Introduction to Data Science

Contents

About this course

Course Content

This course will introduce participants to a fascinating field of statistics. We will see how we can rely on statistical models to gain a deep understanding from data. This often involves finding optimal predictions and classifications. Machine Learning (also known as Statistical Learning) is quickly developing and is being applied in various fields such as business analytics, political science, sociology, and elsewhere.

Machine learning can be divided into supervised learning and unsupervised learning. We cover supervised machine learning. Supervised learning involves models where we have a dependent variable - often referred to as labelled data. In unsupervised learning the outcome variable is not known - often referred to as unlabeled data.

Course Objectives

This course aims to provide an introduction to the data science approach to the quantitative analysis of data using the methods of statistical learning, an approach blending classical statistical methods with recent advances in computational and machine learning. The course will cover the main analytic methods from this field focusing on hands-on applications using example datasets. This will allow participants to gain experience with and confidence in using the methods we cover.

Course Prerequisites

Participants are expected to have a solid understanding of linear regression models and preferably know binary models. Prior exposure to the statistical software R is required. The course will not provide an introduction to R.

Agenda

1. Regression (linear models)
2. Classification
3. Cross-validation
4. Subset selection
5. Regularisation
6. Polynomials
7. Tree based models
8. Simulation and Monte Carlo simulation

Acknowledgements

The material in this course is based on the textbook: James Gareth, Daniela Witten, Trevor Hastie and Robert Tibshirani. 2013. An introduction to statistical learning. Springer. In addition, the material is based on a machine learning class at the Essex Summer School with Lucas Leemann and Philipp Broniecki. The infrastructure for this website is in large parts adopted from carried out at UCL by Altaf Ali, Jack Blumenau, Lucas Leemann, Slava Jankin Mikhaylov, and Philipp Broniecki. The ESRC Business and Local Government Data Research Centre is funded by the Economic and Social Research Council (ESRC).

---

data

slides

1 Linear Regression

1.1 Learning objectives

In this part, we cover the linear regression model. The linear model is commonly applied and versatile enough to be suitable for most tasks. We will use a dataset from the 1990 US Census which provides demographic and socio-economic data. The dataset includes observations from 1994 communities with each observation identified by a `state` and `communityname` variable. Before we start analyzing, we load the dataset and do some pre-processing.

We load a part of the census data using the `read.csv()` function and confirm that the `state` and `communityname` are present in each dataset. The dataset is named `communities.csv` and is included on your memory stick. You can copy it over to your computer and set the working directory in R to work in that folder. Alternatively, you can download the dataset here.

We assign the dataset to an object that resides in working memory. Let's call that object `communities`.

```
communities <- read.csv(file = "communities.csv", stringsAsFactors = FALSE)
```

The `stringsAsFactors` argument stops R from converting text variables into categorical variables called factors in R. The dataset is rather large and we are only interested in a few variables. In the following, we introduce a new package for data manipulation.

1.1.1 Dplyr package

The `dplyr` package is useful for data manipulation. We install it by running `install.packages("dplyr")`. We only install a package once. To update the package, run `update.packages("dplyr")`. Loading multiple packages can cause clashes if packages include functions with similar names. In order to avoid such clashes,

we will not load the package into the session with the `library()` function but instead call dplyr functions directly from the package like so: `dplyr::function_name()`. We demonstrate this as we go along.

1.1.1.1 The `dplyr::select()` function

Since our dataset has more columns (variables) than we need, let's select only a few and rename them using more meaningful names. An easy way to accomplish this is using `dplyr::select()`. The function allows us to select the columns we need and rename them at the same time.

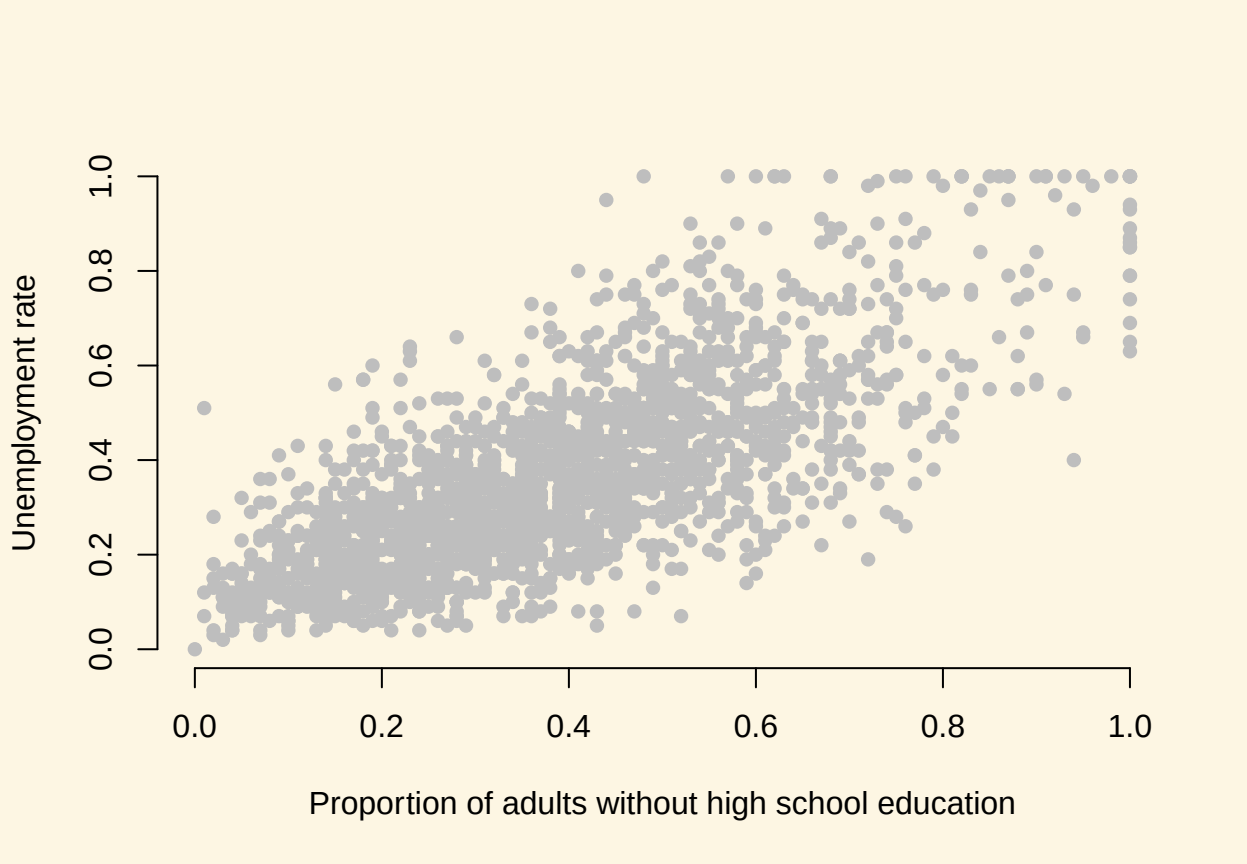
```
communities <- dplyr::select(
  communities,
  state,
  community = communityname,
  UnemploymentRate = PctUnemployed,
  NoHighSchool = PctNotHSGrad,
  white = racePctWhite)
```

Note that the first argument in `dplyr::select` is the name of the dataset (`communities` in our case). The remaining arguments are the variables that we keep. The first variable `state` has a meaningful name and does not need to be renamed. The second variable `communityname` could be shorter and we rename it to `community`. Similarly, we rename `PctUnemployed`, `PctNotHSGrad` and `racePctWhite`.

1.1.2 Visualizing a relationship b/w two continuous variables

A good way gauge whether two variables that both continuous are related is to draw a scatter plot. We do so for the unemployment rate and for the lack of high school education. Both variables are measured in percent, where `NoHighSchool` is the percentage of adults without high school education in a community.

```
plot(
  x = communities$NoHighSchool,
  y = communities$UnemploymentRate,
  xlab = "Proportion of adults without high school education",
  ylab = "Unemployment rate",
  bty = "n",
  pch = 16,
  col = "gray")
```



Use `?plot()` or google `R plot` for a description of the arguments.

It looks like communities with lower education levels suffer higher unemployment. To assess (1) whether that relationship is systematic (not a chance finding) and (2) what the magnitude of the relationship is, we estimate a linear model with the `lm()` function. The two arguments we need to provide to the function are described below.

Argument	Description
`formula`	The `formula` describes the relationship between the dependent and independent variables, for example `dependent.variable ~ independent.variable` In our case, we'd like to model the relationship using the formula: `UnemploymentRate ~ NoHighSchool`
`data`	This is simply the name of the dataset that contains the variable of interest. In our case, this is the merged dataset called `communities`.

For more information on the `lm()` function, run `?lm()`. Let's run the linear model.

```
m1 <- lm(UnemploymentRate ~ NoHighSchool, data = communities)
```

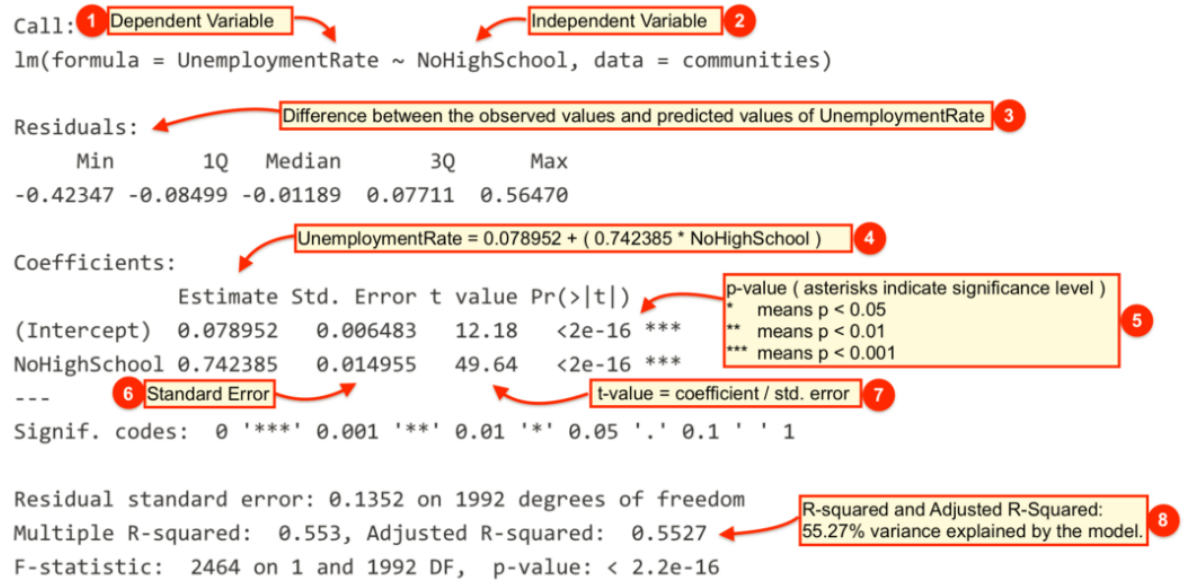
The `lm()` function models the relationship between `UnemploymentRate` and `NoHighSchool` and we've assigned the estimated model to the object `m1`. We can use the `summary()` function on `m1` for the key results.

```
summary(m1)
```

```
Call:
lm(formula = UnemploymentRate ~ NoHighSchool, data = communities)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-0.42347 -0.08499 -0.01189  0.07711  0.56470

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.078952   0.006483   12.18   <2e-16 ***
NoHighSchool 0.742385   0.014955   49.64   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1352 on 1992 degrees of freedom
Multiple R-squared:  0.553, Adjusted R-squared:  0.5527
F-statistic:  2464 on 1 and 1992 DF,  p-value: < 2.2e-16
```

The output from `lm()` might seem overwhelming at first so let's break it down one item at a time.



#	Description
1	The *dependent* variable, also sometimes called the outcome variable. We are trying to model the effects of `NoHighSchool` on `UnemploymentRate` so `UnemploymentRate` is the *dependent* variable.
2	The *independent* variable or the predictor variable. In our example, `NoHighSchool` is the *independent* variable.
3	The differences between the observed values and the predicted values are called *residuals*.

#	Description
4	The *coefficients* for the intercept and the *independent* variables. Using the *coefficients* we can write down the relationship between the *dependent* and the *independent* variables as: `UnemploymentRate` = 0.078952 + (0.7423853 * `NoHighSchool`) This tells us that for each unit increase in the variable `NoHighSchool`, the `UnemploymentRate` increases by 0.7423853.
5	The *p-value* of the model. Recall that according to the null hypotheses, the coefficient of interest is zero. The *p-value* tells us whether can can reject the null hypotheses or not.
6	The *standard error* estimates the standard deviation of the coefficients in our model. We can think of the *standard error* as the measure of precision for the estimated coefficients.
7	The *t statistic* is obtained by dividing the *coefficients* by the *standard error*.
8	The *R-squared* and *adjusted R-squared* tell us how much of the variance in our model is accounted for by the *independent* variable. The *adjusted R-squared* is always smaller than *R-squared* as it takes into account the number of *independent* variables and degrees of freedom.

1.1.2.1 Predictions

We are often interested in predicting values for the dependent variable based on a values for the independent variable. For instance, what is the predicted unemployment rate given 50 percent of the adults without high school education? We use the `predict()` function to assess this. Instead of making the forecast for the case were 50 percent do not have high school education, we make a prediction for each level of low education.

We create a sequence of values for low education using the sequence function first `seq()`. We create 100 values from 0 to 1.

```
edu <- seq(from = 0, to = 1, length.out = 100)
```

We now define a dataset where the variable names are called exactly the same as in our regression model `m1`. Let's check the name of the independent variable in `m1` by calling the object. We then copy and paste the variable name to make sure that we don't have a typo in our code.

```
m1
```

```

Call:
lm(formula = UnemploymentRate ~ NoHighSchool, data = communities)

Coefficients:
 (Intercept)  NoHighSchool
     0.07895       0.74239
```

We now use the `predict()` function to make a prediction for each of the 100 `edu` values.

```
preds <- predict(m1, newdata = data.frame(NoHighSchool = edu), se.fit = TRUE)
```

We create a new dataset including the education values from 0 to 1 and the predictions. In the `predict()` function, we set the argument `se.fit` to TRUE. This returns a standard error for our prediction and lets us

quantify our uncertainty. IN the dataset, we will save the point estimates (the best guesses) as well as values for the upper and lower bound of our confidence intervals

```
out <- data.frame( NoHighSchool = edu,
                   predicted_unemployment_rate = preds$fit,
                   lb = preds$fit - 1.96 * preds$se.fit,
                   ub = preds$fit + 1.96 * preds$se.fit)
```

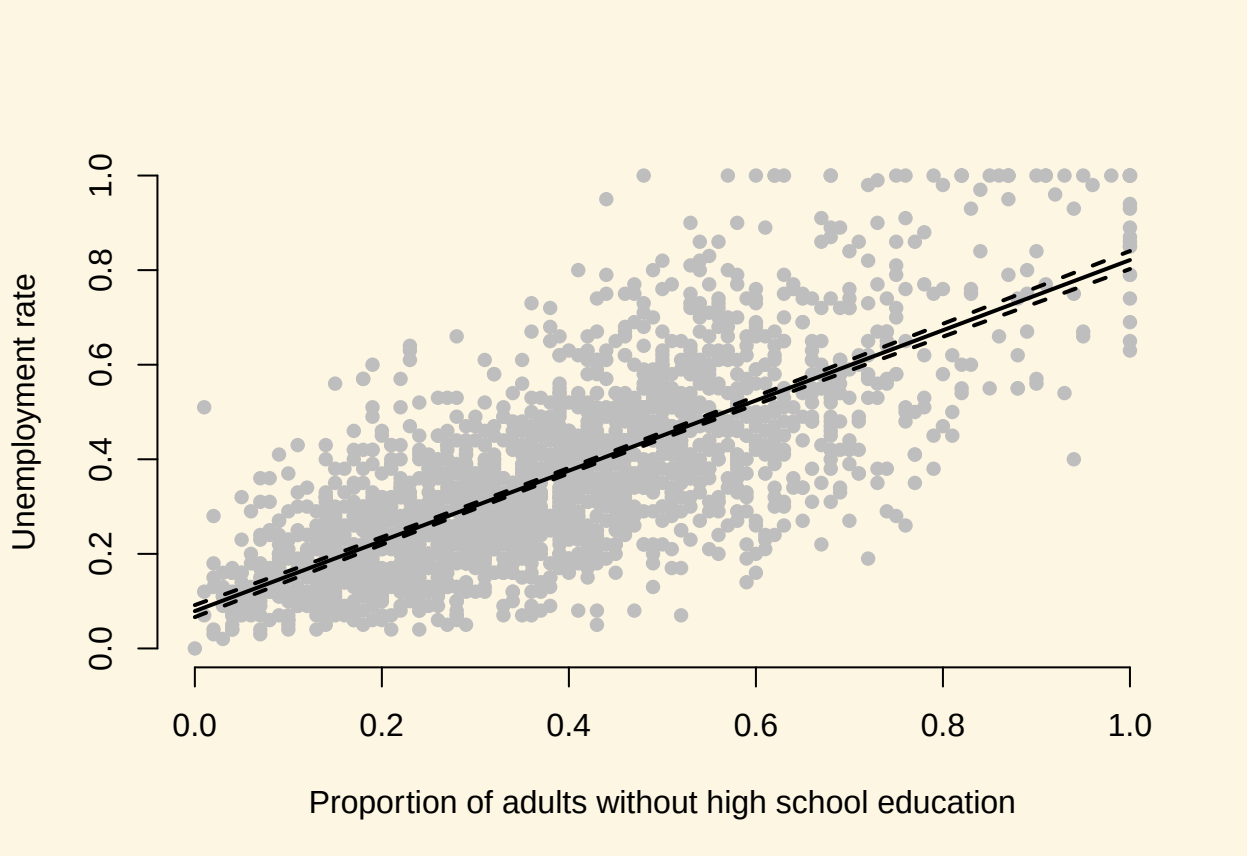
Let's inspect the first ten values of our data.

```
head(out)
```

```
  NoHighSchool predicted_unemployment_rate         lb         ub
1   0.00000000                  0.07895202 0.06624469 0.09165936
2   0.01010101                  0.08645087 0.07400458 0.09889715
3   0.02020202                  0.09394971 0.08176286 0.10613655
4   0.03030303                  0.10144855 0.08951944 0.11337766
5   0.04040404                  0.10894739 0.09727420 0.12062058
6   0.05050505                  0.11644623 0.10502702 0.12786544
```

We now add our prediction to the scatter plot.

```
lines( x = edu, y = out$predicted_unemployment_rate, lwd = 2)
lines( x = edu, y = out$lb, lwd = 2, lty = "dashed")
lines( x = edu, y = out$ub, lwd = 2, lty = "dashed")
```



As the plot shows, the precision of our estimates is quite good (the 95 percent confidence interval is narrow).

Returning to our example, are there other variables that might explain unemployment rates in our communities dataset? For example, is unemployment rate higher or lower in communities with different levels of minority

population?

We first create a new variable called `Minority` by subtracting the percent of `White` population from 1. Alternatively, we could have added up the percent of Black, Hispanic and Asians to get the percentage of minority population since our census data also has those variables.

```
communities$Minority <- 1 - communities$white
```

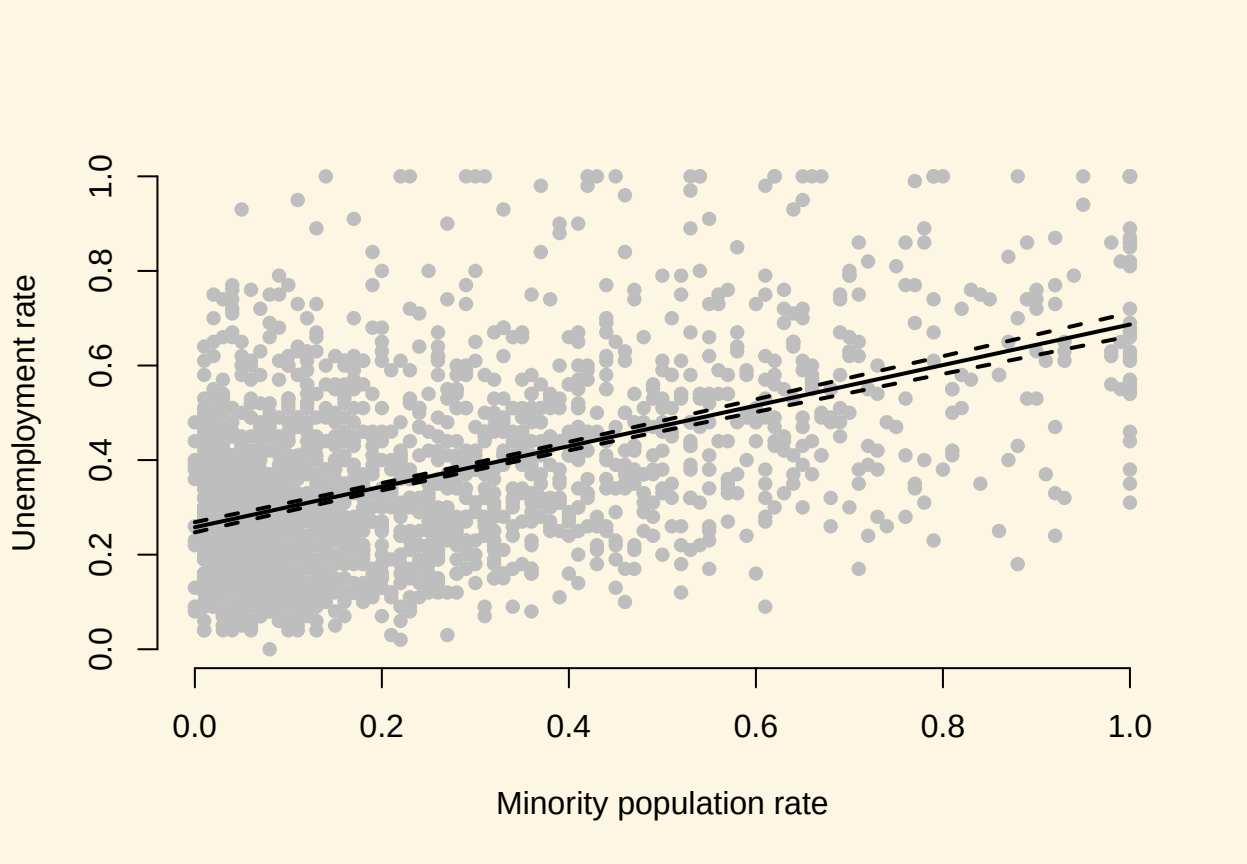
Next we fit a linear model using `Minority` as the independent variable.

```
m2 <- lm(UnemploymentRate ~ Minority, data = communities)
summary(m2)
```

```

Call:
lm(formula = UnemploymentRate ~ Minority, data = communities)

Residuals:
     Min       1Q   Median       3Q      Max
-0.45521 -0.12189 -0.02369  0.10162  0.68203

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.257948   0.005506   46.85   <2e-16 ***
Minority    0.428702   0.015883   26.99   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.173 on 1992 degrees of freedom
Multiple R-squared:  0.2678,    Adjusted R-squared:  0.2674
F-statistic: 728.5 on 1 and 1992 DF,  p-value: < 2.2e-16
```

Now let's see how this model compares to our first model. We can show regression line from `model2` just like we did with our first model.

```
# plot
plot(communities$Minority, communities$UnemploymentRate,
     xlab = "Minority population rate",
     ylab = "Unemployment rate",
     bty = "n",
     pch = 16,
     col = "gray")

# predict outcomes
minority.seq <- seq(from = 0, to = 1, length.out = 100)
preds2 <- predict(m2, newdata = data.frame(Minority = minority.seq), se.fit = TRUE)
out2 <- data.frame(Minority = minority.seq,
                   predicted_unemployment_rate = preds2$fit,
                   lb = preds2$fit - 1.96 * preds2$se.fit,
                   ub = preds2$fit + 1.96 * preds2$se.fit)
lines( x = minority.seq, y = out2$predicted_unemployment_rate, lwd = 2)
lines( x = minority.seq, y = out2$lb, lwd = 2, lty = "dashed")
lines( x = minority.seq, y = out2$ub, lwd = 2, lty = "dashed")
```



Does `m2` offer a better fit than `m1`? Maybe we can answer that question by looking at the regression tables instead. Let's print the two models side-by-side in a single table with the `screenreg()` function contained in the `texreg` package.

Let's install `texreg` first like so:

```
install.packages("texreg")
```

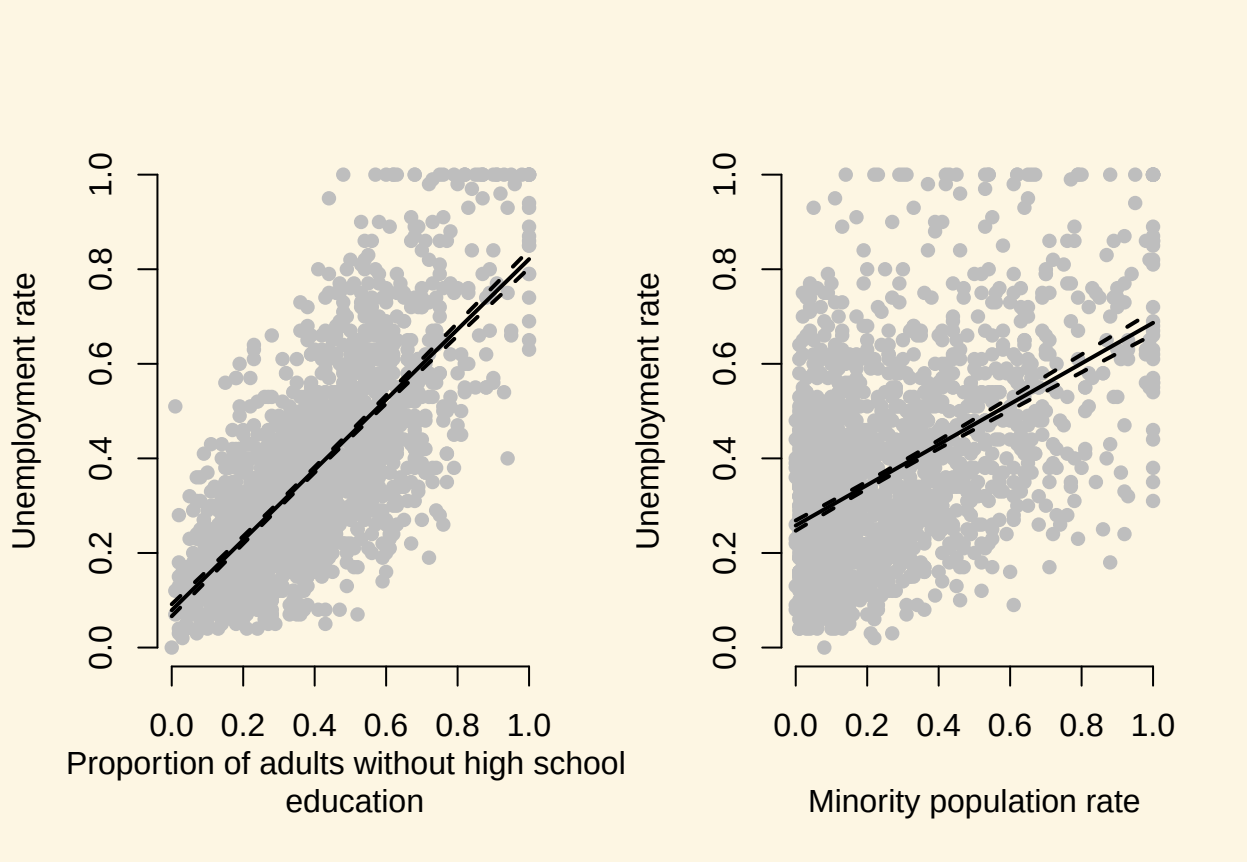
We now compare the models using the `texreg()` function like so:

```
texreg::screenreg(list( m1, m2 ))
```

```
=====================================
              Model 1      Model 2
-------------------------------------
(Intercept)      0.08 ***     0.26 ***
                (0.01)       (0.01)
NoHighSchool     0.74 ***
                (0.01)
Minority                      0.43 ***
                             (0.02)
-------------------------------------
R^2              0.55         0.27
Adj. R^2         0.55         0.27
Num. obs.     1994         1994
RMSE             0.14         0.17
=====================================
```

```
*** p < 0.001, ** p < 0.01, * p < 0.05
```

Contemplate the output from the table for a moment. Slope coefficients (everything except the intercept) are always the effect of a 1-unit change of the independent variable on the dependent variable in the units of the dependent variable. Both our independent variables are proportions. Hence a 1-unit change covers the entire ranges of our independent variables (0 to 1). Model 1 suggests that the unemployment rate is 74 percent larger in a district where no one has a high school degree than in a district where everyone has a high school degree. Similarly, model 2 suggests that in a district where everyone has a minority background (making everyone is a minority an oxymoron), the unemployment rate 43 percent higher than in a community where no one is. Please note that these predictive models should not be mistaken to capture causal relationships.



These are the two plots that we created earlier. In the model using `NoHighSchool` the points which are the actual unemployment rates are much closer to our prediction (the regression line) than in the model using `Minority`. This means that variation in `NoHighSchool` better explains variation in `UnemploymentRate` than variation in `Minority`. This is captured in the `R^2` and `Adj. R^2`. Both `R^2` and `Adj. R^2` are measures of model fit. The difference between them is that `Adj. R^2` is a measure that penalizes model complexity (more variables). In models with more than one independent variable, we rely on `Adj. R^2` and in models with one independent variable, we use `R^2`, i.e. here we would use `R^2`.

2 Classification

2.1 Seminar

2.1.1 The Non-Western Foreigners Data Set

We start by clearing our workspace.

```
# clear workspace
rm(list = ls())
```

Let's check the codebook of our data.

Variable Name	Description
IMMBRIT	Out of every 100 people in Britain, how many do you think are immigrants from Non-western countries?
over.estimate	1 if estimate is higher than 10.7%.
RSex	1 = male, 2 = female
RAge	Age of respondent
Househld	Number of people living in respondent's household
party_self	1 = Conservatives; 2 = Labour; 3 = SNP; 4 = Ukip; 5 = BNP; 6 = GP; 7 = party.other
paper	Do you normally read any daily morning newspaper 3+ times/week?
WWWhourspW	How many hours WWW per week?
religious	Do you regard yourself as belonging to any particular religion?
employMonths	How many mnths w. present employer?
urban	Population density, 4 categories (highest density is 4, lowest is 1)
health.good	How is your health in general for someone of your age? (0: bad, 1: fair, 2: fairly good, 3: good)
HHInc	Income bands for household, high number = high HH income

The dataset is on your memory sticks and also available for download here.

```
# load non-western foreigners data set
load("non_western_immigrants.RData")

# data manipulation
fdata$RSex <- factor(fdata$RSex, labels = c("Male", "Female"))
fdata$health.good <- factor(fdata$health.good, labels = c("bad", "fair", "fairly good", "good") )
fdata$party_self <- factor(fdata$party_self, labels = c("Conservatives", "Labour", "SNP",
                                                        "Ukip", "BNP", "Greens", "Other"))

# urban to dummies (for knn later)
table(fdata$urban) # 3 is the modal category (keep as baseline) but we create all categories
```

```
  1   2   3   4
214 281 298 256
```

```
fdata$rural <- ifelse( fdata$urban == 1, yes = 1, no = 0)
fdata$partly.rural <- ifelse( fdata$urban == 2, yes = 1, no = 0)
fdata$partly.urban <- ifelse( fdata$urban == 3, yes = 1, no = 0)
fdata$urban <- ifelse( fdata$urban == 4, yes = 1, no = 0)
```

In our data manipulation, we first turned `RSex` into a factor variable. Factor is a variable type in R, that is handy because we declare that a variable is categorical. When we run models with a factor variable, R will handle them correctly, i.e. break them up into binary variables internally.

Alternatively, with `urban`, we show how to break up such a variable into binary variables manually. We use the `ifelse()` function were the first argument is a logical condition such as `fdata$urban == 1` meaning "if the variable `urban` in `fdata` takes on the value 1. This condition is evaluated for every observation in the dataset and if it is met we assign a 1 (`yes = 1`) and if not we assign a 0 (`no = 0`).

2.1.2 Logistic Regression

We want to predict whether respondents over-estimate immigration from non-western contexts. We begin by normalizing our variables (we make them comparable). Then we look at the distribution of the dependent variable. We check how well we could predict misperception of immigration in our sample without a statistical model.

```
# create a copy of the original IMMBRIT variable (needed for classification with lm)
fdata$IMMBRIT_original_scale <- fdata$IMMBRIT

# our function for normalization
our.norm <- function(x){
  return((x - mean(x)) / sd(x))
}

# continuous variables
c.vars <- c("IMMBRIT", "RAge", "Househld", "HHInc", "employMonths", "WWWhourspW")

# normalize
fdata[, c.vars] <- apply( X = fdata[, c.vars], MARGIN = 2, FUN = our.norm )
```

First, we copied the variable IMMBRIT before normalizing it. Don't worry about this now, it will become clear why we did this further down in the code.

We then define our own function. A function takes some input which we called `x` and does something with that input. In case, x is a numeric variable. For every value of x, we subtract the mean of x. Therefore, we center the variable on 0, i.e. the new mean will be 0. We then divide by the standard deviation of the variable. This is necessary to make the variables comparable. The units of all variables are then represented in average deviations from their means.

In the next step, we create a character vector with the variable names of all variables that are continuous and lastly we normalize. We do this by sub-setting our data with square brackets. So `fdata[ , c.vars]` is the part of our dataset that includes the continuous variables. The function `apply()` lets us carry out the same operation repeatedly for all the variables. The argument `X` is the data. The argument `MARGIN` says we want to apply our normalization column-wise. The argument `FUN` means function. Here, we input our normalization function.

We now have a look at our dependent variable of interest. The variable `over.estimate` measures whether a respondent over estimates the number of non-western immigrants or not (yes = 1; no = 0). The actual percentage of non-western immigrants was 10.7 percent at the time of the survey.

2.1.2.1 The naive guess

The naive guess is the best prediction without a model. Or put differently, the best prediction we could make without having any context information. Have a look at the variable `over.estimate` and decide on your own what you would do to maximize your predictive accuracy...

```
# proportion of people who over-estimate
mean(fdata$over.estimate)
```

```
[1] 0.7235462
```

```
# naive guess
ifelse( mean(fdata$over.estimate) >= 0.5, yes = 1, no = 0 )
```

```
[1] 1
```

```
# So, to maximise prediction accuracy without a model, we must simply always predict the more common ca
```

Alright, now that we have figured out what to predict, what would be our predictive power based on that prediction? Try to figure this out on your own...

```
# predicitive power based on the naive guess

ifelse( mean(fdata$over.estimate) >= 0.5,
        yes = mean(fdata$over.estimate),
        no = 1 - mean(fdata$over.estimate))
```

```
[1] 0.7235462
```

```
# So our predicitive accuracy depends on the proportion of people who over estimate. If the proportion
```

A predictive model must always beat the predictive power of the naive guess.

2.1.3 The logit model

We use the generalized linear model function `glm()` to estimate a logistic regression. The syntax is very similar to the `lm` regression function that we are already familiar with, but there is an additional argument that we need to specify (the `family` argument) in order to tell R that we would like to estimate a logistic regression model.

Argument	Description
`formula`	As before, the `formula` describes the relationship between the dependent and independent variables, for example `dependent.variable ~ independent.variable` In our case, we will use the formula: `vote ~ wifecoethnic + distance`
`data`	Again as before, this is simply the name of the dataset that contains the variable of interest. In our case, this is the dataset called `afb`.
`family`	The `family` argument provides a description of the error distribution and link function to be used in the model. For our purposes, we would like to estimate a binary logistic regression model and so we set `family = binomial(link = "logit")`

We tell `glm()` that we have a binary dependent variable and we want to use the logistic link function using the `family = binomial(link = "logit")` argument:

```
m.logit <- glm( over.estimate ~ RSex + RAge + Househld + party_self + paper + WWWhourspW +
                  religious +  employMonths + rural + partly.rural + urban + health.good +
                  HHInc, data = fdata, family = binomial(link = "logit"))
summary(m.logit)
```

```
Call:
glm(formula = over.estimate ~ RSex + RAge + Househld + party_self +
    paper + WWWhourspW + religious + employMonths + rural + partly.rural +
    urban + health.good + HHInc, family = binomial(link = "logit"),
    data = fdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2342  -1.1328   0.6142   0.8262   1.3815

Coefficients:
                     Estimate Std. Error z value Pr(>|z|)
(Intercept)           0.72437    0.36094   2.007   0.0448 *
RSexFemale            0.64030    0.15057   4.253 2.11e-05 ***
```

```
RAge                    0.01031    0.09073   0.114   0.9095
Househld                0.02794    0.08121   0.344   0.7308
party_selfLabour       -0.31577    0.19964  -1.582   0.1137
party_selfSNP           1.85513    1.05603   1.757   0.0790 .
party_selfUkip         -0.51315    0.46574  -1.102   0.2706
party_selfBNP           0.05604    0.44846   0.125   0.9005
party_selfGreens        0.92131    0.57305   1.608   0.1079
party_selfOther         0.12542    0.18760   0.669   0.5038
paper                   0.14855    0.15210   0.977   0.3287
WWWhourspW             -0.02598    0.08008  -0.324   0.7457
religious               0.05139    0.15274   0.336   0.7365
employMonths            0.01899    0.07122   0.267   0.7897
rural                  -0.35097    0.21007  -1.671   0.0948 .
partly.rural           -0.37978    0.19413  -1.956   0.0504 .
urban                   0.12732    0.21202   0.601   0.5482
health.goodfair        -0.09534    0.33856  -0.282   0.7782
health.goodfairly good  0.11669    0.31240   0.374   0.7087
health.goodgood         0.02744    0.31895   0.086   0.9314
HHInc                  -0.48513    0.08447  -5.743 9.30e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1236.9  on 1048  degrees of freedom
Residual deviance: 1143.3  on 1028  degrees of freedom
AIC: 1185.3

Number of Fisher Scoring iterations: 5
```

2.1.4 Predict Outcomes from logit

We can use the `predict()` function to calculate fitted values for the logistic regression model, just as we did for the linear model. Here, however, we need to take into account the fact that we model the *log-odds* that $Y = 1$, rather than the *probability* that $Y = 1$. The `predict()` function will therefore, by default, give us predictions for Y on the log-odds scale. To get predictions on the probability scale, we need to add an additional argument to `predict()`: we set the `type` argument to `type = "response"`.

```
# predict probabilities
preds.logit <- predict( m.logit, type = "response")
```

To see how good our classification model is we need to compare the classification with the actual outcomes. We first create an object `exp.logit` which will be either `0` or `1`. In a second step, we cross-tab it with the true outcomes and this allows us to see how well the classification model is doing.

```
# predict whether respondent over-estimates or not
exp.logit <- ifelse( preds.logit > 0.5, yes = 1, no = 0)

# confusion matrix (table of predictions and true outcomes)
table(prediction = exp.logit, truth = fdata$over.estimate)
```

```
          truth
prediction   0   1
         0  41  40
         1 249 719
```

The diagonal elements are the correct classifications and the off-diagonal ones are wrong. We can compute the share of correct classified observations as a ratio.

```
# percent correctly classified
(35 + 728) / 1049
```

```
[1] 0.7273594
```

We can also write code that will estimate the percentage correctly classified for different values.

```
# more generally
mean( exp.logit == fdata$over.estimate)
```

```
[1] 0.7244995
```

This is the performance on the training data and we expect the test error to be higher than this. To get at a better indication of the model's classification error we can split the dataset into a training set and a test set.

This is the performance on the training data and we expect the test error to be higher than this. To get at a better indication of the model's classification error we can split the dataset into a training set and a test set.

```
# set the random number generator
set.seed(123)

# random draw of 80% of the observations (row numbers) to train the model
train.ids <- sample(nrow(fdata), size = as.integer( (nrow(fdata)*.80) ), replace = FALSE)

# the validation data
fdata.test <- fdata[ -train.ids, ]
dim(fdata.test)
```

```
[1] 210  17
```

So, we first set the random number generator with `set.seed()`. It does not matter which number we use to set the RNG but the point is that re-running our script will always lead to the same result (Disclaimer: In April 2019, it was changed how the RNG works. To replicate anything that was created prior to that data or anything that was created on an old R version, the options have to be adjusted like so: `RNGkind(sample.kind = "Rounding")`)

We then take a random sample with `sample()` function. The first argument is what we draw from. Here, we use `nrow()` which returns the number of rows in the data set. We therefore, draw numbers between 1 and the number of observations in our dataset. We draw 80 percent of the observations, so we multiply the number of observations with 0.8. Since that number might not be whole, we cut off decimal places with the `as.integer()` function. Finally, the argument `replace = FALSE` ensures that we can draw an observation only once.

Now we fit the model using the training data only and then test its performance on the test data.

```
# re-fit the model on the raining data
m.logit <- glm(over.estimate ~ RSex + RAge + Househld + party_self + paper + WWWhourspW +  religious +
               employMonths + rural + partly.rural + urban + health.good + HHInc, data = fdata,
               subset = train.ids,
               family = binomial(link = "logit"))

# predict probabilities of over-estimating but for the unseen data
preds.logit <- predict(m.logit, newdata = fdata.test, type = "response")

# classify predictions as over-estimating or not
exp.logit <- ifelse( preds.logit > 0.5, yes = 1, no = 0)
```

```
# confusion matrix of predictions against truth
table( prediction = exp.logit, truth = fdata.test$over.estimate)
```

```
          truth
prediction   0   1
         0   6   9
         1  49 146
```

```
# percent correctly classified
mean( exp.logit == fdata.test$over.estimate )
```

```
[1] 0.7238095
```

The accuracy of the model is slightly lower in the test dataset than in the training data. The difference is not big here but in practice it can be quite large.

Let's try to improve the classification model by relying on the best predictors.

```
# try to improve the prediction model by relying on "good" predictors
m.logit <- glm(over.estimate ~ RSex + rural + partly.rural + urban + HHInc,
             data = fdata, subset = train.ids, family = binomial(link = "logit"))
preds.logit <- predict(m.logit, newdata = fdata.test, type = "response")
exp.logit <- ifelse( preds.logit > 0.5, yes = 1, no = 0)
table( prediction = exp.logit, truth = fdata.test$over.estimate )
```

```
          truth
prediction   0   1
         0   7   2
         1  48 153
```

```
mean( exp.logit == fdata.test$over.estimate )
```

```
[1] 0.7619048
```

We improved our model by removing variables. This will never be the case if we apply the same data for training a model and testing it. But this illustrates that a model that is not parsimonious starts fitting noise and will do poorly with new data.

2.1.5 K-Nearest Neighbors

There are many models for classification. One of the more simple ones is KNN. For it, we need to provide the data in a slightly different format and we need to install the `class` package.

```
# training & test data set of predictor variables only
train.X <- cbind( fdata$RSex, fdata$rural, fdata$partly.rural, fdata$urban, fdata$HHInc )[train.ids, ]
test.X <- cbind( fdata$RSex, fdata$rural, fdata$partly.rural, fdata$urban, fdata$HHInc )[-train.ids, ]

# response variable for training observations
train.Y <- fdata$over.estimate[ train.ids ]

# re-setting the random number generator
set.seed(123)

# run knn
knn.out <- class::knn(train.X, test.X, train.Y, k = 1)

# confusion matrix
table( prediction = knn.out, truth = fdata.test$over.estimate )
```

```
          truth
prediction   0   1
         0  11  15
         1  44 140
```

```
# percent correctly classified
mean( knn.out == fdata.test$over.estimate )
```

```
[1] 0.7190476
```

We can try and increase the accuracy by changing the number of nearest neighbors we are using:

```
# try to increae accuracy by varying k
knn.out <- class::knn(train.X, test.X, train.Y, k = 7)
mean( knn.out == fdata.test$over.estimate )
```

```
[1] 0.752381
```

2.1.6 Model the Underlying Continuous Process

We can try to model the underlying process and classify afterwards. By doing that, the dependent variable provides more information. In effect we turn our classification problem into a regression problem.

```
# fit the linear model on the numer of immigrants per 100 Brits
m.lm <- lm(IMMBRIT ~ RSex + rural + partly.rural + urban + HHInc,
           data = fdata, subset = train.ids)

# preditions
preds.lm <- predict(m.lm, newdata = fdata.test)

# threshold for classfication
threshold <- (10.7 - mean(fdata$IMMBRIT_original_scale)) / sd(fdata$IMMBRIT_original_scale)

# now we do the classfication
exp.lm <- ifelse( preds.lm > threshold, yes = 1, no = 0)

# confusion matrix
table( prediction = exp.lm, truth = fdata.test$over.estimate)
```

```
          truth
prediction   0   1
         1  55 155
```

```
# percent correctly classified
mean( exp.lm == fdata.test$over.estimate)
```

```
[1] 0.7380952
```

We do worse by treating this as a regression problem rather than a classification problem - often, however, this would be the other way around.

3 Cross-Validation

3.1 Seminar

We start by clearing our workspace.

```
# clear workspace
rm( list = ls() )
```

3.1.1 The Validation Set Approach

We use a subset of last weeks non-western immigrants data set (the version for this week includes men only). We can use the `head()` function to have a quick glance at the data. Download the data here

The codebook is:

Variable Name	Description
IMMBRIT	Out of every 100 people in Britain, how many do you think are immigrants from Non-western countries?
over.estimate	1 if estimate is higher than 10.7%.
RAge	Age of respondent
Househld	Number of people living in respondent's household
Cons, Lab, SNP, Ukip, BNP, GP, party.other	Party self-identification
paper	Do you normally read any daily morning newspaper 3+ times/week?
WWWhourspW	How many hours WWW per week?
religious	Do you regard yourself as belonging to any particular religion?
employMonths	How many mnths w. present employer?
urban	Population density, 4 categories (highest density is 4, lowest is 1)
health.good	How is your health in general for someone of your age? (0: bad, 1: fair, 2: fairly good, 3: good)
HHInc	Income bands for household, high number = high HH income

```
# load non-western foreigners data
load("BSAS_manip_men.RData")
```

We first select a random sample of 239 out of 478 observations (check that that's half the observations in our dataset using `nrow(data2)`). We initialize the random number generator with a seed using `set.seed()` to ensure that repeated runs produce consistent results.

```
# initialize random number generator
set.seed(1)

# pick 239 numbers out of 1 to 478
train <- sample(478, 239)
```

We then estimate the effects of age on the perceived number of immigrants per 100 Brits with `lm()` on the selected subset.

```
# linear regression
m.lm <- lm( IMMBRIT ~ RAge, data = data2, subset = train)
```

Next, we use our model that we trained on the training set to predict outcomes in the test set - the test set contains unseen data. We subset the dataset using square brackets such that it excludes the training observations. The `-` operator means except in this case. So `data2[-tain, ]` is the dataset excluding training observations.

```
# predict on test set
preds.lm <- predict( m.lm, data2[-train,] )
```

Next, we compare our predictions on the test set to the real outcomes. Our loss function (evaluation metric) is the mean squared error (MSE):

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
# mse in the validation (test) set
mse <- mean((data2$IMMBRIT[-train] - preds.lm)^2)
 # error rate
mse
```

```
[1] 435.4077
```

The error rate for a linear model is 435.41. We can also fit higher degree polynomials with the `poly()` function. First, let's try a quadratic model.

So far, we have modeled the relationship between `RAge` and `IMMBRIT` as linear. It is possible that the relationship is non-linear. We can model this using polynomials, i.e. raising `RAge` to some power. We start with the square. We could use the `^2` operator to raise `RAge` to the second power like so: `data2$RAge^2`. However, it's generally not a good idea to do this because polynomials are correlated introducing colinearity into the model. We can avoid this using the `poly()` function which de-correlates the variable and its powers.

```
# polynomials (quadratic)
m.lm2 <- lm( IMMBRIT ~ poly(RAge, 2), data = data2, subset = train)
```

Let's have a quick look at the regression table using the `texreg` package.

```
library(texreg)
texreg::screenreg(m.lm2)
```

```
==========================
                Model 1
--------------------------
(Intercept)      24.69 ***
                 (1.17)
poly(RAge, 2)1   25.21
                (26.59)
poly(RAge, 2)2   59.89 *
                (25.07)
--------------------------
R^2               0.03
Adj. R^2          0.02
Num. obs.       239
RMSE             17.92
==========================
*** p < 0.001, ** p < 0.01, * p < 0.05
```

Interpreting polynomials is not straightforward because the effect is not linear, i.e. it is not constant. Here, `poly(RAge, 2)1` is `RAge` and `poly(RAge, 2)1` is the square of `RAge`. The effect is significant. However, to interpret the effect we would need to plot it. Instead, we will proceed by making predictions on the validation set (test set) again and calculate the MSE.

```
preds.lm2 <- predict(m.lm2, data2[-train,])
mse2 <- mean((data2$IMMBRIT - preds.lm2)^2)
mse2
```

```
[1] 391.8855
```

Quadratic regression performs better than a linear model because it reduces the error (MSE) from 435.41 to 391.89 (10%). We move on to a cubic model.

```
# cubic model
m.lm3 <- lm( IMMBRIT ~ poly(RAge, 3), data = data2, subset = train)
mse3  <- mean( (data2$IMMBRIT[-train] -  predict(m.lm3, data2[-train,]))^2 )
mse3
```

```
[1] 412.7887
```

According to our approach, the quadratic model is the best out of the three we tested. However, this might be due to the training/test split that we made. We will try again using a different split of the data.

```
# fit the models on a different training/test split
set.seed(2)
train <- sample(478, 239)
m.lm <- lm( IMMBRIT ~ RAge, data = data2, subset = train)
mse <- mean( (data2$IMMBRIT[-train] -  predict(m.lm, data2[-train,]) )^2 )

# quadratic
m.lm2 <- lm( IMMBRIT ~ poly(RAge, 2), data = data2, subset = train)
mse2 <- mean( (data2$IMMBRIT[-train] -  predict(m.lm2, data2[-train,]))^2 )

# cubic
m.lm3 <- lm( IMMBRIT ~ poly(RAge, 3), data = data2, subset = train)
mse3 <- mean( (data2$IMMBRIT[-train] -  predict(m.lm3, data2[-train,]))^2 )

# outut
output <- cbind( mse, mse2, mse3 )
colnames(output) <- c("linear", "quadratic", "cubic")
output
```

```
       linear quadratic    cubic
[1,] 413.6691  399.0577 394.3029
```

Clearly, the results are different from our initial run. Not only, are the error rates different but in addition, the order of the models changes. In this trial, the cubic model performs best. It appears that we need to split data more often to determine which is the best model overall. We will move on to leave-one-out cross-validation which does exactly that.

3.1.2 Leave-One-Out Cross-Validation (LOOCV)

In LOOCV, we train our model on all but the first observation and subsequently predict the first observation using our model. Next, we train our model on all but the second observation and predict the second observation with that model and so forth for every observation in the dataset. That means, we must estimate as many models as we have observations in the dataset. While there are some tricks to make the computation faster for linear models, LOOCV can take a long time to run.

Before we get into it, we quickly introduce a new function. The `glm()` function offers a generalization of the linear model while allowing for different link functions and error distributions other than gaussian. By default, `glm()` simply fits a linear model identical to the one estimated with `lm()`. Let's confirm this quickly.

```
glm.fit <- glm( IMMBRIT ~ RAge, data = data2)
lm.fit <- lm( IMMBRIT ~ RAge, data = data2)
texreg::screenreg( list(glm.fit, lm.fit), custom.model.names = c("GLM", "LM") )
```

```
==========================================
                GLM            LM
------------------------------------------
```

```
(Intercept)         25.83 ***   25.83 ***
                    (2.88)      (2.88)
RAge                -0.03       -0.03
                    (0.05)      (0.05)
----------------------------------------
AIC               4197.88
BIC               4210.39
Log Likelihood   -2095.94
Deviance        180117.97
Num. obs.          478         478
R^2                              0.00
Adj. R^2                        -0.00
RMSE                            19.45
========================================
*** p < 0.001, ** p < 0.01, * p < 0.05
```

The coefficient estimates are similar but the fit statistics that are reported differ. Generally a GLM maximizes the likelihood whereas LM minimizes the sum of squared deviations from the regression line. Maximum likelihood estimation is more general and used in most statistical models.

We will use the `glm()` function from here on because it can be used with `cv.glm()` which allows us to estimate the k-fold cross-validation prediction error. We also need to install a new package called `boot` using `install.packages("boot")`.

```
library(boot)

# use cv.glm() for k-fold corss-validation on glm
cv.err <- cv.glm(data2, glm.fit)

# cross-validation error
cv.err$delta
```

```
[1] 380.2451 380.2415
```

```
# the number of folds
cv.err$K
```

```
[1] 478
```

The returned value from `cv.glm()` contains a delta vector of components - the raw cross-validation estimate and the adjusted cross-validation estimate respectively. We are interested in the raw cross-validation error.

NOTE: if we do not provide the option **K** in `cv.glm()` we automatically perform leave-one-out cross-validation (LOOCV).

We repeat this process in a `for()` loop to compare the cross-validation error of higher-order polynomials. The following example estimates the polynomial fit of the order 1 through 7 and stores the result in a cv.error vector.

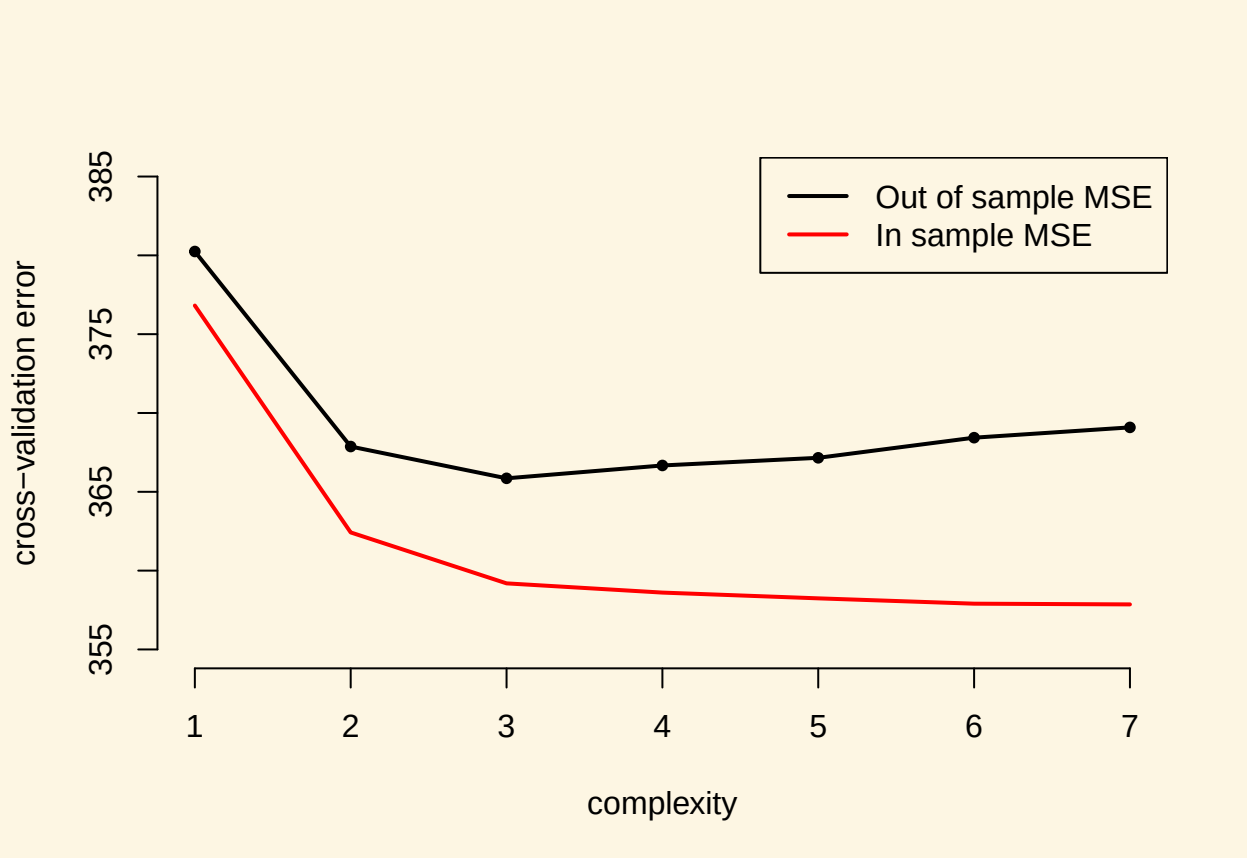
We will also record the in-sample prediction error to illustrate that we do need to test our models using new data rather than improving them in-sample due to the bias-variance trade-off.

```
# container for cv errors
cv.error <- NA

# container for in-sample MSE
in.sample.error <- NA

# loop over age raised to the power 1...7
```

```r
for (i in 1:7){

  glm.fit <- glm( IMMBRIT ~ poly(RAge, i), data = data2 )

  # cv error
  cv.error[i] <- cv.glm(data2, glm.fit)$delta[1]
  # in-sample mse
  in.sample.error[i] <- mean( (data2$IMMBRIT - predict(glm.fit, data2) )^2)

}
```

Next, we plot the effect of increasing the complexity of the model. We also plot the in-sample error

```r
# plot of error rates
plot( cv.error ~ seq(1, 7), bty = "n", pch = 20,
      xlab = "complexity", ylab = "cross-validation error",
      ylim = c(355, 385))
# cv error
lines( y = cv.error, x = seq(1,7), lwd = 2, col = 1)
# in-sample error
lines( y = in.sample.error, x = seq(1,7), lwd = 2, col = 2 )
# legend
legend("topright", c("Out of sample MSE", "In sample MSE"), col = c(1,2), lwd= 2)
```



Apparently, the cubic model performs best. We would have missed this using the initial split of the data into one training set and one test set. Furthermore, the in-sample MSE keeps decreasing the more complex we make our model (although with diminishing marginal returns). However, the more complex models start

fitting idiosyncratic aspects of the sample (noise) and perform badly with new data.

3.1.3 k-Fold Cross-Validation

K-fold cross-validation splits the datset into k datasets. Common choices for k are 5 and 10. Using 5, we would split the data into five folds. We would then train our model on the first fold and predict on the remaining folds. Next, we would train our model on the second fold and predict on the four remaining ones and so on until we train on the fifth fold and predict on the remaining folds. Each time we will get an error (e.g. MSE). We would then average over the five MSEs to obtain the overall k-fold cross-validation MSE.

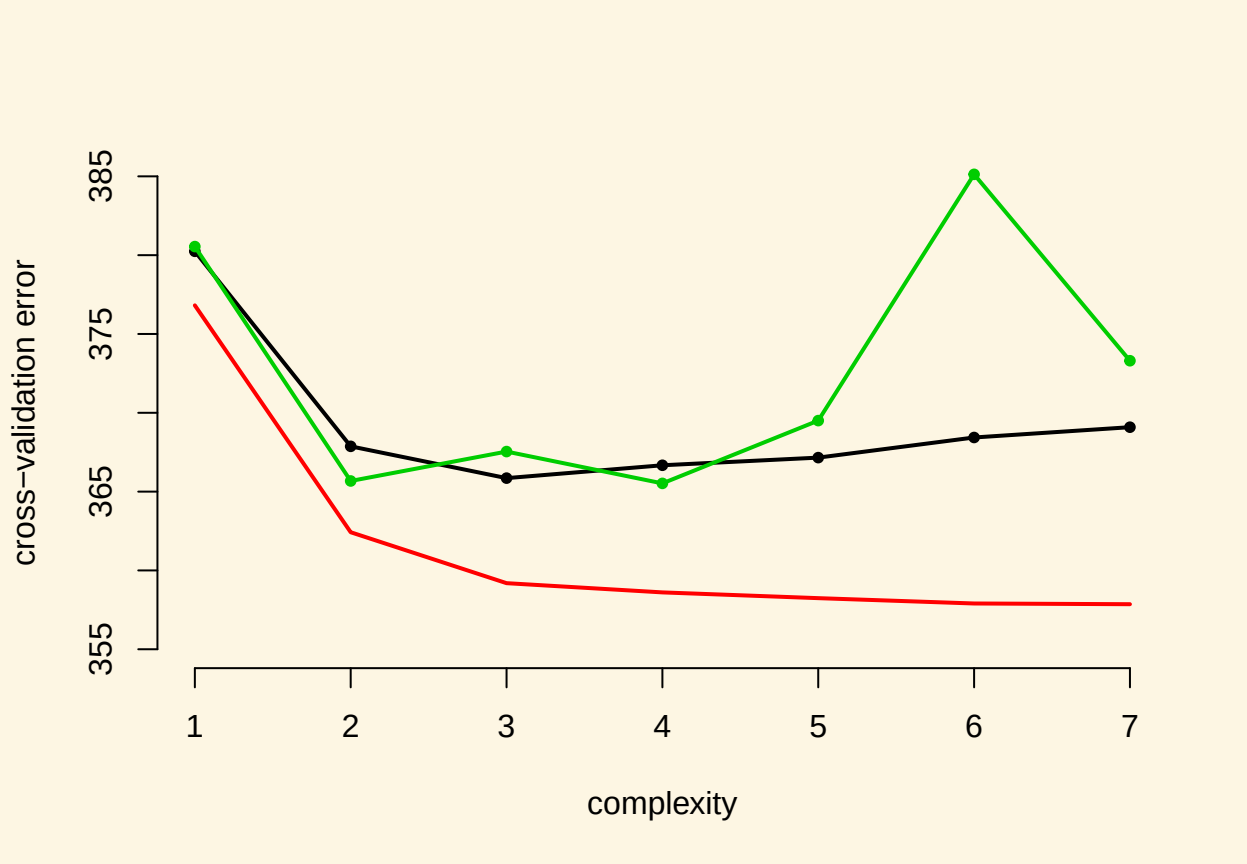
In addition to LOOCV, `cv.glm()` can also be used to run k-fold cross-validation. In the following example, we estimate the cross-validation error of polynomials of the order 1 through 7 using 10-fold cross-validation.

```
# re-initialize random number generator
set.seed(17)

# container for 10-fold cross-validation errors
cv.error.10 <- NA

# loop over 7 different powers of age
for (i in 1:7){
  glm.fit <- glm( IMMBRIT ~ poly(RAge, i), data = data2)
  cv.error.10[i] <- cv.glm( data2, glm.fit, K = 10)$delta[1]
}
cv.error.10
```

```
[1] 380.5447 365.6740 367.5397 365.5214 369.5038 385.1270 373.3029
```

We add the results to the plot:

```
# add to plot
points(x = seq(1,7), y = cv.error.10, col = 3, pch = 20)
lines( x = seq(1,7), y = cv.error.10, col = 3, lwd = 2)
```



The 10-fold cross-validation error is more wiggly. In this example, it estimates the best performance with a square model of age whereas the LOOCV error finds a minimum at the cube of age. Eyeballing the results, we suggest that there are no substantial improvements beyond the squared term. However, using the cubic model would be an alternative.

4 Subset Selection

4.1 Seminar

In this exercise, we learn how to select the best predictors out of a set of variables. This is sometimes referred to as a variable selection model (the Lasso which we introduce later falls into the same category).

We start by clearing our workspace.

```
# clear workspace
rm( list = ls() )
```

4.1.1 Subset Selection Methods

We use a modified data set on non-western immigrants (we inserted some missings). Download the data here.

The codebook is:

Variable Name	Description
IMMBRIT	Out of every 100 people in Britain, how many do you think are immigrants from Non-western countries?

Variable Name	Description
over.estimate	1 if estimate is higher than 10.7%.
RSex	1 = male, 2 = female
RAge	Age of respondent
Househld	Number of people living in respondent's household
Cons, Lab, SNP, Ukip, BNP, GP, party.other	Party self-identification
paper	Do you normally read any daily morning newspaper 3+ times/week?
WWWhourspW	How many hours WWW per week?
religious	Do you regard yourself as belonging to any particular religion?
employMonths	How many mnths w. present employer?
urban	Population density, 4 categories (highest density is 4, lowest is 1)
health.good	How is your health in general for someone of your age? (0: bad, 1: fair, 2: fairly good, 3: good)
HHInc	Income bands for household, high number = high HH income

```
# load foreigners data
load("your directory/BSAS_manip_missings.RData")
```

We check our dataset for missing values variable by variable using `apply()`, `is.na()`, and `table()`.

```
# check for missing values
apply(df, 2, function(x) table(is.na(x))["TRUE"] )
```

```
      IMMBRIT over.estimate          RSex          RAge      Househld
            8            NA            NA            NA            NA
         Cons           Lab           SNP          Ukip          BNP
           NA            NA            NA            NA            NA
           GP   party.other         paper    WWWhourspW     religious
           NA            NA            NA            NA            NA
 employMonths         urban   health.good         HHInc
           NA            NA            NA            NA
```

We next drop variables from the dataset.

```
# we drop missings in IMMBRIT
df <- df[ !is.na(df$IMMBRIT), ]
```

If you ever want to drop variables on an entire dataset you can run `df <- na.omit(df)`. If you want to use the same method for dropping a few variables, you can run `df[, c("some var", "some other var")] <- na.omit(df[, c("some var", "some other var")])`.

We now declare the categorical variables to be factors and create a copy of the main data set that excludes `over.estimate`.

```
# declare factor variables
df$urban <- factor(df$urban, labels = c("rural", "more rural", "more urban", "urban"))
df$RSex <- factor(df$RSex, labels = c("Male", "Female"))
df$health.good <- factor(df$health.good, labels = c("bad", "fair", "fairly good", "good") )

# drop the binary response coded 1 if IMMBRIT > 10.7
df$over.estimate <- NULL
df$Cons <- NULL
```

4.1.2 Best Subset Selection

We use the `regsubsets()` function to identify the best model based on subset selection quantified by the residual sum of squares (RSS) for each model. The function is included in the `leaps` package which we need to install if it is not already like so: `install.packages("leaps")`.

```
library(leaps)

# run best subset selection
regfit.full <- regsubsets(IMMBRIT ~ ., data = df)
summary(regfit.full)
```

```
Subset selection object
Call: regsubsets.formula(IMMBRIT ~ ., data = df)
20 Variables  (and intercept)
                       Forced in Forced out
RSexFemale                 FALSE      FALSE
RAge                       FALSE      FALSE
Househld                   FALSE      FALSE
Lab                        FALSE      FALSE
SNP                        FALSE      FALSE
Ukip                       FALSE      FALSE
BNP                        FALSE      FALSE
GP                         FALSE      FALSE
party.other                FALSE      FALSE
paper                      FALSE      FALSE
WWWhourspW                 FALSE      FALSE
religious                  FALSE      FALSE
employMonths               FALSE      FALSE
urbanmore rural            FALSE      FALSE
urbanmore urban            FALSE      FALSE
urbanurban                 FALSE      FALSE
health.goodfair            FALSE      FALSE
health.goodfairly good     FALSE      FALSE
health.goodgood            FALSE      FALSE
HHInc                      FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         RSexFemale RAge Househld Lab SNP Ukip BNP GP  party.other paper
1  ( 1 ) " "        " "  " "      " " " " " " " " " " " "         " "
2  ( 1 ) "*"        " "  " "      " " " " " " " " " " " "         " "
3  ( 1 ) "*"        " "  "*"      " " " " " " " " " " " "         " "
4  ( 1 ) "*"        " "  "*"      " " " " " " "*" " " " "         " "
5  ( 1 ) "*"        " "  "*"      "*" " " " " "*" " " " "         " "
6  ( 1 ) "*"        "*"  "*"      "*" " " " " "*" " " " "         " "
7  ( 1 ) "*"        "*"  "*"      "*" " " " " "*" " " " "         "*"
8  ( 1 ) "*"        "*"  "*"      "*" " " "*"  "*" " " " "         "*"
         WWWhourspW religious employMonths urbanmore rural urbanmore urban
1  ( 1 ) " "        " "       " "          " "              " "
2  ( 1 ) " "        " "       " "          " "              " "
3  ( 1 ) " "        " "       " "          " "              " "
4  ( 1 ) " "        " "       " "          " "              " "
5  ( 1 ) " "        " "       " "          " "              " "
6  ( 1 ) " "        " "       " "          " "              " "
7  ( 1 ) " "        " "       " "          " "              " "
```

```
8  ( 1 ) " "         " "        " "           " "              " "
          urbanurban health.goodfair health.goodfairly good health.goodgood
1  ( 1 ) " "         " "               " "                      " "
2  ( 1 ) " "         " "               " "                      " "
3  ( 1 ) " "         " "               " "                      " "
4  ( 1 ) " "         " "               " "                      " "
5  ( 1 ) " "         " "               " "                      " "
6  ( 1 ) " "         " "               " "                      " "
7  ( 1 ) " "         " "               " "                      " "
8  ( 1 ) " "         " "               " "                      " "
          HHInc
1  ( 1 ) "*"
2  ( 1 ) "*"
3  ( 1 ) "*"
4  ( 1 ) "*"
5  ( 1 ) "*"
6  ( 1 ) "*"
7  ( 1 ) "*"
8  ( 1 ) "*"
```

In the regression output, variables with a star were selected for the model. Here, we see 8 different models from 1 to 8 variables. With the `nvmax` parameter we control the number of variables in the model. The default used by `regsubsets()` is 8.

```
# increase the max number of variables
regfit.full <- regsubsets(IMMBRIT ~ ., data = df, nvmax = 16)
reg.summary <- summary(regfit.full)
```

We can look at the components of the `reg.summary` object using the `names()` function and examine the R^2 statistic stored in `rsq`.

```
names(reg.summary)
```

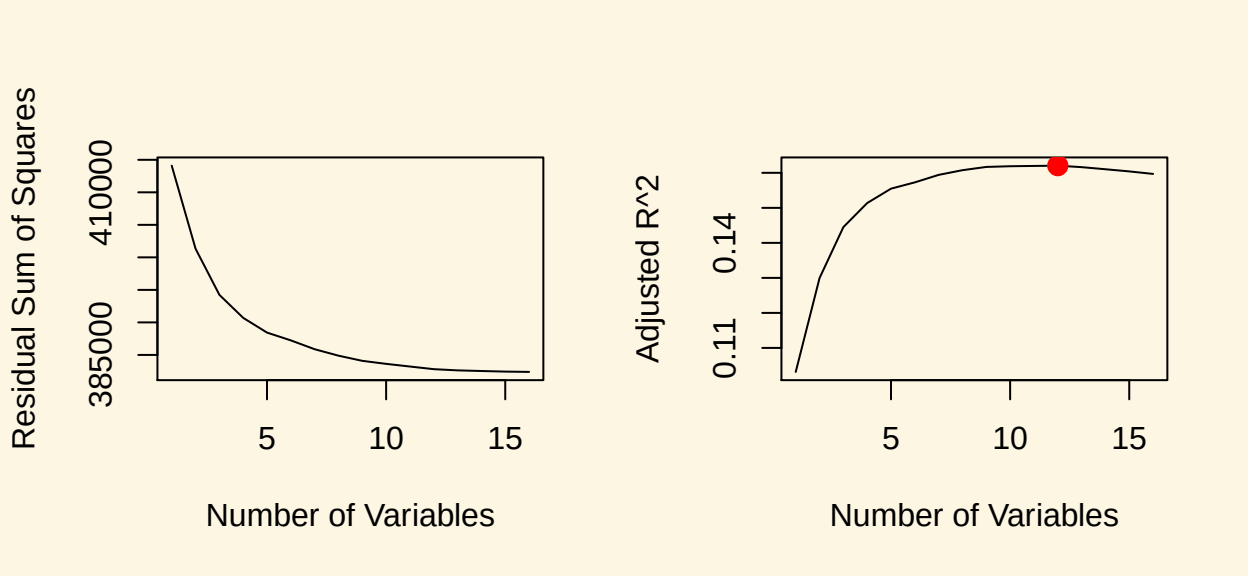
```
[1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
reg.summary$rsq
```

```
 [1] 0.1040145 0.1316228 0.1469748 0.1546376 0.1595363 0.1621379 0.1650727
 [8] 0.1672137 0.1689371 0.1699393 0.1708328 0.1717100 0.1721041 0.1723151
[15] 0.1725153 0.1726176
```

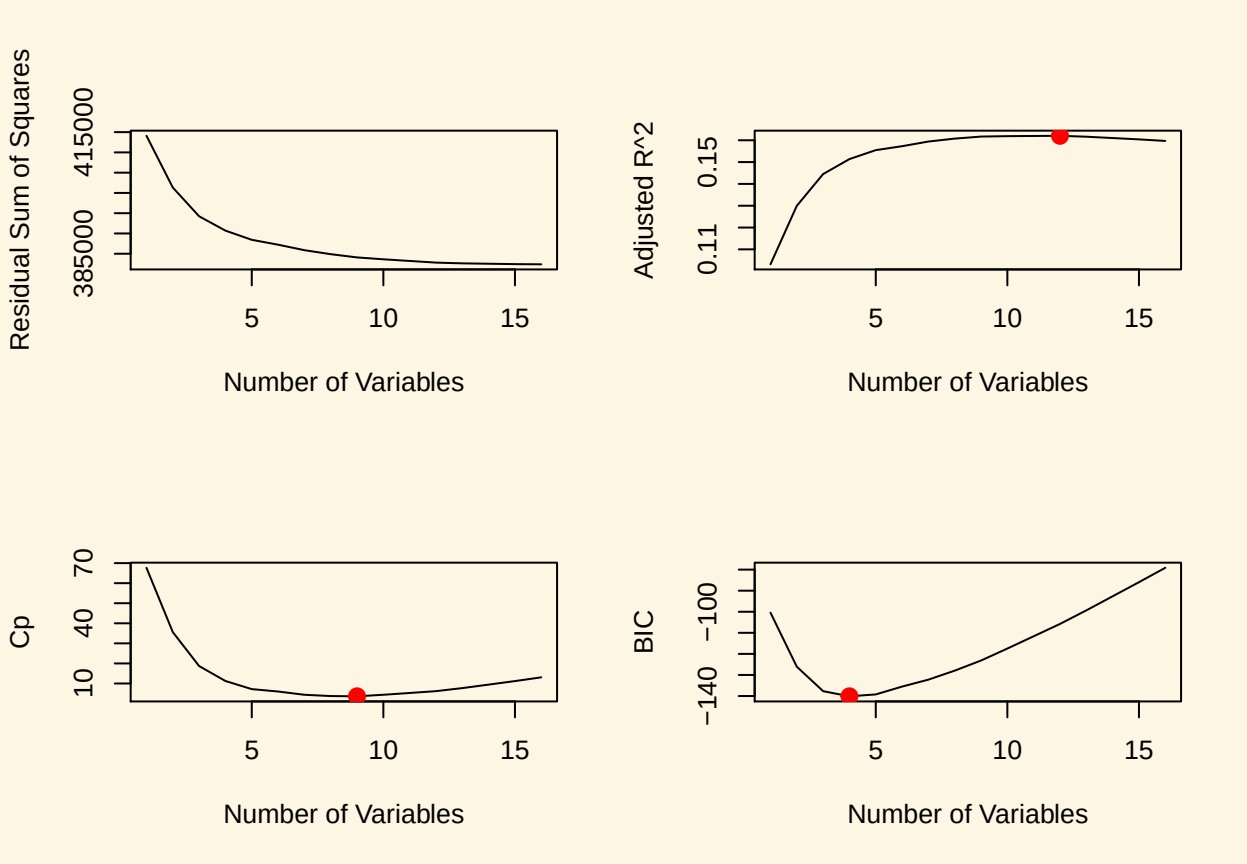
Next, we plot the RSS and adjusted R^2 and add a point where R^2 is at its maximum using the `which.max()` function.

```
par( mfrow =  c(2,2) ) # 2 row, 2 columns in plot window
plot(reg.summary$rss, xlab = "Number of Variables", ylab = "Residual Sum of Squares", type = "l")
plot(reg.summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted R^2", type = "l")

# find the peak of adj. R^2
adjr2.max <- which.max( reg.summary$adjr2 )
points(adjr2.max, reg.summary$adjr2[adjr2.max], col = "red", pch = 20, cex = 2)
```



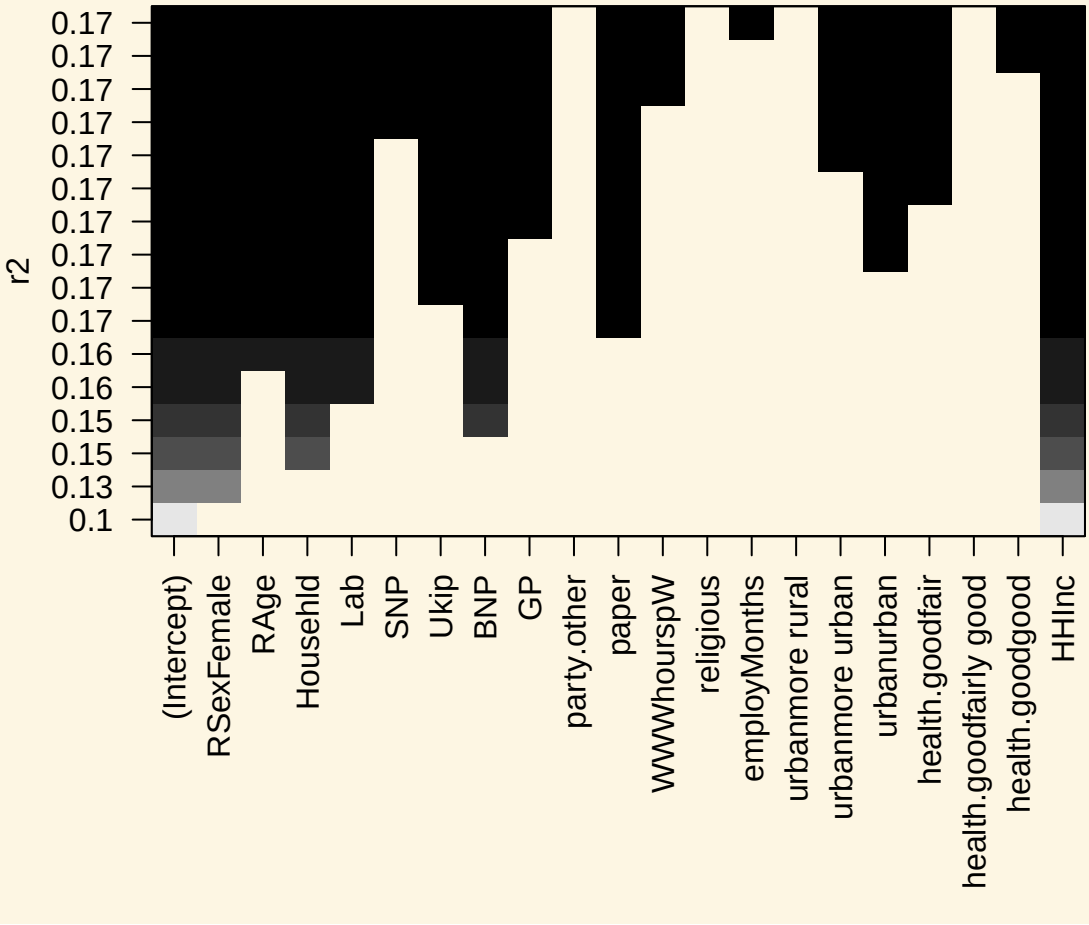
We can also plot the C_p statistic and BIC and identify the minimum points for each statistic using the *which.min*() function.

```
# cp
plot(reg.summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
cp.min <- which.min(reg.summary$cp)
points(cp.min, reg.summary$cp[cp.min], col = "red", cex = 2, pch = 20)

# bic
bic.min <- which.min(reg.summary$bic)
plot(reg.summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
points(bic.min, reg.summary$bic[bic.min], col = "red", cex = 2, pch = 20)
```

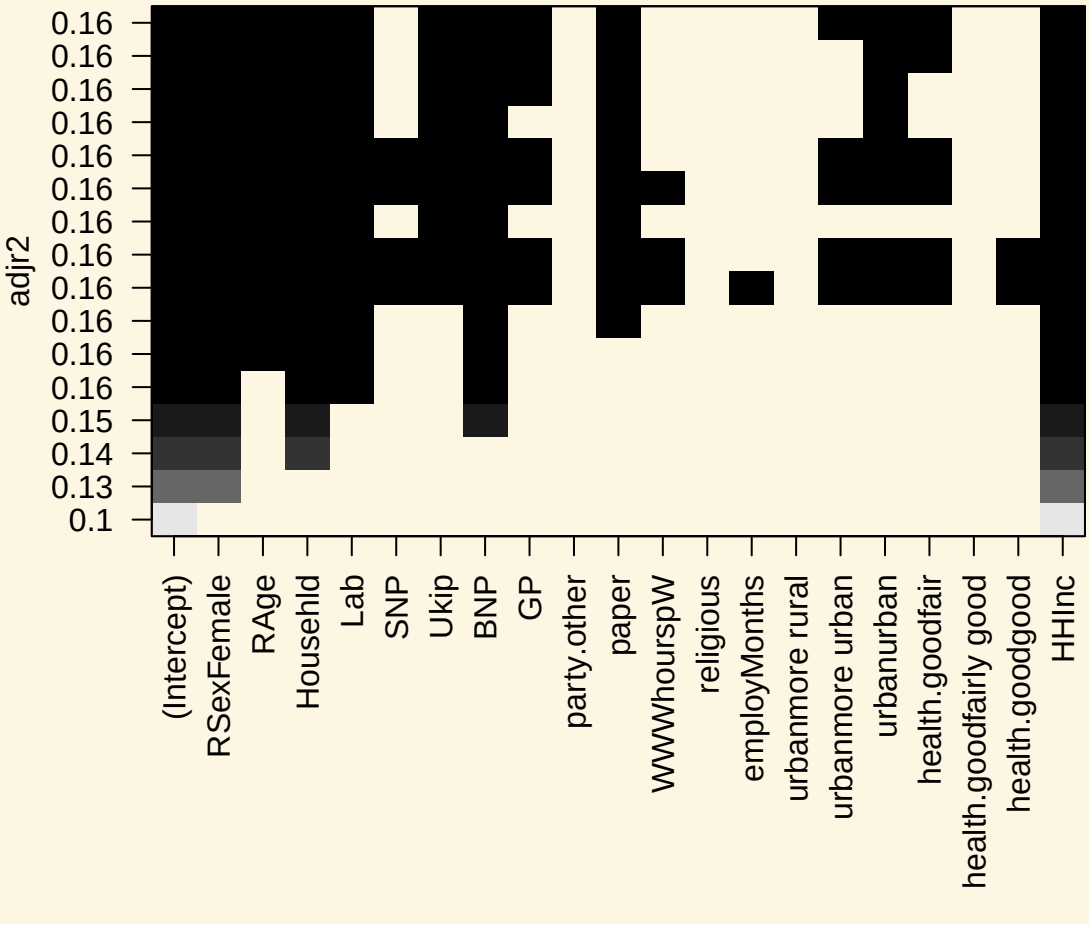


The estimated models from `regsubsets()` can be directly plotted to compare the differences based on the values of R^2, adjusted R^2, C_p and BIC statistics.

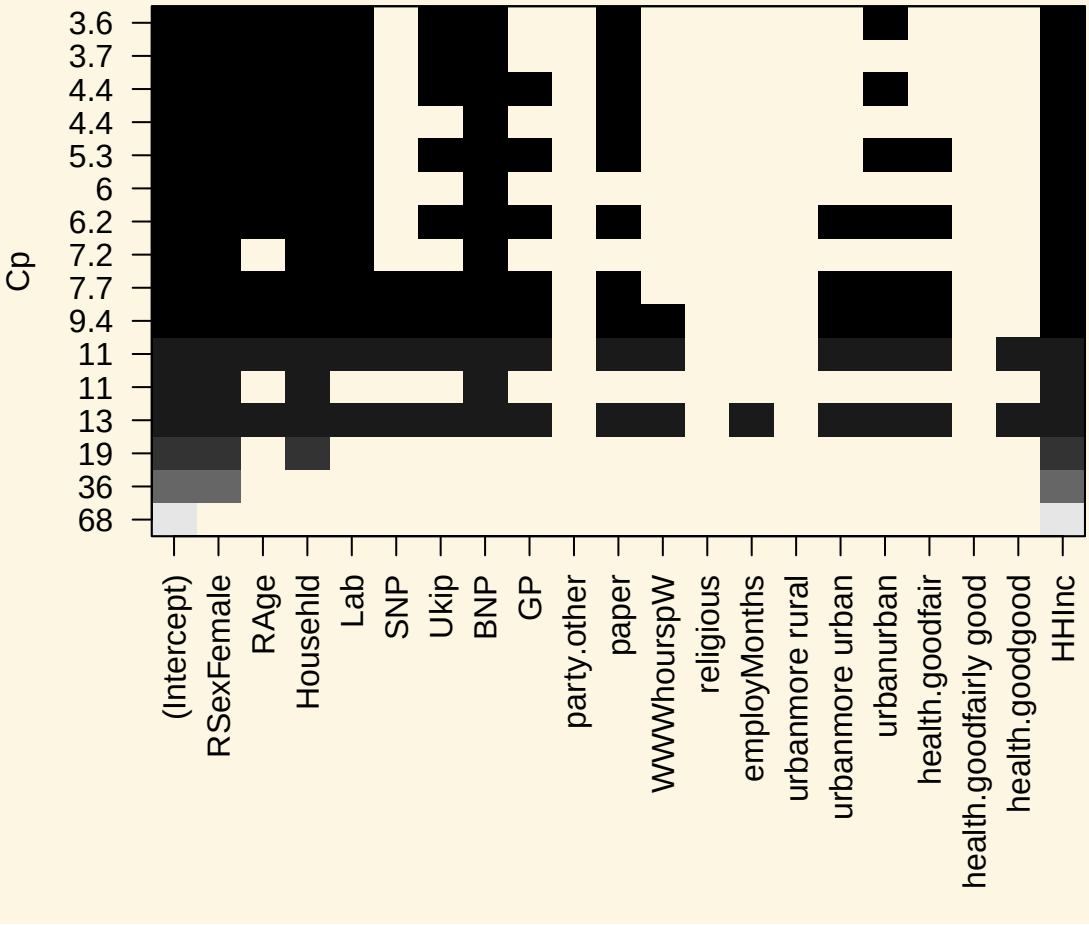
```
par( mfrow = c(1,1), oma = c(3,0,0,0))

# plot model comparison based on R^2
plot(regfit.full, scale = "r2")
```



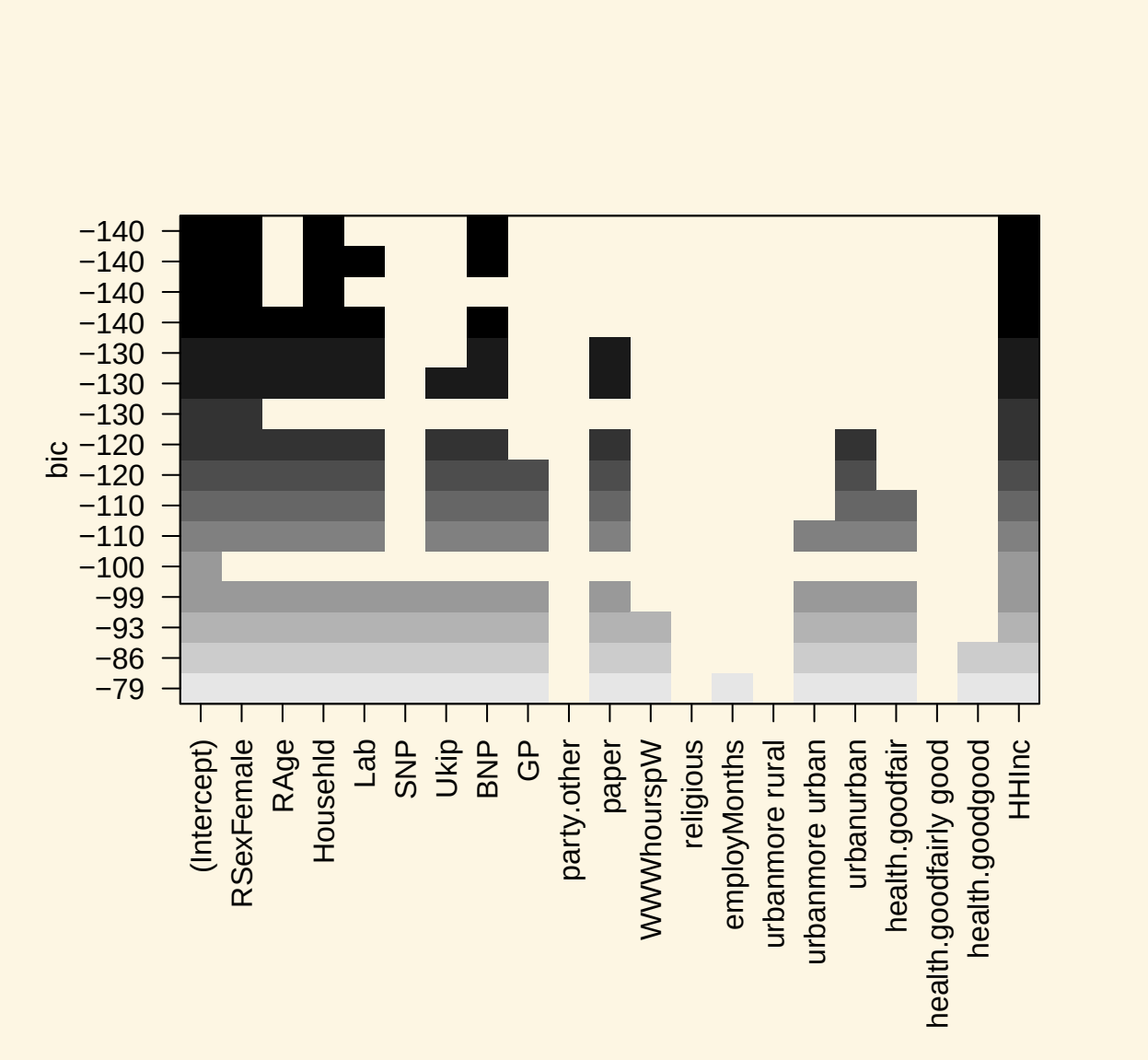

```
# plot model comparison based on adjusted R^2
plot(regfit.full, scale = "adjr2")
```




```
# plot model comparison based on adjusted CP
plot(regfit.full, scale = "Cp")
```




```
# plot model comparison based on adjusted BIC
plot(regfit.full, scale = "bic")
```



To show the coefficients associated with the model with the lowest BIC, we use the `coef()` function.

```
coef(regfit.full, bic.min)
```

```
(Intercept)  RSexFemale    Househld         BNP       HHInc
  34.576036    6.970692    2.000771   10.830195   -1.511176
```

4.1.3 Forward and Backward Stepwise Selection

The default method used by `regsubsets()` is exhaustive but we can change it to forward or backward and compare the results. Best subset selection will take very long with many variables because the number of models to estimate grows exponentially. In these cases, we will want to use forward and backward selection instead.

Let's carry out forward selection.

```
# run forward selection
regfit.fwd <- regsubsets(IMMBRIT ~ ., data = df, nvmax = 16, method = "forward")
summary(regfit.fwd)
```

```
Subset selection object
Call: regsubsets.formula(IMMBRIT ~ ., data = df, nvmax = 16, method = "forward")
20 Variables  (and intercept)
                       Forced in Forced out
RSexFemale                 FALSE      FALSE
RAge                       FALSE      FALSE
Househld                   FALSE      FALSE
Lab                        FALSE      FALSE
SNP                        FALSE      FALSE
Ukip                       FALSE      FALSE
BNP                        FALSE      FALSE
GP                         FALSE      FALSE
party.other                FALSE      FALSE
paper                      FALSE      FALSE
WWWhourspW                 FALSE      FALSE
religious                  FALSE      FALSE
employMonths               FALSE      FALSE
urbanmore rural            FALSE      FALSE
urbanmore urban            FALSE      FALSE
urbanurban                 FALSE      FALSE
health.goodfair            FALSE      FALSE
health.goodfairly good     FALSE      FALSE
health.goodgood            FALSE      FALSE
HHInc                      FALSE      FALSE
1 subsets of each size up to 16
Selection Algorithm: forward
          RSexFemale RAge Househld Lab SNP Ukip BNP GP  party.other paper
1  ( 1 )  " "        " "  " "      " " " " " " " " " " " "         " "
2  ( 1 )  "*"        " "  " "      " " " " " " " " " " " "         " "
3  ( 1 )  "*"        " "  "*"      " " " " " " " " " " " "         " "
4  ( 1 )  "*"        " "  "*"      " " " " " " "*" " " " "         " "
5  ( 1 )  "*"        " "  "*"      "*" " " " " "*" " " " "         " "
6  ( 1 )  "*"        "*"  "*"      "*" " " " " "*" " " " "         " "
7  ( 1 )  "*"        "*"  "*"      "*" " " " " "*" " " " "         "*"
8  ( 1 )  "*"        "*"  "*"      "*" " " "*" "*" " " " "         "*"
9  ( 1 )  "*"        "*"  "*"      "*" " " "*" "*" " " " "         "*"
10  ( 1 ) "*"        "*"  "*"      "*" " " "*" "*" "*" " "         "*"
11  ( 1 ) "*"        "*"  "*"      "*" " " "*" "*" "*" " "         "*"
12  ( 1 ) "*"        "*"  "*"      "*" " " "*" "*" "*" " "         "*"
13  ( 1 ) "*"        "*"  "*"      "*" "*" "*" "*" "*" " "         "*"
14  ( 1 ) "*"        "*"  "*"      "*" "*" "*" "*" "*" " "         "*"
15  ( 1 ) "*"        "*"  "*"      "*" "*" "*" "*" "*" " "         "*"
16  ( 1 ) "*"        "*"  "*"      "*" "*" "*" "*" "*" " "         "*"
          WWWhourspW religious employMonths urbanmore rural
1  ( 1 )  " "        " "       " "          " "
2  ( 1 )  " "        " "       " "          " "
3  ( 1 )  " "        " "       " "          " "
4  ( 1 )  " "        " "       " "          " "
5  ( 1 )  " "        " "       " "          " "
6  ( 1 )  " "        " "       " "          " "
7  ( 1 )  " "        " "       " "          " "
8  ( 1 )  " "        " "       " "          " "
9  ( 1 )  " "        " "       " "          " "
10  ( 1 ) " "        " "       " "          " "
```

```
11  ( 1 ) " "         " "        " "           " "
12  ( 1 ) " "         " "        " "           " "
13  ( 1 ) " "         " "        " "           " "
14  ( 1 ) "*"         " "        " "           " "
15  ( 1 ) "*"         " "        " "           " "
16  ( 1 ) "*"         " "        "*"           " "
          urbanmore urban urbanurban health.goodfair
1   ( 1 )  " "              " "         " "
2   ( 1 )  " "              " "         " "
3   ( 1 )  " "              " "         " "
4   ( 1 )  " "              " "         " "
5   ( 1 )  " "              " "         " "
6   ( 1 )  " "              " "         " "
7   ( 1 )  " "              " "         " "
8   ( 1 )  " "              " "         " "
9   ( 1 )  " "              "*"         " "
10  ( 1 ) " "               "*"         " "
11  ( 1 ) " "               "*"         "*"
12  ( 1 ) "*"               "*"         "*"
13  ( 1 ) "*"               "*"         "*"
14  ( 1 ) "*"               "*"         "*"
15  ( 1 ) "*"               "*"         "*"
16  ( 1 ) "*"               "*"         "*"
          health.goodfairly good health.goodgood HHInc
1   ( 1 )  " "                      " "              "*"
2   ( 1 )  " "                      " "              "*"
3   ( 1 )  " "                      " "              "*"
4   ( 1 )  " "                      " "              "*"
5   ( 1 )  " "                      " "              "*"
6   ( 1 )  " "                      " "              "*"
7   ( 1 )  " "                      " "              "*"
8   ( 1 )  " "                      " "              "*"
9   ( 1 )  " "                      " "              "*"
10  ( 1 ) " "                       " "              "*"
11  ( 1 ) " "                       " "              "*"
12  ( 1 ) " "                       " "              "*"
13  ( 1 ) " "                       " "              "*"
14  ( 1 ) " "                       " "              "*"
15  ( 1 ) " "                       "*"              "*"
16  ( 1 ) " "                       "*"              "*"
```

Let's carry out backwards selection next.

```
# run backward selection
regfit.bwd <- regsubsets(IMMBRIT ~ ., data = df, nvmax = 16, method = "backward")
summary(regfit.bwd)
```

```
Subset selection object
Call: regsubsets.formula(IMMBRIT ~ ., data = df, nvmax = 16, method = "backward")
20 Variables  (and intercept)
                     Forced in Forced out
RSexFemale               FALSE      FALSE
RAge                     FALSE      FALSE
Househld                 FALSE      FALSE
Lab                      FALSE      FALSE
```

```
SNP                         FALSE      FALSE
Ukip                        FALSE      FALSE
BNP                         FALSE      FALSE
GP                          FALSE      FALSE
party.other                 FALSE      FALSE
paper                       FALSE      FALSE
WWWhourspW                  FALSE      FALSE
religious                   FALSE      FALSE
employMonths                FALSE      FALSE
urbanmore rural             FALSE      FALSE
urbanmore urban             FALSE      FALSE
urbanurban                  FALSE      FALSE
health.goodfair             FALSE      FALSE
health.goodfairly good      FALSE      FALSE
health.goodgood             FALSE      FALSE
HHInc                       FALSE      FALSE
1 subsets of each size up to 16
Selection Algorithm: backward
          RSexFemale RAge Househld Lab SNP Ukip BNP GP  party.other paper
1  ( 1 )  " "        " "  " "      " " " " " " " " " " " "         " "
2  ( 1 )  "*"        " "  " "      " " " " " " " " " " " "         " "
3  ( 1 )  "*"        " "  "*"      " " " " " " " " " " " "         " "
4  ( 1 )  "*"        " "  "*"      " " " " " " "*" " " " "         " "
5  ( 1 )  "*"        " "  "*"      "*" " " " " "*" " " " "         " "
6  ( 1 )  "*"        "*"  "*"      "*" " " " " "*" " " " "         " "
7  ( 1 )  "*"        "*"  "*"      "*" " " " " "*" " " " "         "*"
8  ( 1 )  "*"        "*"  "*"      "*" " " "*"  "*" " " " "         "*"
9  ( 1 )  "*"        "*"  "*"      "*" " " "*"  "*" " " " "         "*"
10  ( 1 ) "*"        "*"  "*"      "*" " " "*"  "*" "*" " "         "*"
11  ( 1 ) "*"        "*"  "*"      "*" " " "*"  "*" "*" " "         "*"
12  ( 1 ) "*"        "*"  "*"      "*" " " "*"  "*" "*" " "         "*"
13  ( 1 ) "*"        "*"  "*"      "*" "*" "*"  "*" "*" " "         "*"
14  ( 1 ) "*"        "*"  "*"      "*" "*" "*"  "*" "*" " "         "*"
15  ( 1 ) "*"        "*"  "*"      "*" "*" "*"  "*" "*" " "         "*"
16  ( 1 ) "*"        "*"  "*"      "*" "*" "*"  "*" "*" " "         "*"
          WWWhourspW religious employMonths urbanmore rural
1  ( 1 )  " "        " "       " "          " "
2  ( 1 )  " "        " "       " "          " "
3  ( 1 )  " "        " "       " "          " "
4  ( 1 )  " "        " "       " "          " "
5  ( 1 )  " "        " "       " "          " "
6  ( 1 )  " "        " "       " "          " "
7  ( 1 )  " "        " "       " "          " "
8  ( 1 )  " "        " "       " "          " "
9  ( 1 )  " "        " "       " "          " "
10  ( 1 ) " "        " "       " "          " "
11  ( 1 ) " "        " "       " "          " "
12  ( 1 ) " "        " "       " "          " "
13  ( 1 ) " "        " "       " "          " "
14  ( 1 ) "*"        " "       " "          " "
15  ( 1 ) "*"        " "       " "          " "
16  ( 1 ) "*"        " "       "*"          " "
          urbanmore urban urbanurban health.goodfair
1  ( 1 )  " "              " "         " "
```

```
2  ( 1 )  " "             " "         " "
3  ( 1 )  " "             " "         " "
4  ( 1 )  " "             " "         " "
5  ( 1 )  " "             " "         " "
6  ( 1 )  " "             " "         " "
7  ( 1 )  " "             " "         " "
8  ( 1 )  " "             " "         " "
9  ( 1 )  " "             "*"         " "
10  ( 1 ) " "             "*"         " "
11  ( 1 ) " "             "*"         "*"
12  ( 1 ) "*"             "*"         "*"
13  ( 1 ) "*"             "*"         "*"
14  ( 1 ) "*"             "*"         "*"
15  ( 1 ) "*"             "*"         "*"
16  ( 1 ) "*"             "*"         "*"
          health.goodfairly good health.goodgood HHInc
1  ( 1 )  " "                    " "              "*"
2  ( 1 )  " "                    " "              "*"
3  ( 1 )  " "                    " "              "*"
4  ( 1 )  " "                    " "              "*"
5  ( 1 )  " "                    " "              "*"
6  ( 1 )  " "                    " "              "*"
7  ( 1 )  " "                    " "              "*"
8  ( 1 )  " "                    " "              "*"
9  ( 1 )  " "                    " "              "*"
10  ( 1 ) " "                    " "              "*"
11  ( 1 ) " "                    " "              "*"
12  ( 1 ) " "                    " "              "*"
13  ( 1 ) " "                    " "              "*"
14  ( 1 ) " "                    " "              "*"
15  ( 1 ) " "                    "*"              "*"
16  ( 1 ) " "                    "*"              "*"
```

Then we compare the models.

```
# model coefficients of best 7-variable models
coef(regfit.full, 7)
```

```
(Intercept)  RSexFemale        RAge    Household         Lab         BNP
40.02553709  7.14423868 -0.08205116  1.70838328 -3.34664442  9.11326764
      paper       HHInc
 2.37633989 -1.60436490
```

```
coef(regfit.fwd, 7)
```

```
(Intercept)  RSexFemale        RAge    Household         Lab         BNP
40.02553709  7.14423868 -0.08205116  1.70838328 -3.34664442  9.11326764
      paper       HHInc
 2.37633989 -1.60436490
```

```
coef(regfit.bwd, 7)
```

```
(Intercept)  RSexFemale        RAge    Household         Lab         BNP
40.02553709  7.14423868 -0.08205116  1.70838328 -3.34664442  9.11326764
      paper       HHInc
 2.37633989 -1.60436490
```

In this case, all methods arrived at the same conclusion. This will not always be the case. We can arrive at very different conclusions.

4.1.3.1 Choosing Among Models Using the Validation Set Approach and Cross-Validation

For validation set approach, we split the dataset into a training subset and a test subset. In order to ensure that the results are consistent over multiple iterations, we set the random seed with `set.seed()` before calling `sample()`.

```
set.seed(1)

# sample true or false for each observation
train <- sample( c(TRUE, FALSE), size = nrow(df), replace = TRUE )
# the complement
test <- (!train)
```

We use `regsubsets()` as we did in the last section, but limit the estimation to the training subset.

```
regfit.best <- regsubsets(IMMBRIT ~ ., data = df[train, ], nvmax = 16)
```

We create a matrix from the test subset using `model.matrix()`. Model matrix takes the dependent variable out of the data and adds an intercept to it.

```
# test data
test.mat <- model.matrix(IMMBRIT ~., data = df[test, ])
```

Next, we compute the validation error for each model.

```
# validation error for each model
val.errors <- NA
for (i in 1:16 ){

  coefi <- coef(regfit.best, id = i)
  # this prediction without the predict function for a linear model
  # here we use linear algebra operations were we multiply the data matrix
  # with the coefficient vector
  y_hat <- test.mat[, names(coefi)] %*% coefi

  # we compute the test set MSE for each model
  val.errors[i] <- mean(  (df$IMMBRIT[test] - y_hat)^2   )
}
```

We examine the validation error for each model and identify the best model with the lowest error.

```
val.errors
```

```
 [1] 422.6788 433.3656 422.1876 423.2052 419.1279 428.0234 425.6056
 [8] 424.7795 425.5773 427.3771 428.2082 427.7471 426.4746 427.7618
[15] 426.7668 426.3854
```

```
# which model has smallest error
min.val.errors <- which.min(val.errors)

# coefficients of that model
coef( regfit.best, min.val.errors )
```

```
(Intercept)  RSexFemale        RAge    Household         Lab       HHInc
 47.9382527   6.2858758  -0.1880005   1.5486251  -5.1619813  -1.7403107
```

We can combine these steps into a function that can be called repeatedly when running k-fold cross-validation.

```
# predict function for repeatedly choosing model with lowest test error
predict.regsubsets <- function( object, newdata, id, ... ){
  # get the formula from the model
  m.formula <- as.formula( object$call[[2]] )
  # use that formula to create the model matrix for some new data
  mat <- model.matrix( m.formula, newdata )
  # get coeffiecients where id is the number of variables
  coefi <- coef( object, id = id )
  # get the variable names of current model
  xvars <- names( coefi )
  # multiply data with coefficients
  mat[ , xvars ] %*% coefi
}
```

We run `regsubsets()` on the full dataset and examine the coefficients associated with the model that has the lower validation error.

```
# best subset on full data set
regfit.best <- regsubsets( IMMBRIT ~ ., data = df, nvmax = 16 )

# examine coefficients of the model that had the lowest validation error
coef( regfit.best, min.val.errors )
```

```
(Intercept)  RSexFemale    Househld         Lab         BNP       HHInc
  35.920518    6.886259    2.062351   -3.394188    9.708402   -1.563859
```

4.1.3.2 k-fold cross-validation

For cross-validation, we create the number of folds needed (10, in this case) and allocate a matrix for storing the results.

```
# number of folds
k <- 10
set.seed(1)

# fold assignment for each observation
folds <- sample(1:k, nrow(df), replace = TRUE)

# frequency table of fold assignment (should be relatively balanced)
table(folds)
```

```
folds
  1   2   3   4   5   6   7   8   9  10
 99  83 104 106 110 101 112 100 114 112
```

Let's create an object that will store errors for cross-validation and we then look at that.

```
# container for cross-validation errors
cv.errors <- matrix(NA, nrow = k, ncol = 16, dimnames = list(NULL, paste(1:16)))
# have a look at the matrix
cv.errors
```

```
      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
 [1,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [2,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [3,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [4,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

```
 [5,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [6,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [7,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [8,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 [9,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[10,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

We then run through each fold in a `for()` loop and predict the salary using our predict function. We then calculate the validation error for each fold and save them in the matrix created above.

```
# loop over folds
for (a in 1:k){

  # best subset selection on training data
  best.fit <- regsubsets(IMMBRIT ~ ., data = df[ folds != a, ], nvmax = 16)

  # loop over the 16 subsets
  for (b in 1:16){

    # predict response for test set for current subset
    pred <- predict(best.fit, df[ folds == a ,], id = b )

    # MSE into container; rows are folds; columns are subsets
    cv.errors[a, b] <- mean( (df$IMMBRIT[folds==a] - pred)^2 )

  } # end of loop over the 16 subsets
} # end of loop over folds
# the cross-validation error matrix
cv.errors
```

```
            1        2        3        4        5        6        7
 [1,] 340.6348 360.4143 354.1625 361.5282 354.2123 357.4392 352.4973
 [2,] 452.1263 416.6181 421.1696 423.3392 412.5419 416.3733 415.5115
 [3,] 347.7890 332.3682 325.9761 326.3122 306.9565 310.4999 303.0287
 [4,] 313.2735 337.9460 334.7280 333.0075 325.7449 329.7096 326.7136
 [5,] 386.3464 367.9026 375.7936 384.1864 380.3926 384.2020 380.7137
 [6,] 491.0775 484.0378 480.4096 487.4987 480.6689 478.1541 474.7581
 [7,] 490.9417 479.0285 475.3201 473.5372 481.2985 473.6737 482.0662
 [8,] 457.2409 426.8484 407.9413 407.6739 404.1949 412.6668 415.4127
 [9,] 381.4168 370.7018 358.7797 366.7294 362.5290 363.9266 361.1322
[10,] 343.8041 317.2183 324.9596 325.2615 333.3371 342.0212 347.9536
            8        9       10       11       12       13       14
 [1,] 351.4881 349.5941 350.7354 350.9006 349.8262 348.9046 348.9941
 [2,] 420.6110 427.7911 431.2058 426.3387 419.2075 419.2498 420.0484
 [3,] 305.4976 306.6941 309.1712 310.9844 311.0525 313.8237 314.9565
 [4,] 325.9687 326.9600 328.4326 327.0528 325.3324 326.4723 325.8034
 [5,] 382.5723 388.3581 388.0053 387.1782 388.6989 383.5634 383.1115
 [6,] 474.7654 474.0053 475.5215 471.2198 473.1165 473.3898 472.3392
 [7,] 473.5635 470.9286 466.7013 465.5243 469.4845 467.6969 469.9937
 [8,] 414.4015 407.5812 408.8584 409.2999 410.0517 407.8849 410.3678
 [9,] 353.5598 357.7010 351.5392 352.7202 353.8711 354.3456 352.3022
[10,] 337.8455 337.9669 341.7642 343.5873 348.0795 350.2935 348.9332
           15       16
 [1,] 348.8001 348.3123
 [2,] 421.0930 419.7895
```

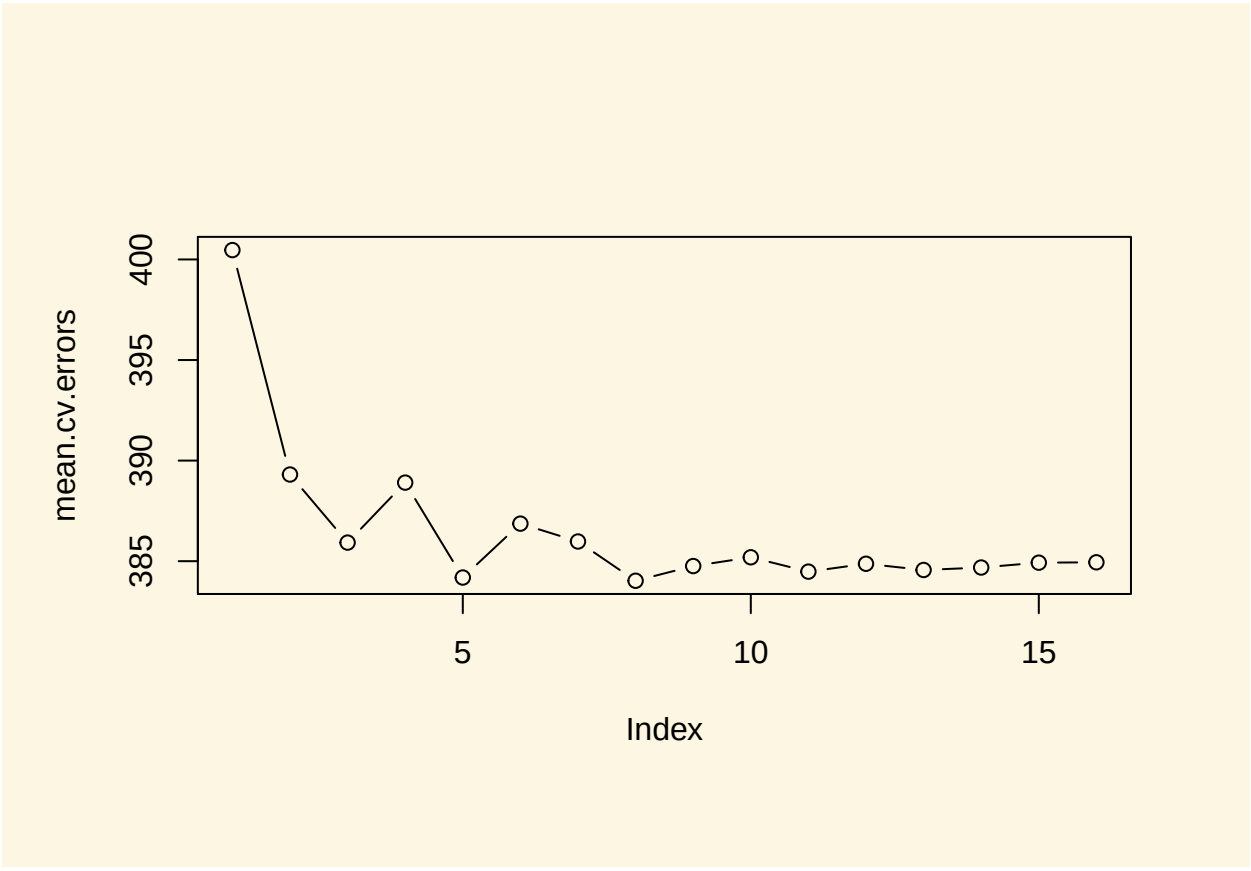
```
 [3,] 314.5607 314.9318
 [4,] 325.5250 325.1381
 [5,] 381.1764 381.5495
 [6,] 473.5295 473.1426
 [7,] 469.6818 469.2276
 [8,] 410.8454 410.9011
 [9,] 352.7259 354.9532
[10,] 351.3258 351.4992
```

We calculate the mean error for all subsets by applying mean to each column using the `apply()` function.

```
# average cross-validation errors over the folds
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
       1        2        3        4        5        6        7        8
400.4651 389.3084 385.9240 388.9074 384.1876 386.8666 385.9788 384.0273
       9       10       11       12       13       14       15       16
384.7580 385.1935 384.4806 384.8721 384.5624 384.6850 384.9264 384.9445
```

```
# visualize
par( mfrow = c(1,1) , oma = c(0,0,0,0))
plot( mean.cv.errors, type = "b" )
```



Finally, we run `regsubsets()` on the full dataset and show the coefficients for the best performing model which we picked using 10-fold cross-validation.

```
# run regsubsets on full data set
reg.best <- regsubsets(IMMBRIT ~ ., data = df, nvmax = 16)
```

```
# coefficients of subset which minimized test error
coef(reg.best, which.min(mean.cv.errors))
```

```
(Intercept)  RSexFemale        RAge    Househld         Lab        Ukip
40.45488881  7.04808104 -0.08129894  1.67265081 -3.60995595 -5.90702725
        BNP       paper       HHInc
 8.81702077  2.46964179 -1.61703898
```

5 Regularization

5.1 Seminar

5.1.1 Ridge Regression and the Lasso

We start by clearing our workspace, loading the foreigners data, and doing the necessary variable manipulations. The data is available here.

We then need to normalize all numeric variables to put them on the same scale. Regularization requires that variables are comparable.

```
# clear workspace
rm(list=ls())

# load foreigners data
load("your directory/BSAS_manip.RData")
head(data2)

# we declare the factor variables
data2$urban <- factor(data2$urban, labels = c("rural", "more rural", "more urban", "urban"))
data2$RSex <- factor(data2$RSex, labels = c("Male", "Female"))
data2$health.good <- factor(data2$health.good, labels = c("bad", "fair", "fairly good", "good") )

# categorical variables
cat.vars <- unlist(lapply(data2, function(x) is.factor(x) | all(x == 0 | x==1) | all( x==1 | x==2) ))
# normalize numeric variables
data2[, !cat.vars] <- apply(data2[, !cat.vars], 2, scale)
```

In order to run ridge regression, we create a matrix from our dataset using the `model.matrix()` function. We also need to remove the intercept from the resulting matrix because the function to run ridge regression automatically includes one. Furthermore, we will use the subjective rate of immigrants as response. Consequently, we have to remove `over.estimate` as it measures the same thing. Lastly, the party affiliation dummies are mutually exclusive, so we have to exclude the model category `Cons`.

```
# covariates in matrix form but remove the intercept, over.estimate, and Cons
x <- model.matrix(IMMBRIT ~ . -1 -over.estimate -Cons, data2)
# check if it looks fine
head(x)
```

```
  RSexMale RSexFemale       RAge   Househld Lab SNP Ukip BNP GP
1        1          0  0.0144845 -0.2925308   1   0    0   0  0
2        0          1 -1.8065476  0.4540989   0   0    0   0  0
3        0          1  0.5835570 -1.0391604   0   0    0   0  0
4        0          1  1.5509804 -0.2925308   0   0    0   0  0
5        0          1  0.9819078 -1.0391604   0   0    0   0  0
6        1          0 -1.1236606  1.2007285   0   0    0   0  0
  party.other paper WWWhourspW religious employMonths urbanmore rural
```

```
1           0     0 -0.5324636         0    -0.203378               0
2           1     0 -0.1566702         0    -0.203378               0
3           1     0 -0.5324636         0     5.158836               0
4           1     1 -0.4071991         1    -0.203378               0
5           1     0 -0.5324636         1    -0.203378               0
6           1     1  1.0959747         0    -0.203378               0
  urbanmore urban urbanurban health.goodfair health.goodfairly good
1               0          1               1                       0
2               0          1               0                       1
3               1          0               0                       0
4               0          0               0                       0
5               1          0               0                       0
6               0          0               0                       1
  health.goodgood      HHInc
1               0  0.7357918
2               0 -1.4195993
3               1 -0.1263647
4               1 -0.3419038
5               1 -0.1263647
6               0 -0.1263647
```

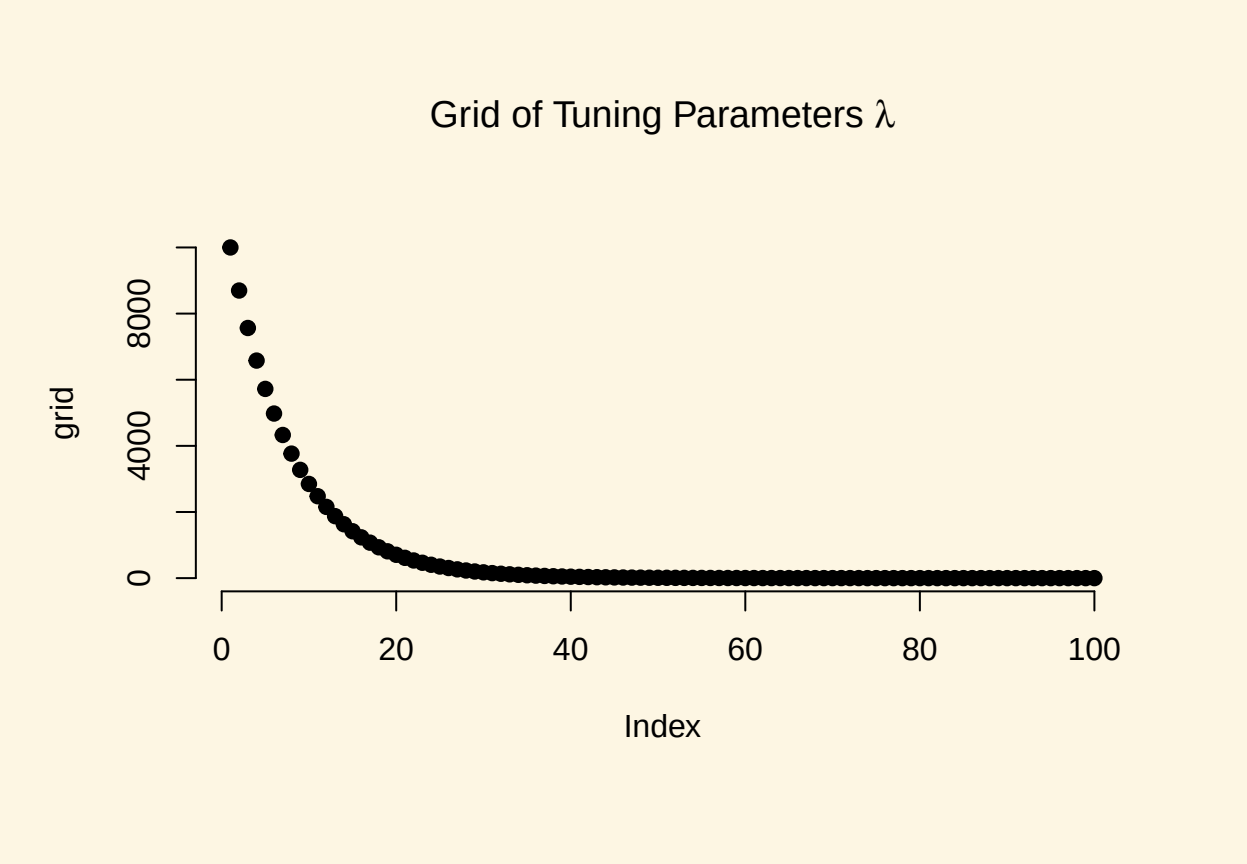
```r
# response vector
y <- data2$IMMBRIT
```

5.1.1.1 Ridge Regression

The `glmnet` package provides functionality to fit ridge regression and lasso models. We load the package and call `glmnet()` to perform ridge regression. Before being able to run this, we have to install the package like so: `install.packages("glmnet")`.

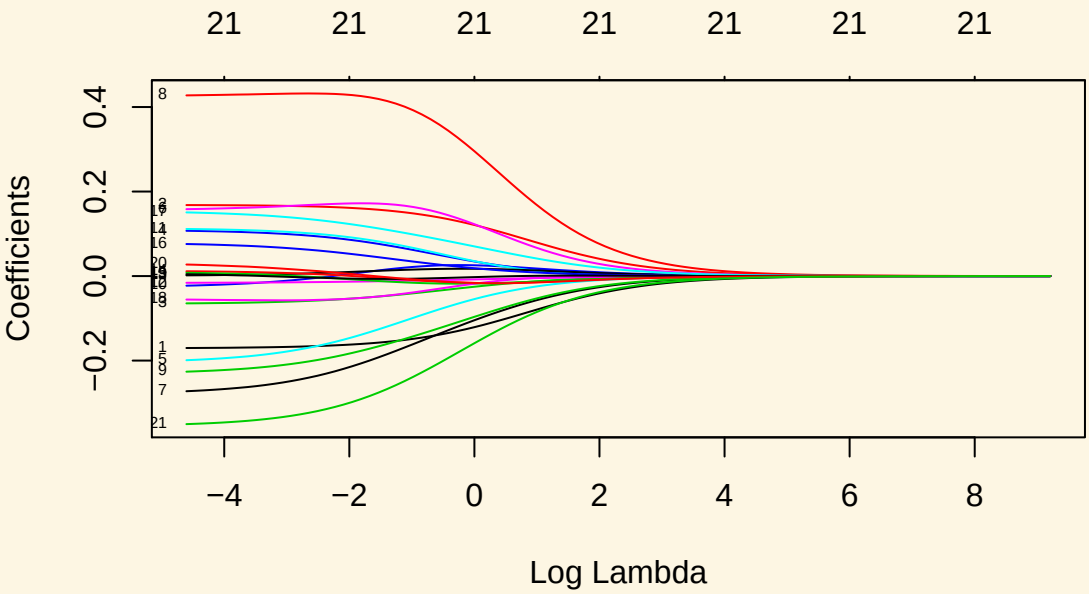
The performance of ridge depends on the right choice of lambda. A tuning parameter is a parameter that we need to set and we need to set correctly. We do this by trying different values. All different values are what we refer to as our grid.

```r
library(glmnet)

# tuning parameter
grid <- 10^seq(4, -2, length = 100)

plot(grid, bty = "n", pch = 19,
     main = expression(paste("Grid of Tuning Parameters ", lambda)))
```



We now run ridge regression. We tune `lambda` and set `alpha` to 0 which means we carry out ridge regression (instead as for instance the Lasso or the Elastic Net).

```
# run ridge; alpha = 0 means do ridge
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)

# coefficient shrinkage visualized
plot(ridge.mod, xvar = "lambda", label = TRUE)
```



the object `ridge.mod` contains a set of coefficients for each of the lambdas which we can access by running the `coef()` function on the object `ridge.mod`. We tried 100 lambda values and therefore we get 100 coefficient sets. The object is a matrix where rows are variables and columns are the coefficients based on the chosen lambda values.

```
# a set of coefficients for each lambda
dim(coef(ridge.mod))
```

```
[1]  22 100
```

We can look at the coefficients at different values for λ. Here, we randomly choose two different values and notice that smaller values of λ result in larger coefficient estimates and vice-versa.

```
# Lambda and Betas
ridge.mod$lambda[80]
```

```
[1] 0.1629751
```

```
coef(ridge.mod)[, 80]
```

```
  (Intercept)      RSexMale    RSexFemale
-0.061607390 -0.160606935  0.159900421
        RAge      Househld         Lab
-0.051728204  0.082363992 -0.138922683
         SNP         Ukip         BNP
 0.172199820 -0.206749980  0.425986910
          GP  party.other        paper
-0.176955926  0.008250000  0.088331267
   WWWhourspW    religious employMonths
```

```
   -0.012922053             0.010790919            -0.001149770
urbanmore rural        urbanmore urban              urbanurban
   -0.009100653             0.049660346             0.119249438
health.goodfair health.goodfairly good        health.goodgood
   -0.051779228            -0.006372962             0.002867504
          HHInc
   -0.291172989
```

```
sqrt( sum(coef(ridge.mod)[-1, 80]^2) )
```

```
[1] 0.6911836
```

```
ridge.mod$lambda[40]
```

```
[1] 43.28761
```

```
coef(ridge.mod)[, 40]
```

```
     (Intercept)               RSexMale             RSexFemale
   -0.0024035442           -0.0086089993            0.0086089995
           RAge                Househld                     Lab
   -0.0009024882            0.0009302020           -0.0016851561
            SNP                    Ukip                     BNP
    0.0051999098           -0.0051751362            0.0145257558
             GP             party.other                   paper
   -0.0045065860            0.0018619440            0.0002790212
      WWWhourspW                religious            employMonths
   -0.0005069474            0.0015920318           -0.0017699007
urbanmore rural        urbanmore urban              urbanurban
   -0.0014946533            0.0006218758            0.0040369054
health.goodfair health.goodfairly good        health.goodgood
    0.0002138639            0.0003043203           -0.0019519248
          HHInc
   -0.0072791705
```

```
sqrt(sum(coef(ridge.mod)[-1, 40]^2))
```

```
[1] 0.02287352
```

We can get ridge regression coefficients for any value of λ using predict.

```
# compute coefficients at lambda = s
predict(ridge.mod, s = 50, type = "coefficients")[1:nrow(coef(ridge.mod)), ]
```

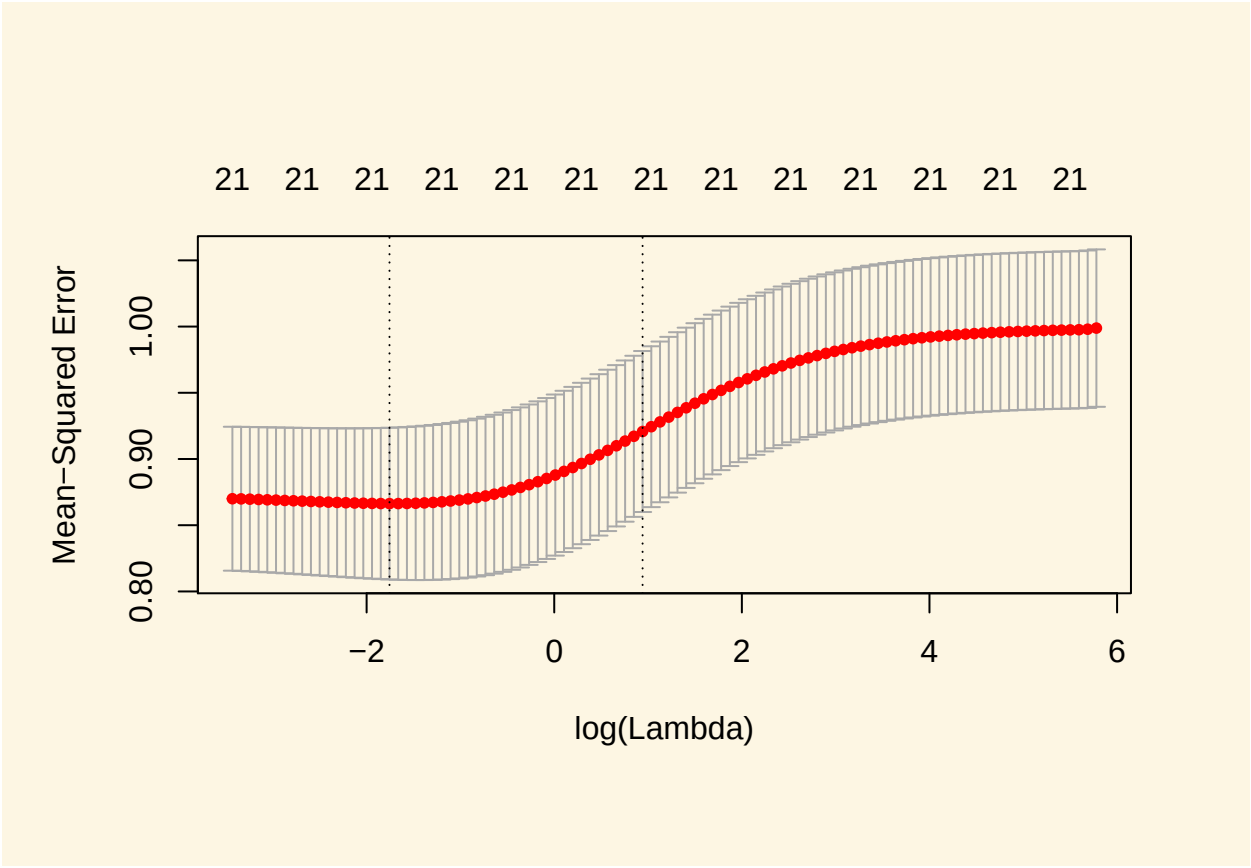
```
     (Intercept)               RSexMale             RSexFemale
   -0.0020916615           -0.0075062040            0.0075062041
           RAge                Househld                     Lab
   -0.0007828438            0.0008050887           -0.0014604733
            SNP                    Ukip                     BNP
    0.0045109868           -0.0045029585            0.0126229971
             GP             party.other                   paper
   -0.0039161618            0.0016221081            0.0002331180
      WWWhourspW                religious            employMonths
   -0.0004418724            0.0013894830           -0.0015446673
urbanmore rural        urbanmore urban              urbanurban
   -0.0013036526            0.0005397837            0.0035149833
health.goodfair health.goodfairly good        health.goodgood
    0.0001920935            0.0002676711           -0.0017039940
```

```
              HHInc
-0.0063276702
```

We would like to know which value of lambda gives us the model with the best predictive power. We use cross-validation on ridge regression by first splitting the dataset into training and test subsets.

We can choose different values for λ by running cross-validation on ridge regression using `cv.glmnet()`.

```
set.seed(1)

# training data for CV to find optimal lambda, but then test data to estimate test error
cv.out <- cv.glmnet(x, y, alpha = 0, nfolds = 5)

# illustrate test MSE based on size of lambda
plot(cv.out)
```




```
# best performing model's lambda value
bestlam <- cv.out$lambda.min
bestlam
```

```
[1] 0.1726934
```

The best performing model is the one with $\lambda = 0.1726934$. We can also extract the mean cross-validated error of the best model.

```
cv.out$cvm[ which(cv.out$lambda == bestlam) ]
```

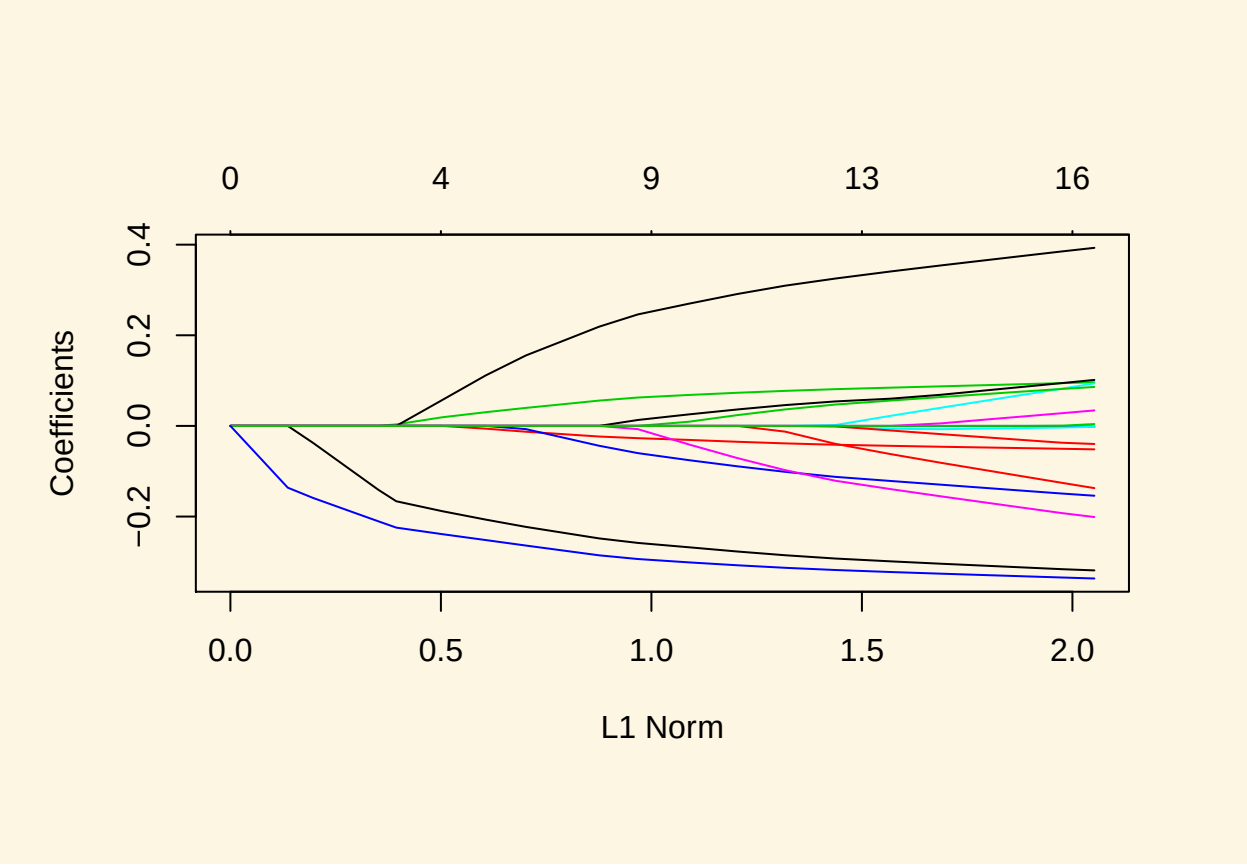
```
[1] 0.8663222
```

5.1.1.2 The Lasso

The lasso model can be estimated in the same way as ridge regression. The `alpha = 1` parameter tells `glmnet()` to run lasso regression instead of ridge regression. Lasso is often used more as a variable selection model because a large shrinkage parameter λ can cause coefficients of some variables to be exactly zero which means that those variables are excluded from the model.

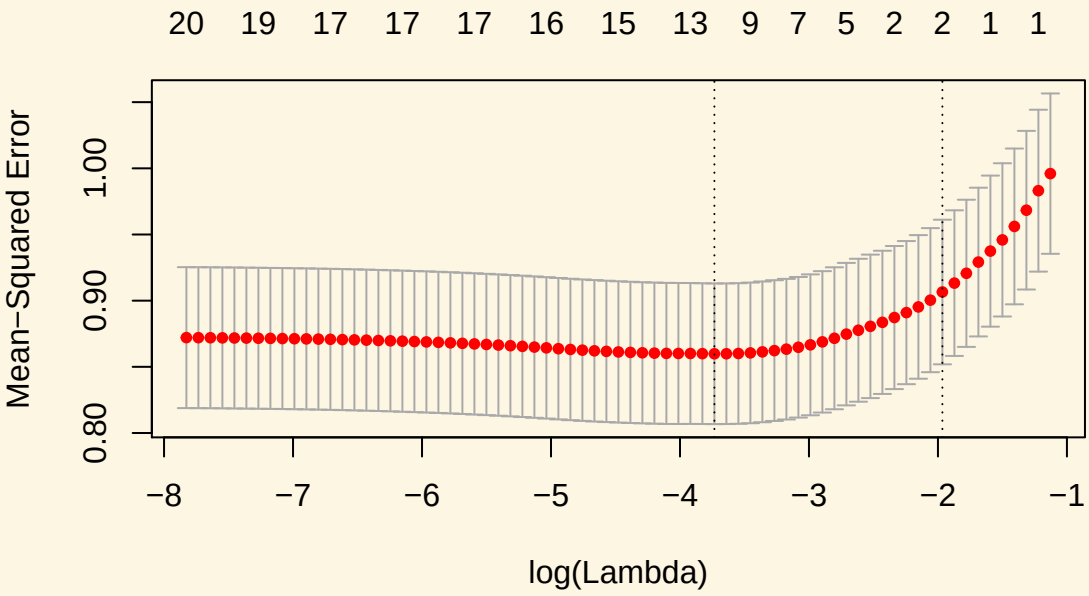
```
lasso.mod <- glmnet(x, y, alpha = 1, lambda = grid)
plot(lasso.mod)
```

```
Warning in regularize.values(x, y, ties, missing(ties)): collapsing to
unique 'x' values
```



Similarly, we can perform cross-validation using identical step as we did on ridge regression.

```
# cross-validation to pick lambda
set.seed(1)
cv.out <- cv.glmnet(x, y, alpha = 1, nfolds = 5)
plot(cv.out)
```



We select the best Lambda value and the cross-validation error.

```
bestlam <- cv.out$lambda.min
cv.out$cvm[ which(cv.out$lambda == bestlam) ]
```

```
[1] 0.8598552
```

```
# compare to ridge regression
out <- glmnet(x, y, alpha = 1, lambda = grid)
lasso.coef <- predict(out, type = "coefficients", s = bestlam)[1:16, ]
lasso.coef[lasso.coef != 0]
```

```
  (Intercept)       RSexMale           RAge        Househld
  0.123666790  -0.290927784  -0.040834908   0.080195229
          Lab           SNP          Ukip           BNP
 -0.109775668   0.001382463  -0.115410323   0.321492204
           GP         paper urbanmore rural
 -0.033150926   0.044601323  -0.001227853
```

6 All non-linear (polynomials to splines)

6.1 Seminar

In this exercise, we will learn how to model non-linear relationships using generalized linear models. This exercise is based on based on James et al. 2013. We begin by loading that `ISLR` package and attaching to the Wage dataset that we will be using throughout this exercise. When we attach a dataset, we do not need to write `dataset.name$variable.name` to access a variable but we can instead just write `variable.name` to access it.

Note: We need to install the `ISLR` package if it is not installed already like so: `install.packages("ISLR")`
Note2: The `Wage` dataset is spelled with a capital W.

```
# clear workspace, load ISLR, attach wage data set
rm(list=ls())
library(ISLR)
attach(Wage)
?Wage # codebook
```

6.1.1 Polynomial Regression

Let's fit a linear model to predict `wage` with a forth-degree polynomial using the `poly()` function.

Note: The dependent variable `wage` is spelled with a lower case w.

```
# linear regression on wage, with age up to a 4th degree polynomial
fit <- lm(wage ~ poly(age, 4), data = Wage)
coef(summary(fit))
```

```
                Estimate Std. Error    t value     Pr(>|t|)
(Intercept)    111.70361  0.7287409 153.283015 0.000000e+00
poly(age, 4)1  447.06785 39.9147851  11.200558 1.484604e-28
poly(age, 4)2 -478.31581 39.9147851 -11.983424 2.355831e-32
poly(age, 4)3  125.52169 39.9147851   3.144742 1.678622e-03
poly(age, 4)4  -77.91118 39.9147851  -1.951938 5.103865e-02
```

We can also obtain raw instead of orthogonal polynomials by passing the `raw = TRUE` argument to `poly()`. The coefficients will change the fit should be largely unaffected. It is not advisable to use the raw argument because it introduces unnecessary multicolinearity into the model.

```
fit2 <- lm(wage ~ poly(age, 4, raw = TRUE), data = Wage)
coef(summary(fit2))
```

```
                             Estimate   Std. Error   t value
(Intercept)                -1.841542e+02 6.004038e+01 -3.067172
poly(age, 4, raw = TRUE)1   2.124552e+01 5.886748e+00  3.609042
poly(age, 4, raw = TRUE)2  -5.638593e-01 2.061083e-01 -2.735743
poly(age, 4, raw = TRUE)3   6.810688e-03 3.065931e-03  2.221409
poly(age, 4, raw = TRUE)4  -3.203830e-05 1.641359e-05 -1.951938
                              Pr(>|t|)
(Intercept)                0.0021802539
poly(age, 4, raw = TRUE)1  0.0003123618
poly(age, 4, raw = TRUE)2  0.0062606446
poly(age, 4, raw = TRUE)3  0.0263977518
poly(age, 4, raw = TRUE)4  0.0510386498
```

There are several ways to specify polynomials. These are, however a little less convenient.

```
fit2a <- lm(wage ~ age + I(age^2) + I(age^3) + I(age^4), data = Wage)
coef(fit2a)
```

```
  (Intercept)           age      I(age^2)      I(age^3)      I(age^4)
-1.841542e+02  2.124552e+01 -5.638593e-01  6.810688e-03 -3.203830e-05
```

A more compact version of the same example uses `cbind()` and eliminates the need to wrap each term in `I()`. The output is less readable though.

```
fit2b <- lm(wage ~ cbind(age, age^2, age^3, age^4), data = Wage)
coef(fit2b)
```

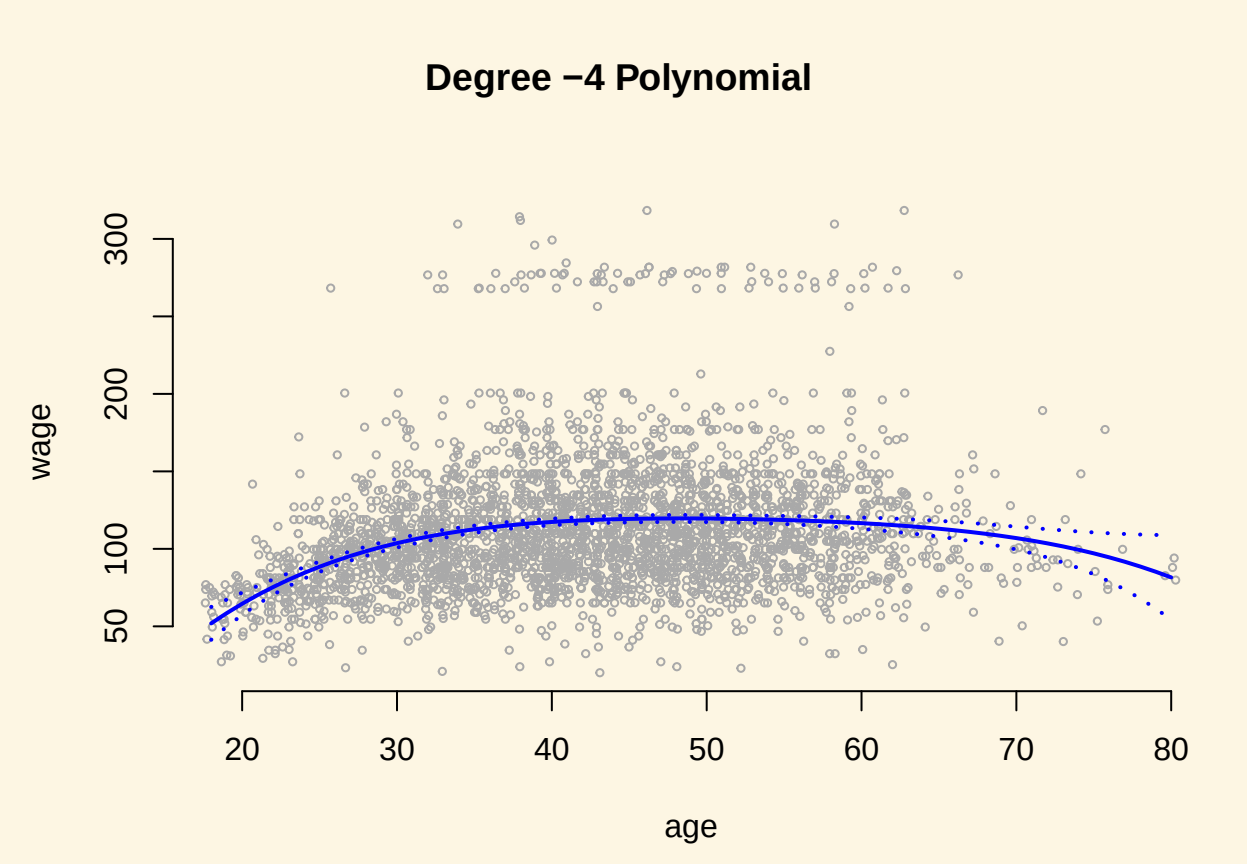
```
                    (Intercept) cbind(age, age^2, age^3, age^4)age
                  -1.841542e+02                       2.124552e+01
cbind(age, age^2, age^3, age^4)    cbind(age, age^2, age^3, age^4)
                  -5.638593e-01                       6.810688e-03
cbind(age, age^2, age^3, age^4)
                  -3.203830e-05
```

We can create an age grid (minimum age to maximum age) and pass the grid to `predict()`. We can set the argument `se=TRUE` in the `predict()` function which will return a list that includes standard errors of the outcome. We can use these to an upper and lower bound of our estimate of y.

```
# minimum and maximum values of age variable
agelims <- range(age)
age.grid <- seq(from = agelims[1], to = agelims[2])

# se=TRUE returns standard errors
preds <- predict(fit, newdata = list(age = age.grid), se = TRUE)

# confidence intervals as estimate + and - 2 standard deviations
se.bands <- cbind(preds$fit + 2 * preds$se.fit, preds$fit - 2 * preds$se.fit)
```

We can plot the data and add the fit from the degree-4 polynomial. We set the margins and outer margins in our plot the later plot a title that will be the overall title for two plots that we plot next to each other. The function `matlines()` lets us draw the lines for the uncertainty bounds in one go.

```
# set margins to plot title in margins
par(mfrow = c(1, 2), mar = c(4.5, 4.5, 1, 1), oma = c(0, 0, 4, 0))

plot(wage ~ jitter(age,2), xlim = agelims, cex = 0.5, col = "darkgrey", bty = "n",
     xlab = "age")

# overall plot window title
title("Degree -4 Polynomial ", outer = TRUE)

# line for mean estimate
lines(age.grid, preds$fit, lwd = 2, col = "blue")

# ~95% ci's
matlines(age.grid, se.bands, lwd = 2, col = "blue", lty = 3)
```



We compare the orthogonolized polynomials that we saved in the object called `fit` with the polynomials that plain polynomials saved in `fit2`. The difference will be close to 0. We predict the outcome from the fit with the raw polynomials and take the difference to the fit with the independent linear combinations of the powers of age.

```
preds2 <- predict(fit2, newdata = list(age = age.grid), se = TRUE)

# average difference
mean(preds$fit - preds2$fit)
```

```
[1] -1.752311e-11
```

```
# maximum difference
max(abs(preds$fit - preds2$fit))
```

```
[1] 7.81597e-11
```

When we have only predictor variable and and its powers we use the `coef()` function to see whether the powers of the variable improve in-sample model fit.

```
fit.5 <- lm(wage ~ poly(age, 5), data = Wage)
coef(summary(fit.5))
```

```
                Estimate Std. Error     t value     Pr(>|t|)
(Intercept)    111.70361  0.7287647 153.2780243 0.000000e+00
poly(age, 5)1  447.06785 39.9160847  11.2001930 1.491111e-28
poly(age, 5)2 -478.31581 39.9160847 -11.9830341 2.367734e-32
poly(age, 5)3  125.52169 39.9160847   3.1446392 1.679213e-03
poly(age, 5)4  -77.91118 39.9160847  -1.9518743 5.104623e-02
```

```
poly(age, 5)5  -35.81289 39.9160847  -0.8972045 3.696820e-01
```

With more variables, we use the `anova()` function and look at the F-test to decide whether in-sample fit improves by including powers of a variable.

```
fit.1 <- lm(wage ~ age, data = Wage)
fit.2 <- lm(wage ~ poly(age, 2), data = Wage)
fit.3 <- lm(wage ~ poly(age, 3), data = Wage)
fit.4 <- lm(wage ~ poly(age, 4), data = Wage)
anova(fit.1, fit.2, fit.3, fit.4, fit.5)
```

```
Analysis of Variance Table

Model 1: wage ~ age
Model 2: wage ~ poly(age, 2)
Model 3: wage ~ poly(age, 3)
Model 4: wage ~ poly(age, 4)
Model 5: wage ~ poly(age, 5)
  Res.Df     RSS Df Sum of Sq        F    Pr(>F)
1   2998 5022216
2   2997 4793430  1    228786 143.5931 < 2.2e-16 ***
3   2996 4777674  1     15756   9.8888  0.001679 **
4   2995 4771604  1      6070   3.8098  0.051046 .
5   2994 4770322  1      1283   0.8050  0.369682
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

With `glm()` we can also fit a polynomial logistic regression. Here, we create a binary variable that is 1 if wage > 250 and 0 otherwise.

```
fit <- glm(I(wage > 250) ~ poly(age, 4), data = Wage, family = binomial)
```

Similar to `lm()` we use the `predict()` function again and also obtain standard errors by setting `se=TRUE`.

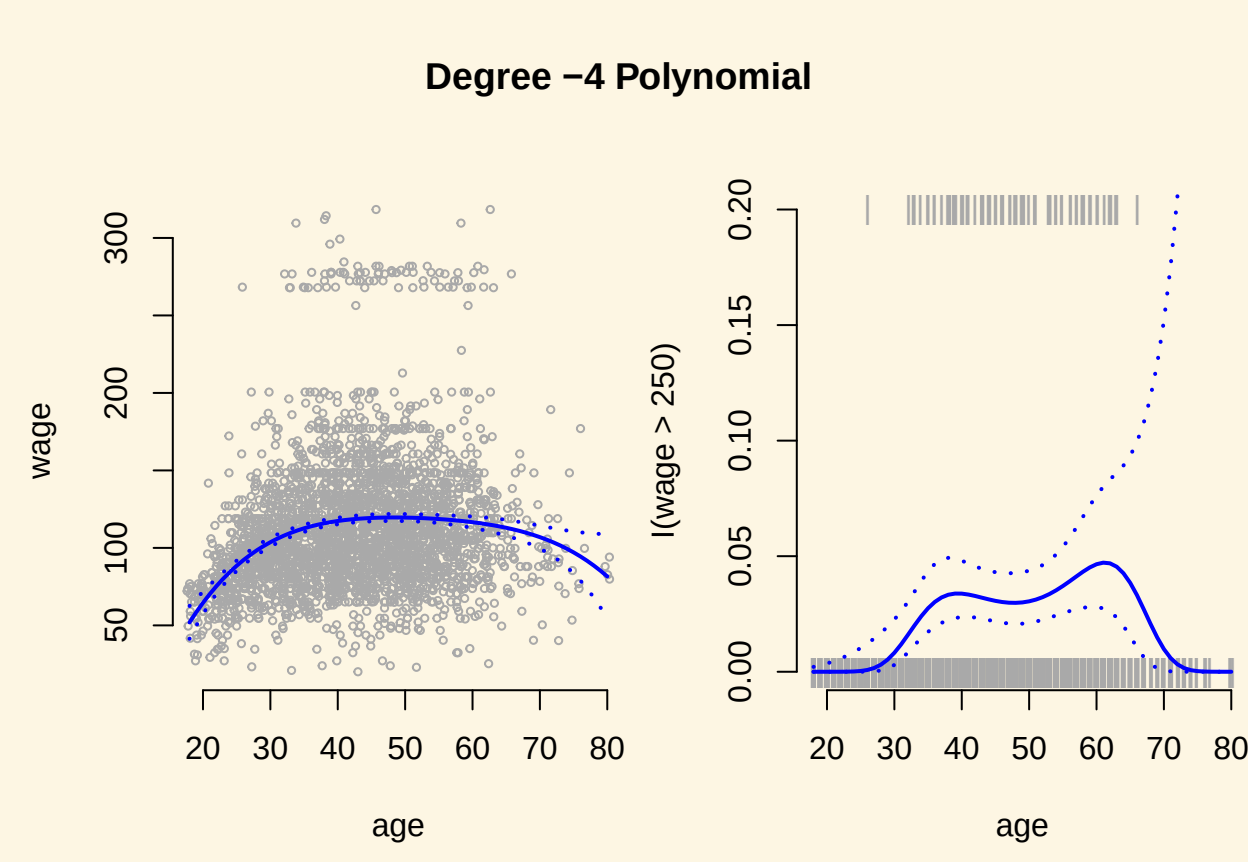
Note: If we do **not** set `type="response"` in the `predict()` function, we get the latent y as $X\beta$. We have to send those values through the link function to get predicted probabilities. We do this, so that we can estimate the standard errors on the latent y. We then send these through the link function as well. This ensures that our confidence intervals will never be outside the logical $[0, 1]$ interval for probabilities. If we would not do this, we could get standard errors outside the $[0, 1]$ interval.

```
# predict latent y
preds <- predict(fit, newdata = list(age = age.grid), se = TRUE)

# send latent y through the link function
pfit <- 1 / (1 + exp(-preds$fit))

# error bands calculate on the latent y
se.bands.logit <- cbind(preds$fit + 2 * preds$se.fit, preds$fit - 2 * preds$se.fit)
se.bands <- 1 / (1 + exp(-se.bands.logit))
```

We add the results next to the plot where wage is continuous. With the `points()` function we add the actual data to the plot. The argument `pch="|"` draws a bar as the symbol for each point. Also notice the y-coordinate of each point. In the `plot()` function we set the range of the y-axis with `ylim = c(0, 0.2)` to range from 0 to 0.2. If the true outcome is 1 we want to draw the | at $y = 0.2$ and otherwise at $y = 0$. We achieve this with `I((wage > 250)/5)`. Play around to see why.

```
plot(I(wage > 250) ~ age, xlim = agelims, type = "n", ylim = c(0, 0.2))
# add data to the plot
```

```
points(jitter(age), I((wage > 250)/5) , cex = 1, pch = "|", col = " darkgrey ")
# mean estimate
lines(age.grid, pfit, lwd = 2, col = "blue")
# 95 ci
matlines(age.grid, se.bands, lwd = 2, col = "blue", lty = 3)
```



Notice, that the confidence interval becomes very large in the range of the data where we have few data and no 1's.

6.1.2 Step Functions

Instead of using polynomials to create a non-linear prediction, we could also use step functions. With step functions we fit different lines for different data ranges.

We use the `cut()` function to create equally spaced cut-points in our data. We use the now categorical variable age as predictor in our linear model.

```
# four equally spaced intervals of age
table(cut(age, 4))
```

```
(17.9,33.5]   (33.5,49]   (49,64.5] (64.5,80.1]
        750        1399         779          72
```

```
# fit the linear regression with the factor variable age that has four categories
fit <- lm(wage ~ cut(age, 4), data = Wage)
# the first category is the baseline.
coef(summary(fit))
```

```
                        Estimate Std. Error   t value     Pr(>|t|)
(Intercept)            94.158392   1.476069 63.789970 0.000000e+00
cut(age, 4)(33.5,49]   24.053491   1.829431 13.148074 1.982315e-38
cut(age, 4)(49,64.5]   23.664559   2.067958 11.443444 1.040750e-29
cut(age, 4)(64.5,80.1]  7.640592   4.987424  1.531972 1.256350e-01
```

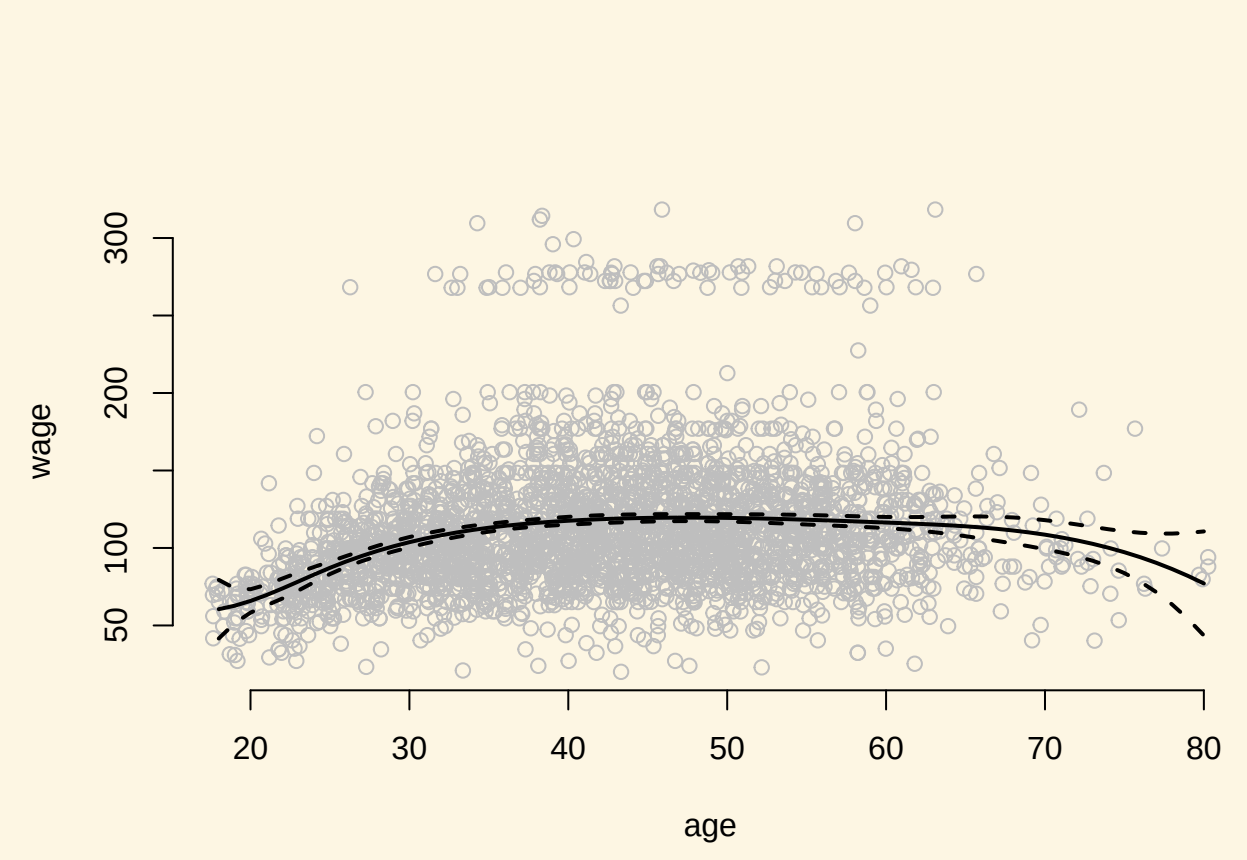
6.1.3 Splines

We use the `splines` package to fit splines.

```
library(splines)
```

We first use `bs()` to generate a basis matrix for a polynomial spline and fit a model with knots at age 25, 40 and 60. `bs` will by default fit a cubic spline with the specified number of knots. To deviate from a cubic spline, change the argument `degree` to some other value.

```
fit <- lm(wage ~ bs(age, knots = c(25, 40, 60)), data = Wage)
pred <- predict(fit, newdata = list(age = age.grid), se = TRUE)
par( mfrow = c(1,1))
plot(jitter(age,2), wage, col = "gray", xlab = "age", bty = "n")
lines(age.grid, pred$fit, lwd = 2)
lines(age.grid, pred$fit + 2 * pred$se, lty = "dashed", lwd = 2)
lines(age.grid, pred$fit - 2 * pred$se, lty = "dashed", lwd = 2)
```



7 Tree Based Models

7.1 Seminar

Tree models are non-parametric models. Depending on the data generation process, these models can be better predictive models than generalized linear models even with regularization. We will start with the highly variable simple tree, followed by pruning, the random forest model and boosting machines.

We load post-election survey data from the 2004 British Election Survey. The data is available here.

```
# clear workspace
rm(list=ls())

# needed because .dta is a foreign file format (STATA format)
library(readstata13)
bes <- read.dta13("bes.dta")

# drop missing values
bes <- na.omit(bes)

# drop id variable
bes$cs_id <- NULL
```

We clean the `in_school` variable which should be binary indicating whether a respondent is attending school or not. However, we estimated missing values (which is a superior treatment of missing values than list-wise deletion) and forgot classify those predictions into 0s and 1s.

```
# clean in_school
table(bes$in_school)
```

```
 -0.405100243979883  -0.286622836951644 -0.0932005119161492
                  1                   1                   1
  -0.08278915151733                   0  0.0403350016659423
                  1                4120                   1
  0.123419680101826   0.247478125358543                   1
                  1                   1                  34
```

We use the `ifelse()` function to classify into 0 and 1.

```
bes$in_school <- ifelse (bes$in_school < 0.5, 0, bes$in_school)
table(bes$in_school)
```

```
   0    1
4127   34
```

Next, we declare the categorical variables in the dataset to be factors en bulk.

```
# data manipulation
categcorical <- c("Turnout", "Vote2001", "Gender", "PartyID", "Telephone", "edu15",
                  "edu16", "edu17", "edu18", "edu19plus", "in_school", "in_uni")
# declare factor variables
bes[, categcorical] <- lapply(bes[, categcorical], factor)
```

7.1.1 Classification Trees

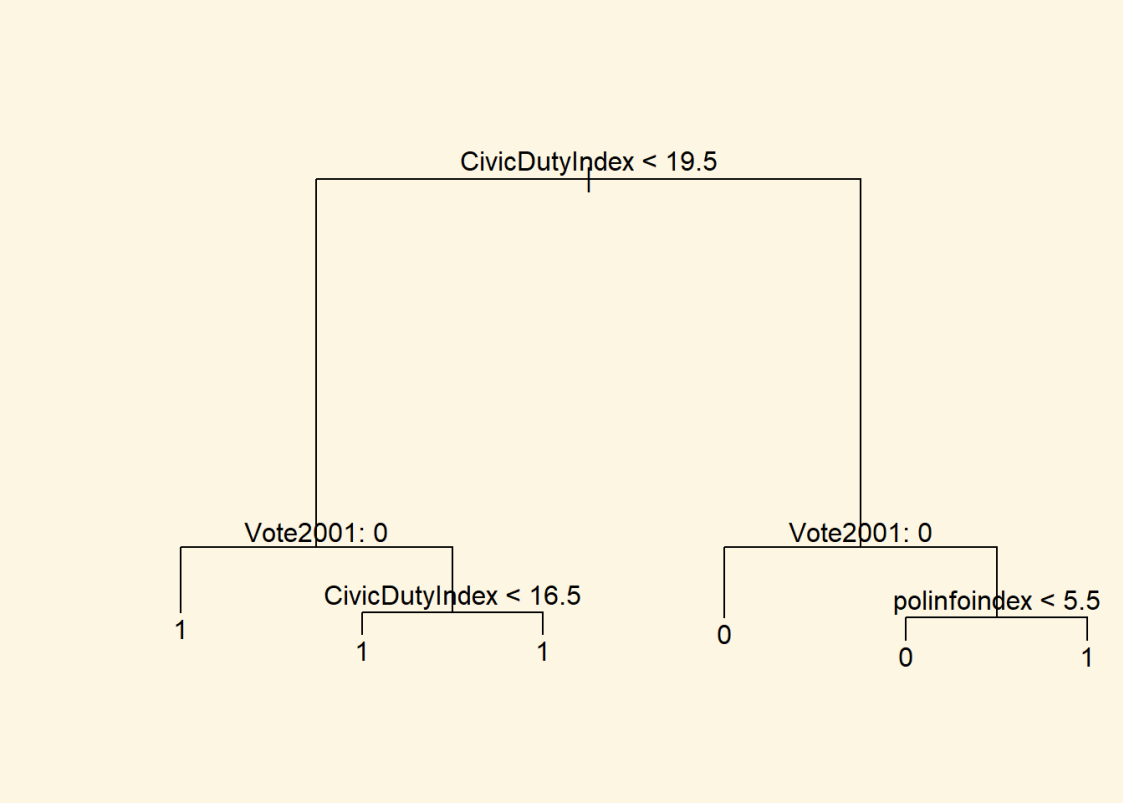
We use trees to classify respondents into voters and non-voters. Here we need the `tree` package which we have to install if is not already `install.packages("tree")`.

```
library(tree)

# build classification tree (- in formula language means except)
t1 <- tree( Turnout ~ . -CivicDutyScores, data = bes)
summary(t1)
```

```
Classification tree:
tree(formula = Turnout ~ . - CivicDutyScores, data = bes)
Variables actually used in tree construction:
[1] "CivicDutyIndex" "Vote2001"       "polinfoindex"
Number of terminal nodes:  6
Residual mean deviance:  0.8434 = 3504 / 4155
Misclassification error rate: 0.1769 = 736 / 4161
```

We can plot the tree using the standard plot function. On every split a condition is printed. The observations in the left branch are those for which the condition is true and the ones on the right are those for which the condition is false.

```
# plot tree object
plot(t1)
text(t1, pretty = 0)
```



We can also examine the splits as text.

```
# examin the tree object
t1
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node
```

```
 1) root 4161 4763.0 1 ( 0.25931 0.74069 )
   2) CivicDutyIndex < 19.5 3066 2446.0 1 ( 0.13666 0.86334 )
     4) Vote2001: 0 243  333.4 1 ( 0.44033 0.55967 ) *
     5) Vote2001: 1 2823 1963.0 1 ( 0.11052 0.88948 )
      10) CivicDutyIndex < 16.5 1748  950.8 1 ( 0.07723 0.92277 ) *
      11) CivicDutyIndex > 16.5 1075  961.7 1 ( 0.16465 0.83535 ) *
   3) CivicDutyIndex > 19.5 1095 1471.0 0 ( 0.60274 0.39726 )
     6) Vote2001: 0 429  391.4 0 ( 0.82984 0.17016 ) *
     7) Vote2001: 1 666  918.2 1 ( 0.45646 0.54354 )
      14) polinfoindex < 5.5 356  483.4 0 ( 0.58427 0.41573 ) *
      15) polinfoindex > 5.5 310  383.7 1 ( 0.30968 0.69032 ) *
```

Now we use the validation set approach for classification. We split our data and re-grow the tree on the training data.

```
# initialize random number generator
set.seed(2)

# training/test split
train <- sample(nrow(bes), size = as.integer(nrow(bes)*.66))
bes.test <- bes[ -train, ]
turnout.test <- ifelse( bes$Turnout[-train] == "1", yes = 1, no = 0)

# grow tree on training data
t2 <- tree( Turnout ~ . , data = bes, subset = train)
```

We predict outcomes using the `predict()` function.

```
# predict outcomes
t2.pred <- predict(t2, newdata = bes.test, type = "class")

# confusion matrix
table(prediction = t2.pred, truth = turnout.test)

# percent correctly classified
mean( t2.pred == turnout.test )
```

```
          truth
prediction   0   1
         0 179  72
         1 186 978
```

We correctly classify 82% of the observations. In classification models, the Brier Score is often used as as measure of model quality. We estimate it as the average of the squared differences between predicted probabilities and true outcomes. It is, thus, similar to the MSE.

```
# using the predict function to predict outcomes from tree
t2.pred <- predict(t2, newdata = bes.test, type = "vector")
head(t2.pred)
```

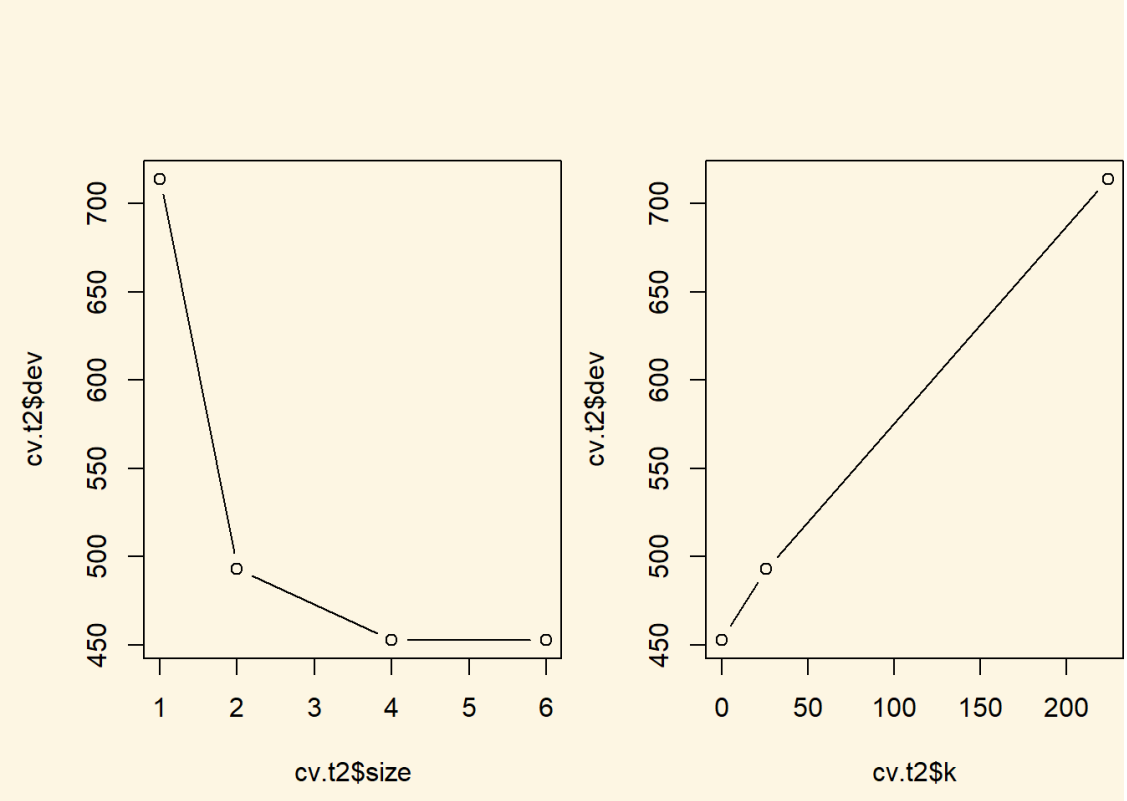
```
            0         1
1  0.39516129 0.6048387
5  0.04152249 0.9584775
6  0.13796477 0.8620352
7  0.39516129 0.6048387
9  0.04152249 0.9584775
14 0.13796477 0.8620352
```

Next we estimate the Brier Score.

```
# the second column of t2.pred is the probabilities that the outcomes is equal to 1
t2.pred <- t2.pred[,2]

# brier score
mse.tree <- mean( (t2.pred - turnout.test)^2 )
```

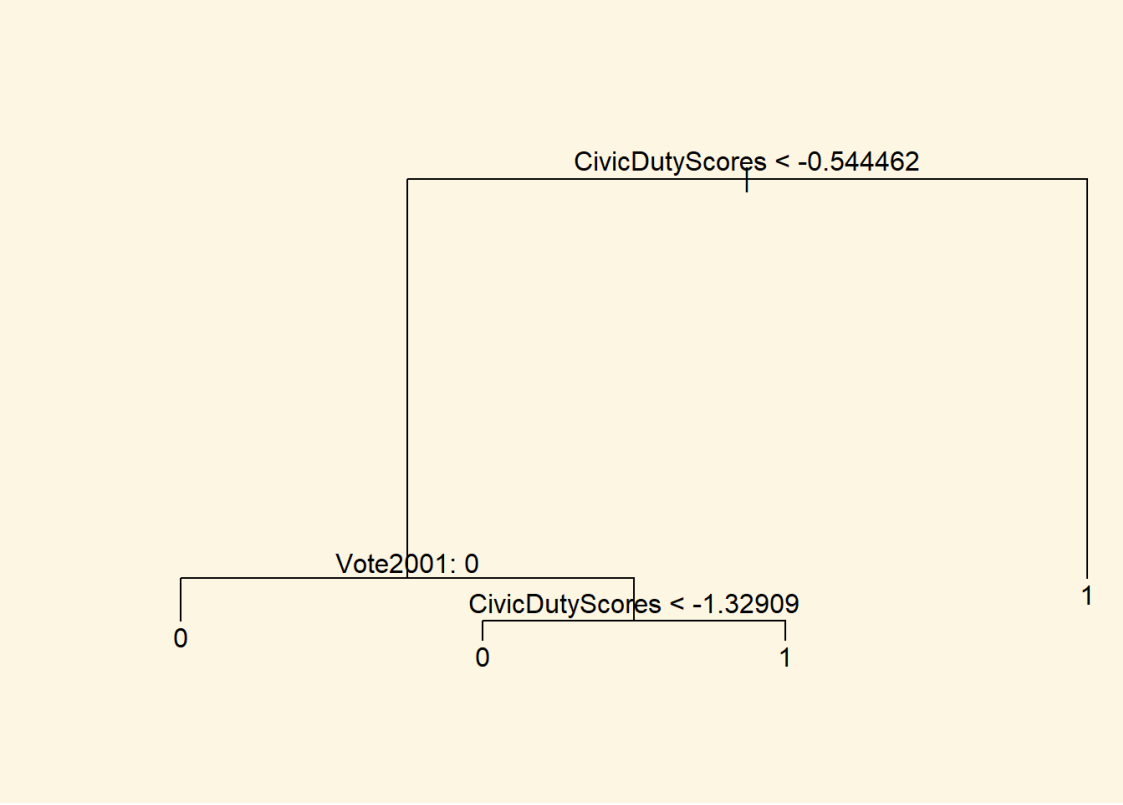
We turn to cost-complexity pruning to see if we can simplify the tree and thus decrease variance without increasing bias. We use k-fold cross-validation to determine the best size of the tree.

```
set.seed(3)
cv.t2 <- cv.tree(t2, FUN = prune.misclass)

# illustrate
par(mfrow = c(1, 2))
plot(cv.t2$size, cv.t2$dev, type = "b")
plot(cv.t2$k, cv.t2$dev, type = "b")
```



We can prune the tree to four terminal nodes.

```
# prune the tree (pick the smallest tree that does not substiantially increase error)
prune.t2 <- prune.misclass(t2, best = 4)
par(mfrow = c(1,1))
plot(prune.t2)
text(prune.t2, pretty = 0)
```



We then predict outcomes.

```
# predict outcomes
t2.pred <- predict(prune.t2, newdata = bes.test, type = "class")

# did we loose predictive power?
mean( t2.pred == turnout.test )
```

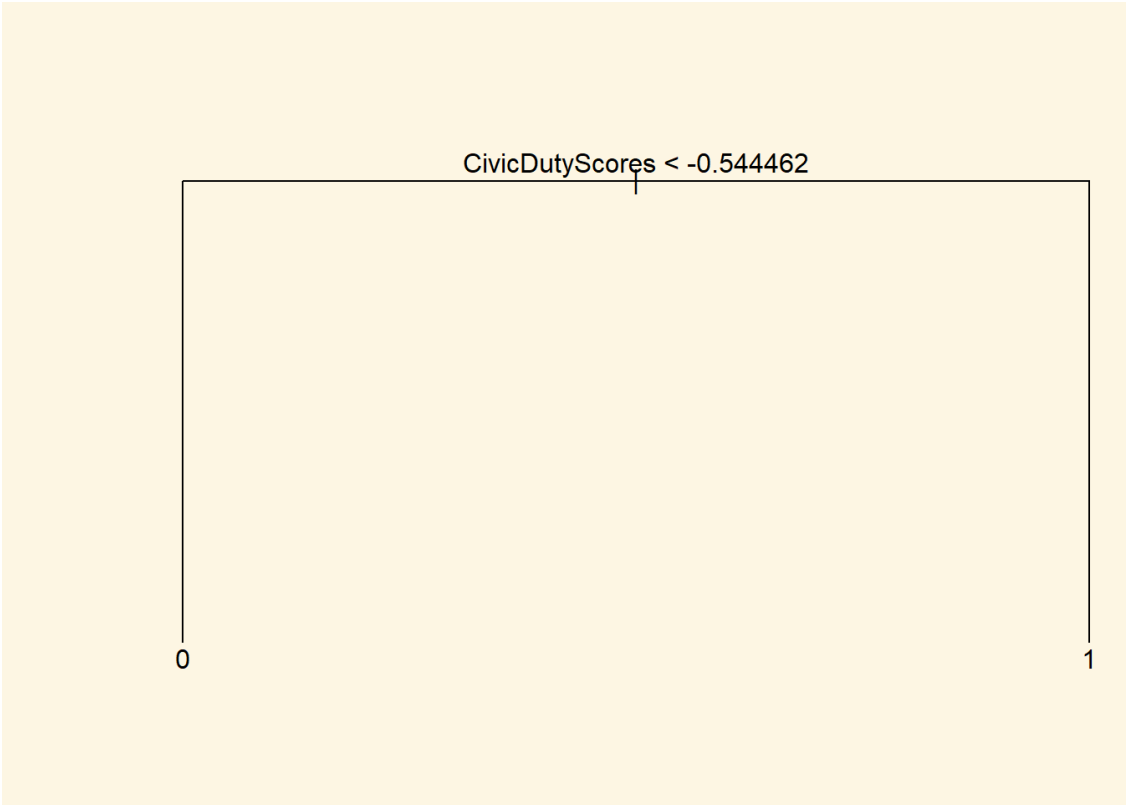
```
[1] 0.8176678
```

Let's estimate the Brier Score

```
# Brier score
t2.pred <- predict(t2, newdata = bes.test, type = "vector")[,2]
mse.pruned <- mean( (t2.pred - turnout.test)^2 )
```

We still correctly classify a similar share of observations and the brier score remained stable. In the previous plots, we saw that we should do worse if we prune back the tree to have less than 4 terminal nodes. We examine what happens if we overdo it.

```
# using "wrong" value for pruning (where the error rate does increase)
prune.t2 <- prune.misclass(t2, best = 2)
plot(prune.t2, bty = "n")
text(prune.t2, pretty = 0)
```



We now predict outcomes based on the tree that is too small.

```
t2.pred <- predict(prune.t2, newdata = bes.test, type = "class")

# our predictive power decreased
mean( t2.pred == turnout.test )
```

```
[1] 0.8007067
```

Let's estimate the Brier Score.

```
# brier score
t2.pred <- predict(prune.t2, newdata = bes.test, type = "vector")[,2]
mse.pruned2 <- mean( (t2.pred - turnout.test)^2 )
```

We see that our test error increases.

7.1.2 Regression Trees

We predict the continuous variable `Income`. The plot of the regression tree is similar. However, in the terminal nodes the mean values of the dependent variable for that group are displayed rather than the class labels.

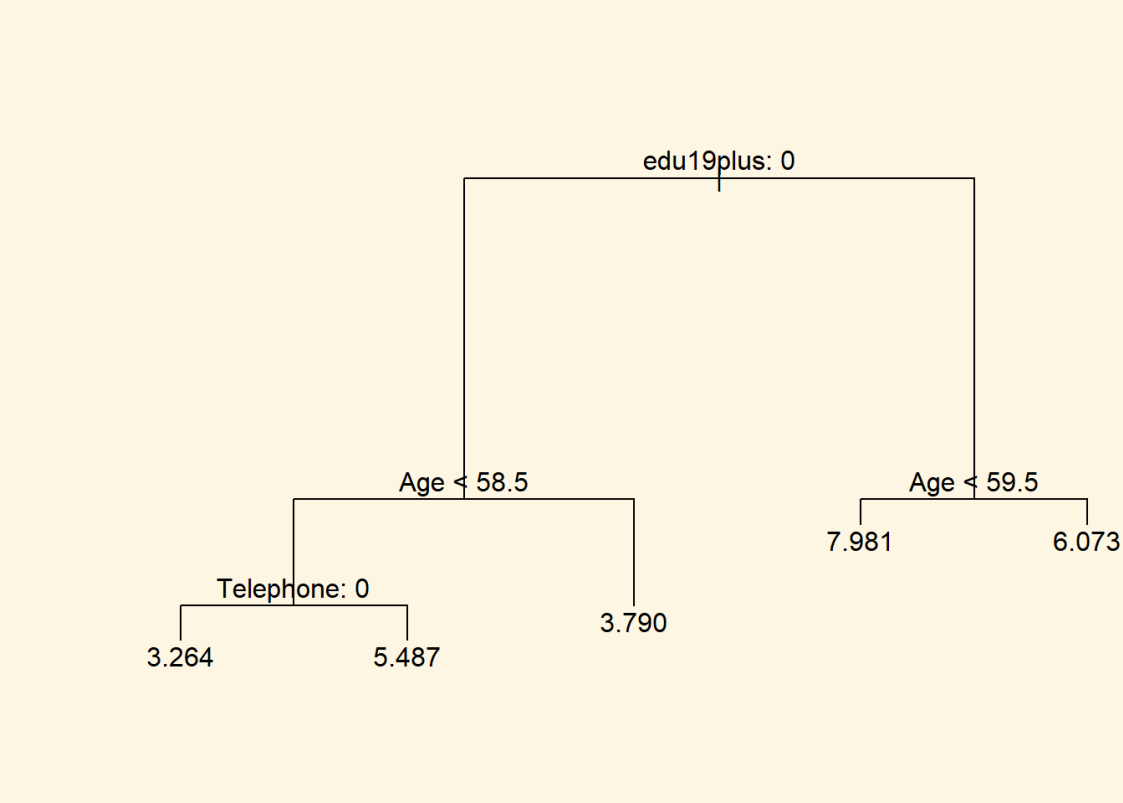
```
# grow a regression tree
set.seed(123)
reg.t1 <- tree(Income ~ ., data = bes, subset = train)
summary(reg.t1)
```

```
Regression tree:
tree(formula = Income ~ ., data = bes, subset = train)
Variables actually used in tree construction:
[1] "edu19plus" "Age"       "Telephone"
```

```
Number of terminal nodes:  5
Residual mean deviance:  3.849 = 10550 / 2741
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-6.98100 -1.48700  0.01935  0.00000  1.21000  9.21000
```

Let's plot the tree.

```
# plot regression tree
plot(reg.t1)
text(reg.t1, pretty = 0)
```



We can also examine the same output as text.

```
# examin the tree object
t1
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 4161 4763.0 1 ( 0.25931 0.74069 )
   2) CivicDutyIndex < 19.5 3066 2446.0 1 ( 0.13666 0.86334 )
     4) Vote2001: 0 243  333.4 1 ( 0.44033 0.55967 ) *
     5) Vote2001: 1 2823 1963.0 1 ( 0.11052 0.88948 )
      10) CivicDutyIndex < 16.5 1748  950.8 1 ( 0.07723 0.92277 ) *
      11) CivicDutyIndex > 16.5 1075  961.7 1 ( 0.16465 0.83535 ) *
   3) CivicDutyIndex > 19.5 1095 1471.0 0 ( 0.60274 0.39726 )
     6) Vote2001: 0 429  391.4 0 ( 0.82984 0.17016 ) *
     7) Vote2001: 1 666  918.2 1 ( 0.45646 0.54354 )
      14) polinfoindex < 5.5 356  483.4 0 ( 0.58427 0.41573 ) *
```

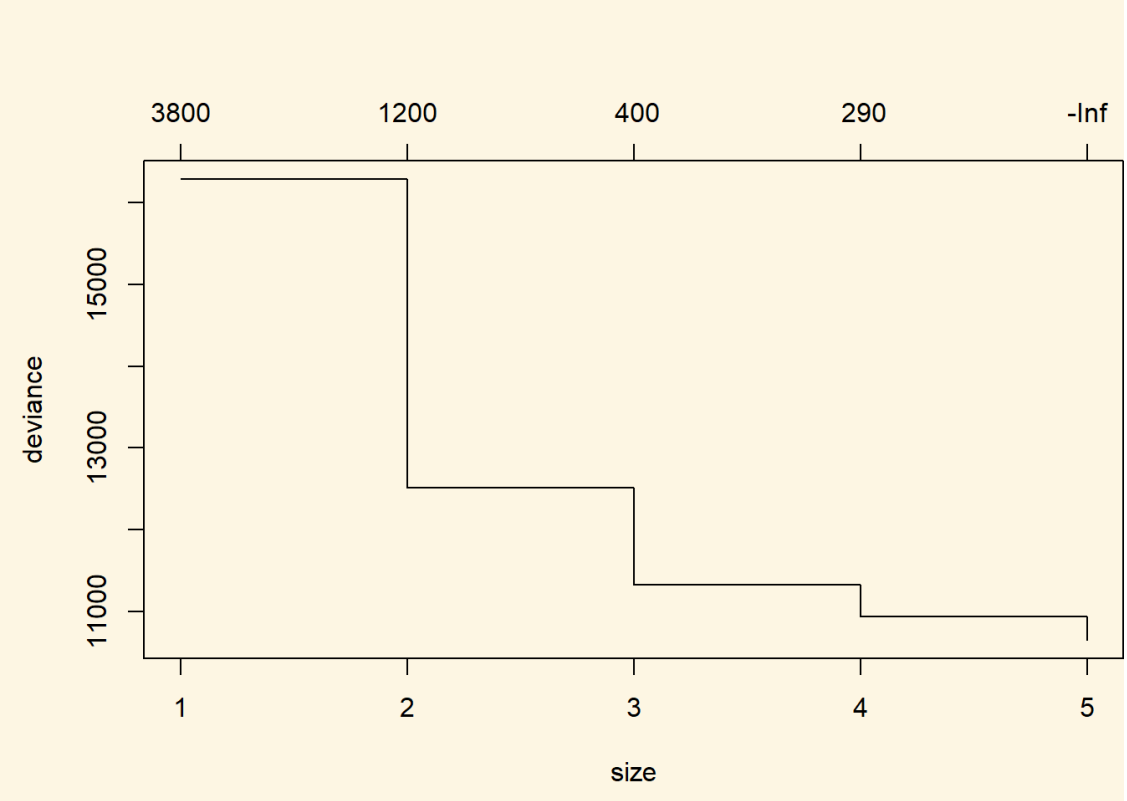
```
      15) polinfoindex > 5.5 310  383.7 1 ( 0.30968 0.69032 ) *
```

We estimate test error of our tree.

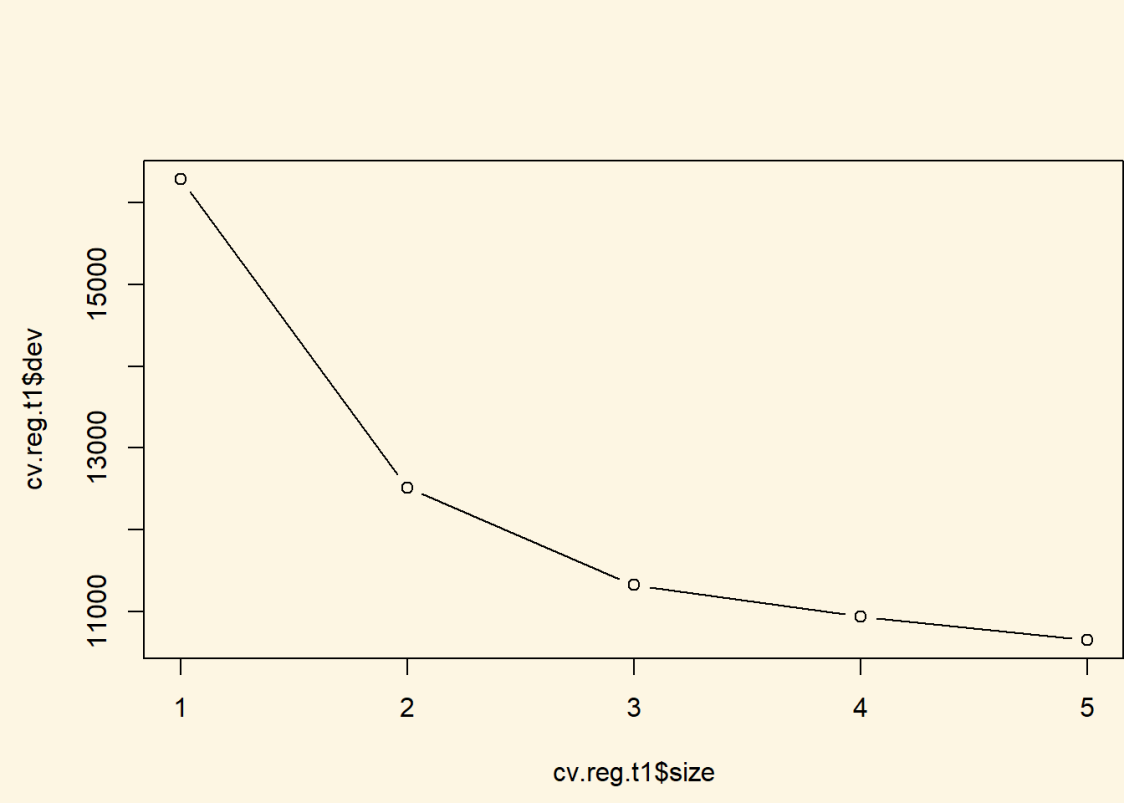
```
# MSE
mse.tree <- mean( (bes.test$Income - predict(reg.t1, newdata = bes.test))^2, na.rm = TRUE)
```

We apply pruning again to get a smaller more interpretable tree.

```
# cross-validation (to determine cutback size for pruning)
cv.reg.t1 <- cv.tree(reg.t1)
plot(cv.reg.t1)
```

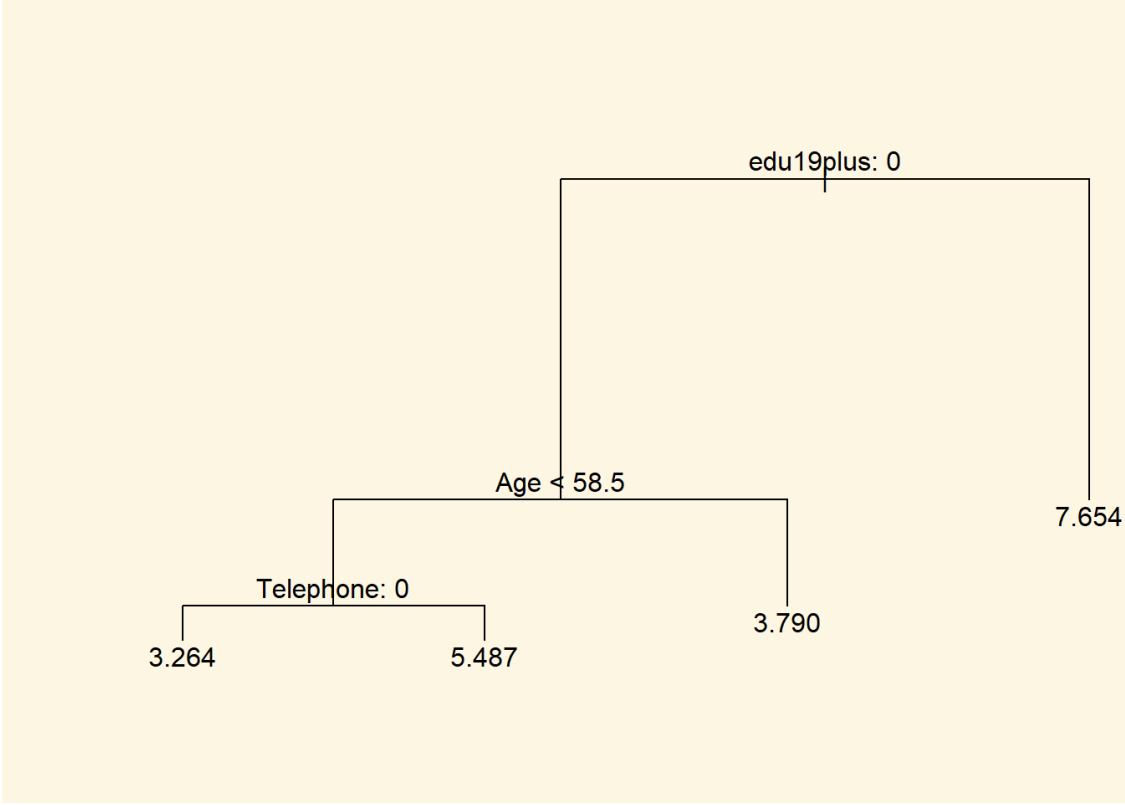



```
plot(cv.reg.t1$size, cv.reg.t1$dev, type = "b")
```



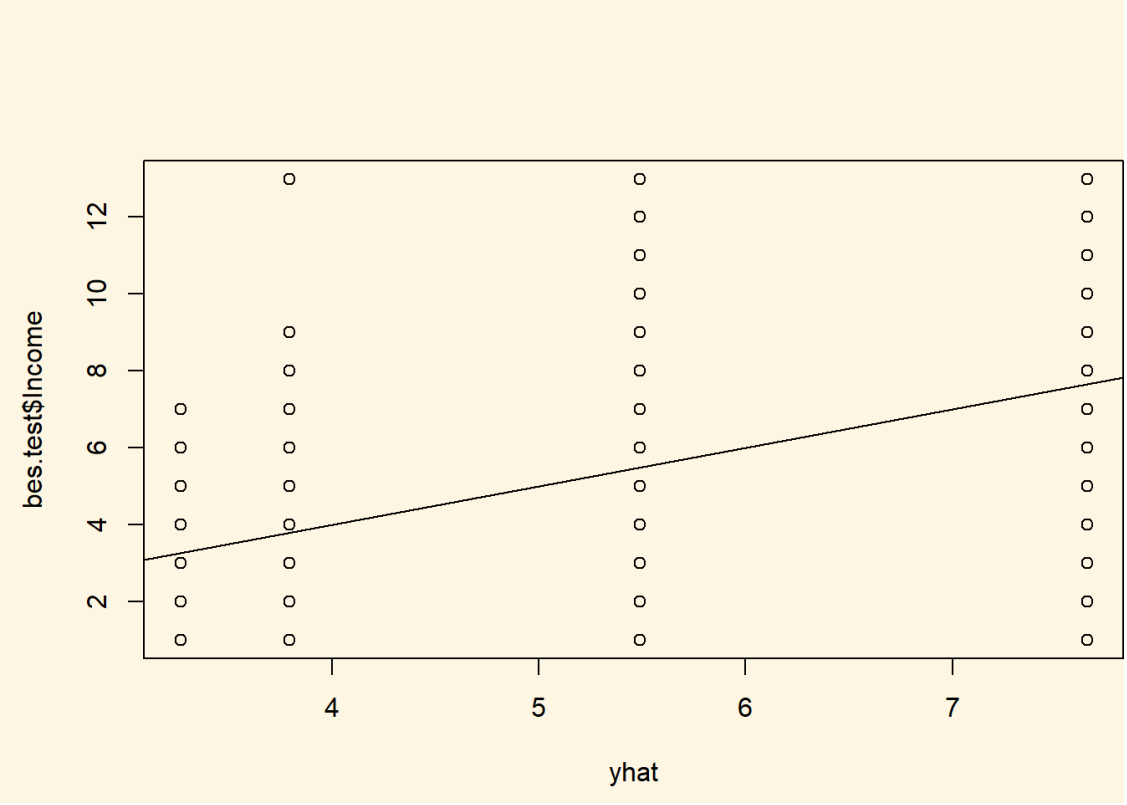
This is time we will increase error by pruning the tree. We choose four as a smaller tree size that does not increase RSS by much.

```
# pruning
prune.reg.t1 <- prune.tree(reg.t1, best = 4)
plot(prune.reg.t1)
text(prune.reg.t1, pretty = 0)
```



We can predict the outcome based on our pruned back tree. We will predict four values because we have four terminal nodes. We can illustrate the groups and their variance and estimate the MSE of our prediction.

```
# predict outcomes
yhat <- predict(prune.reg.t1, newdata = bes.test)
plot(yhat, bes.test$Income)
abline(0, 1)
```



We estimate the Brier Score (prediction error).

```
# MSE
mse.pruned <- mean((yhat - bes.test$Income)^2)
```

7.1.2.1 Bagging and Random Forests

We now apply bagging and random forests to improve our prediction. Bagging is the idea that the high variance of a single bushy tree can be reduced by bootstrapping samples and averaging over trees that were grown on the samples.

Note: Bagging gets an estimate of the test error for free as it always leaves out some observations when a tree is fit. The reported out-of-bag MSE is thus an estimate of test error. We also estimate test error separately on a test set. This is one particular test set, so the test error may vary.

In our run below the OOB MSE may be a better estimate of test error. It is reported to be lower than our test error estimate. We need to install the `randomForest` package like so: `install.packages("randomForest")`.

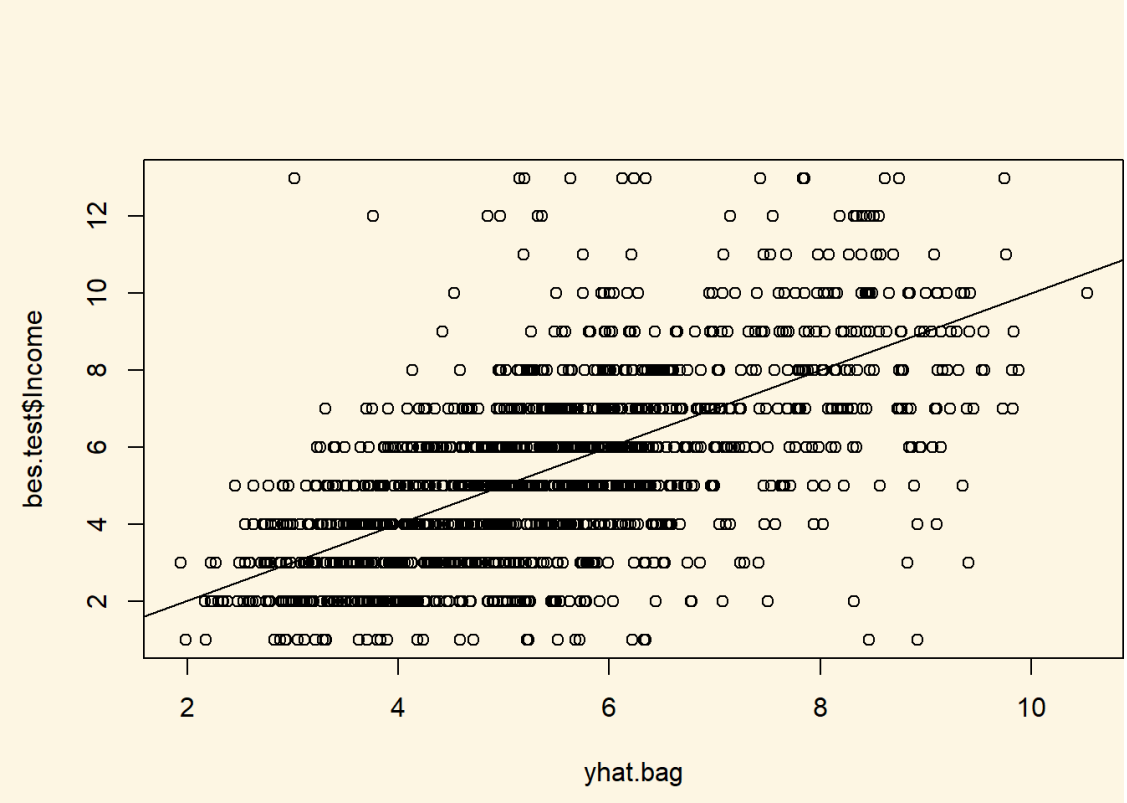
```
set.seed(123)
library(randomForest)

# estiamte random forrests model (this may take a moment)
bag1 <- randomForest(Income ~ . , mtry = 19, data = bes, subset = train, importance = TRUE)
bag1
```

```
Call:
 randomForest(formula = Income ~ ., data = bes, mtry = 19, importance = TRUE,      subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 19
```

```
          Mean of squared residuals: 3.680305
                    % Var explained: 37.89
```

We can use the predict function to predict outcomes from our random forests object.

```
# predict outcome, illustrate, MSE
yhat.bag <- predict(bag1, newdata = bes.test)
plot(yhat.bag, bes.test$Income)
abline(0, 1) # line of 1:1 perfect prediction
```



We estimate the MSE in the validation set

```
mse.bag <- mean( (yhat.bag - bes.test$Income)^2 )

# reduction of error
(mse.bag - mse.tree) / mse.tree
```

```
[1] -0.05762335
```

We reduce the error rate by 5.76% by using bagging. We examine what happens when we reduce the number of trees we grow. The default is 500.

```
# dcrease the number of trees (defaults to 500)
bag2 <- randomForest(Income ~ ., mtry = 19, data = bes, subset = train, ntree = 25, importance = TRUE)

# predict outcome
yhat.bag2 <- predict(bag2, newdata = bes.test)
mse.bag2 <- ( (yhat.bag2 - bes.test$Income)^2 )
```

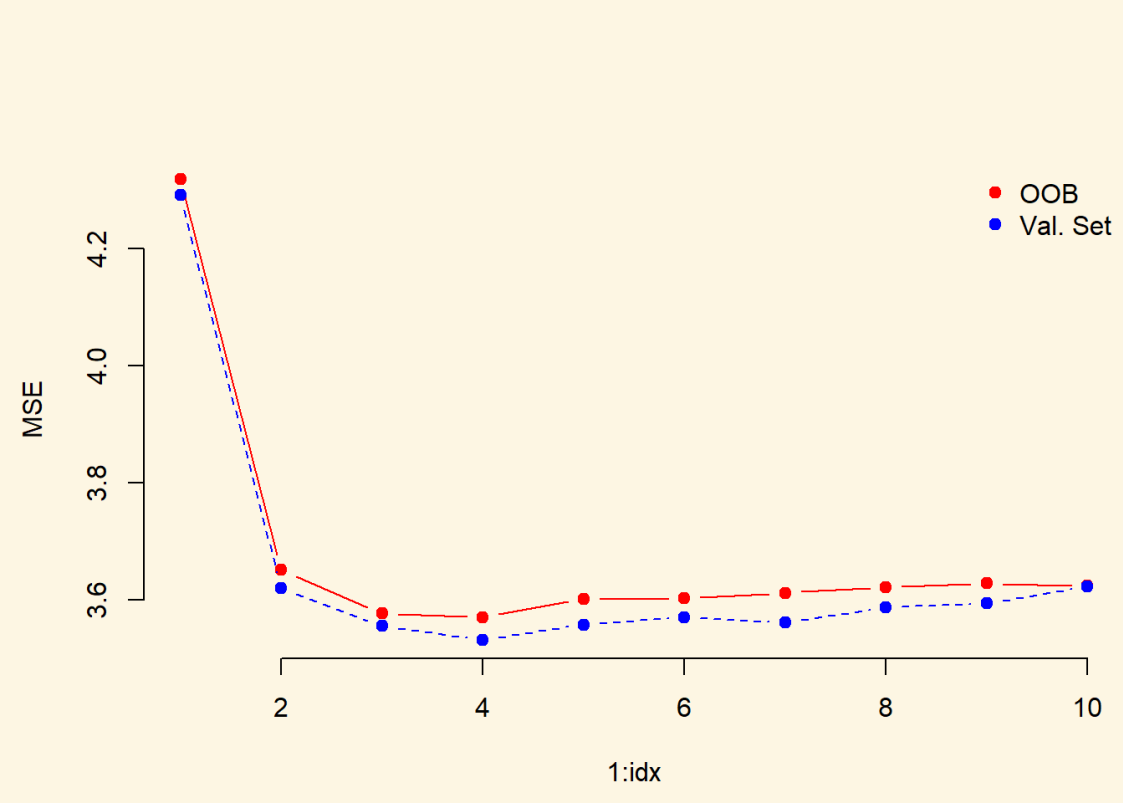
The result is that our rate increases substantially again.

We now apply random forest. The trick is to de-corelate the trees by randomly considering only a subset of variables at every split. We thereby reduce variance further. The number of variables argument `mtry` is a tuning parameter.

```
# Random Forest: not trying all vars at each split decorrelates the trees
set.seed(123)

# we try to find the optimal tuning parameter for the number of variables to use at each split
oob.error <- NA
val.set.error <- NA
for ( idx in 1:10){
  rf1 <- randomForest(Income ~ ., mtry = idx, data = bes, subset = train, importance = TRUE)
  # record out of bag error
  oob.error[idx] <- rf1$mse[length(rf1$mse)]
  cat(paste("\n", "Use ", idx, " variables at each split", sep=""))
  # record validation set error
  val.set.error[idx] <- mean( (predict(rf1, newdata = bes.test) - bes.test$Income)^2 )
}
```

```
Use 1 variables at each split
Use 2 variables at each split
Use 3 variables at each split
Use 4 variables at each split
Use 5 variables at each split
Use 6 variables at each split
Use 7 variables at each split
Use 8 variables at each split
Use 9 variables at each split
Use 10 variables at each split
```

```
# check optimal values for mtry
matplot( 1:idx, cbind(oob.error, val.set.error), pch = 19, col = c("red", "blue"),
         type = "b", ylab = "MSE", frame.plot = FALSE)
legend("topright", legend = c("OOB", "Val. Set"), pch = 19, col = c("red", "blue"),
       bty = "n")
```



We use 3 as the optimal value for `mtry`. In cases where it is hard to decide, it's a good idea to choose the less complex model.

```
rf <- randomForest(Income ~ ., mtry = 4, data = bes, subset = train, importance = TRUE)

# predict outcomes
yhat.rf <- predict(rf, newdata = bes.test)
mse.rf <- mean( (yhat.rf - bes.test$Income)^2 )

# on previous random forests model
(mse.rf - mse.bag) / mse.bag
```

```
[1] -0.03454119
```

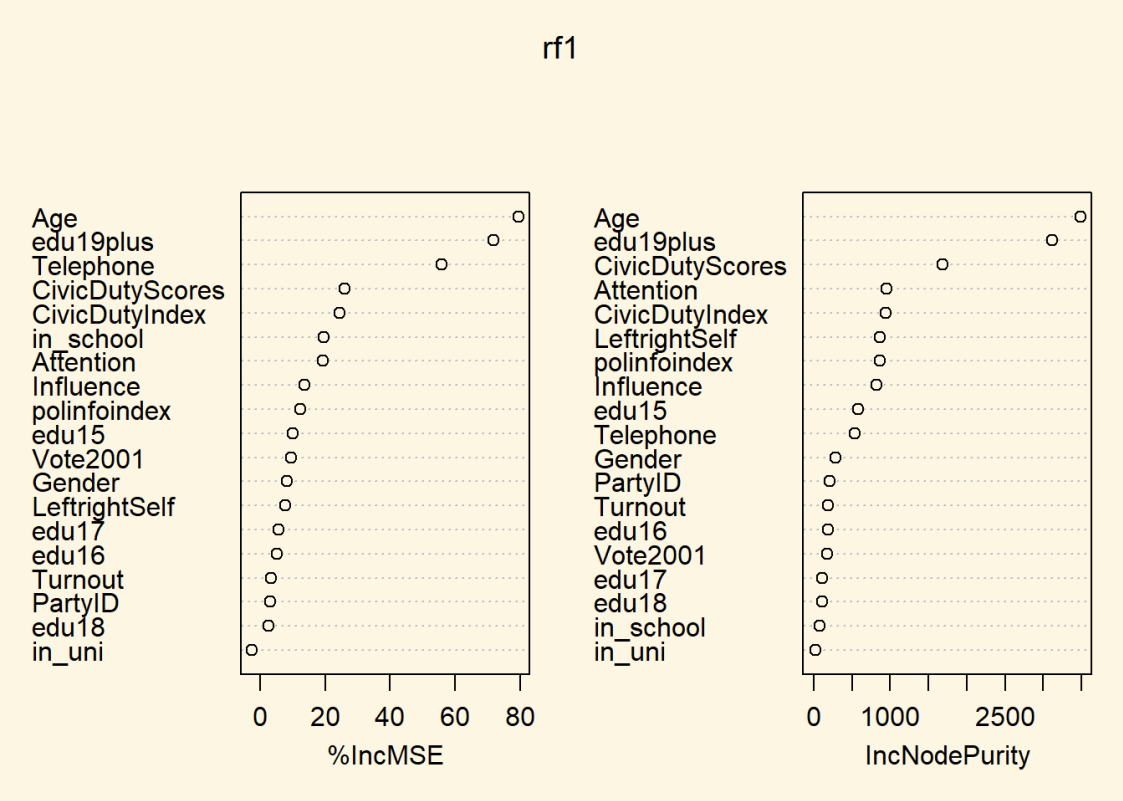
We reduced the error rate by another 3.45% by de-correlating the trees. We can examine variable importance as well. Variable reduction is obtained as the average that a predictor reduces error at splits within a tree where it was used and averaged again over all trees. Similarly, node purity is based on the gini index of how heterogeneous a group becomes due to a split.

```
par(mfrow = c(1,1))
# which varaibles help explain outcome
importance(rf1)
```

```
                %IncMSE IncNodePurity
Turnout        3.485190     186.58860
Vote2001       9.654237     180.45489
Age           79.593695    3492.19404
Gender         8.343223     281.50559
PartyID        3.053795     207.56146
Influence     13.608173     823.65790
```

```
Attention        19.417204     955.05716
Telephone        55.951595     534.66524
LeftrightSelf     7.783832     869.14777
CivicDutyIndex   24.542362     942.35134
polinfoindex     12.441130     867.21040
edu15            10.163296     580.35854
edu16             5.283346     185.01184
edu17             5.651777     112.05356
edu18             2.655123     110.94208
edu19plus        71.813297    3112.61687
in_school        19.692186      80.59582
in_uni           -2.576748      22.41477
CivicDutyScores  25.916963    1684.33658
```

```
# importance plot
varImpPlot(rf1)
```



7.1.2.2 Boosting

The general idea of boosting is that a tree is fit to predict outcome. The second tree is then fit on the residual of the first and so with all following trees. Each additional tree is discounted by a learning rate, so that each tree does not contribute much but slowly the ensemble becomes more predictive.

Install the `gbm` package like so `install.packages("gbm")`.

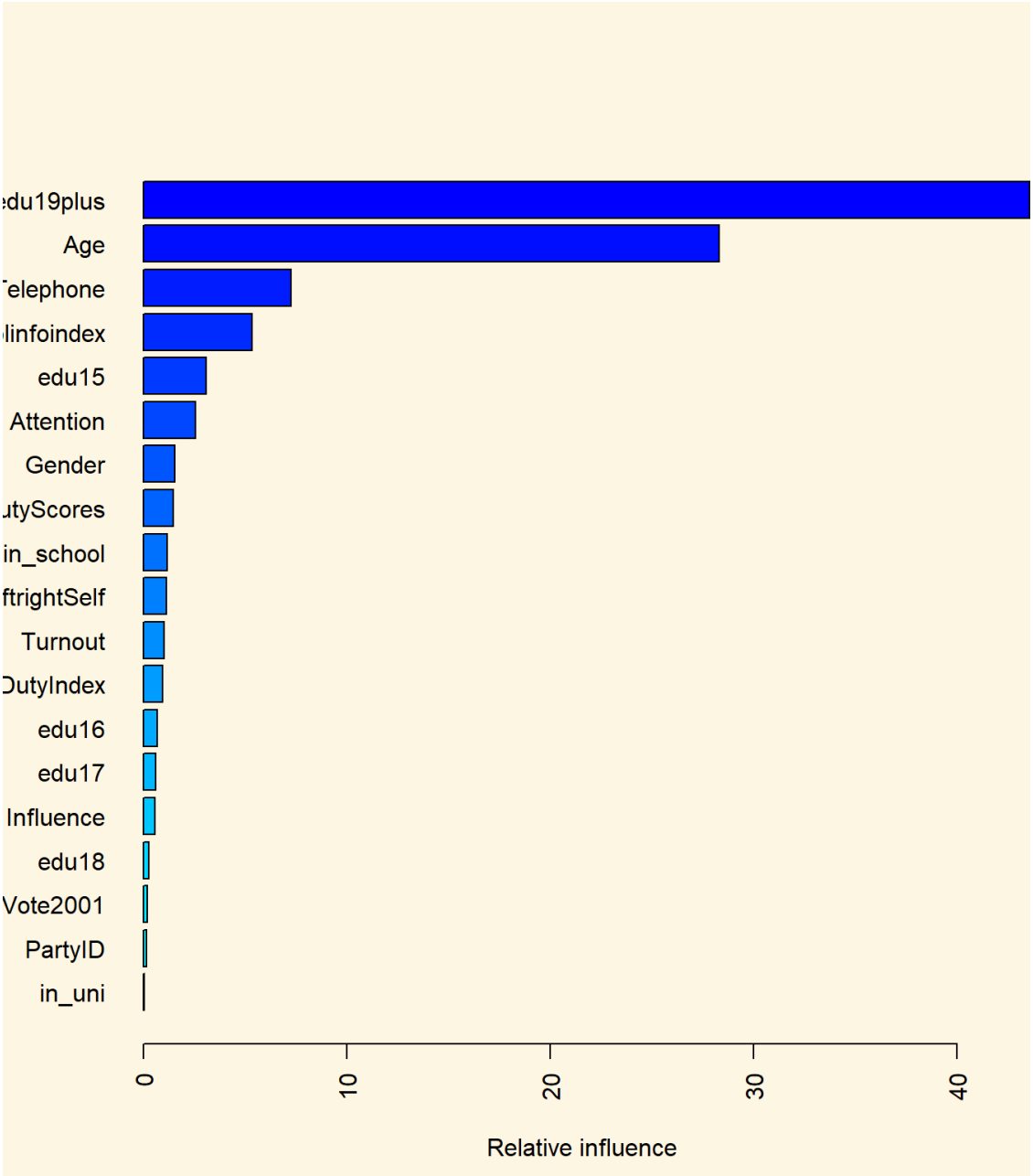
```
library(gbm)
set.seed(1234)
```

We run gradient boosting. The tuning parameters are the tree size. Tree size is directly related to the second tuning parameter: the learning rate. When the learning rate is smaller, we need more trees. The third

tuning parameter interaction.depth determines how bushy the tree is. Common choices are 1, 2, 4, 8. When interaction depth is 1, each tree is a stump. If we increase to two we can get bi-variate interactions with 2 splits and so. A final parameter that is related to the complexity of the tree could be minimum number of observations in the terminal node which defaults to 10.

Notice that we just set hyper-parameters. We might achieve better predictions by training the boosting model properly (however, this would take very long).

```
# gradient boosting
gb1 <- gbm(Income ~ ., data = bes[train, ],
           distribution = "gaussian",
           n.trees = 5000,
           interaction.depth = 4,
           shrinkage = 0.001)

summary(gb1, order = TRUE, las = 2)
```



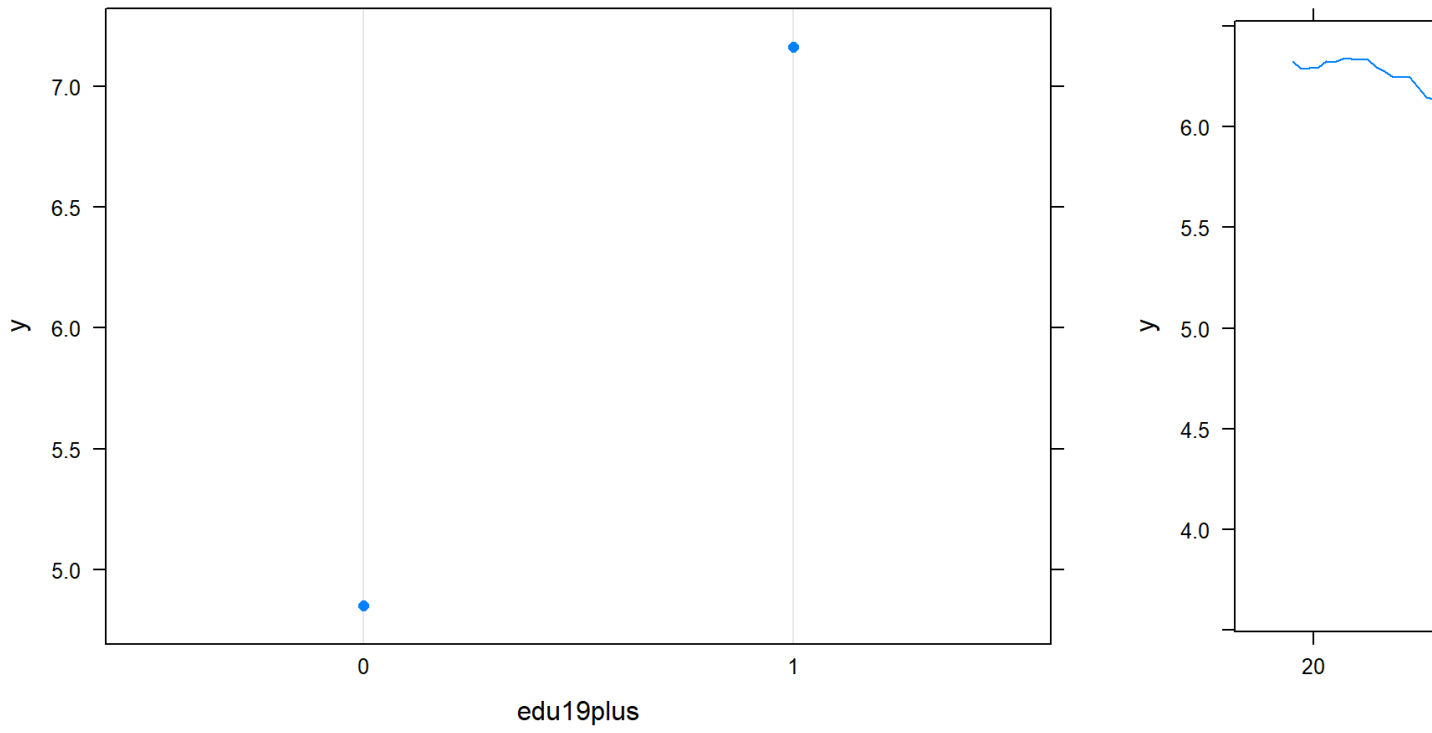
	var	rel.inf
edu19plus	edu19plus	43.60824543
Age	Age	28.30747893
Telephone	Telephone	7.24463534
polinfoindex	polinfoindex	5.33906762
edu15	edu15	3.07150723
Attention	Attention	2.56886790
Gender	Gender	1.56310050
CivicDutyScores	CivicDutyScores	1.48574280
in_school	in_school	1.18768575
LeftrightSelf	LeftrightSelf	1.13485329
Turnout	Turnout	1.03074516
CivicDutyIndex	CivicDutyIndex	0.93773693

```
edu16                  edu16  0.68203332
edu17                  edu17  0.61758676
Influence          Influence  0.57391921
edu18                  edu18  0.25732928
Vote2001            Vote2001  0.19667081
PartyID              PartyID  0.16443143
in_uni                in_uni  0.02836232
```

The variable importance plot gives a good idea about which variables are important predictors. The general weakness of GBM is that the model is somewhat of a black box. However, variables importance gives us some insights into the model. For instance, it seems that high education and age are most predictive. Variables like arty identification or ideology play less of a role in the predictive model. Gender is the sixth strongest predictor.

The importance plot does not inform us about the direction of the relationship. To get insights into such predictors, we can assess partial dependence plots. Let's do so for the high education variable `edu19plus` and for `Age`.

```
# partial dependence plots
plot(gb1, i = "edu19plus")
plot(gb1, i = "Age")
```



We predict the test MSE again and compare to our best model.

```
# predict outcome
yhat.gb <- predict(gb1, newdata = bes.test, n.trees = 5000)
mse.gb <- mean( (yhat.gb - bes.test$Income)^2 )

# reduction in error
(mse.gb - mse.rf) / mse.rf
```

```
[1] -0.03998188
```

We reduce the error rate again quite a bit.

7.1.2.3 Bayesian Additive Regression Trees (BARTs)

BARTs move regression trees into a Bayesian framework were priors are used on several parameters to reduce some of the problems of over-fitting that boosting and random forests are prone to. We will illustrate a small example here. Before you can run this, ***64bit JAVA*** must be installed on your computer. We then need to install `install.packages("rJava")` and then `install.packages("bartMachine")`. Installing Java can be tricky and you may need admin rights on your computer.

There are several tuning parameters for the priors that are set to values that work for most applications. Check the documentation if you want to learn more about these. We set the number of trees to grow, the iterations in the Markow-Chain Monte-Carlo estimations to discard and the draws from the posterior distribution. These parameters should also be tested for instance using cross-validation. The algorithm needs some time to run and therefore we pick out-of-the-box values.

```
options(java.parameters = "-Xmx5g")
library(bartMachine)

# vector of covariate names
Xs <- c("Turnout", "Vote2001", "Age", "Gender", "PartyID", "Influence", "Attention",
        "Telephone", "LeftrightSelf", "CivicDutyScores", "polinfoindex", "edu15",
        "edu16", "edu18", "edu19plus", "in_school", "in_uni")


# run BART
bart1 <- bartMachine(X = bes[train, Xs],
                     y = bes$Income[train],
                     num_trees = 500,
                     num_burn_in = 200,
                     num_iterations_after_burn_in = 1000,
                     seed = 123)
```

```
bartMachine initializing with 500 trees...
bartMachine vars checked...
bartMachine java init...
bartMachine factors created...
bartMachine before preprocess...
bartMachine after preprocess... 29 total features...
bartMachine sigsq estimated...
bartMachine training data finalized...
Now building bartMachine for regression ...Covariate importance prior ON.
evaluating in sample data...done
```

```
# predict outcomes on the test set
pred <- predict(object = bart1, new_data = bes[-train, Xs])

# Brier Score
mse.bart <- mean((bes.test$Income - pred)^2)
```

Let's compare our final BART prediction to the reigning champion gradient boosting.

```
# reduction in error
(mse.bart - mse.gb) / mse.gb
```

```
[1] -0.01670494
```

We have reduced the error by another 1.67%. Not bad... However, keep in mind that we are usig the validation set approach here. A better evaluation would be based on cross-validation or a truly new dataset.

8 Simulation and Monte Carlos

8.1 Seminar

In this exercise, we introduce a simulation approach to quantifying uncertainty and Monte Carlo simulation.

8.1.1 Monte Carlo simulation

In Monte Carlo simulation we create a fake data set were we define a causal model of the outcome directly. That way, we know what the true outcome has to be. We can then keep everything constant but make on change to our estimation to assess the effect of that change on our prediction accuracy. We could, for instance, be interested in finding out whether our prediction accuracy suffers when independent variables are highly correlated.

So, we set up an MC analysis to see whether problems of multicolinearity (high correlation between explanatory variables) go away as the sample size increases. The goal is to see how well we are able to retrieve the true value of β_1 for varying strengths of correlation and sample sizes.

The basic setup of the simulation is as follows:

```
# the number of runs (simulations)
sim.n <- 1000

# sequence of low to high correlation
Rho <- seq(from = 0, to = .9,length.out=10)

# vector of sample sizes
sample.N <- c(100, 500, 1000, 2000)
```

We create a container `beta.catcher` that stores coefficient values for different simulations. The container is a 3-dimensional array where the first dimension is the number of simulations, the second is the sample size, the third is correlation.

```
# rows = simulations, columns = sample size, layers = correlation
beta.catcher <- array(NA, c(sim.n, 4, 10))
```

We start simulating. This will take a while because we are iterating through $sim.n * Rho * sample.N$ iterations, i.e. $40,000$ iterations. Each time we regress y on our covariates.

```
# loop over the correlations
for (i in 1:length(Rho)){

  # loop over sample sizes
  for (j in 1:4){

    # loop over the random draws (number of simulations)
    for (k in 1:sim.n){

      # current correlation
      rho <- Rho[i]
      # current sample size
      sample.n <- sample.N[j]
      # variance covariance matrix current corr on off-diagonal
      varL <- matrix(c (1,rho,rho,1) , nrow = 2, ncol = 2)
```

```
      # random draw of covariates as many as sample size
      XX <- MASS::mvrnorm(sample.n, rep(0,2), varL)
      # random noise
      e <- rnorm(sample.n)
      # the true data generation process b1 = 1; b2 = 1
      y <- XX  %*% c(1,1) + e
      # we regress the true y on the covariates and extract beta 1
      beta.catcher[k,j,i] <- coef(lm(y~XX))[2]

      } # end of loop over number of sims
    } # end of loop over sample sizes
  } # end of loop over correlations
```

You can have a look at the container by calling it with `beta.catcher`. We remind ourselves of the dimensions with:

```
dim(beta.catcher)
```

```
[1] 1000    4   10
```

We take the MSE of each coefficient estimate in the container. This is an element-wise operation so the resulting object `error.sq` has the same dimensions as the container.

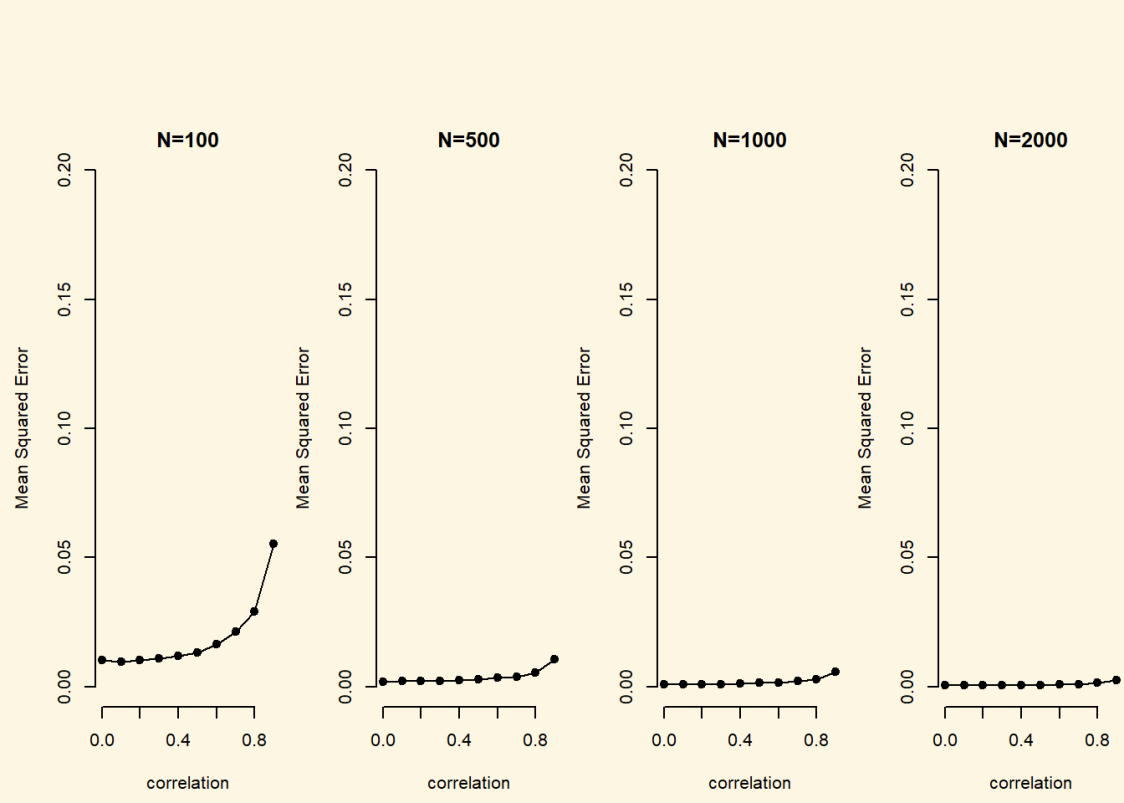
```
error.sq <- (beta.catcher-1)^2
```

We average over the simulations by taking the column means. This returns a matrix where the rows are now the different sample sizes (previously in the columns) and the columns are the correlations (previously in the layers). The `matrix mse.beta`, thus, has 4 rows and 10 columns.

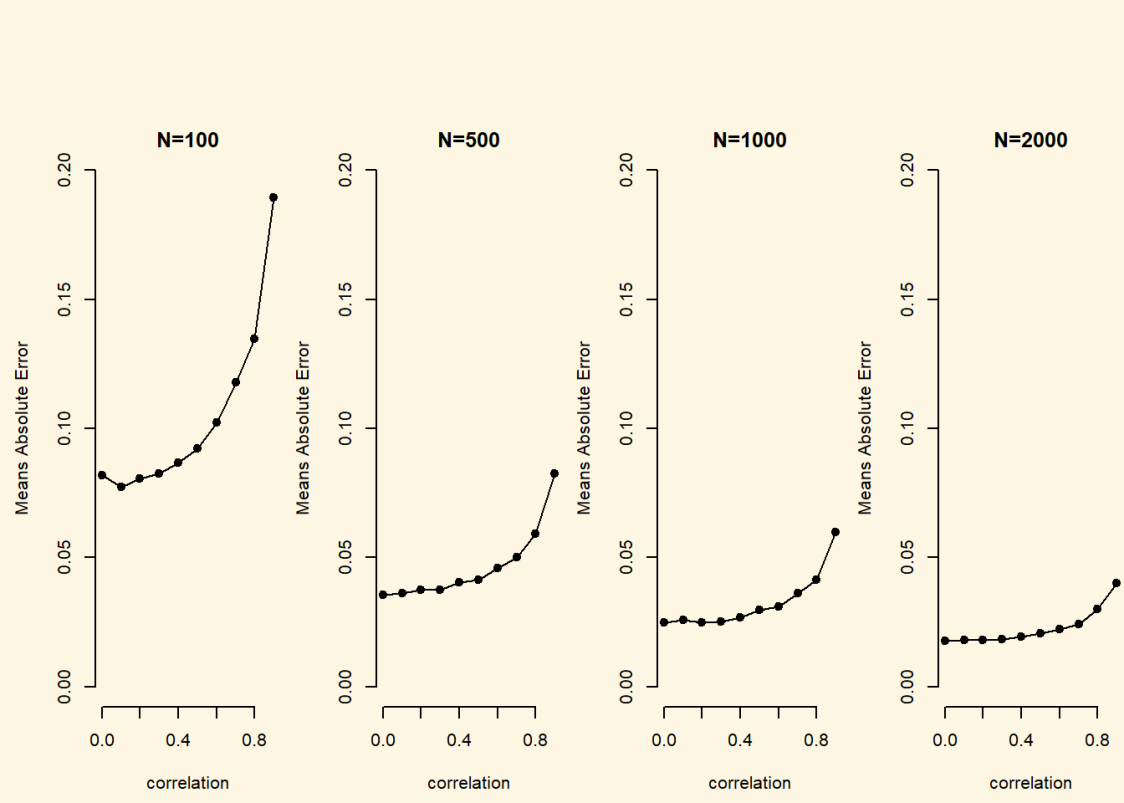
```
mse.beta1 <- colMeans(error.sq)
```

To show the effect of multicolinearity for increasing levels of correlation and for increasing sample sizes, we plot correlation on the x-axis and the MSE on the y-axis in four plots where sample size increases by plot.

```
par(mfrow=c(1,4))
plot(c(0:9)/10,mse.beta1[1,], xlab="correlation", ylab="Mean Squared Error", type="o", pch=19, bty="n",
plot(c(0:9)/10,mse.beta1[2,], xlab="correlation", ylab="Mean Squared Error", type="o", pch=19, bty="n",
plot(c(0:9)/10,mse.beta1[3,], xlab="correlation", ylab="Mean Squared Error", type="o", pch=19, bty="n",
plot(c(0:9)/10,mse.beta1[4,], xlab="correlation", ylab="Mean Squared Error", type="o", pch=19, bty="n",
```




```
error.abs <- abs(beta.catcher-1)
mae.beta1 <- colMeans(error.abs)

par(mfrow=c(1,4))
plot(c(0:9)/10,mae.beta1[1,], xlab="correlation", ylab="Means Absolute Error", type="o", pch=19, bty="n
plot(c(0:9)/10,mae.beta1[2,], xlab="correlation", ylab="Means Absolute Error", type="o", pch=19, bty="n
plot(c(0:9)/10,mae.beta1[3,], xlab="correlation", ylab="Means Absolute Error", type="o", pch=19, bty="n
plot(c(0:9)/10,mae.beta1[4,], xlab="correlation", ylab="Means Absolute Error", type="o", pch=19, bty="n
```



The two take-aways are that (1) with increasing sample size the problem of multicolinearity decreases substantially and (2) bias increases exponentially with increasing levels of correlation.

8.1.2 Simulation approach to uncertainty

Simulation is the Swiss army knife of statistics. Quantifying the uncertainty of an outcome can be tough or even impossible algebraically. Even for the linear model we need to consider the standard errors of all coefficients and their covariance. The formulas can be tedious...

For simulation, the process is always the same regardless of the model.

We start by loading data and fitting a linear model on the unemployment rate.

```
# clear workspace
rm(list=ls())
# load data
df <- read.csv("http://philippbroniecki.github.io/ML2017.io/data/communities.csv")
```

Simulation step 1: Our coefficients each follow a sampling distribution. Jointly, they follow a multivariate distribution which is assumed to be multivariate normal.

To characterize the shape of the distribution we need to know its mean and its variance. The mean is our vector of coefficient point estimates. We extract it using `coef(model_name)`. The variance is the model uncertainty which lives in the variance-covariance matrix. We extract it with `vcov(model_name)`.

As we draw randomly from a distribution we want to set the random number generator with `set.seed()` to make our results replicable and we pick the number of coefficients to draw form the distribution (the number of simulations).

```
# run a model
m1 <- lm(PctUnemployed ~ pctUrban + householdsize + racePctWhite, data = df)
```

```r
# set the random number generator to some value
set.seed(123)

# pick how many coefficients you want to draw from the distribution
n.sim <- 1000

# draw coefficients from the multivariate normal
S <- MASS::mvrnorm(n.sim, coef(m1), vcov(m1))
```

Simulation step 2: Choose a scenario for which you want to make a prediction. That means we have to set our covariates to some value. We will vary the percentage of the urban population and keep all other covariates constant. We also check the range of the variable of interest so that we don't extrapolate to something that is outside of our data range.

```r
# choose a scenario to predict the outcome for
summary(df$pctUrban)

# set the covariates (predictions for changes in pctUrban)
X <- cbind( constant = 1,
            urban = seq(from = 0, to = 1, by = .1),
            householdsize = mean(df$householdsize),
            pctwhite = mean(df$racePctWhite))

# check covariates
X
```

```
      constant urban householdsize  pctwhite
 [1,]        1   0.0     0.4633952 0.7537161
 [2,]        1   0.1     0.4633952 0.7537161
 [3,]        1   0.2     0.4633952 0.7537161
 [4,]        1   0.3     0.4633952 0.7537161
 [5,]        1   0.4     0.4633952 0.7537161
 [6,]        1   0.5     0.4633952 0.7537161
 [7,]        1   0.6     0.4633952 0.7537161
 [8,]        1   0.7     0.4633952 0.7537161
 [9,]        1   0.8     0.4633952 0.7537161
[10,]        1   0.9     0.4633952 0.7537161
[11,]        1   1.0     0.4633952 0.7537161
```

Simulation step 3: Predict the outcome. We have set our covariates and we have drawn our coefficients. This is all we need to predict y. Depending on the flavor of generalized linear model, y may have to be sent through a link function. In logistic regression we would send latent y trough the logit link function: $\frac{1}{1+exp^{-y}}$ to get probabilities that y is 1. Here, we ran a simple linear model so in linear algebra notation our prediction is simply $Y = X\beta$.

We estimate `y_hat` as a matrix where its rows are the number of simulations and its columns are the different covariate scenarios.

```r
# predict outcome, ie betas * X for all scenarios
y_hat <- S %*% t(as.matrix(X))
```

Finally, all that is left is the interpretation of the result. We can simply look at the numerical outcomes similar to using the `summary()` function on the Zelig simulation object.

```r
# output like the zelig summary (including estimation uncertainty)
apply(y_hat, 2, quantile, probs = c(.025, .5, .975))
```

```
           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
2.5%  0.4316010 0.4214979 0.4105675 0.4000181 0.3893180 0.3784349
50%   0.4450405 0.4332109 0.4215157 0.4097547 0.3981805 0.3864607
97.5% 0.4584981 0.4454806 0.4327917 0.4198413 0.4070521 0.3945582
           [,7]      [,8]      [,9]     [,10]     [,11]
2.5%  0.3672170 0.3557150 0.3439413 0.3318641 0.3196062
50%   0.3747705 0.3630690 0.3512949 0.3396307 0.3279723
97.5% 0.3823658 0.3703668 0.3586515 0.3473086 0.3363178
```

We can also draw a plot that shows our mean prediction and the uncertainty around in a few lines.

```
# plot like zelig's ci plot
par( mfrow = c(1,1))
plot(0, bty = "n", xlab = "Pct Urban", ylab = "Unemployment Rate",
     ylim = c(0.3, 0.5), xlim = c(0,1), pch ="")
ci <- apply(y_hat, 2, quantile, probs = c(.025, .975))
polygon(x =  c(rev(X[,"urban"]), X[,"urban"]),
        y = c(rev(t(ci)[,2]), t(ci)[,1]), border = NA,
        col = "lightblue")
lines(x = X[,"urban"], y = apply(y_hat, 2, quantile, probs = .5), lwd = 1)
```