GEO Data Tutorial

Kyle Roell

2021-06-03

Contents

Chapter 1

Gene Expression Omnibus (GEO)

GEO is a public functional genomics data repository supporting MIAME-compliant data submissions. Array- and sequence-based data are accepted. Tools are provided to help users query and download experiments and curated gene expression profiles.

We will be using GEO to access and download data to perform various statistical and graphic procedures.

Data in this tutorial will be downloaded from GSE42394.

The entire RMarkdown code can be found on our UNC-SRP Github site.

Chapter 2

Loading and Manipulating GEO Data

The GEOquery package allows for easy reading in and downloading data from GEO. This will parse the data for you and store it into an R object that you can reference and manipulate for data analysis.

You can both read in data from a file that has manually been downloaded from GEO or let GEOquery download the data for you. Here, we will show how to simply read in a file that we have already downloaded from GEO. All we need to download is the same file that we manipulated in Method 1 above!

```
#First install GEOquery
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager");

BiocManager::install("GEOquery");

#load the package
library(GEOquery);

#Now we can use the getGEO function to load data from our series matrix
geo.getGEO.data = getGEO(filename='~/Downloads/GSE42394_series_matrix.txt');

#We can get a dataset with the sample information using the pData() function
sampleInfo <- pData(geo.getGEO.data);

#Let's look at some of our sample information dataset columns
knitr::kable(
  head(sampleInfo[,c("geo_accession", "cell type:ch1", "time:ch1",
```

Table 2.1: Sample information column names

	geo_accession	cell type:ch1	time:ch1	treatment:ch1
GSM1150937	GSM1150937	Nasal epithelial cells	7 day	unexposed
GSM1150938	GSM1150938	Nasal epithelial cells	7 day	unexposed
GSM1150939	GSM1150939	Nasal epithelial cells	7 day	unexposed
GSM1150940	GSM1150940	Nasal epithelial cells	7 day	2 ppm formaldehyde
GSM1150941	GSM1150941	Nasal epithelial cells	7 day	2 ppm formaldehyde
GSM1150942	GSM1150942	Nasal epithelial cells	7 day	2 ppm formaldehyde

```
                      "treatment:ch1")]), caption = 'Sample information column names',
  booktabs = TRUE
)
```

Now, we can use this to just define the samples we want to analyze.

We only want cell type = Nasal epithelial cells and time = 7 day. To do this, we can use our sampleInfo dataset and the variables "cell type:ch1" and "time:ch1". We will also define treated and untreated using the variable "treatment:ch1".

```
#Define keep variable that will store rows we want to keep
keep = rownames(sampleInfo[which(sampleInfo$`cell type:ch1`=="Nasal epithelial cells"
                                 & sampleInfo$`time:ch1`=="7 day"),]);

#first let's just subset sample info for just those samples we defined in keep variabl
sampleInfo = sampleInfo[keep,];

#get the treated and untreated sample ids
#(which are in the variable geo_accession)
treated  = sampleInfo[which(sampleInfo$`treatment:ch1`=="2 ppm formaldehyde"),
                      "geo_accession"];
untreated  = sampleInfo[which(sampleInfo$`treatment:ch1`=="unexposed"),
                        "geo_accession"];
```

We now have a list of the samples we want to keep, stored in our variable "keep". The next step is to actually get the data we want to use in our analyses. The GEOquery function exprs() allows us to easily get this data.

```
#We can get the actual data using the exprs() function
#and we only want to keep those we previously defined
ge.data = exprs(geo.getGEO.data[,keep]);
```

Table 2.2: Gene expression data

	GSM1150937	GSM1150938	GSM1150939	GSM1150940	GSM1150941
10700001	5786.60	5830.08	5637.34	5313.33	5557.04
10700002	192.92	206.86	220.83	183.12	177.16
10700003	1820.98	1795.79	1735.70	1578.02	1681.58
10700004	66.95	65.61	64.41	60.19	60.41
10700005	770.07	753.41	731.20	684.53	657.25

```
#Let's look at our sample information dataset
knitr::kable(
  ge.data[1:5,1:5], caption = 'Gene expression data',
  booktabs = TRUE
)
```

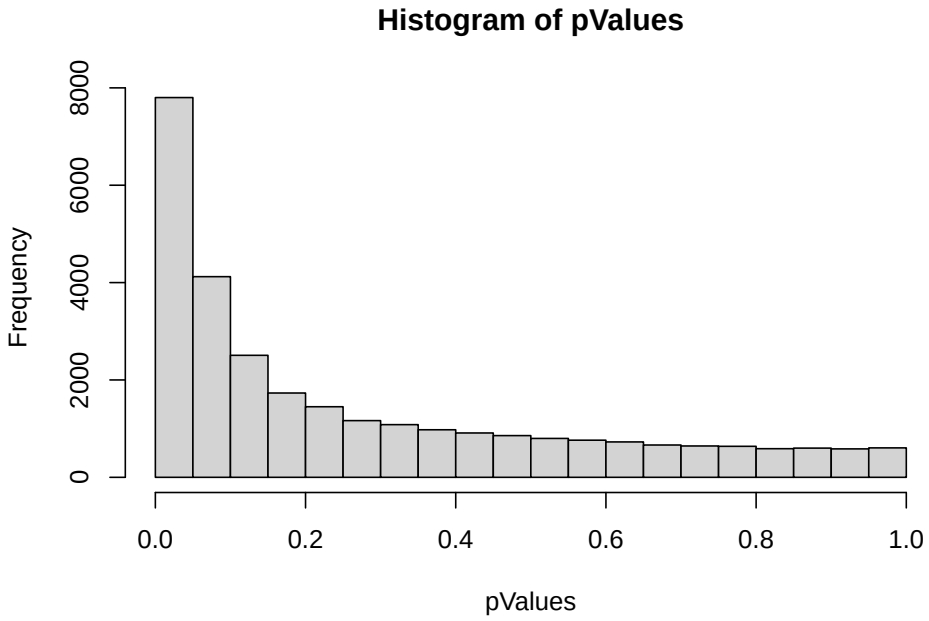
Chapter 3

Statistical Procedures

3.1 T-Test

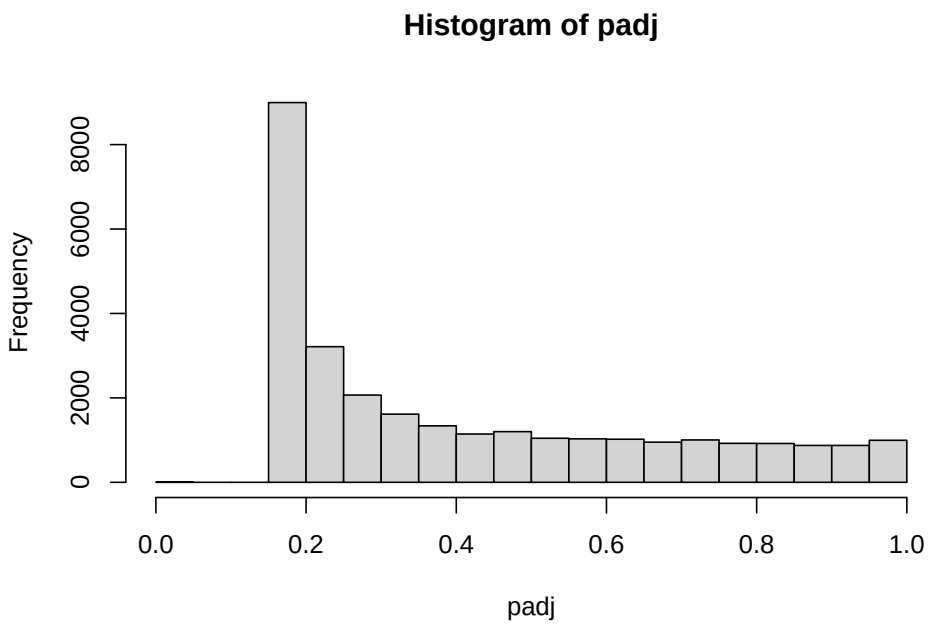
One of the most basic statistical procedures we can perform on this dataset is a set of t-tests to test for differences between the two treatment groups (exposed and unexposed) across each of the genes. One way to do this is to use a loop and test each gene individually, saving the p-value from the t-test into a separate dataframe. Another solution is to use the build in "apply" function. This function will return results obtained by applying a specified function (the "t.test" function) to the margins of our dataset.

```
#Use ge.data dataset
#use the t.test function comparing treated and untreated samples
#1 specifies to apply the function to rows
pValues <- apply(ge.data, 1, function(x) t.test(x[treated],x[untreated])$p.value);

#We can look at a histogram of the pvalues to see distribution
hist(pValues);
```



Now we have a list of p-values for t-tests for each gene comparing the treated and untreated samples. Because we have run many, many tests (one for each gene), we need to adjust the p-values to correct for multiple testing. We will use an FDR correction using the "p.adjust" function.

```
#Adjust p-values
padj = p.adjust(pValues, method="fdr");

#Let's look at a histogram of these new values
hist(padj);
```




```
#We can also list the smallest 10 p-values
#First we need to sort the list
padj.sorted = sort(padj);

#Now we can view it
knitr::kable(
  padj[1:10], caption = '10 Adjusted p-values, sorted',
  booktabs = TRUE
)
```

After running the t-tests and adjusting them for multiple testing, we can see that we have two p-values that are still significant. These p-values belong to gene ids 10837582, 10783648. We can look up information on these genes using the tables provided on the Platform GPL6247 page.

Table 3.1: 10 Adjusted p-values, sorted

	x
10700001	0.1713985
10700002	0.2683722
10700003	0.1664339
10700004	0.1664339
10700005	0.1664339
10700006	0.6812546
10700007	0.5660878
10700008	0.4958739
10700009	0.1915826
10700010	0.2116867

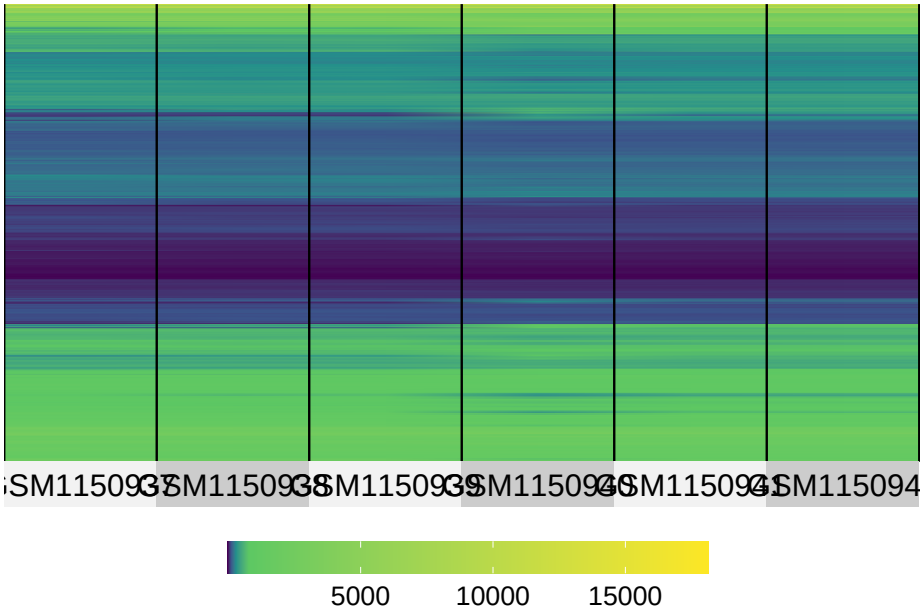
Chapter 4

Heatmap

Heatmaps are a compact way to globally visualize large datasets. Here we will use this to view our gene expression data across all of the samples at once. We can additionally cluseter these data to try to better understand which genes (or samples) are more similar or dissimilar based on this clustering.

There are many R packages and functions to generate heatmaps. Here, we will be using the superheat package. To generate a basic heatmap, we simply need to pass in our dataset. We will also do some basic clustering on the data.

```
#We can use the superheat package to visualize the data
library(superheat);

#Creates a nice heatmap
superheat(ge.data,
          pretty.order.rows = T);
```



From this, we can see that there appear to be various sets of genes that cluster together. Considering we only have 6 samples and two treatment conditions, we did not cluster by samples.