Babraham Bioinformatics

Exercises:
Plotting Complex Figures Using R

Version 2017-11

Licence

Exercise 1: Customising simple plots

- The file `weight_chart.txt` contains data for a growth chart for a typical baby over the first 9 months of its life. Read this file into a data frame using the `read.delim` function. Use the `plot` function to draw this as a point and line graph (`type="b"`) with the following changes. Apply each one sequentially so you can see the effect it has on the plot.

    - Change the point character to be a filled square (`pch=15`)
    - Change the plot point size to be 1.5x normal size (`cex=1.5`)
    - Change the line thickness to be twice the default size (`lwd=2`)
    - Change the y-axis to scale between 2 and 10kg (`ylim=c(2,10)`)
    - Change the x-axis title to be Age (months) (`xlab="Age (months)"`)
    - Change the y-axis title to be Weight (kg) (`ylab="Weight (kg)"`)
    - Add a suitable title to the top of the plot (`main="Some title`")

- The file `feature_counts.txt` contains a summary of the number of features of different types in the mouse GRCm38 genome. Load this into a data frame using `read.delim` and then plot it as a `barplot`. Note that the data you need to `barplot` is contained within the `Count` column of the data frame, so you pass only that single column as the data for the plot.

  Once you have the basic plot make the following changes:

    - The bars should be horizontal rather than vertical (`horiz=TRUE`).
    - The count axis should be labelled (`xlab="A title"`)
    - The feature names should be added to the y axis. (set `names.arg` to the `Feature` column of the data frame)
    - The plot should be given a suitable title (`main="Some title"`)
    - The text labels should all be horizontal (`las=1`) Note that you can pass this parameter either via `par`, or as an additional option to `barplot`.
    - The margins should be adjusted to accommodate the labels (`par mar` parameter). You need to supply a 4 element vector for the bottom,left,top and right margin values. Look at the value of `par()$mar` to see what the default values are so you know where to start. Note that you will have to redraw the `barplot` after making the changes with `par`.

- **[Extension if you have time**] Use this `hist` function to plot out the distribution of 10000 points sampled from a standard normal distribution (`rnorm`) along with another 10000 points sampled from the same distribution but with an offset of 4.
    - Example: `c(rnorm(10000),rnorm(10000)+4)`
    - Find a suitable number of breaks to make the plot look nicer (`breaks=10` for example)

Exercise 2: Using colour

- The file `male_female_counts.txt` contains a time series split into male and female count values.
    - Plot this as a `barplot`
    - Make all bars different colours using the `rainbow` function
        - `rainbow` takes a single argument, which is the number of colours to generate, eg `rainbow(10)` Try making the vector of colours separately before passing it as the col argument to `barplot` (`col=rainbow(10)`).
        - Rather than hard coding the number of colours, think how you could use `nrow` to automatically generate the correct number of colours for the size of dataset.
    - Replot, and make the bars for the males a different colour to those for the females. In this case the male and female samples alternate so you can just pass a 2 colour vector to the `col` parameter to achieve this effect. (`col=c("blue2","red2")`) for example.

- The file `up_down_expression.txt` contains an expression comparison dataset, but has an extra column which classifies the rows into one of 3 groups (up,down or unchanging). Plot this as a scatterplot (`plot`) with the **up being red, the down being blue and the unchanging being grey**.
    - Read in the file using `read.delim`
    - Start by just plotting the `Condition1` column against the `Condition2` column in a `plot`
    - Pass the `State` column as the `col` parameter (`col=up.down$State` for example). This will set the colour according to the state of each point, but the colours will be set automatically from the output of `palette` Run `palette()` to see what colours are there initially and check that you can see how these relate to the colours you get in your plot.
    - Run `levels()` on the '`State`' column data and match this with what you saw in `palette()` to see how each colour was selected. Work out what colours you would need to put into `palette` to get the colour selection you actually want.
    - Use the `palette` function to set the corresponding colours you want to use (eg `palette(c("red","green","blue"))` – but using the correct colours in the correct order.
    - Redraw the plot and check that the colours are now what you wanted.

- The file `colour_to_value_map.r` contains a function to map a value from a range to a colour from a predefined palette. The file `expression_methylation.txt` contains data for gene body methylation, promoter methylation and gene expression.
    - Draw a scatterplot (`plot`) of the `promoter.meth` column against the `gene.meth` column.
    - Now you can work on colouring this plot by the `expression` column. For this you are going to need to run the `map.colours` function we provided. This requires two bits of

data – one is simply the values in the expression column, but the other is a vector of colours which define your colour scale.

- You will need to start by constructing a colour palette function from grey to red.
  - Run `colorRampPalette(c("grey","red"))` to show that you can generate a function which will make colours running from grey to red.
  - Call the function to generate a set of 100 colours and save these into a suitably named variable ie: `colorRampPalette(c("grey","red"))(100)`.
- Now generate your actual colour vector you're going to use. Call the `map.colours` function you've been given with the set of expression column values, and the vector of colours you just generated. Save the per point colours which are returned into a new variable.

- Finally redraw the plot passing the per point colours to the `col` parameter.

Exercise 3: Using Overlays

- The file `chromosome_position_data.txt` contains positional count data for 3 different datasets (a WT and two mutants). Plot this as a line graph (using `plot(type="l")`) showing the 3 different datasets overlaid.
    - You'll need to do an initial `plot` with `type="l"` specified followed by two additional layers using the `lines` function. The x values will be the `Position` column in each case. The y values will be the `WT` column for the initial plot, and then the `Mut1` and `Mut2` columns for the overlays.
    - Remember to calculate the full `range` of values across all 3 datasets when doing the initial plot so that all of the data will fit into the plot area. You can pass the whole data portion of the data frame (excluding the positions) to range to do the calculation ie: `range(chr_positions[,2:4])`
    - For the colours generate a 3 colour palette from the `"Set1" RColorBrewer` set using the `brewer.pal(3,"Set1")` function and save it. Pass a different colour from the palette to each of the plotting functions you call.
    - Use the `legend` function, to put a legend at the `"topleft"` of the plot with the data names and the corresponding colours (using the `fill` parameter).
    - Make the lines 2x as thick as standard (`lwd=2`)
    - Add suitable labels to each axis (using `xlab` and `ylab`).
    - The general structure for your code will therefore be:

```
plot(
      [position data], [wt data],
      ylim=[range data],
      type="l",
      col=[first colour],
      lwd=2
     )

lines([position data], [mut1 data], col=[second colour], lwd=2)
lines([position data], [mut2 data], col=[third colour] , lwd=2)

legend("topleft",c("wt","mut1","mut2"),fill=[vector of 3 colours])
```

- The file `brain_bodyweight.txt` contains data for the log10 brain and bodyweight for a range of species, along with an SEM measure for each point. Plot these data on a scatterplot (using `plot`) with error bars showing the mean +/- SEM and the names of the datasets under each point.

    - You will initially need to do a `plot` passing the `Bodyweight` and `Brainweight` columns as the data. Make sure this works before trying to add error bars.
    - You will need to add 2 sets of error bars, one for the `Bodyweight` and one for the `Brainweight`. These will be two separate additional layers, each consisting of an `arrows` function.
    - The arguments for each `arrows` function are:
        - The x (brainweight) start position
        - The y (bodyweight) start position
        - The x (brainweight) end position
        - The y (bodyweight) end position
        - Additional common options will be
            - `angle=90` – to produce a flat head to the bars
            - `code=3` – to get a bar at both ends of the line
            - `length=0.05` – so the bars aren't too long.

    - The start and end positions for one error bar will be the original value plus and minus the corresponding SEM. For example for the brainweight confidence interval the code would be something like:

```
arrows (
   data$Bodyweight,
   data$Brainweight - data$Brainweight.SEM,
   data$Bodyweight,
   data$Brainweight + data$Brainweight.SEM,
   angle=90,
   code=3,
   length=0.05
)
```

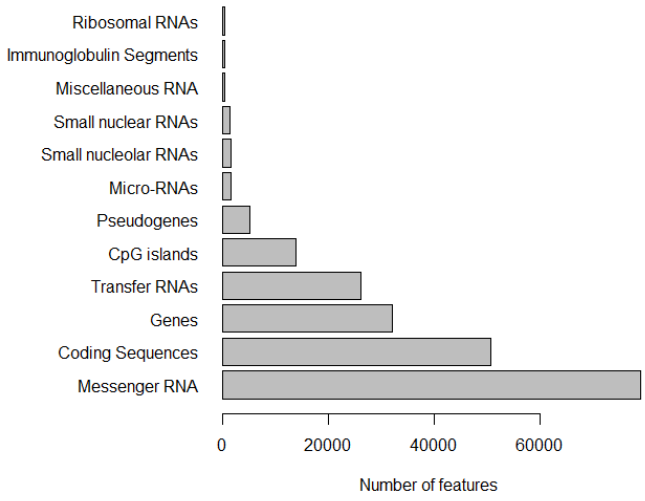
      You will, of course, need to replace "data" with whatever you actually called the data frame containing the data for this exercise. You would also need to repeat this modifying the Bodyweight values to get the Bodyweight error bars.

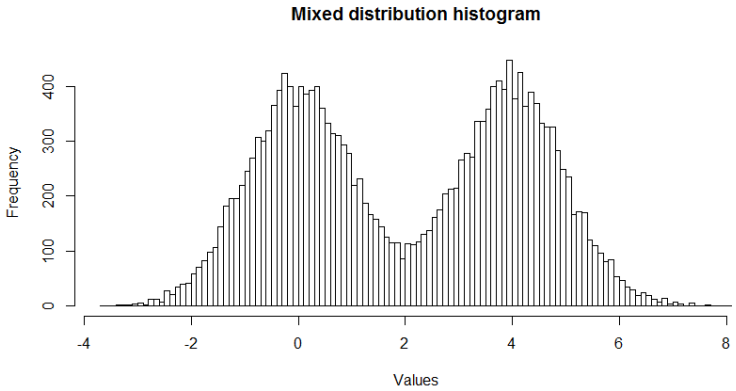
    - To add the names of the species just below each data point you would use the text function. The x and y values will just be the same as for the original plot (`Brainweight` and `Bodyweight`). The labels will be the `Species` column. Other options you will need to set will be:
        - `pos=1` – to put the text underneath each point
        - `cex=0.7` – to make the text slightly smaller

Plot Results

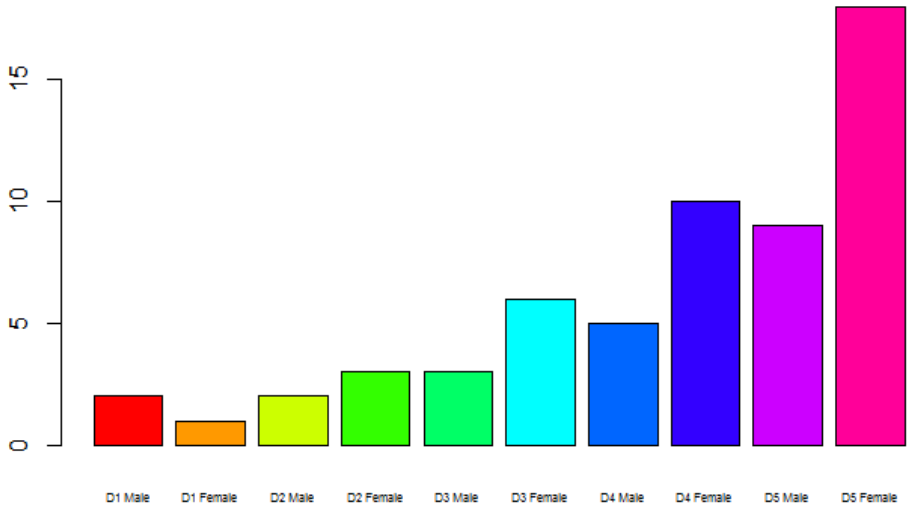
Below are examples of what the plots you will draw in these exercises should end up looking like.

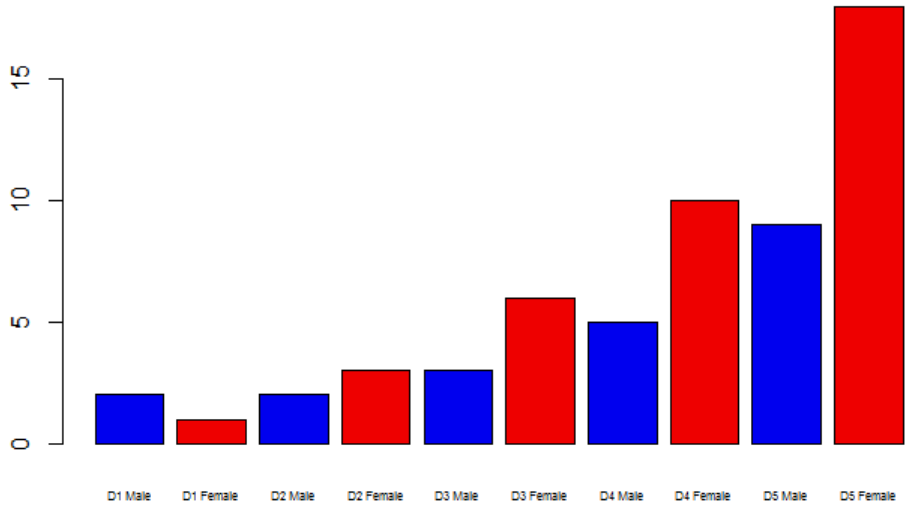
Exercise 1:

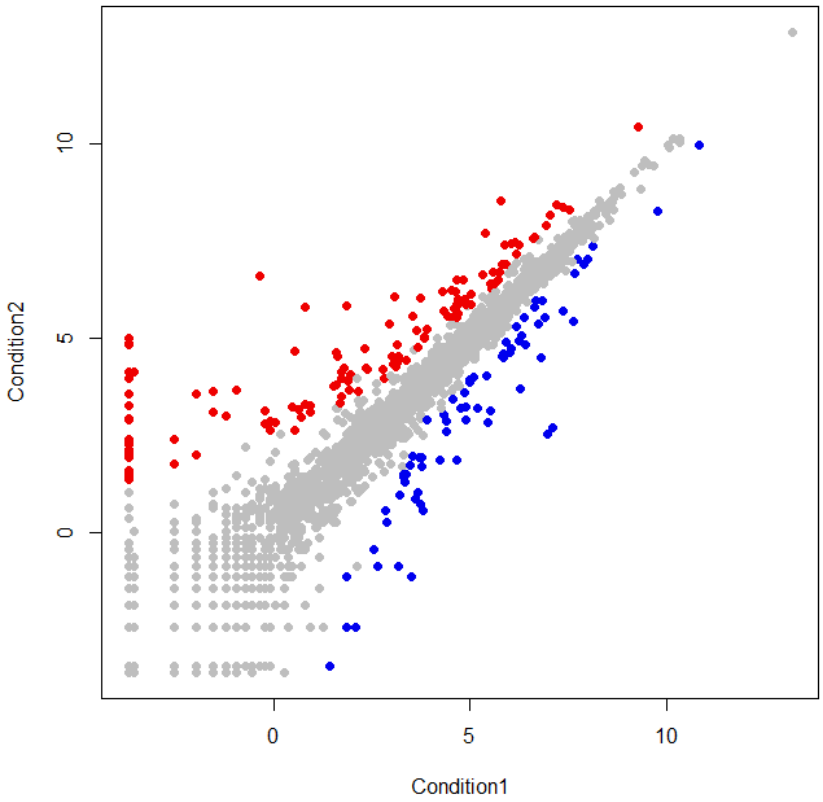


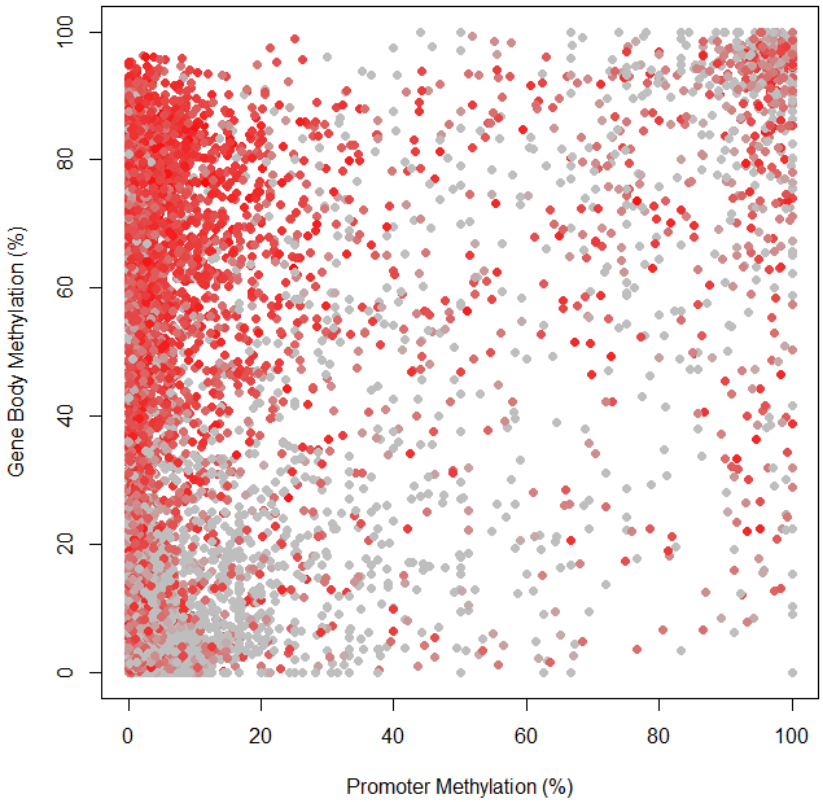


Exercise 2:

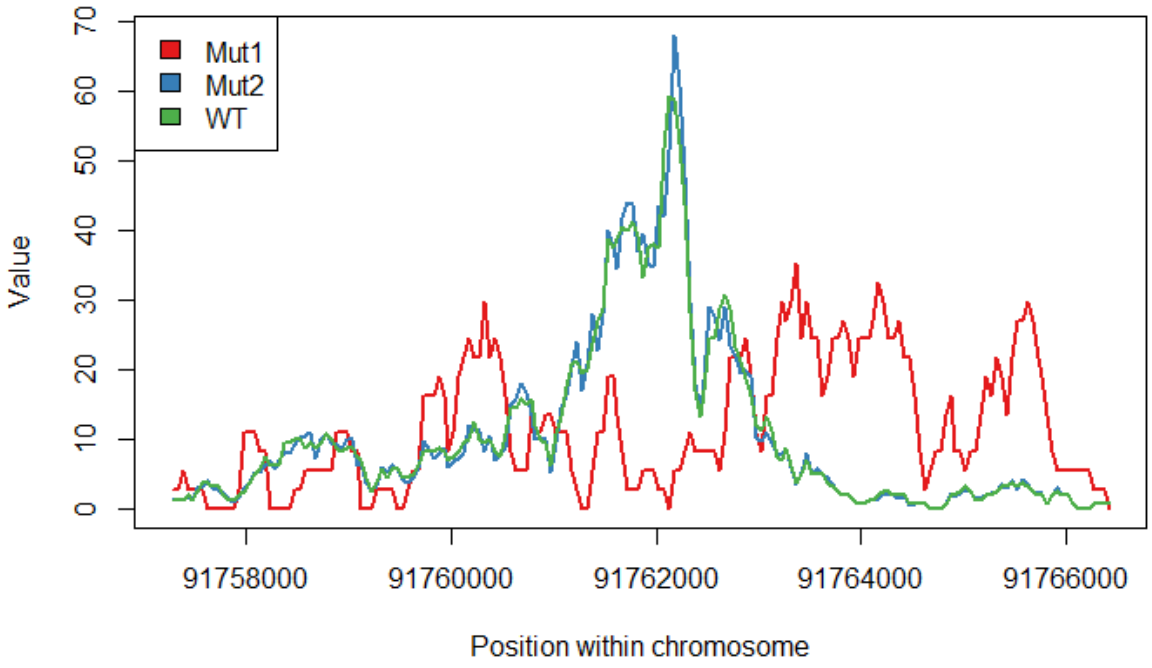

0
5
10
15
D1 Male
D1 Female
D2 Male
D2 Female
D3 Male
D3 Female
D4 Male
D4 Female
D5 Male
D5 Female

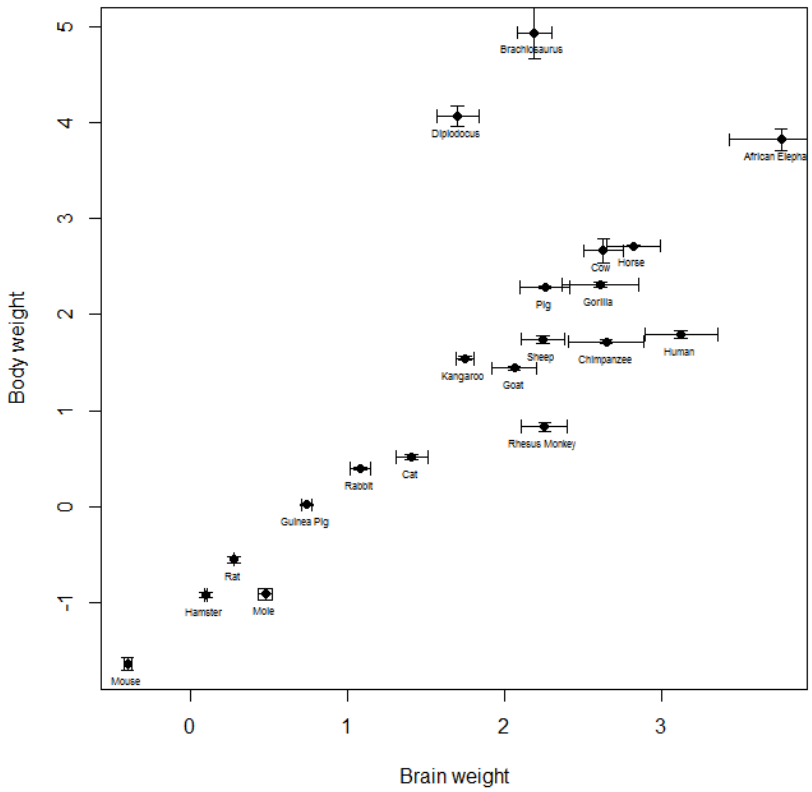

0
5
10
15
D1 Male
D1 Female
D2 Male
D2 Female
D3 Male
D3 Female
D4 Male
D4 Female
D5 Male
D5 Female


Condition2
0
5
10
0
5
10
Condition1


Gene Body Methylation (%)
0
20
40
60
80
100
0
20
40
60
80
100
Promoter Methylation (%)

Exercise 3:


Mut1
Mut2
WT
Value
0
10
20
30
40
50
60
70
91758000
91760000
91762000
91764000
91766000
Position within chromosome


Body weight
Brain weight
-1
0
1
2
3
4
5
0
1
2
3
Brachiosaurus
Diplodocus
African Elepha
Cow
Horse
Pig
Gorilla
Sheep
Chimpanzee
Human
Kangaroo
Goat
Rhesus Monkey
Rabbit
Cat
Guinea Pig
Rat
Hamster
Mole
Mouse