Positive' Class : 0
##
```

Question 10.3.2 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.

Let's define the costs for correct and incorrect classifications. Print a simple confusion matrix to visualize this:

```
# define costs
costs = matrix(c(0, 5, 1, 0), nrow = 2)
dimnames(costs) = list(Actual=c("good","bad"),
                       Predicted=c("good","bad"))
print(costs)
```

```
##       Predicted
## Actual good bad
##   good    0   1
##   bad     5   0
```

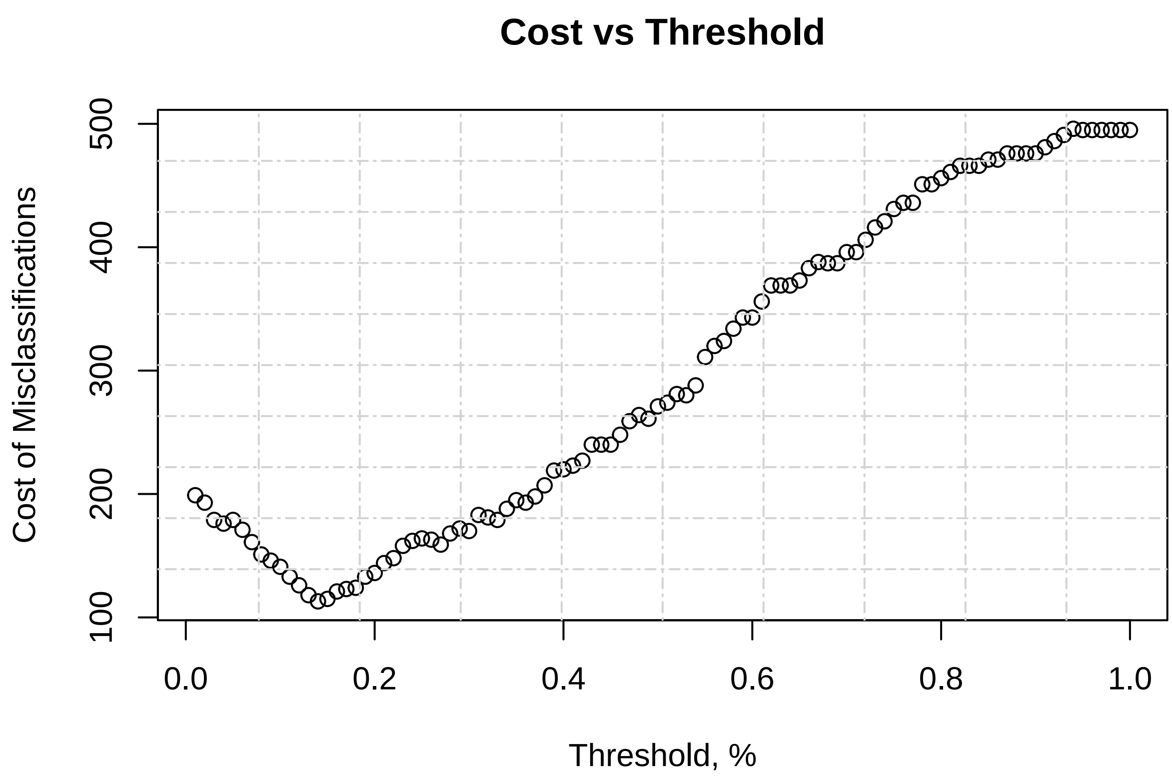
We loop through each threshold (denoted `i/100`) and calculate the cost for that threshold. Then plot the cost against the range of thresholds.

```
#initialize list
cost <- vector(mode = "list")
for (i in 1:100){
  preds_rounded <- as.integer(preds > i/100 )
  cm_matrix <- as.matrix(table(test$V21, preds_rounded))

  #Ensuring NO out of bounds issues while looping
  if(nrow(cm_matrix)==2) {fp<-cm_matrix[2,1]} else {fp=0}
  if(ncol(cm_matrix)==2){fn<-cm_matrix[1,2]} else {fn=0}

  cost<-c(cost, fn*1+fp*5)
  }

#Plots ov Total cost vs % thresholds
plot(x=seq(0.01,1,by=0.01),
     y=cost,
     xlab="Threshold, %",
     ylab="Cost of Misclassifications",
     main="Cost vs Threshold")
grid (10,10,lty=6)
```




```r
numerator <- which.min(cost)
min.threshold <-numerator/100
print(sprintf("optimal threshold: %0.3f", min.threshold))
```

```
## [1] "optimal threshold: 0.140"
```

Finally obtain the minimum cost and its corresponding optimal threshold, which is 0.140.