Essentials of Mathematics and Statistics

Practical: Module 1

Anas A Rana

2019-10-16

Contents

1 Introduction **1**
1.1 Prerequisites . . . 2
1.2 Data sets . . . 2

2 Practical: Linear regression **2**
2.1 Data . . . 2
2.2 Simulating data . . . 2
2.3 Fitting simple linear regression model . . . 4
2.4 Effect of variance . . . 9
2.5 Exercise . . . 10

Model answers: Linear regression **12**
2.6 Exercise I . . . 12
2.7 Exercise II . . . 13
2.8 Exercise II . . . 18

3 Practical: Principal component analysis **19**
3.1 Data . . . 19
3.2 Introduction . . . 19
3.3 Exercise I . . . 20
3.4 Exercise II . . . 22
3.5 Exercise III . . . 23
3.6 Exercise IV: Single cell data . . . 23

4 Practical: Multiple regression **26**
4.1 Multiple regression . . . 26
4.2 Categorical covariates . . . 27
4.3 Residuals . . . 31
4.4 Gradient descent algorithm (+) . . . 32

5 Practical: Generalised linear models **36**
5.1 Data . . . 36
5.2 Detecting SNP associations . . . 37
5.3 GWAS and logistic regression . . . 39
5.4 Negative binomial and Poisson regression . . . 40
5.5 Negative-Binomial vs Poisson GLMs . . . 43
5.6 Further understanding the model (**OPTIONAL**) . . . 43

1 Introduction

This is part of the practical for Module 1 - Essentials of Mathematics and Statistics part of the MSc Bioinformatics 2018/19 at the University of Birmingham.

This website hosts the practicals for week two of **Module 1**, which covers:

- Linear regression
- Principal Component Analysis (PCA)
- Multivariate Regression
- Generalised Linear Models

1.1 Prerequisites

Ensure you have attended Module 1 lectures and have the installed `R` and/or `Rstudio`. `Rstudio` is recommended, any packages required for each practical are mentioned at the beginning of each practical.

1.2 Data sets

For each practical there are some datasets required, you will find all data required for week two of practicals in the data folder here. Links to individual datasets requried can be found at the beginning of each practical or on Canvas.

2 Practical: Linear regression

In this practical you will go through some of the basics of linear modeling in `R` as well as simulating data. The practical contains the following elements:

- simulate linear regression model
- investigate parameters
- characterize prediction accuracy
- correlation of real world data

We will use `reshape2`, `ggplo2`, and `bbmle` packages. Run the following command to make sure they are installed and loaded

```
install.packages("ggplot2")
install.packages("reshape2")
install.packages("bbmle")
```

```
library(ggplot2)
library(reshape2)
library(bbmle)
```

2.1 Data

For this practical you will require three datasets:

- `stork.txt` (download)
- `lr_data1.Rdata` (download)
- `lr_data2.Rdata` (download).

2.2 Simulating data

You will simulate data based on the simple linear regression model:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i,$$

where (x_i, y_i) represent the i-th measurement pair with $i = 1, \ldots, N$, β_0 and β_1 are regression coefficients representing intercept and slope respectively. We assume the noise term $\epsilon_i \sim N(0, \sigma^2)$ is normally distributed with zero mean and variance σ^2.

First we define the values of the parameters of linear regression $(\beta_0, \beta_1, \sigma^2)$:

```r
b0 <- 10 # regression coefficient for intercept
b1 <- -8 # regression coefficient for slope
sigma2 <- 0.5 # noise variance
```

In the next step we will simulate $N = 100$ covariates x_i by randomly sampling from the standard normal distribution:

```r
set.seed(198) # set a seed to ensure data is reproducible
N <- 100 # no of data points to simulate
x <- rnorm(N, mean = 0, sd = 1) # simulate covariate
```

Next we simulate the error term:

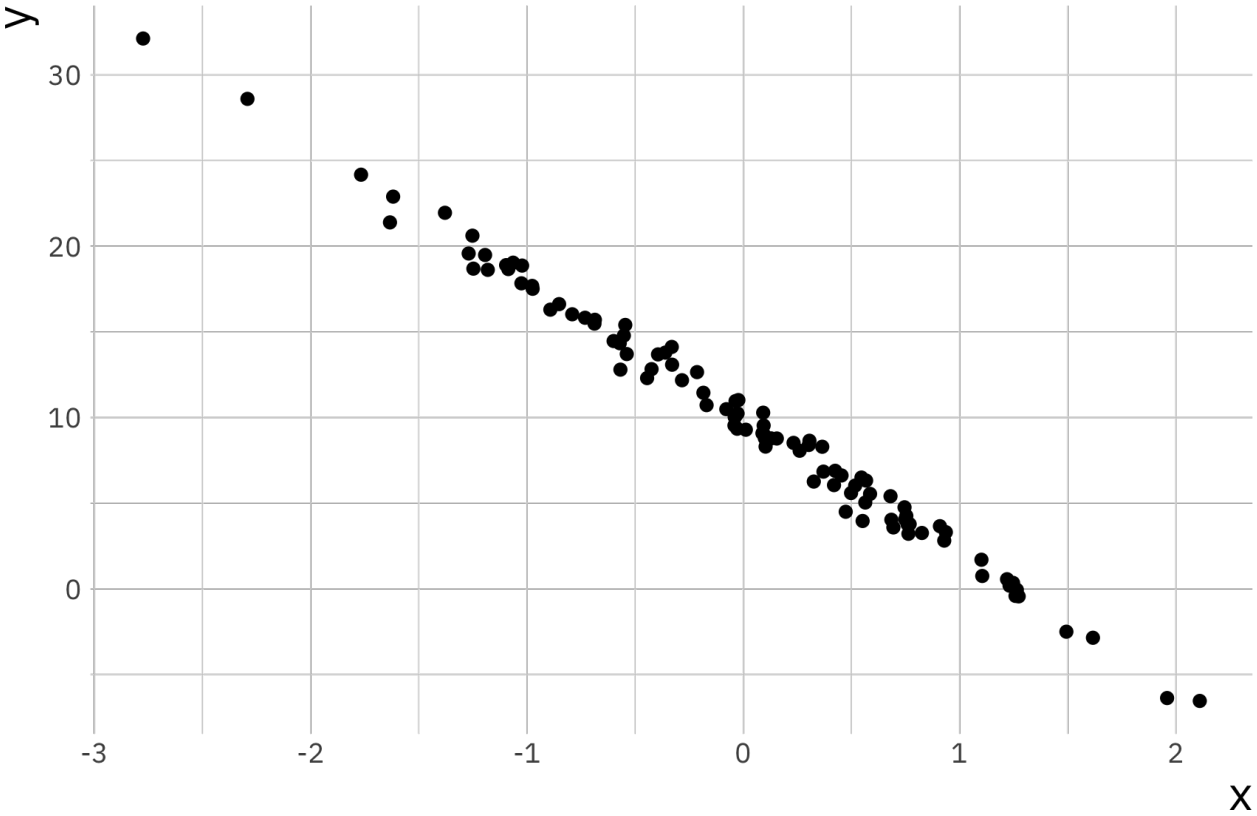
```r
# simulate the noise terms, rnorm requires the standard deviation
e <- rnorm(N, mean = 0, sd = sqrt(sigma2))
```

Finally we have all the parameters and variables to simulate the response variable y:

```r
# compute (simulate) the response variable
y = b0 + b1 * x + e
```

We will plot our data using `ggplot2` so the data need to be in a `data.frame` object:

```r
# Set up the data point
sim_data <- data.frame(x = x, y = y)

# create a new scatter plot using ggplot2
ggplot(sim_data, aes(x = x, y = y)) +
  geom_point()
```



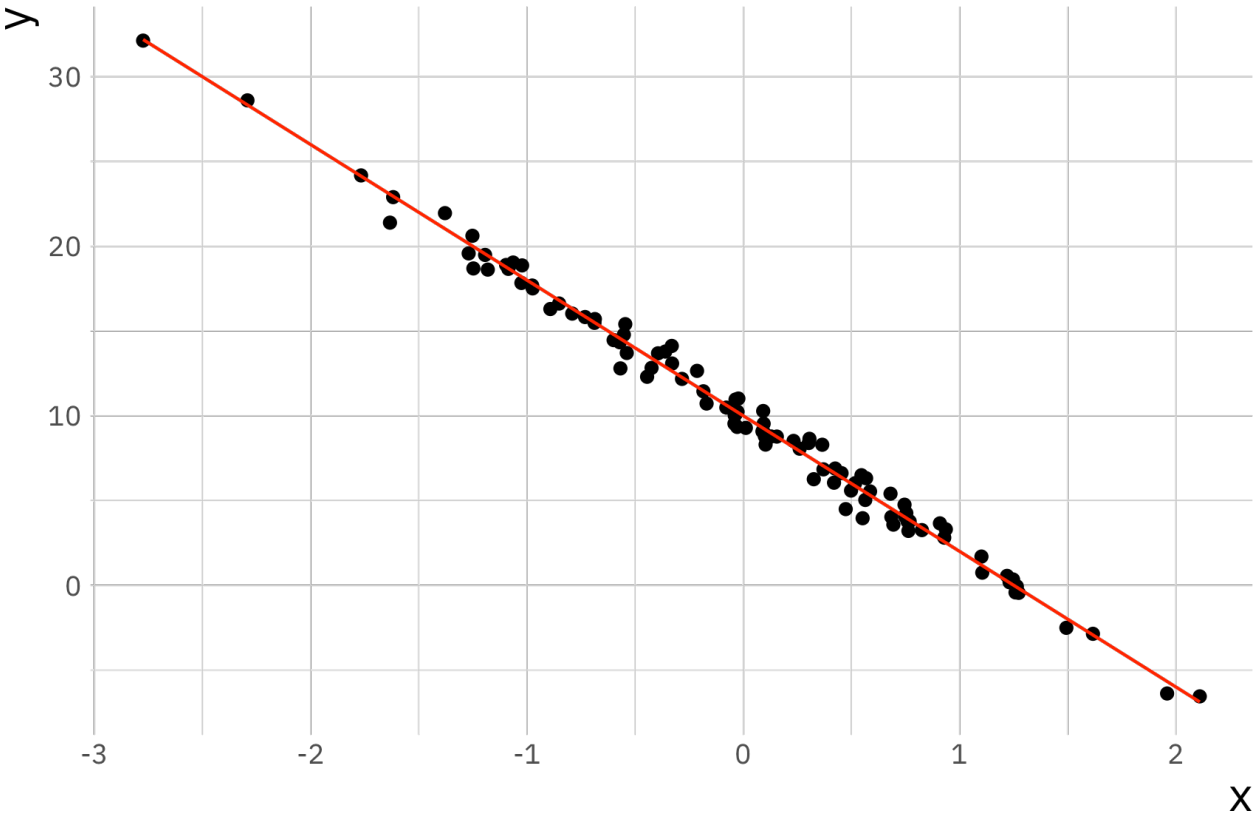
We define the true data `y_true` to be the true linear relationship between the covariate and the response

without the noise.

```
# Compute true y values
y_true <- b0 + b1 * x

# Add the data to the existing data frame
sim_data$y_true <- y_true
```

Now we will add the true values of y to the scatter plot:

```
lr_plot <- ggplot(sim_data, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(x = x, y = y_true), colour = "red")

print(lr_plot)
```



2.3 Fitting simple linear regression model

2.3.1 Least squared estimation

Now that you have simulated data you can use it to regress y on x, since this is simulated data we know the parameters and can make a comparison. In `R` we can use the function `lm()` for this, by default it implements a least squares estimate:

```
# Use the lm function to fit the data
ls_fit <- lm(y ~ x, data = sim_data)

# Display a summary of fit
summary(ls_fit)
```

```
##
```

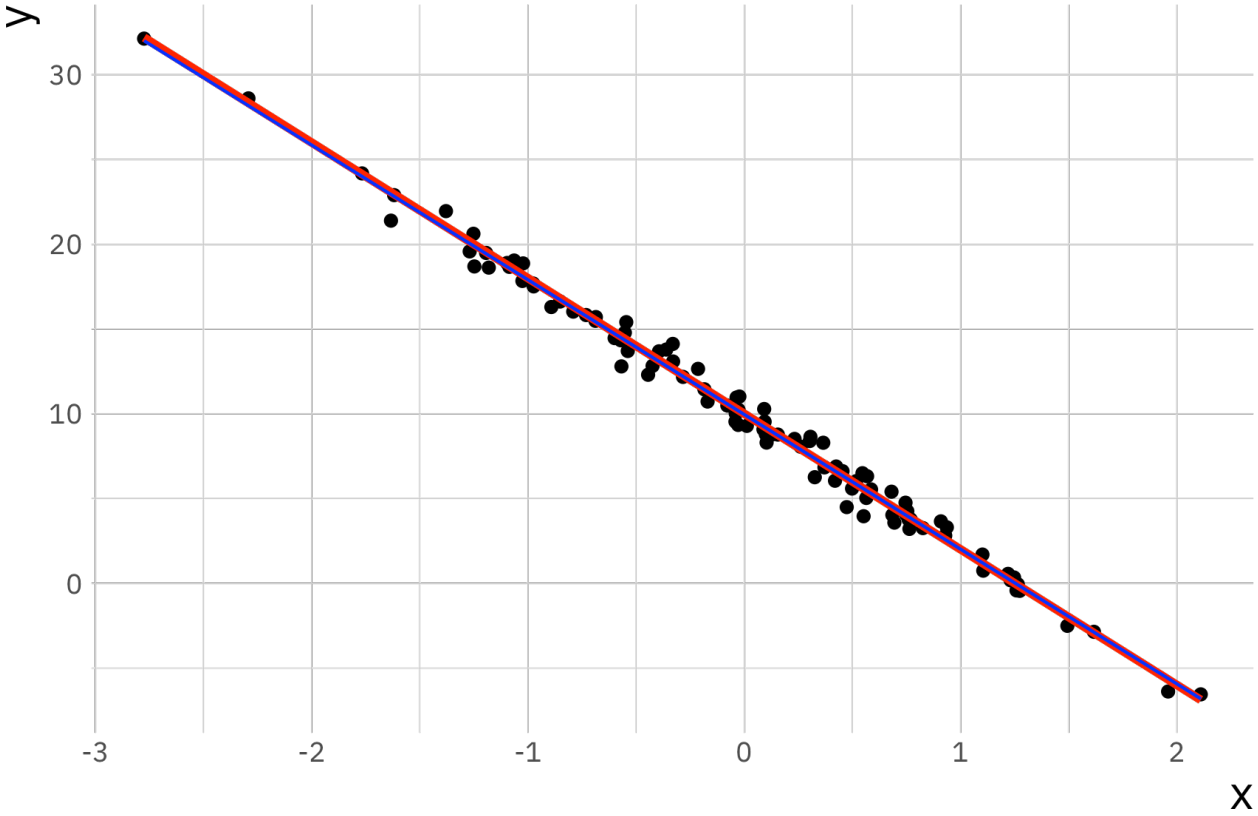
```
## Call:
## lm(formula = y ~ x, data = sim_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.69905 -0.41534  0.02851  0.41265  1.53651
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.95698    0.06701   148.6   <2e-16 ***
## x           -7.94702    0.07417  -107.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6701 on 98 degrees of freedom
## Multiple R-squared:  0.9915, Adjusted R-squared:  0.9914
## F-statistic: 1.148e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

The output for `lm()` is an object (in this case `ls_fit`) which contains multiple variables. To access them there are some built in functions, e.g. `coef()`, `residuals()`, and `fitted()`. We will explore these in turn:

```r
# Extract coefficients as a named vector
ls_coef <- coef(ls_fit)

print(ls_coef)
```

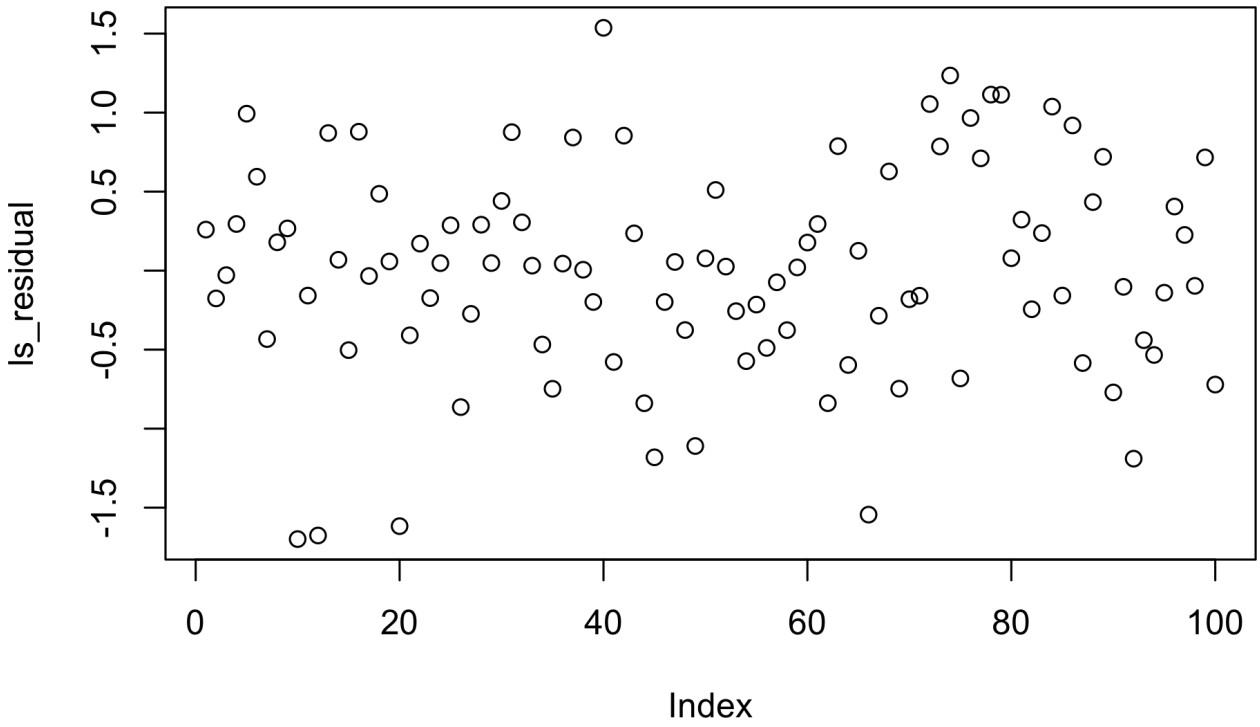
```
## (Intercept)           x
##    9.956981   -7.947016
```

```r
# Extract intercept and slope
b0_hat <- ls_coef[1] # alternative ls_fit$coefficients[1]
b1_hat <- ls_coef[2] # alternative ls_fit$coefficients[2]

# Generate the predicted data based on estimated parameters
y_hat <- b0_hat + b1_hat * x
sim_data$y_hat <- y_hat # add to the existing data frame

# Create scatter plot and lines for the original and fitted
lr_plot <- ggplot(sim_data, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(x = x, y = y_true), colour = "red", size = 1.3) +
  # plot predicted relationship in blue
  geom_line(aes(x = x, y = y_hat), colour = "blue")

# force Rstudio to display the plot
  print(lr_plot)
```



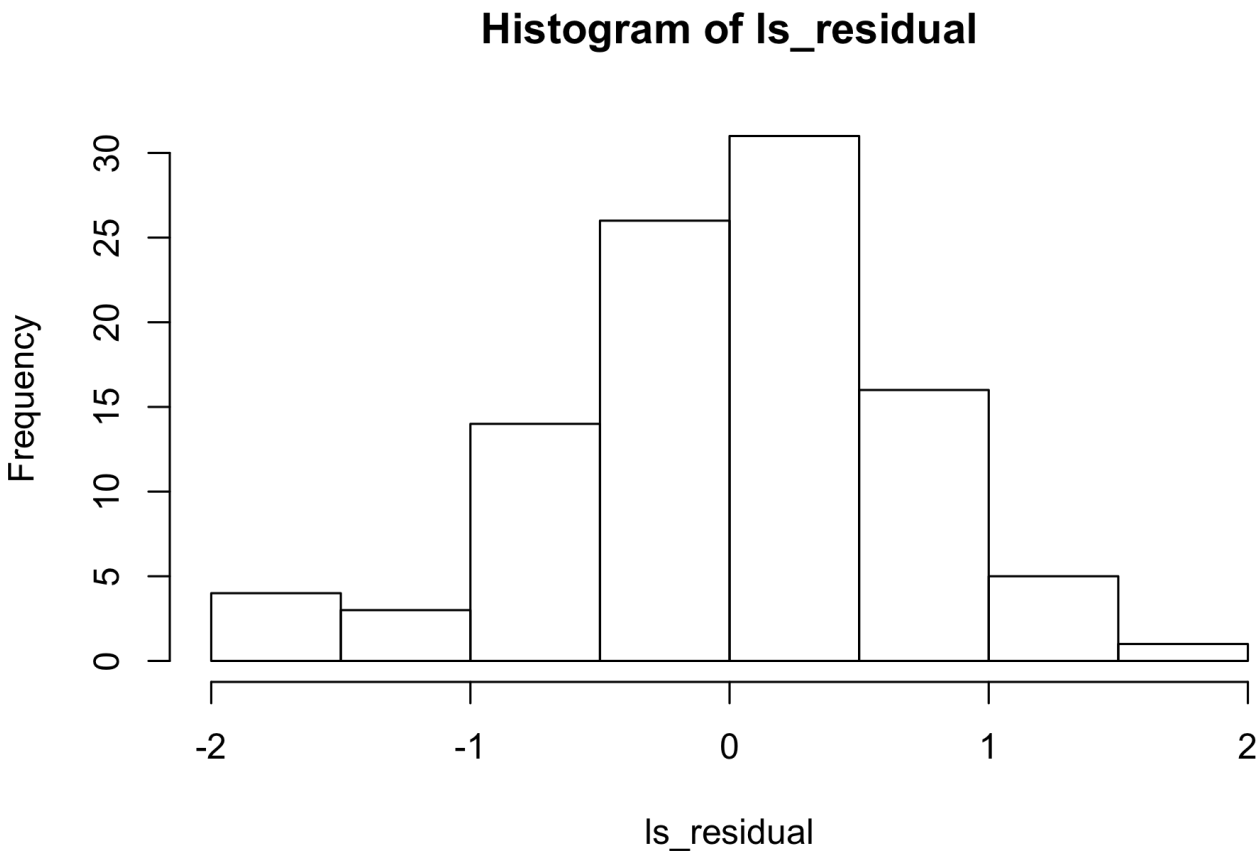
The estimated parameters and the plot shows a good correspondence between fitted regression parameters and the true relationship between y and x. We can check this by plotting the residuals, this data is stored as the `residuals` parameter in the `ls_fit` object.

```
# Residuals
ls_residual <- residuals(ls_fit) # can also be accessed via ls_fit$residuals

# scatter plot of residuals
plot(ls_residual)
```



A better way of summarising the data is to visualise them as a histogram:

```
hist(ls_residual)
```



We expect the mean and variance of the residuals to be close to the level used to generate the data.

```r
print(mean(ls_residual))
```

```
## [1] -7.157903e-18
```

```r
print(var(ls_residual))
```

```
## [1] 0.4444955
```

This is as expected since subtracting a good fit from the data leaves ϵ which has 0 mean and 0.5 variance.

2.3.2 Maximum likelihood estimation

Next you will look at maximum likelihood estimation based on the same data you simulated earlier. This is a bit more involved as it requires you to explicitly write the function you wish to minimise. The function we use is part of the `bbmle` package.

```r
# Loading the required package
library(bbmle)

# function that will be minimised. It takes as arguments all parameters
# Here we are helped by the way R works we don't have to explicitly pass x.
# The function will use the existing estimates in the environment
mle_ll <- function(beta0, beta1, sigma) {
  # first we predict the response variable based on the guess for our response
  y_pred = beta0 + beta1 * x

  # next we calculate the normal distribution based on the predicted value
  # the guess for sigma and return the log
  log_lh <- dnorm(y, mean = y_pred, sd = sigma, log = TRUE)

  #  We returnr the negative sum of the log likelihood
  return(-sum(log_lh))
}

# This is the function that actually performs the estimation
# The first variable here is the function we will use
# The second variable passed is a list of initial guesses of parameters
mle_fit <- mle2(mle_ll, start = list(beta0 = -1, beta1 = 20, sigma = 10))

# With the same summary function as above we can output a summary of the fit
summary(mle_fit)
```

```
## Maximum likelihood estimation
##
## Call:
## mle2(minuslogl = mle_ll, start = list(beta0 = -1, beta1 = 20,
##     sigma = 10))
##
## Coefficients:
##        Estimate Std. Error  z value     Pr(z)
## beta0  9.957019   0.066336  150.099 < 2.2e-16 ***
## beta1 -7.947005   0.073426 -108.231 < 2.2e-16 ***
## sigma  0.663347   0.046904   14.143 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## -2 log L: 201.7011
```

The estimated parameters using the maximum likelihood are also a very good estimate of the true values.

2.4 Effect of variance

Now investigate the quality of the predictions further by simulating more data sets and seeing how the variance affects the quality of the fit as indicated by the mean-squared error (mse).

To start you will define some parameter for the simulations, the number of simulations to run for each variance, and the variance values to try.

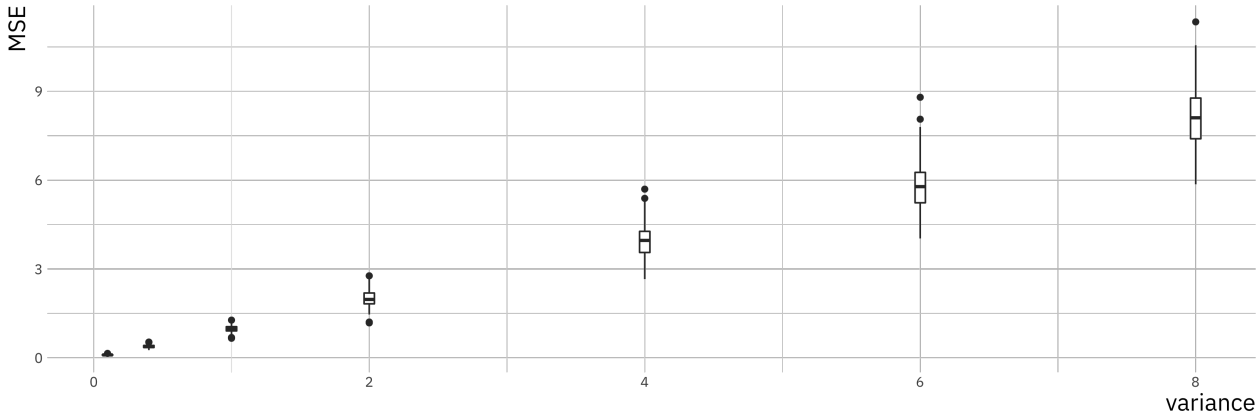
```
# number of simulations for each noise level
n_simulations <- 100

# A vector of noise levels to try
sigma_v <- c(0.1, 0.4, 1.0, 2.0, 4.0, 6.0, 8.0)
n_sigma <- length(sigma_v)

# Create a matrix to store results
mse_matrix <- matrix(0, nrow = n_simulations, ncol = n_sigma)

# name row and column
rownames(mse_matrix) <- c(1:n_simulations)
colnames(mse_matrix) <- sigma_v
```

Next we will write a nested `for` loop. The first loop will be over the variances and a second loop over the number of repeats. We will simulate the data, perform a fit with `lm()`. We can use the `fitted()` function on the resulting object to extract the fitted values $\hat{y}$ and use this to compute the mean-squared error from the true value y.

```
# loop over variance
for (i in 1:n_sigma) {
  sigma2 <- sigma_v[i]

  # for each simulation
  for (it in 1:n_simulations) {

    # Simulate the data
    x <- rnorm(N, mean = 0, sd = 1)
    e <- rnorm(N, mean = 0, sd = sqrt(sigma2))
    y <- b0 + b1 * x + e

    # set up a data frame and run lm()
    sim_data <- data.frame(x = x, y = y)
    lm_fit <- lm(y ~ x, data = sim_data)

    # compute the mean squared error between the fit and the actual y's
    y_hat <- fitted(lm_fit)
    mse_matrix[it, i] <- mean((y_hat - y)^2)

  }
}
```

We created a matrix to store the mse values, but to plot them using `ggplot2` we have to convert them to a `data.frame`. This can be done using the `melt()` function form the `reshape2` library. We can compare the results using boxplots.

```r
library(reshape2)

# convert the matrix into a data frame for ggplot2
mse_df <- melt(mse_matrix)
# rename the columns
names(mse_df) <- c("Simulation", "variance", "MSE")

# now use a boxplot to look at the relationship between
# mean-squared prediction error and variance
mse_plt <- ggplot(mse_df, aes(x = variance, y = MSE)) +
  geom_boxplot(aes(group = variance))

print(mse_plt)
```



You can see that the variances of the mse and the value of the mse go up with increasing variance in the simulation.

What changes do you need to make to the above function to plot the accuracy of the estimated regression coefficients as a function of variance?

2.5 Exercise

2.5.1 Part I

Read in the data in `stork.txt`, compute the correlation and comment on it.

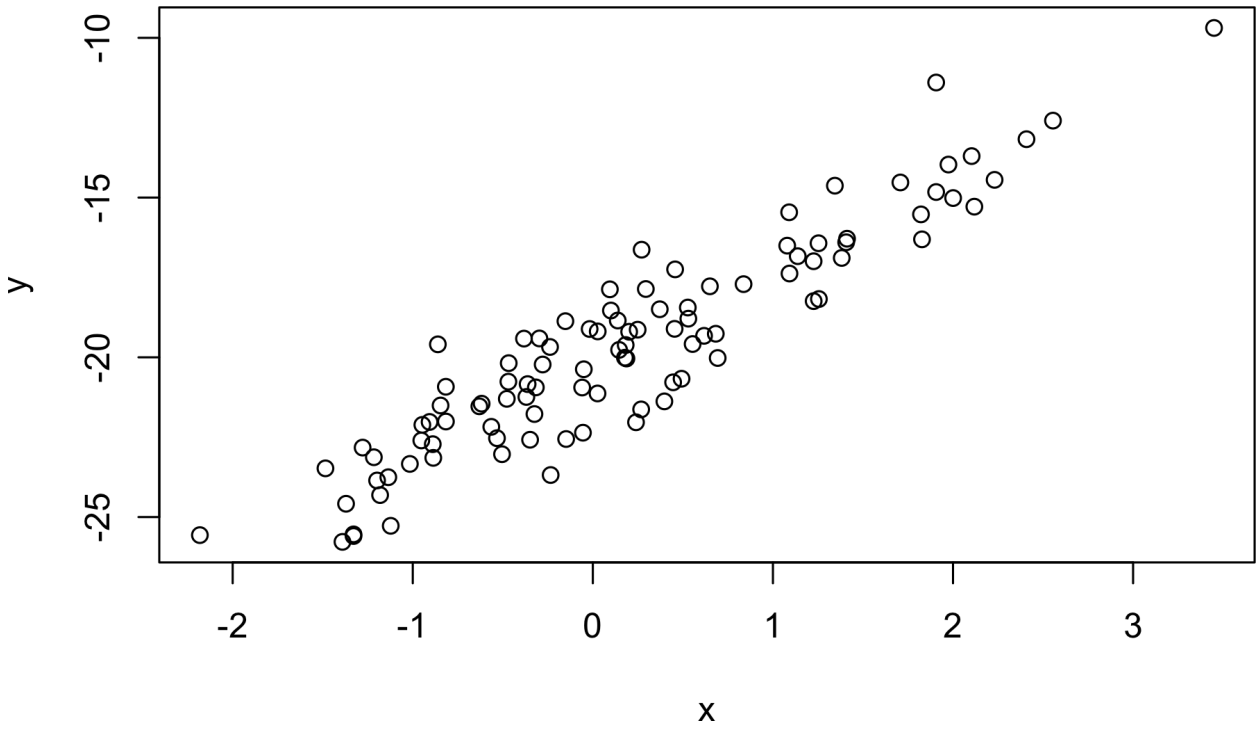
The data represents `no of storks` (column 1) in Oldenburg Germany from $1930 - 1939$ and the number of people (column 2).

2.5.2 Part II

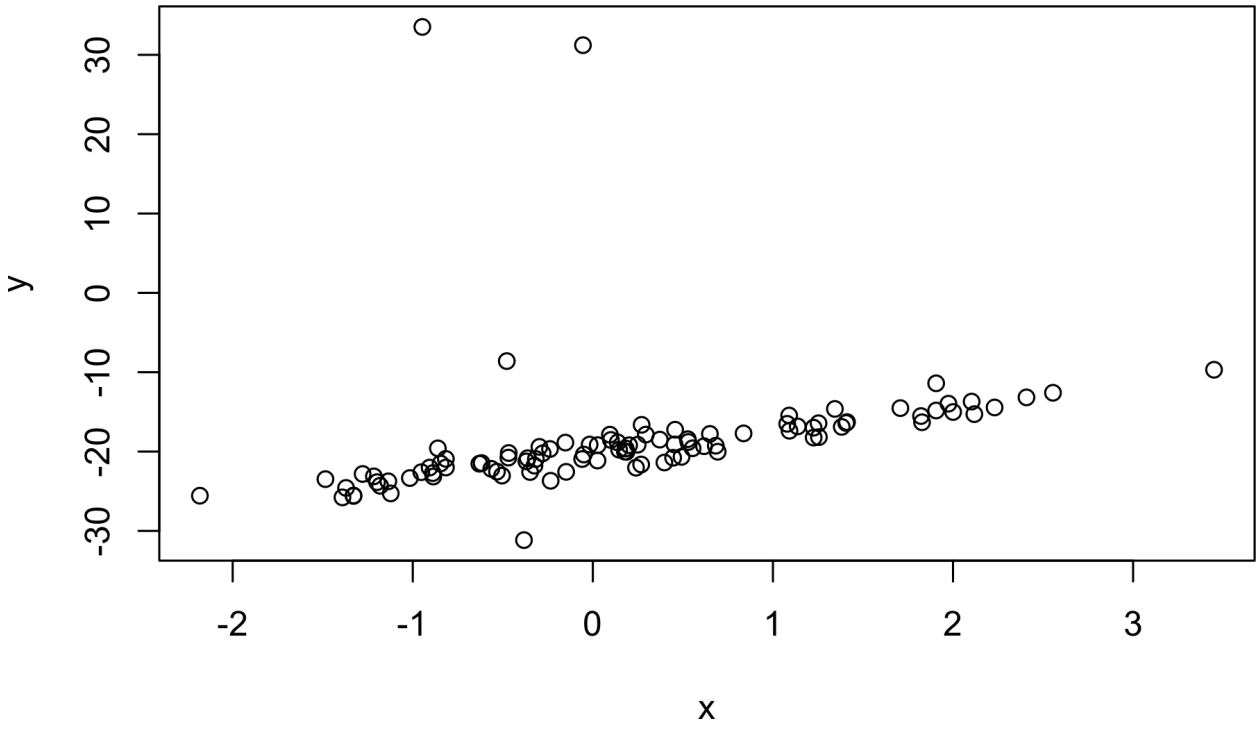
Fit a simple linear model to the two data sets supplied (`lr_data1.Rdata` and `lr_data2.Rdata`). In both files the (x, y) data is saved in two vectors, x and y.

Download the data from Canvas, you can read it into `R` and plot it with the following commands:

```r
load("lr_data1.Rdata")
plot(x, y)
```




```
load("lr_data2.Rdata")
plot(x, y)
```



Fit the linear model and comment on the differences between the data.

2.5.3 Part III

Investigate how the sample size will affect the quality of the fit using mse, use the code for investigating the affect of variance as inspiration.

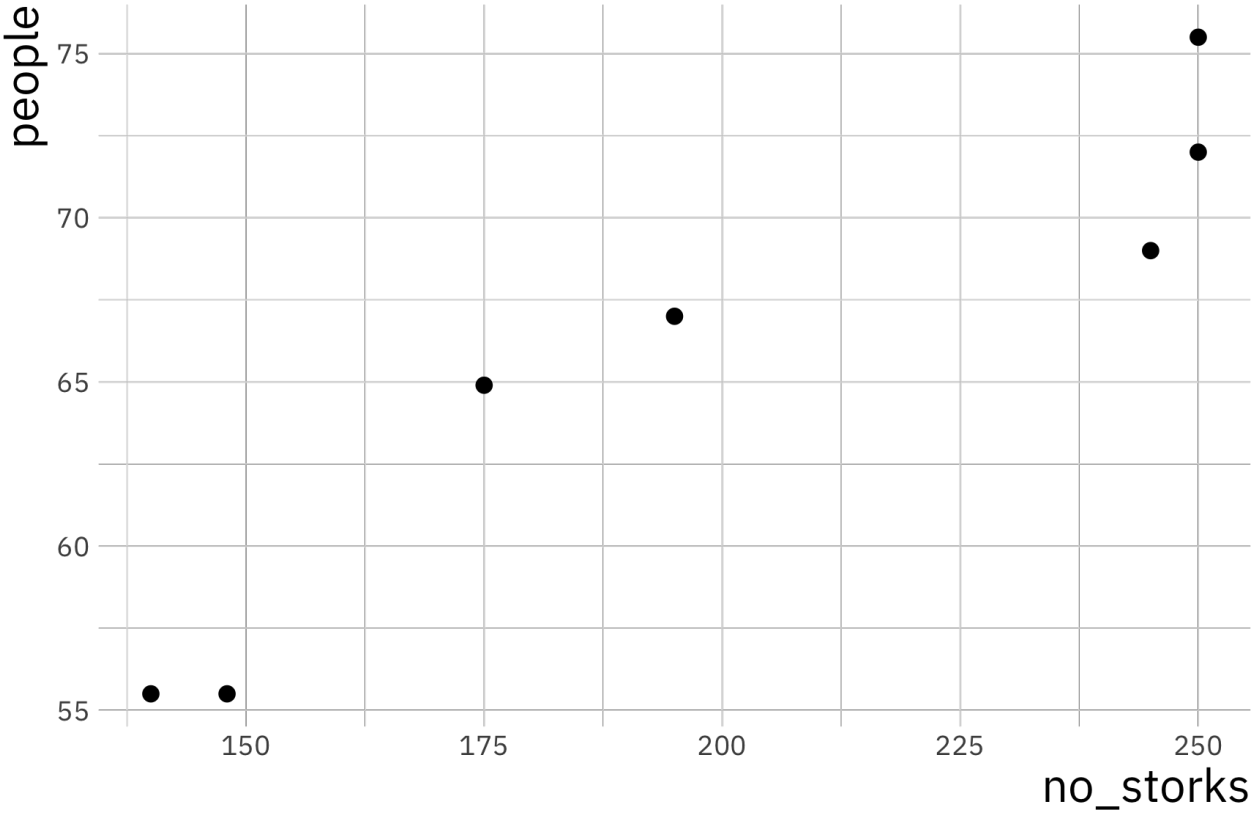
Model answers: Linear regression

2.6 Exercise I

```
library(ggplot2)
library(reshape2)
```

```
stork_dat <- read.table("stork.txt", hedaer = TRUE)
```

```
ggplot(stork_dat, aes(x = no_storks, y = people)) +
  geom_point(size = 2)
```



This is a plot of number of people in Oldenburg (Germany) against the number of storks. We can calculate the correlation in R

```
cor(stork_dat$no_storks, stork_dat$peopl)
```

```
## [1] 0.9443965
```

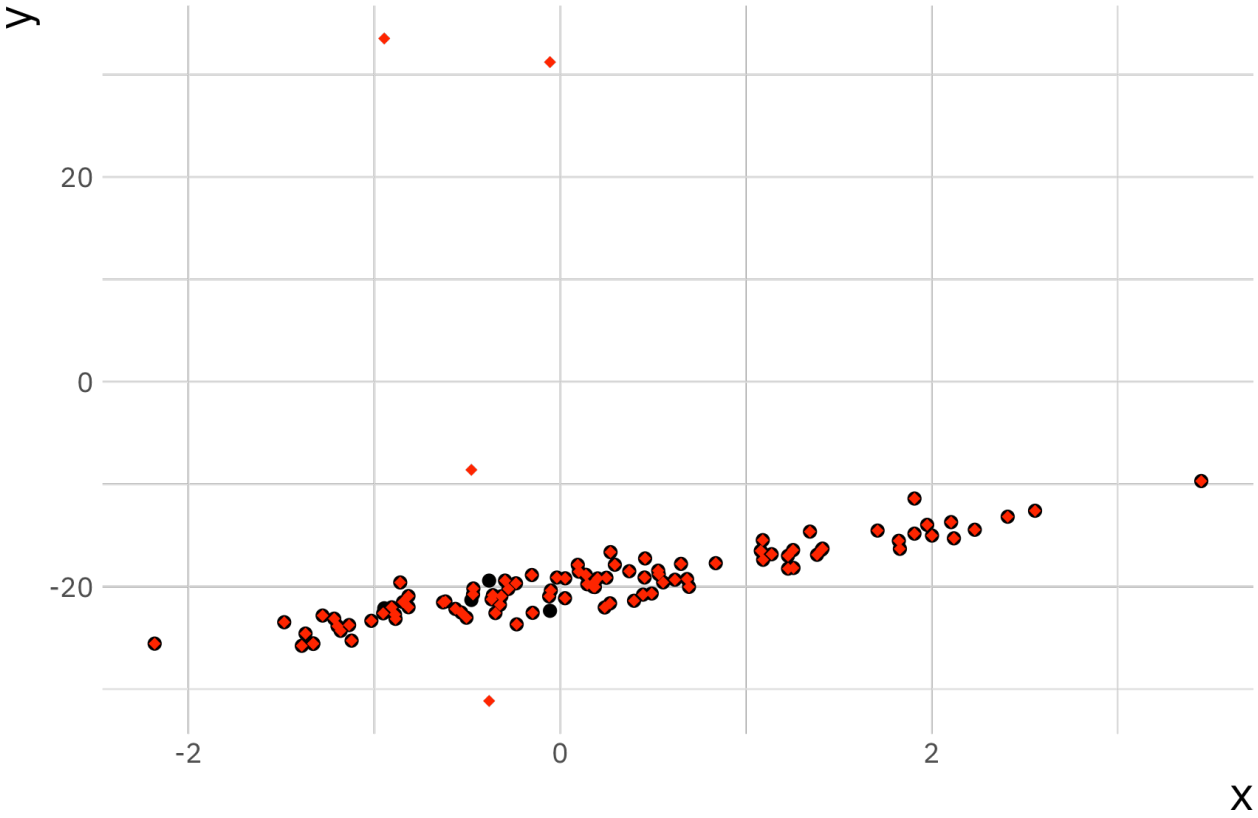
This is a very high correlation, and obviously there is no causation. Think about why there would be a correlation between these two random variables.

2.7 Exercise II

```
# load first data set and create data.frame
load("lr_data1.Rdata")
sim_data1 <- data.frame(x = x, y = y)

# load second data set and create data.frame
load("lr_data2.Rdata")
sim_data2 <- data.frame(x = x, y = y)

lr_fit1 <- lm(y ~ x, data = sim_data1)
lr_fit2 <- lm(y ~ x, data = sim_data2)
```

2.7.1 Comparison of data

```
ggplot(sim_data1, aes(x = x, y = y)) +
  geom_point(size = 1.5) +
  geom_point(data = sim_data2, color = "red", shape = 18)
```



If we plot the data on top of each other, the first data set in black and the second one in red, we can see a small number of points are different between the two data sets.

```
summary(lr_fit1)
```

```
##
## Call:
## lm(formula = y ~ x, data = sim_data1)
##
## Residuals:
```

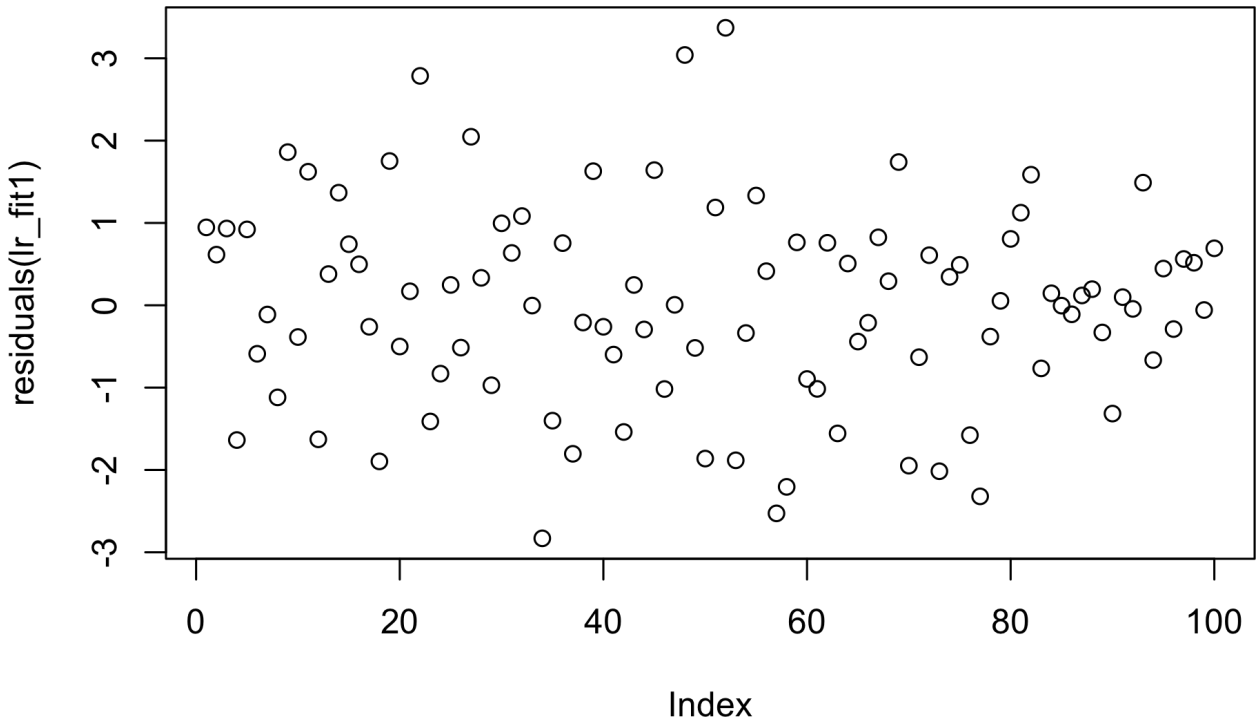
```
##     Min      1Q  Median      3Q     Max
## -2.8309 -0.6910  0.0296  0.7559  3.3703
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -20.1876     0.1250 -161.46   <2e-16 ***
## x             2.8426     0.1138   24.98   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.229 on 98 degrees of freedom
## Multiple R-squared:  0.8643, Adjusted R-squared:  0.8629
## F-statistic: 624.2 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
summary(lr_fit2)
```

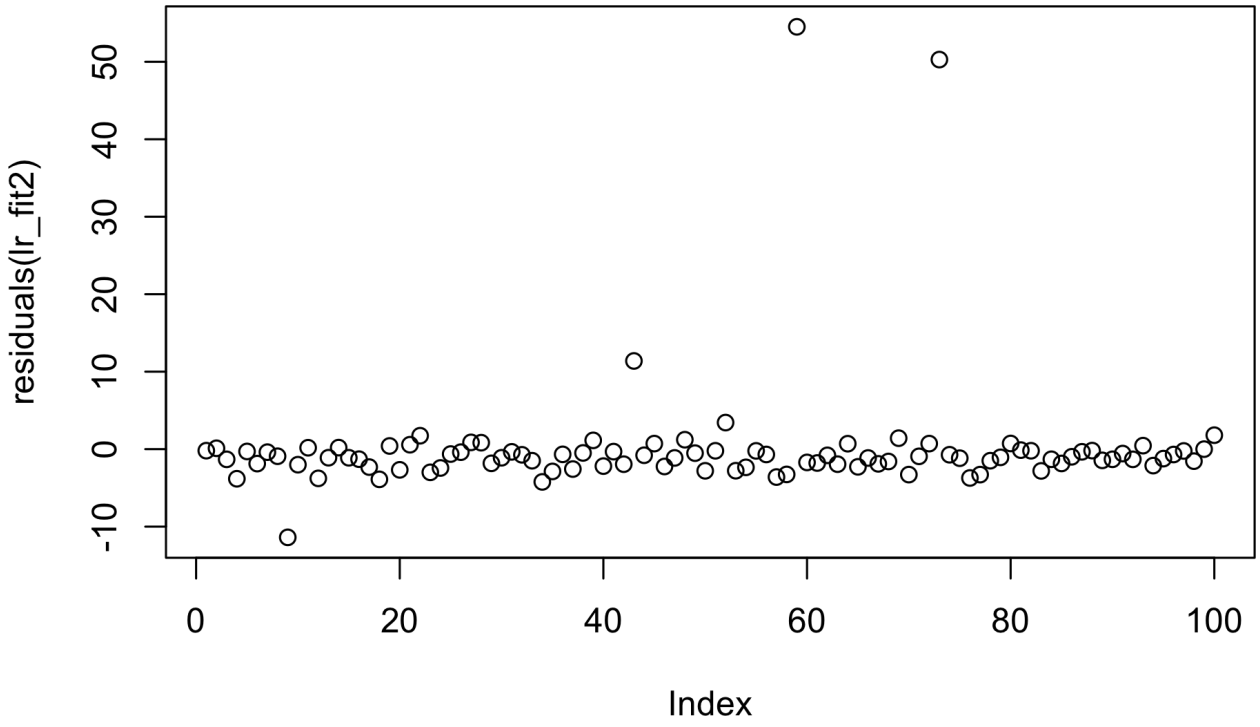
```
##
## Call:
## lm(formula = y ~ x, data = sim_data2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.386  -1.960  -1.084  -0.206  54.516
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -18.9486     0.8006 -23.669  < 2e-16 ***
## x             2.1620     0.7285   2.968  0.00377 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.87 on 98 degrees of freedom
## Multiple R-squared:  0.08245,    Adjusted R-squared:  0.07309
## F-statistic: 8.806 on 1 and 98 DF,  p-value: 0.003772
```

From the summary data we can see a discrepancy between the two estimates in the regression coefficients (≈ 1), though the error in the estimate is quite large. The other thing to notice is that the summary of the residuals look quite different. If we investigate further and plot them we see:

```
plot(residuals(lr_fit1))
```

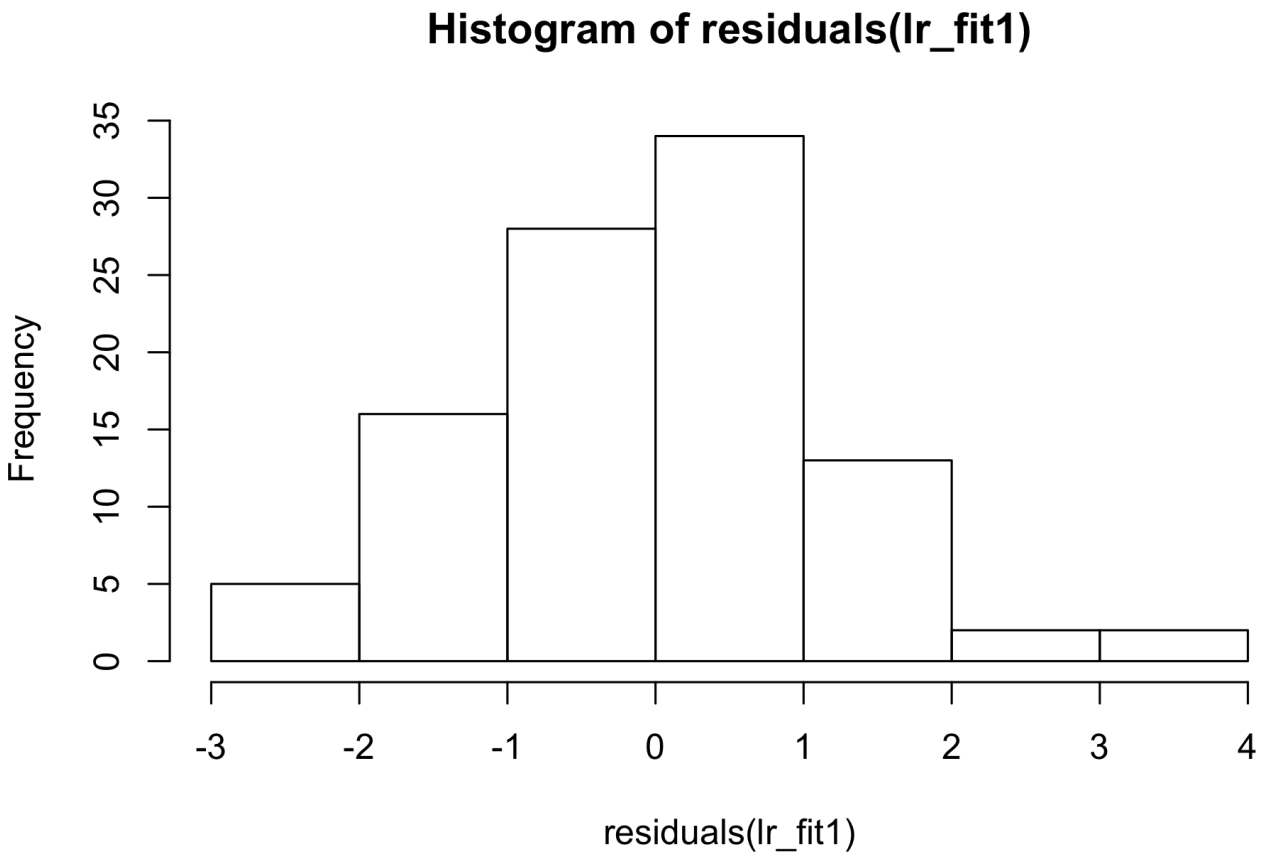



```
plot(residuals(lr_fit2))
```

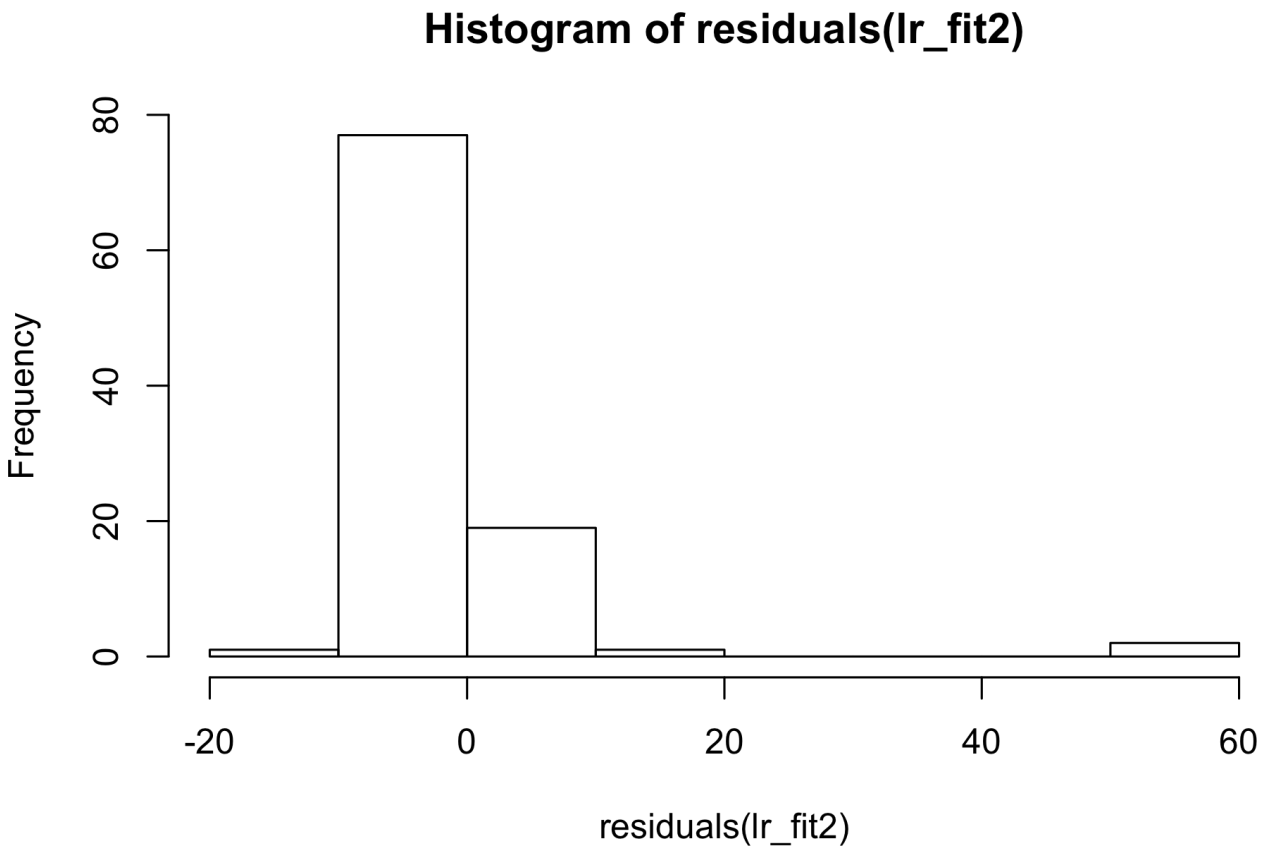


Here we can once again see the outliers in the second data set which affect the estimation. We now plot the histogram and boxplots for comparison:

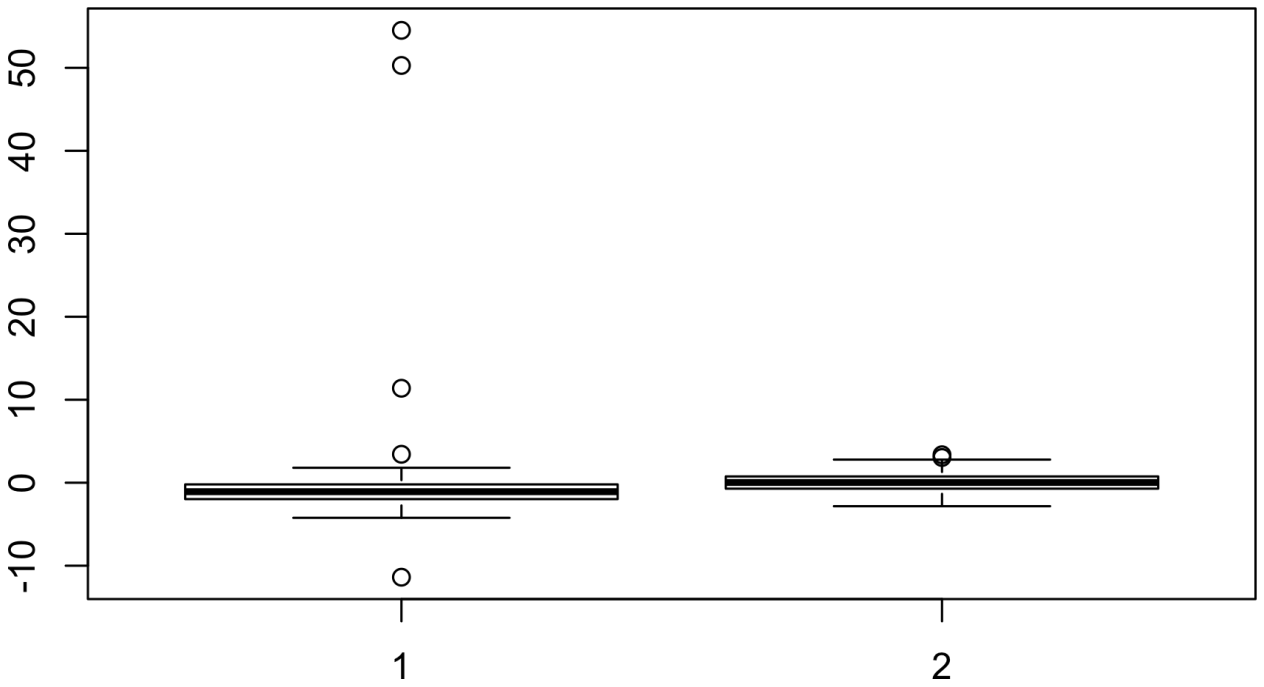
```
hist(residuals(lr_fit1))
```




```
hist(residuals(lr_fit2))
```




```
boxplot(residuals(lr_fit2), residuals(lr_fit1))
```



Her we can see that the distribution of the residuals has significantly changed in data set 2.

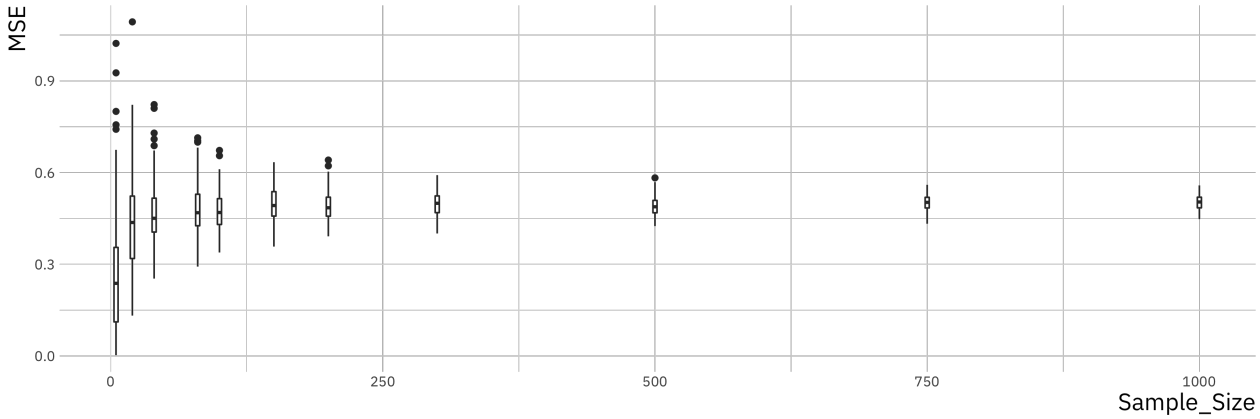
A change in only 4 data points was sufficient to change the regression coefficients.

2.8 Exercise II

```r
b0 <- 10 # regression coefficient for intercept
b1 <- -8 # regression coefficient for slope
sigma2 <- 0.5 # noise variance

# number of simulations for each sample size
n_simulations <- 100

# A vector of sample sizes to try
sample_size_v <- c( 5, 20, 40, 80, 100, 150, 200, 300, 500, 750, 1000 )

n_sample_size <- length(sample_size_v)

# Create a matrix to store results
mse_matrix <- matrix(0, nrow = n_simulations, ncol = n_sample_size)

# name row and column
rownames(mse_matrix) <- c(1:n_simulations)
colnames(mse_matrix) <- sample_size_v
```

```r
# loop over sample size
for (i in 1:n_sample_size) {
  N <- sample_size_v[i]

  # for each simulation
  for (it in 1:n_simulations) {

    x <- rnorm(N, mean = 0, sd = 1)
    e <- rnorm(N, mean = 0, sd = sqrt(sigma2))
    y <- b0 + b1 * x + e

    # set up a data frame and run lm()
    sim_data <- data.frame(x = x, y = y)
    lm_fit <- lm(y ~ x, data = sim_data)

    # compute the mean squared error between the fit and the actual y's
    y_hat <- fitted(lm_fit)
    mse_matrix[it, i] <- mean((y_hat - y)^2)

  }
}
```

```r
library(reshape2)

mse_df = melt(mse_matrix) # convert the matrix into a data frame for ggplot
names(mse_df) = c("Simulation", "Sample_Size", "MSE") # rename the columns

# now use a boxplot to look at the relationship between mean-squared prediction error and sample size
mse_plt = ggplot(mse_df, aes(x=Sample_Size, y=MSE))
mse_plt = mse_plt + geom_boxplot( aes(group=Sample_Size) )
print(mse_plt)
```



You should see that the variance of the mean-squared error goes down as the sample size goes up and converges towards a limiting value. Larger sample sizes help reduce the variance in our estimators but do not make the estimates more accurate.

Can you do something similar to work out the relationship between how accurate the regression coefficient estimates are as a function of sample size?

3 Practical: Principal component analysis

In this practical we will practice some of the ideas outlined in the lecture on Principal Component Analysis (PCA), this will include computing principal components, visualisation techniques and an application to real data.

3.1 Data

For this practical we will use some data that is built into R and we require two additional files you will download:

- `Pollen2014.txt` (download)
- `SupplementaryLabels.txt` (download)

3.2 Introduction

We use PCA in order to explore complex datasets. By performing dimensionality reduction we can better visualize the data that has many variables. This technique is probably the most popular tool applied across bioscience problems (e.g. for gene expression problems).

In many real-world dataset we deal with a high dimensional data, e.g. for a number of individuals we can take a number of health related measurement (called variables). This is great, however having a large number of variables also means that it is difficult to plot the data as it is (in its "*raw*" format), and in turn it might be difficult to understand if this dataset contains any interesting patterns/trends/relationships across individuals. Using PCA we visualize such data in a more "*human friendly*" fashion.

Recall:

- PCA performs a linear transformation to data.
- This means that any input data can be visualized in a new coordinate system. The first coordinate (PC 1) variance is found on the first coordinate; each subsequent coordinate is orthogonal to the previous one and contains the larges variance from what was left.
- Each principal component is associated with certain percentage of the total variation in the dataset.
- If variables are strongly correlated with one another, a first few principal components will enable us to visualize the relationships present in any dataset.

- Eigenvectors describe new directions, whereas accompanying eigenvalues tell us how much variance there is in the data in given direction.
- The eigenvector with the highest eigenvalue is called the first principal component. The second highest eigenvalue would correspond to a second principle component and etc.
- There exist a d number of eigenvalues and eigenvectors; d is also equal to the size of the data (number of dimensions).
- For the purpose of visualization we preselect the first q components, where $q < d$.

3.3 Exercise I

There are many datasets built into `R`. Wed will look at the `mtcars` dataset. Type `?mtcars` to get a description of data. Then use `head()` function to have a look at the first few rows; and `dim()` to get the dimensions of the data.

```
library(ggplot2)
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
dim(mtcars)
```

```
## [1] 32 11
```

In this case we have 32 examples (cars in this case), and 11 features. Now we can perform a principal component analysis, in `R` it is implemented as the `prcomp()` function. We can type `?prcomp` to see a description of the function and some help on possible arguments. Here we set `center` and `scale` arguments to `TRUE`, recall from the lecture why this is important. We can try to perform PCA without scaling and centering and compare.

```
cars_pca <- prcomp(mtcars, center = TRUE, scale = TRUE)
```

We can use the `summary()` function to summarise the results from PCA, it will return the standard deviation, the proportion of variance explained by each principal component, and the cumulative proportion.

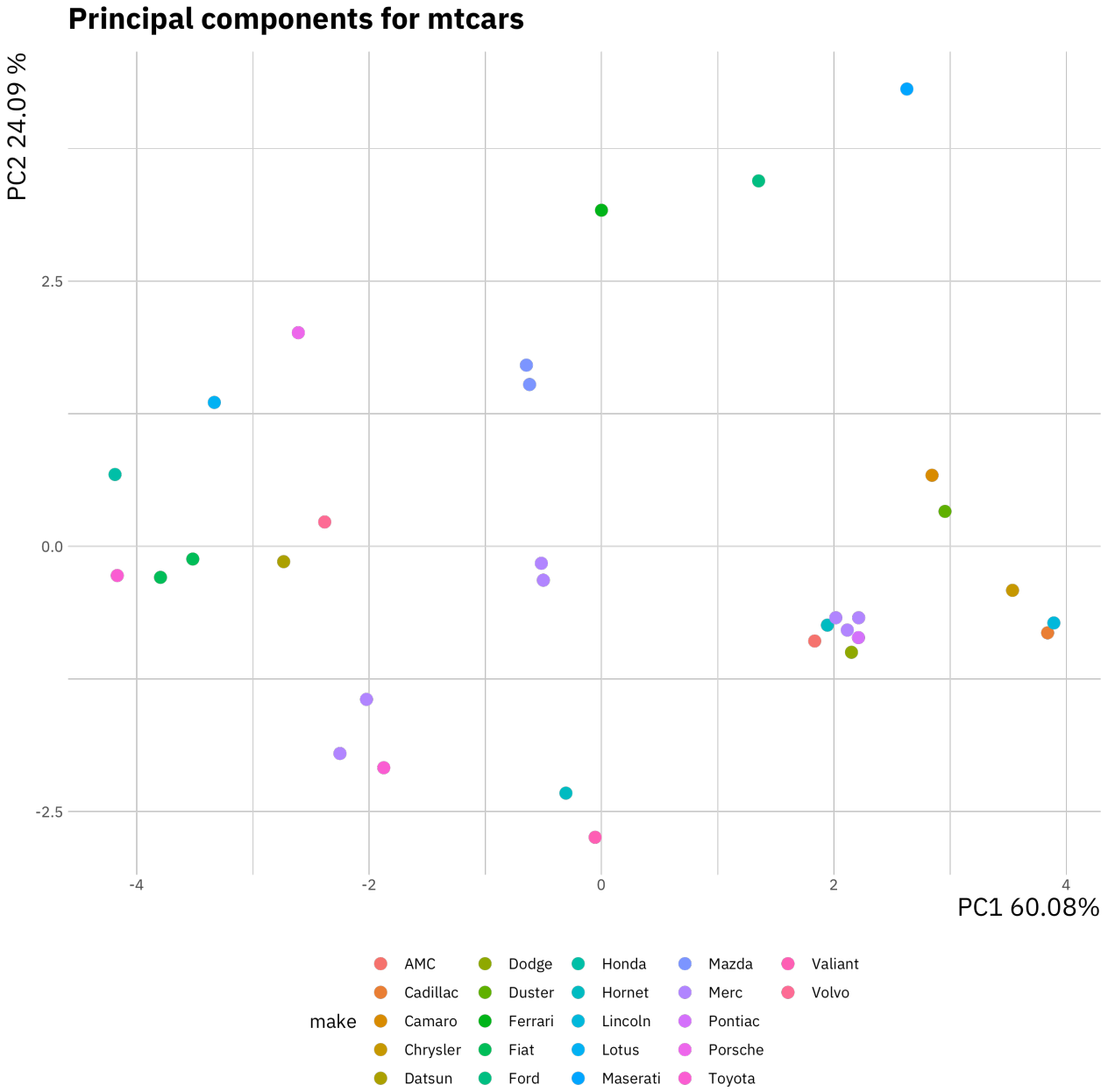
```
pca_summary <- summary(cars_pca)
print(pca_summary)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     2.5707 1.6280 0.79196 0.51923 0.47271 0.46000
## Proportion of Variance 0.6008 0.2409 0.05702 0.02451 0.02031 0.01924
## Cumulative Proportion  0.6008 0.8417 0.89873 0.92324 0.94356 0.96279
##                           PC7     PC8    PC9    PC10   PC11
## Standard deviation     0.3678 0.35057 0.2776 0.22811 0.1485
## Proportion of Variance 0.0123 0.01117 0.0070 0.00473 0.0020
## Cumulative Proportion  0.9751 0.98626 0.9933 0.99800 1.0000
```

Note, `Proportion of Variance` will always add up to 1. Here the `PC1` explain 60.08 of the variance, and `PC2` explains 24.09, which means together `PC1` and `PC2` account for 84.17 of the variance.

Using the `str()` function we can see the full structure of an `R` object, or alternatively using `?prcomp` in the *Value* section. In this case the `cars_pca` variable is a list containing several variables, `x` is the data represented using the new principal components. We can now plot the data in the first two principal components:

```r
pca_df <- data.frame(cars_pca$x, make = stringr::word(rownames(mtcars), 1))

ggplot(pca_df, aes(x = PC1, y = PC2, col = make)) +
geom_point(size = 3) +
labs(x = "PC1 60.08%",
     y = "PC2 24.09 %",
     title = "Principal components for mtcars") +
theme(legend.position = "bottom")
```



Here we added a color based on the make of each car. We can observe which samples (or cars) cluster together. Have a look at these variables and decide why certain cars or models would cluster together.

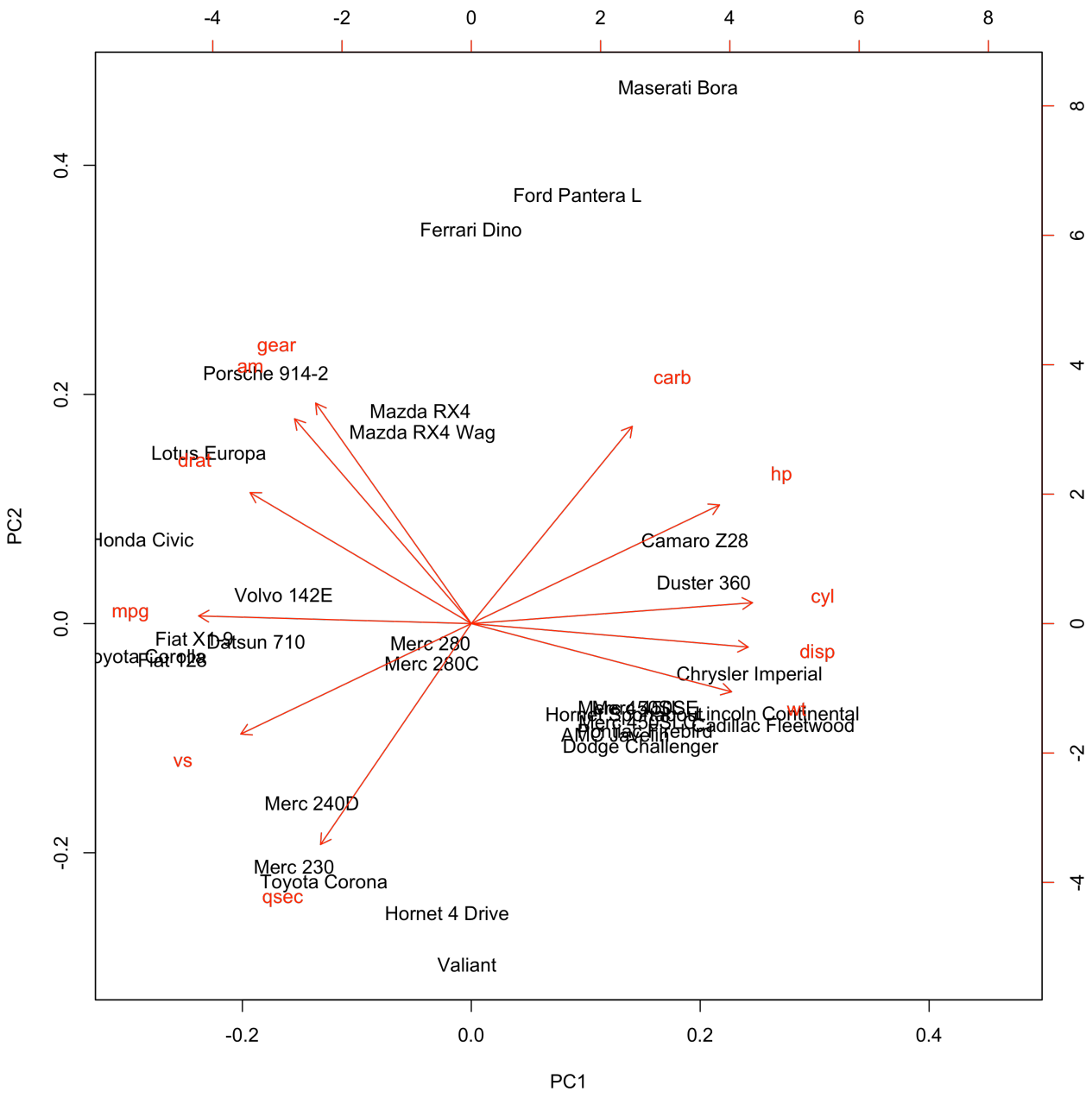
We created this plot using the `ggplot2` package, it is also possible to do this using base plot if you prefer.

```
plot(pca_df$PC1, pca_df$PC2)
```

3.4 Exercise II

Next we look at another representation of the data, the *biplot*. This is a combination of a PCA plot of the data and a *score plot*. We saw the PCA plot in the previous section in a *biplot* we add the original axis as arrows.

```
biplot(cars_pca)
```



We can see the original axis starting from the origin. Therefore we can make observations about the original variables (e.g. `cyl` and `mpg` contribute to PC1) and how the data points relates to these axes.

3.5 Exercise III

Now try to perform a PCA on the `USArrests` data also build into `R`. Typing `?USArrests` will give you further information on the data. Perform the analysis on the subset `USArrests[, -3]` data.

3.6 Exercise IV: Single cell data

We can now try to apply what we learned above on a more realistic datasets. You can download the data either on *canvas* or using these links `Pollen2014.txt` and `SupplementaryLabels.txt`. Her we will be dealing with single cell RNA-Seq (scRNA-Seq) data, which consist of 300 single cells measured across 8686 genes.

```
pollen_df <-read.table("Pollen2014.txt", sep=',', header = T,row.names=1)

label_df <-read.table("SupplementaryLabels.txt", sep=',', header = T)

pollen_df[1:10, 1:6]
```

```
##          Cell_2338_1 Cell_2338_10 Cell_2338_11 Cell_2338_12 Cell_2338_13
## MTND2P28          78          559          811          705          384
## MTATP6P1        2053         1958         4922         4409         2610
## NOC2L              1          125          126            0          487
## ISG15           2953         4938          580          523         2609
## CPSF3L             2           42           19            0           37
## MXRA8              0            0            0            0            0
## AURKAIP1         302          132           64          492           11
## CCNL2              0          235            0           84           13
## MRPL20           330          477          288          222           44
## SSU72            604          869         2046          158          530
##          Cell_2338_14
## MTND2P28          447
## MTATP6P1         3709
## NOC2L              66
## ISG15               1
## CPSF3L             12
## MXRA8               0
## AURKAIP1          182
## CCNL2              11
## MRPL20            282
## SSU72             272
```

```
dim(pollen_df)
```

```
## [1] 8686  300
```

Measurements of scRNA-Seq data are integer counts, this data does not have good properties so we perform a transformation on the data. The most commonly used transformation on RNA-Seq count data is $\log_2$. We will also transpose the data matrix to rows representing cells and columns representing genes. This is the data we can use to perform PCA.

```
# scRNA-Seq data transformation
pollen_mat <- log2(as.matrix(pollen_df) + 1)
# transpose the data
pollen_mat <- t(pollen_mat)
```

We will now use information that we read into the `label_df` variable to rename cells.

```r
# Check which columns we have available
colnames(label_df)
```

```
## [1] "Cell_Identifier"        "Population"
## [3] "Cell_names"             "TrueLabel_CellLevel"
## [5] "Tissue_name"            "TrueLabel_TissuelLevel"
```

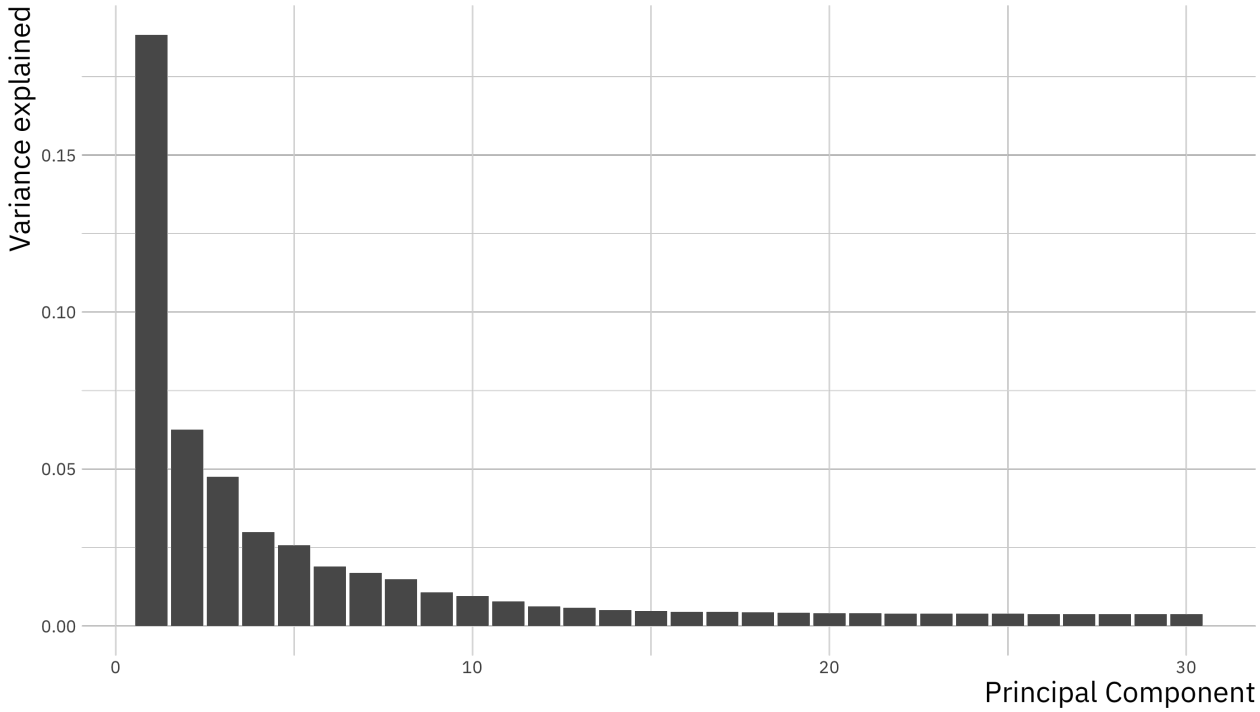
```r
# rename rows
rownames(pollen_mat) <- label_df$Cell_names
```

Next we perform PCA on the data and extract the proportion of variance explained by each component.

```r
sc_pca <- prcomp(pollen_mat)

# variance is the square of the standard deviation
pr_var <- sc_pca$sdev^2

# compute the variance explained by each principal component
prop_var_exp <- pr_var / sum(pr_var)
```

Think about the calculation and what exactly it means. We can visualise this

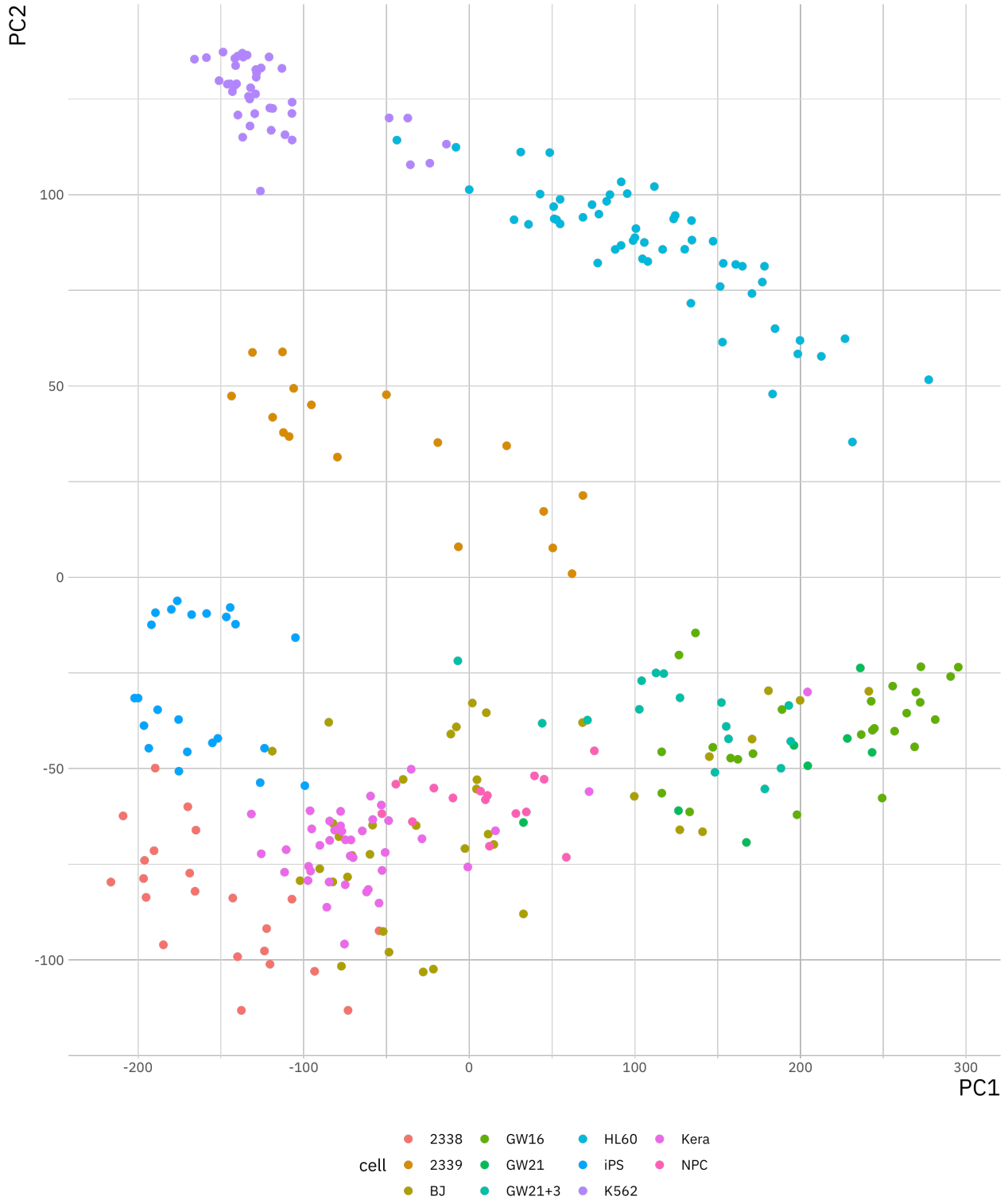
```r
var_exp <- data.frame(variance = prop_var_exp, pc = 1:length(prop_var_exp))

ggplot(var_exp[1:30, ], aes(x = pc, y = variance)) +
    geom_bar(stat = "identity") +
    labs(x = "Principal Component",
         y  = "Variance explained")
```



We see that the first few principal components explain significant variance, but after about the PC10, there is very limited contribution. Next we will plot the data using the first two Principal components as before.

```r
sc_pca_df <- data.frame(sc_pca$x, cell = rownames(sc_pca$x),
                        var_exp = prop_var_exp)
```

```
ggplot(sc_pca_df, aes(x = PC1, y = PC2, col = cell)) +
    geom_point(size = 2) +
    theme(legend.position = "bottom")
```



Why is it not useful to create biplot for this example?

4 Practical: Multiple regression

Previously we have only considered simple linear regression with one response variable and one feature. In this practical we will go through examples with multiple features:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

For this practical we will use data that is already inbuilt in R or is part of the `MASS` package. The only thing we need to do to make the data available is load the `MASS` package.

4.1 Multiple regression

For this part we will use the inbuilt `trees` dataset containing `Volume`, `Girth` and `Height` data for 31 trees.

First we revisit linear regression on this example, recall the function to fit a linear model `lm()`. Consider `Volume` to be the response variable and `Girth` to be the covariate.

```
lr_fit <- lm(Volume ~ Girth, data = trees)
summary(lr_fit)
```

```
##
## Call:
## lm(formula = Volume ~ Girth, data = trees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.065 -3.107  0.152  3.495  9.587
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -36.9435     3.3651  -10.98 7.62e-12 ***
## Girth         5.0659     0.2474   20.48  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.252 on 29 degrees of freedom
## Multiple R-squared:  0.9353, Adjusted R-squared:  0.9331
## F-statistic: 419.4 on 1 and 29 DF,  p-value: < 2.2e-16
```

We will now consider a linear regression example with multiple covariates, `Girth` as well as `Height`. In this case of course we know that they are related so we do expect both covariates to be significant.

```
mr_fit <- lm(Volume ~ Girth + Height, data = trees)


summary(mr_fit)
```

```
##
## Call:
## lm(formula = Volume ~ Girth + Height, data = trees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4065 -2.6493 -0.2876  2.2003  8.4847
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -57.9877     8.6382  -6.713 2.75e-07 ***
## Girth         4.7082     0.2643  17.816  < 2e-16 ***
## Height        0.3393     0.1302   2.607   0.0145 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.882 on 28 degrees of freedom
## Multiple R-squared:  0.948,  Adjusted R-squared:  0.9442
## F-statistic:   255 on 2 and 28 DF,  p-value: < 2.2e-16
```

Note, in the formula you only enter the covariates and not the regression coefficients or any information regarding the noise.

Let us now look at RSS values, we can calculate the RSS for the `lf_fit` object by using `sum(residuals(lr_fit)^2)`. We see that the RSS for LR $= 524.3$ and the RSS for MR $= 421.92$. Therefore the fit has improved but the regression coefficient for `Height` is very small and not significant.

One reason for this is that the in the relationship between `Volume`, `Girth`, and `Height` is not additive but rather `Girth` and `Height` are multiplied. Using the fact that $\log(a * b) = \log(a) + \log(b)$ we can consider the log-transformed data in a linear model.

```
mrl_fit <- lm(log(Volume) ~ log(Girth) + log(Height), data = trees)

summary(mrl_fit)
```

```
##
## Call:
## lm(formula = log(Volume) ~ log(Girth) + log(Height), data = trees)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.168561 -0.048488  0.002431  0.063637  0.129223
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.63162    0.79979  -8.292 5.06e-09 ***
## log(Girth)   1.98265    0.07501  26.432  < 2e-16 ***
## log(Height)  1.11712    0.20444   5.464 7.81e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08139 on 28 degrees of freedom
## Multiple R-squared:  0.9777, Adjusted R-squared:  0.9761
## F-statistic: 613.2 on 2 and 28 DF,  p-value: < 2.2e-16
```

Now we see that the regression coefficient is large and both covariates are significant. This shows that we need to ensure we understand the relationship between covariates before we construct our model.

4.2 Categorical covariates

Recall from the lecture that covariates don't need to be numerical but can also be *categorical*. We will now explore regression with a categorical variable. Load a new dataset which is included in the `MASS` package, you won't be able to load this dataset if package isn't installed. Load the dataset explore what the data looks like.

```r
library(MASS)
data("birthwt")

head(birthwt)
```

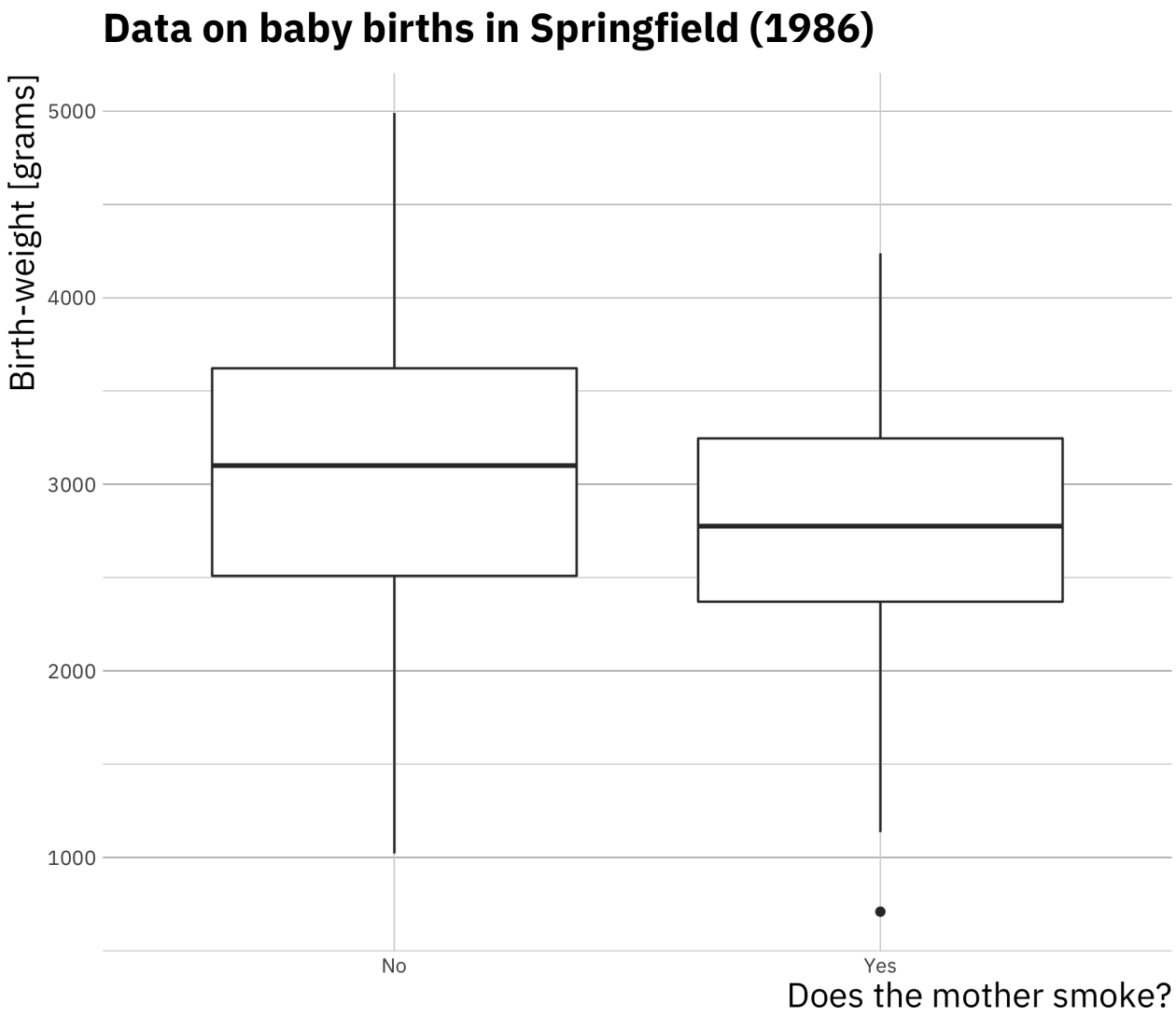
```
##    low age lwt race smoke ptl ht ui ftv  bwt
## 85   0  19 182    2     0   0  0  1   0 2523
## 86   0  33 155    3     0   0  0  0   3 2551
## 87   0  20 105    1     1   0  0  0   1 2557
## 88   0  21 108    1     1   0  0  1   2 2594
## 89   0  18 107    1     1   0  0  1   0 2600
## 91   0  21 124    3     0   0  0  0   0 2622
```

```r
summary(birthwt)
```

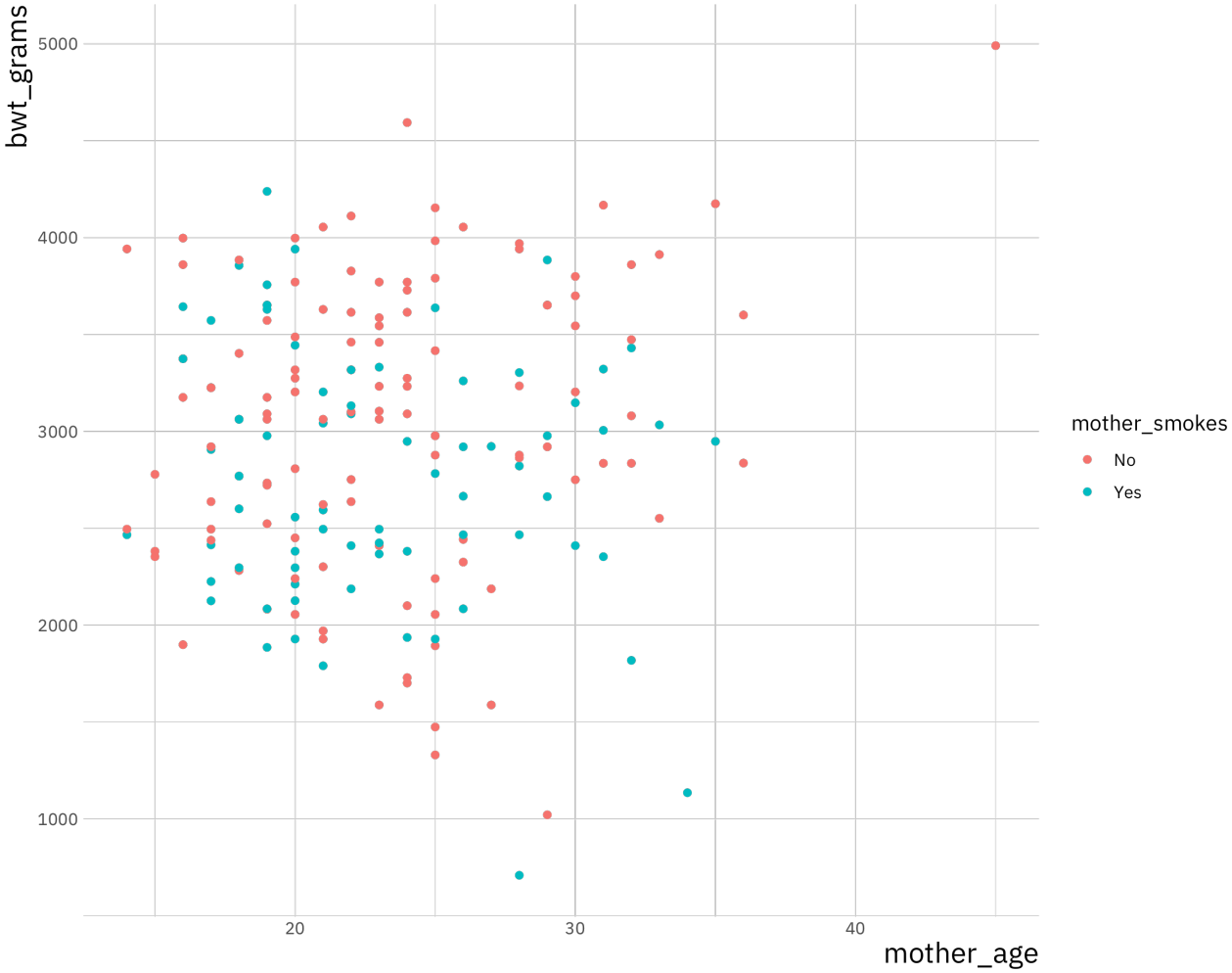
```
##       low              age              lwt             race      
##  Min.   :0.0000   Min.   :14.00   Min.   : 80.0   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:19.00   1st Qu.:110.0   1st Qu.:1.000  
##  Median :0.0000   Median :23.00   Median :121.0   Median :1.000  
##  Mean   :0.3122   Mean   :23.24   Mean   :129.8   Mean   :1.847  
##  3rd Qu.:1.0000   3rd Qu.:26.00   3rd Qu.:140.0   3rd Qu.:3.000  
##  Max.   :1.0000   Max.   :45.00   Max.   :250.0   Max.   :3.000  
##      smoke            ptl               ht                ui        
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.0000   Median :0.0000   Median :0.00000   Median :0.0000  
##  Mean   :0.3915   Mean   :0.1958   Mean   :0.06349   Mean   :0.1481  
##  3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.0000  
##  Max.   :1.0000   Max.   :3.0000   Max.   :1.00000   Max.   :1.0000  
##       ftv              bwt      
##  Min.   :0.0000   Min.   : 709  
##  1st Qu.:0.0000   1st Qu.:2414  
##  Median :0.0000   Median :2977  
##  Mean   :0.7937   Mean   :2945  
##  3rd Qu.:1.0000   3rd Qu.:3487  
##  Max.   :6.0000   Max.   :4990
```

We will give the data more interpretable names and generally cleanup the data a little bit.

```r
# rename columns
colnames(birthwt) <- c("bwt_below_2500", "mother_age", "mother_weight", "race",
                       "mother_smokes", "previous_prem_labor", "hypertension",
                       "uterine_irr", "physician_visits", "bwt_grams")

birthwt$race <- factor(c("white", "black", "other")[birthwt$race])
birthwt$mother_smokes <- factor(c("No", "Yes")[birthwt$mother_smokes + 1])
birthwt$uterine_irr <- factor(c("No", "Yes")[birthwt$uterine_irr + 1])
birthwt$hypertension <- factor(c("No", "Yes")[birthwt$hypertension + 1])

ggplot(birthwt, aes(x = mother_smokes, y = bwt_grams)) +
    geom_boxplot() +
    labs(title = "Data on baby births in Springfield (1986)",
         x = "Does the mother smoke?",
         y = "Birth-weight [grams]")
```




```
ggplot(birthwt, aes(x = mother_age, y = bwt_grams, col = mother_smokes)) +
    geom_point()
```



Now we perform linear regression using the categorical variable, it is no different than performing linear regression on numeric data. The difference is in interpretation.

```
bwt_fit <- lm(bwt_grams ~ mother_smokes, data = birthwt)

summary(bwt_fit)
```

```
##
## Call:
## lm(formula = bwt_grams ~ mother_smokes, data = birthwt)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2062.9  -475.9    34.3   545.1  1934.3
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       3055.70      66.93  45.653  < 2e-16 ***
## mother_smokesYes  -283.78     106.97  -2.653  0.00867 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 717.8 on 187 degrees of freedom
```

```
## Multiple R-squared:  0.03627,    Adjusted R-squared:  0.03112
## F-statistic: 7.038 on 1 and 187 DF,  p-value: 0.008667
```

When you put a categorical variable in the formula for `lm` as in this case `bwt_grams ~ mother_smokes` where we have two levels in the categorical variable. If we consider this model as $y = \beta_0 + \beta_1 x + \epsilon$ The coefficients in the model can be interpreted as follows:

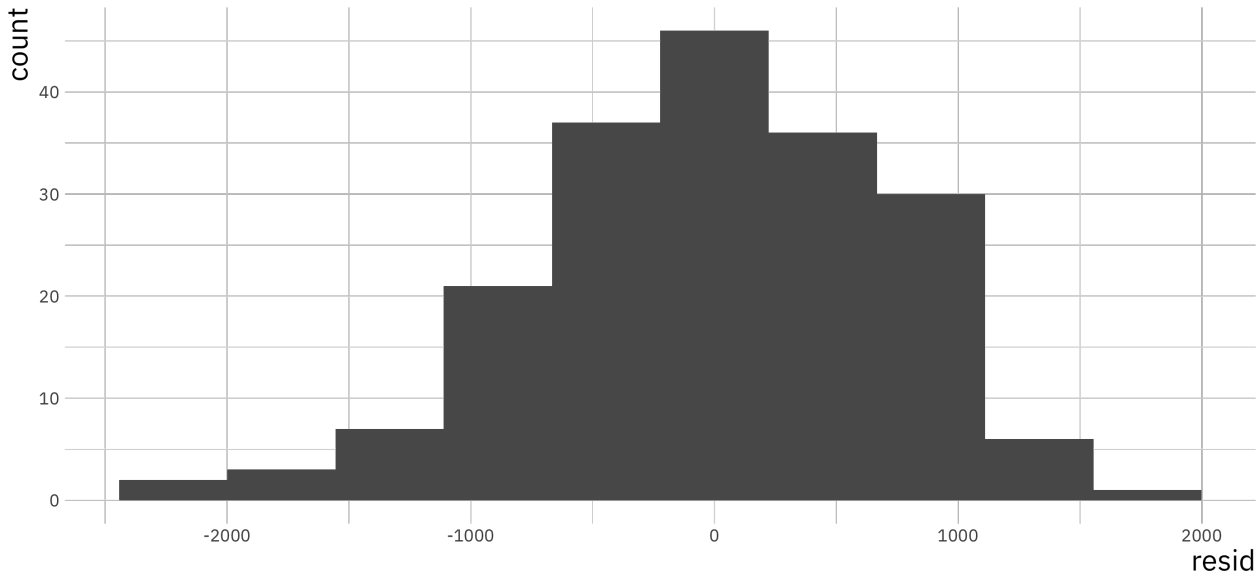
- β_0 is average birth weight where the mother was a non smoker
- $\beta_0 + \beta_1$ is the average birth weight where the mother is a smoker
- β_1 is the average difference in birth weight for babies between mother that were smokers and mothers that were non smokers.

Categorical variables can also have more than two levels and in those cases each additional level can be interpreted in the same way.

4.3 Residuals

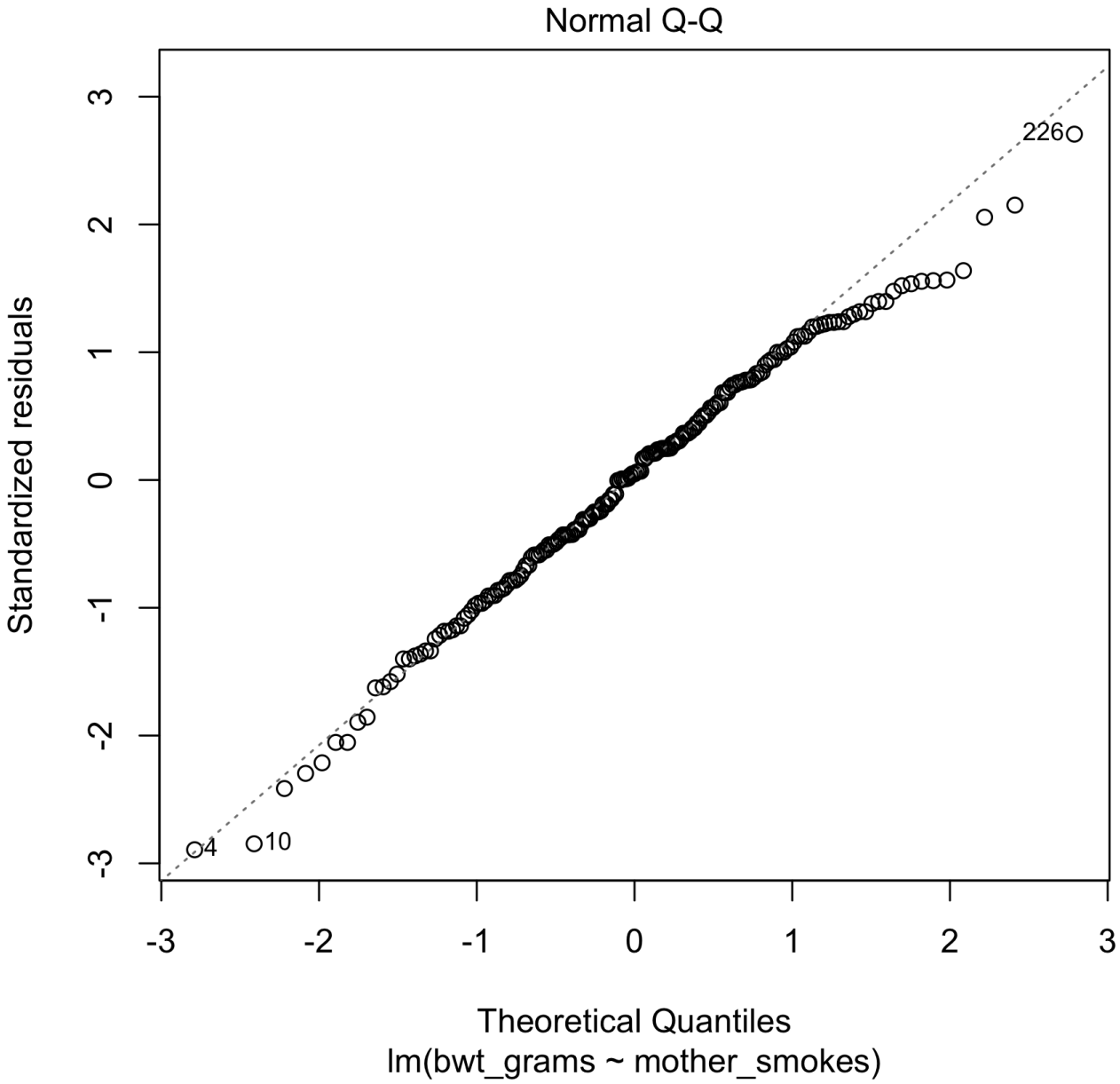
Recall from the lectures the residuals are the differences between the observed data y and the fitted values $\hat{y}$. One of the assumptions we make in the simple linear regression model is that the residuals should be normally distributed. To extract residuals from an `lm` object we will use the `residuals()` function.

```
residuals_df <-
data.frame(resid = residuals(bwt_fit))

ggplot(residuals_df, aes(x = resid)) +
    geom_histogram(bins = 10)
```



Even if we consider that these residuals look like they are normally distributed we need to get better understanding of this we will use the Q-Q Plot. You can take a look at the wiki to get a better understanding (Q–Q plot - Wikipedia). In simple terms if the residuals are normally distributed we expect them to be on the diagonal straight line on a Q-Q plot. The simplest way to get such a plot is using the `plot()` function and specifically for an `lm` object it has an option `which =` that takes a numeric value depending on which plot you want to plot.

```
plot(bwt_fit, which = 2)
```



As we can see in this example the residuals are very close to normal with some outliers especially towards larger values of the residual. This would indicate that the model as it stands does not fulfill that assumption fully but comes close.

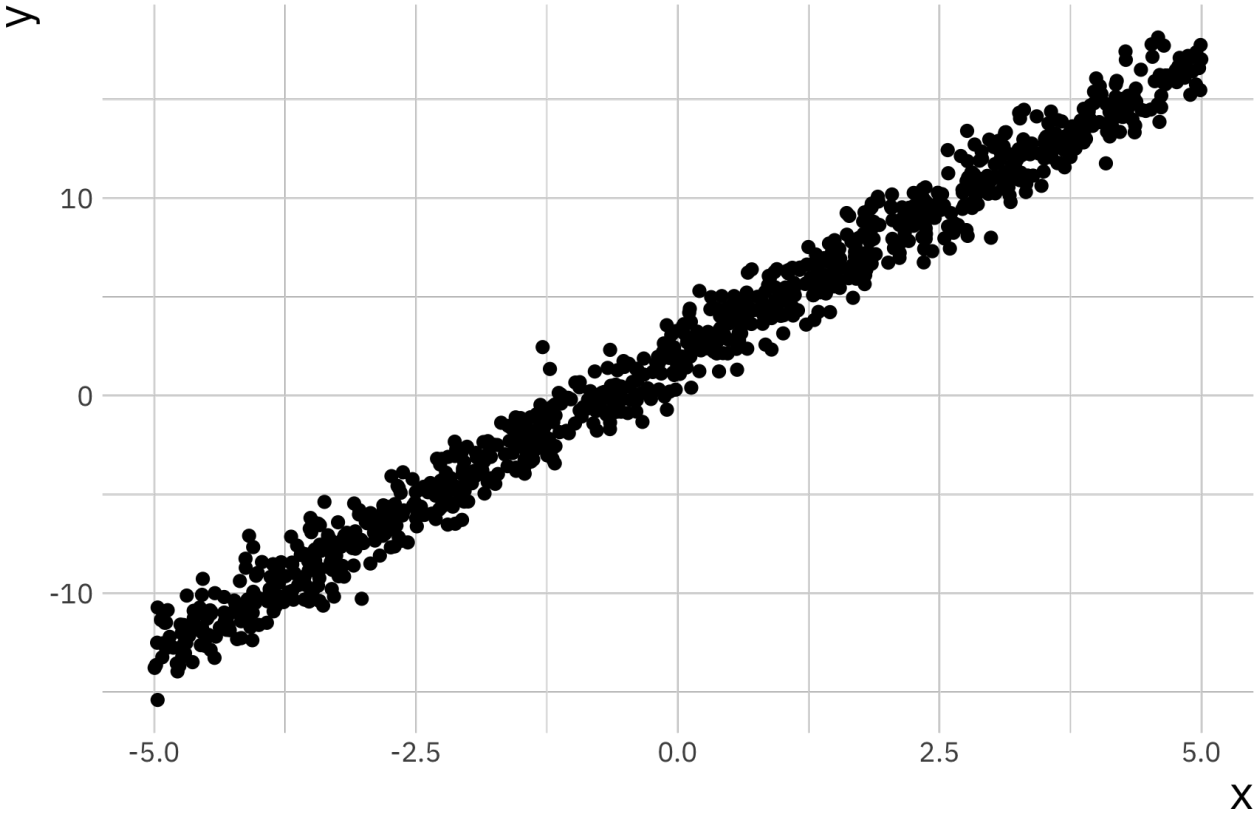
4.4 Gradient descent algorithm (+)

Finally, in todays practical we will implement the *gradient descent algorithm* which we discussed in the lecture.

For simplicity we will only consider the case with one covariate. In this section we will use simulated data and compare the results with `lm()`. The model we will simulate from is:

$$y = 2 + 3x + \epsilon$$

```r
# setting seed to be able to reproduce the simulation
set.seed(200)

# number of samples
n_sample <- 1000

# We sample x values from a uniform distribution in the range [-5, 5]
x <- runif(n_sample, -5, 5)
# Next we compute y
y <- 3 * x + 2 + rnorm(n = n_sample, mean = 0, sd = 1)

sim_df <- data.frame(x = x, y = y)

ggplot(sim_df, aes(x = x, y = y)) +
    geom_point()
```



Recall that in gradient descent we want to minimise the Mean Squared Error ($J(\beta)$) which is the cost function. The first step is to write this cost function in R. For simplicity we will use matrix multiplication, which in R is implemented as `%*%`. (*Note*, to get help on these function with special characters you can't simply run the command `?%*%` instead you have to put it in quotes `?"%*%"`.)

```r
cost_fn <- function(X, y, coef) {
    sum( (X %*% coef - y)^2 ) / (2*length(y))
}
```

To perform an optimisation we will have to initialise parameters, in general optimisation algorithms won't always produce the same results for all choices of initialisations.

```
# First we set alpha and the number of iterations we will perform
alpha <- 0.2
num_iters <- 100

# next we will initialise regression coefficients
coef <- matrix(c(0,0), nrow=2)
X <- cbind(1, matrix(x))
res <- vector("list", num_iters)
```

We now write a for loop to compute the optimisation, where we store the full history of the opmtimisation.

```
for (i in 1:num_iters) {
  error <- (X %*% coef - y)
  delta <- t(X) %*% error / length(y)
  coef <- coef - alpha * delta
  res_df <- data.frame(itr = i , cost = cost_fn(X, y, coef),
                   b0 = coef[1], b1 = coef[2])

  res[[i]] <- res_df
}
```

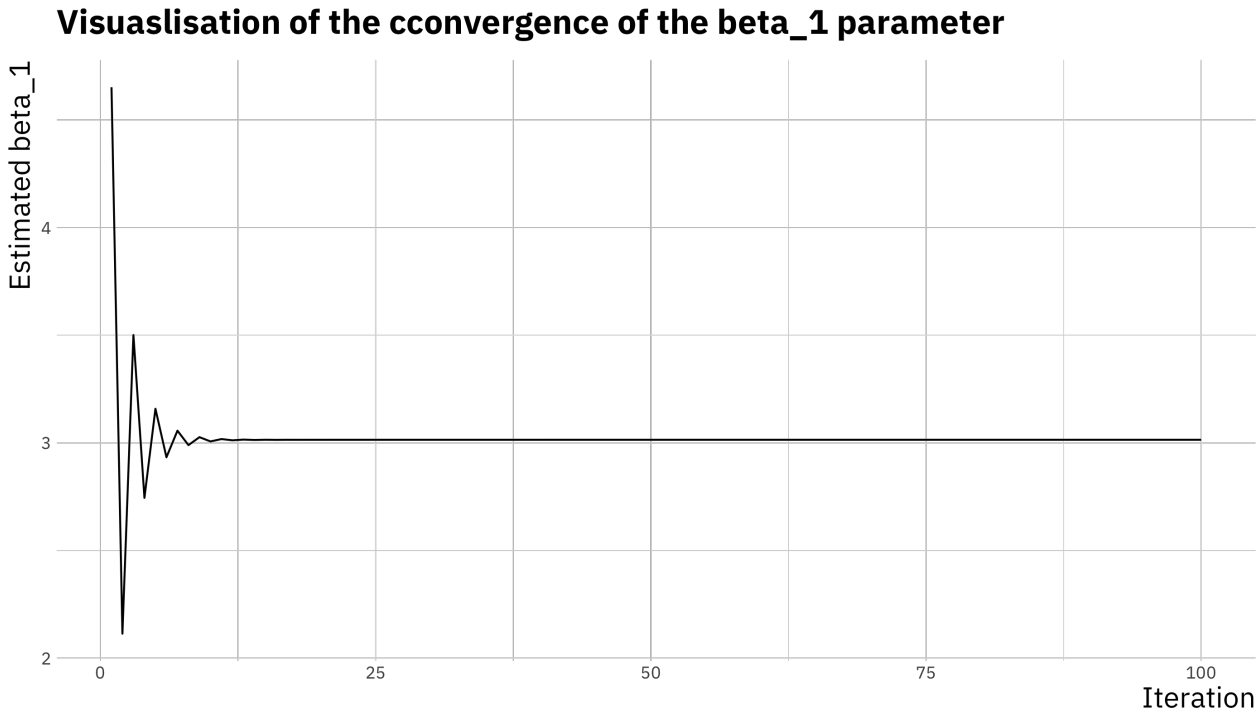
We created a list to store results `res` it is possible to combine all results into a simple `data.frame` using the `bind_rows()` function from the `dplyr` package. If we look at the final values in the resulting variable we will

```
library(dplyr)
res_df <- bind_rows(res)
tail(res_df)
```

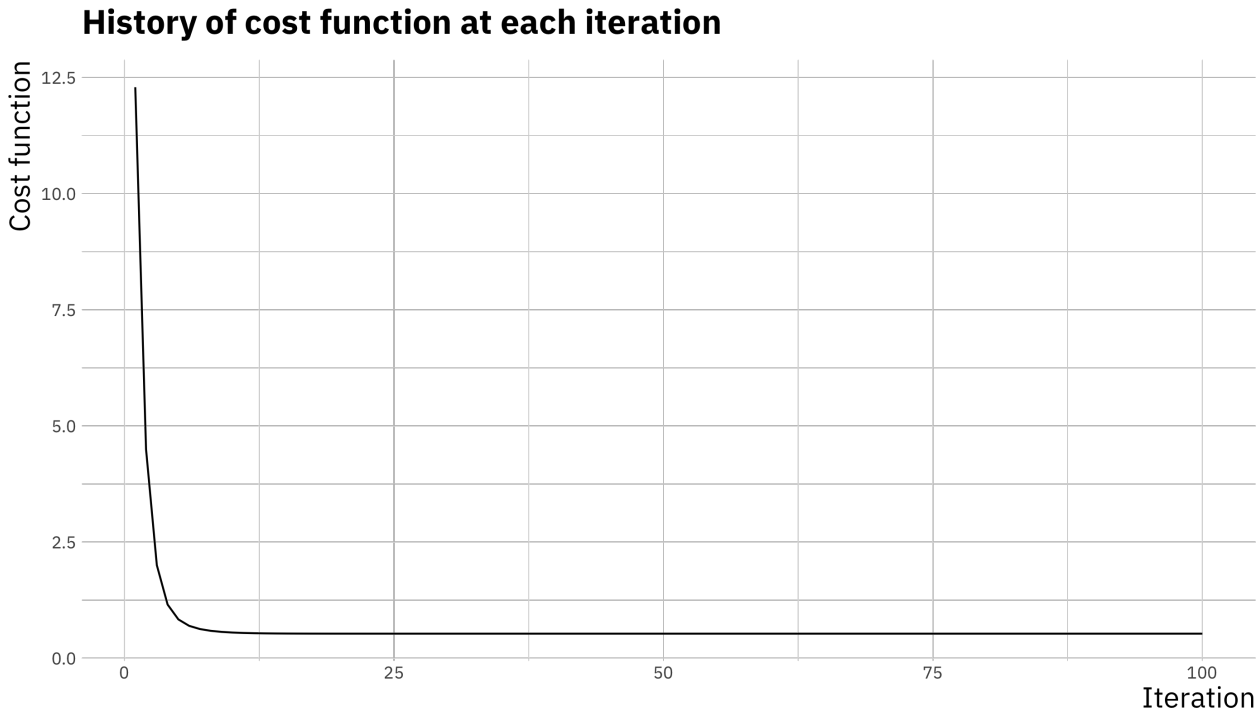
```
##     itr      cost       b0       b1
## 95   95 0.5275707 2.034285 3.014512
## 96   96 0.5275707 2.034285 3.014512
## 97   97 0.5275707 2.034285 3.014512
## 98   98 0.5275707 2.034285 3.014512
## 99   99 0.5275707 2.034285 3.014512
## 100 100 0.5275707 2.034285 3.014512
```

We can see that $\beta_0 = 2$ and $\beta_1 = 3$ are reproduced faithfully. There are a few ways to visualise the optimisation. We can look at the convergence of the parameters, the cost function itself or even the estimated y at each step of the optimisation.

```
ggplot(res_df, aes(x = itr, y = b1)) +
    geom_line() +
    labs(x = "Iteration",
         y = "Estimated beta_1",
         title = "Visuaslisation of the cconvergence of the beta_1 parameter")
```

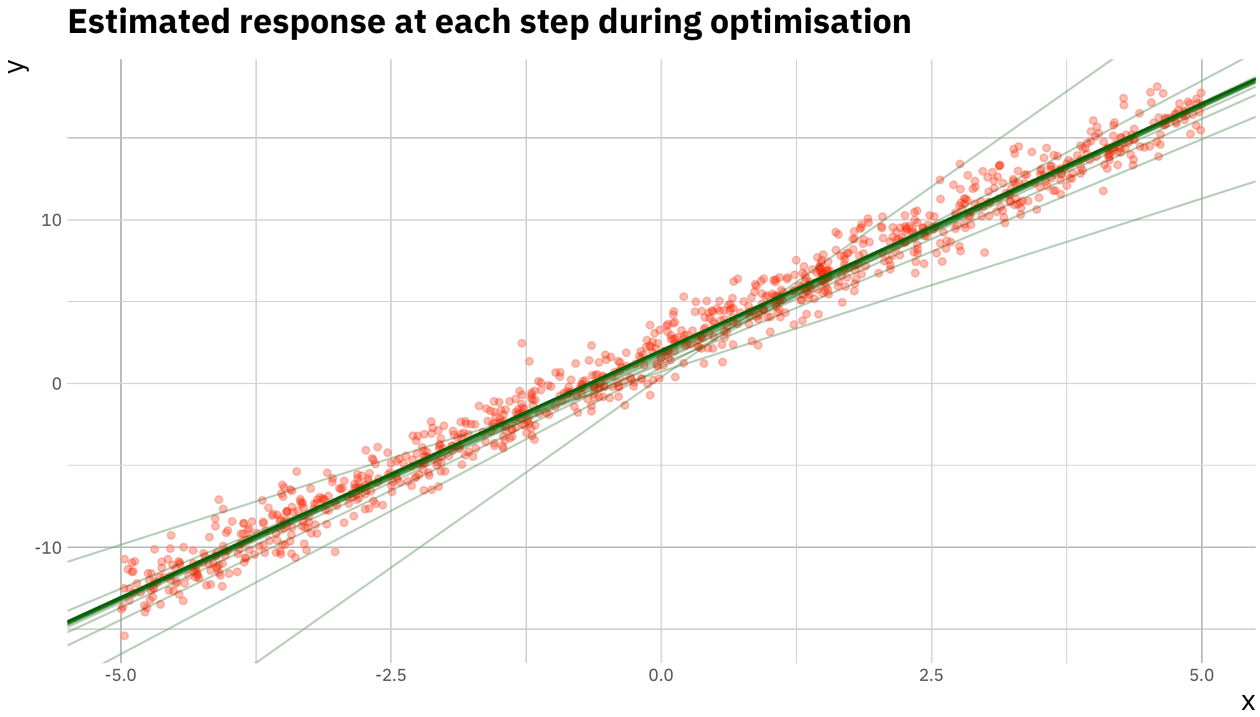



```
ggplot(res_df, aes(x = itr, y = cost)) +
    geom_line() +
    labs(x = "Iteration",
         y = "Cost function",
         title = "History of cost function at each iteration")
```




```
ggplot(sim_df, aes(x = x, y = y)) +
    geom_point(color = "red", alpha = 0.3) +
    geom_abline(data = res_df, aes(intercept = b0, slope = b1),
```

```
              alpha = 0.3, col = "darkgreen", size = 0.5) +
  labs(x = "x", y = "y",
       title = "Estimated response at each step during optimisation")
```



Now compare these results to the ones obtained by fitting a linear model in R using the function `lm()`, how different are the results. Try to reproduce these plots with $\alpha = (0.02, 0.1, 0.5)$, and different number of iterations in the optimisation and compare the estimated $\hat{\beta}_0$, and $\hat{\beta}_1$ to the values you use during the simulation step. This will give you an idea how important the right choice of these two parameters is.

5 Practical: Generalised linear models

In a genome-wide association study, we perform an experiment where we select n individuals with a disease (cases) and n individuals without the diseases (controls) and look for genetic differences between these two groups. In particular, we are interested in specific genetic variants (SNPs) that might induce some predisposition towards the disease.

Suppose I observe the following genotypes for a SNP in 4,000 individuals (2,000 cases, 2,000 controls):

- Genotypes: `AA Aa aa`
- Controls: `3 209 1788`
- Cases: `83 621 1296`

The cases seem to have relatively more `A` alleles than the controls. This might make us suspect that having `A` alleles at this SNP is associated with the disease.

5.1 Data

For this practical we will use two files you can use these links to download them:

- `gwas-cc-ex1.Rdata` (download)
- `gwas-cc-ex2.Rdata` (download)

5.2 Detecting SNP associations

We have seen in lectures that we can do statistical tests for this type of contingency table using Chi Squared Tests. Let's load example data set and and prepare

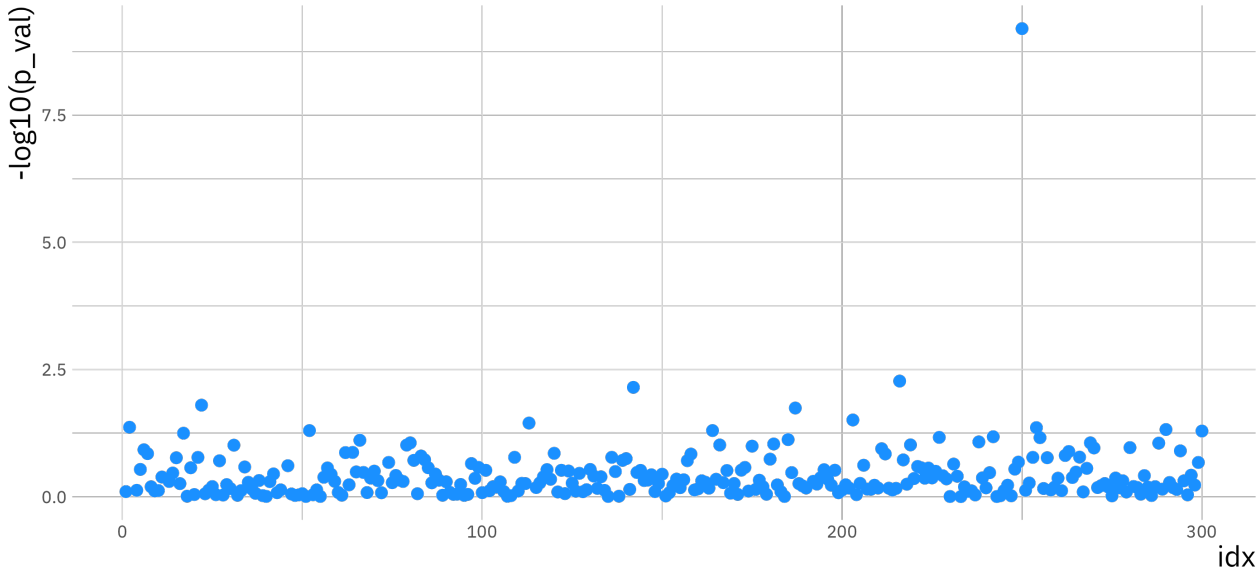
```
library(ggplot2) # for plots later
load("gwas-cc-ex1.Rdata")

# how many individuals are there
n <- length(y)
# How many SNPs do we have data for
p <- nrow(X)

# samples that are controls are encoded as 0 in y
control <- which(y == 0)
# disease cases are encoded as 1 in y
cases <- which(y == 1)
```

Now we need to write a loop that scans through all, p, SNPs:

```
# create a vector where p-values will be stored
p_vals <- rep(0, p)

# Loop over SNPs
for (i_p in 1:p) {
    # 1. obtain genotype counts
    counts <- matrix(0, nrow = 2, ncol = 3)
    counts[1, ] <- c(sum(X[i_p, control] == 0),
                     sum(X[i_p, control] == 1),
                     sum(X[i_p, control] == 2))

    counts[2, ] <- c(sum(X[i_p, cases] == 0),
                     sum(X[i_p, cases] == 1),
                     sum(X[i_p, cases] == 2))

    # 2. expected probability of AA
    # (assuming no dependence on case/control status)
    expected_pr_AA <- sum(counts[, 1]) / n
    # expected probability of Aa
    expected_pr_Aa <- sum(counts[, 2]) / n
    # expected probability of aa
    expected_pr_aa <- sum(counts[, 3]) / n

    expected_probs <- c(expected_pr_AA, expected_pr_Aa, expected_pr_aa )

    # 3. do my chi-squared test
    out <- chisq.test(counts, p = expected_probs)
    # extract p value of test and store
    p_vals[i_p] <- out$p.value
}
```

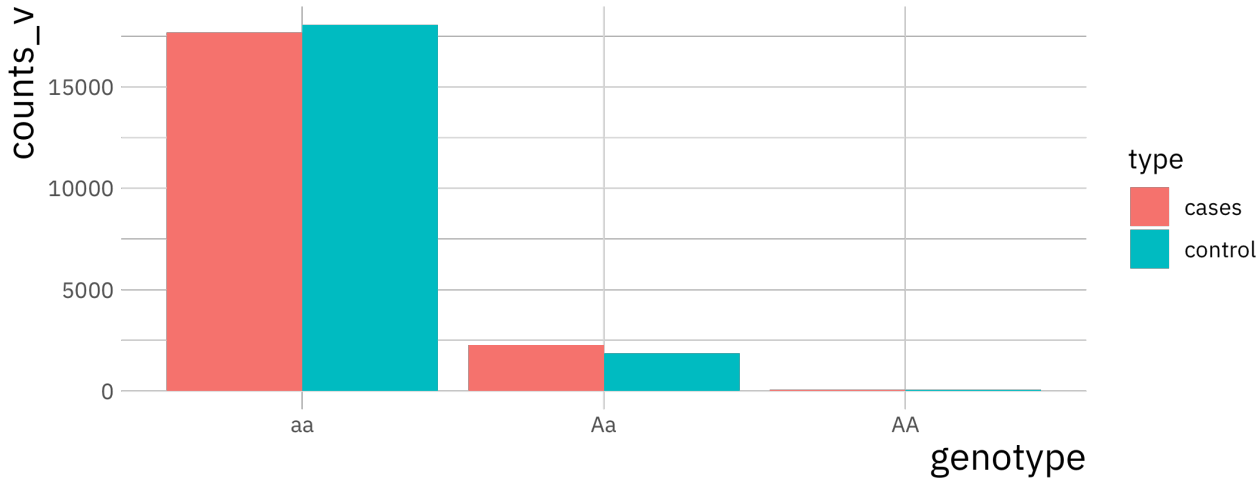
We went through each SNP (rows in matrix `X`), extracted the counts of each genotype (marked `1.` in code) for cases and controls, then we compute expected probability (marked `2.` in code). Finally, we perform a chi-squared contingency table test comparing those observed counts to expected probabilities assuming that genotype is not related to disease status (marked `3.` in code).

```r
p_val_df <- data.frame(p_val = p_vals, idx = 1:p)

ggplot(p_val_df, aes(x = idx, y = -log10(p_val))) +
    geom_point(size = 2.5, col = "dodgerblue1")
```



This plot is knows as a *Manhattan* plot. One SNP (`i_p = 250`) will pop out as being highly associated with the disease process. Look at the genotype counts (or MAF) for this SNP in the cases and controls to see for yourself that there is large difference in the distribution of genotypes (or MAF).

```r
i_p <- 250
counts_v <- c(sum(X[i_p, control] == 0), sum(X[i_p, control] == 1),
              sum(X[i_p, control] == 2), sum(X[i_p, cases] == 0),
              sum(X[i_p, cases] == 1), sum(X[i_p, cases] == 2))

snp_procs <- data.frame(counts_v, type = rep(c("control", "cases"), each = 3),
           genotype = rep(c("AA", "Aa", "aa"), 2))

ggplot(snp_procs, aes(x = genotype, y = counts_v, fill = type)) +
    geom_bar(stat = "identity", position = "dodge")
```



5.3 GWAS and logistic regression

Now lets approach this problem using Generalised Linear Models. Lets load a data set containing genotypes in `X` and case-control status in `y`:

```
# load an example data set (genotypes in X, case-control (1/0) status in y)
load("gwas-cc-ex2.Rdata")

n <- length(y) # how many individuals do we have in total?
p <- nrow(X) # how many SNPs do I have data for?
```

For each of the `p` SNPs we are going to call the `R` GLM function `glm` using the `binomial` family option with the `logit` link function because my outcomes are binary. We will then extract the p-value associated with the regression coefficient for the genotype. This is obtained from applying a hypothesis test (the *Wald Test*) on whether the coefficient has a null value zero.

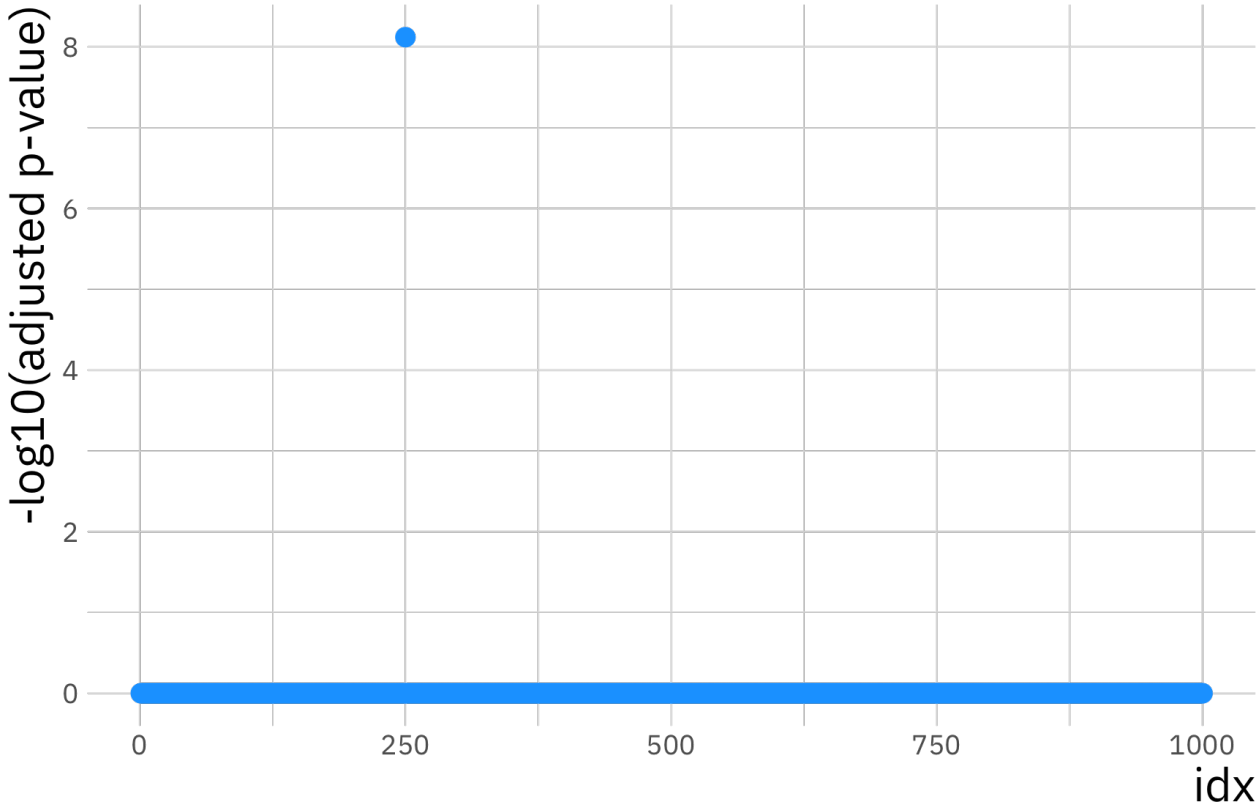
```
p_vals <- rep(0, p)
for ( j in 1:p ) {
  snp_data <- data.frame(y = y, x = X[j, ])
  glm.fit <- glm(y ~ x, family = binomial(link = logit), data = snp_data  )
  p_vals[j] <- summary(glm.fit)$coefficients[2,4]
}
```

We are testing 1,000 SNPs so lets use Bonferroni correction to adjust these p-values to take into account multiple testing:

```
adj_p_vals <- p.adjust(p_vals, "bonferroni")
```

Lets use the adjusted -log10 p-values to plot a Manhattan plot:

```
# create data.frame with p-values for plotting with ggplot
p_val_df <- data.frame(p_val = adj_p_vals, idx = 1:p)

ggplot(p_val_df, aes(x = idx, y = -log10(p_val))) +
    geom_point(size = 2.5, col = "dodgerblue1") +
    labs(y = "-log10(adjusted p-value)")
```



You should see a single SNP showing a strong association with disease status.

5.4 Negative binomial and Poisson regression

Molecular biologists study the behavior of protein expression in normal and cancerous tissues. The hypothesis is that the total number of over-expressed proteins depends on the histopathological-derived tumor subtype and an immune cell contexture measure.

You are provided with data on 314 tumours in the file `nb_data.Rdata`. The file contains one `data frame` with the following variables:

- **`overexpressed_proteins`**: response variable of interest.
- **`immunoscore`**: gives a standardized measure of immune cell contexture.
- **`tumor_subtype`**: three-level nominal variable indicating the histopathological sub-type of the tumour. The three levels are Unstable, Stable, and Complex

Let's load some prerequisite R libraries and the data to produce some summary statistics (*install if required using `install.package()` command*):

```
# required libraries
library(MASS)
library(foreign)


load("nb_data.Rdata")

# print summary statistics to Console
summary(dat)
```

```
##    sample_id      gender     immunoscore    overexpressed_proteins
##  1001   :  1   female:160   Min.   : 1.00   Min.   : 0.000
```

```
##  1002   :  1   male  :154   1st Qu.:28.00   1st Qu.: 1.000
##  1003   :  1                Median :48.00   Median : 4.000
##  1004   :  1                Mean   :48.27   Mean   : 5.955
##  1005   :  1                3rd Qu.:70.00   3rd Qu.: 8.000
##  1006   :  1                Max.   :99.00   Max.   :35.000
##  (Other):308
##   tumor_subtype
##  Complex : 40
##  Unstable:167
##  Stable  :107
##
##
##
##
```

5.4.1 Count-based GLMs

The `overexpressed_proteins` measurements are counts. This implies we should use a Poisson based GLM.

In Poisson regression models, the conditional variance is by definition equal to the conditional mean. This can be limiting.

Negative binomial regression can be used for over-dispersed count data, that is when the conditional variance exceeds the conditional mean.

It can be considered as a generalization of Poisson regression since it has the same mean structure as Poisson regression but it has an extra parameter to model the over-dispersion. If the conditional distribution of the outcome variable is over-dispersed, the confidence intervals for the Poisson regression are likely to be narrower as compared to those from a Negative Binomial regression model.

In the following we will try both models to see which fits best.

5.4.2 Fitting a GLM

Below we use the `glm.nb` function from the `MASS` package to estimate a negative binomial regression. The use of the function is similar to that of `lm` for linear models but with the additional requirement of a link function. As count data is always positive, a log link function is useful here.

```
glm_1 <- glm.nb(overexpressed_proteins ~ immunoscore + tumor_subtype + gender, data = dat, link=log)

# print summary statistics of glm.nb output object to Console
summary(glm_1)
```

```
##
## Call:
## glm.nb(formula = overexpressed_proteins ~ immunoscore + tumor_subtype +
##     gender, data = dat, link = log, init.theta = 1.047288915)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1567  -1.0761  -0.3810   0.2856   2.7235
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)              2.707484   0.204275  13.254  < 2e-16 ***
## immunoscore             -0.006236   0.002492  -2.502   0.0124 *
## tumor_subtypeUnstable -0.424540   0.181725  -2.336   0.0195 *
```

```
## tumor_subtypeStable   -1.252615   0.199699  -6.273 3.55e-10 ***
## gendermale            -0.211086   0.121989  -1.730   0.0836 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(1.0473) family taken to be 1)
##
##     Null deviance: 431.67  on 313  degrees of freedom
## Residual deviance: 358.87  on 309  degrees of freedom
## AIC: 1740.3
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  1.047
##           Std. Err.:  0.108
##
##  2 x log-likelihood:  -1728.307
```

R first displays the call and the deviance residuals. Next, we see the regression coefficients for each of the variables, along with standard errors, z-scores, and p-values. The variable `immunoscore` has a coefficient of -0.006, which is statistically significant at the 5% level (`Pr(>|z|) = 0.0124*`). This means that for each one-unit increase in `immunoscore`, the expected log count of the number of `overexpressed_proteins` decreases by 0.006.

The indicator variable shown as `tumor_subtypeUnstable` is the expected difference in log count between group Unstable and the reference group (`tumor_subtype=Complex`). The expected log count for the Unstable type is approximately 0.4 lower than the expected log count for the Complex type.

The indicator variable for Stable type is the expected difference in log count between the Stable type and the reference Complex group. The expected log count for Stable is approximately 1.2 lower than the expected log count for the Complex type.

5.4.3 Comparing nested models

To determine if `tumor_subtype` itself, overall, is statistically significant, we can compare a model with and without `tumor_subtype`. The reason it is important to fit separate models is that, unless we do, the overdispersion parameter is held constant and it would not be a fair comparison.

```
glm_2 <- glm.nb(overexpressed_proteins ~ immunoscore + gender, data = dat, link = log)
```

We use the `anova` function to compare models using a likelihood ratio test (LRT):

```
anova(glm_1, glm_2, test = "LRT")
```

```
## Warning in anova.negbin(glm_1, glm_2, test = "LRT"): only Chi-squared LR
## tests are implemented
```

```
## Likelihood ratio tests of Negative Binomial Models
##
## Response: overexpressed_proteins
##                                   Model     theta Resid. df    2 x log-lik.
## 1                 immunoscore + gender 0.8705939       311       -1772.074
## 2 immunoscore + tumor_subtype + gender 1.0472889       309       -1728.307
##     Test    df LR stat.      Pr(Chi)
## 1
## 2 1 vs 2     2 43.76737 3.133546e-10
```

The two degree-of-freedom chi-square test indicates that `tumor_subtype` is a statistically significant predictor of `overexpressed_proteins` (`Pr(Chi) = 3.133546e-10`).

The `anova` function performs a form of *LRT*. It computes the likelihood of the data under the two models being compared and then uses the ration of these likelihood values as a test statistic.

Theory tells us that, for large samples sizes, the (2x) log likelihood ratio has a chi-squared distribution with degrees of freedom equal to the difference in the number of free parameters between the two models being compared. The LRT only applies to *nested models*, i.e. a pair of models where one is a less complex subset of the other.

5.5 Negative-Binomial vs Poisson GLMs

Negative binomial models assume the conditional means are not equal to the conditional variances. This inequality is captured by estimating a dispersion parameter (not shown in the output) that is held constant in a Poisson model. Thus, the Poisson model is actually nested in the negative binomial model. We can then use a likelihood ratio test to compare these two models.

To do this, we will first fit a GLM Poisson regression:

```
glm_3 <- glm(overexpressed_proteins ~ immunoscore + tumor_subtype + gender, family = "poisson", data = d
```

Now, lets do our likelihood ratio test, we can extract the log-likelihood using `logLik()` and then use `pchisq()` to extract the probability of getting a statistic at least as extreme as this:

```
pchisq(2 * (logLik(glm_1) - logLik(glm_3)), df = 1, lower.tail = FALSE)
```

```
## 'log Lik.' 3.847622e-198 (df=6)
```

Note that the more complex model goes first because more complex models always have the larger likelihood.

In this example the associated chi-squared value estimated from `2*(logLik(m1) - logLik(m3))` is around 900 with one degree of freedom. This strongly suggests the negative binomial model, estimating the dispersion parameter, is more appropriate than the Poisson model.

5.6 Further understanding the model (OPTIONAL)

For assistance in further understanding the model, we can look at predicted counts for various levels of our predictors. Below we create new datasets with values of `immunoscore` and `tumor_subtype` and then use the predict command to calculate the predicted number of overexpressed proteins

First, we can look at predicted counts for each value of `tumor_subtype` while holding `immunoscore` at its mean. To do this, we create a new dataset with the combinations of `tumor_subtype` and `immunoscore` for which we would like to find predicted values, then use the `predict()` command.

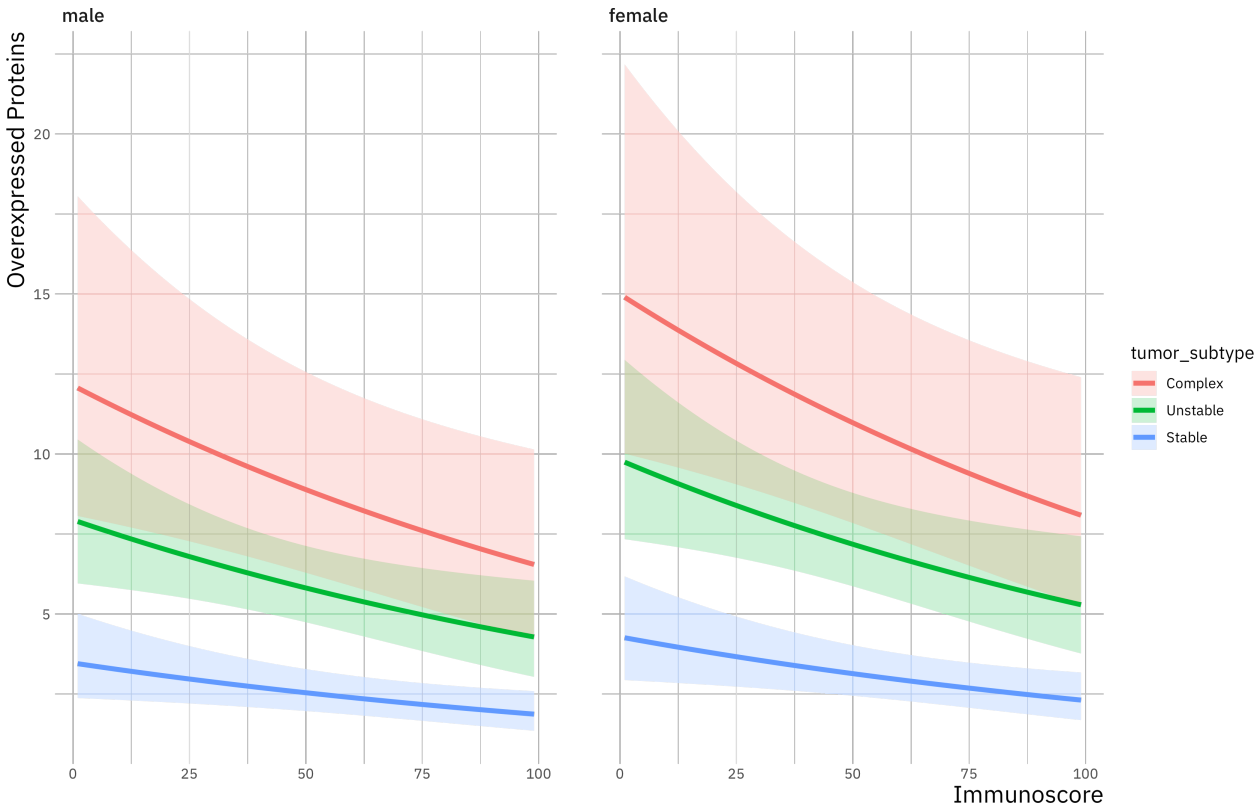
```
newdata_1 <-
data.frame(
    immunoscore = mean(dat$immunoscore),
    tumor_subtype = factor(c("Complex", "Unstable", "Stable"), labels = levels(dat$tumor_subtype)),
    gender="male")

newdata_2 <-
data.frame(
    immunoscore = mean(dat$immunoscore),
    tumor_subtype = factor(c("Complex", "Unstable", "Stable"), labels = levels(dat$tumor_subtype)),
    gender="female")

new_data <- rbind(newdata_1, newdata_2)
```

```r
new_data$phat <- predict(glm_1, new_data, type = "response")

print(new_data)
```

```
##   immunoscore tumor_subtype gender      phat
## 1    48.26752       Complex   male  8.983829
## 2    48.26752        Stable   male  2.567187
## 3    48.26752      Unstable   male  5.876060
## 4    48.26752       Complex female 11.095193
## 5    48.26752        Stable female  3.170523
## 6    48.26752      Unstable female  7.257042
```

```r
newdata_3 <-
data.frame(
    immunoscore = rep(seq(from = min(dat$immunoscore), to = max(dat$immunoscore), length.out = 100), 3),
    tumor_subtype = rep(factor(c("Complex", "Unstable", "Stable"), labels = levels(dat$tumor_subtype)),
    gender="male")

newdata_4 <-
data.frame(
    immunoscore = rep(seq(from = min(dat$immunoscore), to = max(dat$immunoscore), length.out = 100), 3),
    tumor_subtype = rep(factor(c("Complex", "Unstable", "Stable"), labels = levels(dat$tumor_subtype)),
    gender="female")

new_data <- rbind(newdata_3, newdata_4)

new_data <- cbind(new_data, predict(glm_1, new_data, type = "link", se.fit=TRUE))

new_data <- within(new_data, {
  overexpressed_proteins <- exp(fit)
  LL <- exp(fit - 1.96 * se.fit)
  UL <- exp(fit + 1.96 * se.fit)
})
```

```r
library(ggplot2)

ggplot(new_data, aes(immunoscore, overexpressed_proteins)) +
    geom_ribbon(aes(ymin = LL, ymax = UL, fill = tumor_subtype), alpha = 0.2) +
    geom_line(aes(colour = tumor_subtype), size = 1.5) +
    labs(x = "Immunoscore",
         y = "Overexpressed Proteins") +
    facet_wrap(~ gender)
```


male
female
Overexpressed Proteins
20
15
10
5
0
25
50
75
100
Immunoscore
tumor_subtype
Complex
Unstable
Stable