Teaching and Learning Materials

Sean Davis

6/29/2017

Contents

Right now, this page serves as the home for my materials for the CSHL Statistical Methods for Functional Genomics.

The materials are located in the "R and Bioconductor" tab at the top right, mainly. Links to slides are under the "slides" tab above. Finally, there are some additional and miscellaneous materials in the "Misc." tab.

To get the materials downloaded to your computer, click here.

0.1 What is R?

R is a number of things, simultaneously. Depending on who is being asked, R is:

- A software package
- A programming language
- A toolkit for developing statistical and analytical tools
- An extensive library of statistical and mathematical software and algorithms
- A scripting language
- much, much more

0.2 Why use R?

- R is cross-platform and runs on Windows, Mac, and Linux (as well as more obscure systems).
- R provides a vast number of useful statistical tools, many of which have been painstakingly tested.
- R produces publication-quality graphics in a variety of formats.
- R plays well with FORTRAN, C, and scripts in many languages.

- R scales, making it useful for small and large projects. It is NOT Excel.
- R does not have a meaningfully useful graphical user interface (GUI).

I can develop code for analysis on my Mac laptop. I can then install the *same* code on our 20k core cluster and run it in parallel on 100 samples, monitor the process, and then update a database (for example) with R when complete.

0.3 Why not use R?

- R cannot do everything.
- R is not always the "best" tool for the job.
- R will *not* hold your hand. Often, it will *slap* your hand instead.
- The documentation can be opaque (but there is documentation).
- R can drive you crazy (on a good day) or age you prematurely (on a bad one).
- Finding the right package to do the job you want to do can be challenging; worse, some contributed packages are unreliable.]{}
- R does not have a meaningfully useful graphical user interface (GUI).

0.4 R License and the Open Source Ideal

R is free (yes, totally free!) and distributed under GNU license. In particular, this license allows one to:

- Download the source code
- Modify the source code to your heart's content
- Distribute the modified source code and even charge money for it, but you must distribute the modified source code under the original GNU license]{}

This license means that R will always be available, will always be open source, and can grow organically without constraint.

Data analysis involves a large amount of janitor work – munging and cleaning data to facilitate downstream data analysis. This lesson demonstrates techniques for advanced data manipulation and analysis with the split-apply-combine strategy. We will use the dplyr package in R to effectively manipulate and conditionally compute summary statistics over subsets of a "big" dataset containing many observations.

Recommended reading: Review the *Introduction* (10.1) and *Tibbles vs. data.frame* (10.3) sections of the ***R for Data Science* book**. We will initially be using the `read_*` functions from the **readr** package. These functions load data into a *tibble* instead of R's traditional data.frame. Tibbles are data frames, but they tweak some older behaviors to make life a little easier. These sections explain the few key small differences between traditional data.frames and tibbles.

0.5 Review

0.5.1 The data

We are going to use a yeast gene expression dataset. This is a cleaned up version of a gene expression dataset from Brauer et al. Coordination of Growth Rate, Cell Cycle, Stress Response, and Metabolic Activity in Yeast (2008) *Mol Biol Cell* 19:352-367. These data are from a gene expression microarray, and in this paper the authors are examining the relationship between growth rate and gene expression in yeast cultures limited by one of six different nutrients (glucose, leucine, ammonium, sulfate, phosphate, uracil). If you give yeast a rich media loaded with nutrients except restrict the supply of a *single* nutrient, you can control the growth rate to any rate you choose. By starving yeast of specific nutrients you can find genes that:

1. **Raise or lower their expression in response to growth rate**. Growth-rate dependent expression patterns can tell us a lot about cell cycle control, and how the cell responds to stress. The authors found that expression of >25% of all yeast genes is linearly correlated with growth rate, independent of the limiting nutrient. They also found that the subset of negatively growth-correlated genes is enriched for peroxisomal functions, and positively correlated genes mainly encode ribosomal functions.
2. **Respond differently when different nutrients are being limited**. If you see particular genes that respond very differently when a nutrient is sharply restricted, these genes might be involved in the transport or metabolism of that specific nutrient.

You can download the cleaned up version of the data at the link above. The file is called **brauer2007_tidy.csv**. Later on we'll actually start with the original raw data (minimally processed) and manipulate it so that we can make it more amenable for analysis.

0.5.2 Reading in data

We need to load both the dplyr and readr packages for efficiently reading in and displaying this data. We're also going to use many other functions from the dplyr package. Make sure you have these packages installed as described on the setup page.

```
# Load packages
library(readr)
library(dplyr)

# Read in data
ydat <- read_csv("http://bioconnector.org/data/brauer2007_tidy.csv")
## Parsed with column specification:
## cols(
##   symbol = col_character(),
##   systematic_name = col_character(),
##   nutrient = col_character(),
##   rate = col_double(),
##   expression = col_double(),
##   bp = col_character(),
##   mf = col_character()
## )

# Display the data
ydat

# Optionally, bring up the data in a viewer window
# View(ydat)
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Glucose   0.05       0.28 biological~ molecular ~
## 3 QRI7   YDL104C         Glucose   0.05      -0.02 proteolysi~ metalloend~
## 4 CFT2   YLR115W         Glucose   0.05      -0.33 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Glucose   0.05       0.05 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Glucose   0.05      -0.69 biological~ molecular ~
## # ... with 1.984e+05 more rows
```

0.6 The dplyr package

The dplyr package is a relatively new R package that makes data manipulation fast and easy. It imports functionality from another package called magrittr that allows you to chain commands together into a pipeline that will completely change the way you write R code such that you're writing code the way you're thinking about the problem.

When you read in data with the readr package (`read_csv()`) and you had the dplyr package loaded already, the data frame takes on this "special" class of data frames called a `tbl` (pronounced "tibble"), which you can see with `class(ydat)`. If you have other "regular" data frames in your workspace, the `as_tibble()` function will convert it into the special dplyr `tbl` that displays nicely (e.g.: `iris <- as_tibble(iris)`). You don't have to turn all your data frame objects into tibbles, but it does make working with large datasets a bit easier.

You can read more about tibbles in Tibbles chapter in *R for Data Science* or in the tibbles vignette. They keep most of the features of data frames, and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors). You can read more about the differences between data frames and tibbles in this section of the tibbles vignette, but the major convenience for us concerns **printing** (aka displaying) a tibble to the screen. When you print (i.e., display) a tibble, it only shows the first 10 rows and all the columns that fit on one screen. It also prints an abbreviated description of the column type. You can control the default appearance with options:

- `options(tibble.print_max = n, tibble.print_min = m)`: if there are more than n rows, print only the first m rows. Use `options(tibble.print_max = Inf)` to always show all rows.
- `options(tibble.width = Inf)` will always print all columns, regardless of the width of the screen.

0.7 dplyr verbs

The dplyr package gives you a handful of useful **verbs** for managing data. On their own they don't do anything that base R can't do. Here are some of the *single-table* verbs we'll be working with in this lesson (single-table meaning that they only work on a single table – contrast that to *two-table* verbs used for joining data together, which we'll cover in a later lesson).

1. `filter()`
2. `select()`
3. `mutate()`
4. `arrange()`
5. `summarize()`
6. `group_by()`

They all take a data frame or tibble as their input for the first argument, and they all return a data frame or tibble as output.

0.7.1 filter()

If you want to filter **rows** of the data where some condition is true, use the `filter()` function.

1. The first argument is the data frame you want to filter, e.g. `filter(mydata, ....`
2. The second argument is a condition you must satisfy, e.g. `filter(ydat, symbol == "LEU1")`. If you want to satisfy *all* of multiple conditions, you can use the "and" operator, `&`. The "or" operator `|` (the pipe character, usually shift-backslash) will return a subset that meet *any* of the conditions.

- `==`: Equal to
- `!=`: Not equal to
- `>`, `>=`: Greater than, greater than or equal to
- `<`, `<=`: Less than, less than or equal to

Let's try it out. For this to work you have to have already loaded the dplyr package. Let's take a look at LEU1, a gene involved in leucine synthesis.

```
# First, make sure you've loaded the dplyr package
library(dplyr)

# Look at a single gene involved in leucine synthesis pathway
filter(ydat, symbol == "LEU1")
## # A tibble: 36 x 7
##   symbol systematic_name nutrient  rate expression bp        mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>     <chr>
## 1 LEU1   YGL009C         Glucose   0.05      -1.12 leucine ~ 3-isopropylm~
## 2 LEU1   YGL009C         Glucose   0.1       -0.77 leucine ~ 3-isopropylm~
## 3 LEU1   YGL009C         Glucose   0.15      -0.67 leucine ~ 3-isopropylm~
## 4 LEU1   YGL009C         Glucose   0.2       -0.59 leucine ~ 3-isopropylm~
## 5 LEU1   YGL009C         Glucose   0.25      -0.2  leucine ~ 3-isopropylm~
## 6 LEU1   YGL009C         Glucose   0.3        0.03 leucine ~ 3-isopropylm~
## # ... with 30 more rows

# Optionally, bring that result up in a View window
# View(filter(ydat, symbol == "LEU1"))

# Look at multiple genes
filter(ydat, symbol=="LEU1" | symbol=="ADH2")
## # A tibble: 72 x 7
##   symbol systematic_name nutrient  rate expression bp        mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>     <chr>
## 1 LEU1   YGL009C         Glucose   0.05      -1.12 leucine ~ 3-isopropylm~
## 2 ADH2   YMR303C         Glucose   0.05       6.28 fermenta~ alcohol dehy~
## 3 LEU1   YGL009C         Glucose   0.1       -0.77 leucine ~ 3-isopropylm~
## 4 ADH2   YMR303C         Glucose   0.1        5.81 fermenta~ alcohol dehy~
## 5 LEU1   YGL009C         Glucose   0.15      -0.67 leucine ~ 3-isopropylm~
## 6 ADH2   YMR303C         Glucose   0.15       5.64 fermenta~ alcohol dehy~
## # ... with 66 more rows

# Look at LEU1 expression at a low growth rate due to nutrient depletion
# Notice how LEU1 is highly upregulated when leucine is depleted!
filter(ydat, symbol=="LEU1" & rate==.05)
## # A tibble: 6 x 7
##   symbol systematic_name nutrient   rate expression bp        mf
##   <chr>  <chr>           <chr>     <dbl>      <dbl> <chr>     <chr>
## 1 LEU1   YGL009C         Glucose    0.05      -1.12 leucine ~ 3-isopropyl~
## 2 LEU1   YGL009C         Ammonia    0.05      -0.76 leucine ~ 3-isopropyl~
## 3 LEU1   YGL009C         Phosphate  0.05      -0.81 leucine ~ 3-isopropyl~
## 4 LEU1   YGL009C         Sulfate    0.05      -1.57 leucine ~ 3-isopropyl~
## 5 LEU1   YGL009C         Leucine    0.05       3.84 leucine ~ 3-isopropyl~
## 6 LEU1   YGL009C         Uracil     0.05      -2.07 leucine ~ 3-isopropyl~

# But expression goes back down when the growth/nutrient restriction is relaxed
filter(ydat, symbol=="LEU1" & rate==.3)
## # A tibble: 6 x 7
##   symbol systematic_name nutrient   rate expression bp        mf
##   <chr>  <chr>           <chr>     <dbl>      <dbl> <chr>     <chr>
## 1 LEU1   YGL009C         Glucose     0.3       0.03 leucine ~ 3-isopropyl~
```

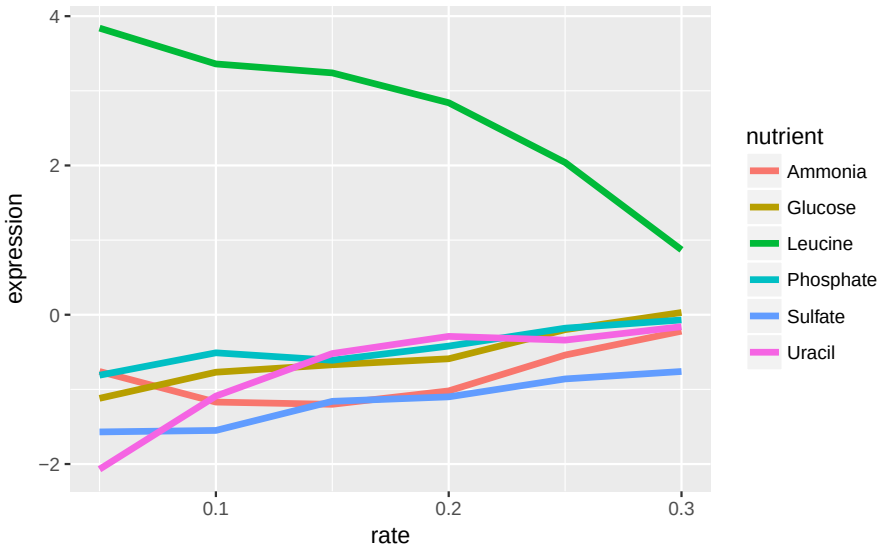
```
## 2 LEU1   YGL009C         Ammonia     0.3      -0.22 leucine ~ 3-isopropyl~
## 3 LEU1   YGL009C         Phosphate   0.3      -0.07 leucine ~ 3-isopropyl~
## 4 LEU1   YGL009C         Sulfate     0.3      -0.76 leucine ~ 3-isopropyl~
## 5 LEU1   YGL009C         Leucine     0.3       0.87 leucine ~ 3-isopropyl~
## 6 LEU1   YGL009C         Uracil      0.3      -0.16 leucine ~ 3-isopropyl~

# Show only stats for LEU1 and Leucine depletion.
# LEU1 expression starts off high and drops
filter(ydat, symbol=="LEU1" & nutrient=="Leucine")
## # A tibble: 6 x 7
##   symbol systematic_name nutrient  rate expression bp        mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>     <chr>
## 1 LEU1   YGL009C         Leucine   0.05       3.84 leucine ~ 3-isopropylm~
## 2 LEU1   YGL009C         Leucine   0.1        3.36 leucine ~ 3-isopropylm~
## 3 LEU1   YGL009C         Leucine   0.15       3.24 leucine ~ 3-isopropylm~
## 4 LEU1   YGL009C         Leucine   0.2        2.84 leucine ~ 3-isopropylm~
## 5 LEU1   YGL009C         Leucine   0.25       2.04 leucine ~ 3-isopropylm~
## 6 LEU1   YGL009C         Leucine   0.3        0.87 leucine ~ 3-isopropylm~

# What about LEU1 expression with other nutrients being depleted?
filter(ydat, symbol=="LEU1" & nutrient=="Glucose")
## # A tibble: 6 x 7
##   symbol systematic_name nutrient  rate expression bp        mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>     <chr>
## 1 LEU1   YGL009C         Glucose   0.05      -1.12 leucine ~ 3-isopropylm~
## 2 LEU1   YGL009C         Glucose   0.1       -0.77 leucine ~ 3-isopropylm~
## 3 LEU1   YGL009C         Glucose   0.15      -0.67 leucine ~ 3-isopropylm~
## 4 LEU1   YGL009C         Glucose   0.2       -0.59 leucine ~ 3-isopropylm~
## 5 LEU1   YGL009C         Glucose   0.25      -0.2  leucine ~ 3-isopropylm~
## 6 LEU1   YGL009C         Glucose   0.3        0.03 leucine ~ 3-isopropylm~
```

Let's look at this graphically. Don't worry about what these commands are doing just yet - we'll cover that later on when we talk about ggplot2. Here's I'm taking the filtered dataset containing just expression estimates for LEU1 where I have 36 rows (one for each of 6 nutrients $\times$ 6 growth rates), and I'm *piping* that dataset to the plotting function, where I'm plotting rate on the x-axis, expression on the y-axis, mapping the value of nutrient to the color, and using a line plot to display the data.

```
library(ggplot2)
filter(ydat, symbol=="LEU1") %>%
  ggplot(aes(rate, expression, colour=nutrient)) + geom_line(lwd=1.5)
```



Look closely at that! LEU1 is *highly expressed* when starved of leucine because the cell has to synthesize its own! And as the amount of leucine in the environment (the growth *rate*) increases, the cell can worry less about synthesizing leucine, so LEU1 expression goes back down. Consequently the cell can devote more energy into other functions, and we see other genes' expression very slightly raising.

EXERCISE 1

1. Display the data where the gene ontology biological process (the `bp` variable) is "leucine biosynthesis" (case-sensitive) *and* the limiting nutrient was Leucine. (Answer should return a 24-by-7 data frame – 4 genes × 6 growth rates).
2. Gene/rate combinations had high expression (in the top 1% of expressed genes)? *Hint:* see `?quantile` and try `quantile(ydat$expression, probs=.99)` to see the expression value which is higher than 99% of all the data, then `filter()` based on that. Try wrapping your answer with a `View()` function so you can see the whole thing. What does it look like those genes are doing? Answer should return a 1971-by-7 data frame.

0.7.1.1 Aside: Writing Data to File

What we've done up to this point is read in data from a file (`read_csv(...)`), and assigning that to an object in our *workspace* (`ydat <- ...`). When we run operations like `filter()` on our data, consider two things:

1. The `ydat` object in our workspace is not being modified directly. That is, we can `filter(ydat, ...)`, and a result is returned to the screen, but `ydat` remains the same. This effect is similar to what we demonstrated in our first session.

```
# Assign the value '50' to the weight object.
weight <- 50

# Print out weight to the screen (50)
weight

# What's the value of weight plus 10?
weight + 10

# Weight is still 50
```

```
weight

# Weight is only modified if we *reassign* weight to the modified value
weight <- weight+10
# Weight is now 60
weight
```

2. More importantly, the *data file on disk* (`data/brauer2007_tidy.csv`) is *never* modified. No matter what we do to ydat, the file is never modified. If we want to *save* the result of an operation to a file on disk, we can assign the result of an operation to an object, and `write_csv` that object to disk. See the help for `?write_csv` (note, `write_csv()` with an underscore is part of the **readr** package – not to be confused with the built-in `write.csv()` function).

```
# What's the result of this filter operation?
filter(ydat, nutrient=="Leucine" & bp=="leucine biosynthesis")

# Assign the result to a new object
leudat <- filter(ydat, nutrient=="Leucine" & bp=="leucine biosynthesis")

# Write that out to disk
write_csv(leudat, "leucinedata.csv")
```

Note that this is different than saving your *entire workspace to an Rdata file*, which would contain all the objects we've created (weight, ydat, leudat, etc).

0.7.2 select()

The `filter()` function allows you to return only certain *rows* matching a condition. The `select()` function returns only certain *columns*. The first argument is the data, and subsequent arguments are the columns you want.

```
# Select just the symbol and systematic_name
select(ydat, symbol, systematic_name)
## # A tibble: 198,430 x 2
##   symbol systematic_name
##   <chr>  <chr>
## 1 SFB2   YNL049C
## 2 <NA>   YNL095C
## 3 QRI7   YDL104C
## 4 CFT2   YLR115W
## 5 SSO2   YMR183C
## 6 PSP2   YML017W
## # ... with 1.984e+05 more rows

# Alternatively, just remove columns. Remove the bp and mf columns.
select(ydat, -bp, -mf)
## # A tibble: 198,430 x 5
##   symbol systematic_name nutrient  rate expression
##   <chr>  <chr>           <chr>    <dbl>      <dbl>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24
## 2 <NA>   YNL095C         Glucose   0.05       0.28
## 3 QRI7   YDL104C         Glucose   0.05      -0.02
## 4 CFT2   YLR115W         Glucose   0.05      -0.33
## 5 SSO2   YMR183C         Glucose   0.05       0.05
```

```
## 6 PSP2   YML017W         Glucose   0.05      -0.69
## # ... with 1.984e+05 more rows

# Notice that the original data doesn't change!
ydat
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Glucose   0.05       0.28 biological~ molecular ~
## 3 QRI7   YDL104C         Glucose   0.05      -0.02 proteolysi~ metalloend~
## 4 CFT2   YLR115W         Glucose   0.05      -0.33 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Glucose   0.05       0.05 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Glucose   0.05      -0.69 biological~ molecular ~
## # ... with 1.984e+05 more rows
```

Notice above how the original data doesn't change. We're selecting out only certain columns of interest and throwing away columns we don't care about. If we wanted to *keep* this data, we would need to *reassign* the result of the `select()` operation to a new object. Let's make a new object called `nogo` that does not contain the GO annotations. Notice again how the original data is unchanged.

```
# create a new dataset without the go annotations.
nogo <- select(ydat, -bp, -mf)
nogo
## # A tibble: 198,430 x 5
##   symbol systematic_name nutrient  rate expression
##   <chr>  <chr>           <chr>    <dbl>      <dbl>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24
## 2 <NA>   YNL095C         Glucose   0.05       0.28
## 3 QRI7   YDL104C         Glucose   0.05      -0.02
## 4 CFT2   YLR115W         Glucose   0.05      -0.33
## 5 SSO2   YMR183C         Glucose   0.05       0.05
## 6 PSP2   YML017W         Glucose   0.05      -0.69
## # ... with 1.984e+05 more rows

# we could filter this new dataset
filter(nogo, symbol=="LEU1" & rate==.05)
## # A tibble: 6 x 5
##   symbol systematic_name nutrient   rate expression
##   <chr>  <chr>           <chr>     <dbl>      <dbl>
## 1 LEU1   YGL009C         Glucose    0.05      -1.12
## 2 LEU1   YGL009C         Ammonia    0.05      -0.76
## 3 LEU1   YGL009C         Phosphate  0.05      -0.81
## 4 LEU1   YGL009C         Sulfate    0.05      -1.57
## 5 LEU1   YGL009C         Leucine    0.05       3.84
## 6 LEU1   YGL009C         Uracil     0.05      -2.07

# Notice how the original data is unchanged - still have all 7 columns
ydat
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Glucose   0.05       0.28 biological~ molecular ~
```

```
## 3 QRI7   YDL104C         Glucose   0.05      -0.02 proteolysi~ metalloend~
## 4 CFT2   YLR115W         Glucose   0.05      -0.33 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Glucose   0.05       0.05 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Glucose   0.05      -0.69 biological~ molecular ~
## # ... with 1.984e+05 more rows
```

0.7.3 mutate()

The `mutate()` function adds new columns to the data. Remember, it doesn't actually modify the data frame you're operating on, and the result is transient unless you assign it to a new object or reassign it back to itself (generally, not always a good practice).

The expression level reported here is the log_2 of the sample signal divided by the signal in the reference channel, where the reference RNA for all samples was taken from the glucose-limited chemostat grown at a dilution rate of 0.25 h^{-1}. Let's mutate this data to add a new variable called "signal" that's the actual raw signal ratio instead of the log-transformed signal.

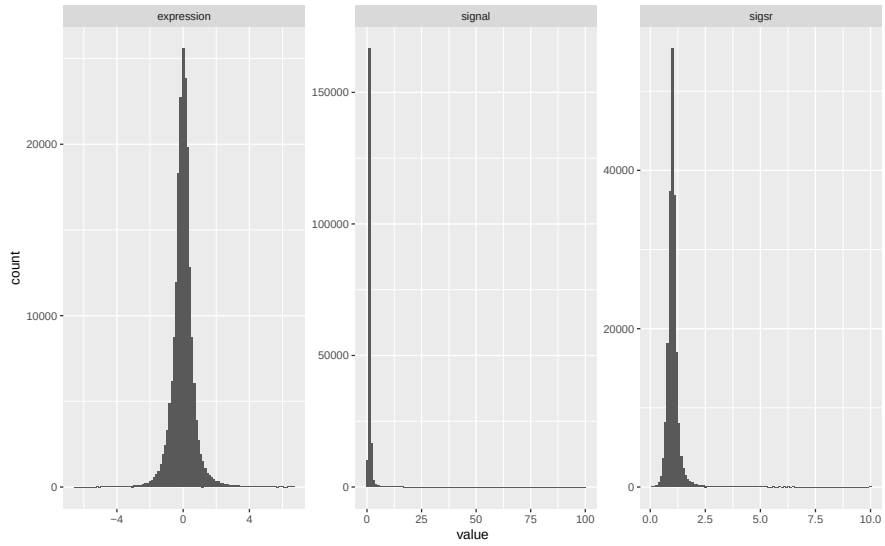
```
mutate(nogo, signal=2^expression)
```

Mutate has a nice little feature too in that it's "lazy." You can mutate and add one variable, then continue mutating to add more variables based on that variable. Let's make another column that's the square root of the signal ratio.

```
mutate(nogo, signal=2^expression, sigsr=sqrt(signal))
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient  rate expression signal sigsr
##   <chr>  <chr>           <chr>    <dbl>      <dbl>  <dbl> <dbl>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24  0.847 0.920
## 2 <NA>   YNL095C         Glucose   0.05       0.28  1.21  1.10
## 3 QRI7   YDL104C         Glucose   0.05      -0.02  0.986 0.993
## 4 CFT2   YLR115W         Glucose   0.05      -0.33  0.796 0.892
## 5 SSO2   YMR183C         Glucose   0.05       0.05  1.04  1.02
## 6 PSP2   YML017W         Glucose   0.05      -0.69  0.620 0.787
## # ... with 1.984e+05 more rows
```

Again, don't worry about the code here to make the plot – we'll learn about this later. Why do you think we log-transform the data prior to analysis?

```
library(tidyr)
mutate(nogo, signal=2^expression, sigsr=sqrt(signal)) %>%
  gather(unit, value, expression:sigsr) %>%
  ggplot(aes(value)) + geom_histogram(bins=100) + facet_wrap(~unit, scales="free")
```



0.7.4 arrange()

The `arrange()` function does what it sounds like. It takes a data frame or tbl and arranges (or sorts) by column(s) of interest. The first argument is the data, and subsequent arguments are columns to sort on. Use the `desc()` function to arrange by descending.

```
# arrange by gene symbol
arrange(ydat, symbol)
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient  rate expression bp         mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>      <chr>
## 1 AAC1   YMR056C         Glucose   0.05       1.5  aerobic r~ ATP:ADP ant~
## 2 AAC1   YMR056C         Glucose   0.1        1.54 aerobic r~ ATP:ADP ant~
## 3 AAC1   YMR056C         Glucose   0.15       1.16 aerobic r~ ATP:ADP ant~
## 4 AAC1   YMR056C         Glucose   0.2        1.04 aerobic r~ ATP:ADP ant~
## 5 AAC1   YMR056C         Glucose   0.25       0.84 aerobic r~ ATP:ADP ant~
## 6 AAC1   YMR056C         Glucose   0.3        0.01 aerobic r~ ATP:ADP ant~
## # ... with 1.984e+05 more rows

# arrange by expression (default: increasing)
arrange(ydat, expression)
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient   rate expression bp         mf
##   <chr>  <chr>           <chr>     <dbl>      <dbl> <chr>      <chr>
## 1 SUL1   YBR294W         Phosphate  0.05      -6.5  sulfate ~ sulfate tra~
## 2 SUL1   YBR294W         Phosphate  0.1       -6.34 sulfate ~ sulfate tra~
## 3 ADH2   YMR303C         Phosphate  0.1       -6.15 fermenta~ alcohol deh~
## 4 ADH2   YMR303C         Phosphate  0.3       -6.04 fermenta~ alcohol deh~
## 5 ADH2   YMR303C         Phosphate  0.25      -5.89 fermenta~ alcohol deh~
## 6 SUL1   YBR294W         Uracil     0.05      -5.55 sulfate ~ sulfate tra~
## # ... with 1.984e+05 more rows

# arrange by decreasing expression
arrange(ydat, desc(expression))
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient  rate expression bp         mf
```

```
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>      <chr>
## 1 GAP1   YKR039W         Ammonia   0.05       6.64 amino aci~ L-proline p~
## 2 DAL5   YJR152W         Ammonia   0.05       6.64 allantoat~ allantoate ~
## 3 GAP1   YKR039W         Ammonia   0.1        6.64 amino aci~ L-proline p~
## 4 DAL5   YJR152W         Ammonia   0.1        6.64 allantoat~ allantoate ~
## 5 DAL5   YJR152W         Ammonia   0.15       6.64 allantoat~ allantoate ~
## 6 DAL5   YJR152W         Ammonia   0.2        6.64 allantoat~ allantoate ~
## # ... with 1.984e+05 more rows
```

EXERCISE 2

1. First, re-run the command you used above to filter the data for genes involved in the "leucine biosynthesis" biological process *and* where the limiting nutrient is Leucine.
2. Wrap this entire filtered result with a call to `arrange()` where you'll arrange the result of #1 by the gene symbol.
3. Wrap this entire result in a `View()` statement so you can see the entire result.

0.7.5 summarize()

The `summarize()` function summarizes multiple values to a single value. On its own the `summarize()` function doesn't seem to be all that useful. The dplyr package provides a few convenience functions called `n()` and `n_distinct()` that tell you the number of observations or the number of distinct values of a particular variable.

Notice that summarize takes a data frame and returns a data frame. In this case it's a 1x1 data frame with a single row and a single column. The name of the column, by default is whatever the expression was used to summarize the data. This usually isn't pretty, and if we wanted to work with this resulting data frame later on, we'd want to name that returned value something easier to deal with.

```
# Get the mean expression for all genes
summarize(ydat, mean(expression))
## # A tibble: 1 x 1
##   `mean(expression)`
##                <dbl>
## 1            0.00337

# Use a more friendly name, e.g., meanexp, or whatever you want to call it.
summarize(ydat, meanexp=mean(expression))
## # A tibble: 1 x 1
##   meanexp
##     <dbl>
## 1 0.00337

# Measure the correlation between rate and expression
summarize(ydat, r=cor(rate, expression))
## # A tibble: 1 x 1
##         r
##     <dbl>
## 1 -0.0220

# Get the number of observations
summarize(ydat, n())
```

```
## # A tibble: 1 x 1
##    `n()`
##    <int>
## 1 198430

# The number of distinct gene symbols in the data
summarize(ydat, n_distinct(symbol))
## # A tibble: 1 x 1
##   `n_distinct(symbol)`
##                  <int>
## 1                 4211
```

0.7.6 group_by()

We saw that `summarize()` isn't that useful on its own. Neither is `group_by()` All this does is takes an existing data frame and coverts it into a grouped data frame where operations are performed by group.

```
ydat
## # A tibble: 198,430 x 7
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Glucose   0.05       0.28 biological~ molecular ~
## 3 QRI7   YDL104C         Glucose   0.05      -0.02 proteolysi~ metalloend~
## 4 CFT2   YLR115W         Glucose   0.05      -0.33 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Glucose   0.05       0.05 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Glucose   0.05      -0.69 biological~ molecular ~
## # ... with 1.984e+05 more rows
group_by(ydat, nutrient)
## # A tibble: 198,430 x 7
## # Groups:   nutrient [6]
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Glucose   0.05       0.28 biological~ molecular ~
## 3 QRI7   YDL104C         Glucose   0.05      -0.02 proteolysi~ metalloend~
## 4 CFT2   YLR115W         Glucose   0.05      -0.33 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Glucose   0.05       0.05 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Glucose   0.05      -0.69 biological~ molecular ~
## # ... with 1.984e+05 more rows
group_by(ydat, nutrient, rate)
## # A tibble: 198,430 x 7
## # Groups:   nutrient, rate [36]
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Glucose   0.05      -0.24 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Glucose   0.05       0.28 biological~ molecular ~
## 3 QRI7   YDL104C         Glucose   0.05      -0.02 proteolysi~ metalloend~
## 4 CFT2   YLR115W         Glucose   0.05      -0.33 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Glucose   0.05       0.05 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Glucose   0.05      -0.69 biological~ molecular ~
## # ... with 1.984e+05 more rows
```

The real power comes in where `group_by()` and `summarize()` are used together. First, write the `group_by()`

statement. Then wrap the result of that with a call to `summarize()`.

```
# Get the mean expression for each gene
# group_by(ydat, symbol)
summarize(group_by(ydat, symbol), meanexp=mean(expression))
## # A tibble: 4,211 x 2
##   symbol meanexp
##   <chr>    <dbl>
## 1 AAC1    0.529
## 2 AAC3   -0.216
## 3 AAD10   0.438
## 4 AAD14  -0.0717
## 5 AAD16   0.242
## 6 AAD4   -0.792
## # ... with 4,205 more rows

# Get the correlation between rate and expression for each nutrient
# group_by(ydat, nutrient)
summarize(group_by(ydat, nutrient), r=cor(rate, expression))
## # A tibble: 6 x 2
##   nutrient        r
##   <chr>       <dbl>
## 1 Ammonia   -0.0175
## 2 Glucose   -0.0112
## 3 Leucine   -0.0384
## 4 Phosphate -0.0194
## 5 Sulfate   -0.0166
## 6 Uracil    -0.0353
```

0.8 The pipe: %>%

0.8.1 How %>% works

This is where things get awesome. The dplyr package imports functionality from the magrittr package that lets you *pipe* the output of one function to the input of another, so you can avoid nesting functions. It looks like this: **%>%**. You don't have to load the magrittr package to use it since dplyr imports its functionality when you load the dplyr package.

Here's the simplest way to use it. Remember the `tail()` function. It expects a data frame as input, and the next argument is the number of lines to print. These two commands are identical:

```
tail(ydat, 5)
## # A tibble: 5 x 7
##   symbol systematic_name nutrient  rate expression bp           mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>        <chr>
## 1 KRE1   YNL322C         Uracil     0.3       0.28 cell wall o~ structura~
## 2 MTL1   YGR023W         Uracil     0.3       0.27 cell wall o~ molecular~
## 3 KRE9   YJL174W         Uracil     0.3       0.43 cell wall o~ molecular~
## 4 UTH1   YKR042W         Uracil     0.3       0.19 mitochondri~ molecular~
## 5 <NA>   YOL111C         Uracil     0.3       0.04 biological ~ molecular~
ydat %>% tail(5)
## # A tibble: 5 x 7
##   symbol systematic_name nutrient  rate expression bp           mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>        <chr>
```

```
## 1 KRE1   YNL322C         Uracil     0.3       0.28 cell wall o~ structura~
## 2 MTL1   YGR023W         Uracil     0.3       0.27 cell wall o~ molecular~
## 3 KRE9   YJL174W         Uracil     0.3       0.43 cell wall o~ molecular~
## 4 UTH1   YKR042W         Uracil     0.3       0.19 mitochondri~ molecular~
## 5 <NA>   YOL111C         Uracil     0.3       0.04 biological ~ molecular~
```

Let's use one of the dplyr verbs.

```
filter(ydat, nutrient=="Leucine")
## # A tibble: 33,178 x 7
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Leucine   0.05       0.18 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Leucine   0.05       0.16 biological~ molecular ~
## 3 QRI7   YDL104C         Leucine   0.05      -0.3  proteolysi~ metalloend~
## 4 CFT2   YLR115W         Leucine   0.05      -0.27 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Leucine   0.05      -0.59 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Leucine   0.05      -0.17 biological~ molecular ~
## # ... with 3.317e+04 more rows
ydat %>% filter(nutrient=="Leucine")
## # A tibble: 33,178 x 7
##   symbol systematic_name nutrient  rate expression bp          mf
##   <chr>  <chr>           <chr>    <dbl>      <dbl> <chr>       <chr>
## 1 SFB2   YNL049C         Leucine   0.05       0.18 ER to Golg~ molecular ~
## 2 <NA>   YNL095C         Leucine   0.05       0.16 biological~ molecular ~
## 3 QRI7   YDL104C         Leucine   0.05      -0.3  proteolysi~ metalloend~
## 4 CFT2   YLR115W         Leucine   0.05      -0.27 mRNA polya~ RNA binding
## 5 SSO2   YMR183C         Leucine   0.05      -0.59 vesicle fu~ t-SNARE ac~
## 6 PSP2   YML017W         Leucine   0.05      -0.17 biological~ molecular ~
## # ... with 3.317e+04 more rows
```

0.8.2 Nesting versus %>%

So what?

Now, think about this for a minute. What if we wanted to get the correlation between the growth rate and expression separately for each limiting nutrient only for genes in the leucine biosynthesis pathway, and return a sorted list of those correlation coeffients rounded to two digits? Mentally we would do something like this:

0. Take the `ydat` dataset
1. *then* `filter()` it for genes in the leucine biosynthesis pathway
2. *then* `group_by()` the limiting nutrient
3. *then* `summarize()` to get the correlation (`cor()`) between rate and expression
4. *then* `mutate()` to round the result of the above calculation to two significant digits
5. *then* `arrange()` by the rounded correlation coefficient above

But in code, it gets ugly. First, take the `ydat` dataset

```
ydat
```

then `filter()` it for genes in the leucine biosynthesis pathway

```
filter(ydat, bp=="leucine biosynthesis")
```

then `group_by()` the limiting nutrient

```
group_by(filter(ydat, bp=="leucine biosynthesis"), nutrient)
```

then `summarize()` to get the correlation (`cor()`) between rate and expression

```
summarize(group_by(filter(ydat, bp == "leucine biosynthesis"), nutrient), r = cor(rate,
    expression))
```

then `mutate()` to round the result of the above calculation to two significant digits

```
mutate(summarize(group_by(filter(ydat, bp == "leucine biosynthesis"), nutrient),
    r = cor(rate, expression)), r = round(r, 2))
```

then `arrange()` by the rounded correlation coefficient above

```
arrange(
  mutate(
    summarize(
      group_by(
        filter(ydat, bp=="leucine biosynthesis"),
      nutrient),
    r=cor(rate, expression)),
  r=round(r, 2)),
r)
## # A tibble: 6 x 2
##   nutrient       r
##   <chr>      <dbl>
## 1 Leucine   -0.580
## 2 Glucose   -0.04
## 3 Ammonia    0.16
## 4 Sulfate    0.33
## 5 Phosphate  0.44
## 6 Uracil     0.580
```

Now compare that with the mental process of what you're actually trying to accomplish. The way you would do this without pipes is completely inside-out and backwards from the way you express in words and in thought what you want to do. The pipe operator `%>%` allows you to pass the output data frame from one function to the input data frame to another function.

This is how we would do that in code. It's as simple as replacing the word "then" in words to the symbol `%>%` in code. (There's a keyboard shortcut that I'll use frequently to insert the `%>%` sequence – you can see what it is by clicking the *Tools* menu in RStudio, then selecting *Keyboard Shortcut Help*. On Mac, it's CMD-SHIFT-M.)

```
ydat %>%
  filter(bp=="leucine biosynthesis") %>%
  group_by(nutrient) %>%
  summarize(r=cor(rate, expression)) %>%
  mutate(r=round(r,2)) %>%
  arrange(r)
## # A tibble: 6 x 2
##   nutrient       r
##   <chr>      <dbl>
## 1 Leucine   -0.580
## 2 Glucose   -0.04
## 3 Ammonia    0.16
## 4 Sulfate    0.33
## 5 Phosphate  0.44
```

```
## 6 Uracil     0.580
```

0.8.3 Piping exercises

EXERCISE 3

Here's a warm-up round. Try the following.

Show the limiting nutrient and expression values for the gene ADH2 when the growth rate is restricted to 0.05. *Hint:* 2 pipes: `filter` and `select`.

```
ydat %>% filter(symbol=="ADH2" & rate==0.05) %>% select(nutrient, expression)
## # A tibble: 6 x 2
##   nutrient  expression
##   <chr>          <dbl>
## 1 Glucose         6.28
## 2 Ammonia         0.55
## 3 Phosphate      -4.6
## 4 Sulfate        -1.18
## 5 Leucine         4.15
## 6 Uracil          0.63
```

What are the four most highly expressed genes when the growth rate is restricted to 0.05 by restricting glucose? Show only the symbol, expression value, and GO terms. *Hint:* 4 pipes: `filter`, `arrange`, `head`, and `select`.

```
ydat %>%
  filter(nutrient=="Glucose" & rate==.05) %>%
  arrange(desc(expression)) %>%
  head(4) %>%
  select(symbol, expression, bp, mf)
## # A tibble: 4 x 4
##   symbol expression bp                  mf
##   <chr>       <dbl> <chr>               <chr>
## 1 ADH2         6.28 fermentation*       alcohol dehydrogenase activity
## 2 HSP26        5.86 response to stress* unfolded protein binding
## 3 MLS1         5.64 glyoxylate cycle    malate synthase activity
## 4 HXT5         5.56 hexose transport    glucose transporter activity*
```

When the growth rate is restricted to 0.05, what is the average expression level across all genes in the "response to stress" biological process, separately for each limiting nutrient? What about genes in the "protein biosynthesis" biological process? *Hint:* 3 pipes: `filter`, `group_by`, `summarize`.

```
ydat %>%
  filter(rate==.05 & bp=="response to stress") %>%
  group_by(nutrient) %>%
  summarize(meanexp=mean(expression))
## # A tibble: 6 x 2
##   nutrient  meanexp
##   <chr>       <dbl>
## 1 Ammonia     0.943
## 2 Glucose     0.743
## 3 Leucine     0.811
## 4 Phosphate   0.981
## 5 Sulfate     0.743
## 6 Uracil      0.731
```

```
ydat %>%
  filter(rate==.05 & bp=="protein biosynthesis") %>%
  group_by(nutrient) %>%
  summarize(meanexp=mean(expression))
## # A tibble: 6 x 2
##   nutrient  meanexp
##   <chr>       <dbl>
## 1 Ammonia    -1.61
## 2 Glucose    -0.691
## 3 Leucine    -0.574
## 4 Phosphate  -0.750
## 5 Sulfate    -0.913
## 6 Uracil     -0.880
```

EXERCISE 4

That was easy, right? How about some tougher ones.

First, some review. How do we see the number of distinct values of a variable? Use `n_distinct()` within a `summarize()` call.

```
ydat %>% summarize(n_distinct(mf))
## # A tibble: 1 x 1
##   `n_distinct(mf)`
##              <int>
## 1             1086
```

Which 10 biological process annotations have the most genes associated with them? What about molecular functions? *Hint:* 4 pipes: `group_by`, `summarize` with `n_distinct`, `arrange`, `head`.

```
ydat %>%
  group_by(bp) %>%
  summarize(n=n_distinct(symbol)) %>%
  arrange(desc(n)) %>%
  head(10)
## # A tibble: 10 x 2
##   bp                                                          n
##   <chr>                                                   <int>
## 1 biological process unknown                                269
## 2 protein biosynthesis                                      182
## 3 protein amino acid phosphorylation*                        78
## 4 protein biosynthesis*                                      73
## 5 cell wall organization and biogenesis*                     64
## 6 regulation of transcription from RNA polymerase II promoter*    49
## # ... with 4 more rows

ydat %>%
  group_by(mf) %>%
  summarize(n=n_distinct(symbol)) %>%
  arrange(desc(n)) %>%
  head(10)
## # A tibble: 10 x 2
##   mf                                    n
##   <chr>                             <int>
```

```
## 1 molecular function unknown           886
## 2 structural constituent of ribosome   185
## 3 protein binding                      107
## 4 RNA binding                           63
## 5 protein binding*                      53
## 6 DNA binding*                          44
## # ... with 4 more rows
```

How many distinct genes are there where we know what process the gene is involved in but we don't know what it does? *Hint:* 3 pipes; `filter` where `bp!="biological process unknown" & mf=="molecular function unknown"`, and after `select`ing columns of interest, pipe the output to `distinct()`. The answer should be **737**, and here are a few:

```
ydat %>%
  filter(bp!="biological process unknown" & mf=="molecular function unknown") %>%
  select(symbol, bp, mf) %>%
  distinct()
## # A tibble: 737 x 3
##   symbol bp                                          mf
##   <chr>  <chr>                                       <chr>
## 1 SFB2   ER to Golgi transport                       molecular function~
## 2 EDC3   deadenylylation-independent decapping       molecular function~
## 3 PER1   response to unfolded protein*               molecular function~
## 4 PEX25  peroxisome organization and biogenesis*     molecular function~
## 5 BNI5   cytokinesis*                                molecular function~
## 6 CSN12  adaptation to pheromone during conjugation w~ molecular function~
## # ... with 731 more rows
```

When the growth rate is restricted to 0.05 by limiting Glucose, which biological processes are the most upregulated? Show a sorted list with the most upregulated BPs on top, displaying the biological process and the average expression of all genes in that process rounded to two digits. *Hint:* 5 pipes: `filter`, `group_by`, `summarize`, `mutate`, `arrange`.

```
ydat %>%
  filter(nutrient=="Glucose" & rate==.05) %>%
  group_by(bp) %>%
  summarize(meanexp=mean(expression)) %>%
  mutate(meanexp=round(meanexp, 2)) %>%
  arrange(desc(meanexp))
## # A tibble: 881 x 2
##   bp                                              meanexp
##   <chr>                                             <dbl>
## 1 fermentation*                                      6.28
## 2 glyoxylate cycle                                   5.29
## 3 oxygen and reactive oxygen species metabolism      5.04
## 4 fumarate transport*                                5.03
## 5 acetyl-CoA biosynthesis*                           4.32
## 6 gluconeogenesis                                    3.64
## # ... with 875 more rows
```

Group the data by limiting nutrient (primarily) then by biological process. Get the average expression for all genes annotated with each process, separately for each limiting nutrient, where the growth rate is restricted to 0.05. Arrange the result to show the most upregulated processes on top. The initial result will look like the result below. Pipe this output to a `View()` statement. What's going on? Why didn't the `arrange()` work? *Hint:* 5 pipes: `filter`, `group_by`, `summarize`, `arrange`, `View`.

```
ydat %>%
  filter(rate==0.05) %>%
  group_by(nutrient, bp) %>%
  summarize(meanexp=mean(expression)) %>%
  arrange(desc(meanexp))
## # A tibble: 5,257 x 3
## # Groups:   nutrient [6]
##   nutrient  bp                                meanexp
##   <chr>     <chr>                               <dbl>
## 1 Ammonia   allantoate transport                 6.64
## 2 Ammonia   amino acid transport*                6.64
## 3 Phosphate glycerophosphodiester transport      6.64
## 4 Glucose   fermentation*                        6.28
## 5 Ammonia   allantoin transport                  5.56
## 6 Glucose   glyoxylate cycle                     5.28
## # ... with 5,251 more rows
```

Let's try to further process that result to get only the top three most upregulated biolgocal processes for each limiting nutrient. Google search "dplyr first result within group." You'll need a `filter(row_number()......)` in there somewhere. *Hint:* 5 pipes: `filter`, `group_by`, `summarize`, `arrange`, `filter(row_number()....`. *Note:* dplyr's pipe syntax used to be `%.%` before it changed to `%>%`. So when looking around, you might still see some people use the old syntax. Now if you try to use the old syntax, you'll get a deprecation warning.

```
ydat %>%
  filter(rate==0.05) %>%
  group_by(nutrient, bp) %>%
  summarize(meanexp=mean(expression)) %>%
  arrange(desc(meanexp)) %>%
  filter(row_number()<=3)
## # A tibble: 18 x 3
## # Groups:   nutrient [6]
##   nutrient  bp                                meanexp
##   <chr>     <chr>                               <dbl>
## 1 Ammonia   allantoate transport                 6.64
## 2 Ammonia   amino acid transport*                6.64
## 3 Phosphate glycerophosphodiester transport      6.64
## 4 Glucose   fermentation*                        6.28
## 5 Ammonia   allantoin transport                  5.56
## 6 Glucose   glyoxylate cycle                     5.28
## # ... with 12 more rows
```

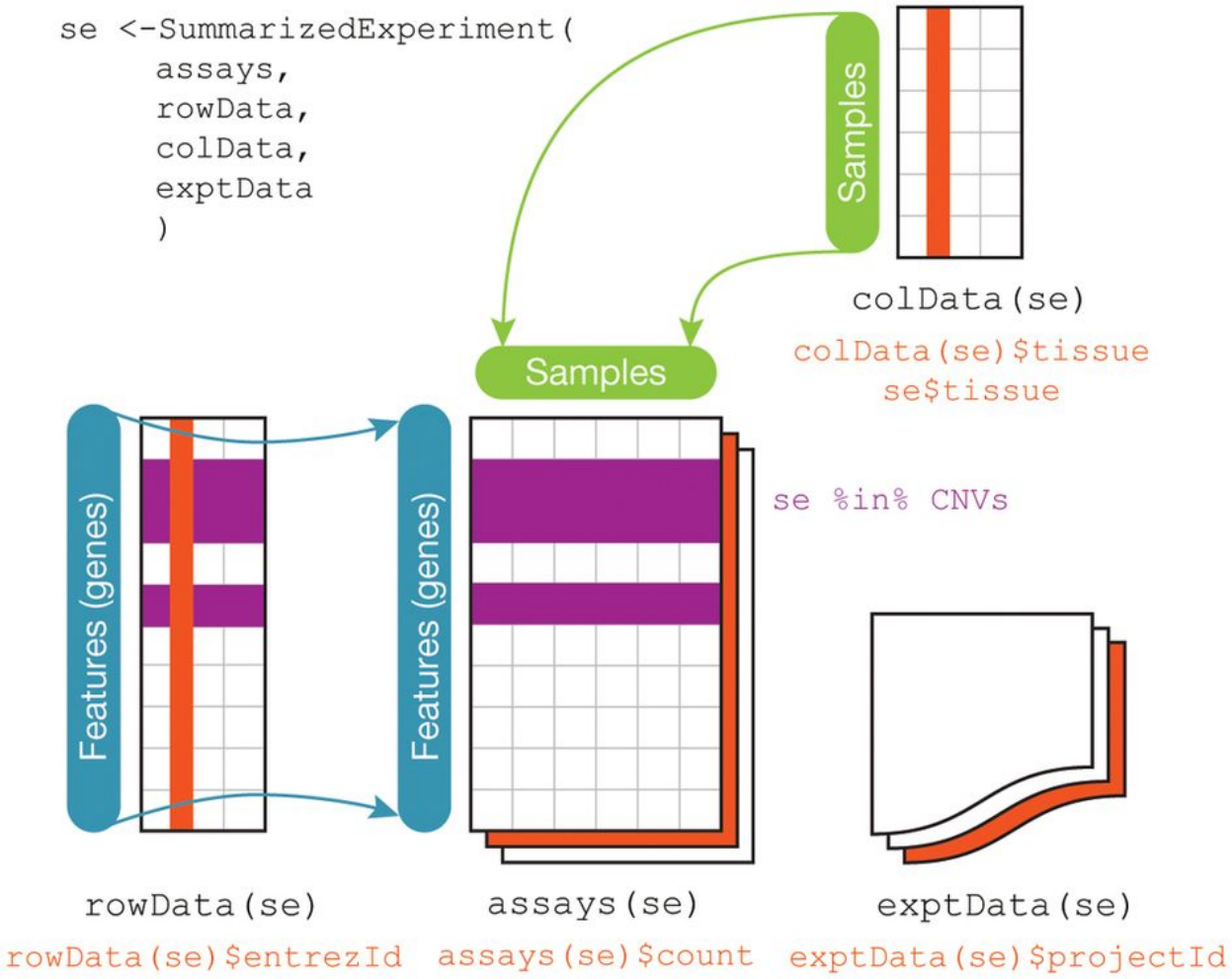
Part I

Bioconductor

1 Examples

1.1 GEOquery to multidimensional scaling

Data containers–SummarizedExperiment



Use the GEOquery package to fetch data about GSE103512.

```
library(GEOquery)
gse = getGEO("GSE103512")[[1]]
gse
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 54715 features, 280 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: GSM2772660 GSM2772661 ... GSM2772939 (280 total)
##   varLabels: title geo_accession ... weight:ch1 (72 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: 1007_PM_s_at 1053_PM_at ... AFFX-TrpnX-M_at (54715
##     total)
##   fvarLabels: ID GB_ACC ... Gene Ontology Molecular Function (16
##     total)
##   fvarMetadata: Column Description labelDescription
## experimentData: use 'experimentData(object)'
## Annotation: GPL13158
```

Examine two variables of interest, cancer type and tumor/normal status.

```
with(pData(gse),table(`cancer type:ch1`,`normal:ch1`))
##                 normal:ch1
## cancer type:ch1 no yes
##           BC    65  10
##           CRC   57  12
##           NSCLC 60   9
##           PCA   60   7
```

Information about features measured are also included.

```
         Gene Symbol                                                Gene Title
```

1007_PM_s_at DDR1 discoidin domain receptor tyrosine kinase 1 1053_PM_at RFC2 replication factor C (activator 1) 2, 40kDa 117_PM_at HSPA6 heat shock 70kDa protein 6 (HSP70B') 121_PM_at PAX8 paired box 8 1255_PM_g_at GUCA1A guanylate cyclase activator 1A (retina) 1294_PM_at UBA7 ubiquitin-like modifier activating enzyme 7 ENTREZ_GENE_ID 1007_PM_s_at 780 1053_PM_at 5982 117_PM_at 3310 121_PM_at 7849 1255_PM_g_at 2978 1294_PM_at 7318

Filter gene expression by variance to find most informative genes.

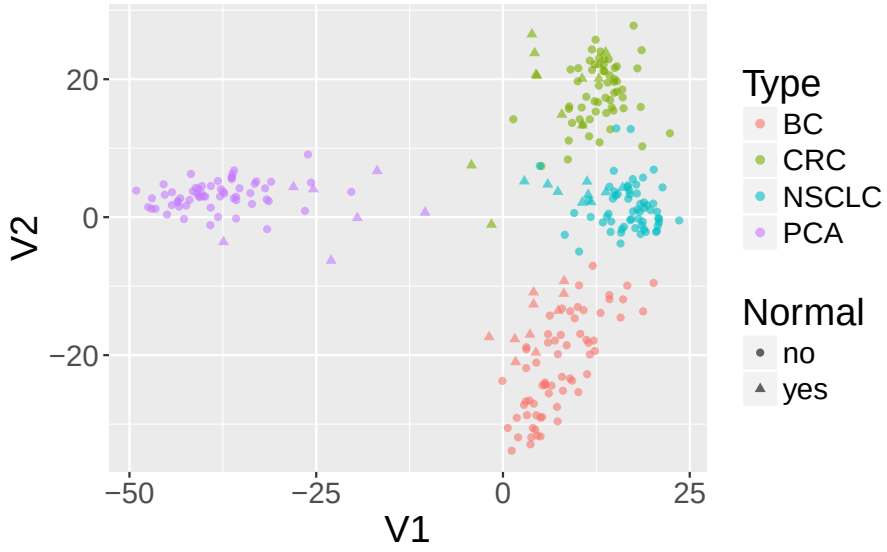
```
sds = apply(exprs(gse),1,sd)
dat = exprs(gse)[order(sds,decreasing = TRUE)[1:500],]
```

Perform multidimensional scaling and prepare for plotting.

```
mdsvals = cmdscale(dist(t(dat)))
mdsvals = as.data.frame(mdsvals)
mdsvals$Type=factor(pData(gse)[,'cancer type:ch1'])
mdsvals$Normal = factor(pData(gse)[,'normal:ch1'])
```

And do the plot.

```
library(ggplot2)
ggplot(mdsvals, aes(x=V1,y=V2,shape=Normal,color=Type)) +
    geom_point( alpha=0.6) + theme(text=element_text(size = 18))
```



A Appendix – Data Sets

- BRFSS subset
- ALL clinical data
- ALL expression data

B Appendix – Swirl

The following is from the swirl website.

> The swirl R package makes it fun and easy to learn R programming and data science. If you are new to R, have no fear.

To get started, we need to install a new package into R.

```
install.packages('swirl')
```

Once installed, we want to load it into the R workspace so we can use it.

```
library('swirl')
```

Finally, to get going, start swirl and follow the instructions.

```
swirl()
```