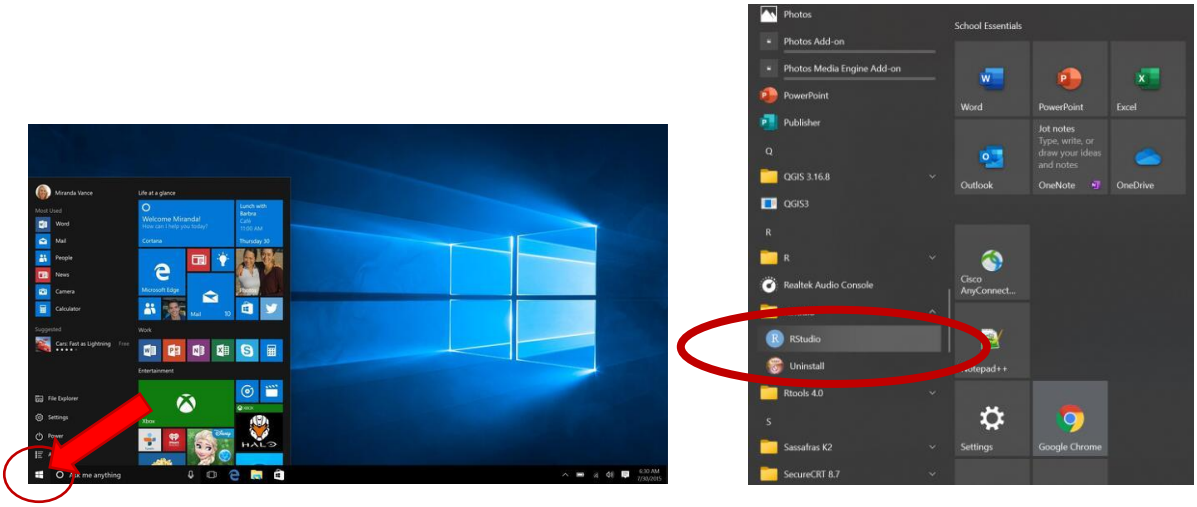
OVERVIEW

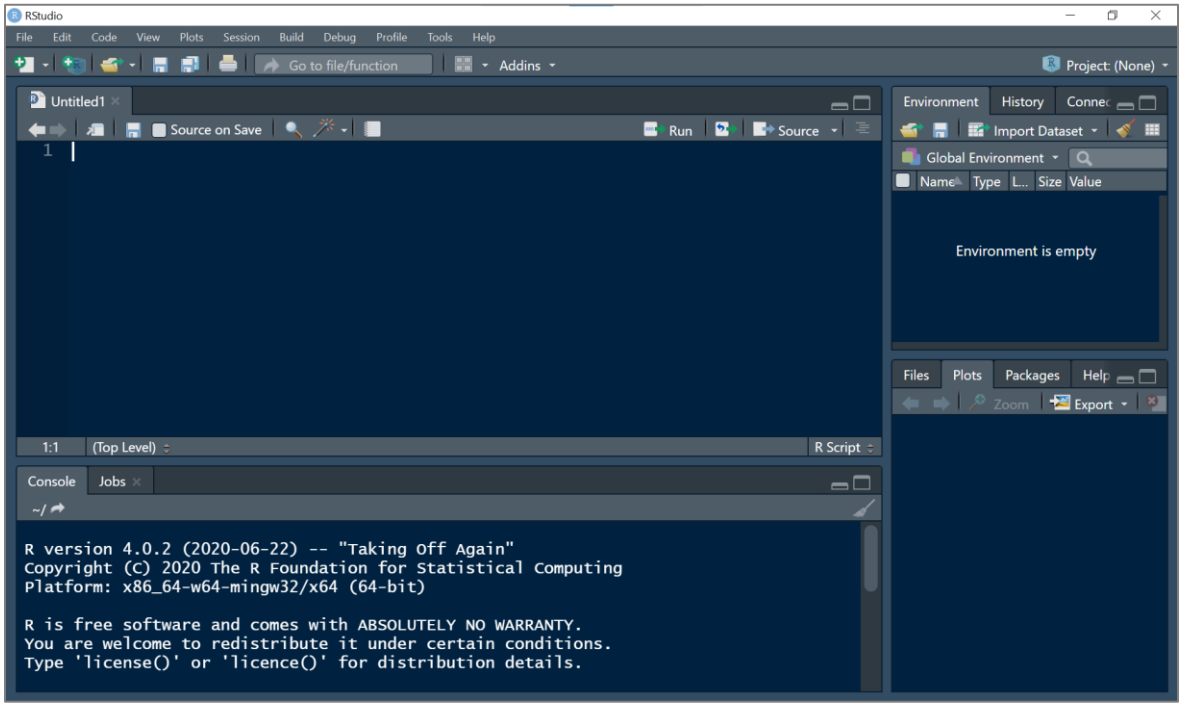
This in-class exercise provides a hands-on introduction to R and RStudio and reinforces several of the key takeaways outlined in the lecture. Because this is an in-class exercise, there is nothing you need to submit. The goal is to build familiarity with RStudio and build capacity for working with data in R.

LOCATE AND LAUNCH RSTUDIO

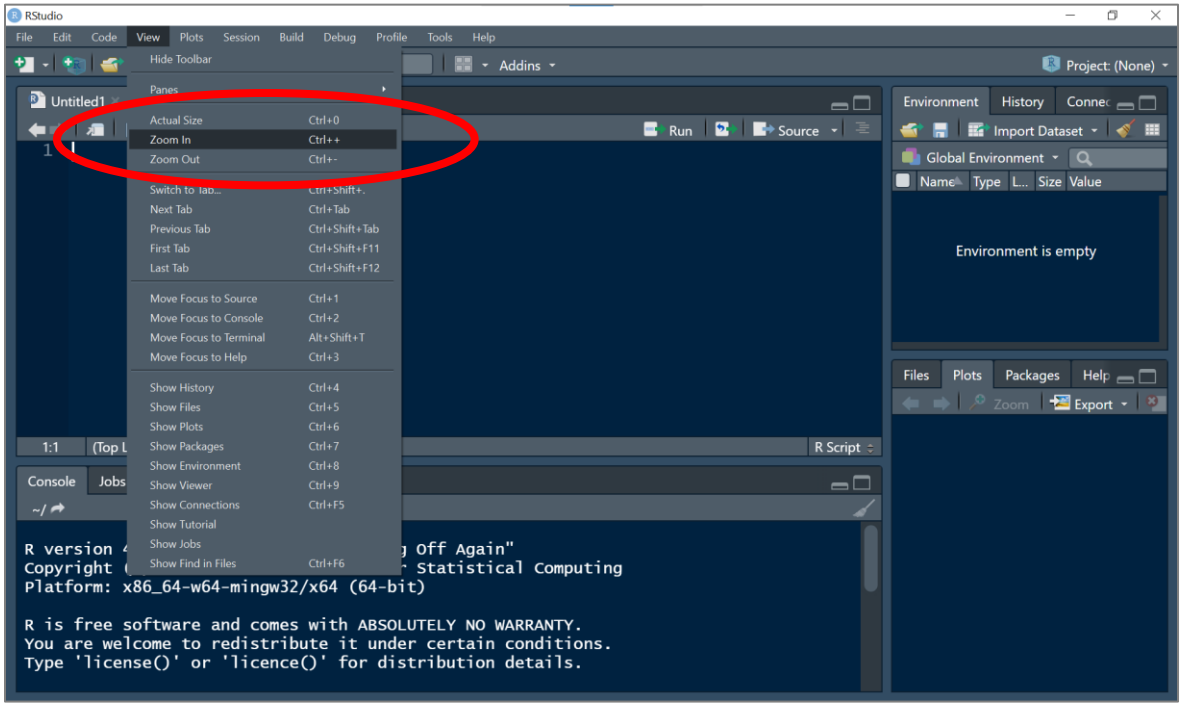
If you were able to successfully install R and RStudio on your laptop please launch RStudio there. If you do not yet have this configured on your machine, launch RStudio from one of the machines in computer lab (instructions for connecting to the UVA A-School Virtual Workstation are available here if you happen to be off grounds). The screen captures below assume that you are working with a Windows operating system, but R and RStudio also run on MacOS and Linux.



Recall that R and RStudio are separate applications. RStudio is an integrated development environment or IDE that makes it easier for us to interact with R, but you have to install both R and RStudio (i.e., RStudio alone is useless for our purposes) and RStudio is where we will be working. When RStudio opens, you will see something approximating this:

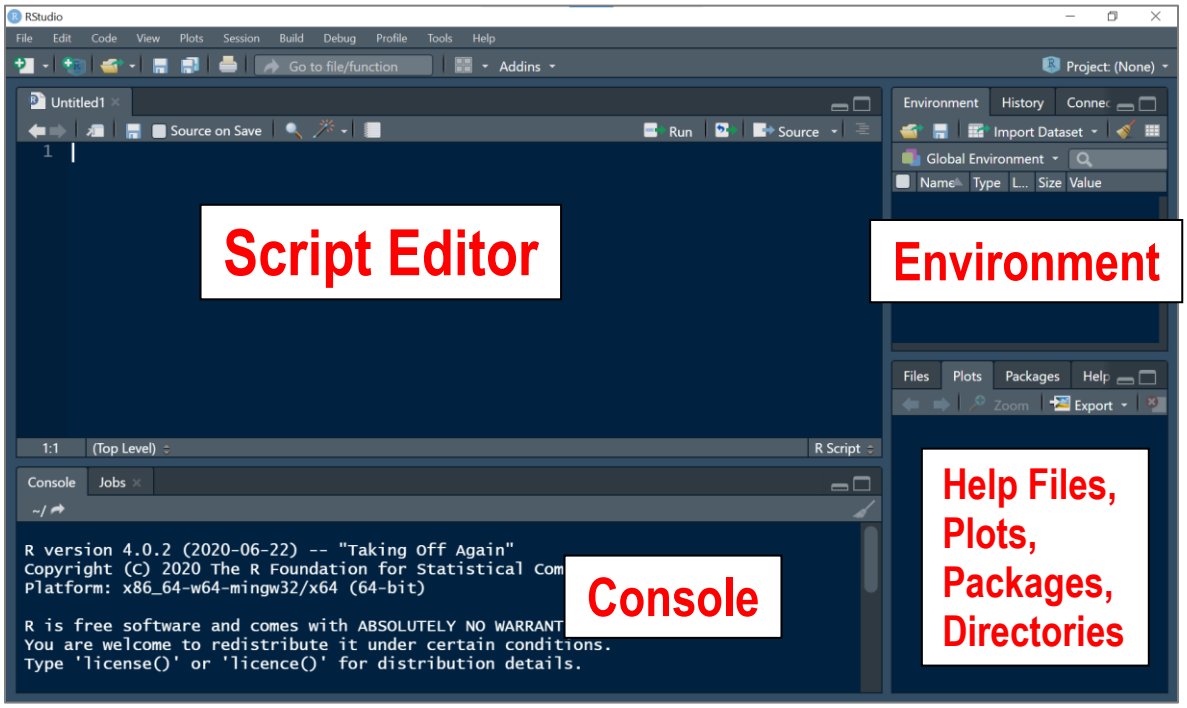


If you need to make the displayed text larger or smaller, use **Zoom In** or **Zoom Out** shown in the **View** drop-down menu below:



Take a few minutes to experiment with these and adjust the text size to your preference.

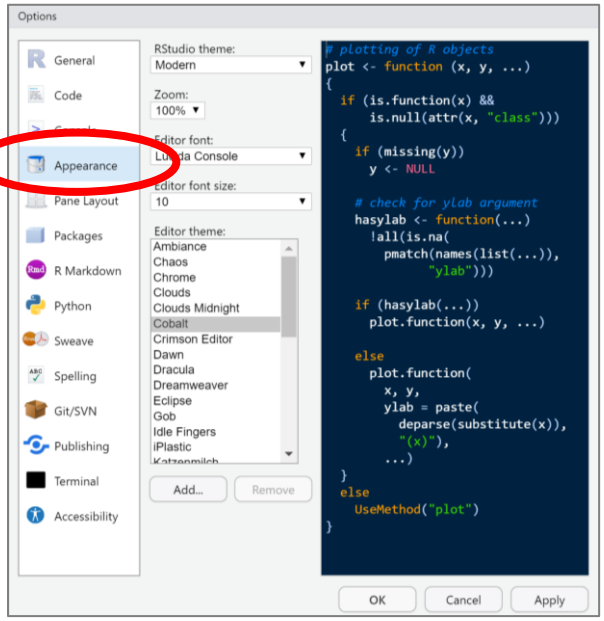
When you initially open RStudio, you will see three of the four windows shown in the image below. The panel labeled "Script Editor" can be accessed by clicking the **File** drop-down menu, then **New File,** and then **R Script** (or **R Notebook**).



- **Console**: The console panel allows you to quickly and immediately execute R code. You can experiment with functions here, or quickly print data for viewing.
    - Type commands next to the > and press Enter to execute.
    - To clear the contents, press Ctrl + L, choose **Clear Console** from the **Edit** drop-down menu, or click the broom icon in the upper right corner
    - The **Jobs** tab allows scripts to be run in the background
- **Script Editor**: In contrast to the **Console**, which quickly runs code, the **Script Editor** does not automatically execute code. The **Script Editor** allows you to save the code essential to your analysis. You can re-use that code in the moment, refer back to it later, or publish it for replication.
    - To open a new R Script, go to **File** → **New File** → **R Script** or press Ctrl + Shift + N
    - You should mouse over the icons in this window to get a sense for what they do—the **Run** button executes the line of code that your cursor is on or the highlighted lines of code
    - The **Source** button will execute the entire active document, but will not produce output unless **Source with Echo** is chosen
- Top Right:
    - **Environment**: Lists all objects that are saved in memory.
        - The **Import Dataset** and broom icon (delete all objects) are most frequently used
    - **History**: Lists all **Console** commands that has been used since the project started
        - The **To Console** button lets you revive past code quickly and broom icon deletes all history entries
    - **Connections**: Allows you to connect to external data stores (e.g., relational databases)
- Bottom Right:

- **Files**: Shows the files available to you in your working directory.
  - Navigate directories as you would in say Windows Explorer
  - Click the ellipsis (three dots) at the upper right corner to navigate directly
  - Click the **More** button and **Set As Working Directory** to designate a home folder for your work
- **Plots**: Graphical output appears here.
  - The **Export** button allows you to save graphics to the hard disk and the broom icon deletes all plots
  - Use the arrow button to toggle between plots
- **Packages**: List of all the packages you have installed.
  - You can install, load, remove, etc. R packages here, or from your code
- **Help**: Find help for R packages and functions.
  - Access same help features from the **Help** drop-down menu (including cheatsheets)
  - You can type ? before a function name in the console to get info too
- **Viewer**: View locally stored web content (e.g., .Rmd files knitted to .html)

If you want to change the default RStudio theme, from the **Tools** drop-down menu choose **Global Options** followed by **Appearance**:

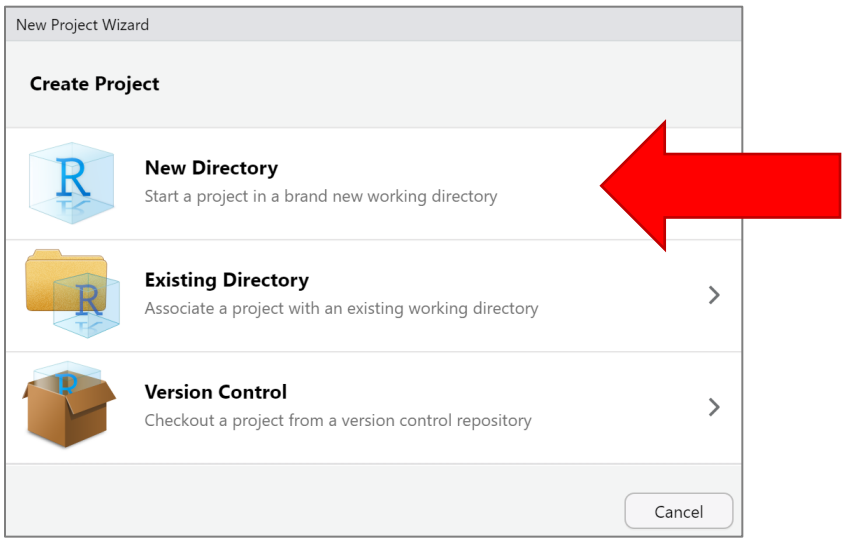


I am partial to the "Cobalt" theme, but find one that speaks to you.

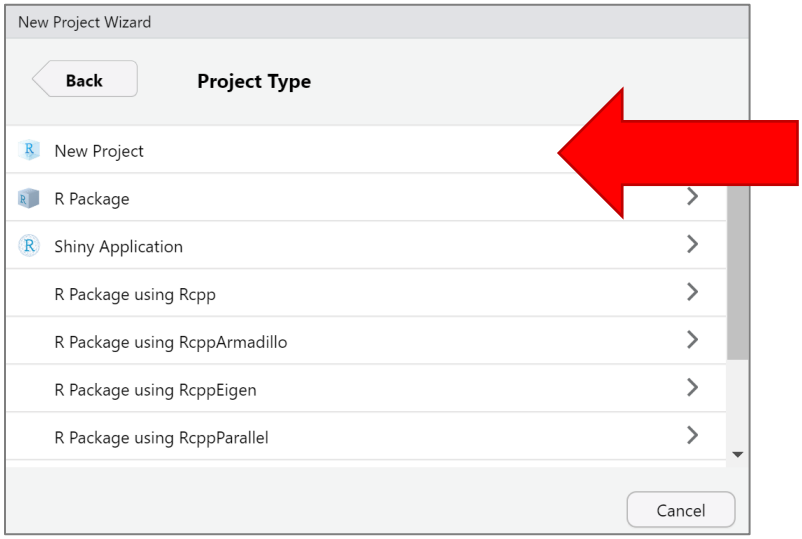
CREATE PROJECT AND SCRIPT

As outlined in the assigned reading, an **R Project** offers several features that may be of interest in your work. It eliminates the need to specify a working directory and reduces the frustration that can come with referencing datasets or outputs using absolute pathnames.

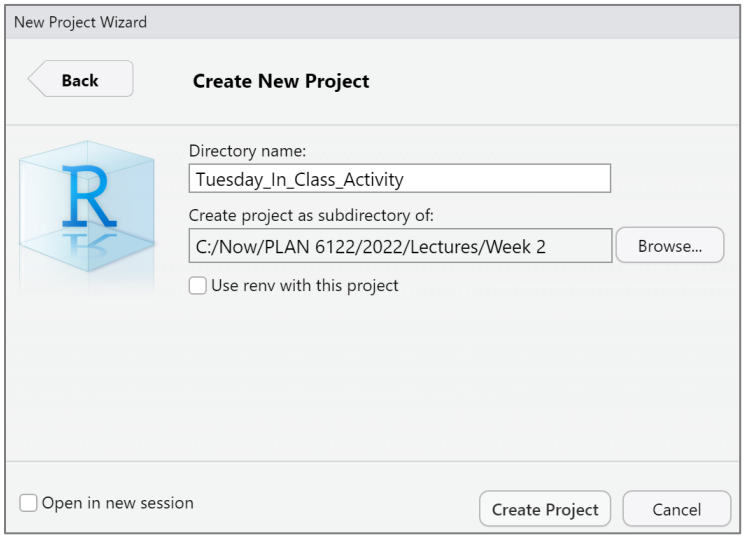
You can decide whether using projects in RStudio make sense for you and your work style—personally, I do not always make use of them in my own work. However, let's create one now!
From the **File** drop-down menu, choose **New Project**. As shown below, you can now decide if this project should be based on a new or an existing directory (if you already have datasets, scripts, etc.).



Let's choose **New Directory** followed by **New Project** as shown below:

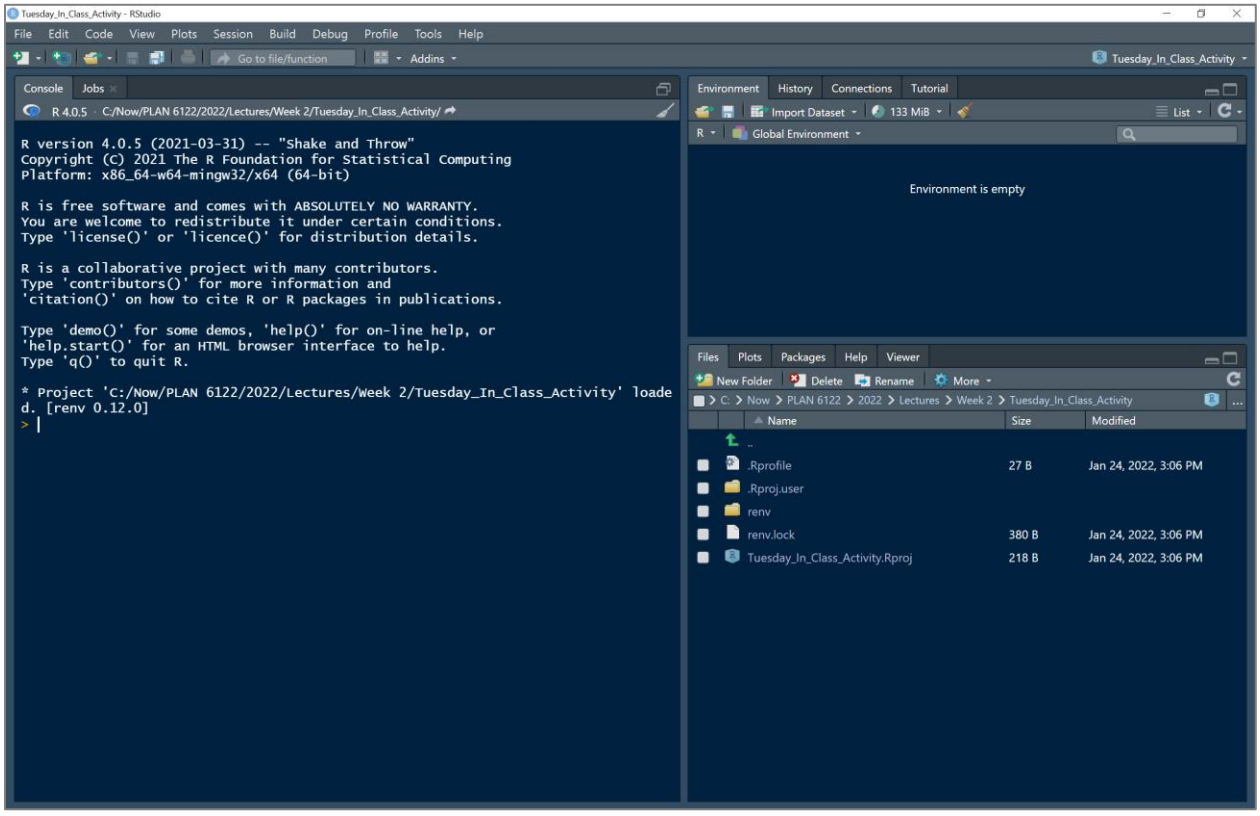


Use the **Browse** button to determine where the project will "live" and give it a name of your choosing:



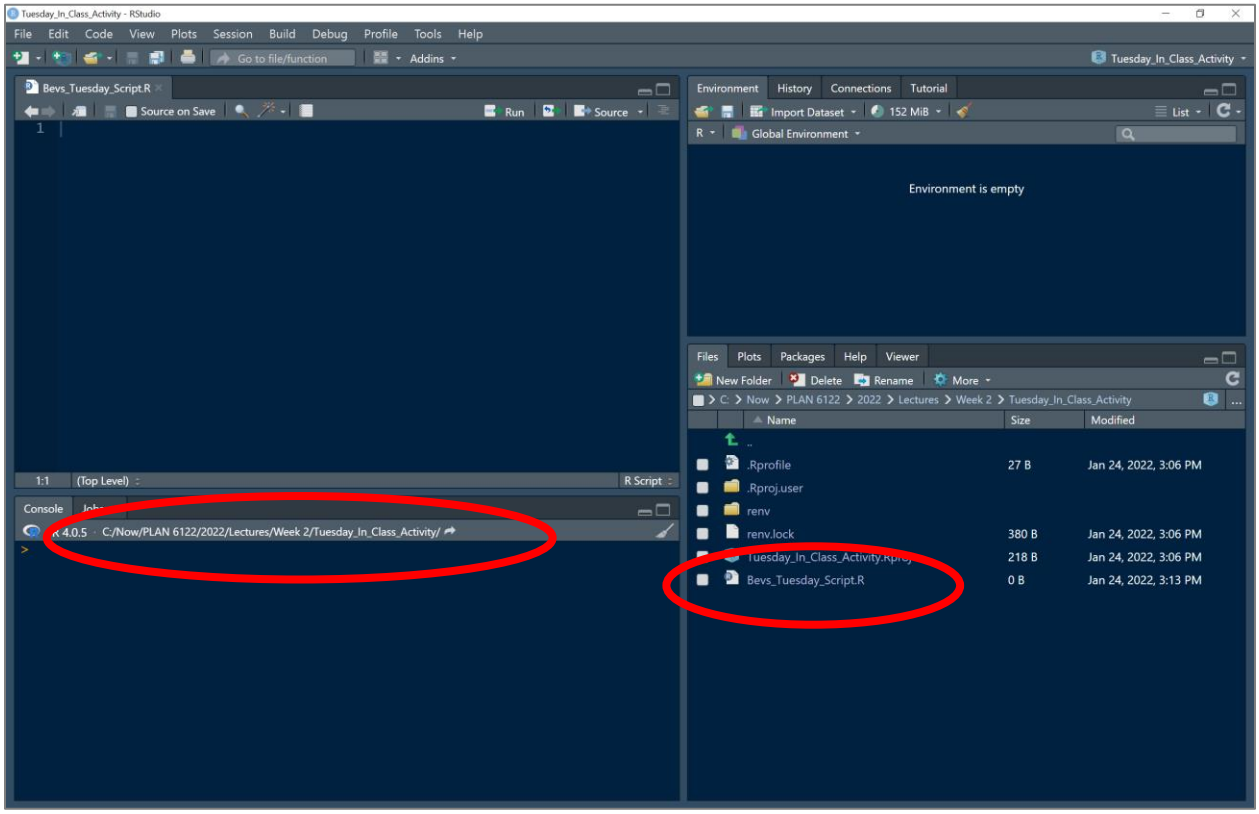
You can activate the **Use renv with this project** checkbox if you like. This uses the renv package to help ensure that package dependencies are proactively managed (e.g., if package A requires that package B is also available, package B will be installed/loaded along with package A) and is designed to facilitate reproducible work.

Note that the **Files** tab now shows the contents of the newly created directory.

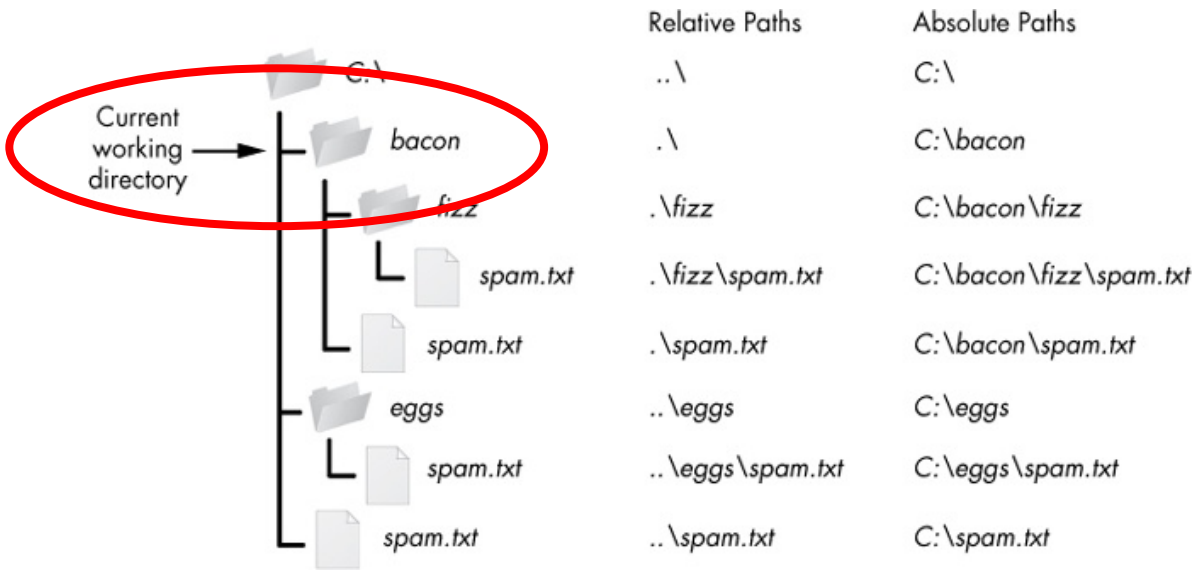


Clear the text shown in the **Console**. Refer to the preceding pages if you need help.

Now create a new R Script and save it. Note that the script now appears in the **Files** tab below and that the working directory for the project is also displayed in the **Console**:



Because we have created an **R Project**, all scripts plus the datasets they reference and the output they generate will be located in this directory, by default. Because everything is located in the same directory, we can use relative pathnames to reference datasets and output, which is generally helpful.



WHY USE SCRIPTS?

Scripts are essentially text files with a **.R** extension, but they are important because they allow us to create self-contained, documented, and reproducible workflows. Typically, we create a script file, populate it with code to accomplish one or more related tasks. An individual script might be called or sourced from another script or multiple scripts can be stored as an **R Project** to simplify the sharing of files between users or across computers where the directory structure is likely to be different. Maintaining your R code as scripts also means you do not have to re-invent the wheel for each project and over time, you will accumulate your own personal library of R code that can be tweaked and re-purposed as needed.

HOW TO ACCESS HELP?

The documentation can be accessed by typing a question mark, followed by the name of the function you are interested in. Alternatively, you can click the **Help** tab in the lower right RStudio panel and enter search terms there.

*** **Your turn**: Review the help documentation for the install.packages and the library functions. Now, write two lines of code in your script:

- One that installs the tidyverse package;
- One that loads the tidyverse package into memory so that we can access it.

Run the script (or those two lines of code).

HOW TO SET THE WORKING DIRECTORY?

The working directory is the default location where R stores files (output) and looks for files (input) and is typically only relevant when **one is not operating within an R Project** environment. This is where your datasets, scripts, etc. are found and the working directory can be any folder. In order to check which folder has been designated as the working directory, enter getwd in the **Console** or click the **Files** tab in the lower right RStudio panel followed by the **More** button.

*** **Your turn**: Review help documentation for the getwd function, then add a line of code to your script that displays the current working directory in the **Console**..

You can also set the working directory (change it to another directory) from a script using the setwd function, but more on this next week.

HOW TO ACCESS R PACKAGES?

In R, we rely on functions that allow us to manipulate data and objects. Packages contain functions, and all functions belong a particular package. For example, getwd belongs to the base package.

The standard R install gives you access to roughly about 30 packages, collectively referred to as “base R”. There are over 10,000 user-contributed packages which have been created for specific purposes and you

can explore these online in Comprehensive R Archive Network (CRAN), which is a collection of mirror sites hosted around the world.

One popular examples is tidyverse, which is actually a collection of several packages. In order to use a package, you must install it first.

- You can install packages via point-and-click: **Tools** → **Install Packages** then enter tidyverse (or a different package name) then click on Install.
- Or you can use this command in the console or script: install.packages("tidyverse")

Once you have installed the package, you must load the package in any new R session when you want to use that package: type library(tidyverse) to load the tidyverse package.

If you are working on the same computer, you only have to install a package once, but you must load it into memory each and every time you want to access its functionality.

ACCESSING BUILT-IN DATASETS

There are a variety of datasets that are installed as part of base R or along with individual contributed packages. If you want to view the available datasets, use the data() function. The package argument limits the results to a specific package and specifying the name of a dataset loads that dataset so that we can work with it. Note that data(package = .packages(all.available = TRUE)) will list all available datasets across all packages.

```
data()
data(package = "ggplot2")


data(txhousing)
```

Once we have loaded a dataset, we can explore it. Let's check the class, dimensions, contents, and structure of the dataset we have loaded.

*** **Your turn**: Write a few lines of code in your script that:

- Displays the class, dimensions, and attribute (column) names of the **txhousing** dataset that is built-in with the ggplot2 package;