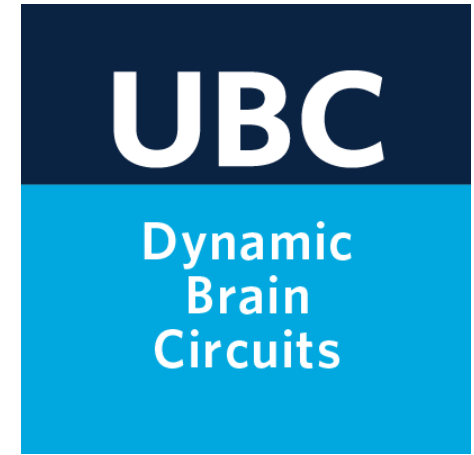
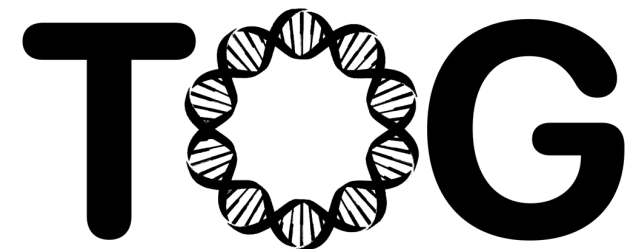


BCCHRI-DBC- ICORD 2022 Intro to R Workshop.

Keegan Flanagan,
Jeffrey LeDue, Megan Pawluk, William
Casazza , Nikita Telkar.



icord



BCCHR Trainee Omics Group

Why are we doing this workshop?

- Biomedical Researchers and Graduate Students come from a wide variety of training backgrounds.
- Not everyone has had the opportunity to spend time on learning how to do statistics and visualizations using coding languages such as Python and R.
- Course offerings at UBC (like Stats 540, 545) presume a working knowledge of R or have a very steep learning curve.
- Completing analyses with R allows for the creation of computationally reproducible analyses.
- R code can be published along with manuscripts.

What will we cover in this workshop?

- Getting oriented with RStudio.
- The basics of coding in R.
- The basics of working with Dataframes in R.
- Introduction to plotting in R using ggplot2.
- Brief introduction to statistics in R.

Learning goals:

- Understand the difference between R and RStudio.
- Be familiar with the RStudio layout.
- Understand the basics of workflow and how to create R-projects.
- Be able to handle basic coding tasks in R (mathematics, using and defining functions, working with dataframes)
- Understand how to load up new packages and how to get help information for packages.
- Understand how to make visually appealing plots with R and ggplot2.
- Learn how to complete a simple statistical test in R.

What is R?

- R is both a coding language and a multiplatform software environment for statistical computing and graphics.
- R is an open-source software environment, which means that anyone can work with it or contribute to it.
- R is an extremely popular tool for biological statistical analysis.

Why use R?

- Pros:
 - It is Free.
 - It has powerful graphical capabilities. With the right know-how, you can create publication ready graphs and visualizations.
 - It has an incredible variety of packages, especially statistical packages.
- Cons:
 - May be slower for certain tasks than alternatives like Python and MATLAB.
 - Unique language (syntactically very different than other languages).
 - Too many packages (redundant or poor-quality packages exist).

My personal experience: R is a great language for statistical analysis, but it is not the best when it comes to creating programs/pipelines.

R vs RStudio.

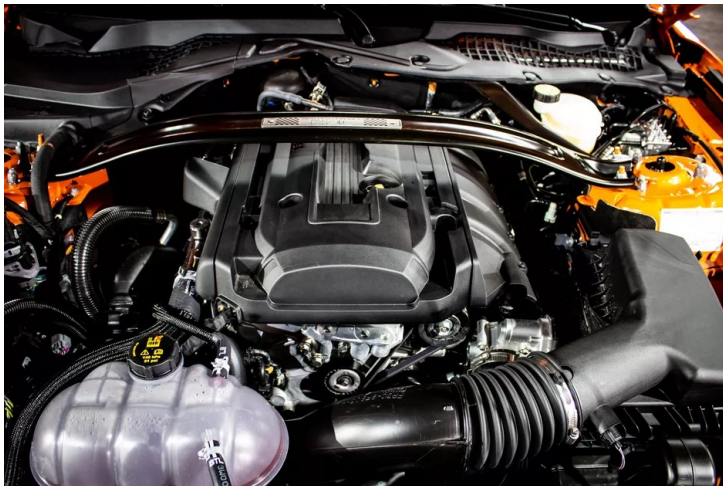


- A coding language and software environment.
- What runs the code.
- Not user friendly.



- An integrated Development Environment (IDE).
- What helps you create and test the code.
- User friendly.

An Analogy.

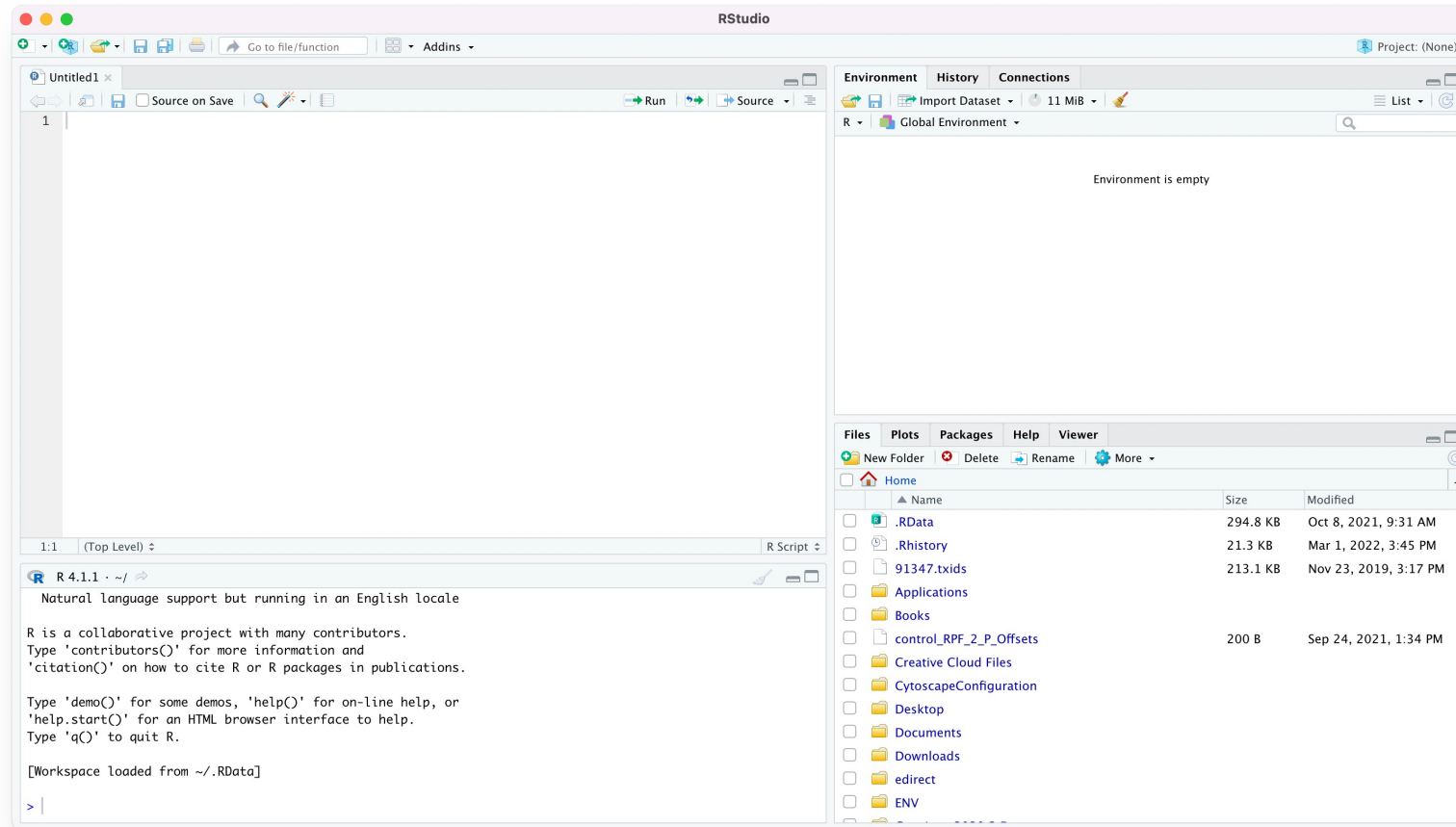


Getting Oriented with R Studio.

RStudio is split up into 4 main Sections:

Scripts
Section.

Console
Section.



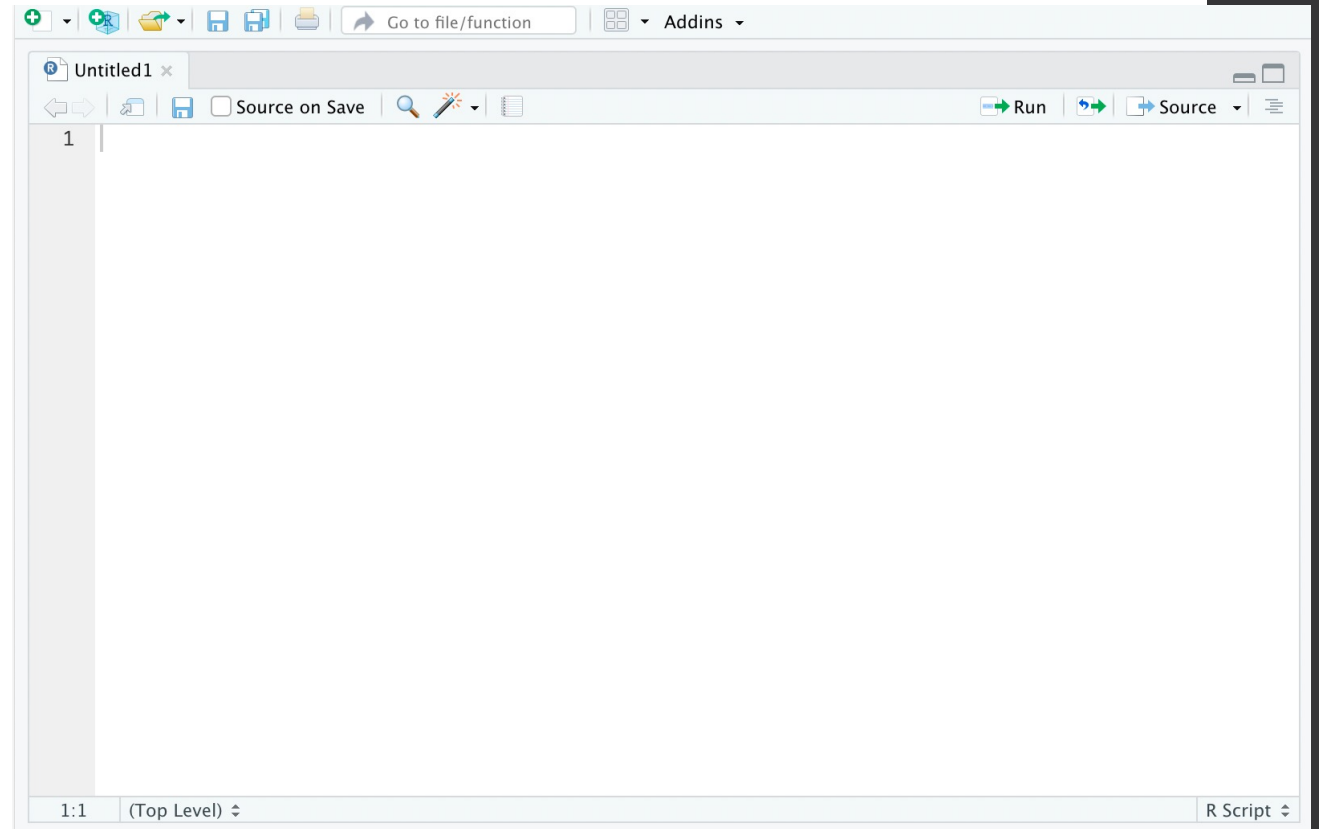
Environment,
History, and
Connections
Section.

Files, Plots,
Packages, Help,
and Viewer
Section.

Let's go through each of these individually.

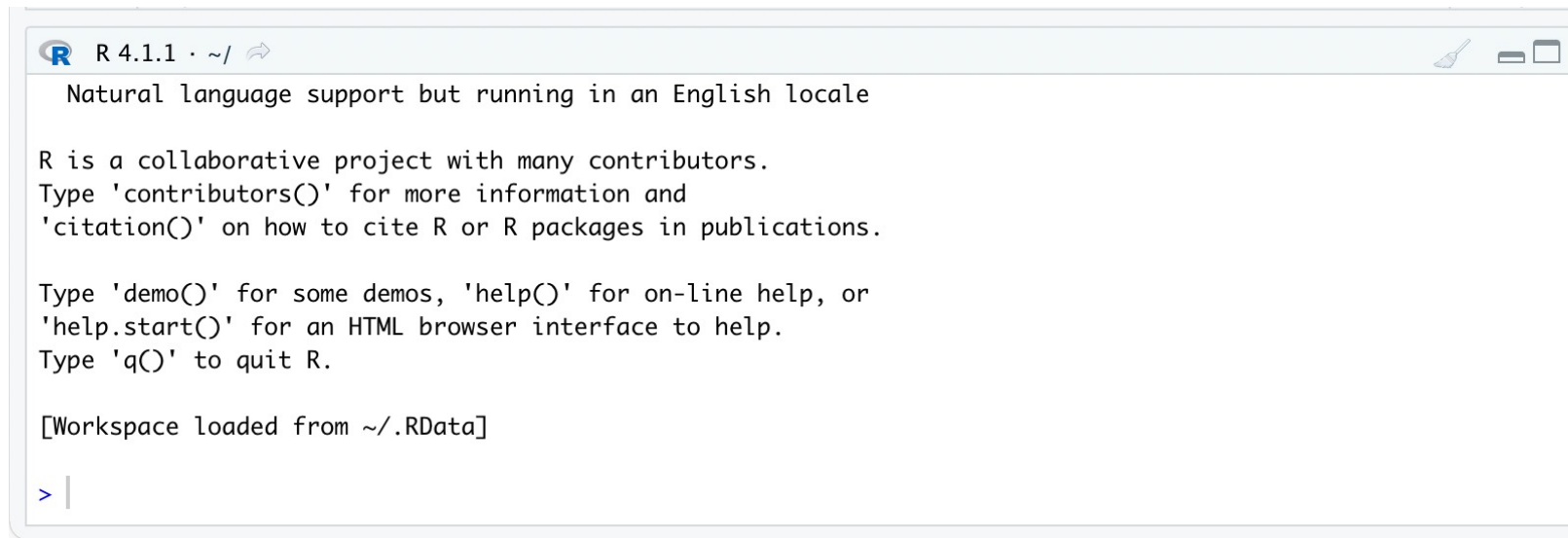
The Scripts Section

- A text editor where we will be creating Rscripts
- Rscripts are where you can write down all the commands you want to use (along with some helpful comments) and then run them in R.
- Coding in Rscripts makes our work repeatable.
- 99% of the work we do in RStudio is going to be within the scripts section.



The Console Section

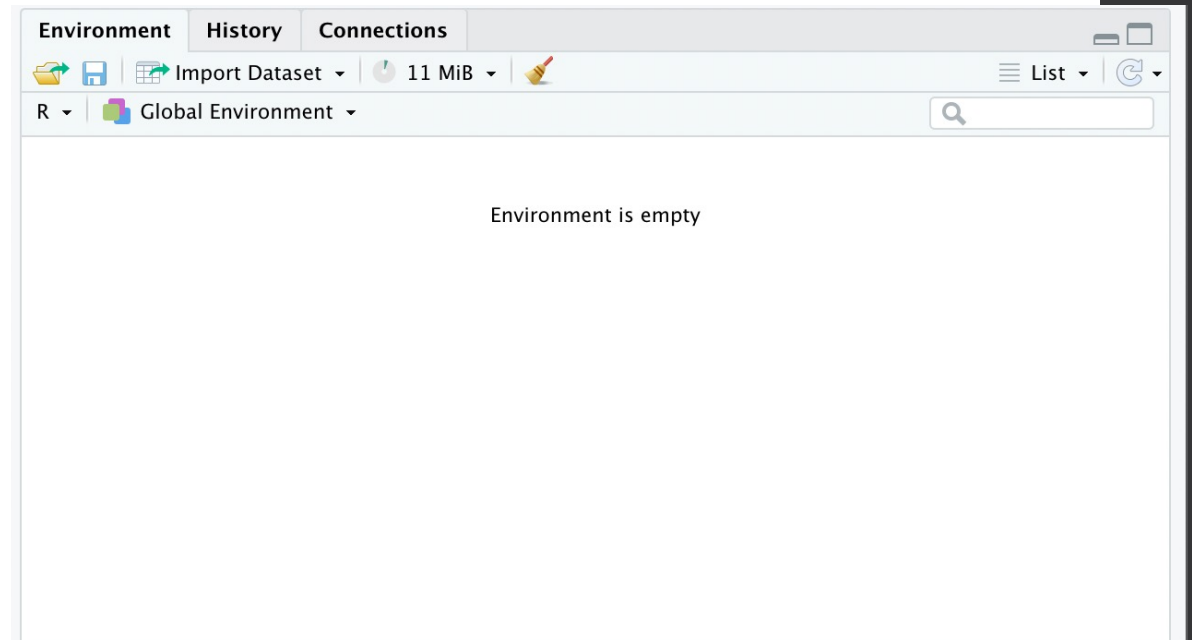
- The portion of RStudio that actually runs R.
- Commands can be typed directly into the console, but we will usually be typing are commands into Rscripts.
- The output of our code will be displayed in the console.



```
R 4.1.1 · ~/ ➤  
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.  
  
[Workspace loaded from ~/.RData]  
  
> |
```

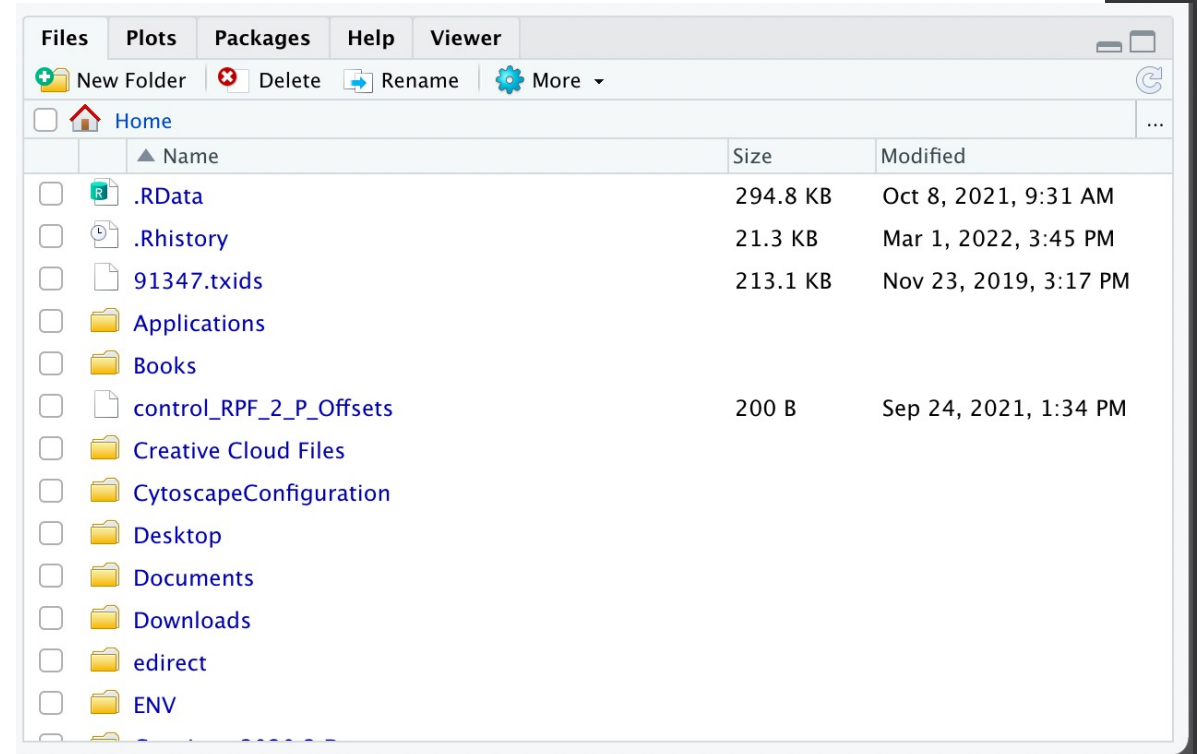
The Upper Right Section.

- Environment Panel: Gives you easy access to all the objects and datasets you create and download.
- History Panel: Contains a record of all the commands you have entered into the command line already.
- Connections Panel: Used to connect to a variety of different databases through RStudio. Will not be covered in detail during this tutorial.



The Lower Right Section.

- Files Panel: A basic file browser where you can open, delete, or rename files. It is usually preferable to use your systems built in file manager (e.g., Finder).
- Plot Panel: View plots created in RStudio.
- Packages Panel: See which packages you have installed, and which packages are loaded.
- Help Panel: View help files for functions or packages.
- Viewer Panel: View special media created in RStudio.

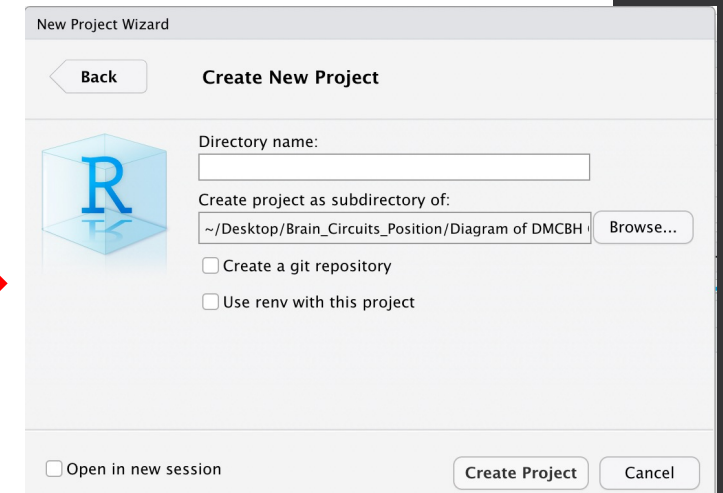
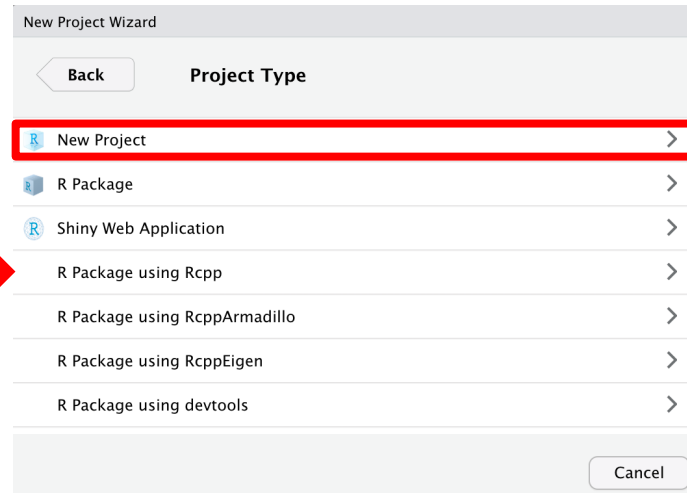
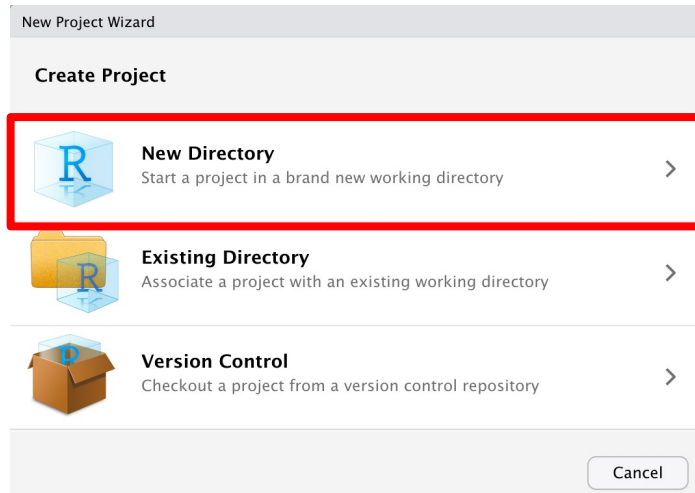
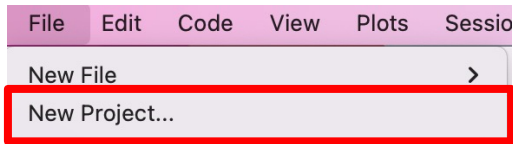


R-projects.

- What are R-projects?
 - R-projects are essentially just special directories that contain an R-project file.
 - Think of them like little boxes that contain everything you need for your project.
- Why use R-projects?
 - Allows you to easily define a working directory.
 - Preserves command history for the project.
 - Can integrate projects into version control (beyond the scope of this tutorial).

Creating an R project.

- Go to the file tab and select new Project.
- Select “New Directory”.
- Select “New Project”.
- Create the project as a subdirectory of our tutorial directory.
- Name your project directory.
- Hit “Create Project”.



R Markdown.

- R Markdown files are a special kind of Rscript.
- R Markdown is a powerful tool for integrating R-code into documents and websites.
- It is also good for creating structured tutorials which is why we will be using it today!

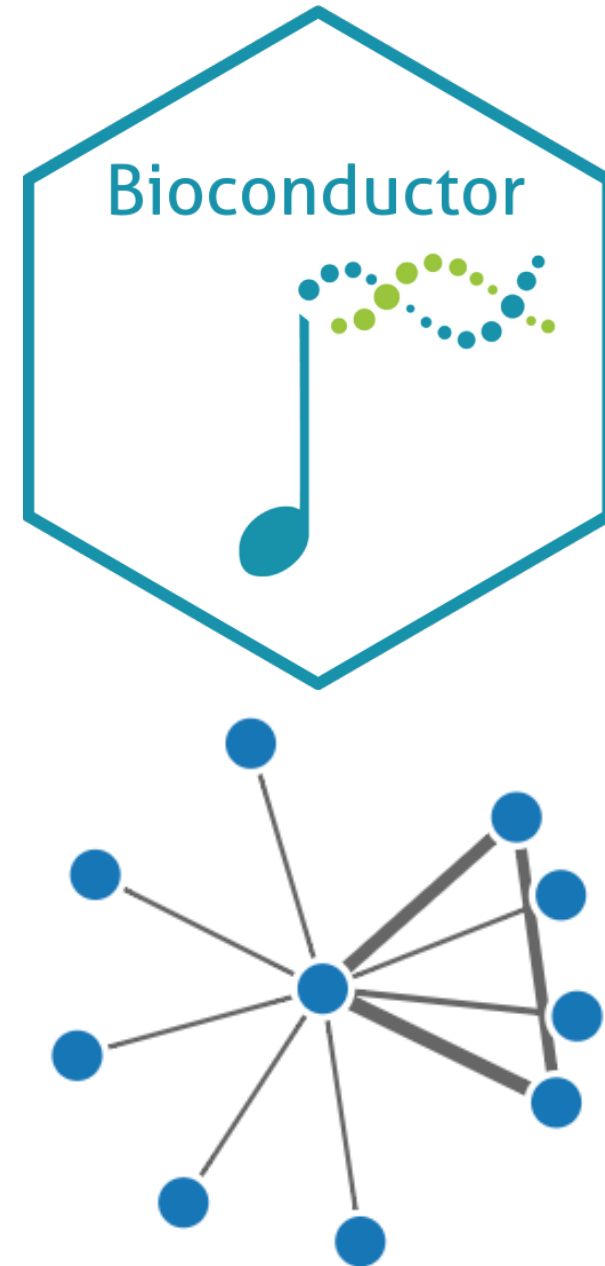
Our R Markdown files have two types of section in them.

- Markdown sections, which are essentially just text and cannot be run in R.
- Code sections which can actually be run in R. These sections are delimited by three grave accents (this thing ``` in the upper left corner of your keyboard) and `{r}`.



Packages.

- The best thing about R is the variety of open-source R packages that can be installed on your system.
- Packages are installed and subsequently loaded into your R environment using:
 - `install.packages("package_name")`
 - `library(package_name)`
- Packages are almost always required when needing to perform tasks like:
 - Performing a specific kind of statistical analysis
 - Working with specialized biological data (e.g. fasta files).
 - You need to create a complex figure and or visualization of your data.



Tidyverse

- Perhaps the most ubiquitous set of packages used in R are the tidyverse suite of packages.
- Rather than being one singular package, tidyverse is a set of packages that all work in harmony to improve data analysis in R.
- Notable packages include:
 - ggplot2 for improved graphic.
 - readr and readxl for loading different kinds of data into R (e.g. csv and xlsx).
 - stringr for string/text manipulation.
 - dplyr for advanced data manipulation.
 - tidyr for creating “tidy” data.



Variables.

- Whenever we calculate or create something in R, we can save that value in a variable (sometimes referred to as an object).
- Two ways to assign a value to a variable.
 - Using an equals sign like: `variable_name = value`
 - Using a little arrow like: `variable_name <- value`
- Personally, I much prefer using `=`.
 - Much faster to type.
 - More consistent with other coding languages.
 - Consistent with how we assign arguments in functions.
 - The arrow only exists as a relic of an older language called S.

Vectors.

- Vectors in computer science are essentially just lists of values which we can work with.
- Note that the way we are going to be using vectors is distinct from the way they are used in vector calculus and mathematics (objects with direction and magnitude).
- Vectors can be created by listing values separated by commas inside the `c()` function which is actually short for concatenate.
- Vectors can be indexed using square brackets like: `vector[index#]`

Functions.

- Functions are basically how we are going to do anything interesting in R.
- The complexity of R functions can range from completing simple tasks like calculating the mean of every value in a vector to performing incredibly complex tasks such as the completion of complex algorithms, intensive statistical analyses, or the creation and training of neural networks.
- Some R functions are made available in base R, but most of the most useful functions are accessed through packages.
- It is also possible to create our own functions which can be incredibly useful for performing tasks that are specific to our analyses (beyond the scope of this tutorial).

The Help Function.

- Almost every function in R has a dedicated help file attached to it that can be accessed using: `help("function_name")`
- Help files contain 6 sections:
 - A description of the function.
 - An overview of the usage of the function which includes the argument names and their order.
 - A list of all the arguments in the function.
 - Extra details on the functions
 - References to similarly useful functions.
 - Examples of the function in use..
- Note: I would also suggest trying to google complicated functions as there are often useful tutorials or more detailed documentation available online.

Standard Deviation

Description

This function computes the standard deviation of the values in `x`. If `na.rm` is `TRUE` then missing values are removed before computation proceeds.

Usage

```
sd(x, na.rm = FALSE)
```

Arguments

`x` a numeric vector or an R object but not a [factor](#) coercible to numeric by `as.double(x)`.

`na.rm` logical. Should missing values be removed?

Details

Like [var](#) this uses denominator $n - 1$.

The standard deviation of a length-one or zero-length vector is NA.

See Also

[var](#) for its square, and [mad](#), the most robust alternative.

Examples

```
sd(1:2) ^ 2
```

Dataframes.

- Dataframes are the fundamental data structure in R.
- Dataframes can be indexed in two ways:
 - By selecting individual rows using square brackets and index numbers like: `dataframe[index#,]`
 - By selecting individual columns using a dollar symbol (\$) followed up by the name of the column like: `dataframe$column_name`

Column names



	name	vore	bodywt
1	Cheetah	carni	50.000
2	Owl monkey	omni	0.480
3	Mountain beaver	herbi	1.350
4	Greater short-tailed shrew	omni	0.019
5	Cow	herbi	600.000
6	Three-toed sloth	herbi	3.850
7	Northern fur seal	carni	20.490
8	Vesper mouse	NA	0.045
9	Dog	carni	14.000
10	Roe deer	herbi	14.800
11	Goat	herbi	33.500
12	Guinea pig	herbi	0.728
13	Grivet	omni	4.750
14	Chinchilla	herbi	0.420
15	Star-nosed mole	omni	0.060
16	African giant pouched rat	omni	1.000
17	Lesser short-tailed shrew	omni	0.005
18	Long-nosed armadillo	carni	3.500
19	Tree hyrax	herbi	2.950



Index
Numbers

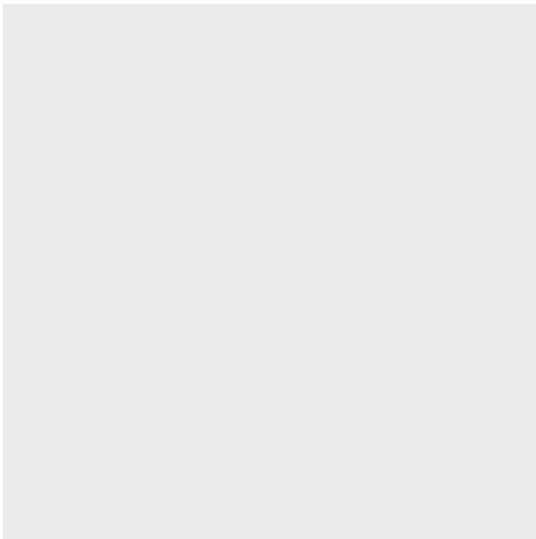
Relational operators

- Relational operators are small bits of code that return a TRUE or FALSE value depending on whether or not certain conditions have been met:
 - ==: check if two things are equal.
 - !=: check if two things are not equal.
 - >, <: check if two things are greater than or less than each other.
 - <=, >=: check if two things are greater than or equal or less than or equal.
- We can link relational operators together into "or" and "and" statements using "|" and "&" respectively.

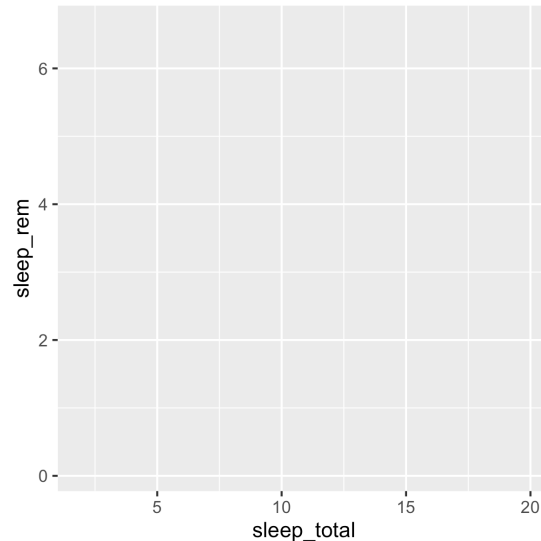
Plotting with ggplot2.

- ggplot2 is a powerful graphical package for R that follows the graphig philosophy outlined in “The Grammar of Graphics”
- At its most basic level, ggplot involves three steps:

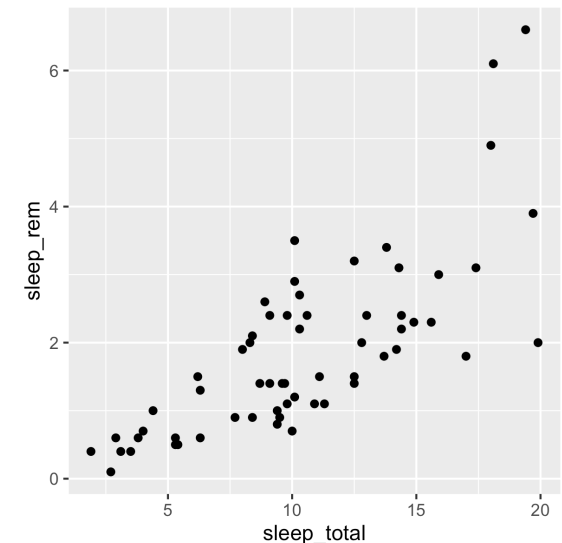
1. The initialization of the ggplot.



2. The addition of aesthetic mappings for the data, so that each data point maps to a visual property for the graph.



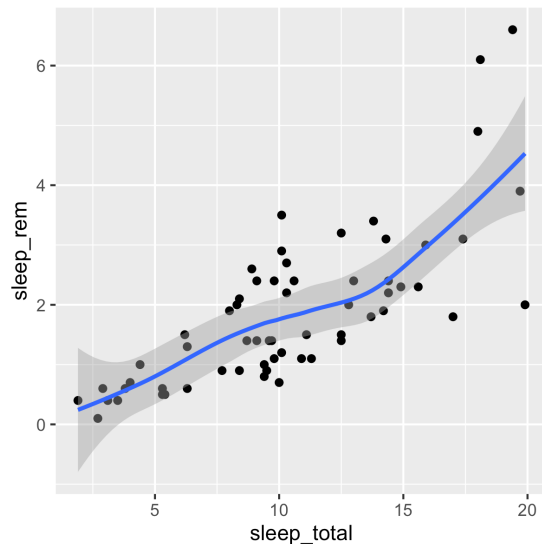
3. The addition of geometrical objects (geoms) which decide how each datapoint will be visualized (graph type).



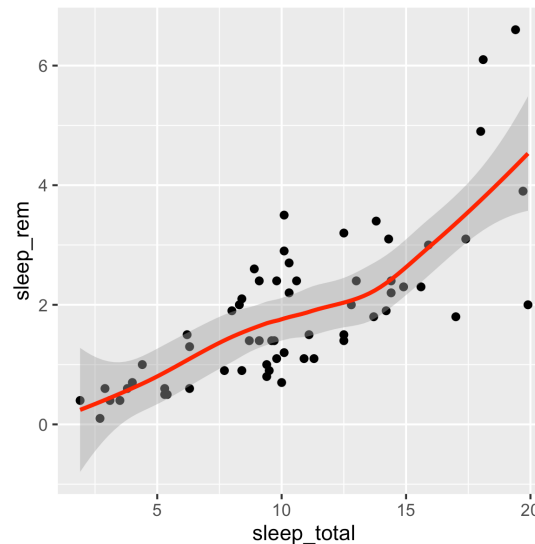
Adding complexity to plots.

Once we have created a basic plot in ggplot 2, we can add additional information to our plot by adding more graphs, adding labels, and through a variety of other techniques that are beyond the scope of this workshop (e.g. legends, scales, and faceting specifications).

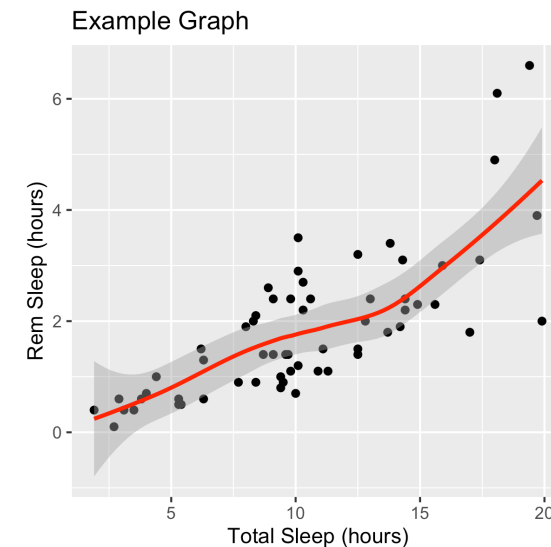
1. Adding another geom object to the plot in order to overlay a new plot on top of the old one.



2. Changing the color of the plots.



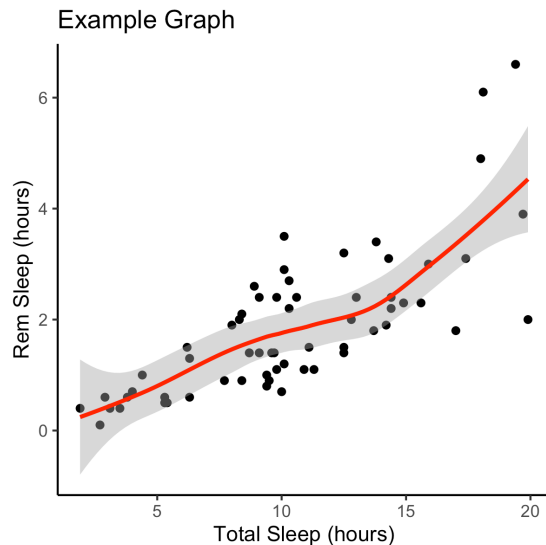
3. Adding titles and changing the axis labels.



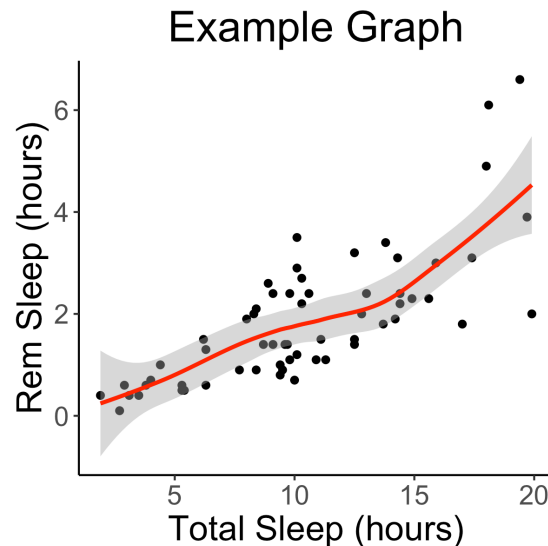
Adding style.

Once we are happy with the general adding new aspects to our plot, we are going to want to change the plot's style to make something more visually appealing. The easiest way to do this is by altering the plot's theme. Themes allow us to control the aesthetics of the plot in many different ways.

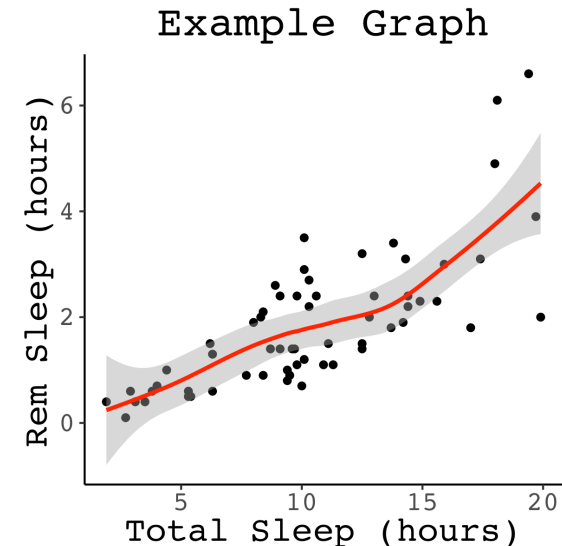
1. Adding a new theme can radically change the design of the entire plot.



2. We can use theme to alter the font size of the graph text.



3. We can use themes to change the font for our graph text.



Saving plots

- When we are done creating a plot using ggplot, we always want to save it using the ggsave function.
- Saving with ggsave allows us to:
 - Easily determine the name, path, and file type of the plot.
 - Squeeze or stretch our graph into a more appropriate shape.
 - Alter the resolution of the plot image.
- We do NOT recommend right clicking and saving a plot as this will give you no control over the image parameters and will make your work harder to share and reproduce as part of the analysis process is missing from the code.

T-tests.

- One sample t-test
 - Used to test if the sample statistic is significantly different from what we would expect if the null hypothesis was true.
- Two sample t-test
 - Used to test if two independent populations are significantly different from each other.

Assumptions of the t-test:

- One sample t-test
 - Random sampling
 - The variable is normally distributed. Robust to this assumption if the sample size is very large (central limit theory).
- Two sample t-test
 - Random sampling in both populations
 - The variable is normally distributed in both populations. Robust to this assumption if the sample size is very large (central limit theory).
 - The standard deviation of the variable is the same in both populations.

Additional Information

- R for Data science. Available online [here](#).
- An introductory R tutorial made by the Brain Circuits Research Cluster (link [here](#)).
- tidyverse. You can find more information on the packages in tidyverse on the tidyverse website [here](#).
- A ggplot 2 reference page that lists different geom names and the plots they make can be accessed from the tidyverse website [here](#).
- R Graph Gallery. A very useful website with well made tutorials on how to make beautiful graphs in R. For example, you can find a page detailing all of the available colors in R [here](#).
- ggthemes. A package that offers more themes than you would ever need for ggplot (link [here](#)).

Wrap-up slide

If you would like to ask any questions about this tutorial or about data analysis in general, then we encourage you to attend Databinge. Databinge is an interactive drop in meeting which you can go to for any and all data analysis questions:

- Link to Databinge Webpage:
 - <https://braincircuits.med.ubc.ca/activities/databinge/>
- Link for survey:
 - https://ubc.ca1.qualtrics.com/jfe/form/SV_9FhiWbxxs0Z9e3I