Babraham Bioinformatics

Exercises: Introduction to R

Version 2018-09

Licence

Exercise 1: Simple Calculations

- Use R to calculate the following:
  - 31 * 78
  - 697 / 41
- Assign the value of 39 to `x`
- Assign the value of 22 to `y`
- Make `z` the value of `x - y`
- Display the value of `z` in the console
- Calculate the square root of 2345, and perform a log2 transformation on the result.

Exercise 2: Working with Vectors

- Create a vector called `vec1` containing the numbers 2,5,8,12 and 16
- Use [lower]`:`[upper] notation to make a second vector called `vec2` containing the numbers 5 to 9
- Subtract `vec2` from `vec1` and look at the result
- Use `seq()` to make a vector of 100 values starting at 2 and increasing by 3 each time and store it in a new variable
- Extract the values at positions 5,10,15 and 20 in the vector of 100 values you made
- Extract the values at positions 10 to 30 in the vector of 100 values you made

Exercise 3: Lists and Data Frames

- Enter the following into a vector with the name `mouse.colour`. Remember to surround each piece of text with quotes.

  purple
  red
  yellow
  brown
- Display the 2nd element in the vector (red) in the console.
- Enter the following into a vector with the name `mouse.weight`:

  23
  21
  18
  26
- Join the 2 vectors together using the `data.frame` function to make a data frame named `mouse.info` with 2 columns and 4 rows. Call the first column 'colour' and the second one 'weight'.
- Display the data frame in the console.
- Display just row 3 in the console
- Display just column 1 in the console
- Display the item of data in row 4, column 1.

Exercise 4: Reading in data from a file

Set your working directory to where the data files are stored. Make sure that the folder of data files has been unzipped. e.g. `setwd("D:/Data_folder")`

4a

- Read the file 'small_file.txt' into a new data structure. This is a tab delimited file so you should use `read.delim()`. Remember to assign a name to the data that you read in using the assignment operator, e.g.

```
my.small.file <- read.delim("small_file.txt")
```

- View the data set to check that it has imported correctly.

4b

- Read the file 'Child_Variants.csv' into a new data structure. This is a comma separated file so you should use `read.csv()`. Again, remember to assign a name to the data when you import it.
- Use `head` and `View` to look at the data set to check that it has imported correctly.
- Calculate the `mean` of the column named `MutantReadPercent.` Think about how you are going to access a single column first (probably by using the `$` notation), then once you can access the data pass it to the `mean` function.

Exercise 5: Filtering

For all filtering think of the problem in a structured way:

1. Extract the vector of values you want to filter against
2. Apply the logical test
3. Either use the logical vector produced to filter the data using a selection, or use sum() to get the count of the hits.

5a

- Create a filtered version of the child variants dataset which only includes rows where the `MutantReadPercent` is `>=70`. Save this under a new name. The process you should use is:
  - Make sure you can access the `MutantReadPercent` column from the data frame
  - Do a logical test on this data for `>=70` to obtain a logical vector
  - Use this logical vector as the row selector to filter the original dataset. Remember that you need to say which rows AND columns you want, so as you want all columns you'll need to put a comma after the logical vector in your data frame filter.

5b

- From the filtered data frame (the one you created in 5a) we want to know how often G, A, T and C bases were mutated. You will need to use an exact match filter (`==`) against the text in the `REF` column to match each of the bases in turn. The process you should use is:
  - Make sure you can access the `REF` column from the data frame.
  - Do a logical test on this data to obtain a logical vector. Remember to use the `==` operator, and to put quotes around the text that you are matching.
  - To get the count you can `sum()` the logical vector values (true counts as 1, false counts as 0).
  - Repeat these steps for A, T and C.
  - Create a vector called `mutation.counts` of the counts for the different mutations (you will need to make this manually using `c()` having calculated the values).

Exercise 6: Histograms, Boxplots and Barplots

- In the original child variants dataset draw a histogram (`hist`) of the `MutantReadPercent` column, try increasing the number of categories (`breaks`) to 50.
- Plot a `boxplot` of the `MutantReadPercent` values from both the original child variants and the same column from the filtered dataset you made in Exercise 5 (`MutantReadPercent` >=70). Check that the distributions look different. To do this you can just pass two vectors to the boxplot function.
- Plot the results of the vector created in Exercise 5 (`mutation.counts`) as a `barplot`. Use the `names.arg` argument to show which mutation is which and make a vector of colours (`c("black","green","red","blue")`) to give the bars different colours (using the `col` parameter).

[Optional] Exercise 7: Scatterplots and Line graphs

- Read in the file `'neutrophils.csv'`. This is a comma-delimited file so use `read.csv()`.
- Create a `boxplot` of the 4 samples and put a suitable title on the plot (you should be able to pass the entire data frame to `boxplot` since you're plotting all of the data there).
- Use the `colMeans()` function to calculate the mean of each dataset. Note: the data contains missing values (NA) so look at the help file for `colMeans` to find out how to ignore these. Plot the means as a `barplot`.
- Read in the file `'brain_bodyweight.txt'`. This is a tab delimited file so you can use `read.delim()`. The first column contains species names, not data, so use `row.names=1` to set these up correctly in your data frame.
- Log transform the data (base 2).
- Create a scatterplot with default parameters with the log transformed data.
- Read in the file `'chr_data.txt'`.
- Remove the first column (you can simply select the second and third columns and then overwrite the original variable)
- Create a line graph like the one below. The process will be:
    - Use `plot()` to plot the genome position vs the ABL1 dataset. You will need to set the `type` parameter to `l` (lower case L) to get this to be a line graph.
    - Use `legend` to add in a legend to the plot, and `title` to add a title to the plot (you could also do this by adding a main parameter to the original plot function)