

Module 3.2: Learn - Coding

Overview

In this section, you will begin to analyze real data! We will begin by running the differential expression on raw data with no trimming applied to develop a baseline to compare to the results from the rest of the trimming data sets. If (or really when) you get errors when you are running code, the Professional Development section below reviews some tips for how to do an internet search for help with those errors.

More specifically, you will:

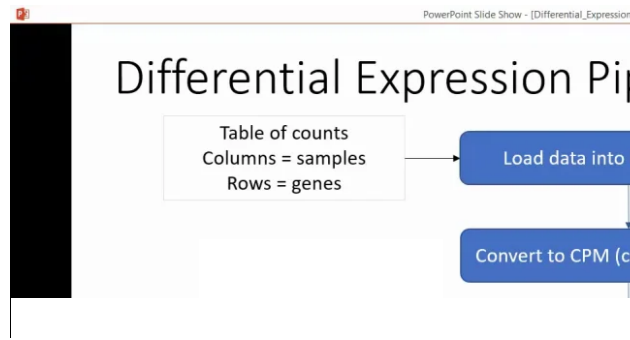
1. Run Rmd of differential expression analysis for default (untrimmed data)
2. Visualize gene expression differences with volcano plots and box plots
3. Visualize trends in samples based on their gene expression profiles with MDS plots and hierarchical clustering

We will begin with a quick summary of the steps we are going to take and then continue with descriptions of those steps executed in the chunks of the Rmd provided for you. We expect you might jump back and forth between these learning materials and the running the code. The learning materials below are higher level explanations of what is going on, while the descriptions between chunks and in the comments of the Rmd that will help you to understand what each line is doing specifically.

Quick summary of the steps of our differential expression pipeline

In this section, we will be running `Week_3/DE_Pipeline_UntrimmedData.Rmd` on the untrimmed data.

This video will give you an overview of what is being done in this code:



Video. Differential expression pipeline.

In this video, Dr. Plaisier reviews the DE pipeline, starting with table of gene counts loaded into a DGEList object, converted to CPM, filtered for lowly expressed genes, and then the normalization factors are calculated.

View Transcript. (<https://canvas.asu.edu/courses/122165/files/54792275?wrap=1>) 
(https://canvas.asu.edu/courses/122165/files/54792275/download?download_frd=1)

Open and start running the Rmd for differential gene expression with the u

Please open up the Rmd now in an RStudio session on ASU Agave using the instructions from previous modules working directory to where you downloaded your own copies of the template code for this class

(/home/<ASUrite_id>/CURE_2022_Scripts): select the Session menu, Set Working Directory, Choose Directory to source file, then to the Week_3 subdirectory. You should then see DE_Pipeline_UntrimmedData.Rmd in the list of files. Clicking should open it for you and you can see when output files and reports are generated in that same directory. Then go to the File menu and select Open File to simply open the Rmd.

As you go through the code, you will see there are functions that perform the steps of differential expression analysis that will help you to understand those functions.

A good way to use these resources is to read the section below about each chunk, then go back to your RStudio lines of that chunk. To remind you, the name of the chunk is given after ```{r`.

To begin, activate the needed packages by running the *Packages* chunk.

Next, set the working directory to the Week_3 subdirectory in your home directory by changing the username in the `working_directory` variable in the *WorkingDirectory* chunk and running it. This ensures that you are looking at the correct directory even if you don't change the working directory in the current RStudio session.

Input for the pipeline [*DataSetup* chunk]

The main input for the DE pipeline is a table of count values, known as a count matrix, which contains all the samples as columns, all the genes as rows, and the values as all the counts of how many reads align to each gene. These are generated by a program called `featureCounts`, which takes in an alignment file which matches the reads to the reference genome and an annotation file which tells which gene goes with the coordinates of the matched sequence reads. This data goes into a `DGEList` object, the specialized data for gene expression analysis that you learned about in a previous module. We will use the samples as phenotype data because we are interested in sex differentially expressed genes.

In this study, we sampled each placenta sample two times to get run two RNA sequencing samples; these serve as technical replicates for each placenta sample. Counts for both technical replicates are summed for each sample using the `sumTech` function. Since the counts for the technical replicates came from the same tissue sample. Additionally, this ensures that the technical replicates are treated like individual samples in the analysis.

The path to the input directory will be in the shared data directory for this class. We do not need to change this as we will be reading files from the same location and writing results to the subdirectory in the own /home directory. This saves space as we don't have to make copies of the input data for everyone to use. Making copies of data is inefficient and costly whenever possible.

The shared data directory is:

```
/data/mwilsons/GenomicsResearchExperience/Placenta_SexDiff/
```

The gene counts data will be in the geneCounts subdirectory:

```
/data/mwilsons/GenomicsResearchExperience/Placenta_SexDiff/geneCounts/
```

The count matrix for the untrimmed data will be in a comma separated value (CSV) file in the trimq_NONE_minle

```
/data/mwilsons/GenomicsResearchExperience/Placenta_SexDiff/geneCounts/trimq_NONE_r  
Counts_all.csv
```

Convert to counts per million (CPM) [Preprocessing chunk]

The count matrix is converted from raw read counts to counts per million (CPM) using the `cpm` function. This is a technique that accounts for differences in library size, the overall number of reads in the sample. If certain samples are general, we would likely see a higher read count for all genes in that sample. Additionally, the data is log-transformed to see genes on higher or lower ends of expression values on the same scale.

Filtering low expression genes [*FilterLowExpression* chunk]

Next we filter out genes with very low expression (low CPM values) using the `filterByExpr` function. All samples have genes that are expressed and not expressed, depending on what genes are activated given the specific conditions of the cells in the tissue we collected. This function was written to very simply threshold expression levels to find genes with expression that it is likely that they are inactive in these cells. Genes with expression values below thresholds determined by the `filterByExpr` function are likely noisy because we can not easily detect differential expression when there are low counts that map to the gene. The thresholds are calculated as minimum CPM values which amount to keeping genes with at least 3 reads or more in at least three samples.

Normalization [*Normalization* chunk]

In this chunk we will use a box plot to show the expression distribution of all the samples before and after normalization using the `calcNormFactors` function. After calculating and applying normalization factors within the `DGEList` object, we can see that the samples have data on the same range, so that they can be compared to each other more easily. In our specific case, we see that all the samples are pretty much in the same range before normalization in the box plot, so after normalization

more similar box plots and the `norm.factors` list within the `samples` table in the `DGEList` is filled with values pretty close to 1 (if the values are not 1, it doesn't hurt to run it anyway). This is done mainly as a demonstration for you; typically we can just apply this factor before differential gene expression analysis.

Multidimensional Scaling (MDS) Plot [*MDSPlot* chunk]

Here we will cluster the samples by gene expression profile and label the samples by sex. In this way we can visualize if XX female samples cluster together and the XY male samples cluster together. If so, we know that sex is an important factor in gene expression profile in this tissue. This plot is produced using the `plotMDS` function using the expression values `logCPM`.

Differential Expression Analysis [*DesignMatrixContrasts* and *GeneStats* chunks]

To output our list of differentially expressed genes from gene counts from reads with no trimming applied, this chunk continues the modeling process described above using the `limma` package in R. The design matrix will be used to associate the samples with sex (stored in the `group` variable in the `DataSetup` chunk). We set up the comparison we are interested in using the `makeContrasts` function, labeling the variable `FemalevsMale` with `female - male`, which represents the ratio of expression in female samples divided by the mean expression in the male samples (since the values are in log scale, taking the difference is mathematically equivalent to taking the ratio without log-transforming).

Next, we apply a `voom` transformation to account for different variance at different expression levels using the `voom` function. We then fit the model using the `lmFit` function. The `eBayes` function is used to rank the genes in order of evidence for differential expression. The mean-variance plot is produced by the `plotSA` function so you can see the variance flatten out over the range of expression.


From here we can filter for genes that are differentially expressed in XX females vs XY males and then produce a set of diagnostic metrics to show us which genes show statistically significant differences. In future modules, you will be generating different figures to help you answer the research question.

From here we can determine which genes have evidence of differential expression. We can get a count of how many genes are upregulated and downregulated using the `decideTests` function. We can calculate important differential expression statistics and use them for further investigation.

Visualizing differential gene expression [*VolcanoPlot* and *PlotS* chunks]

This chunk will show you how to use R packages to create the visualizations we learned about in the previous sections. In the *VolcanoPlot* chunk, we use an awesome package called *EnhancedVolcano* that takes in the data and creates a beautiful volcano plot with colors and labels. Finding packages like this can save you a lot of time in the long run. The next chunk shows you how to create plots to compare the expression of genes in the females and males using *ggplot2*. You learned a bit about this in the last module, you will see that it allows you to try different visualization formats so you can choose the one you think best shows the results you are interested in. We have put a few genes in the code as examples; you can use the same code for any other gene(s) in the data set.

Module 3.2 Additional Resources

- Flowchart on general RNAseq analysis: https://biocorecrg.github.io/RNAseq_course_2019/alignment.html (https://biocorecrg.github.io/RNAseq_course_2019/alignment.html)
- **Bioconductor limma workflow** 
(<https://bioconductor.org/packages/release/workflows/vignettes/RNAseq123/inst/doc/limmaWorkflow.html>)
- Common troubleshooting solution:
If you ever feel like some package installations are timing out or taking extra-long to load, there may be something working out of your source pane and have the console minimized, you may be missing the prompt to finish the image below, R is asking if you want to update the dependencies in the Bioconductor software package:

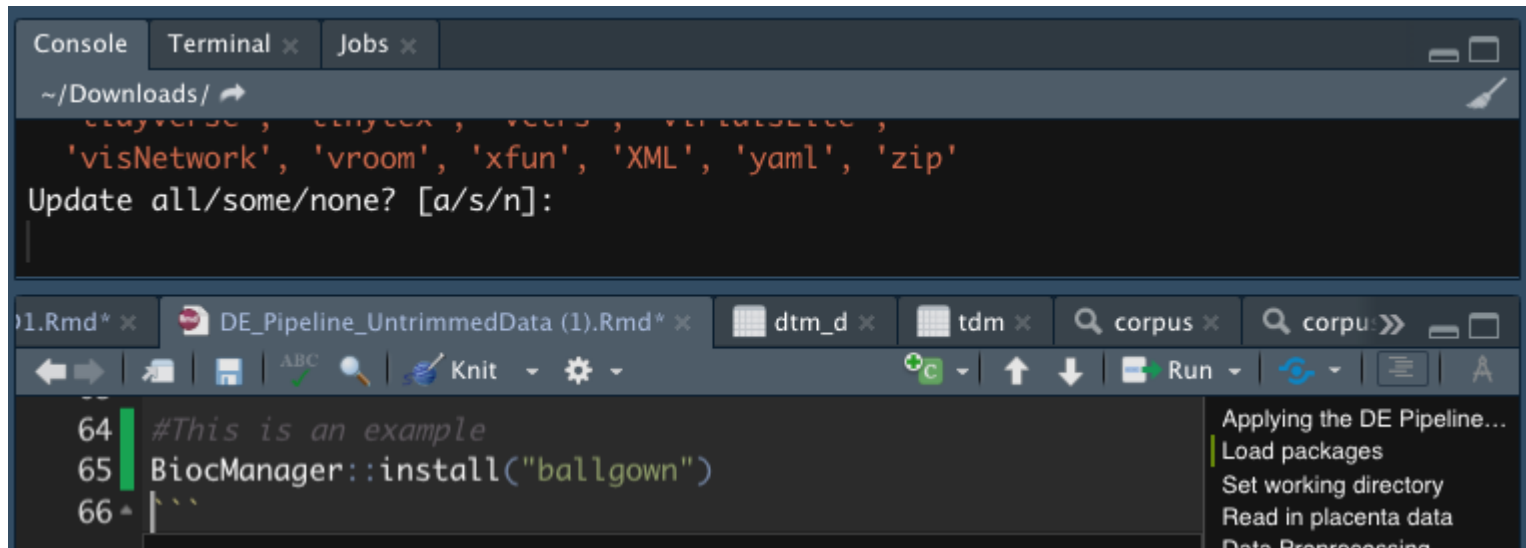


Figure. Bioconductor software packages installation prompt.

```
Update all/some/none? [a/s/n]:  
a
```

The messages from the install are typically asking if you want to update all, some, or none of the dependencies. Usually you want to type the letter `a` for "all" to continue with the installation. Alternatively, you may also see `y/n` or `Yes/No` and you can just type `y` to proceed.