Babraham Bioinformatics

Exercises: Introduction to R

(With Tidyverse)

Version 2024-03

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.

- Put your name into a variable called `my_name`
- Use the `nchar()` function to find out how many characters are in your name
- Use the `substr()` function to extract the first character of your name

Exercise 2: Working with Vectors

Making vectors and doing vectorised maths

- Create a vector called `some_numbers` containing the numbers 2,5,8,12 and 16
- Use the colon operator to make a vector called `number_range` containing the numbers 5 to 9
- Subtract (using the `-` operator like any other subtraction) `number_range` from `some_numbers` and look at the result

Vector creating functions

- 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 called `number_series`
- Multiply every value in `number_series` by 1000 and overwrite the original variable
- Use `rep()` to make a vector of 100 values containing 25 each of `WT`, `KO1`, `KO2` and `KO3`

Statistical functions to create and test vectors

- Use the `rnorm()` function to generate a set of 20 numbers randomly sampled from a normal distribution with `mean=0` and `sd=1` (the default values) save these into a variable called `normal_numbers`
- Pass normal numbers to the `t.test()` function to see whether they differ significantly from having a mean of 0 (look for a p-value of less than 0.05)
- Replace `normal_numbers` with a new set of 20 numbers from a distribution with a mean of 1. Test this with `t.test()` and see if the result is now significant.

If you have time:

- Median centre the values in `number_series`. Calculate the `median` of all of the values and then subtract this from the individual values. What is the `mean` and standard deviation (`sd()`) of this modified dataset?
- Create two vectors of values, one drawn from a normal distribution with `mean=10` and `sd=2` and the other from a normal distribution of `mean=10.1` and `sd=2`. How many values do you need to put in each vector before you can reliably tell that the two means are not equal using a `t.test`?

Exercise 3: Reading Data from Files

Make sure that the `tidyverse` packages are installed on your machine. If they're not then install them using:

```
install.packages("tidyverse")
```

Set your working directory to where the course data files are stored using
RStudio Session > Set Working Directory > Choose Directory

Make sure that you have tidyverse loaded into your session by running:

```
library(tidyverse)
```

Reading a small file

- Read the file '`small_file.txt`' into a new data structure. This is a tab delimited file so you should use `read_delim()` to load it.
- When entering the file name – make sure you use the Tab key to list the files in the working directory and select the correct one rather than typing out the whole file name.
- Remember that `read_delim()` will load the file but it won't save it for you. You need to use the arrow operator to save it to an R variable so that you can work with it afterwards, the same way you did with all of the other variables you have created.
- View the data set to check that it has imported correctly. Check that the guessed column types are correct.
- Find the median of the `log2()` transformed lengths from this file.
  - You will need to use the `$` notation to extract a single column to a vector
  - You will need to use the `log2()` function to transform the values and then take the result of this into the `median()` function

Reading a larger file

- Read the file '`Child_Variants.csv`' into a new data structure. This is a comma separated file but you should still use `read_delim()`. Again, remember to assign a name to the data when you import it.
- Display the new data into the console and see what it looks like. Try clicking on the data in the Environment tab to run `View()` to open it in the viewer pane.
- Calculate the `mean()` and `sd()` of the column named `MutantReadPercent`.

Exercise 4: Filtering

Use the tidyverse selection and filtering tools to achieve the following selections in your data. You don't need to save each of the results below but you should be able to look at them to show that the filtering has worked correctly.

In the small file data:

Each of the lines below is a separate selection, you don't need to combine them at this stage.

- Extract all of the rows where the `Category` is "A"
- Extract the data for samples with a `Length` of more than 80
- Remove the `Sample` column to just leave just `Length` and `Category`

In the child variants data:

Again, each of the lines below is a separate selection; you don't need to combine them.

- Extract the rows where the chromosome is mitochrondrion ("MT")
- Extract the variants which have a `MutantReadPercent` of >= 70
- Extract the variants with a quality of exactly 200 (look at the file to see the correct column name)
- Extract all variants in the IGFN1 gene
- Remove the `ENST` and `dbSNP` columns

If you have time:

Load the file `broken_selects.R` from your data folder using File > Open File in RStudio. This is a small script which replicates some of the exercise you've just done, but in which there are numerous errors. Work your way through the script fixing the errors. Make sure that you've sanity checked the results from every line, before moving on to the next one.

Exercise 5: Multi-Stage filtering and Selection

All of these selections should be made on the child variants data. As before, you don't need to save the results of any of them.

Quality filtering

Extract variants where all of the conditions below are true:

- Quality is 200
- Coverage is more than 50
- Mutant Read Percent is more than 70

Positional Filtering

- Remove all variants on the X, Y and MT chromosomes

Annotation filtering

- Show only the chromosome and position for variants which have a valid dbSNP ID

Transformation filtering (if you have time)

In some more advanced filtering you might want to manipulate the data when you filter it to be able to do something more clever than you can do with standard tests. Below we're going to give you some examples where you can see how this would work.

Earlier you used the `nchar()` function which calculates the number of characters in each element of a character vector. Use this on the REF column in a `filter()` statement to select variants where there is a deletion, ie where the number of characters in the reference is more than 1.

We saw the `substr()` function at the start of the day. It allows us to extract parts of a string of characters. Use the `substr()` function in a `filter()` statement to find all variants in genes whose name starts with Q.

Exercise 6: Your first ggplot plot

Load the data from the `brain_bodyweight.txt` file using `read_delim()`.

This file contains the details of the brainweight and the bodyweight for different animal species.

- Draw a scatterplot (using `geom_point()`) of the log brainweight (`log.brain`) vs the log bodyweight (`log.body`). When defining your aesthetics the `brain` will be the `x` and `body` will be the `y`.

- Make all of the points filled with `blue2`, and give them a size of 3.

- Change the point colouring from a fixed value of `blue` to an aesthetic mapping where the points are coloured by `Category`. For the aesthetic mapping to have any effect you'll need to remove the fixed aesthetic. What do you see about the extinct animals?

If you have time:

- Re-plot your last graph but remove the extinct species from the plot. Use your previous filtering skills to remove them from the data and then pipe the filtered data straight into `ggplot()`.

Exercise 7: More plotting

- Plot out a barplot of the lengths of each sample from category A in the small file data you loaded previously.
    - Start by filtering the data to keep only Category A samples
    - Pass this filtered tibble to `ggplot()`
    - Your `x` aesthetic will be `Sample` and your `y` will be `Length`
    - Since you are explicitly providing bar heights you need to use `geom_col()` as your geometry

- Plot out the distribution of `MutantReadPercent` values in your child variants data. You can try both `geom_histogram()` and `geom_density()`. Try changing the colour and fill parameters to make the plot look prettier. In `geom_histogram()` try changing the `binwidth` parameter to alter the resolution of the distribution.

- Plot a barplot of the frequency with which different bases are mutated. To do this you will need to take the following steps

    - Filter child variants to keep **only** positions with a **single** letter `REF` (use `nchar()` in the filter) – note that this is different to the `REF` filter you did before

    - Pass the filtered data to `ggplot()` and set `REF` as the `x` aesthetic, and set `geom_bar()` as the geometry. Since there isn't a bar height in the aesthetic, ggplot will count the number of occurrences of the different refs and plot those.

If you have time…

Improved aesthetics

Go back to the plots you have created and add appropriate titles using `ggtitle()` and axis labels using `xlab()` and `ylab()`

Change some fixed aesthetic values to the geometry calls to change the appearance of your plots. Look at the help page for the geometry you are using to see which aesthetic values you can change.

For example:

- Change the colour of plotted points (try using `colour` and `fill` and see the difference)
- Change the size of the plotted points / lines

Multiple layers

Try adding text labels to your brain bodyweight plot. To do this you'll need to add a new `geom_text()` geometry level, and you will need to define an aesthetic mapping for `label` (you already have the x and y aesthetics defined). You can add the new mapping either in the original `ggplot aes` call, or in a new `aes()` call inside `geom_text()`.

Once you have the text labels showing you can use the `hjust` parameter in `geom_text()` to move them from being directly over the corresponding points.

Statistical testing

You can see a relationship between `log.brain` and `log.body` but we can also test whether this is statistically significant and we can work out exactly how strong the relationship is.

You are going to use two different statistical functions to assess this. Although they are running similar analyses on the same data, the way they are called is quite different – this illustrates the importance of reading the documentation for functions before running them.

The `cor.test` function takes in two vectors of values (which must be the same length) as arguments and performs a Pearson correction test on the linkage between the values. It will give you a summary of the relationship, including a p-value. Run this on your `log.brain` and `log.body` columns. Look back to exercise 3 if you can't remember how to extract one column from a tibble into a vector. Note down what the p-value from the test is so you can add it to the title of your graph.

The `lm` function performs a linear model on your data. In our case we're going to use this to fit a line of best fit through the `log.brain` and `log.body` data. From the model it generates we can extract the slope and intercept values. We can then put these into a call to the `geom_abline()` geometry to draw the line of best fit onto our plot.

The documentation for `lm()` tells us that the only required parameters are:

```
lm(formula, data)
```

The data is easy, since that's our brain bodyweight tibble (with whatever name you gave it). The formula is a little different. This is a structure which defines the relationship we want to test based on the column names in the data. In our case we want to see how well `log.body` is predicted by `log.brain`. The formula is therefore:

```
log.body ~ log.brain
```

Details of building statistical models in R are beyond the scope of this introductory course, but we have a separate course on statistical analysis in R which you can attend if you want to build up your knowledge about this side of R.

Once you have successfully run lm, note the slope and intercept values and then include these in a call to `geom_abline()` which you can then add as a new layer in your brain bodyweight plot.