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 unlabelled 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 analytical methods from this field focussing 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 text book: 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 Leeman.

Placeholder

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 analysing, 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.

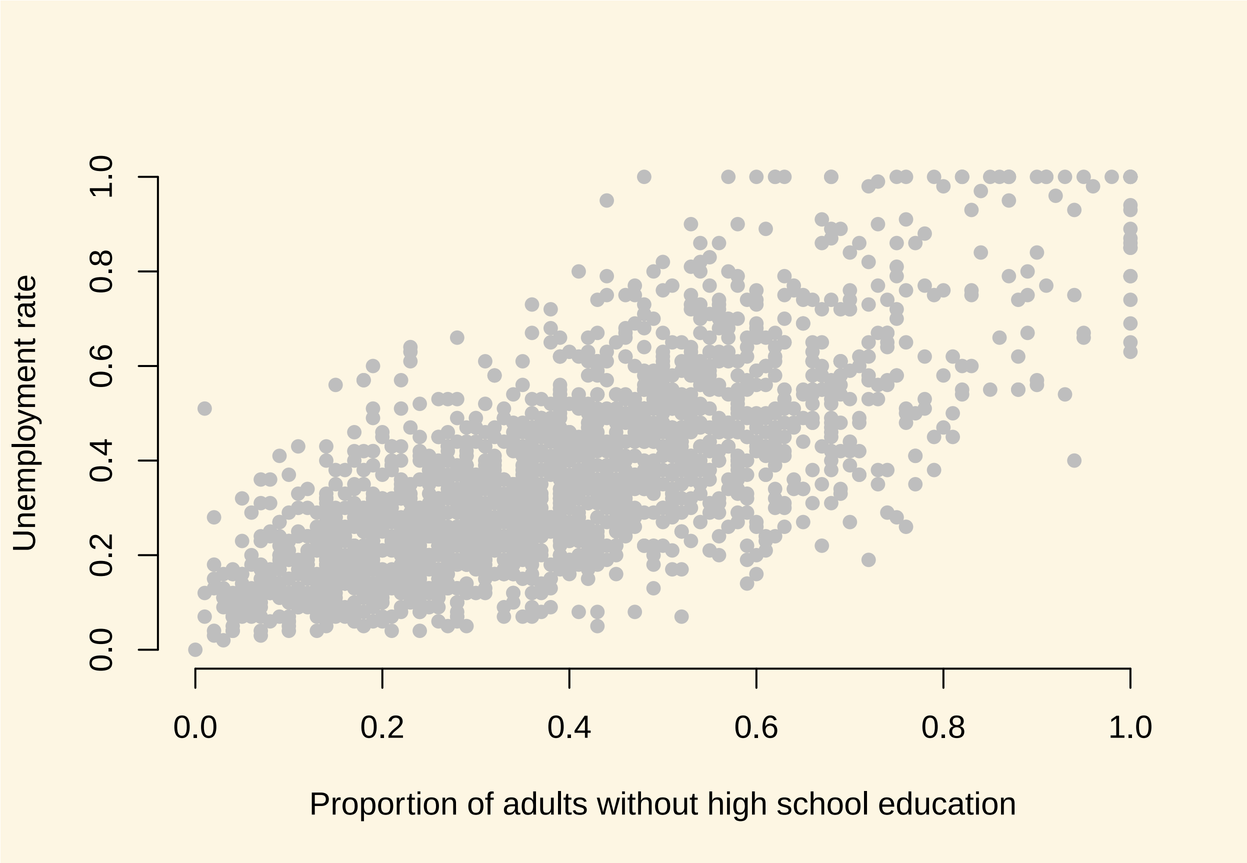
```
communities <- dplyr::select(
  communities,
  state,
  community = communityname,
  UnemploymentRate = PctUnemployed,
  NoHighSchool = PctNotHSGrad,
  white = racePctWhite)
```

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 Visualising a relationship b/w two continuous variables

A good way gauge whether two variables that both contiunous are related is to draw a scatter plot. We do so for the unemployment rate and for the lack of high shool 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)
```

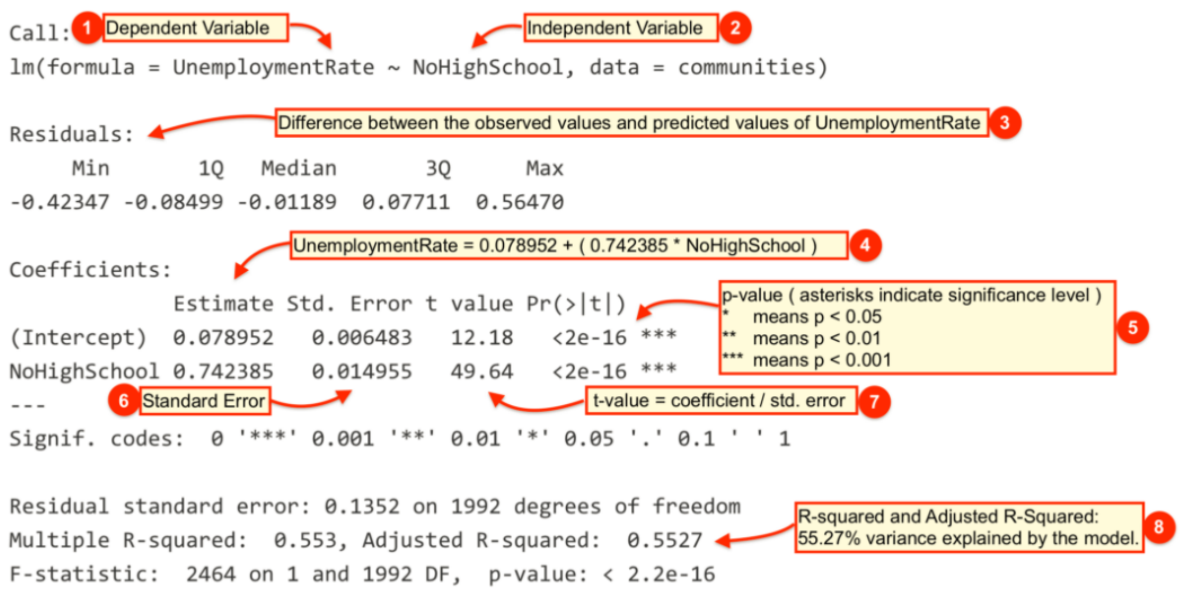
The `lm()` function modele the relationship between `UnemploymentRate` and `NoHighSchool` and we've assigned the estimated model to the object `m1`. We can use the `summary()` function on `m1` for the key results.

```
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 forecaset 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 quesses) 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)
```

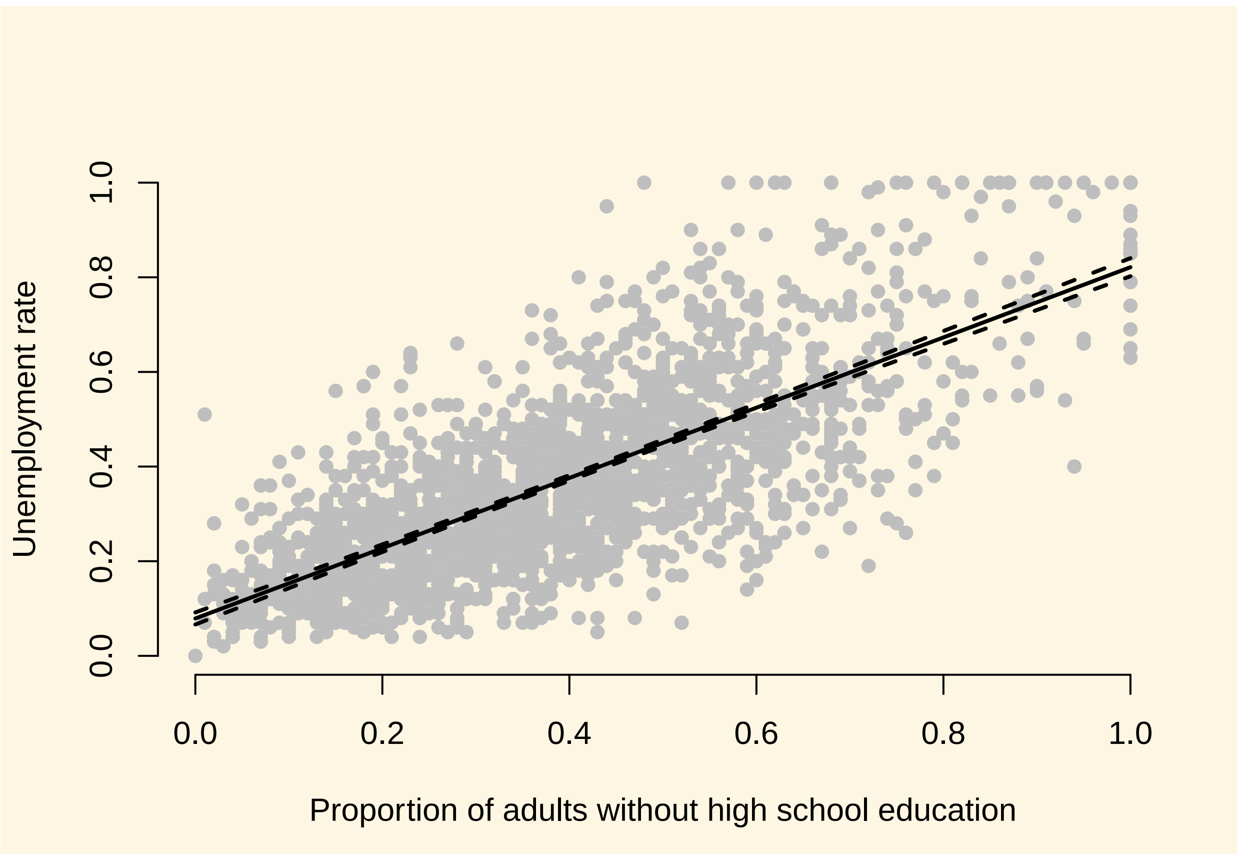
Let's inspect the first ten values of our data.

```
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
```

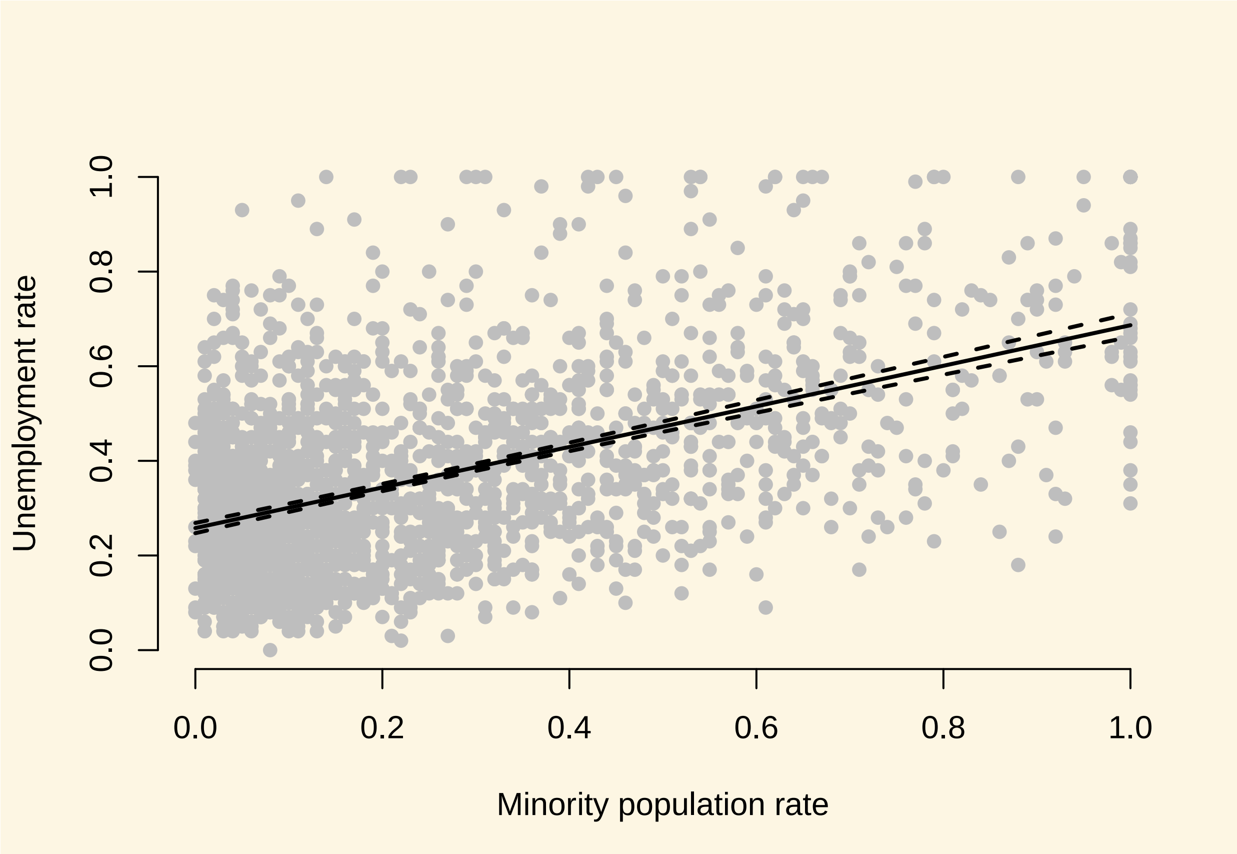
Next we fit a linear model using `Minority` as the independent variable.

```
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")
```

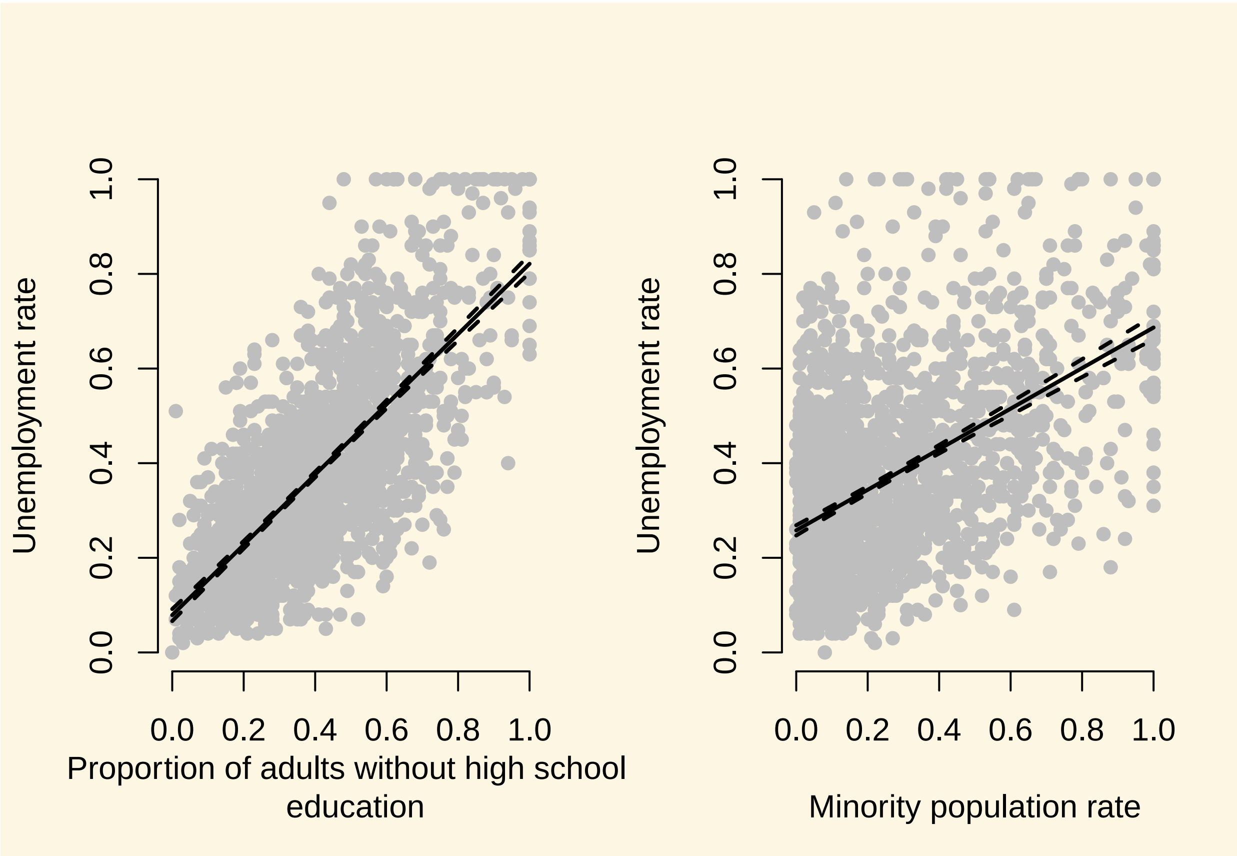
We now compare the models using the `texreg()` function like so:

```
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 indepedent 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 everone has a high school degree. Similarly, model 2 suggests that in a district where everone 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 measues of model fit. The difference between them is that `Adj. R^2` is a measure that penalises 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.

```r
# 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.

```r
# 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 
```

```r
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 internaly.

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 asign 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 normalising 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 substract the mean of x. Therefore, we centre 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 characer vector with the variable names of all variables that are continuous and lastly we normalise. We do this by subsetting 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 repeadetly for all the variables. The argument `X` is the data. The argument `MARGIN` says we want to apply our normalisation column-wise. The argument `FUN` means function. Here, we input our normlisation 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 maximise 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"`.

```r
# 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.

```r
# 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)
```

```r
# 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
```

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

```
[1] 0.7238095
```

The accuarcy 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.

```r
# 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
```

```r
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

Thera 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.

```r
# 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
```

```r
# 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:

```r
# 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 depdendent variable provides more information. In effect we turn our classification problem into a regression problem.

```r
# 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
```

```r
# 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

Placeholder

4 Subset Selection

4.1 Seminar

Placeholder

5 Regularisation

5.1 Seminar

Placholder

6 Polynomials

6.1 Seminar

Placeholder

7 Tree Based Models

7.1 Seminar

Placeholder

8 Simulation and Monte Carlo Simulation

8.1 Seminar

Placeholder