Data Preparation, Processing and Reporting with R

QFAB Bioinformatics

Contents

1 Introduction 5

1.1 How to read this book . . . 5

1.2 Getting started . . . 6

1.3 A quick refresher . . . 6

2 Data Preparation 7

2.1 Data Lifecycle & Management Plan . . . 7

2.2 Good Data Protocol . . . 8

2.3 Further Reading . . . 8

3 Reproducible reporting in R 9

3.1 Literate programming . . . 9

3.2 Open a new knitR document . . . 9

4 Importing and reformatting data files 11

4.1 Importing data . . . 11

4.2 Reformat the file . . . 12

4.3 Transform data . . . 12

4.4 Normalise data . . . 13

4.5 Export data . . . 14

5 Data manipulation - extracting and exporting records 15

5.1 Removing samples . . . 15

5.2 Remove uninformative measurements . . . 16

5.3 Extracting specific measurements . . . 16

6 Integrating datasets 17

6.1 Combining two files with identical row identifiers . . . 17

6.2 Combining two files with rows in common . . . 18

7 Charts and graphical output 21

7.1 Refresher - standard R plotting functions . . . 21

7.2 ggplot2 . . . 26

8 Generating report with R 33

8.1 Other reporting formats . . . 34

8.2 Interactive reporting with Shiny . . . 34

9 Conclusion 35

Appendix: Exercise answers 37

4 Importing and reformatting data files . . . 37

5.1 Data manipulation . . . 39

5.2 An alternative practice data set . . . 41

6 Integrating and summarising datasets . . . 42

6.2 Summarising data . . . 44

7 Charts and graphical output . . . 46

7.2 ggplot2 51

Appendix - SessionInfo for this version of the course material **53**

Chapter 1

Introduction

Welcome to QFAB's **Data prepration, processing and reporting with R** workshop. This manual contains all the exercises that you will do during this workshop. There will be hints, tips and examples along the way.

The format of the workshop will be that we will introduce a topic with some examples, and then set you some practical exercises to carry out. We hope that you will be able to complete these exercises using just the information that you have learnt in the workshop in combination with the R help functions. If you do get stuck though, you will find example answers to all of them at the back of this manual. Please use these examples as a guide and reference rather than just a solution: don't just copy the code, but think about what each line is doing and see if you can come up with a better way.

1.1 How to read this book

In the shaded dialog boxes with the thick left border are commands that should be typed into your R *Console* window. This corresponds to **input**, the **output** will sometimes follow below with lines preceeded with two hashmarks (##).

```
print("HELLO WORLD")
## [1] "HELLO WORLD"
```

Tips, Suggestions and Traps

Comments and salient advice are provided in dialogs such as this. We'll use this format to warn you of traps that you could slip into and to provide some hints.

Tips, Suggestions and Traps

Warnings are provided in dialogs such as this. We'll use this format to warn you of traps that you could slip into and to provide some hints.

Time for you to make R work

Exercises are the best way to learn. A document can provide an insight to the process but hands-on interaction with R is the only way to learn. In the exercise boxes are some suggested exercises that will apply the knowledge that is being shared.

1.2 Getting started

Open an RStudio server client

1. Log onto the server URL that is provided to you by the trainer with the username and password
2. Create a New Project that you will use for this course using the **File** > **New Project** in RStudio. To keep things tidy, use the option to create this project in a new directory (e.g. "Data_processing")
3. Execute the following 2 commands and afterwards check that you have a new folder called "figure" and inside this folder there is a "logo.png" after copying the file.

```
dir.create('figure')
file.copy("../data/Data_Interpretation/logo.png","figure/logo.png")
```

Remember if you ever forget where your **working directory** is, follow the steps:

- In the **Files** tab, click on **More** > **Go to Working Directory**

1.3 A quick refresher

If you haven't used R for a while, or are a relatively new R user, then take a moment to remind yourself about the various data types in R. Try creating an example of the different types of structure using various data types:

- Numeric
- Integer
- Logical
- Character
- Complex

And the structures in which those data types can be stored:

- Vector
- Matrix
- List
- Data frame
- Factors (not really a structure!)

Chapter 2

Data Preparation

Before we can process and analyze data files in R to generate a report, we first need to collect, collate, and prepare the data for R.

How you set up and define your data is critical to the future value of that data. Implementing a good data management strategy need not be an onerous task; it may simply require the adoption of a number of standard practices by members of a research group, and can have many benefits, including:

- Better usability and so increased value of data within a research group, along with reduced training time for new staff and less risk of loss of key knowledge when staff leave
- Easier error checking and correction, as well as avoidance of misinterpretation caused by ambiguous or unclear data
- Lower time and financial cost of analysis: well-formatted data is simpler and easier to import into analysis software, and also simplifies engagement with core analysis facilities and external service providers
- If data meets with community standards, it is easier to submit it to public resources to support research publications. Researchers who share their data have been shown to be cited more often In addition to these benefits, funding bodies such as the ARC now require a data management plan and researchers and their data may be subject to regulatory and institutional obligations, such as the **Code for the Responsible Conduct of Research**.

2.1 Data Lifecycle & Management Plan

A Data Management Plan is a document that describes how you will conduct the Data Lifecycle for your project. This would include:

- **Collecting data**
  - design research
  - plan data management (formats, storage etc)
  - plan consent for sharing
  - locate existing data
  - collect data (experiment, observe, measure, simulate)
  - capture and create metadata
- **Processing data**
  - enter data, digitise, transcribe, translate
  - check, validate, clean data
  - anonymise data where necessary
  - describe data
  - manage and store data
- **Preserving data**
  - migrate data to best format
  - migrate data to suitable medium
  - back-up and store data

  - create metadata and documentation
  - archive data
- **Giving Access to data**
  - distribute data
  - share data
  - control Access
  - establish copyright
  - promote data
- **Re-using data**
  - follow-up research
  - new research
  - undertake research reviews
  - scrutinise findings
  - teach and learn

2.2 Good Data Protocol

In preparing your data for analysis in R you will want to consider using the following approaches:

- Data Log
- Archiving or Version control
- Data Dictionary (Meta Data)
- Data Monitoring and Safety
- Data Cleaning and Preprocessing
- Data Formatting
- Summary Graphs & Stats

Today's workshop will focus on the last two steps: Data Formatting and Summary Graphs. For more information on the remainder, the two supporting documents can be found at the following links:

- QFAB Good Data Guidelines
- QFAB Data Specs

2.3 Further Reading

- Key Data Concepts
- Wickham, H.. (2014) *Tidy Data.* JStatSoft
- Hulley S. B. et al. (2013) *Designing Clinical Research.* Ch16-17
- McFadden E.. (2007) Management of Data in Clinical Trials

Chapter 3

Reproducible reporting in R

3.1 Literate programming

Literate programming is a computing approach where the programming code is provided as subsections in a document that is mostly written in a natural language. This creates an environment where the documentation is the main component of the program, rather than an afterthought.

The text is typically written in some sort of 'mark-up' format, with sections highlighted as title, emphasised or italicised and so on. The code itself can be run on demand in combination with this marked-up text, to produce a report or other document combining the pre-defined text in the literate program with the output of the code.

In terms of using R literate programming techniques for bioinformatics, what this allows us to do is to generate a reusable report template that can be repeatedly run as each lot of data becomes available. The output may be a pdf file suitable for printing or e-mailing, html code for direct display through a web browser, or a number of other formats.

For this workshop, we will use the knitR format, which is built directly into the RStudio platform and generates an html report at the click of a button.

3.2 Open a new knitR document

1. Under the RStudio **File** menu, select **New File** > **R Notebook**.
2. In the first line next to **Title**, replace the *"R Notebook"* with an appropriate title.
3. Save the file.
4. If you are not already familiar with the knitR Markdown syntax, click on **Help** > **Markdown Quick Reference**.
5. Once you have made some changes, save it using the disk icon, then click on the **Preview** button to generate and preview your HTML report. As we are using an Rstudio version, a window may pop up, click on **Try again** to enable the preview window to appear. Alternatively, if you click on the little arrow next to the Preview button, you can choose to what format you want to knit (render) your notebook: PDF, HTML or even WORD.

Tip: to turn of preivew mode but still knit your document, click on the **cogwheel** > **(No Preview)**

6. Try some of the Markdown syntax shown in the *Markdown Quick Reference*.

For the rest of this workshop, try to incorporate all the exercises that we do, along with comments and notes, into an RNotebook (or other knitR documents). You may want to create new documents for different chapters or exercises. The goal is that at the end of the workshop, you will be able to produce a final, reproducible report of all the analyses that you have done.

Chapter 4

Importing and reformatting data files

4.1 Importing data

R provides a variety of tools for reading in data from a file. One of the more useful ones for bioinformatics data is the read.table function, which we will use to read in the first test dataset. This file contains microarray gene expression results from the widely-used Golub et al. (1999) paper. This data set contains calls from around 7,000 gene probes on samples from 38 leukemia patients. Twenty seven patients are diagnosed as having acute lymphoblastic leukemia (ALL) and eleven as having acute myeloid leukemia (AML).

Read in the datafile

The data file is in the folder `/data/Data_interpretation_with_R` and is called **data_set_ALL_AML_train.tsv**.

Note: Remember the location of the data file uses a **relative path** to find the dataset, so it depends on the current directory from which you are running the Markdown document. If you are running the document from the newly created Data_Workshop folder, then you need to use the relative path `../data/Data_Interpretation/data_set_ALL_AML_train.tsv` to read the file. This will go up one directory using the `..` annotation for parent directory.

1. Use the `head()` and `readLines()` functions to look at the structure of the file
2. With the read.table function, read the file contents into a data frame
   - Use the second column (labelled **Gene Accession Number**, actually the microarray probe ID) as the row names

```
## [1] "Gene Description\tGene Accession
Number\t1\tcall\t2\tcall\t3\tcall\t4\tcall\t5\tcall\t6\tcall\t7\tcall\t8\tcall\t9\tcall\t10\tcall\t1
## [2] "AFFX-BioB-5_at (endogenous
control)\tAFFX-BioB-5_at\t-214\tA\t-139\tA\t-76\tA\t-135\tA\t-106\tA\t-138\tA\t-72\tA\t-413\tA\t5\tA
## [3] "AFFX-BioB-M_at (endogenous
control)\tAFFX-BioB-M_at\t-153\tA\t-73\tA\t-49\tA\t-114\tA\t-125\tA\t-85\tA\t-144\tA\t-260\tA\t-127\
## [4] "AFFX-BioB-3_at (endogenous
control)\tAFFX-BioB-3_at\t-58\tA\t-1\tA\t-307\tA\t265\tA\t-76\tA\t215\tA\t238\tA\t7\tA\t106\tA\t42\t
## [5] "AFFX-BioC-5_at (endogenous
control)\tAFFX-BioC-5_at\t88\tA\t283\tA\t309\tA\t12\tA\t168\tA\t71\tA\t55\tA\t-2\tA\t268\tA\t219\tM\
## [6] "AFFX-BioC-3_at (endogenous
control)\tAFFX-BioC-3_at\t-295\tA\t-264\tA\t-376\tA\t-419\tA\t-230\tA\t-272\tA\t-399\tA\t-541\tA\t-2
```

4.2 Reformat the file

We now have a data frame containing the following:

Column number	Contents
Row names	Microarray probe ID
1	Gene names/descriptions
2, 4, 6…	(up to 54) Intensity calls for 27 ALL patients
56, 58, 60…	(up to 76) Intensity calls for 11 AML patients
3, 5, 7… (up to 77)	Status calls for the intensity values

To make this data easier to work with, we want to transform this data frame into a matrix containing just the intensity calls. We will also need to create an object to store the gene names associated with the probe identifiers; this could be a vector of gene names with probe IDs as item names, a two column matrix (probe ID, gene name), or even a list, whichever you prefer.

1. Use a matrix indexing ([row,col]) to create a matrix of just the intensity values from the `golub` data frame

Hint:

```
* use all the even numbered columns
* as an example of a slice, `data[, c(2,4,6)]` will output all rows for
columns 2, 4 and 6 of the matrix `data`
* find another function that will create a sequence of even numbers instead of
manually typing them
```

2. Create a vector, matrix or other object to link gene names with probe identifiers

We should now have two objects: (i) the numerical data in one matrix and (ii) the gene name information in whatever format we chose to use. The next stage is to transform and normalise the data

Below are the first 5 rows and 5 columns of the `golub.matrix`:

```
##                  ALL  ALL  ALL  ALL  ALL
## AFFX-BioB-5_at -214 -139  -76 -135 -106
## AFFX-BioB-M_at -153  -73  -49 -114 -125
## AFFX-BioB-3_at  -58   -1 -307  265  -76
## AFFX-BioC-5_at   88  283  309   12  168
## AFFX-BioC-3_at -295 -264 -376 -419 -230
```

Below is the first 6 items of the gene information (`golub.names`):

```
##                          AFFX-BioB-5_at
##  "AFFX-BioB-5_at (endogenous control)"
##                          AFFX-BioB-M_at
##  "AFFX-BioB-M_at (endogenous control)"
##                          AFFX-BioB-3_at
##  "AFFX-BioB-3_at (endogenous control)"
##                          AFFX-BioC-5_at
##  "AFFX-BioC-5_at (endogenous control)"
##                          AFFX-BioC-3_at
##  "AFFX-BioC-3_at (endogenous control)"
##                         AFFX-BioDn-5_at
## "AFFX-BioDn-5_at (endogenous control)"
```

4.3 Transform data

Our data is now in a structured format suitable for further filtering and processing. The first step that we will carry out will be to apply a floor and ceiling to the values in the dataset. Microarrays, like many other measurement platforms in biology, may give inaccurate readings at the high and low end of their operating range. In this case, any values of less than 100 or more than 16,000 are likely to be unreliable.

Also, because expression values are typically spread over an extreme range (in this case, around 2.5 orders of magnitude), it is conventional to log transform them to convert the data into a more linear and simple to analyse form.

Often in transformation steps, it is good practice to create a new data object to store the transformed values. Remember to appropriately name the data objects so that you remember it has been transformed, e.g. `data.norm` to indicate the data has been normalised.

If the data being analysed requires lots of memory, then remove hte raw data after it has been transformed to free some space.

1. Replace all values <100 with the value 100. *Hint*: a command of the format `data[data < 100]` will provide the positions of all the matrix elements with a value of less than 100
2. Replace all values >16,000 with the value 16,000.
3. Convert all values to their log10 equivalent.

Below shows the first 5 rows and 5 columns of the matrix after transformation:

```
##                 ALL      ALL      ALL      ALL      ALL
## AFFX-BioB-5_at   2 2.000000 2.000000 2.000000 2.000000
## AFFX-BioB-M_at   2 2.000000 2.000000 2.000000 2.000000
## AFFX-BioB-3_at   2 2.000000 2.000000 2.423246 2.000000
## AFFX-BioC-5_at   2 2.451786 2.489958 2.000000 2.225309
## AFFX-BioC-3_at   2 2.000000 2.000000 2.000000 2.000000
```

4.4 Normalise data

There is one final stage to go with this data set before it is suitable for full analysis. To allow us to make valid comparisons between the data from different arrays, they need to be normalised. To do this, use the `scale()` function to give all samples (i.e. columns in the matrix) a mean value of zero and a standard deviation of one.

1. Compare the means and standard deviations of the samples using the apply function
2. Normalise the data in the intensity values matrix using scale
3. Use the `apply()` function again to confirm the normalised means and standard deviations
   - *Hint*: review the function of the `apply()` and `scale()` commands using the R help function

Below is the first 5 rows and 5 columns of the matrix after normalisation:

```
##                       ALL        ALL         ALL
## AFFX-BioB-5_at -0.8127832 -0.7582651 -0.83906446
## AFFX-BioB-M_at -0.8127832 -0.7582651 -0.83906446
## AFFX-BioB-3_at -0.8127832 -0.7582651 -0.83906446
## AFFX-BioC-5_at -0.8127832  0.1126268  0.09173842
## AFFX-BioC-3_at -0.8127832 -0.7582651 -0.83906446
```

4.5 Export data

To conclude this section, export the data out of R as a comma separated file (csv). This can them be opened in other data processing applications, or reloaded back into R at a later stage for further analysis. At this time, it would also be a very good idea to save your markup document (which you should have been doing regularly anyway), and save the R workspace so that next time we need to use this data, you can load all the objects from file rather than having to rebuild them again.

1. Use the `write.csv()` or `write.table()` function to export the data matrix to a csv file. Repeat for the gene names.
2. Save your R notebook and knit your document if you have not done so recently.

Chapter 5

Data manipulation - extracting and exporting records

5.1 Removing samples

Our statistical colleagues have now performed some tests on the Golub data, and have warned us that there are some outliers in the dataset. These are characterised by low or high mean intensity values, and large or small standard deviations. Being statisticians who like a joke though, they haven't told us which samples are affected. To be careful, we will remove the eight most extreme samples, two with the highest and two with the lowest mean intensity, and likewise two each with the highest and lowest standard deviation

Some functions that you might find useful in this exercise are `apply`, `colMeans`, `order`, `which.min` and `which.max` .

Using the full Golub matrix (**before log transforming and normalisation**), identify and remove:

- The two samples with the highest mean
- The two samples with the lowest mean
- The two samples with the highest standard deviation
- The two samples with the lowest standard deviation

Warning: When performing such data transformations, becareful about how you are performing the steps.

1. Do you perform each filtration one after the other? That is, filter (a) then filter (b) then filter (c) and then filter (d).

or

2. Do you first find the answers to (a), (b), (c) and (d) before performing the filtration in one go?

Try both approaches above and compare your results. Hint: After each step (a) to (d) look at which sample has the two highest or lowest mean and standard deviation.

5.2 Remove uninformative measurements

The Golub dataset that we have been using so far contains around 7,000 measurements (gene intensities). Much of this data may be redundant or uninformative, so in this section we will explore ways of filtering out low-value information from a dataset.

During our transformation stage, we applied a floor and ceiling value to our measurements, because values outside of this range are considered unreliable. If all the measurements for a gene were below the floor value, or above the ceiling, we can conclude that the expression value of that gene is uniformly outside the useful range of measurement, and ignore the data for that gene. Also, since this is a differential expression analysis, we are not interested in genes which show low levels of variability across the samples, so we will remove them too.

1. Use the `apply()` function to generate a new matrix which excludes genes with an intensity of 100 or less across **all** samples (use the dataset from which you have removed the outliers).

Hint: break the problem down first, find the maximum intensity for each gene, then identify for which genes that maximum is more than 100, then finally slice just those rows to a new matrix

2. Next, filter this new matrix to remove all genes with an intensity of 16,000 or more for **all** samples
3. Finally, filter the matrix once more to remove all genes with less than a five-fold change in signal across the samples

5.3 Extracting specific measurements

To make up for their practical joke earlier, our statistician friends have now come back and provided us with a list of the one hundred genes that their analysis tells them are most important in differentiating between the *ALL* and *AML* groups in our experiment. They have also admitted that there weren't really any outliers after all. To study these further, we therefore want to retrieve the log transformed and normalised intensity values of just these one hundred probes for all 38 of our original samples.

1. Read in the list of 100 probe IDs and associated gene names from the data file **"golub100.txt"**

Hint: the gene names are in column 2 of this tab-delimited file, the probe IDs in column 1

2. Generate a new matrix of log normalised values for all 38 samples and the 100 genes in this list

We will now try further practice on an alternative input dataset. Before doing this, it would again be worth saving your R workspace and your markdown code. Perhaps open a new markdown document for this second dataset

Chapter 6

Integrating datasets

So far in this workshop, we have concentrated on reading in and analysing the contents of a single file. In this section, we are going to look at what we can do if the data is in multiple files that need to be combined or linked.

6.1 Combining two files with identical row identifiers

The first situation might be where the output from our measurement platform is produced in one file for each sample, and then needs to be merged so all results for all samples are in the same matrix or dataframe for analysis. In the simplest case here, the order and length of the files is identical, so they can just be joined as columns in the same dataset. You can do this using the `cbind()` function

```r
# Create several vectors of equal length. Each represents the data from one sample
vec.one <- (1:10)
vec.two <- (2:11)
vec.three <- c(rep(1,3), rep(4,3), rep(8,4))

# Then join them together to make a matrix with one column per sample vector
joined.data <- cbind(vec.one, vec.two, vec.three)
joined.data
```

```
##       vec.one vec.two vec.three
##  [1,]       1       2         1
##  [2,]       2       3         1
##  [3,]       3       4         1
##  [4,]       4       5         4
##  [5,]       5       6         4
##  [6,]       6       7         4
##  [7,]       7       8         8
##  [8,]       8       9         8
##  [9,]       9      10         8
## [10,]      10      11         8
```

Depending on the needs of your analysis, you can alternatively use the `rbind()` function to join them as rows.

`cbind` will work to join matrices as well as vectors, as long as they all have the same number of rows. Try this using some pre-prepared subsets of the `Golub` data.

1. Load in the three datasets stored in file **golub_cbind.RData** using the `load()` function.

  - This file contains three objects:
    - * `golub.names` - a vector of the gene names
    - * `golub.all` - a dataframe of the *ALL* sample data, and
    - * `golub.aml` - a dataframe of the *AML* sample data

2. Use `cbind()` to create a duplicate of the Golub dataframe from these three objects.

6.2 Combining two files with rows in common

`cbind` relies on the datasets being of the same size and in the same order. But what if that is not the case, and not all values are recorded for all samples? Well, if there is a common identifier between the datasets, we can use that to merge two dataframes. As an example, we will try with the built-in R datasets "beaver1" and "beaver2", which give body temperature plots of two beavers over a period of about 18 hours.

For a full list and descriptions of R's built in example datasets, use the `data()` command

```
head(beaver1)
```

```
##   day time  temp activ
## 1 346  840 36.33     0
## 2 346  850 36.34     0
## 3 346  900 36.35     0
## 4 346  910 36.42     0
## 5 346  920 36.55     0
## 6 346  930 36.69     0
```

For this example, we will merge the tables on the 'time' column, so we can start to look how beavers' body temperature fluctuates through the day

```
dim(beaver1)
```

```
## [1] 114   4
```

```
dim(beaver2)
```

```
## [1] 100   4
```

```
beaver.temp <- merge(beaver1, beaver2, by = "time", all = TRUE, sort = FALSE)
dim(beaver.temp)
```

```
## [1] 115   7
```

```
head(beaver.temp)
```

```
##   time day.x temp.x activ.x day.y temp.y activ.y
## 1  930   346  36.69       0   307  36.58       0
## 2  940   346  36.71       0   307  36.73       0
## 3  950   346  36.75       0   307  36.93       0
## 4 1000   346  36.81       0   307  37.15       0
## 5 1010   346  36.88       0   307  37.23       0
## 6 1020   346  36.89       0   307  37.24       0
```

What's happening here?

Function/Argument	Description
`by`	is the column to match the two datasets with. You can use column names, numbers or (to merge on rownames) `row.names`. *Note*, if you sort by `row.names`, then an extra column will be created in the output object containing the rownames information, while the rownames themselves of this new object will revert to 1, 2, 3...
`all = TRUE`	means that a row will be created for every time point in either of the two samples. If one sample is missing data at that point, then values of "NA" will be entered in the output
`sort = FALSE`	by default, merge sorts the output object in the order of the merge column. In this dataset, the timecourse starts in mid-morning, and continues until after midnight the next day, so sorting is not appropriate in this case.

Unfortunately, even turning off sort doesn't do what we want in this case. Rather than leaving the rows in the original order, they are reordered but in a somewhat unpredicatable way. There are functions in non-core R packages (like join in the `plyr` package that address this, but for the moment, a quick and dirty way is to add 2400 to all times after midnight and then sort the data by the time column:

```
beaver.temp[beaver.temp[, 'time'] < 800, 'time'] <-
      beaver.temp[beaver.temp[, 'time'] < 800, 'time'] + 2400
beaver.temp <- beaver.temp[order(beaver.temp[,'time']),]
```

1. The **golub_cbind.RData** file you loaded earlier contains two, partially overlapping, datasets from the all and aml samples (all.subset and aml.subset). Use `merge()` to join these two dataframes using rownames as identifiers
2. For the dataframe generated by the merge, relabel the rownames correctly, and remove the now redundant "Row.names" column from the dataframe

Chapter 7

Charts and graphical output

R has a wide range of built in charts and graphics, and with packages such as ggplot, even more advanced plotting functions are available. In many cases, a crude but valid plot can be produced simply by supplying a matrix or vector directly to one of the plot functions.

7.1 Refresher - standard R plotting functions

7.1.1 Plot() function

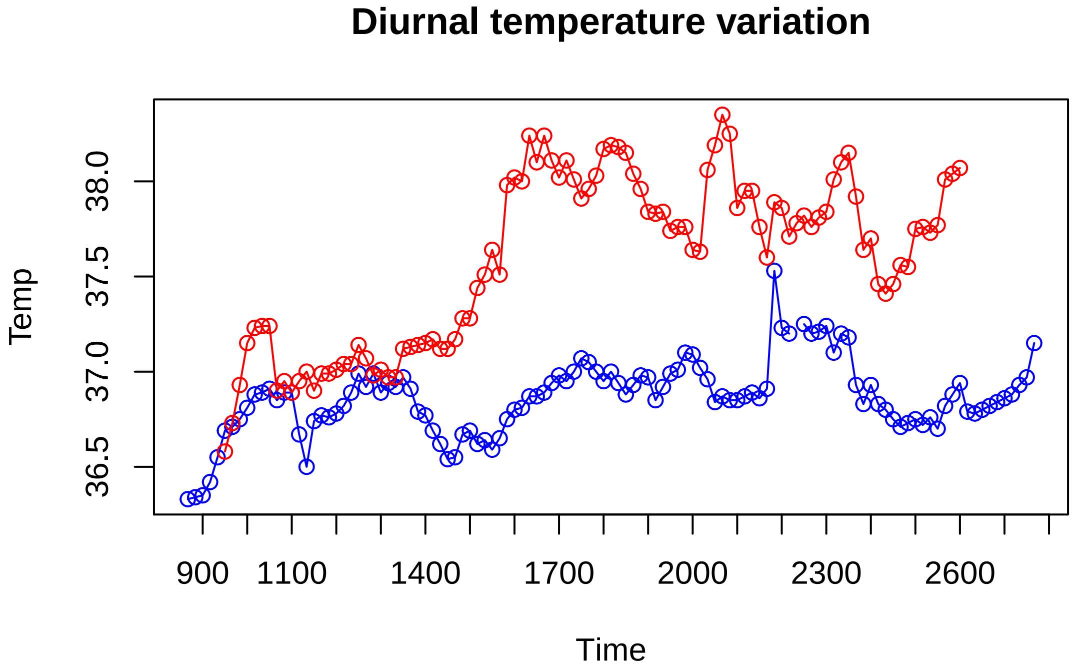
`plot()` can be used for several other applications. These include line graphs (showing, for example, changes over time) and volcano plots (for gene expression analysis). Let's revisit this using our beaver temperature dataset, and preparing a scatterplot of our time-merged data.

1. Make a scatterplot comparing body temperature of the two beavers from our merged data.frame.
2. Review the `plot` documentation to add the following:
   - a **title** to the plot,
   - replace the axis names with `Beaver 1` and `Beaver 2`
   - change the plot symbols
   - add a trend line

This gives us a correlation, but does not show how data changes over time. For this we need a different type of plot, either `type = "l"`, which gives a line plot; or `type = "o"`, which standard for `overplot` and gives points connected by lines. Drawing this plot is a little more involved, since we need to plot the two datasets separately, both scaled against time.

Before we start anything we need to know the maximum range required for the axes. By default, when we plot the graph, it will scale for the first animal, but adding others may go outside the display range

```
temp.range <- range(beaver.temp$temp.x, beaver.temp$temp.y, na.rm = TRUE)

# Set up the plot for the first animal
# In this case, we'll define the axis labels at the outset
# xaxt = 'n' means not to give the x-axis scale - we'll add that in a moment
plot(beaver.temp$temp.x, type="o", col="blue",
     ylim = temp.range, xaxt = 'n',
     xlab = "Time", ylab="Temp")

# Now add the second animal data
lines(beaver.temp$temp.y, type = "o", col="red")

axis(1, at = 3+6*c(0:19), lab = beaver.temp$time[3+6*c(0:19)])
title(main="Diurnal temperature variation")
```



(Optional)

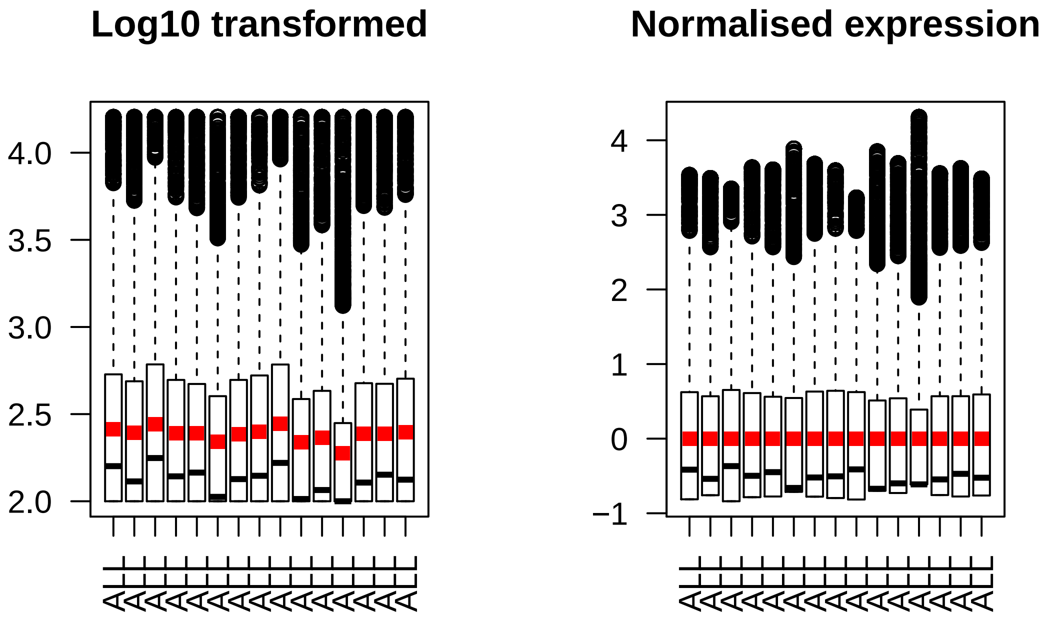
Further timecourse plotting practice

1. Using another built-in R dataset `Indometh`, examine the dataset and plot time course changes for two or more of the six samples.
2. Give each sample a different colour.
3. Try to label and scale the axes appropriately.

7.1.2 Boxplots

Let's plot our microarray data from the previous exercise side by side to show the unnormalised data alongside the normalised dataset. The red squares on each boxplot shows the location of the mean value.

```
par(mfrow=c(1,2))
means <- colMeans(golub.log)
boxplot(golub.log[,1:15],las=2,main="Log10 transformed")
points(1:15, means[1:15], pch=15, col='red')

means <- colMeans(golub.norm)
boxplot(golub.norm[,1:15],las=2,main="Normalised expression")
points(1:15, means[1:15], pch=15, col='red')
```



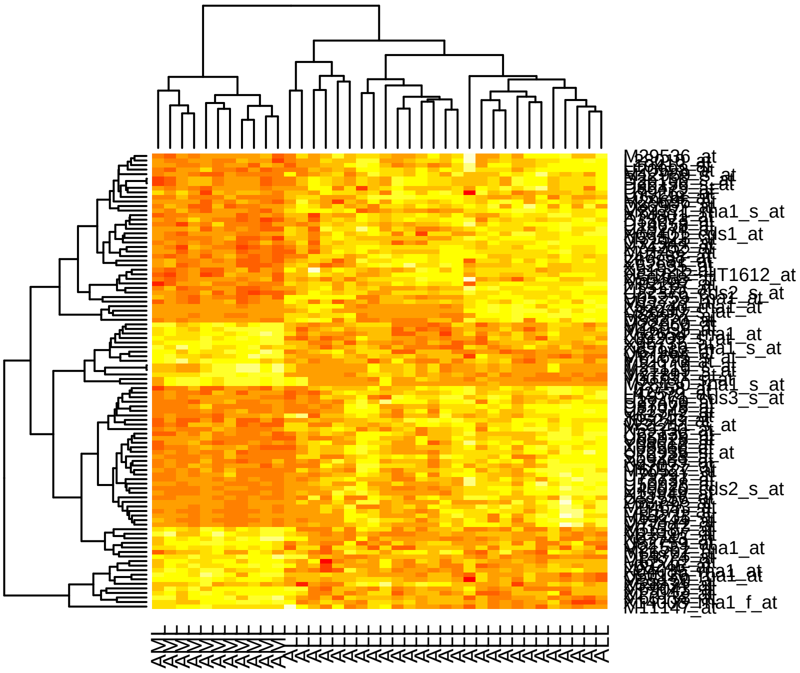
Make the y-axes to be the same range in both plots so that it is not misleading.

7.1.3 Heatmaps

Another commonly used way of displaying biological datasets is the heatmap. This is commonly used for gene expression analysis, but it can be used for any dataset where multiple quantitative measurements are carried out on several samples. Not only can an R-generated heatmap help in the visualisation of these sort of datasets, but the heatmap command can also be used to automatically organise samples and/or measurements into dendrograms, showing the similarity relationship between them.

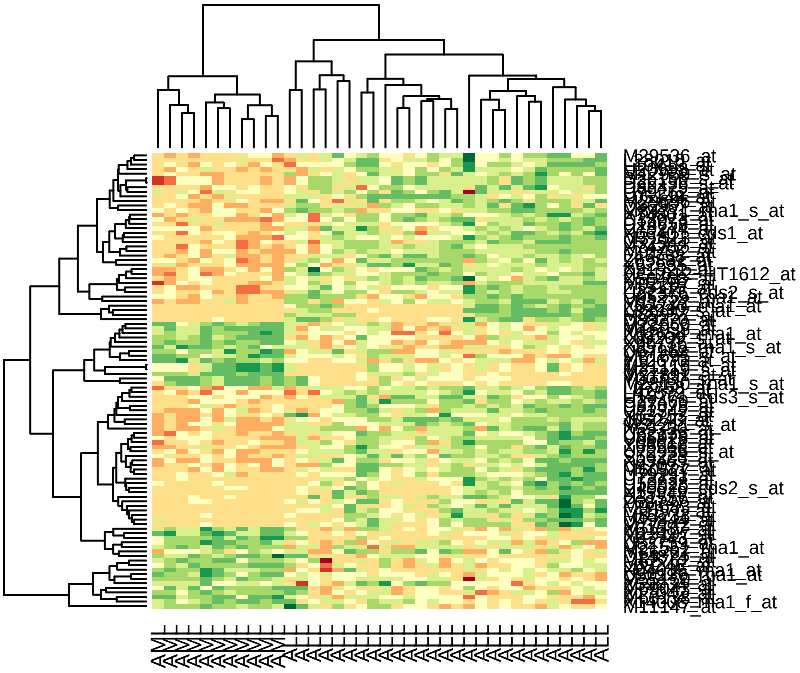
For a first example, let's plot a heatmap for our 100 key Golub dataset genes.

```
# Run the simplest heatmap, with all default settings
heatmap(golub.100)
```



If we want to follow the conventional red-green microarray style, we can use the `RdYlGn` colour palette from the `RColourBrewer` package.

```r
library(RColorBrewer)
heatmap(golub.100, col = brewer.pal(11, name='RdYlGn'))
```



Use `display.brewer.all()` or `display.brewer.pal(n,name)` to see the range of palettes available from this package.

By default, this actually gives a very presentable plot. Both rows and columns are automatically sorted to cluster most alike genes and samples together, and dendrograms are plotted for us automatically. Encouragingly, we can see that all the *AML* and *ALL* samples form two separate clusters. It might look better without the crowded gene names column, but that can be turned off easily with the setting `labRow = NA`.

For more practice, see if you can prepare a heat plot of the Indometh dataset:

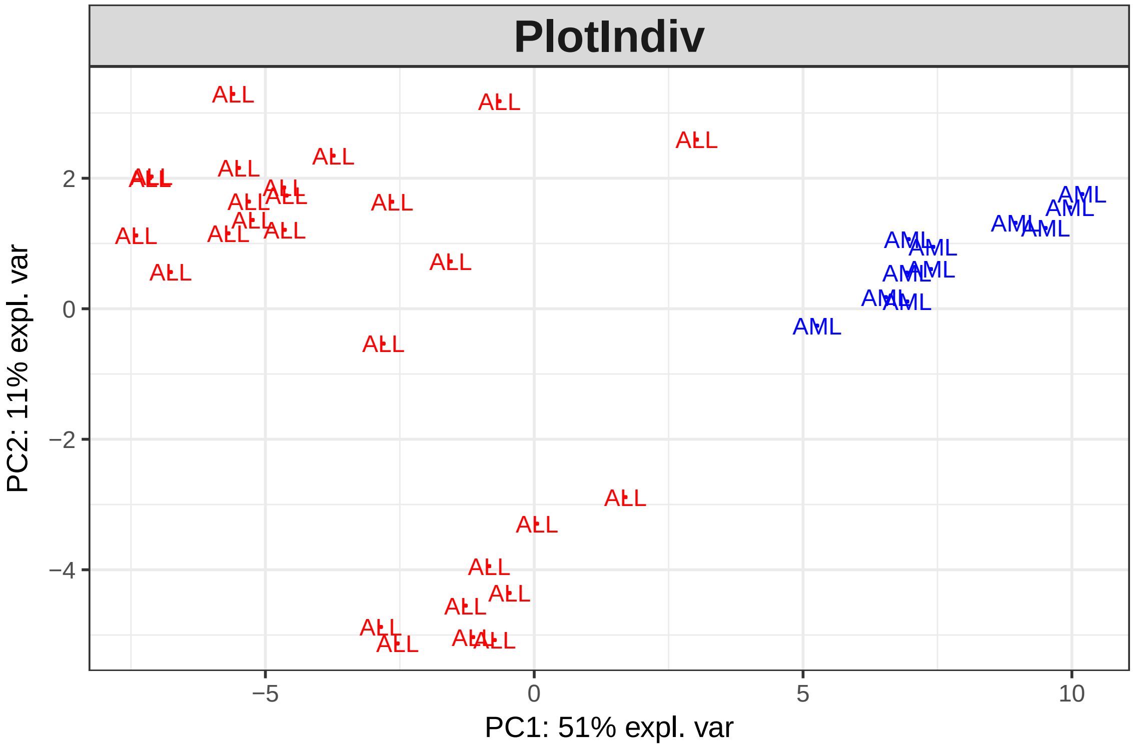
- *Hints*:
  - you will need to convert the dataset into a matrix with one column per *Subject*. Set column names as Subject number and rownames as timepoints
  - review the Rowv setting to keep time points in order in the dendrogram
  - think about transforming the data to compensate for the exponential decrease of the concentration value
  - try different colour palettes to highlight the data

7.1.4 PCA plots

The final plotting option we will explore is principle component analysis, or PCA. PCA is often used where there are two or more categories of data, and multiple quantitative measurements on each, to quickly review whether the difference between the categories outweighs any noise or batch effect signals. Essentially, PCA collapses the dataset down to a small number of representative dimensions which can be plotted on a chart.

PCA analysis is widely used in expression datasets, which have large numbers of measurements (one per gene), multiple replicates, and normally two or more categorical conditions, so we will test it with our Golub dataset. We will use the PCA function of the 'mixOmics' package.

```r
library(mixOmics)

# Carry out the PCA on just the top 100 genes, with default parameters
# N.B. PCA data needs to have the samples in rows. Our data is by
# columns, so we need to transpose it
golub.100.pca <- pca(t(golub.100))

# Then plot the data for the first two components, colouring the
# sample names for clarity
plotIndiv(golub.100.pca, comp = c(1,2),
          col = c(rep('red', 27), rep('blue', 11)))
```



1. Generate a PCA plot for other components of the analysis
2. Carry out a PCA analysis for the full (normalised, transformed) Golub dataset and generate a plot
3. Does the data differentiate between the two sample types?

7.2 ggplot2

Most of the plots that we have generated in this workshop have been using built-in R plotting functions. These are generally adequate, but for the highest quality charts we may choose to use a specialist package such as `ggplot2`. This is an extremely powerful and adaptable graphics package, but at the expense of being more challenging to learn and use than the standard tools.

Today we will just explore `ggplot2` briefly to draw some better line graphs than earlier, but there are plenty of books and web resources to help you learn more, http://docs.ggplot2.org/current/ is a good place to start.

One of the first challenges of ggplot is that it doesn't take data in the *wide* format of the matrices and data.frames we have been using up to now, that is, where a row contains measurements from multiple samples, or multiple measurements from one sample. Instead, data has to be reformatted into the **long** format, with just one measurement per row (along with sample identifiers, categorical variables and so on). To get our data into the right format, we will use the `tidyr` package.

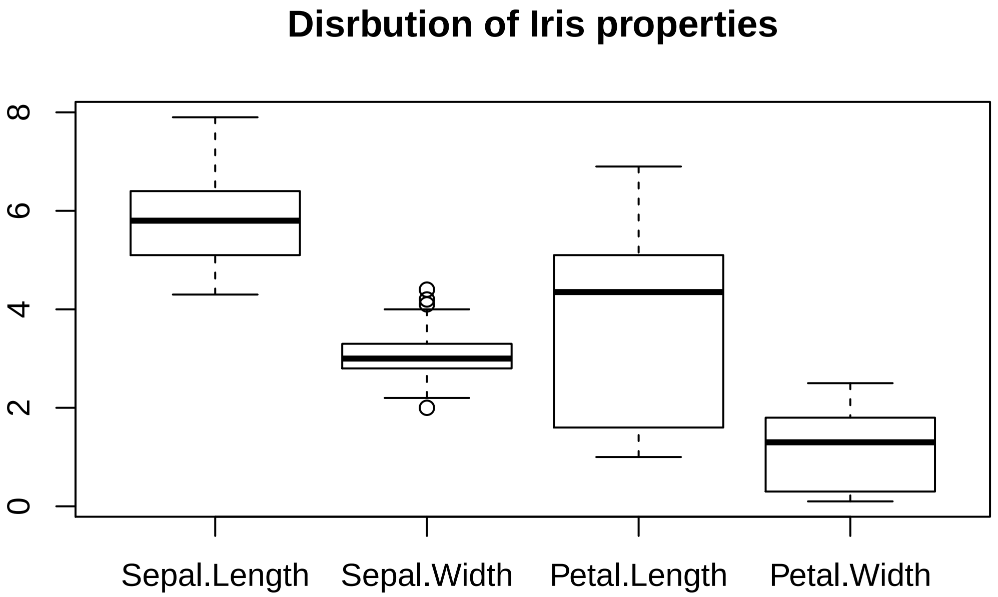
```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

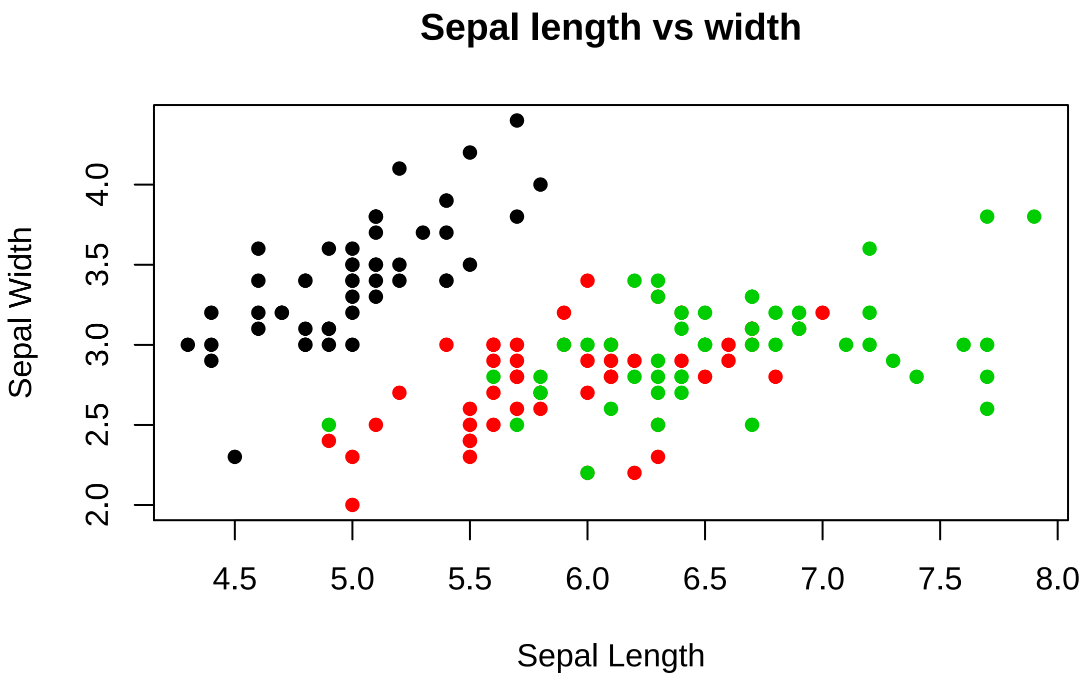
```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
boxplot(iris[,1:4], main="Disrbution of Iris properties")
```



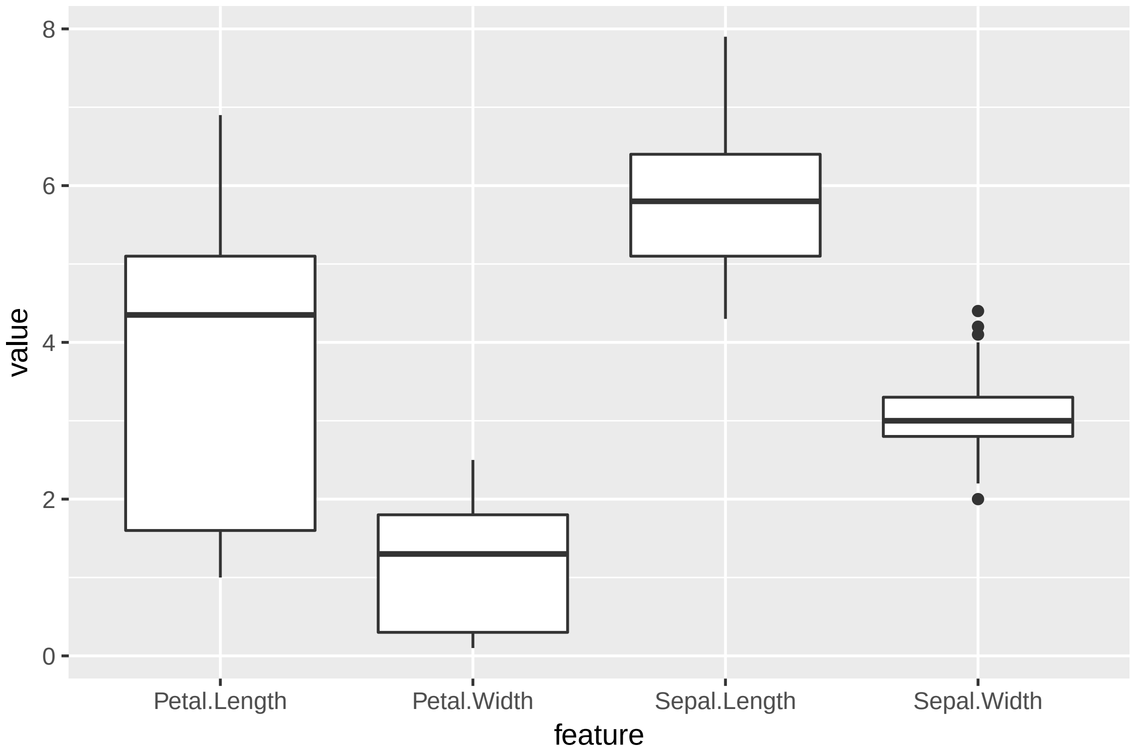
```
plot(iris$Sepal.Length, iris$Sepal.Width, pch=16, col=iris$Species,
     main="Sepal length vs width",
     xlab="Sepal Length",
     ylab="Sepal Width")
```



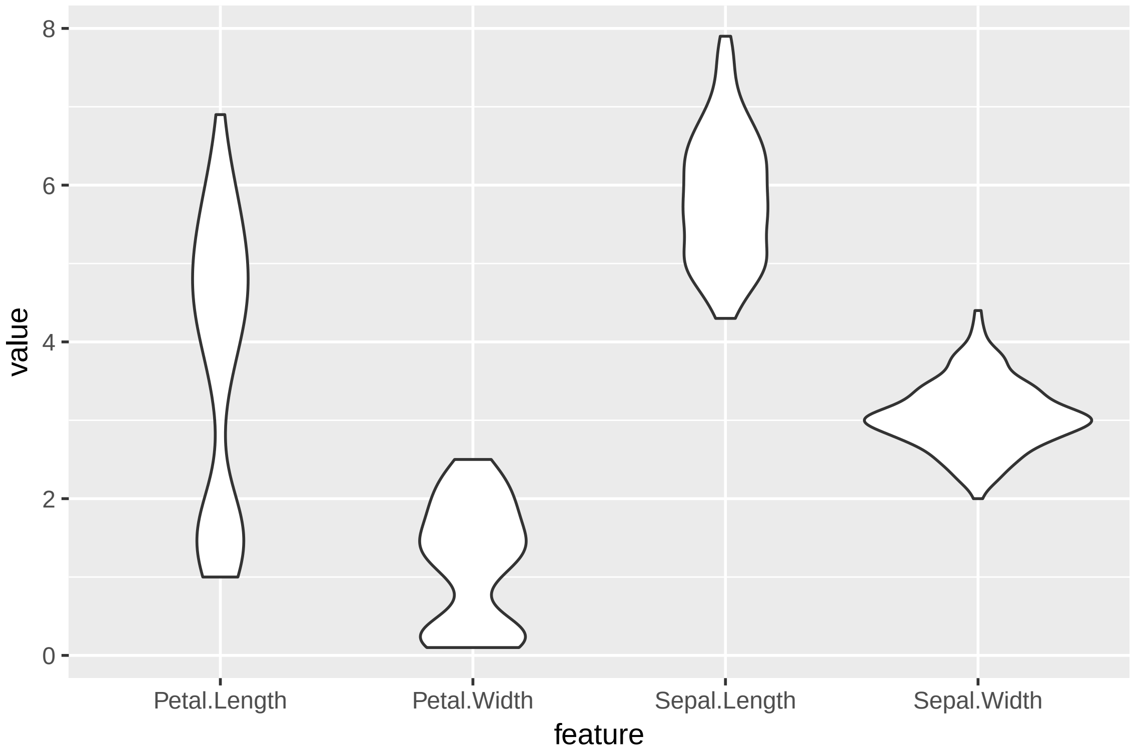
```
library(tidyr)
long.form <- gather(iris,feature,value,-Species)
head(long.form)
```

```
##   Species      feature value
## 1  setosa Sepal.Length   5.1
## 2  setosa Sepal.Length   4.9
## 3  setosa Sepal.Length   4.7
## 4  setosa Sepal.Length   4.6
## 5  setosa Sepal.Length   5.0
## 6  setosa Sepal.Length   5.4
```

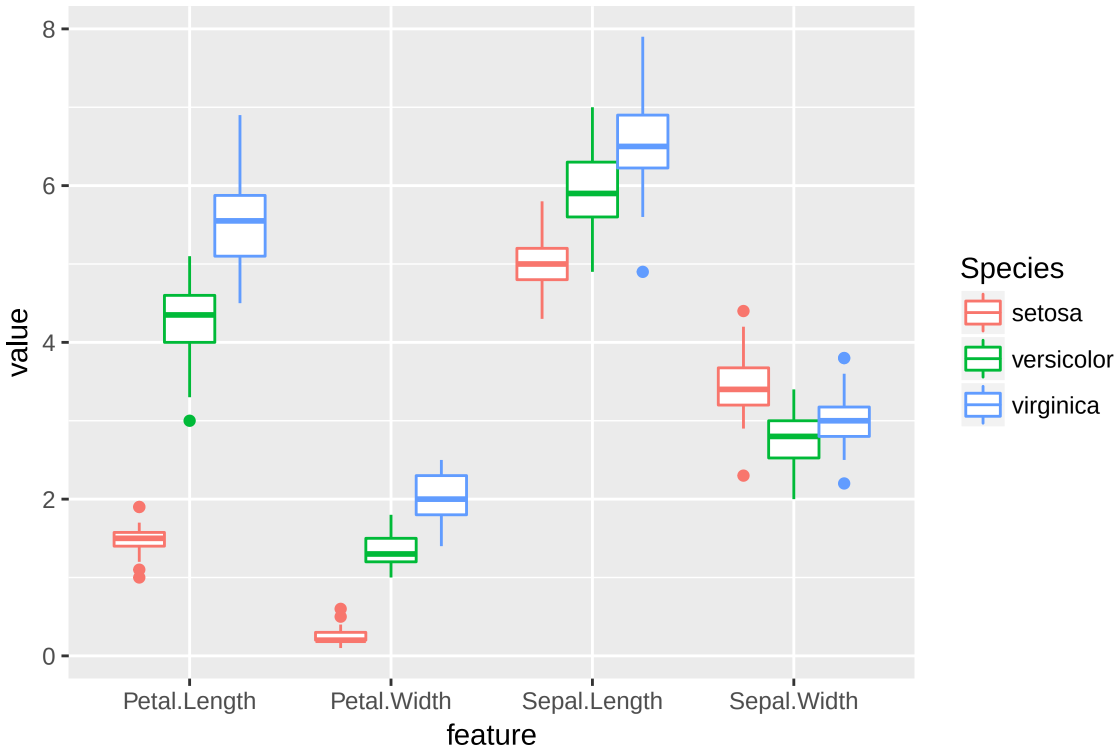
```
ggplot(long.form) + geom_boxplot(aes(x=feature,y=value))
```



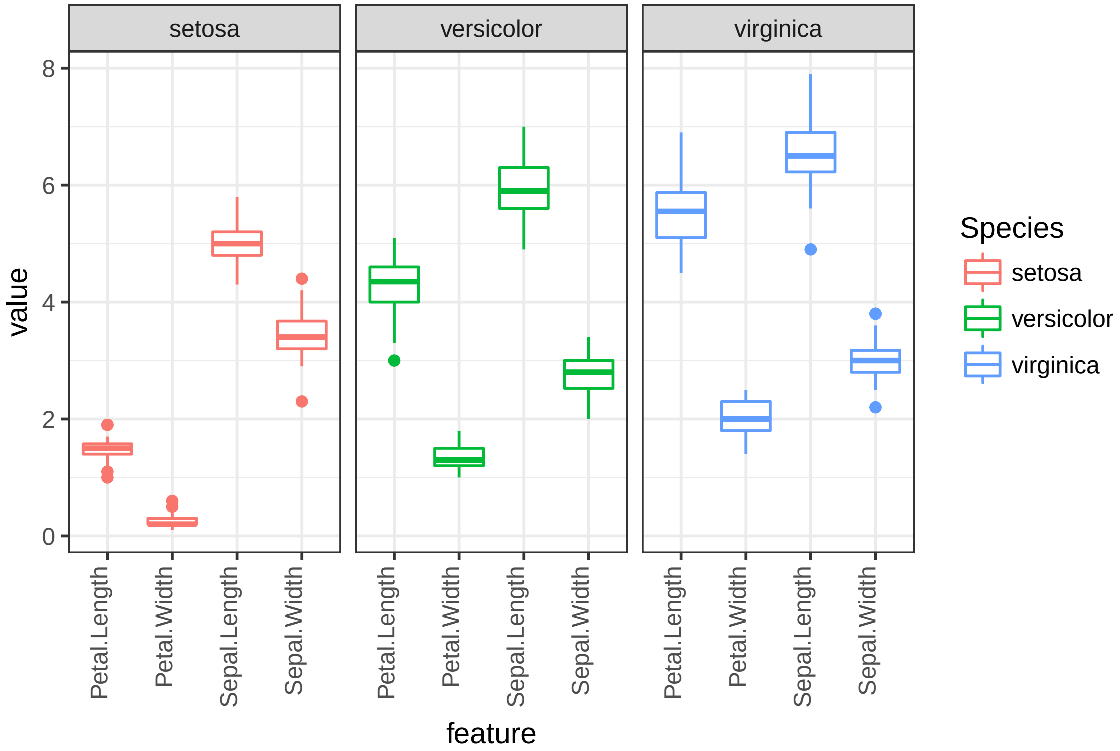
```
ggplot(long.form) + geom_violin(aes(x=feature,y=value))
```



```
ggplot(long.form) + geom_boxplot(aes(x=feature,y=value,col=Species))
```



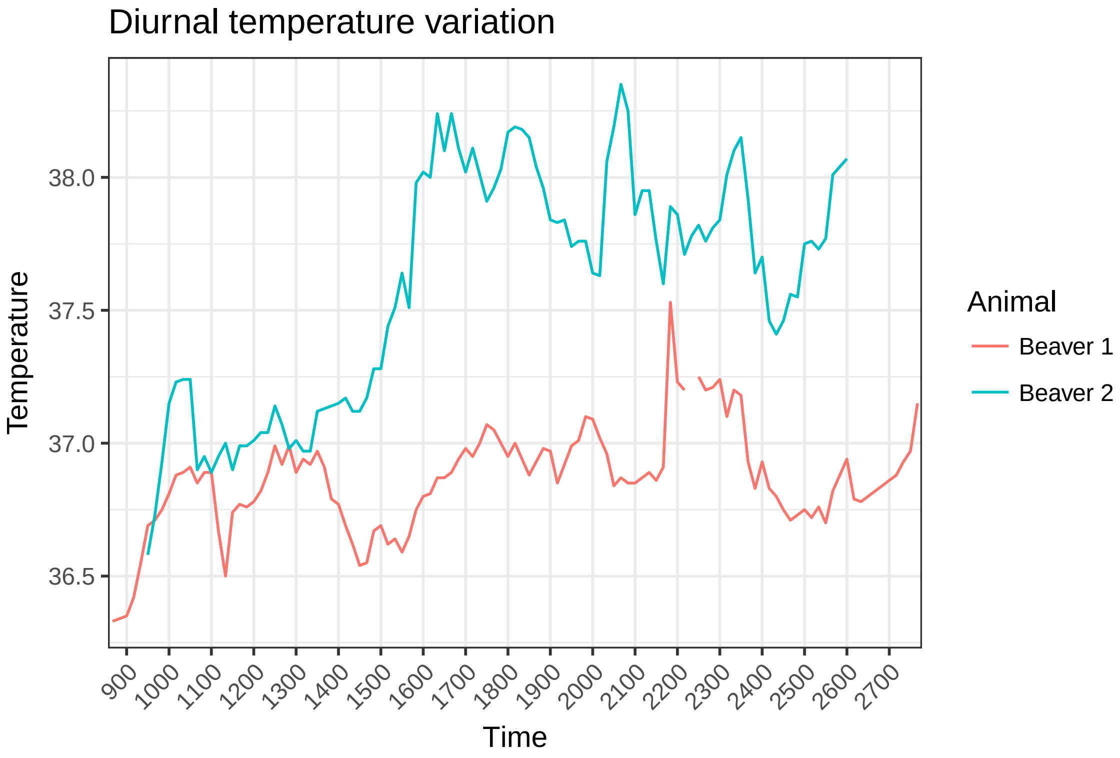
```
ggplot(long.form) + geom_boxplot(aes(x=feature,y=value,col=Species)) + facet_grid(.~Species) +
  theme_bw() + theme(axis.text.x = element_text(angle=90,hjust=1,vjust=0))
```



Challenge exercise Reformat our `beaver.temp` dataset into the long format and use `ggplot()` to plot the change in temperature for the two beavers over time.

Try to reproduce the following plot:

```
## Warning: Removed 15 rows containing missing values (geom_path).
```



(Optional) Use the in-built `diamonds` dataset and plot some graphs using ggplot. e.g. are there any difference in the quantitative attributes: *carat*, *depth*, *price*, versus the categorical propreties: *cut*, *color*, *clarity*.

Chapter 8

Generating report with R

Hopefully through this workshop, you have been editing your R code in a knitR (R markdown) document, and now have a record of all the analyses that you have carried out over the course of the workshop. The final exercise is to tidy up these documents to produce the final report.

Typically at QFAB, we might produce two R markdown documents for an analysis. The first is a template which contains all the R code, commentry on what the code is doing, section headers and so on, while the second is build around this template with data-specific interpretive comments inserted where relevant.

The first document allows us to reuse the template and re-run the same analysis pipeline on a comparable input sample at a later date, while the second forms the final analysis report for a specific sample (the comments have to be inserted after the analysis of course, because the same analysis process on two different samples may well give two very different outcomes).

1. Working with one (or more) of your knitR documents, tidy up the code and documentation to produce a reusable template format of that dataset.
2. Create a copy of the template and add some interpretation and commentary to generate a final, reproducible report
   - *Hints*:
     - Use the RStudio **File** > **Save As** function to create a copy of the template for the report
     - Give the chunks names to make their purpose easier to remember (to do this, use the format `{r chunk_name}`, instead of just `{r}` at the start of the chunk)

You can hide R code in your report but still include the output from that code by adding `echo = FALSE` to the chunk header (e.g. `{r chunk_name, echo = FALSE}`). If you want to run R code without showing either the code or the output, use `include = FALSE`

- See http://kbroman.org/knitr knutshell/pages/Rmarkdown.html for descriptions of some of the other chunk display options

Once you have generated your final knitR report, click on the Save As link at the top of the preview window, and you can download the report to your local computer in HTML format. When using the web-based version of RStudio, this is the only option to save knitR output; however, if you install and run R and RStudio locally, you can also generate PDF and Word format reports.

8.1 Other reporting formats

During this workshop, we have focussed on using knitR. You may have noticed that under the New File menu option, there are a number of other formats. These all provide similar functionality to knitR, but with various different strengths and weaknesses. This workbook is written in Sweave, which is somewhat better at producing high-quality PDF format documents, but is also less user friendly than knitR and cannot generate HTML output. The best way to learn about these other formats is just to install R and RStudio on your own computer and start experimenting!

8.2 Interactive reporting with Shiny

Using knitR, Sweave or any other version of literate programming can give us reproducible, automated data analysis and report generation. However, the reports produced in this way are ultimately still static, with graphs and charts locked to the specifications of the R code. Shiny is an R tool that can be used to overcome this, by generating dynamic HTML plots that can be changed by the viewer to suit their preferences.

As an example, a static report may define a histogram with data divided into five levels. Using Shiny, a viewer of this data may choose to look in higher resolution, with 10 or 12 levels; conversely, they may generate a very broad summary of the data in just three levels. Similarly, another analysis may output a list of samples that are relevant at a p-value of 0.05, but with Shiny, the user can change that selection to list samples significant only at 0.01, or 0.001.

Unfortunately Shiny does not work well with the web-based RStudio version that we are using for this workshop, so in this session your instructor will demonstrate examples of interactive reporting with Shiny. These are based on the examples at http://shiny.rstudio.com/tutorial/, so if you have R and RStudio installed on your laptop, you are welcome to try them out for yourself at the same time. Even if you don't join in now, have a go at them later - the examples on this page are specifically designed to help you teach yourself how to use Shiny.

Chapter 9

Conclusion

That brings us to the end of the workshop. Don't worry if you didn't manage to complete all the exercises - we didn't expect that you would be able to. Instead, try to take some time over the next few days or weeks to return to the workbook and spend a little more time practicing. You can continue to access your account on the training server for the next few weeks, or you can install R and RStudio on your own machine and download the example files from the training resource site. And don't forget, before you go, save your R workspace and download this and all your markdown documents to your laptop. That way, you will be able to recreate everything you have done today quickly and simply when you come back to it.

Appendix: Exercise answers

4 Importing and reformatting data files

4.1 Read in the datafile

```
## [1] "Gene Description\tGene Accession
Number\t1\tcall\t2\tcall\t3\tcall\t4\tcall\t5\tcall\t6\tcall\t7\tcall\t8\tcall\t9\tcall\t10\tcall\t1
## [2] "AFFX-BioB-5_at (endogenous
control)\tAFFX-BioB-5_at\t-214\tA\t-139\tA\t-76\tA\t-135\tA\t-106\tA\t-138\tA\t-72\tA\t-413\tA\t5\tA
## [3] "AFFX-BioB-M_at (endogenous
control)\tAFFX-BioB-M_at\t-153\tA\t-73\tA\t-49\tA\t-114\tA\t-125\tA\t-85\tA\t-144\tA\t-260\tA\t-127\
## [4] "AFFX-BioB-3_at (endogenous
control)\tAFFX-BioB-3_at\t-58\tA\t-1\tA\t-307\tA\t265\tA\t-76\tA\t215\tA\t238\tA\t7\tA\t106\tA\t42\t
## [5] "AFFX-BioC-5_at (endogenous
control)\tAFFX-BioC-5_at\t88\tA\t283\tA\t309\tA\t12\tA\t168\tA\t71\tA\t55\tA\t-2\tA\t268\tA\t219\tM\
## [6] "AFFX-BioC-3_at (endogenous
control)\tAFFX-BioC-3_at\t-295\tA\t-264\tA\t-376\tA\t-419\tA\t-230\tA\t-272\tA\t-399\tA\t-541\tA\t-2
```

```r
# Let's set up a few shortcuts first
DATADIR <- "~/data/Data_interpretation_with_R"
Golub.file <- file.path(DATADIR, "data_set_ALL_AML_train.tsv")

# Inspect the file content
head(readLines(Golub.file))

# Read the file into the golub.df dataframe
golub.df <- read.table(Golub.file,sep="\t", quote="", header=T,
                       row.names=2, comment.char="",
                       stringsAsFactors = FALSE)
```

4.2 Reformat the file

```r
# Create a golub matrix containing just the intensity values
# 2*[1:38] returns even numbers from 2 to 2x38 (i.e. 76)
golub.matrix <- as.matrix(golub.df[,2*(1:38)])

# Name the columns with the sample status
# The first 27 are "ALL", the final 11 are "AML"
colnames(golub.matrix) <- c(rep("ALL",27),rep("AML",11))

# Then create a vector of gene names
golub.gnames <- golub.df$Gene.Description
# or golub.gnames <- golub.df[,1]
names(golub.gnames) <- rownames(golub.df)
```

4.3 Transform data

```
# Find the positions of values in golub below our floor or above our ceiling.
# Then replace the values in those positions by our floor or ceiling value.
golub.matrix[golub.matrix < 100] <- 100
golub.matrix[golub.matrix > 16000] <- 16000

# Log_10 transformation
golub.log <- log(golub.matrix,10)
```

4.4 Normalise data

```
# Check the existing mean and standard deviation
apply(golub.log, 2, mean)
```

```
##      ALL      ALL      ALL      ALL      ALL      ALL      ALL      ALL
## 2.412221 2.393360 2.441669 2.391413 2.390291 2.341846 2.384387 2.399676
##      ALL      ALL      ALL      ALL      ALL      ALL      ALL      ALL
## 2.444495 2.338539 2.363399 2.274308 2.386571 2.388623 2.395650 2.380295
##      ALL      ALL      ALL      ALL      ALL      ALL      ALL      ALL
## 2.482811 2.324345 2.328069 2.499497 2.306725 2.313102 2.331595 2.409902
##      ALL      ALL      ALL      AML      AML      AML      AML      AML
## 2.343512 2.352611 2.360633 2.382545 2.336199 2.427405 2.401558 2.386542
##      AML      AML      AML      AML      AML      AML
## 2.406045 2.296222 2.400678 2.361889 2.418098 2.451635
```

```
apply(golub.log, 2, sd)
```

```
##       ALL       ALL       ALL       ALL       ALL       ALL       ALL
## 0.5071723 0.5187630 0.5263826 0.4988002 0.5032199 0.4792356 0.4942427
##       ALL       ALL       ALL       ALL       ALL       ALL       ALL
## 0.5020369 0.5447880 0.4846996 0.4989016 0.4477448 0.5113511 0.5011462
##       ALL       ALL       ALL       ALL       ALL       ALL       ALL
## 0.5190887 0.4920936 0.5537143 0.4649320 0.4727689 0.5794321 0.4687063
##       ALL       ALL       ALL       ALL       ALL       ALL       AML
## 0.4640572 0.4649017 0.5043298 0.4843511 0.4877606 0.4773038 0.5001143
##       AML       AML       AML       AML       AML       AML       AML
## 0.4653518 0.5418554 0.5233421 0.5008276 0.5119954 0.4655509 0.5325124
##       AML       AML       AML
## 0.5000293 0.5186136 0.5380754
```

```
# Make average log value for each column equal to zero, then divide all
# values by the standard deviation
golub.norm <- scale(golub.log, center = TRUE, scale = TRUE)

# Now recheck means and standard deviations
apply(golub.norm, 2, mean)
```

```
##           ALL           ALL           ALL           ALL           ALL
## -1.643583e-17 -5.344799e-17  2.832026e-16  9.722091e-17 -4.462088e-16
##           ALL           ALL           ALL           ALL           ALL
## -3.077379e-16 -1.595621e-16  3.048740e-16  3.319556e-16  2.381528e-16
##           ALL           ALL           ALL           ALL           ALL
## -3.858281e-16  5.794228e-17  3.542128e-16  1.424590e-16  1.303877e-16
##           ALL           ALL           ALL           ALL           ALL
## -1.626928e-16  3.069186e-16 -9.231404e-17  4.130346e-16 -3.053645e-16
##           ALL           ALL           ALL           ALL           ALL
##  2.734703e-16 -9.293533e-17  4.028515e-16  6.367562e-17 -4.141509e-16
##           ALL           ALL           AML           AML           AML
## -3.691710e-16 -2.962760e-16  6.240772e-17  3.223573e-16  2.943473e-16
##           AML           AML           AML           AML           AML
## -3.177875e-16 -2.160962e-16 -1.514798e-16  2.100244e-16  3.420917e-16
##           AML           AML           AML
##  2.160032e-17 -1.629890e-16 -7.954762e-17
```

```
apply(golub.norm, 2, sd)
```

```
## ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## ALL ALL ALL ALL ALL ALL ALL ALL ALL AML AML AML AML AML AML AML AML AML
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## AML AML
##   1   1
```

4.5 Export data

```
# Export a csv version of the matrix
write.table(golub.norm, file="golub_normalised_data.csv",sep=",", quote=FALSE)
# OR
write.csv(golub.norm, file="golub_normalised_data.csv")
# And of the gene name conversion table
write.table(golub.gnames, file="golub_gene_names.csv", sep=",", quote=TRUE)
```

- Save the knitR document by just clicking on the disk icon at the top of the knitR markdown document. This may be greyed out if you have just created HTML, since knitR saves the file automatically before running.
- Save your workspace by going to the Session menu at the top of the RStudio window, and selecting "Save Workspace As"

5.1 Data manipulation

5.1.2 Removing samples

```
# See what we're starting with
dim(golub.matrix)
```

```
## [1] 7129   38
```

```
# Identify and remove the two samples with the highest and lowest mean
# Yes, we can do this in one, rather complicated fcommand
golub.no.outliers <- golub.matrix[,-(order(colMeans(golub.matrix))[
                                    c(1,2,ncol(golub.matrix),ncol(golub.matrix)-1)])]

# Leaving us with just 34 columns
dim(golub.no.outliers)
```

```
## [1] 7129   34
```

What's happening here?

Function	Description
`colMeans`	is calculating the mean for each column (you could also use apply)
`order`	is giving the column numbers in ascending order of mean value
`c(1,2, ncol...)`	retrieves the first two and the last two entries from that order list - i.e. the column numbers with the two lowest and highest means

Finally we take a slice of the golub matrix, exluding those four columns using the - annotation.

```
# Repeat the process for standard deviation. There is no colSD function,
# so we will use apply instead
golub.no.outliers <- golub.no.outliers[,-(order(apply(golub.no.outliers, 2,sd))[
                                      c(1,2,ncol(golub.no.outliers),ncol(golub.no.outliers)-1)])]
```

While there is a certain satisfaction in building a command that identifies and removes the two highest and lowest mean samples in a single step, any solution which reliably produces the same outcome is a suitable answer. There are any number of ways to answer this exercise. For example, you could create a vector containing the mean values for each column and use the `which.min` and `which.max` commands on that to find which column numbers to remove.

5.1.2 Remove uninformative measurements

```
## to be save, we make a copy of the original data before filtering
tmp <- golub.no.outliers
nsamples <- ncol(tmp)
dim(tmp)
```

```
## [1] 7129   30
```

```
## remove rows where all genes are <=100
below.floor <- tmp <= 100
remove <- which(rowSums(below.floor) == nsamples)
tmp <- tmp[-remove,]
dim(tmp)
```

```
## [1] 5954   30
```

```
## remove rows where all genes are >=16,000
above.ceil <- tmp >= 16000
remove <- which(rowSums(above.ceil) == nsamples)
tmp <- tmp[-remove,]
dim(tmp)
```

```
## [1] 5939   30
```

```
## remove rows with less than 5 fold change across samples
ratios <- apply(tmp,1,max)/apply(tmp,1,min)
keep <- which(ratios > 5)
tmp <- tmp[keep,]
dim(tmp)
```

```
## [1] 2902   30
```

```
golub.filtered <- tmp;
rm(tmp);
```

5.1.3 Extracting specific measurements

```
# First, read in the file and convert the contents into a named vector
golub100.gnames.tmp <- read.table(file.path(DATADIR, "golub100.txt"),
                                  stringsAsFactors = FALSE)
golub100.gnames <- golub100.gnames.tmp[,2]
names(golub100.gnames) <- golub100.gnames.tmp[,1]

# From the log normalised matrix, extract the 100 rows of interest
golub.100 <- golub.norm[names(golub100.gnames),]
```

5.2 An alternative practice data set

```
# Import the file
species <- read.csv(file.path(DATADIR, "species.csv"), header = TRUE,
                    as.is = c("Scientific_name","Common_name", "Taxon_author"))

# Identify the classes of organism
levels(species$Class)
```

```
##  [1] ""                     " "                    "amphibians"
##  [4] "arachnids"            "birds"                "bivalves"
##  [7] "blue-green algae"     "branchiopods"         "brown algae"
## [10] "cartilaginous fishes" "club fungi"           "club mosses"
## [13] "conifers"             "cycads"               "diatoms"
## [16] "dinoflagellates"      "euglenoids"           "ferns"
## [19] "golden-brown algae"   "green algae"          "higher dicots"
## [22] "hornworts"            "insects"              "lampreys"
## [25] "liverworts"           "lobe-finned fishes"   "lower dicots"
## [28] "malacostracans"       "mammals"              "maxillopods"
## [31] "monocots"             "mosses"               "quillworts"
## [34] "ray-finned fishes"    "red algae"            "reptiles"
## [37] "sac fungi"            "slime molds"          "snails"
## [40] "spike mosses"         "uncertain"            "whisk ferns"
## [43] "yellow-green algae"
```

```
# OR
unique(species$Class)
```

```
##  [1] amphibians                                arachnids
##  [4] birds                bivalves             branchiopods
##  [7] cartilaginous fishes insects              lampreys
## [10] lobe-finned fishes   malacostracans       mammals
## [13] maxillopods          ray-finned fishes    reptiles
## [16] snails               club fungi           sac fungi
## [19] uncertain                                 slime molds
## [22] higher dicots        liverworts           ferns
## [25] monocots             mosses               lower dicots
## [28] hornworts            conifers             cycads
## [31] quillworts           club mosses          whisk ferns
## [34] spike mosses         red algae           green algae
## [37] diatoms              blue-green algae    brown algae
## [40] yellow-green algae   golden-brown algae  euglenoids
## [43] dinoflagellates
## 43 Levels:    amphibians arachnids birds bivalves ... yellow-green algae
```

```
# Select just the entries for a single class
species.class <- species[(species[,"Class"] == "amphibians"),]

# Remove the unwanted fields
species.class <- subset(species.class,
select = -c(Taxon_Id, Kingdom, Class, EPBC_status))

# Then to finish things off cleanly, export the data
write.csv(species.class, file = "Queensland_amphibians.csv", row.names = FALSE)
```

6 Integrating and summarising datasets

6.1.1 Combining two files with identical row identifiers

```
# Load the R data file
load(file.path(DATADIR, "golub_cbind.RData"))

# Check the objects to make sure that they have the same number of elements/rows
# Remember, golub.names is a vector, the others are dataframes
length(golub.names)
```

```
## [1] 7129
```

```
dim(golub.all)
```

```
## [1] 7129   55
```

```
dim(golub.aml)
```

```
## [1] 7129   21
```

```
# Then just join them together
golub.new.df <- cbind(golub.names, golub.all, golub.aml)
dim(golub.new.df)
```

```
## [1] 7129   77
```

6.1.2 Combining two files with rows in common

```
# Merge the two datasets by 'row.names'
# Set all = TRUE to include values present in only one or other set (or
# experiment with other options)
# Turn off sorting, although for expression data, that's probably unnecessary
golub.merge <- merge(all.subset, aml.subset, by = 'row.names',
all = TRUE, sort = FALSE)

# Reassign rownames
rownames(golub.merge) = golub.merge[,1]

# And finally, remove the 'Row.names' column
golub.merge <- golub.merge[,-1]
head(golub.merge[,1:5])
```

```
##                    X1 call   X2 call.1    X3
## AFFX-BioB-M_st    -41    A   19      A    19
## AFFX-BioB-3_st   -831    A -743      A -1135
## AFFX-BioC-5_st   -653    A -239      A  -962
## AFFX-BioC-3_st   -462    A  -83      A  -232
## AFFX-BioDn-5_st    75    A  182      A   208
## AFFX-BioDn-3_st   381    A  164      A   432
```

```
# And if we want to add back in gene names
golub.merge.named <- merge(golub.names, golub.merge,
by = 'row.names', all.y = TRUE)
rownames(golub.merge.named) = golub.merge.named[,1]
golub.merge.named <- golub.merge.named[,-1]

# Have a quick look at what we've ended up with
golub.merge.named[1:3,1:6]
```

```
##                                                    x  X1 call  X2 call.1
## A28102_at   GB DEF = GABAa receptor alpha-3 subunit 151    A 263      P
## AB000114_at                              Osteomodulin  72    A  21      A
## AB000115_at                                      mRNA 281    A 250      P
##              X3
## A28102_at    88
## AB000114_at -27
## AB000115_at 358
```

6.1.3 Linking in reference data

```
# There are at least two ways of doing this exercise, but both need to
# read the file in first.
species.annot = read.csv(file.path(DATADIR, "species-status-codes.csv"))

# Method one - create look-up vectors and use cbind
# Create a vector of the nca.status descriptions, named by code
nca.status <- species.annot[(species.annot[,"Field"] == "NCA_status"), 3]
names(nca.status) <- species.annot[(species.annot[,"Field"] == "NCA_status"), 2]

# Create a vector of the NCA descriptions by looking up the description
# for the NCA_status column value. Note the 'as.character' to convert
# NCA_status from a factor integer to a text string
NCA.description <- nca.status[as.character(species[,'NCA_status'])]

# Then repeat the whole process for Endemicity
endemicity.status <- species.annot[(species.annot[,"Field"] == "Endemicity"), 3]
names(endemicity.status) <- species.annot[(species.annot[,"Field"] == "Endemicity"), 2]
Endemicity.description <- endemicity.status[as.character(species[,'Endemicity'])]

# Finally, cbind all the data together
species.annotated <- cbind(species, NCA.description, Endemicity.description)
head(species.annotated[,1:5])
```

```
##   Taxon_Id Kingdom      Class          Family              Scientific_name
## 1      706 animals amphibians Limnodynastidae              Adelotus brevis
## 2      687 animals amphibians Limnodynastidae        Kyarranus kundagungan
## 3      688 animals amphibians Limnodynastidae         Kyarranus loveridgei
## 4      690 animals amphibians Limnodynastidae         Lechriodus fletcheri
## 5      677 animals amphibians Limnodynastidae Limnodynastes convexiusculus
## 6      678 animals amphibians Limnodynastidae      Limnodynastes dumerilii
```

```
# The second way is to use merge, and do some tidying up on the way
# First, pull out just the NCA_status entries from the annotation file
nca.status <- species.annot[species.annot[, "Field"] == "NCA_status",]
species.nca <- merge(species, nca.status,by.x = "NCA_status", by.y = "Code",
                     all.x = TRUE, sort = FALSE)

# Remove the unnecessary new 'Field' column
species.nca[, "Field"] <- NULL

# And the same for the endemicity
endemicity.status <- species.annot[species.annot[, "Field"] == "Endemicity",]
species.annotated <- merge(species.nca, endemicity.status,by.x = "Endemicity",
                           by.y = "Code", all.x = TRUE, sort = FALSE)
species.annotated[, "Field"] <- NULL

# Finally, rename the final two (description) columns to something useful
colnames(species.annotated)[12:13] <- c("NCA.description","Endemicity.description")
```

6.2 Summarising data

6.2.1 Table summaries

```
# To generate a table with the full descriptions
species.description.table <- with(species.annotated,
table(NCA.description, Endemicity.description))
head(species.description.table[,1:2],10)
```

```
##                                   Endemicity.description
## NCA.description                     Australian endemic Australian vagrant
##   Australian endemic                                 0                  0
##   Australian vagrant                                 0                  0
##   Conservation dependent                             0                  0
##   Critically endangered                              0                  0
##   Critically endangered wildlife                     0                  0
##   Endangered                                         0                  0
##   Endangered wildlife                               24                  0
##   Extinct                                            0                  0
##   Extinct in the wild wildlife                       5                  0
##   International vagrant                              0                  0
```

But this table includes endemicity under the NCA status, and vice versa. This is because when we used cbind or merge earlier, those two columns were created as factors. The values of those factors were not just those present in the list, but everything in the original annotation file (i.e. all NCA, EPBC and Endemicity descriptions).

```
# To get rid of all the irrelevent or inappropriate factor values, we can
# re-factor those two columns
species.annotated$NCA.description <- factor(species.annotated$NCA.description)
species.annotated$Endemicity.description <-
      factor(species.annotated$Endemicity.description)

# Then repeat the table generation, and check again
species.description.table <- with(species.annotated,
                                  table(NCA.description, Endemicity.description))
species.description.table[,1:2]
```

```
##                                   Endemicity.description
## NCA.description                     Australian endemic Australian vagrant
##   Endangered wildlife                               24                  0
##   Extinct in the wild wildlife                       5                  0
##   Least concern wildlife                          1968                  2
##   Near threatened wildlife                          30                  0
##   Special least concern wildlife                     4                  0
##   Vulnerable wildlife                               53                  0
```

```
# The NCA.description column is, like all factors, sorted alphabetically by default.
# To put it in a custom order, we can reassign the levels
levels(species.annotated$NCA.description)
```

```
## [1] "Endangered wildlife"            "Extinct in the wild wildlife"
## [3] "Least concern wildlife"         "Near threatened wildlife"
## [5] "Special least concern wildlife" "Vulnerable wildlife"
```

```
species.annotated$NCA.description <- factor(species.annotated$NCA.description,
                          levels(species.annotated$NCA.description)[c(2,1,6,4,5,3)])
levels(species.annotated$NCA.description)
```

```
## [1] "Extinct in the wild wildlife"   "Endangered wildlife"
## [3] "Vulnerable wildlife"            "Near threatened wildlife"
## [5] "Special least concern wildlife" "Least concern wildlife"
```

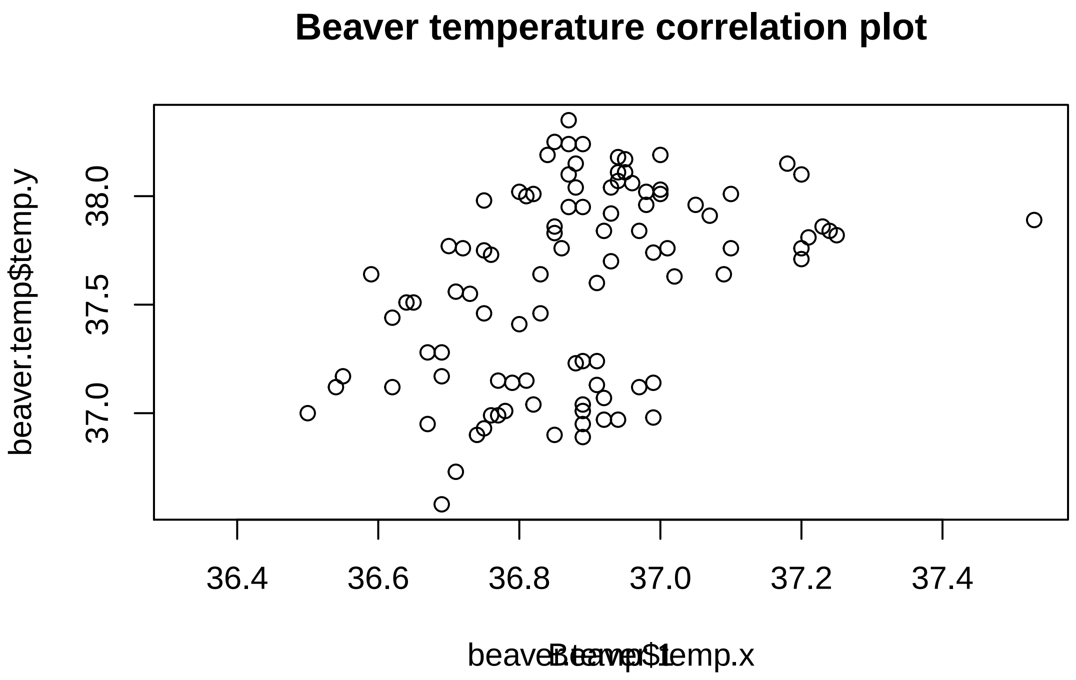
```
# And now produce the final, non-redundant and ordered table
species.description.table <- with(species.annotated,
                                  table(NCA.description, Endemicity.description))
species.description.table[,1:2]
```

```
##                                  Endemicity.description
## NCA.description                    Australian endemic Australian vagrant
##   Extinct in the wild wildlife                      5                  0
##   Endangered wildlife                              24                  0
##   Vulnerable wildlife                              53                  0
##   Near threatened wildlife                         30                  0
##   Special least concern wildlife                    4                  0
##   Least concern wildlife                         1968                  2
```

7 Charts and graphical output

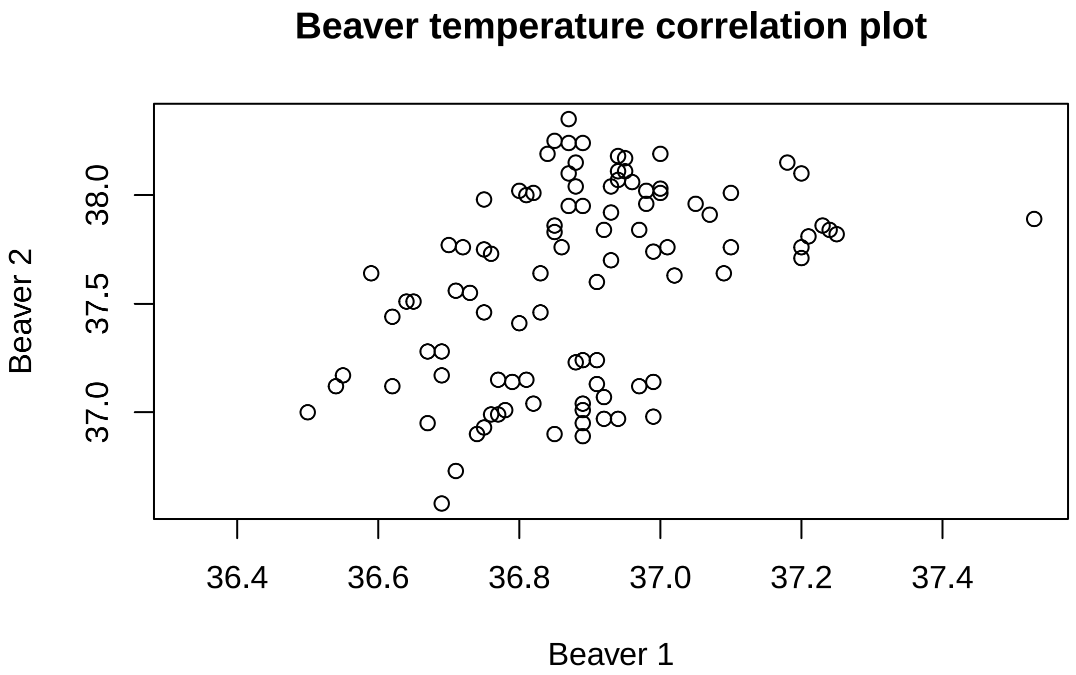
7.1.1 Plot

```
# Make a scatterplot
beaver.temp <- merge(beaver1, beaver2, by = "time", all = TRUE, sort = FALSE)
plot(beaver.temp$temp.x, beaver.temp$temp.y)

# Add a title to the plot
title(main="Beaver temperature correlation plot")

# Change the axis labels
title(xlab = 'Beaver 1')
```



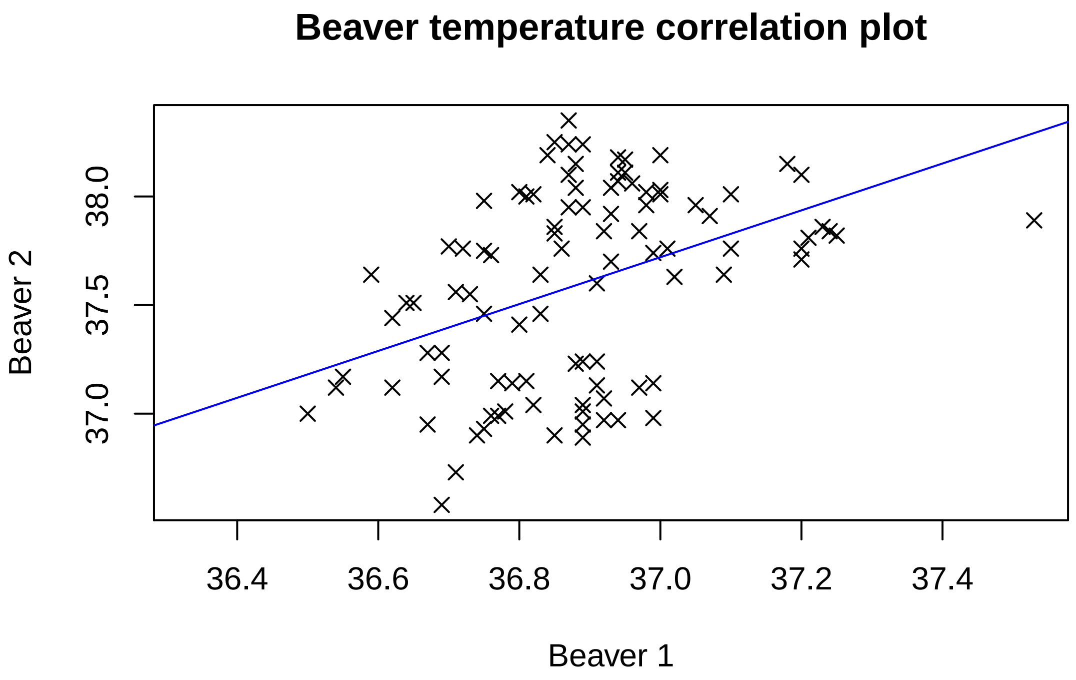
Oops, this doesn't work - it just overwrites the existing axis label

```
# Redraw the initial graph without the default labels
plot(beaver.temp$temp.x, beaver.temp$temp.y,ann = FALSE)

# Now we can add the title and axis labels
title(main = "Beaver temperature correlation plot",
      xlab = "Beaver 1", ylab = "Beaver 2")
```



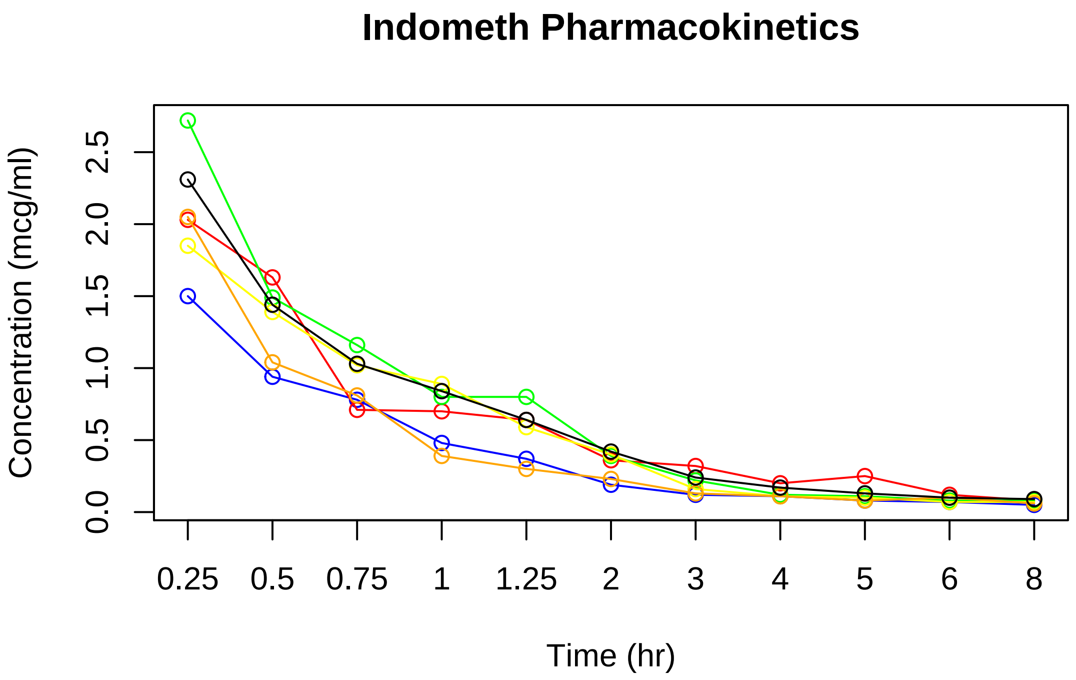
We can change point types to a ’x’ symbol, but we have to draw the graph yet again.

```
plot(beaver.temp$temp.x, beaver.temp$temp.y,ann = FALSE, pch = 4)
title(main = "Beaver temperature correlation plot",
      xlab = "Beaver 1", ylab = "Beaver 2")

# And add a trend line
lm.beaver <- lm(beaver.temp[c('temp.y','temp.x')])
abline(lm.beaver, col = 'blue')
```



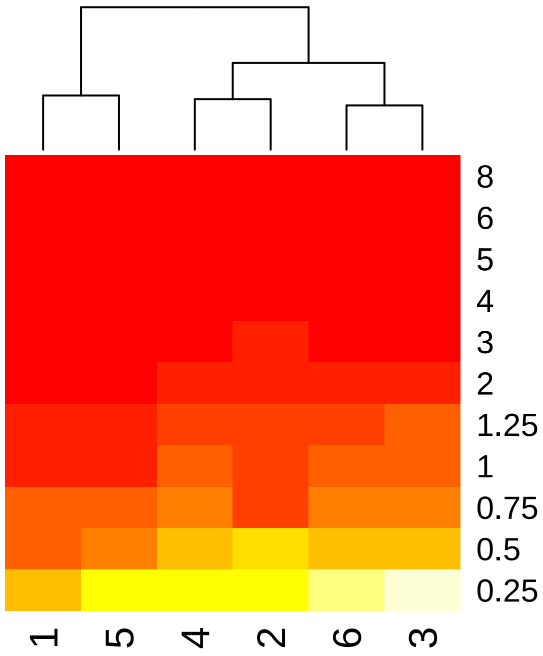
7.1.2 Further timecourse plotting practice

```
# Remember, check our range
g.range <- range(Indometh[,'conc'])

# Do the plot for the first subject (in blue)
plot(Indometh[Indometh[,"Subject"] == 1, "conc"],
     type = 'o',col = 'blue', ylim = g.range,
     xlab = "Time (hr)", ylab = "Concentration (mcg/ml)", xaxt = 'n')

# Then add the others
lines(Indometh[Indometh[,"Subject"] == 2, "conc"], type = 'o', col = 'red')
lines(Indometh[Indometh[,"Subject"] == 3, "conc"], type = 'o', col = 'green')
lines(Indometh[Indometh[,"Subject"] == 4, "conc"], type = 'o', col = 'yellow')
lines(Indometh[Indometh[,"Subject"] == 5, "conc"], type = 'o', col = 'orange')
lines(Indometh[Indometh[,"Subject"] == 6, "conc"], type = 'o', col = 'black')

# Then the x-axis scale and title
axis(1, at = 1:11, lab = Indometh[1:11, "time"])
title(main = "Indometh Pharmacokinetics")
```

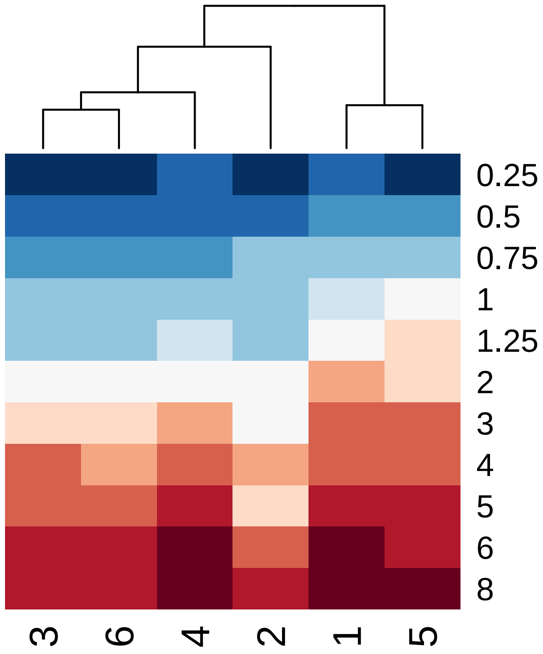


7.1.3 Heatmaps

```
library(RColorBrewer)

# Set up the matrix for our data
# This takes concentration data from Indometh 11 rows at a time
indo.matrix = matrix(Indometh$conc, nrow = 11)

# Then apply Subject number as column names and timepoints as rownames
colnames(indo.matrix) = 1:6
rownames(indo.matrix) <- Indometh$time[1:11]

# Then plot the heatmap.
# Rowv = NA keeps timepoints in order
# Scale = 'none' stops R trying to rescale colours
heatmap(indo.matrix, Rowv = NA, scale = 'none')
```



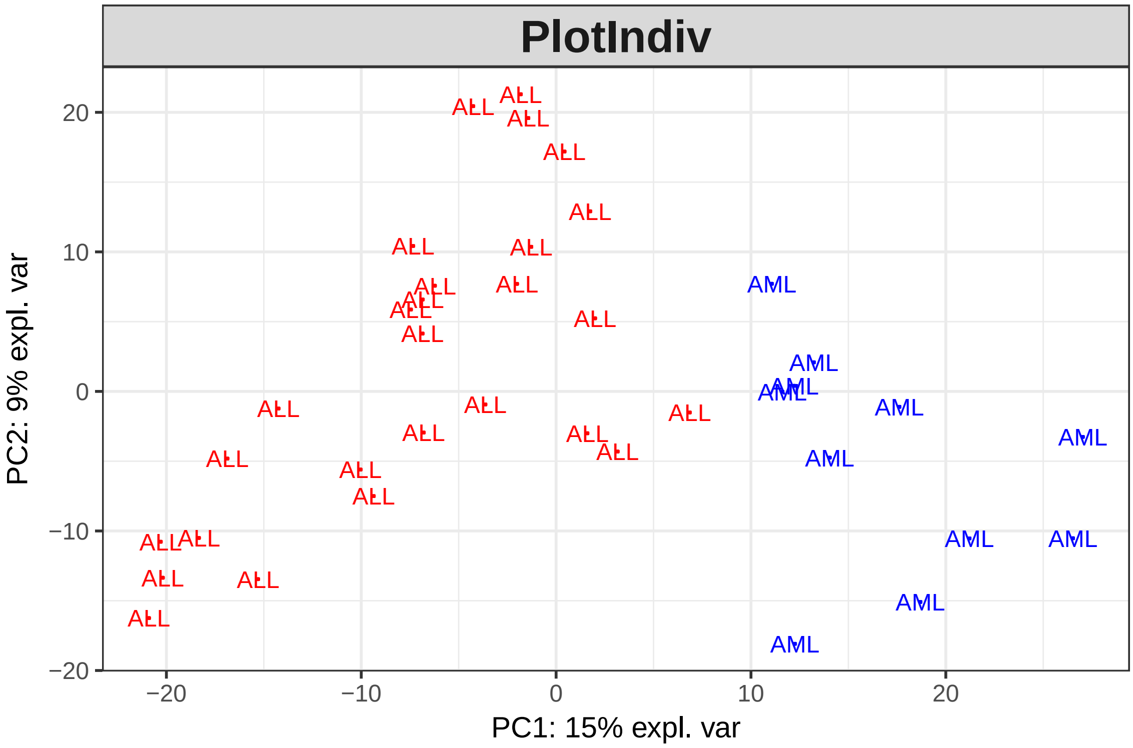
```r
# But it still has a few problems. The time points are last to first, the
# colours are ugly, and everything from 2h on is the same colour
heatmap(log(indo.matrix[11:1, ]),scale=c("none"), Rowv = NA, col = brewer.pal(11,name='RdBu'))
```



```r
# That's better - reverse the row order of the matrix supplied, log
# transform the data, and use a red-blue scaling
```

7.1.4 PCA plots

```
# Carry out the PCA analysis. Don't forget to transpose the dataset
golub.full.pca <- pca(t(golub.norm))

# Then plot the results
plotIndiv(golub.full.pca, comp = c(1,2),col = c(rep('red', 27), rep('blue', 11)))
```

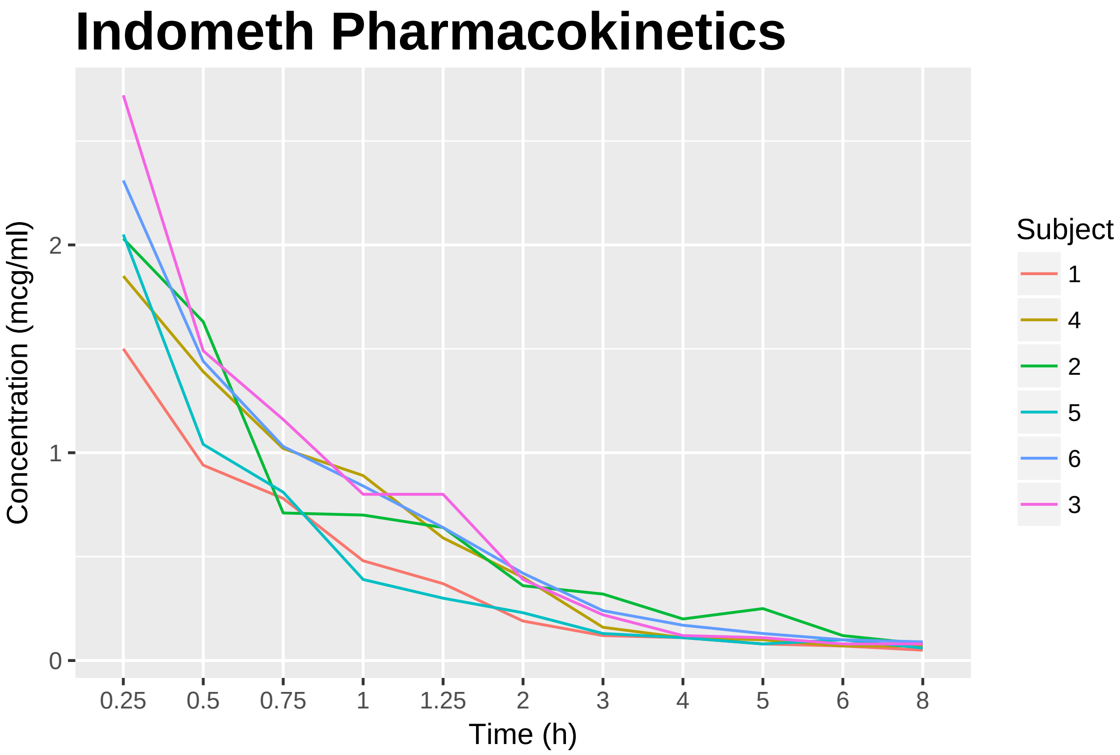


Looking at the outcome from both this plot and the selected one hundred genes, there is a clear separation between the ALL and the AML samples. This is a good indication that there is a real biological difference between the two categories which has been captured in the expression data

7.2 ggplot2

Draw the Indometh graph using ggplot2:

```
# First, activate the ggplot2 library
library(ggplot2)
ggplot(Indometh, aes(factor(time), conc, col = Subject, group = Subject)) +
    geom_line() +
    labs(x = "Time (h)", y = "Concentration (mcg/ml)",
         title = "Indometh Pharmacokinetics") +
    theme(plot.title=element_text(size=20,face="bold"))
```



What's happening here?

Function/Argument	Description
`aes`	sets the aesthetics of the plot. In this case, that we want to plot 'time' on the x-axis (the factor), 'conc' on the y-axis, and group and colour the lines by 'Subject'
`goem_line()`	specifies that this is a line plot
`labs()`	sets the x and y axis labels and the plot title
`theme()`	in this case defines the plot title as bold 24 point
`+`	plut signs at the end of the lines tells R that the command hasn't finished and that it continues on the next line

Appendix - SessionInfo for this version of the course material

```
## R version 3.3.3 (2017-03-06)
## Platform: x86_64-redhat-linux-gnu (64-bit)
## Running under: CentOS Linux 7 (Core)
##
## locale:
##  [1] LC_CTYPE=en_AU.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_AU.UTF-8        LC_COLLATE=en_AU.UTF-8
##  [5] LC_MONETARY=en_AU.UTF-8    LC_MESSAGES=en_AU.UTF-8
##  [7] LC_PAPER=en_AU.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_AU.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] tidyr_0.6.1        mixOmics_6.1.2     ggplot2_2.2.1
## [4] lattice_0.20-35    MASS_7.3-45        RColorBrewer_1.1-2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.10     plyr_1.8.4       tools_3.3.3      digest_0.6.12
##  [5] jsonlite_1.4     evaluate_0.10    tibble_1.3.0     gtable_0.2.0
##  [9] igraph_1.0.1     shiny_1.0.1      DBI_0.6-1        rstudioapi_0.6
## [13] yaml_2.1.14      parallel_3.3.3   stringr_1.2.0    dplyr_0.5.0
## [17] knitr_1.15.1     htmlwidgets_0.8  rprojroot_1.2    grid_3.3.3
## [21] ellipse_0.3-8    R6_2.2.0         rmarkdown_1.4    bookdown_0.3
## [25] reshape2_1.4.2   corpcor_1.6.9    magrittr_1.5     codetools_0.2-15
## [29] backports_1.0.5  scales_0.4.1     htmltools_0.3.5  assertthat_0.2.0
## [33] mime_0.5         colorspace_1.3-2 xtable_1.8-2     httpuv_1.3.3
## [37] labeling_0.3     stringi_1.1.5    lazyeval_0.2.0   munsell_0.4.3
```