Introduction to R and the tidyverse

RSTR SNP edition

Kim Dill-McFarland, kadm@uw.edu

version July 01, 2020

Contents

Overview

In this workshop, we introduce you to R and RStudio at the beginner level as well as begin to work in the tidyverse. In it, we cover:

- R and RStudio including projects, scripts, and packages
- The help function
- Reading in data as a data frame and RData
- Data types
- Manipulating data within the `tidyverse`
- Running multiple linear models with `lm`

We will do all of our work in RStudio. RStudio is an integrated development and analysis environment for R that brings a number of conveniences over using R in a terminal or other editing environments.

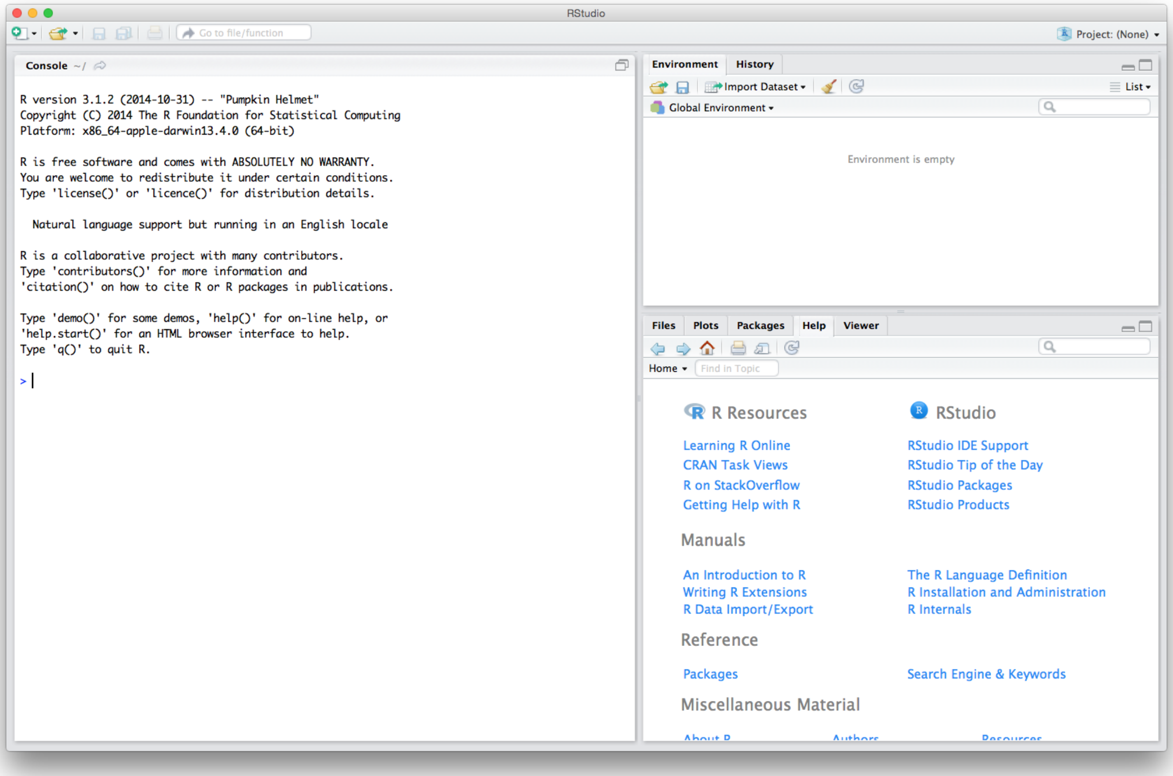
During the workshop, we will build an R script together, which will be posted as 'live_notes' after the workshop here.

Prior to the workshop

Please install R and RStudio. See the setup instructions for more details.

A Tour of RStudio

When you start RStudio, you will see something like the following window appear:



```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Notice that the window is divided into three “panes”:

- Console (the entire left side): this is your view into the R engine. You can type in R commands here and see the output printed by R. (To make it easier to tell them apart, your input is printed in blue, while the output is black.) There are several editing conveniences available: use up and down arrow keys to go back to previously entered commands, which can then be edited and re-run; TAB for completing the name before the cursor; see more in online docs.
- Environment/History (tabbed in upper right): view current user-defined objects and previously-entered commands, respectively.
- Files/Plots/Packages/Help (tabbed in lower right): as their names suggest, these are used to view the contents of the current directory, graphics created by the user, install packages, and view the built-in help pages.

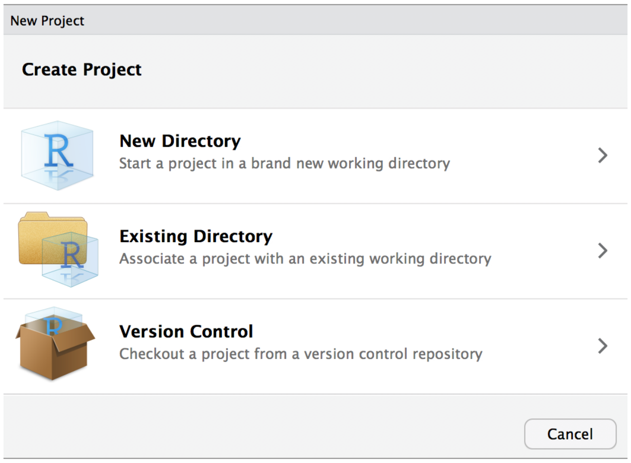
To change the look of RStudio, you can go to Tools -> Global Options -> Appearance and select colors, font size, etc. If you plan to be working for longer periods, we suggest choosing a dark background color scheme to save your computer battery and your eyes.

RStudio Projects

Projects are a great feature of RStudio. When you create a project, RStudio creates an `.Rproj` file that links all of your files and outputs to the project directory. When you import data, R automatically looks for the file in the project directory instead of you having to specify a full file path on your computer like `/Users/username/Desktop/`. R also automatically saves any output to the project directory. Finally, projects allow you to save your R environment in `.RData` so that when you close RStudio and then re-open it, you can start right where you left off without re-importing any data or re-calculating any intermediate steps.

RStudio has a simple interface to create and switch between projects, accessed from the button in the top-right corner of the RStudio window. (Labelled “Project: (None)”, initially.)

Create a Project Let’s create a project to work in for this workshop. Start by clicking the “Project” button in the upper right or going to the “File” menu. Select “New Project” and the following will appear.


New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

You can either create a project in an existing directory or make a new directory on your computer - just be sure you know where it is.

After your project is created, navigate to its directory using your Finder/File explorer. You will see the `.RProj` file has been created.

To access this project in the future, simply double-click the `RProj` and RStudio will open the project or choose File > Open Project from within an already open RStudio window.

R Scripts

R script files are the primary way in which R facilitates reproducible research. They contain the code that loads your raw data, cleans it, performs the analyses, and creates and saves visualizations. R scripts maintain a record of everything that is done to the raw data to reach the final result. That way, it is very easy to write up and communicate your methods because you have a document listing the precise steps you used to conduct your analyses. This is one of R's primary advantages compared to traditional tools like Excel, where it may be unclear how to reproduce the results.

Generally, if you are testing an operation (*e.g.* what would my data look like if I applied a log-transformation to it?), you should do it in the console (left pane of RStudio). If you are committing a step to your analysis (*e.g.* I want to apply a log-transformation to my data and then conduct the rest of my analyses on the log-transformed data), you should add it to your R script so that it is saved for future use.

Additionally, you should annotate your R scripts with comments. In each line of code, any text preceded by the `#` symbol will not execute. Comments can be useful to remind yourself and to tell other readers what a specific chunk of code does.

Let's create an R script (File > New File > R Script) and save it as `live_notes.R` in your main project directory. If you again look to the project directory on your computer, you will see `live_notes.R` is now saved there.

We will work together to create and populate the `live_notes.R` script throughout this workshop.

R packages

CRAN R packages are units of shareable code, containing functions that facilitate and enhance analyses. Let's install `tidyverse`, which is actually meta-package containing several packages useful in data manipulation and plotting. Packages are typically installed from CRAN (The Comprehensive R Archive Network), which is a database containing R itself as well as many R packages. Any package can be installed from CRAN using the `install.packages` function. You can input this into your console (as opposed to `live_notes.R`) since once a package is installed on your computer, you won't need to re-install it again.

```r
install.packages("tidyverse", Ncpus=2)
```

This can take several minutes.

After installing a package, and *every time* you open a new RStudio session, the packages you want to use need to be loaded into the R workspace with the `library` function. This tells R to access the package's functions and prevents RStudio from lags that would occur if it automatically loaded every downloaded package every time you opened it.

```r
# Data manipulation and visualization
library(tidyverse)
```

```
## -- Attaching packages --------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Bioconductor Bioconductor is another repository of R packages. It has different requirements for upload and houses many of the biology-relevant packages. To install from Bioconductor, you first install its installer from CRAN.

```r
install.packages("BiocManager")
```

Then install your package of choice using it's installer. Here, we install `limma`, a package for analysis of microarray and RNA-seq data.

If prompted, say `a` to "Update all/some/none? [a/s/n]" and `no` to "Do you want to install from sources the packages which need compilation? (Yes/no/cancel)"

```r
BiocManager::install("limma")
```

Getting started

Before doing anything in R, it is a good idea to set your random seed. Your analyses may not end up using a seed but by setting it, you ensure that *everything* is exactly reproducible.

```r
set.seed(4389)
```

Organize data

Create a directory called `data` and move the 3 data files from Dropbox to this directory.

Loading data into an R

Data frames from `.csv`, `.tsv`, etc.

One of R's most essential data structures is the data frame, which is simply a table of `m` columns by `n` rows. First, we will read in the SNP RNA-seq cleaning metrics data into RStudio using the base R `read.table` function.

Each R function follows the following basic syntax, where `Function` is the name of the function.

```
Function(argument1=..., argument2=..., ...)
```

The read.table has many arguments; however, we only need to specify 3 arguments to correctly read in our data as a data frame. For our data, we will need to specify:

- `file` - gives the path to the file that we want to load from our working directory (current project directory).
- `sep` - tells R that our data are comma-separated
- `header` - tells R that the first row in our data contains the names of the variables (columns).

We will store the data as an *object* named `snp` using the assignment operator `<-`, so that we can re-use it in our analysis.

```r
# read the data and save it as an object
snp <- read.table(file="data/Hawn_RSTR_SNPlist.PRKAG2.csv",
                  sep=",", header=TRUE)
```

Now whenever we want to use these data, we simply call `snp`

Help function

You can read up about the different arguments of a specific function by typing `?Function` or `help(Function)` in your R console.

```r
?read.table
```

You will notice that there are multiple functions of the `read.table` help page. This include similar and related functions with additional options. For example, since our data are in `.csv` format, we could've instead read them into R with `read.csv` which assumes the options `sep=",", header=TRUE` by default.

```
# read the data with different function
snp <- read.csv(file="data/Hawn_RSTR_SNPlist.PRKAG2.csv")
```

Complex data from `.RData`

You may have data that do not fit nicely into a single table or into a table at all (like plots). You can save these as `.RData`, which can be loaded directly into R. You can also save multiple tables and/or other objects in a single `.RData` file to make loading your data quick and easy. Moreover, `.RData` are automatically compressed so they take up less storage space than multiple tables.

```
load("data/RSTR_RNAseq_dat.voom.RData")
```

Notice that these data appear already named in your R environment. Object names are determined when saving so be sure to create short but descriptive names before saving to `.RData`.

Also notice that you get the message `Loading required package: limma`. If you did not have `limma` installed, you could not load these data because they are a data type specific to `limma`. You can see this by viewing the type of these data with

```
class(dat.norm.voom)
```

```
## [1] "EList"
## attr(,"package")
## [1] "limma"
```

Data types

Simple Let's return to the simpler SNP data for now. This data frame consists of 270 rows (observations) and 112 columns (variables). You can see this quickly using the dimension function `dim`

```
dim(snp)
```

```
## [1] 270 112
```

Each column and each row of a data frame are individual R vectors. R vectors are one-dimensional arrays of data. For example, we can extract column vectors from data frames using the `$` operator.

```
# Extract the SNP IDs
snp$snpID
```

```
##   [1] "JHU_7.151253265" "JHU_7.151254994" "rs5017429"       "JHU_7.151257035"
##   [5] "JHU_7.151257523" "rs28763998"      "JHU_7.151258416" "rs11982878"
##   [9] "rs17173197"      "rs77133357"      "rs7791529"       "rs4726052"
##  [13] "rs75705628"      "rs1029946"       "JHU_7.151277901" "rs6953318"
##  [17] "JHU_7.151281262" "rs2538039"       "rs12164128"      "rs4726058"
##  [21] "rs56189422"      "rs138640368"     "rs3789809"       "rs2538036"
##  [25] "JHU_7.151291496" "JHU_7.151300192" "JHU_7.151301751" "JHU_7.151304363"
##  [29] "rs6464150"       "JHU_7.151306394" "rs2374229"       "rs2538034"
##  [33] "rs4726070"       "rs6964957"       "rs4726075"       "rs3109944"
##  [37] "rs17173208"      "rs7796163"       "JHU_7.151347611" "JHU_7.151347769"
##  [41] "JHU_7.151348046" "JHU_7.151349226" "rs10251264"      "JHU_7.151349448"
##  [45] "rs4726081"       "rs868766"        "rs113108636"     "rs2727529"
##  [49] "rs953221"        "rs1105842"       "JHU_7.151364416" "JHU_7.151367983"
##  [53] "rs2727535"       "rs116226629"     "rs2727537"       "rs6978142"
##  [57] "rs6951177"       "rs2536090"       "rs56020328"      "JHU_7.151382859"
##  [61] "rs4726084"       "rs1860746"       "JHU_7.151386715" "rs10277859"
##  [65] "rs2536085"       "rs62478181"      "JHU_7.151391303" "JHU_7.151391466"
##  [69] "JHU_7.151391533" "rs73484108"      "rs143259477"     "rs76061449"
##  [73] "rs56765232"      "rs7780804"       "JHU_7.151395753" "JHU_7.151396309"
##  [77] "rs6947064"       "rs6967507"       "rs116774231"     "JHU_7.151399789"
##  [81] "JHU_7.151402498" "rs73158180"      "JHU_7.151402935" "rs60815793"
##  [85] "rs114166988"     "rs6464164"       "rs10480299"      "rs10480300"
##  [89] "JHU_7.151406825" "JHU_7.151407104" "JHU_7.151407372" "rs6945264"
##  [93] "rs4725424"       "rs79685042"      "rs11980120"      "rs10224002"
##  [97] "rs10952316"      "JHU_7.151416887" "JHU_7.151417087" "rs62478182"
## [101] "rs77903099"      "rs78308051"      "JHU_7.151420857" "rs59938864"
## [105] "rs4726086"       "JHU_7.151422899" "rs2536069"       "kgp8853492"
## [109] "JHU_7.151423487" "rs9640176"       "rs11760870"      "JHU_7.151428620"
## [113] "rs4726088"       "rs116677629"     "rs1108722"       "rs1104897"
## [117] "JHU_7.151434319" "JHU_7.151434443" "JHU_7.151436055" "rs76563225"
## [121] "rs28416498"      "rs10278273"      "rs1001116"       "rs11771330"
## [125] "rs1860740"       "rs6978479"       "JHU_7.151446136" "rs11773668"
## [129] "rs61612573"      "rs62478188"      "rs115389468"     "rs79215320"
## [133] "rs7801616"       "rs12537154"      "JHU_7.151454520" "rs6965926"
## [137] "rs12668627"      "JHU_7.151456632" "JHU_7.151456735" "rs1104840"
## [141] "rs11520869"      "rs11982124"      "rs7800069"       "JHU_7.151464922"
## [145] "rs3934597"       "JHU_7.151466470" "JHU_7.151468071" "rs4726091"
## [149] "JHU_7.151473009" "rs7809589"       "JHU_7.151474450" "rs3935852"
## [153] "JHU_7.151474597" "rs73728288"      "rs55964814"      "JHU_7.151475760"
## [157] "rs9632641"       "rs78667038"      "rs80039831"      "rs78291918"
## [161] "JHU_7.151479513" "JHU_7.151480349" "JHU_7.151480717" "rs62478216"
## [165] "rs73728291"      "rs6953882"       "rs11975504"      "rs6959849"
## [169] "JHU_7.151484100" "rs4425665"       "rs56859244"      "rs79637471"
## [173] "rs11770585"      "JHU_7.151489803" "rs34735358"      "rs79918575"
## [177] "rs34659848"      "JHU_7.151491553" "rs34698098"      "rs56037571"
## [181] "rs12112702"      "rs4726098"       "rs4128396"       "rs76923703"
## [185] "rs9648724"       "rs4442045"       "rs13224758"      "rs56729506"
## [189] "rs7802319"       "rs12539356"      "rs377649951"     "rs74535090"
## [193] "rs56129741"      "rs76727740"      "JHU_7.151513101" "rs4726101"
## [197] "rs1881633"       "rs77961133"      "rs114008100"     "rs1881632"
## [201] "JHU_7.151517481" "JHU_7.151517732" "rs13309585"      "rs66972858"
## [205] "rs11770376"      "rs75379928"      "rs73487739"      "JHU_7.151520306"
## [209] "rs13225852"      "rs11773373"      "rs7800364"       "rs116508619"
## [213] "JHU_7.151528914" "JHU_7.151529175" "rs13233608"      "JHU_7.151529363"
## [217] "JHU_7.151529449" "rs77954465"      "JHU_7.151529595" "JHU_7.151529647"
## [221] "JHU_7.151529991" "rs11771216"      "rs1881625"       "rs11764602"
## [225] "rs74782537"      "rs75697045"      "JHU_7.151534273" "rs114497371"
## [229] "rs6950343"       "rs61462038"      "rs9648727"       "rs114016536"
## [233] "JHU_7.151539874" "rs10261999"      "rs62480473"      "rs7795096"
## [237] "rs57194047"      "rs1881639"       "rs3813852"       "rs11971588"
## [241] "JHU_7.151551975" "rs13245627"      "rs6961830"       "rs11971243"
## [245] "rs6955784"       "rs7789674"       "rs9640300"       "rs12703164"
## [249] "rs4726104"       "JHU_7.151562456" "rs59404899"      "JHU_7.151563452"
## [253] "JHU_7.151563721" "JHU_7.151563769" "JHU_7.151563945" "rs73160061"
## [257] "JHU_7.151564516" "JHU_7.151564713" "rs56118280"      "rs78168617"
## [261] "JHU_7.151566710" "rs12669153"      "rs13240743"      "JHU_7.151566976"
## [265] "rs13235096"      "rs4725435"       "rs75188986"      "rs35503973"
## [269] "JHU_7.151572648" "rs116605521"
```

R objects have several different classes (types). Our data frame contains 2 R data types. The base R `class` function will tell you what data type an object is.

```
class(snp)
```

```
## [1] "data.frame"
```

```
class(snp$snpID)
```

```
## [1] "character"
```

```
class(snp$CHR)
```

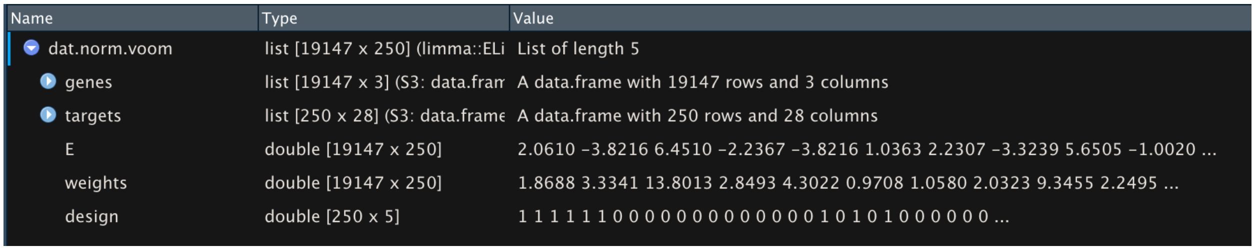
```
## [1] "integer"
```

We see that our `snpID` column is `character`, meaning it is non-numeric. On the other hand, `CHR` is `integer`, meaning whole number numeric.

Common data types not found in these data include:

- `factor`: non-numeric value with a set number of unique levels
- `numeric`: numeric with decimal values
- `logical`: TRUE/FALSE designation

Complex (`S3, S4`) Now, let's look at the `limma EList` data. These data are in `S3` format meaning they have 3 dimensions. In essence, they are a list of multiple data frames and vectors. If you click on the `dat.norm.voom` object in your Environment tab, you will see multiple pieces.



Name	Type	Value
dat.norm.voom	list [19147 x 250] (limma::ELi	List of length 5
genes	list [19147 x 3] (S3: data.fram	A data.frame with 19147 rows and 3 columns
targets	list [250 x 28] (S3: data.frame	A data.frame with 250 rows and 28 columns
E	double [19147 x 250]	2.0610 -3.8216 6.4510 -2.2367 -3.8216 1.0363 2.2307 -3.3239 5.6505 -1.0020 ...
weights	double [19147 x 250]	1.8688 3.3341 13.8013 2.8493 4.3022 0.9708 1.0580 2.0323 9.3455 2.2495 ...
design	double [250 x 5]	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 ...

`genes` and `targets` are data frames while the other pieces are matrices. You can again see this with `class` only this time you specify a part of the `dat.norm.voom` object with `$`

```
class(dat.norm.voom$genes)
```

```
## Loading required package: limma
```

```
## [1] "data.frame"
```

```
class(dat.norm.voom$E)
```

```
## [1] "matrix" "array"
```

Or going deeper, you can specify one column in the `genes` data frame

```
class(dat.norm.voom$genes$symbol)
```

```
## [1] "character"
```

Of note, working with `S4` objects is very similar to `S3` except that they are accessed with `@` instead of `$`.

Working with vectors and data frames

Operating on vectors

A large proportion of R functions operate on vectors to perform quick computations over their values. Here are some examples:

```r
# Compute the variance of library size
var(dat.norm.voom$targets$lib.size)
```

```
## [1] 6.523173e+12
```

```r
# Find whether any samples have greater than 1 billion  sequences
dat.norm.voom$targets$lib.size > 1E9
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [157] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [181] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [193] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [205] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [217] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [229] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [241] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
# Find the unique values of SNP type
unique(snp$type)
```

```
## [1] "exon"             "intron, exon"     "intron"           "intron, promoter"
```

Using the correct class

Functions executed on an object in R may respond exclusively to one or more data types or may respond differently depending on the data type. For example, you cannot take the mean of a factor or character.

```r
# Compute the mean of sampID
mean(snp$snpID)
```

```
## Warning in mean.default(snp$snpID): argument is not numeric or logical:
## returning NA
```

```
## [1] NA
```

Subsetting vectors and data frames

Since vectors are 1D arrays of a defined length, their individual values can be retrieved using vector indices. R uses 1-based indexing, meaning the first value in an R vector corresponds to the index 1. Each subsequent element increases the index by 1. For example, we can extract the value of the 5th element of the `snpID` vector using the square bracket operator `[ ]` like so.

```
snp$snpID[5]
```

```
## [1] "JHU_7.151257523"
```

In contrast, data frames are 2D arrays so indexing is done across both dimensions as `[rows, columns]`. So, we can extract the same oxygen value directly from the data frame knowing it is in the 5th row and 1st column.

```
snp[5, 1]
```

```
## [1] "JHU_7.151257523"
```

The square bracket operator is most often used with logical vectors (TRUE/FALSE) to subset data. For example, we can subset our data frame to all observations (rows) that are only exons.

```
# Create logical vector for which lib.size values are > 10 million
logical.vector <- snp$type == "exon"
#View vector
logical.vector
```

```
##   [1]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [157] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [181] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [193] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [205] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [217] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [229] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [241] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [253] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [265] FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
#Apply vector to data frame to select only observations where the logical vector is TRUE
snp[logical.vector, ]
```

```
##                snpID        rsID gene_id symbol type CHR       POS allele.0
## 1   JHU_7.151253265 rs199973452   51422 PRKAG2 exon   7 151253266        A
## 6        rs28763998  rs28763998   51422 PRKAG2 exon   7 151257665        A
## 270     rs116605521 rs116605521   51422 PRKAG2 exon   7 151573647        C
##     allele.1 X84165.1.06 X84182.1.02 X84183.1.02 X84183.1.05 X84186.1.03
## 1          C         0/1         0/0         0/0         0/0         0/0
## 6          G         0/0         0/0         0/0         0/0         0/0
## 270        A         0/0         0/0         0/0         0/1         0/0
##     X84186.1.04 X84218.1.02 X84218.1.05 X84218.1.06 X84222.1.03 X84222.1.17
## 1           0/0         0/0         0/0         0/0         0/0         1/1
## 6           0/0         0/1         0/0         0/1         0/0         0/0
## 270         0/0         0/0         0/0         0/0         0/0         0/0
##     X84222.1.18 X84222.1.19 X84222.1.20 X84222.1.23 X84224.1.13 X84224.1.14
## 1           0/1         1/1         0/1         0/0         1/1         0/1
## 6           0/0         0/0         0/0         0/0         0/0         0/1
## 270         0/0         0/0         0/0         0/0         0/0         0/1
##     X84237.1.02 X84275.1.03 X84275.1.04 X84299.1.04 X84299.1.06 X84302.1.02
## 1           0/0         0/0         0/0         0/1         0/1         0/1
## 6           0/1         0/0         0/0         0/0         0/0         0/0
## 270         0/0         0/0         0/1         0/0         0/0         0/0
##     X84302.1.03 X84317.1.03 X84319.1.02 X84319.1.03 X84329.1.08 X84334.1.03
## 1           0/0         0/0         0/0         0/1         0/1         0/0
## 6           0/0         0/0         0/0         0/0         0/0         0/0
## 270         0/0         0/0         0/0         0/0         0/0         0/0
##     X84348.1.02 X84351.1.02 X84351.1.03 X84358.1.03 X84358.1.04 X84406.1.04
## 1           0/0         0/0         0/0         0/0         0/0         0/0
## 6           0/0         0/1         0/0         0/0         0/1         0/1
## 270         0/0         0/0         0/0         0/0         0/0         0/0
##     X84407.1.06 X84437.1.02 X84437.1.03 X84437.1.04 X84441.1.02 X84441.1.03
## 1           0/1         0/0         0/0         0/0         0/0         0/1
## 6           0/0         0/1         0/1         0/1         0/1         0/0
## 270         0/0         0/1         0/0         0/0         0/0         0/0
##     X84441.1.04 X84457.1.02 X84495.1.05 X84523.1.02 X84523.1.04 X88159.1.04
## 1           0/1         0/0         0/1         0/0         0/1         0/0
## 6           0/1         0/0         0/0         1/1         0/0         0/0
## 270         0/0         0/0         0/1         0/0         0/0         0/0
##     X88234.1.04 X88253.1.03 X88553.1.03 X88620.1.03 X88899.1.02 X89011.1.03
## 1           0/1         0/1         0/0         0/1         0/0         0/0
## 6           0/0         0/0         0/0         0/0         0/0         0/0
## 270         0/1         0/0         0/0         0/0         0/0         0/0
##     X89011.1.04 X89048.1.03 X89260.1.05 X89337.1.06 X89337.1.11 X89448.1.05
## 1           0/0         0/0         0/0         0/0         0/1         0/0
## 6           1/1         0/1         0/1         0/1         0/0         0/1
## 270         0/0         0/0         0/0         0/0         0/0         0/0
##     X89773.1.04 X89880.1.02 X89902.1.07 X90118.1.03 X90480.1.02 X90561.1.03
## 1           0/1         0/1         1/1         1/1         0/0         0/1
## 6           0/0         0/0         0/0         0/0         0/0         0/0
## 270         0/1         0/0         0/0         0/1         0/0         0/0
##     X90561.1.05 X90689.1.07 X90826.1.02 X90897.1.02 X90897.1.03 X90899.1.02
## 1           0/0         0/0         0/1         0/1         0/0         0/0
## 6           0/0         0/0         0/1         0/0         0/1         0/0
## 270         0/0         0/0         0/0         0/1         0/1         0/0
##     X91053.1.04 X91083.1.05 X91152.1.02 X91352.1.03 X91360.1.04 X91360.1.05
## 1           0/0         0/0         0/1         1/1         0/1         0/1
## 6           0/1         0/0         0/0         0/0         0/0         0/0
## 270         0/0         0/1         0/0         0/1         0/1         0/0
##     X92065.1.04 X92527.1.06 X92527.1.08 X92527.1.15 X92529.1.02 X92607.1.02
## 1           0/0         0/1         0/0         0/1         0/0         0/0
## 6           0/0         0/0         0/0         0/0         0/0         0/0
## 270         0/0         0/0         0/1         0/0         0/0         0/0
##     X92608.1.07 X92608.1.08 X92608.1.09 X92902.1.03 X93132.1.02 X93184.1.04
## 1           0/0         0/0         0/0         0/0         0/0         0/1
## 6           0/1         0/1         0/1         0/0         0/0         0/0
## 270         0/0         0/0         0/0         0/0         0/1         0/0
##     X93254.1.03 X93260.1.03 X93334.1.02 X93444.1.03 X93506.1.02 X93541.1.03
## 1           0/0         0/1         0/0         0/0         0/0         0/1
## 6           0/0         0/0         0/0         0/1         0/0         0/0
## 270         0/0         0/0         0/0         0/0         0/0         0/0
##     X93541.1.07 X93597.1.03 X93597.1.05 X93664.1.03 X93664.1.07 X93705.1.04
## 1           1/1         0/0         0/0         0/0         1/1         0/1
## 6           0/0         0/0         0/0         0/0         0/0         0/0
## 270         0/0         0/1         0/0         0/0         0/0         0/0
##     X93774.1.05 X94295.1.02
## 1           0/0         0/0
## 6           0/0         0/0
## 270         0/0         0/0
```

Subsetting is extremely useful when working with large data. We will learn more complex subsets next using the tidyverse. But first…

Quick reference: Conditional statements

Statement	Meaning
`<-`	Assign to object in environment
`==`	Equal to
`!=`	Not equal to
`>`	Greater than
`>=`	Greater than or equal to
`<`	Less than
`<=`	Less than or equal to
`%in%`	In or within
`is.na()`	Is missing, *e.g* NA
`!is.na()`	Is not missing
`&`	And
`|`	Or

Exercises: Part 1

1. Install `tidyverse` and `limma` packages.

*Please note that if you have **R v3.3 or older**, you may not be able to install `tidyverse`. In this case, you need to separately install each package within the tidyverse. This includes:* `readr`, `tibble`, `dplyr`, `tidyr`, `stringr`, `ggplot2`, `purr`, `forcats`

2. Using help to identify the necessary arguments for the log function compute the natural logarithm of 4, base 2 logarithm of 4, and base 4 logarithm of 4.

Using the `snp` data frame:

3. Using an R function, determine what data type the `allele.0` variable is.
4. Using indexing and the square bracket operator `[]`:
   - determine what `rsIS` value occurs in the 20th row
   - return the cell where `snpID` equals “rs76923703”
5. Subset the data to observations where `allele.0` equals “A” or “T”. *Hint*: Use a logical vector.

RSTR SNP analysis example

The next section will be a targeted analysis of PRKAG2 SNPs and their relationship with PRKAG2 gene expression. In the process, we will cover some (but not all) functions in the `tidyverse`.

Currently, our data are

`snp`: snp information including location and genotype. Rows are SNPs

```
##             snpID        rsID gene_id symbol         type CHR       POS
## 1 JHU_7.151253265 rs199973452   51422 PRKAG2         exon   7 151253266
## 2 JHU_7.151254994   rs5016446   51422 PRKAG2 intron, exon   7 151254995
## 3       rs5017429   rs5017429   51422 PRKAG2 intron, exon   7 151255160
## 4 JHU_7.151257035 rs141296078   51422 PRKAG2 intron, exon   7 151257036
## 5 JHU_7.151257523  rs75273149   51422 PRKAG2 intron, exon   7 151257524
##   allele.0 allele.1 X84165.1.06
## 1        A        C         0/1
## 2        A        C         0/0
## 3        A        G         0/0
## 4        G        A         0/0
## 5        A        G         0/1
```

`dat.norm.voom$genes`: gene information including HGNC symbol and ENSEMBL ID. Rows are genes

```
##          symbol         ensembl         locus.group
## A1BG       A1BG ENSG00000121410 protein-coding gene
## A1CF       A1CF ENSG00000148584 protein-coding gene
## A2M         A2M ENSG00000175899 protein-coding gene
## A2ML1     A2ML1 ENSG00000166535 protein-coding gene
## A3GALT2 A3GALT2 ENSG00000184389 protein-coding gene
```

`dat.norm.voom$targets`: sample metadata including FULLIDNO, RSID, age, sex, etc. Rows are samples as RSID_condition

```
##                group lib.size norm.factors          libID   sampID
## RS102051_MEDIA     1  7069703    1.0656982 RS102051_MEDIA RS102051
## RS102051_TB        1  5006956    0.7595452    RS102051_TB RS102051
## RS102057_MEDIA     1  7611102    1.1345472 RS102057_MEDIA RS102057
## RS102057_TB        1  5961425    0.9915710    RS102057_TB RS102057
## RS102231_MEDIA     1  9656030    1.0779592 RS102231_MEDIA RS102231
```

```
##  [1] "group"                         "lib.size"
##  [3] "norm.factors"                  "libID"
##  [5] "sampID"                        "condition"
##  [7] "dataset"                       "FULLIDNO"
##  [9] "KIFW_VISIT_TYPE"               "KIFW_COLLECTION_DATE"
## [11] "Sample_Group"                  "CONCORDANT_STATUS"
## [13] "KQFT_TST_QFT_CLASSIFICATION_C" "KQFT_CERTAINTY"
## [15] "M0_KCVSEX"                     "M0_KCVAGE"
## [17] "M0_KCVDATE"                    "avgBMI"
## [19] "HIVSTAT_CURRENT"               "KCB_BCGSCAR"
## [21] "RISK_SCORE"                    "KCB_RELATION"
## [23] "FULLMID"                       "FULLFID"
## [25] "family.3"                      "family.2"
## [27] "family.1"                      "sample.weights"
```

`dat.norm.voom$E`: voom normalized log counts per million (CPM). Rows are genes, columns are samples

```
##          RS102051_MEDIA RS102051_TB RS102057_MEDIA RS102057_TB RS102231_MEDIA
## A1BG          2.060993   2.230655       2.411744   1.633796       1.705850
## A1CF         -3.821650  -3.323934      -2.343143  -1.990695      -4.271430
## A2M           6.450980   5.650481       6.657796   5.865212       7.459464
## A2ML1        -2.236687  -1.002006      -1.606177  -1.990695      -1.949502
## A3GALT2      -3.821650  -3.323934      -3.928106  -3.575658      -4.271430
```

There is also `dat.norm.voom$weights` (gene specific weights) and `dat.norm.voom$design` (design matrix) which were used for voom normalization but will not be needed in this analysis.

What is the tidyverse?

The R tidyverse is a set of packages aimed at making, manipulating, and plotting tidy data. Everything we've done thus far has been in base R. Now we will move into the tidyverse!

While base R can accomplish most tasks, base R code is rather slow and can quickly become extremely convoluted. Compared to base R, `tidyverse` code runs much faster. It is also much more readable because all operations are based on using *verbs* (select, filter, mutate…) rather than base R's more difficult to read indexing system (brackets, parentheses…).

First, we need to load the package, which will give us a message detailing all the packages this one command loads for us.

```
library(tidyverse)
```

Common tidyverse functions

Though we will not use all of these in this workshop, here is a list of some commonly used tidyverse functions.

The `dplyr` package provides many functions for manipulating data frames including typical tasks like:

- `select` a subset of variables (columns)
- `filter` out a subset of observations (rows)
- `rename` variables
- `arrange` the observations by sorting a variable in ascending or descending order
- `mutate` all values of a variable (apply a transformation)
- `group_by` a variable and `summarise` data by the grouped variable
- `*_join` two data frames into a single data frame

The `tidyr` package contains functions for manipulating entire data frames including

- `pivot_longer` convert wide to long format
- `pivot_wider` convert long to wide format

Loading data with readr

The readr functions `read_csv` and `read_tsv` help read in data at quick speeds compared to base R's `read.csv` and `read.tsv` functions. Furthermore, readr's data loading functions automatically parse your data into data types (numeric, character, etc) based on the values in the first 1000 rows.

Let's start by re-loading in the data we previously loaded with base R's `read.table`.

```
snp <- read_csv(file="data/Hawn_RSTR_SNPlist.PRKAG2.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   gene_id = col_double(),
##   CHR = col_double(),
##   POS = col_double()
## )
## See spec(...) for full column specifications.
```

We can then view all the classes it automatically assigned to our variables.

```
spec(snp)
```

```
## cols(
##   snpID = col_character(),
##   rsID = col_character(),
##   gene_id = col_double(),
##   symbol = col_character(),
##   type = col_character(),
##   CHR = col_double(),
##   POS = col_double(),
##   allele.0 = col_character(),
##   allele.1 = col_character(),
##   `84165-1-06` = col_character(),
##   `84182-1-02` = col_character(),
##   `84183-1-02` = col_character(),
##   `84183-1-05` = col_character(),
##   `84186-1-03` = col_character(),
##   `84186-1-04` = col_character(),
##   `84218-1-02` = col_character(),
##   `84218-1-05` = col_character(),
##   `84218-1-06` = col_character(),
##   `84222-1-03` = col_character(),
##   `84222-1-17` = col_character(),
##   `84222-1-18` = col_character(),
##   `84222-1-19` = col_character(),
##   `84222-1-20` = col_character(),
##   `84222-1-23` = col_character(),
##   `84224-1-13` = col_character(),
##   `84224-1-14` = col_character(),
##   `84237-1-02` = col_character(),
##   `84275-1-03` = col_character(),
##   `84275-1-04` = col_character(),
##   `84299-1-04` = col_character(),
##   `84299-1-06` = col_character(),
##   `84302-1-02` = col_character(),
##   `84302-1-03` = col_character(),
##   `84317-1-03` = col_character(),
##   `84319-1-02` = col_character(),
##   `84319-1-03` = col_character(),
##   `84329-1-08` = col_character(),
##   `84334-1-03` = col_character(),
##   `84348-1-02` = col_character(),
##   `84351-1-02` = col_character(),
##   `84351-1-03` = col_character(),
##   `84358-1-03` = col_character(),
##   `84358-1-04` = col_character(),
##   `84406-1-04` = col_character(),
##   `84407-1-06` = col_character(),
##   `84437-1-02` = col_character(),
##   `84437-1-03` = col_character(),
##   `84437-1-04` = col_character(),
##   `84441-1-02` = col_character(),
##   `84441-1-03` = col_character(),
##   `84441-1-04` = col_character(),
##   `84457-1-02` = col_character(),
##   `84495-1-05` = col_character(),
##   `84523-1-02` = col_character(),
##   `84523-1-04` = col_character(),
##   `88159-1-04` = col_character(),
##   `88234-1-04` = col_character(),
##   `88253-1-03` = col_character(),
##   `88553-1-03` = col_character(),
##   `88620-1-03` = col_character(),
##   `88899-1-02` = col_character(),
##   `89011-1-03` = col_character(),
##   `89011-1-04` = col_character(),
##   `89048-1-03` = col_character(),
##   `89260-1-05` = col_character(),
##   `89337-1-06` = col_character(),
##   `89337-1-11` = col_character(),
##   `89448-1-05` = col_character(),
##   `89773-1-04` = col_character(),
##   `89880-1-02` = col_character(),
##   `89902-1-07` = col_character(),
##   `90118-1-03` = col_character(),
##   `90480-1-02` = col_character(),
##   `90561-1-03` = col_character(),
##   `90561-1-05` = col_character(),
##   `90689-1-07` = col_character(),
##   `90826-1-02` = col_character(),
##   `90897-1-02` = col_character(),
##   `90897-1-03` = col_character(),
##   `90899-1-02` = col_character(),
##   `91053-1-04` = col_character(),
##   `91083-1-05` = col_character(),
##   `91152-1-02` = col_character(),
##   `91352-1-03` = col_character(),
##   `91360-1-04` = col_character(),
##   `91360-1-05` = col_character(),
##   `92065-1-04` = col_character(),
##   `92527-1-06` = col_character(),
##   `92527-1-08` = col_character(),
##   `92527-1-15` = col_character(),
##   `92529-1-02` = col_character(),
##   `92607-1-02` = col_character(),
##   `92608-1-07` = col_character(),
##   `92608-1-08` = col_character(),
##   `92608-1-09` = col_character(),
##   `92902-1-03` = col_character(),
##   `93132-1-02` = col_character(),
##   `93184-1-04` = col_character(),
##   `93254-1-03` = col_character(),
##   `93260-1-03` = col_character(),
##   `93334-1-02` = col_character(),
##   `93444-1-03` = col_character(),
##   `93506-1-02` = col_character(),
##   `93541-1-03` = col_character(),
##   `93541-1-07` = col_character(),
##   `93597-1-03` = col_character(),
##   `93597-1-05` = col_character(),
##   `93664-1-03` = col_character(),
##   `93664-1-07` = col_character(),
##   `93705-1-04` = col_character(),
##   `93774-1-05` = col_character(),
##   `94295-1-02` = col_character()
## )
```

You'll see that all our numeric/integer values are now `double`. This is another number class in R that stands for "double precision floating point numbers". Under the hood, doubles are more exact than numeric and more flexible than integer. So, tidyverse preferentially assigns number data to double.

Also note that the tidyverse was able to read the FULLIDNO column names as is without adding the leading "X" that read.table did.

Not that there is no tidyverse-specific function for loading `.RData`. You would still load it with `load`.

Data wrangling

Our goal is to get the following data frame. When you have multiple (and complex) objects like we have in these data, it is often a good idea to sketch our your goal before beginning the wrangling process.

sample	PRKAG2	RSID	FULLIDNO	condition	snp1	snp2	age	...
sample1	-2	RS102051	84222-1-20	MEDIA	0	1	12	...
sample2	2	RS102051	84222-1-20	TB	1	1	11	...

Then we can model with a linear model like

```
PRKAG2 ~ condition*snp1 + age
```

Extract from `S3` object

Our metadata is contained in `snp` and `dat.norm.voom$targets` and indexed by different IDs (RSID vs FULLIDNO). This includes everything in our goal data frame except PRKAG2 expression.

First extract the `dat.norm.voom` metadata into a separate data frame to work with.

```
meta <- dat.norm.voom$targets
```

Select columns

Then separate the `snp` data into its two main parts: info on SNPs vs. genotypes of samples.

We use the tidyverse function `select` to keep the columns of interest either listing them all by name or using `A:B` to indicate keeping all columns from A to B.

```
meta.snp <- select(snp,
                   snpID, rsID, gene_id, symbol, type,
                   CHR, POS, allele.0, allele.1)

geno <- select(snp,
               snpID, `84165-1-06`:`94295-1-02`)
```

Note that since the FULLIDNO are not accepted variable names (they start with a number), we need to surround them with “ for R to treat them as column names. This was also why read.table appended an “X” to those names before.

Pivot wide to long format

We want to combine `geno` and `meta` matching their FULLIDNO. However, FULLIDNO are column names in `geno` while they are in 1 single variable in `meta`. If we transpose `geno`, we will have a FULLIDNO variable to match with `meta`.

There are several ways to do this including a simple transpose with `t()`. However, this function does not play well with column vs row names and get mess up your data. So, we'll use the tidyverse!

First, we convert the `geno` table from wide format (1 row per SNP) to long format (1 column per variable). Here, we specify the `geno` data frame, the variables we want to collapse (written as all but snpID with `-snpID`), and then the names we want to give the new columns.

Pivoting can be tricky so it is best to *always* look at your table before and after!

```
geno
```

```
## # A tibble: 270 x 104
##    snpID `84165-1-06` `84182-1-02` `84183-1-02` `84183-1-05` `84186-1-03`
##    <chr> <chr>        <chr>        <chr>        <chr>        <chr>
##  1 JHU_~ 0/1          0/0          0/0          0/0          0/0
##  2 JHU_~ 0/0          0/0          0/1          0/1          0/1
##  3 rs50~ 0/0          1/1          0/0          0/0          0/0
##  4 JHU_~ 0/0          0/0          0/0          0/0          0/0
##  5 JHU_~ 0/1          0/0          0/0          0/0          0/0
##  6 rs28~ 0/0          0/0          0/0          0/0          0/0
##  7 JHU_~ 0/0          0/0          0/0          0/0          0/0
##  8 rs11~ 0/1          0/1          0/0          0/0          0/0
##  9 rs17~ 0/1          0/0          0/1          0/1          0/0
## 10 rs77~ 0/0          0/1          0/1          0/0          0/0
## # ... with 260 more rows, and 98 more variables: `84186-1-04` <chr>,
## #   `84218-1-02` <chr>, `84218-1-05` <chr>, `84218-1-06` <chr>,
## #   `84222-1-03` <chr>, `84222-1-17` <chr>, `84222-1-18` <chr>,
## #   `84222-1-19` <chr>, `84222-1-20` <chr>, `84222-1-23` <chr>,
## #   `84224-1-13` <chr>, `84224-1-14` <chr>, `84237-1-02` <chr>,
## #   `84275-1-03` <chr>, `84275-1-04` <chr>, `84299-1-04` <chr>,
## #   `84299-1-06` <chr>, `84302-1-02` <chr>, `84302-1-03` <chr>,
## #   `84317-1-03` <chr>, `84319-1-02` <chr>, `84319-1-03` <chr>,
## #   `84329-1-08` <chr>, `84334-1-03` <chr>, `84348-1-02` <chr>,
## #   `84351-1-02` <chr>, `84351-1-03` <chr>, `84358-1-03` <chr>,
## #   `84358-1-04` <chr>, `84406-1-04` <chr>, `84407-1-06` <chr>,
## #   `84437-1-02` <chr>, `84437-1-03` <chr>, `84437-1-04` <chr>,
## #   `84441-1-02` <chr>, `84441-1-03` <chr>, `84441-1-04` <chr>,
## #   `84457-1-02` <chr>, `84495-1-05` <chr>, `84523-1-02` <chr>,
## #   `84523-1-04` <chr>, `88159-1-04` <chr>, `88234-1-04` <chr>,
## #   `88253-1-03` <chr>, `88553-1-03` <chr>, `88620-1-03` <chr>,
## #   `88899-1-02` <chr>, `89011-1-03` <chr>, `89011-1-04` <chr>,
## #   `89048-1-03` <chr>, `89260-1-05` <chr>, `89337-1-06` <chr>,
## #   `89337-1-11` <chr>, `89448-1-05` <chr>, `89773-1-04` <chr>,
## #   `89880-1-02` <chr>, `89902-1-07` <chr>, `90118-1-03` <chr>,
## #   `90480-1-02` <chr>, `90561-1-03` <chr>, `90561-1-05` <chr>,
## #   `90689-1-07` <chr>, `90826-1-02` <chr>, `90897-1-02` <chr>,
## #   `90897-1-03` <chr>, `90899-1-02` <chr>, `91053-1-04` <chr>,
## #   `91083-1-05` <chr>, `91152-1-02` <chr>, `91352-1-03` <chr>,
## #   `91360-1-04` <chr>, `91360-1-05` <chr>, `92065-1-04` <chr>,
## #   `92527-1-06` <chr>, `92527-1-08` <chr>, `92527-1-15` <chr>,
## #   `92529-1-02` <chr>, `92607-1-02` <chr>, `92608-1-07` <chr>,
## #   `92608-1-08` <chr>, `92608-1-09` <chr>, `92902-1-03` <chr>,
## #   `93132-1-02` <chr>, `93184-1-04` <chr>, `93254-1-03` <chr>,
## #   `93260-1-03` <chr>, `93334-1-02` <chr>, `93444-1-03` <chr>,
## #   `93506-1-02` <chr>, `93541-1-03` <chr>, `93541-1-07` <chr>,
## #   `93597-1-03` <chr>, `93597-1-05` <chr>, `93664-1-03` <chr>,
## #   `93664-1-07` <chr>, `93705-1-04` <chr>, `93774-1-05` <chr>,
## #   `94295-1-02` <chr>
```

```r
geno <- pivot_longer(geno,
                -snpID, names_to="FULLIDNO", values_to="genotype")

geno
```

```
## # A tibble: 27,810 x 3
##    snpID           FULLIDNO   genotype
##    <chr>           <chr>      <chr>
##  1 JHU_7.151253265 84165-1-06 0/1
##  2 JHU_7.151253265 84182-1-02 0/0
##  3 JHU_7.151253265 84183-1-02 0/0
##  4 JHU_7.151253265 84183-1-05 0/0
##  5 JHU_7.151253265 84186-1-03 0/0
##  6 JHU_7.151253265 84186-1-04 0/0
##  7 JHU_7.151253265 84218-1-02 0/0
##  8 JHU_7.151253265 84218-1-05 0/0
##  9 JHU_7.151253265 84218-1-06 0/0
## 10 JHU_7.151253265 84222-1-03 0/0
## # ... with 27,800 more rows
```

Mutate 0/0 to 0

While we have a single column for genotype, now is a good time to convert it to the format we'll use in linear models.

The current genotype data are character variables of 0/0, 0/1, 1/1. When we run our linear models later on, we want these to be numeric variables of 0, 1, 2 so that we can estimate the effects of how many alternate allele copies an individual has.

The best way to do this is using factors.

```r
#Recode to factor
geno <- mutate(geno, geno.num = factor(genotype))

#Change factor levels
geno <- mutate(geno, geno.num = recode_factor(geno.num,
                                      "0/0" = 0,
                                      "0/1" = 1,
                                      "1/1" = 2))
#Recode to numeric
geno <- mutate(geno,
               geno.num = as.numeric(as.character(geno.num)))
```

Pivot long to wide format

This is all the manipulation we want to do to the internal data, so that last step is to convert the table back to wide format (e.g. undo `pivot_longer`). This time, we want rows to be samples so we use the snpID as names

Select our cleaned variables.

```
geno <- select(geno, snpID, FULLIDNO, geno.num)
```

Convert to wide format.

```
geno <- pivot_wider(geno,
                    names_from = "snpID", values_from = "geno.num")
```

And so we have

```
geno
```

```
## # A tibble: 103 x 271
##    FULLIDNO JHU_7.151253265 JHU_7.151254994 rs5017429 JHU_7.151257035
##    <chr>              <dbl>           <dbl>     <dbl>           <dbl>
##  1 84165-1~               1               0         0               0
##  2 84182-1~               0               0         2               0
##  3 84183-1~               0               1         0               0
##  4 84183-1~               0               1         0               0
##  5 84186-1~               0               1         0               0
##  6 84186-1~               0               0         1               0
##  7 84218-1~               0               0         1               0
##  8 84218-1~               0               0         2               0
##  9 84218-1~               0               0         1               0
## 10 84222-1~               0               0         1               0
## # ... with 93 more rows, and 266 more variables: JHU_7.151257523 <dbl>,
## #   rs28763998 <dbl>, JHU_7.151258416 <dbl>, rs11982878 <dbl>,
## #   rs17173197 <dbl>, rs77133357 <dbl>, rs7791529 <dbl>, rs4726052 <dbl>,
## #   rs75705628 <dbl>, rs1029946 <dbl>, JHU_7.151277901 <dbl>, rs6953318 <dbl>,
## #   JHU_7.151281262 <dbl>, rs2538039 <dbl>, rs12164128 <dbl>, rs4726058 <dbl>,
## #   rs56189422 <dbl>, rs138640368 <dbl>, rs3789809 <dbl>, rs2538036 <dbl>,
## #   JHU_7.151291496 <dbl>, JHU_7.151300192 <dbl>, JHU_7.151301751 <dbl>,
## #   JHU_7.151304363 <dbl>, rs6464150 <dbl>, JHU_7.151306394 <dbl>,
## #   rs2374229 <dbl>, rs2538034 <dbl>, rs4726070 <dbl>, rs6964957 <dbl>,
## #   rs4726075 <dbl>, rs3109944 <dbl>, rs17173208 <dbl>, rs7796163 <dbl>,
## #   JHU_7.151347611 <dbl>, JHU_7.151347769 <dbl>, JHU_7.151348046 <dbl>,
## #   JHU_7.151349226 <dbl>, rs10251264 <dbl>, JHU_7.151349448 <dbl>,
## #   rs4726081 <dbl>, rs868766 <dbl>, rs113108636 <dbl>, rs2727529 <dbl>,
## #   rs953221 <dbl>, rs1105842 <dbl>, JHU_7.151364416 <dbl>,
## #   JHU_7.151367983 <dbl>, rs2727535 <dbl>, rs116226629 <dbl>, rs2727537 <dbl>,
## #   rs6978142 <dbl>, rs6951177 <dbl>, rs2536090 <dbl>, rs56020328 <dbl>,
## #   JHU_7.151382859 <dbl>, rs4726084 <dbl>, rs1860746 <dbl>,
## #   JHU_7.151386715 <dbl>, rs10277859 <dbl>, rs2536085 <dbl>, rs62478181 <dbl>,
## #   JHU_7.151391303 <dbl>, JHU_7.151391466 <dbl>, JHU_7.151391533 <dbl>,
## #   rs73484108 <dbl>, rs143259477 <dbl>, rs76061449 <dbl>, rs56765232 <dbl>,
## #   rs7780804 <dbl>, JHU_7.151395753 <dbl>, JHU_7.151396309 <dbl>,
## #   rs6947064 <dbl>, rs6967507 <dbl>, rs116774231 <dbl>, JHU_7.151399789 <dbl>,
## #   JHU_7.151402498 <dbl>, rs73158180 <dbl>, JHU_7.151402935 <dbl>,
## #   rs60815793 <dbl>, rs114166988 <dbl>, rs6464164 <dbl>, rs10480299 <dbl>,
## #   rs10480300 <dbl>, JHU_7.151406825 <dbl>, JHU_7.151407104 <dbl>,
## #   JHU_7.151407372 <dbl>, rs6945264 <dbl>, rs4725424 <dbl>, rs79685042 <dbl>,
## #   rs11980120 <dbl>, rs10224002 <dbl>, rs10952316 <dbl>,
## #   JHU_7.151416887 <dbl>, JHU_7.151417087 <dbl>, rs62478182 <dbl>,
## #   rs77903099 <dbl>, rs78308051 <dbl>, JHU_7.151420857 <dbl>,
## #   rs59938864 <dbl>, ...
```

Piping with %>%

Recall the basic dplyr verb syntax:

- input data frame in the first argument
- other arguments can refer to variables as if they were local objects
- output is another data frame

Our data cleaning code continuously overwrites the `geno` object every time we call a verb. Instead, we can chain commands together using the pipe `%>%` operator. This works nicely to condense code and to improve readability.

`f(x) %>% g(y)` is the same as `g(f(x),y)`

`select(geno, snpID)` is the same as `genp %>% select(snpID)`

Let's return to our `snp` and perform all the above wrangling in 1 piped function. Note how I've added comments within the function to aid the reader.

```
geno <- snp %>%
  #Select genotype data
  select(snpID, `84165-1-06`:`94295-1-02`) %>%
  #Convert to long format
  pivot_longer(-snpID, names_to="FULLIDNO", values_to="genotype") %>%
  #Convert genotype 0/0 to 0 format
  mutate(geno.num = recode_factor(factor(genotype),
                                   "0/0" = 0,
                                   "0/1" = 1,
                                   "1/1" = 2)) %>%
  mutate(geno.num = as.numeric(as.character(geno.num))) %>%
  #Convert back to wide format
  select(snpID, FULLIDNO, geno.num) %>%
  pivot_wider(names_from = "snpID", values_from = "geno.num")
```

Joining data frames

We can now combine `meta` and our reformatted `geno` data by the FULLIDNO column that they both have.

There are a suite of tidyverse functions for combining two tables, all starting with `join_`. They differ in which rows that keep from each of the two data frames being combined. For example, `left_join` keep all the rows in the left (first) data frame given and removes any rows from the right (second) data frame that do not have matches in the left one.

A great reference on join functions can be found at https://stat545.com/join-cheatsheet.html

Here, we will use `inner_join` which only keeps rows with data in BOTH data frames. For us, this means we only keep data with both `geno` genotype data and RNA-seq since `meta` came from `dat.norm.voom`.

We also don't need all the columns from `meta` so we select what we want first.

```
meta.geno <- meta %>%
  select(libID, sampID, FULLIDNO, condition, M0_KCVAGE) %>%
  inner_join(geno, by="FULLIDNO")
```

If you wanted to use different variables, checkout all those available.

```
colnames(meta)
```

```
##  [1] "group"                         "lib.size"
##  [3] "norm.factors"                  "libID"
##  [5] "sampID"                        "condition"
##  [7] "dataset"                       "FULLIDNO"
##  [9] "KIFW_VISIT_TYPE"               "KIFW_COLLECTION_DATE"
## [11] "Sample_Group"                  "CONCORDANT_STATUS"
## [13] "KQFT_TST_QFT_CLASSIFICATION_C" "KQFT_CERTAINTY"
## [15] "MO_KCVSEX"                     "MO_KCVAGE"
## [17] "MO_KCVDATE"                    "avgBMI"
## [19] "HIVSTAT_CURRENT"               "KCB_BCGSCAR"
## [21] "RISK_SCORE"                    "KCB_RELATION"
## [23] "FULLMID"                       "FULLFID"
## [25] "family.3"                      "family.2"
## [27] "family.1"                      "sample.weights"
```

Filtering rows

Our final piece of data to add is the PRKAG2 expression from `dat.norm.voom$E`. First, we extract these data and force it into a data frame (since it is a matrix).

```
counts <- as.data.frame(dat.norm.voom$E)
```

We then filter the PRKAG2 row.

```
prkag2 <- counts %>%
  #Move rownames to a variable/column
  rownames_to_column("symbol") %>%
  #Keep only PRKAG2 row
  filter(symbol == "PRKAG2")
```

More pivoting and joining

Similar to steps with `geno`, we convert our PRKAG2 data to long format and join it to `meta.geno`

```
meta.geno.prkag2 <- prkag2 %>%
  #Convert to long format
  pivot_longer(-symbol, names_to="libID", values_to="PRKAG2") %>%
  #Remove HGNC symbol column
  ## If you have > 1 gene, you would not do this
  select(-symbol) %>%
  #Merge with meta.geno
  inner_join(meta.geno, by="libID")
```

Linear model

Finally, we have our formatted data! Here are all our variables.

```
colnames(meta.geno.prkag2)
```

```
##    [1] "libID"           "PRKAG2"          "sampID"          "FULLIDNO"
##    [5] "condition"       "MO_KCVAGE"       "JHU_7.151253265" "JHU_7.151254994"
##    [9] "rs5017429"       "JHU_7.151257035" "JHU_7.151257523" "rs28763998"
##   [13] "JHU_7.151258416" "rs11982878"      "rs17173197"      "rs77133357"
##   [17] "rs7791529"       "rs4726052"       "rs75705628"      "rs1029946"
##  [21] "JHU_7.151277901" "rs6953318"       "JHU_7.151281262" "rs2538039"
##  [25] "rs12164128"      "rs4726058"       "rs56189422"      "rs138640368"
##  [29] "rs3789809"       "rs2538036"       "JHU_7.151291496" "JHU_7.151300192"
##  [33] "JHU_7.151301751" "JHU_7.151304363" "rs6464150"       "JHU_7.151306394"
##  [37] "rs2374229"       "rs2538034"       "rs4726070"       "rs6964957"
##  [41] "rs4726075"       "rs3109944"       "rs17173208"      "rs7796163"
##  [45] "JHU_7.151347611" "JHU_7.151347769" "JHU_7.151348046" "JHU_7.151349226"
##  [49] "rs10251264"      "JHU_7.151349448" "rs4726081"       "rs868766"
##  [53] "rs113108636"     "rs2727529"       "rs953221"        "rs1105842"
##  [57] "JHU_7.151364416" "JHU_7.151367983" "rs2727535"       "rs116226629"
##  [61] "rs2727537"       "rs6978142"       "rs6951177"       "rs2536090"
##  [65] "rs56020328"      "JHU_7.151382859" "rs4726084"       "rs1860746"
##  [69] "JHU_7.151386715" "rs10277859"      "rs2536085"       "rs62478181"
##  [73] "JHU_7.151391303" "JHU_7.151391466" "JHU_7.151391533" "rs73484108"
##  [77] "rs143259477"     "rs76061449"      "rs56765232"      "rs7780804"
##  [81] "JHU_7.151395753" "JHU_7.151396309" "rs6947064"       "rs6967507"
##  [85] "rs116774231"     "JHU_7.151399789" "JHU_7.151402498" "rs73158180"
##  [89] "JHU_7.151402935" "rs60815793"      "rs114166988"     "rs6464164"
##  [93] "rs10480299"      "rs10480300"      "JHU_7.151406825" "JHU_7.151407104"
##  [97] "JHU_7.151407372" "rs6945264"       "rs4725424"       "rs79685042"
## [101] "rs11980120"      "rs10224002"      "rs10952316"      "JHU_7.151416887"
## [105] "JHU_7.151417087" "rs62478182"      "rs77903099"      "rs78308051"
## [109] "JHU_7.151420857" "rs59938864"      "rs4726086"       "JHU_7.151422899"
## [113] "rs2536069"       "kgp8853492"      "JHU_7.151423487" "rs9640176"
## [117] "rs11760870"      "JHU_7.151428620" "rs4726088"       "rs116677629"
## [121] "rs1108722"       "rs1104897"       "JHU_7.151434319" "JHU_7.151434443"
## [125] "JHU_7.151436055" "rs76563225"      "rs28416498"      "rs10278273"
## [129] "rs1001116"       "rs11771330"      "rs1860740"       "rs6978479"
## [133] "JHU_7.151446136" "rs11773668"      "rs61612573"      "rs62478188"
## [137] "rs115389468"     "rs79215320"      "rs7801616"       "rs12537154"
## [141] "JHU_7.151454520" "rs6965926"       "rs12668627"      "JHU_7.151456632"
## [145] "JHU_7.151456735" "rs1104840"       "rs11520869"      "rs11982124"
## [149] "rs7800069"       "JHU_7.151464922" "rs3934597"       "JHU_7.151466470"
## [153] "JHU_7.151468071" "rs4726091"       "JHU_7.151473009" "rs7809589"
## [157] "JHU_7.151474450" "rs3935852"       "JHU_7.151474597" "rs73728288"
## [161] "rs55964814"      "JHU_7.151475760" "rs9632641"       "rs78667038"
## [165] "rs80039831"      "rs78291918"      "JHU_7.151479513" "JHU_7.151480349"
## [169] "JHU_7.151480717" "rs62478216"      "rs73728291"      "rs6953882"
## [173] "rs11975504"      "rs6959849"       "JHU_7.151484100" "rs4425665"
## [177] "rs56859244"      "rs79637471"      "rs11770585"      "JHU_7.151489803"
## [181] "rs34735358"      "rs79918575"      "rs34659848"      "JHU_7.151491553"
## [185] "rs34698098"      "rs56037571"      "rs12112702"      "rs4726098"
## [189] "rs4128396"       "rs76923703"      "rs9648724"       "rs4442045"
## [193] "rs13224758"      "rs56729506"      "rs7802319"       "rs12539356"
## [197] "rs377649951"     "rs74535090"      "rs56129741"      "rs76727740"
## [201] "JHU_7.151513101" "rs4726101"       "rs1881633"       "rs77961133"
## [205] "rs114008100"     "rs1881632"       "JHU_7.151517481" "JHU_7.151517732"
## [209] "rs13309585"      "rs66972858"      "rs11770376"      "rs75379928"
## [213] "rs73487739"      "JHU_7.151520306" "rs13225852"      "rs11773373"
## [217] "rs7800364"       "rs116508619"     "JHU_7.151528914" "JHU_7.151529175"
## [221] "rs13233608"      "JHU_7.151529363" "JHU_7.151529449" "rs77954465"
## [225] "JHU_7.151529595" "JHU_7.151529647" "JHU_7.151529991" "rs11771216"
## [229] "rs1881625"       "rs11764602"      "rs74782537"      "rs75697045"
## [233] "JHU_7.151534273" "rs114497371"     "rs6950343"       "rs61462038"
## [237] "rs9648727"        "rs114016536"      "JHU_7.151539874" "rs10261999"
## [241] "rs62480473"       "rs7795096"        "rs57194047"      "rs1881639"
## [245] "rs3813852"        "rs11971588"       "JHU_7.151551975" "rs13245627"
## [249] "rs6961830"        "rs11971243"       "rs6955784"       "rs7789674"
## [253] "rs9640300"        "rs12703164"       "rs4726104"       "JHU_7.151562456"
## [257] "rs59404899"       "JHU_7.151563452" "JHU_7.151563721" "JHU_7.151563769"
## [261] "JHU_7.151563945" "rs73160061"       "JHU_7.151564516" "JHU_7.151564713"
## [265] "rs56118280"       "rs78168617"       "JHU_7.151566710" "rs12669153"
## [269] "rs13240743"       "JHU_7.151566976" "rs13235096"       "rs4725435"
## [273] "rs75188986"       "rs35503973"       "JHU_7.151572648" "rs116605521"
```

Fit model

We fit a linear model with `lm( )` where the formula is given as `y ~ x`. Here we want to know if the effect of genotype at SNP JHU_7.151349226 differs in MEDIA vs TB samples (condition). We also include age as a co-variate as an example of how to correct with variables such as this.

```
model <- lm(PRKAG2 ~ condition*JHU_7.151349226 + M0_KCVAGE,
   data=meta.geno.prkag2)
```

And then we estimate P-values with `summary( )`

```
summary(model)
```

```
##
## Call:
## lm(formula = PRKAG2 ~ condition * JHU_7.151349226 + M0_KCVAGE,
##     data = meta.geno.prkag2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.68343 -0.21794 -0.04634  0.26267  0.98214
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 5.5285560  0.0621495  88.956   <2e-16 ***
## conditionTB                 0.7413534  0.0561486  13.203   <2e-16 ***
## JHU_7.151349226             0.2285340  0.1128986   2.024   0.0444 *
## M0_KCVAGE                   0.0002543  0.0032872   0.077   0.9384
## conditionTB:JHU_7.151349226 -0.3635256  0.1596372  -2.277   0.0239 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3801 on 189 degrees of freedom
## Multiple R-squared:  0.4813, Adjusted R-squared:  0.4703
## F-statistic: 43.85 on 4 and 189 DF,  p-value: < 2.2e-16
```

Or you can get a clean data frame summary with `tidy( )` in the `broom` package.

```
install.packages("broom")
```

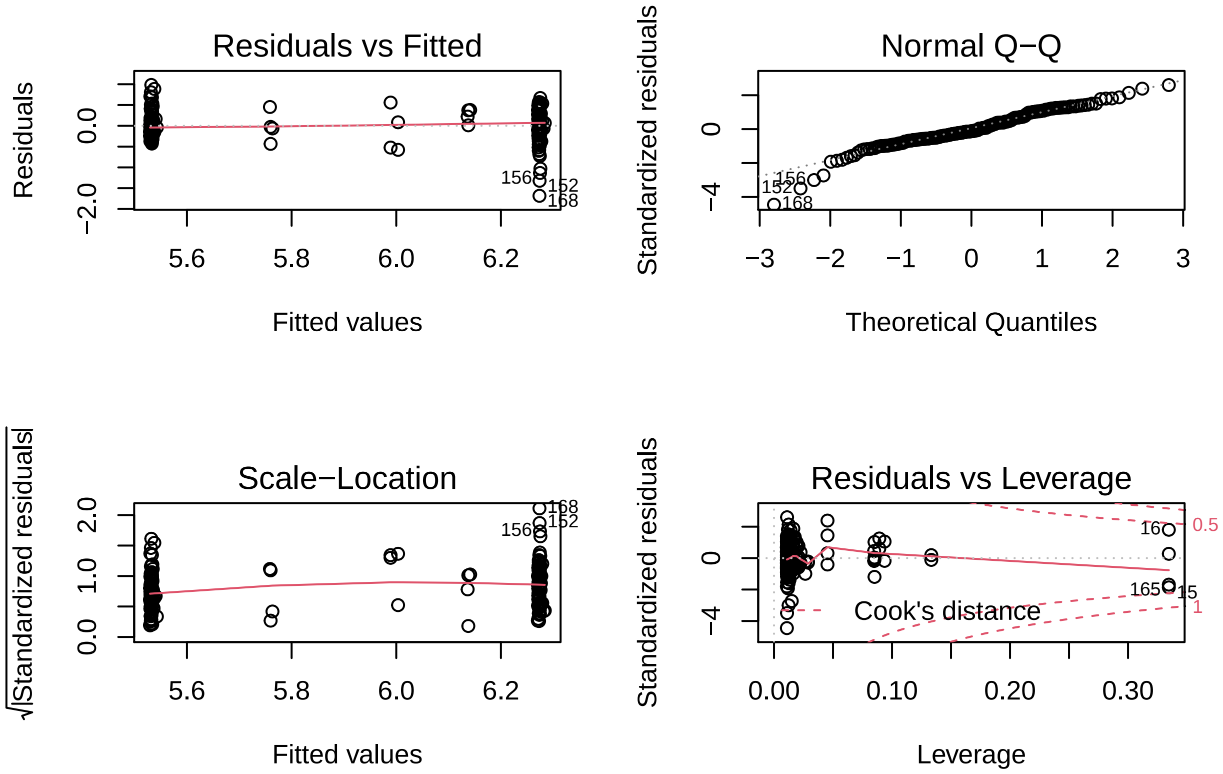
```
library(broom)
tidy(model)
```

```
## # A tibble: 5 x 5
##   term                        estimate std.error statistic   p.value
##   <chr>                          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)                  5.53      0.0621    89.0    3.40e-156
## 2 conditionTB                  0.741     0.0561    13.2    1.25e- 28
## 3 JHU_7.151349226              0.229     0.113      2.02   4.44e-  2
## 4 MO_KCVAGE                    0.000254  0.00329    0.0773 9.38e-  1
## 5 conditionTB:JHU_7.151349226 -0.364     0.160     -2.28   2.39e-  2
```

Assess model

A quick way to assess your model, is by plotting the residuals. A nice explanation of the Q-Q plot can be found at https://stats.stackexchange.com/questions/101274/how-to-interpret-a-qq-plot and some more tests are shown at https://www.statmethods.net/stats/rdiagnostics.html

```
#Set plot frame to be 2x2
par(mfrow=c(2,2))
#Plot model fit
plot(model)
```



Graphics with ggplot2

ggplot2 is the tidyverse's main plotting package. Full documentation is available at docs.ggplot2.org

Why ggplot?

ggplot2 is an implementation of *Grammar of Graphics* (Wilkinson 1999) for R

Benefits:

- handsome default settings
- snap-together building block approach
- automatic legends, colors, facets
- statistical overlays: regressions lines and smoothers (with confidence intervals)

Drawbacks:

- it can be hard to get it to look *exactly* the way you want
- requires having the input data in a certain format

ggplot building blocks

- data: 2D table (`data.frame`) of *variables*
- *aesthetics*: map variables to visual attributes (e.g., position)
- *geoms*: graphical representation of data (points, lines, etc.)
- *stats*: statistical transformations to get from data to points in the plot(binning, summarizing, smoothing)
- *scales*: control *how* to map a variable to an aesthetic
- *facets*: juxtapose mini-plots of data subsets, split by variable(s)
- *guides*: axes, legend, etc. reflect the variables and their values

Idea: independently specify and combine the blocks to create the plotyou want.

There are at least three things we have to specify to create a plot:

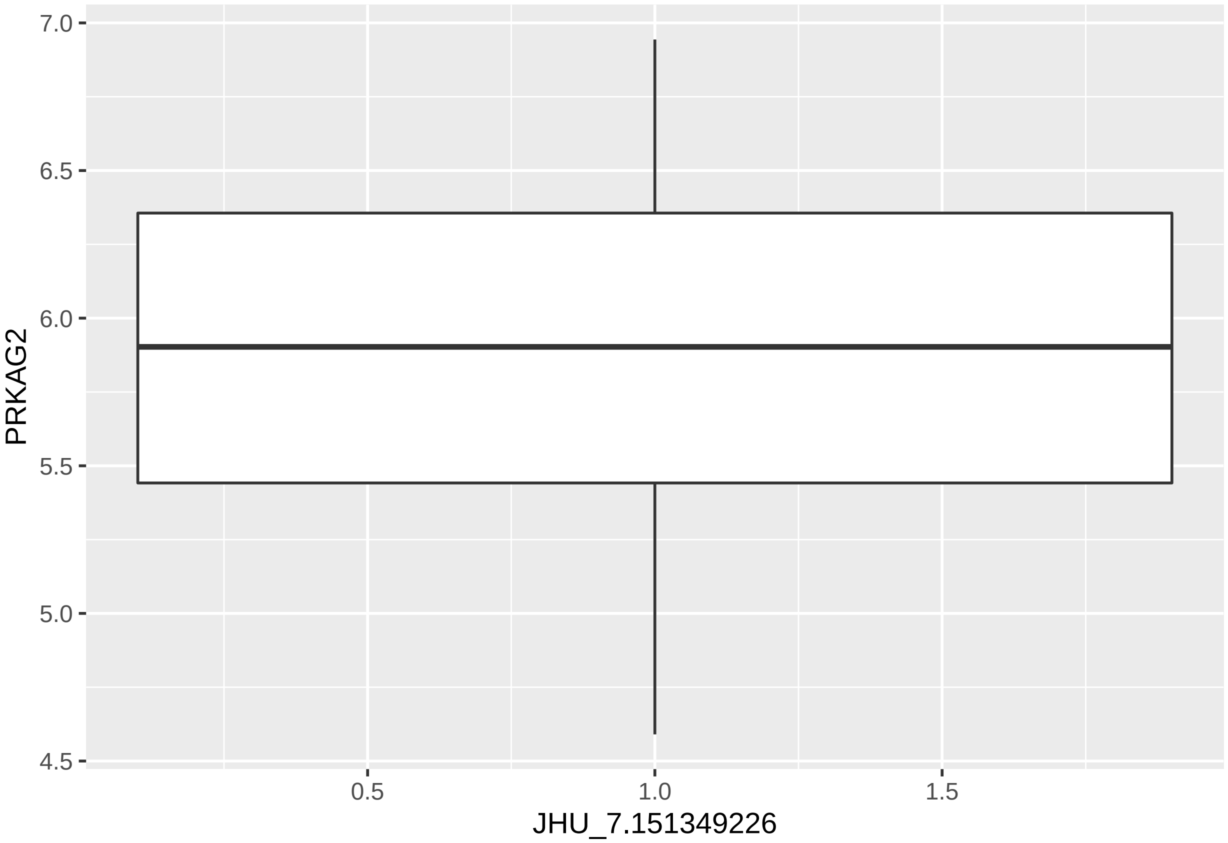
1. data
2. aesthetic mappings from data variables to visual properties
3. a layer describing how to draw those properties

Boxplot

We will start with a boxplot of our data with the SNP genotype on X and gene expression on Y.

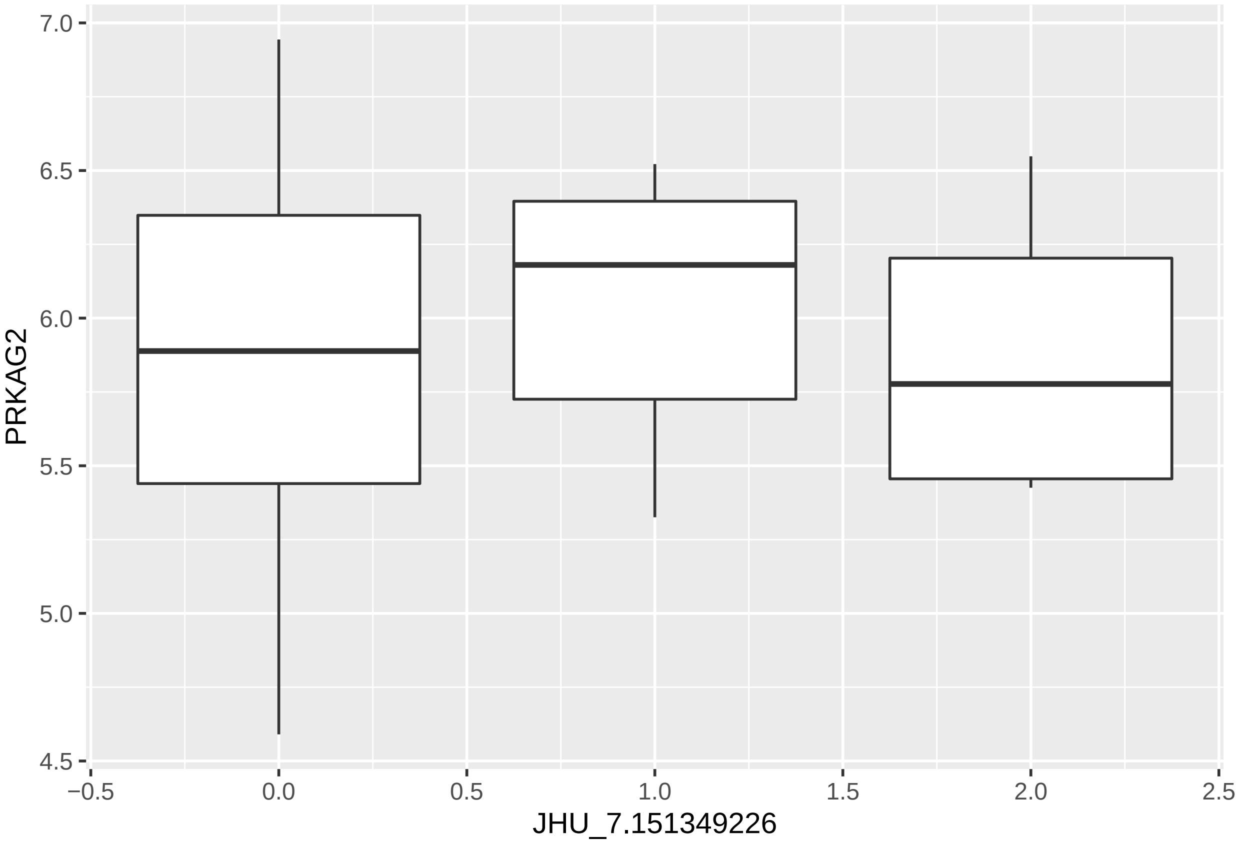
```r
ggplot(meta.geno.prkag2, aes(x=JHU_7.151349226, y=PRKAG2)) +
  geom_boxplot()
```

```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```



But as we see above, boxplots don't work with numeric x-variables. The warning gives us a hint of what to do with the group option. Since we want groups to apply to just the boxplots (and not any other geoms we may add later), we put the group aesthetic in geom_boxplot.

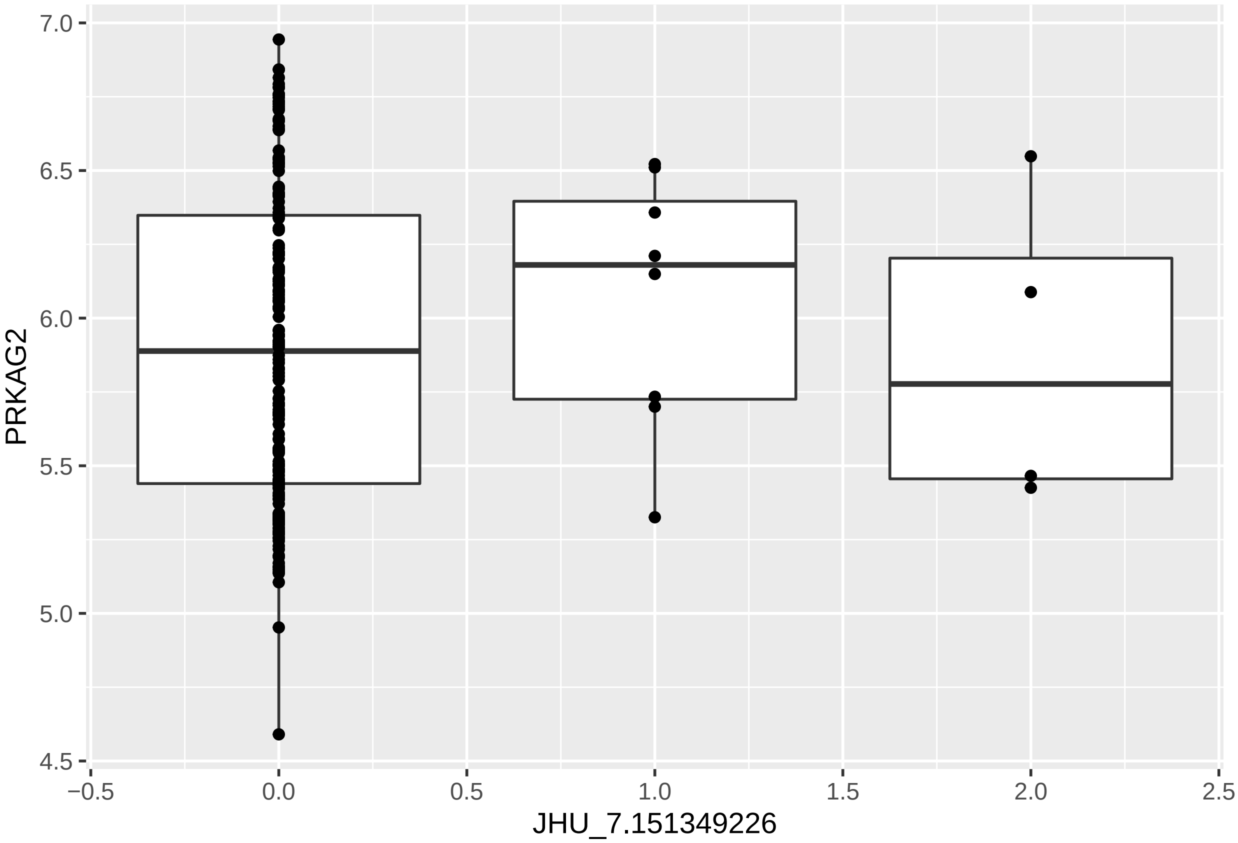
```
ggplot(meta.geno.prkag2, aes(x=JHU_7.151349226, y=PRKAG2)) +
  geom_boxplot(aes(group=JHU_7.151349226))
```



Multiple geoms

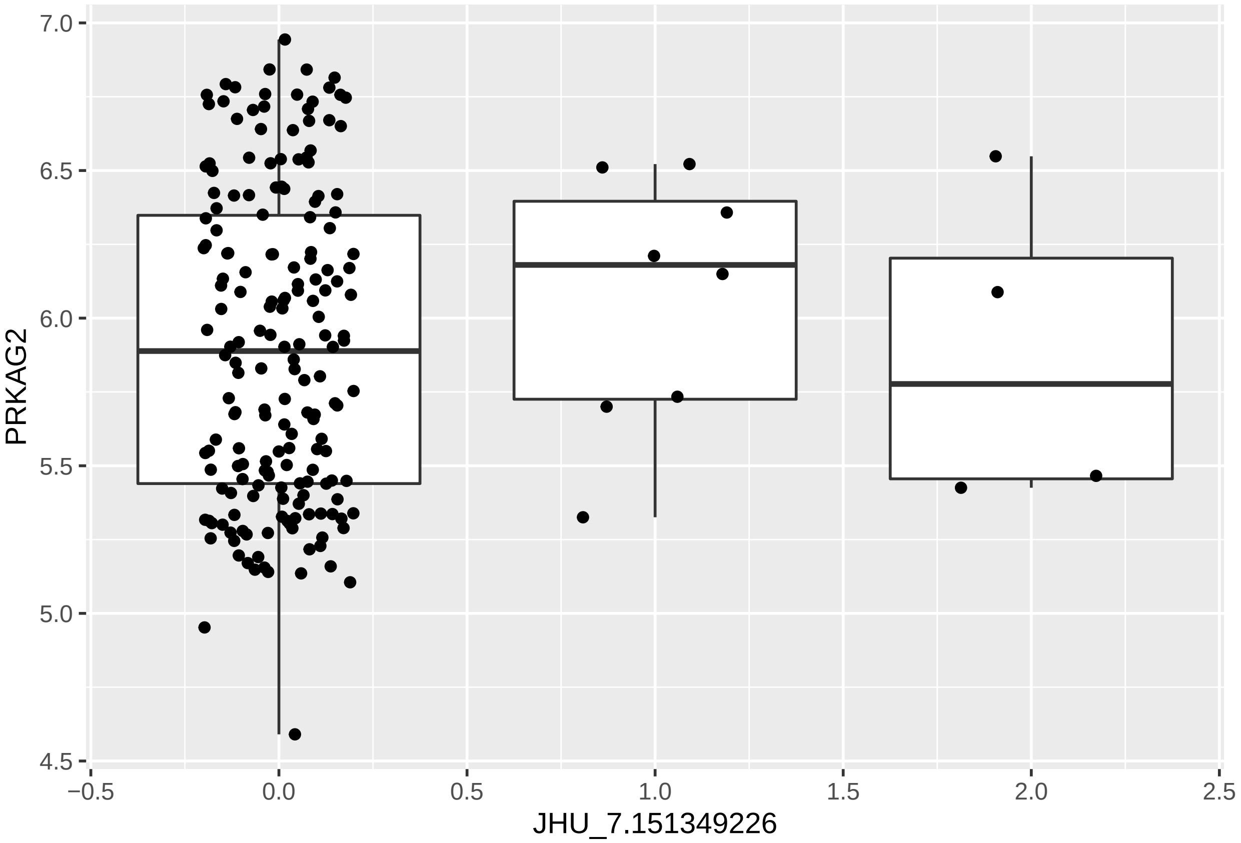
The building block nature of ggplot allows us to add multiple geoms on top of each other. So we can include individual dots for each sample with geom_point to see the sample sizes within each boxplot.

```
ggplot(meta.geno.prkag2, aes(x=JHU_7.151349226, y=PRKAG2)) +
  geom_boxplot(aes(group=JHU_7.151349226)) +
  geom_point()
```



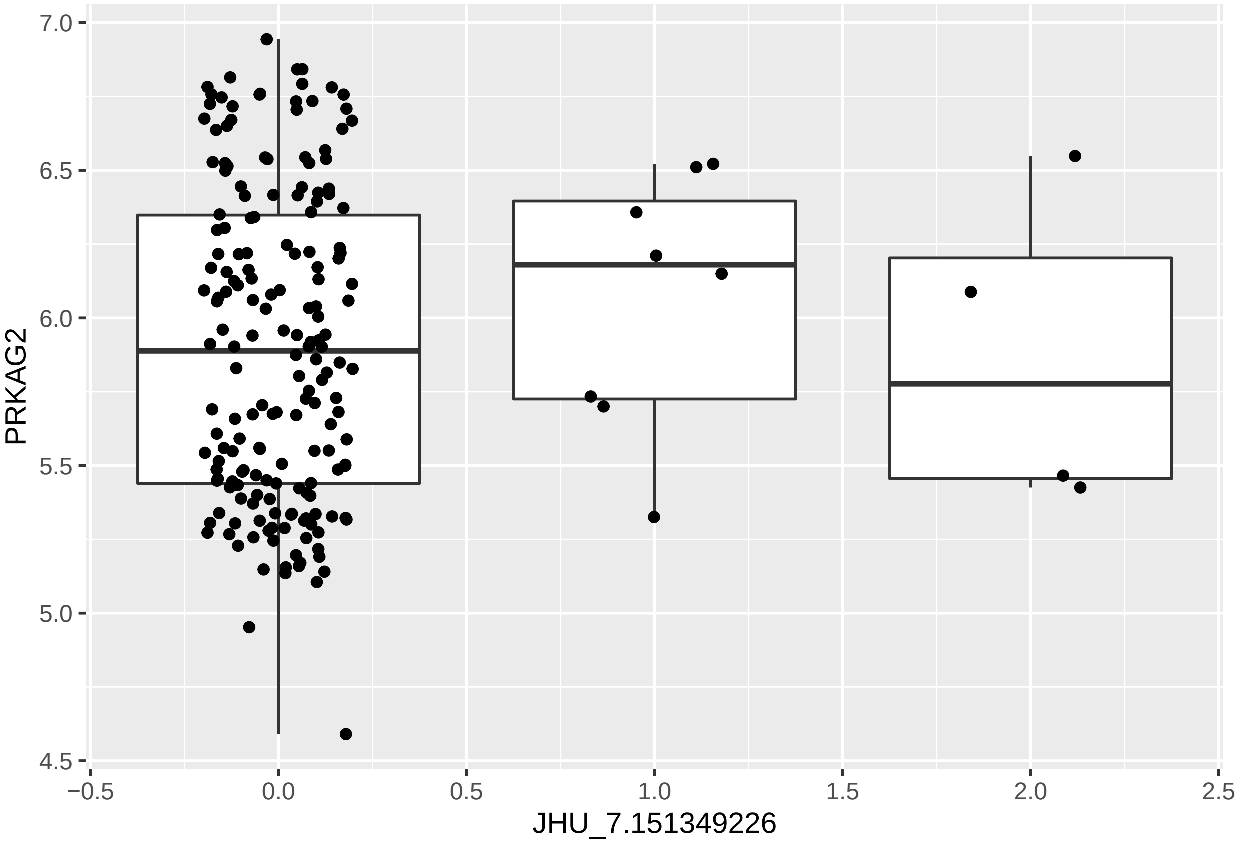
But a lot of points overlap so we switch to geom_jitter to spread them out. Setting the jitter height to 0 ensures that the perceived y values are correct for these data.

```
ggplot(meta.geno.prkag2, aes(x=JHU_7.151349226, y=PRKAG2)) +
  geom_boxplot(aes(group=JHU_7.151349226)) +
  geom_jitter(width = 0.2, height=0)
```



And while this is not an issue in this plot (since there are no outliers), you should also remove the outliers plotted by geom_boxplot or else there will be duplicate points with geom_jitter.

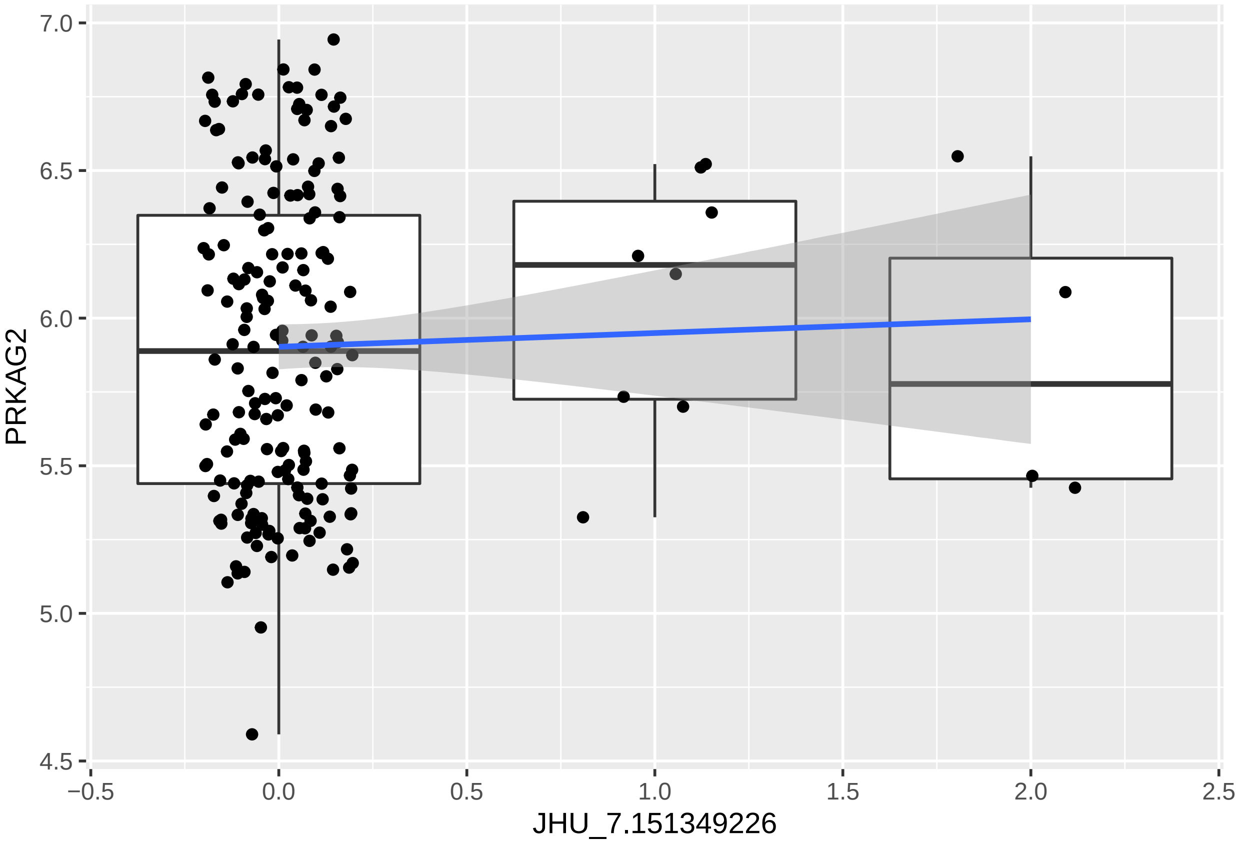
```
ggplot(meta.geno.prkag2, aes(x=JHU_7.151349226, y=PRKAG2)) +
  geom_boxplot(aes(group=JHU_7.151349226),
               outlier.shape=NULL) +
  geom_jitter(width = 0.2, height=0)
```



Fit lines

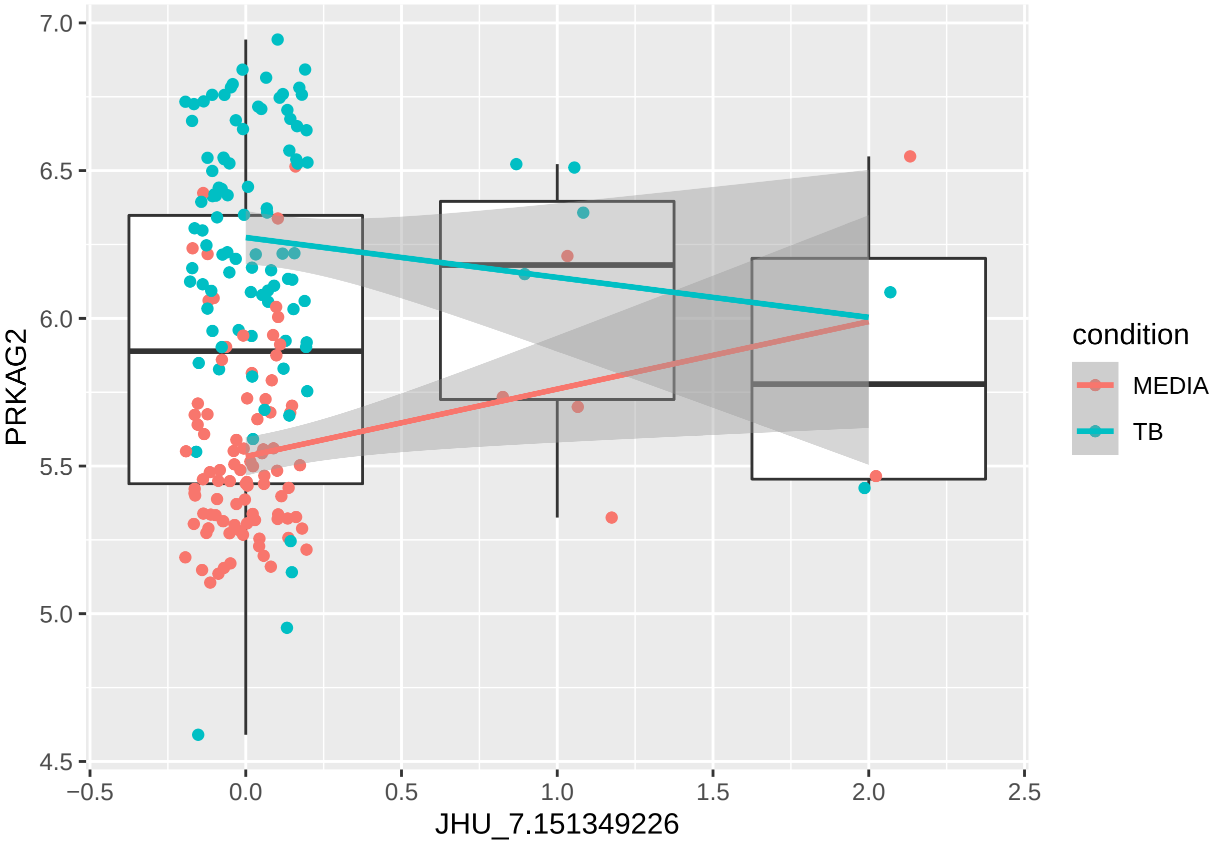
We fit a linear fit to our data so let's add it to the plot. This is done with geom_smooth.

```
ggplot(meta.geno.prkag2, aes(x=JHU_7.151349226, y=PRKAG2)) +
  geom_boxplot(aes(group=JHU_7.151349226),
               outlier.shape=NULL) +
  geom_jitter(width = 0.2, height=0) +
  geom_smooth(method="lm")
```



But this isn't exactly what we modeled, right? We grouped the samples by SNP genotype and condition. We can add this in by coloring the points and the fit line.

```
ggplot(meta.geno.prkag2, aes(x=JHU_7.151349226, y=PRKAG2)) +
  geom_boxplot(aes(group=JHU_7.151349226),
               outlier.shape=NULL) +
  geom_jitter(aes(color=condition), width = 0.2, height=0) +
  geom_smooth(aes(color=condition), method="lm")
```

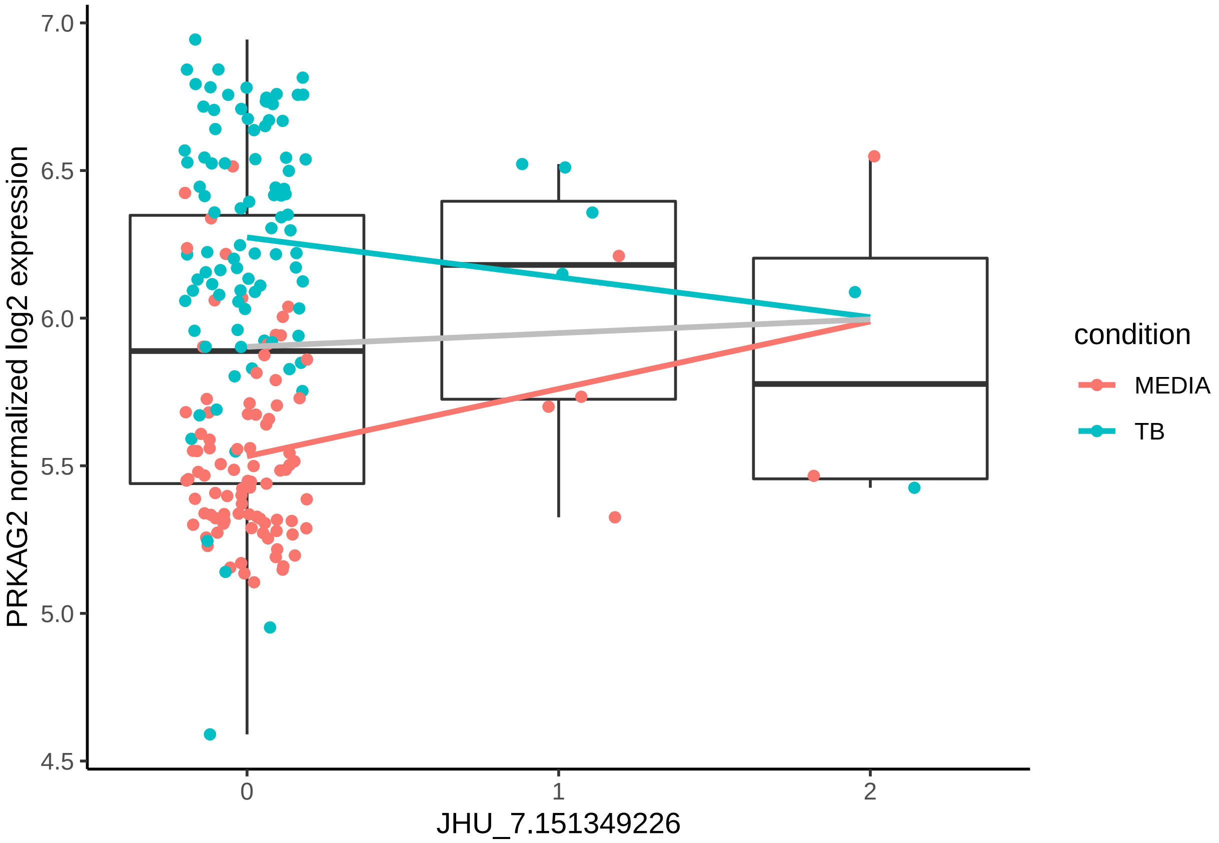


Now we clearly see why the interaction term is significant as the slopes of our two lines differ.

Further beautification

Below are just a couple other tricks to perfect this plot.

```
ggplot(meta.geno.prkag2,
       #Define x and y variables
         aes(x=JHU_7.151349226, y=PRKAG2)) +
  #Create boxplots
  ## Use group to force gglot to treat genotype like character
  ## instead of numeric
  ## Set the outlier shape to NULL so that you don't get duplicate
  ## dots in the next layer.
  geom_boxplot(aes(group=JHU_7.151349226), outlier.shape=NULL) +
  #Add points for each sample. Color by MEDIA/TB and "jitter" left
  # and right (width) to avoid overlap
  geom_jitter(aes(color=condition), width = 0.2, height=0) +
  # Add a linear fit for each condition
  geom_smooth(aes(color=condition), method="lm", se=FALSE) +
  #Add a linear fit for genotype overall
  geom_smooth(color="grey", method="lm", se=FALSE) +
  #Format axis labels
  labs(y="PRKAG2 normalized log2 expression") +
  scale_x_continuous(breaks=c(0,1,2)) +
  #Change theme to classic to remove grey background and other
  #default aspects of ggplot
  theme_classic()
```



In the above, the grey line is the overall effect of SNP genotype, which is significant in the model (P = 0.0444). The other colored lines are the fit within all MEDIA or TB samples separately. In our model, `conditionTB:JHU_7.151349226` corresponds to this interaction term and shows that the slopes are significantly different (P = 0.0239).

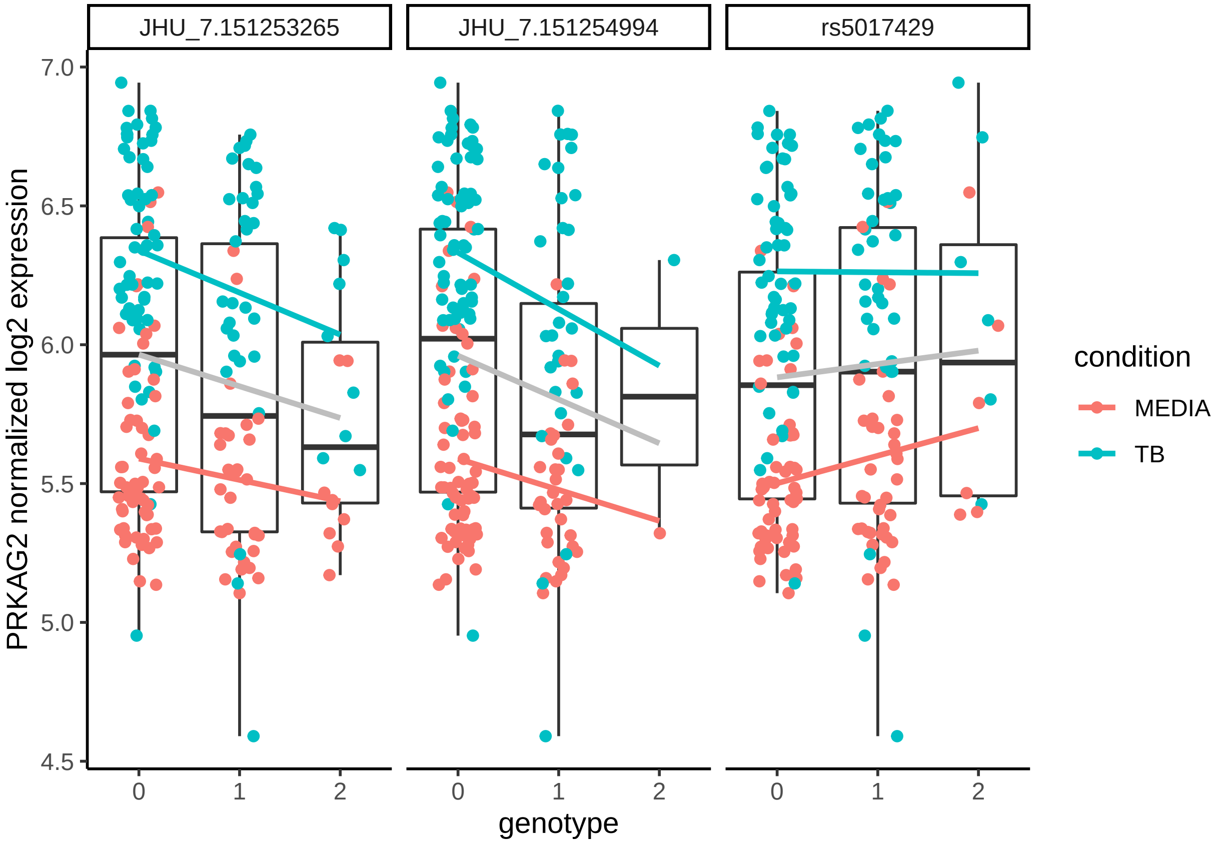
We don't specifically see the final significant term from our model (conditionTB P < 2e-16) but can estimate it by observing that overall, TB samples have higher expression than MEDIA samples.

Complex plots

Facets A quick way to make a lot of plots is with facets. The power of facets is harnessed by combining tidyverse pipes with ggplot. Here, we modify the `meta.geno.prkag2` data but don't save it as an object in the environment. We instead pipe the modified data directly into ggplot.

```
meta.geno.prkag2 %>%
  #Select metadata and first 3 SNPs
  select(libID:rs5017429) %>%
  #Pivot SNP data long into 1 column
  pivot_longer(JHU_7.151253265:rs5017429,
               names_to = "snpID", values_to = "genotype") %>%

ggplot(aes(x=genotype, y=PRKAG2)) +
  geom_boxplot(aes(group=genotype), outlier.shape=NULL) +
  geom_jitter(aes(color=condition), width = 0.2, height=0) +
  geom_smooth(aes(color=condition), method="lm", se=FALSE) +
  geom_smooth(color="grey", method="lm", se=FALSE) +
  labs(y="PRKAG2 normalized log2 expression") +
  scale_x_continuous(breaks=c(0,1,2)) +
  theme_classic() +
  #Add facet
  facet_wrap(~snpID)
```



Note that here, our X and Y scales are similar but facets allow you to vary scales with the `scales` parameter within `facet_wrap`.

Cautions with these data

Duplicates There are two types of sample duplicates in these data.

1. Paired MEDIA and TB samples
   - If you do an analysis with both conditions like above, your model needs to block by sample (RSID) to take into account the experimental design. This can be done with the package `lme4`. See this past workshop for examples https://github.com/hawn-lab/workshops_UW_Seattle/tree/master/2020.05.12_linear_models
2. There are multiple samples (RSID) for some individuals (FULLIDNO). If these exist in the subset of data you are interested in, your model needs to take this into account OR you should choose 1 RNA-seq sample per individual.

```
##      FULLIDNO     RSIDs
## [1,] "84186-1-04" "RS102338,RS102132"
## [2,] "84186-1-04" "RS102338,RS102132"
## [3,] "84222-1-17" "RS102057,RS102230"
## [4,] "84222-1-17" "RS102057,RS102230"
## [5,] "84222-1-20" "RS102051,RS102329"
## [6,] "84222-1-20" "RS102051,RS102329"
## [7,] "84222-1-23" "RS102049,RS102262"
##  [8,] "84222-1-23" "RS102049,RS102262"
##  [9,] "84224-1-14" "RS102287,RS102096"
## [10,] "84224-1-14" "RS102287,RS102096"
## [11,] "84275-1-04" "RS102054,RS102233"
## [12,] "84275-1-04" "RS102054,RS102233"
## [13,] "84441-1-04" "RS102115,RS102299"
## [14,] "84441-1-04" "RS102115,RS102299"
## [15,] "84495-1-05" "RS102539,RS102319"
## [16,] "84495-1-05" "RS102539,RS102319"
## [17,] "84523-1-02" "RS102320,RS102466"
## [18,] "84523-1-02" "RS102320,RS102466"
## [19,] "88159-1-04" "RS102073,RS102445"
## [20,] "88159-1-04" "RS102073,RS102445"
## [21,] "90118-1-03" "RS102083,RS102249"
## [22,] "90118-1-03" "RS102083,RS102249"
## [23,] "90480-1-02" "RS102537,RS102400"
## [24,] "90480-1-02" "RS102537,RS102400"
## [25,] "90561-1-05" "RS102500,RS102071"
## [26,] "90561-1-05" "RS102500,RS102071"
## [27,] "92527-1-08" "RS102531,RS102389"
## [28,] "92527-1-08" "RS102531,RS102389"
## [29,] "93597-1-05" "RS102062,RS102459"
## [30,] "93597-1-05" "RS102062,RS102459"
```

Sample size Be aware of your sample sizes. For example, in our analysis above, there are only 2 individuals (4 RNA-seq samples) with genotype 2. This is very small so how confident are we in the model fit?

Exercises: Part 2

Using `meta.geno.prkag2`

1. Fit linear models on MEDIA and TB sample subsets separately. How do the results compare to the interaction term model we ran above? (*Hint* tidyverse pipes can be used to pipe modified data directly into ggplot)

Using your own gene of interest

2. Recapitulate these analyses with another gene.
3. Challenge: How would you run the same linear model on all SNPs in a gene?

Additional resources

Groups

- Rladies Seattle Not just for ladies! A pro-actively inclusive R community with both in-person and online workshops, hangouts, etc.
- R code club Dr. Pat Schloss opened his lab's coding club to remote participation.
- Seattle useR Group

Online

- R cheatsheets also available in RStudio under Help > Cheatsheets
- The Carpentries

- Introduction to dplyr
- dplyr tutorial
- dplyr video tutorial
- More functions in dplyr and tidyr
- ggplot tutorial 1
- ggplot tutorial 2
- ggplot tutorial 3

R session

```r
sessionInfo()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.5
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] broom_0.5.6     limma_3.44.3    forcats_0.5.0   stringr_1.4.0
##  [5] dplyr_1.0.0     purrr_0.3.4     readr_1.3.1     tidyr_1.1.0
##  [9] tibble_3.0.1    ggplot2_3.3.2   tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.0 xfun_0.15        splines_4.0.0    haven_2.3.1
##  [5] lattice_0.20-41  colorspace_1.4-1 vctrs_0.3.1      generics_0.0.2
##  [9] htmltools_0.5.0  mgcv_1.8-31      yaml_2.2.1       utf8_1.1.4
## [13] blob_1.2.1       rlang_0.4.6      pillar_1.4.4     glue_1.4.1
## [17] withr_2.2.0      DBI_1.1.0        dbplyr_1.4.4     modelr_0.1.8
## [21] readxl_1.3.1     lifecycle_0.2.0  munsell_0.5.0    gtable_0.3.0
## [25] cellranger_1.1.0 rvest_0.3.5      evaluate_0.14    labeling_0.3
## [29] knitr_1.29       fansi_0.4.1      Rcpp_1.0.4.6     scales_1.1.1
## [33] backports_1.1.8  jsonlite_1.7.0   farver_2.0.3     fs_1.4.1
## [37] hms_0.5.3        digest_0.6.25    stringi_1.4.6    grid_4.0.0
## [41] cli_2.0.2        tools_4.0.0      magrittr_1.5     crayon_1.3.4
## [45] pkgconfig_2.0.3  Matrix_1.2-18    ellipsis_0.3.1   xml2_1.3.2
## [49] reprex_0.3.0     lubridate_1.7.9  assertthat_0.2.1 rmarkdown_2.3
## [53] httr_1.4.1       rstudioapi_0.11  R6_2.4.1         nlme_3.1-148
## [57] compiler_4.0.0
```