BIO720_Week2

Ian Dworkin

2024-09-20

Contents

In class activities for week 2

Now that you have had a bit of an introduction to the fundamentals of R, let's start working with some examples of how we can use these skills with real data.

We are going to work with the count data of RNAseq sequence reads from a large data set from a paper we recently submitted. It was part of a study of the genomic basis of the evolutionary of sociability in *Drosophila melanogaster".

The link to the data and script repository is here

But for the moment, all we need is the counts that were generated. For this particular analysis this was done by aligning the sequence reads to the *Drosophila melanogaster* genome using a *splice aware* alignment tool, STAR. While it is not relevant for the moment, if you want to read about this approach, you can take a look at this tutorial.

In any case, the count data can be downloaded like this, directly from the github repository.

```
rna_counts <- read.table("https://raw.githubusercontent.com/DworkinLab/DrosophilaSociabilityTranscripto
                         header = TRUE)
```

What we did yesterday (Sept 10th, 2024)

We first thought through some things we should check after we downloaded the data

what should we look at first?

- does the data look like it is expected
  - size of the data. Dimensions of the data?

    - names of variables?
    - how is missing data encoded?
    - what object class are we working with? Is it what we expected?
    - Is all the data there? how do we check
    - are column headers being read as headers?
        * Is the data possible?

We realized there were a variety of ways to check the number of rows (and columns of the data

Probably the most concise

```r
dim(rna_counts)
```

Has lots of other useful information, but maybe too much information in this case!

```r
str(rna_counts)
```

We can ask about how many rows and how many columns, *per se*

```r
nrow(rna_counts)
ncol(rna_counts)
```

We then checked to see if there was any missing data (encoded by `NA`)

```r
anyNA(rna_counts)
```

Let's look at the names of the variables we have

So what are we looking at?

```r
class(rna_counts)
mode(rna_counts)
```

A small but important change

Typically in R and bioconductor for genomics experiments, we have columns as samples and rows as features (transcripts, genes, SNPs, taxa, etc). How is it currently set up?

How would we make a new object (generally don't over-write the old one!) where it corresponds to this format? Write down the steps you need to take in order to do so (the pseudo-code).

Row names need to be the feature names (in this case the genes)

What steps do we need?

1. extract out gene names as a variable
2. generate a copy of the data, with a new name, without first column
3. use gene names as rownames

or

1. generate a copy of the data, with a new name, without first column
2. use gene names as rownames via extracting the gene identifier from the original data set

```r
rna_counts2 <- rna_counts[, 2:143] # or [, -1]

rownames(rna_counts2) <- rna_counts$gene

rna_counts2[1:6, 1:6]
```

extract the information about the experimental design from the column names for each sample

We used underscore "_" for our delimiter in the files

- AS: Initials of who collected the samples
- the next is for the evolutionary treatments
- Replicate lineage
- Sex
- Environment
- Unique sample
- Lane of Illumina flowcell

Pseudo code

Write pseudo-code to take the sample names and generate an object (but what kind of object?) to store all of the experimental meta-data.

- such that rows represent individual samples
- and columns experimental variables.

1. make a new variable storing all samples names as a vector. 2a. initialize a data frame to store our new information. Thankfully in R, it does it for us. 2b. separate based on the underscore delimiter
2. label the variables (columns)
3. Label the rows by their unique sample names

```r
sample_names <- colnames(rna_counts2)
class(sample_names)
```

we are going to use a function in the stringr library

```r
#install.packages("stringr") # only install once!
```

To use the library. . .

```r
library("stringr")
```

```r
sample_names_matrix <- str_split_fixed(string = sample_names,
                                       pattern = "_",
                                       n = 7)
```

```
# alternative (but can cause issues if there are problems with the delimiters)
sample_names_matrix2 <- str_split(string = sample_names,
                                  pattern = "_",
                                  simplify = TRUE)
```

For the metadata, make sure we have each experimental variable (columns) labelled, along with the sample identifiers (along each row).

```
colnames(sample_names_matrix) <- c("initials", "treatment", "lineage", "sex", "environment", "sample",

rownames(sample_names_matrix) <- sample_names
```

let's look at the data!

How many mapped reads do we have in each sample?

pseudo-code 1. Compute sum of each columns (i.e. compute sum of column 1, then column 2…)

For column 1 you could compute the sum like:

```
sum(rna_counts2[,1])
```

I don't want to do that for each column!

We could write a for loop (which I do below), but there are two "Rish" ways that are worth knowing about.

First is `colSums` which computes the sum for each column. This is generally the fastest way (both coding and computing)

```


read_sums <- colSums(rna_counts2)

length(read_sums)

read_sums
```

colSums

apply Alternatively, you could use a function like `apply` to do the same thing, by applying the `sum` function to each column. apply functions tend to be somewhat slower (but not for a tiny problem like this)

```
read_sums_alt1 <- apply(X = rna_counts2, MARGIN = 2, sum)
```

for loop In most other languages, a `for` loop would be fine and you could use it like this:

First we initialize the variable and make it the "right size" vector to pre-allocate memory to store the data we are going to generate (the sums from each column). In this case I am creating a vector with `NA`, which we will fill with the sums we compute in the next step.

```r
read_sums_alt2 <- rep(x = NA,
                      times = ncol(rna_counts2))


names(read_sums_alt2) <- colnames(rna_counts2) # just so we have the sample names
length(read_sums_alt2)

head(read_sums_alt2)
```

Then the for loop

```r
for (i in 1:ncol(rna_counts2)) {
  read_sums_alt2[i] <- sum(rna_counts2[, i])
}

rm(i) # a bit of cleanup
```

All of these achieve the same thing. A small note, for some reason using `colSums` stores it as numeric not integers.

```r
head(read_sums)
head(read_sums_alt1)
head(read_sums_alt2)
```