Module 2.2: Learn - Coding

Overview

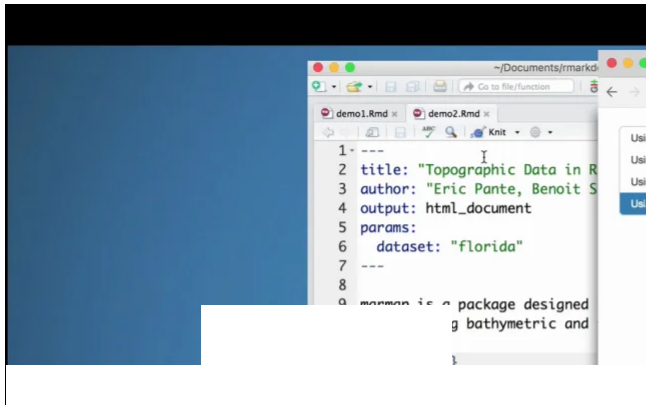
In this section, we will be introducing you to R by programming in R Markdown (Rmd) format. This week, you will work with three Rmds that have been generated for you to help you learn how to program in R using RStudio, how to take notes and print reports to share your analysis with others, and understand data types that we will be working with to do differential gene expression analysis in rest of the modules. We have provided some higher level explanations below to give you some background for what you will be trying out when you run the code.

In any programming language, there are lots of ways to do things, but we hope that the code that we have provided for you, will help you to understand the general workings of R so that you can more easily figure out how to write code to perform a particular analytical task and communicate with others about your code.

Getting started with R Markdown (Rmd)

In this course, we will be using R Markdown (Rmd) to write programs in R with lots of descriptions of what we are doing.

This video introduces the basic parts of an Rmd and the benefits of using it:



Video. What is RMarkdown?

This video discusses the RMarkdown authoring framework, types of content to run and display text or code, and how you can use it to replicate and share your work in different formats.

View transcript. (https://canvas.asu.edu/courses/122165/files/54792246?wrap=1) **(https://canvas.asu.edu/courses/122165/files/54792246/download?download_frd=1)**

In Module 1.2, we started an instance of RStudio using the RStudio Server on ASU Agave and tested some code will be working through some Rmds that introduce you to tools that you will be using to do gene expression analy the R Markdown format to keep notes, run code, and create reports.

RMarkdown (Rmd) is a simple formatting syntax that allows you to save, execute (or “run”), and print R code into Word documents. The user-friendly interface of RStudio gives you the ability to connect data to code that applies visualizations that help you interpret your data. In addition, when your code is complete, you can export reports r

reproducing. This allows you to share your work with an audience who can then replicate exactly what was done you're reporting.

Activity 1: Running a Rmd in RStudio

To learn how to run a basic Rmd in Rstudio, let's open one up. Last module, you copied and unzipped the code p directory:

```
/home/<ASUrite_id>/CURE_2022_Scripts/Week_2
```

Repeat the process you went through in Module 1 to start an instance of RStudio from ASU Agave RStudio serve login.rc.asu.edu and selecting the RStudio Server Interactive App.

Use the File menu, Open File, to navigate into your Week_2 subdirectory and open:

```
Week_2_IntroToRmd-Map_of_RStudio.Rmd
```

You should see the Rmd file open at the top of RStudio. The first section of lines are the YAML header containin which instructors wrote the code (update this field to add your name), and a formula that will print the current date code to a report.

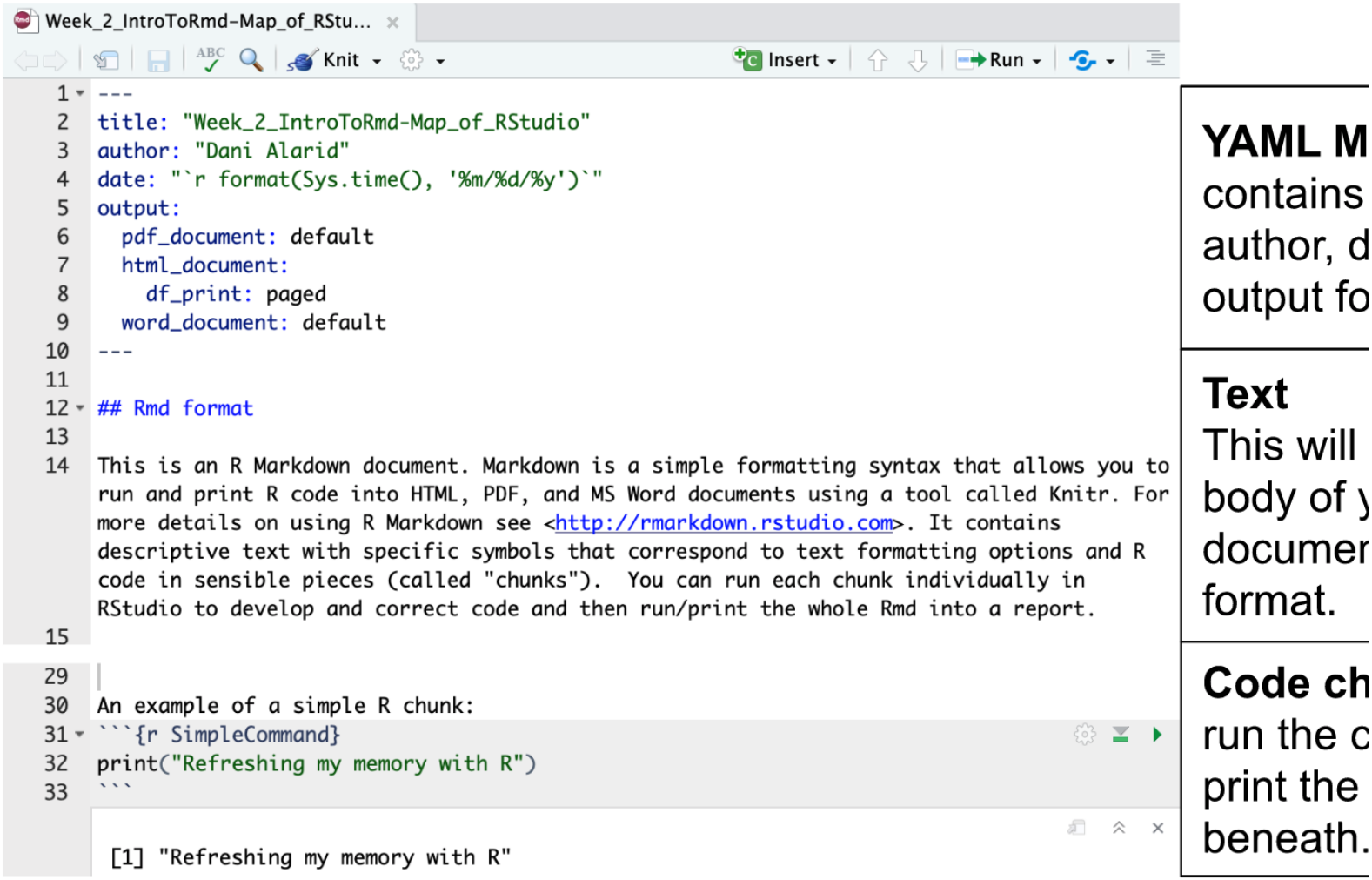


Figure. The anatomy of R Markdown (Rmd).
The three main content types of an Rmd are (1) the YAML metadata, (2) the text, and (3) code sec that allow you to format the header, body, and code output, respectively.

As you scroll down, you will see headings with hashtags at the front of the line and a block of simple text to expla information. The number of hashtags are used to decide the level of heading, with 1 hash tag being a title, 2 hash

heading, 3 being a subsection heading, and so on. This will be turned into a clickable outline when you print you skip to specific sections.

The parts in gray are blocks of code called chunks. Code chunks that contain code that can be executed line-by- using the play button (▶) on the right side. Code chunks start with three backticks (```) and have the programr case, R) in curly brackets (```{r}) and stop with three backticks (```). You can also build chunks by clicking th Insert) and selecting 'R'. When you run the chunk line-by-line, you will see the output in the Console and if yo the play button you will see the output just below the chunk in the tab containing the code. Once you know what a do, you can just press the play button if you want to rerun the whole thing.

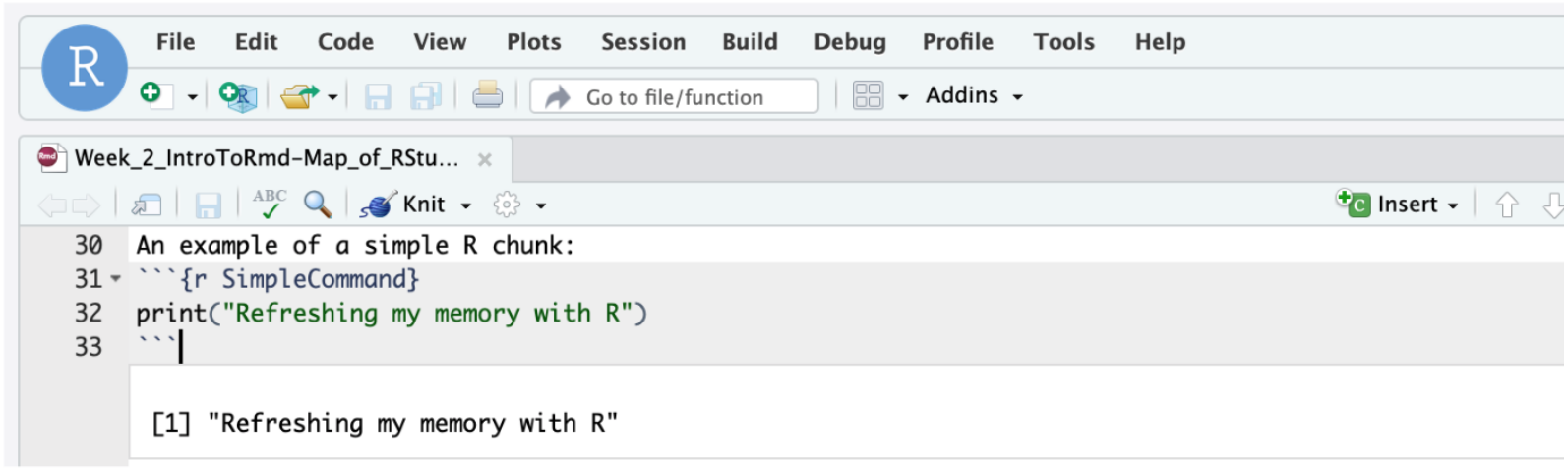


Figure. Running a code of chunk.

The code chunks in the grey can be run by clicking the Run button.

You can also run each chunk by selecting the Run button (Run), highlighting the desired code, and selecting (or by keying 'Ctrl+Enter' on Windows or 'Cmd + Enter' on Mac as a shortcut). Notice you have many options for your chunks and the keyboard shortcuts are listed to the right of each option. As you get more comfortable worki RStudio, you will get used to which ways of running code work best for what you want to do.

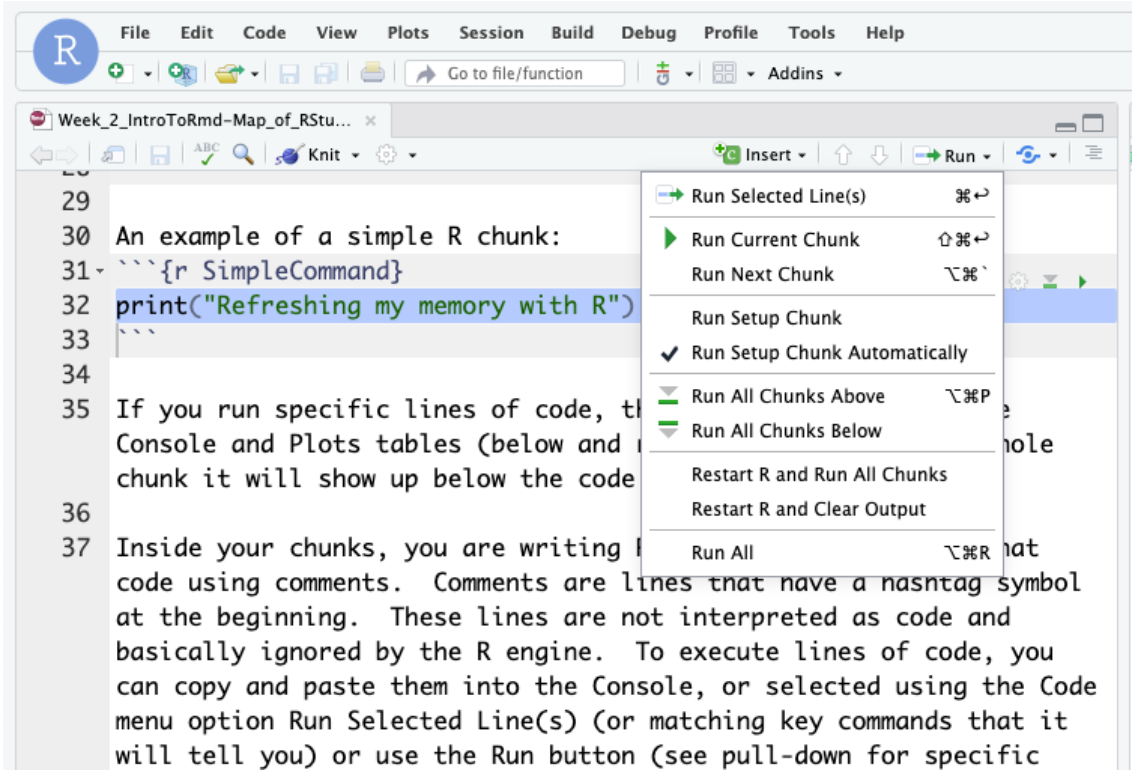


Figure. Running a code chunk by selected lines.

You can choose which lines of code to run by selecting them with the cursor then clicking "Run Selected Line(s)" or pressing 'Command + Enter'

Around the lines of code you will see lines that have a hashtag at the front, these are called comments and are ig engine. They are just there to help you to understand what the code means. For example, in the `Week_2_Intro Map_of_RStudio.Rmd`, one code chunk contains over 25 lines but due to the silencing of the instructional comm prints out three (3).

```{r CodeComments}

# this is a comment, indicated by a # symbol at the beginning of the line
# comments are not interpreted as code
# they are used to help explain code to others as they go through it

# below this line is real code
print ("This is code-- print is a function and this message is passed into it")

# we can use variables to store information
# here the variable called print_this_line is created and set to store specific test
# variables in R can be set using the arrow operator '<-' or an equal sign '='
print_this_line = "Hello World!"

# the value of this variable can then be passed into a function
# here we use the function 'print' that comes with R by default
print (print_this_line)

# you can change the value of variables
print_this_line = "And now print this!"

# and then pass those into a function
print (print_this_line)

# this variable is a data frame, a type of variable you can set in R
# if you run this line, you should see a variable called emp.data in your Environment on the upper right
emp.data <- data.frame(emp_id = c (1:5), emp_name = c("Rick","Dan","Michelle","Ryan","Gary"), salary = c(623.3,51
# double click the variable so you can look at it in a new tab above

```

```
[1] "This is code-- print is a function and this message is passed into it"
[1] "Hello World!"
[1] "And now print this!"
```

Figure. Rmd chunk with Code Comments.

The hash (#) is used to exclude lines from being run and to leave comments.

Run the template code line by line so that you know what the commands mean.

If you enter a line of code and see that there is an error or the output is not what you wanted, you can click inside up arrow to go back to the line of code that you think should change, navigate the line using the left and right arrc then try running it again. Once the line is modified for the desired result, you can copy the corrected code and pa and save it.

You can also see all the code you have run in the console in the ‘History’ tab on the top right.

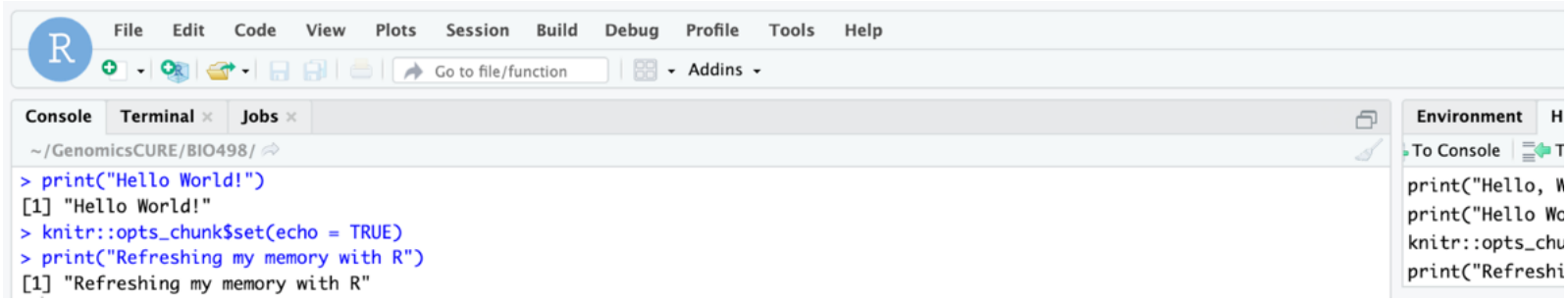
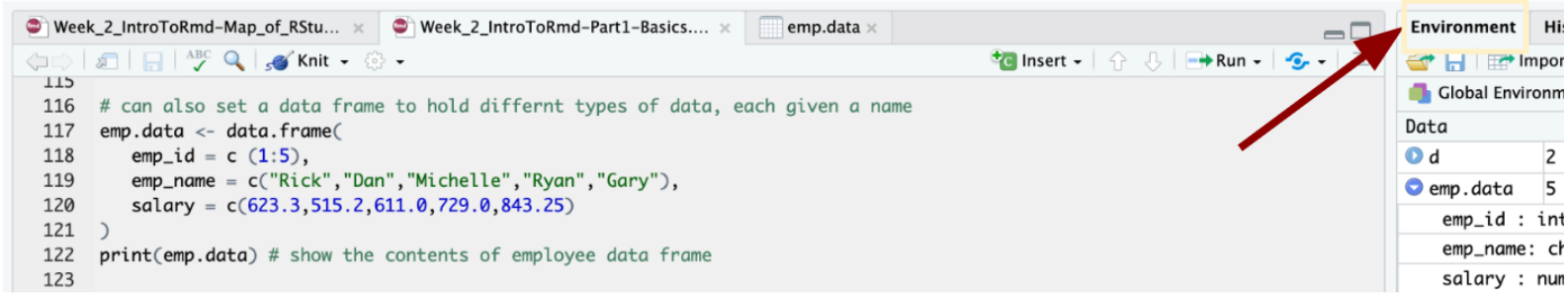


Figure. The History tab.

The History tab shows the last commands run or entered into the console.

As you run code, you will see the code going line by line in the Console and data being stored in variables in you the top right. You can click on the variables to see what has been stored inside.



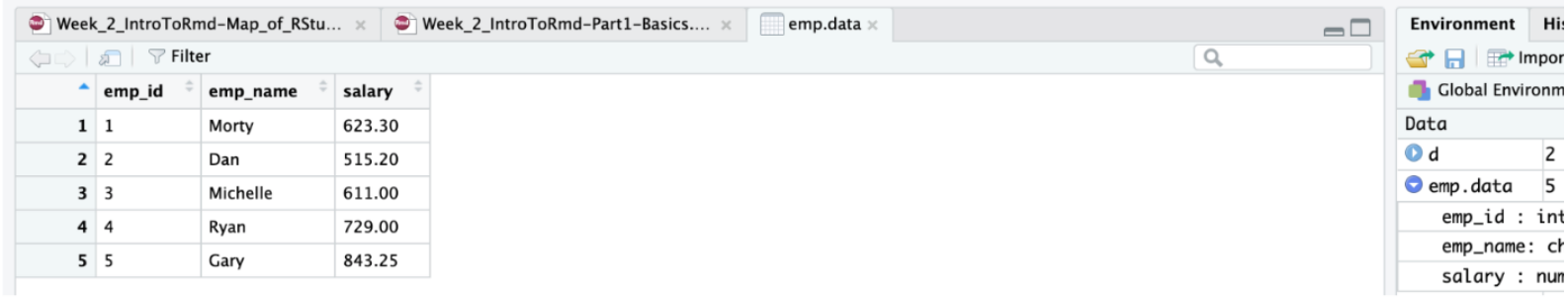


Figure. The Environment tab.

The Environment tab contains the document's variables. Clicking on the table icon to the right will open up the table in its own Source pane tab.

If you look at the bottom right, you will find some more tabs that are useful.

Figure. The bottom right pane and its tabs.

Files, plots, packages, help, and viewer are all useful RStudio tabs with different functions; see the table below for the description of each. Note: pane position is not absolute and some users may rearrange them. We are referring to the bottom right corner of the default settings (as pictured).

Table. The tabs in the Files pane in RStudio.

Tab	Description

Files	Lists the files and folders in the current working directory so you can their names if you want to read in and process data from them.
Plots	This is where plots might show up depending on how you generate them; you may also see th code in the Rmd tab.
Packages	This is where you can see a list of publicly available packages that have been installed that yo simply clicking the check box. If you press the Install button, you can install packages that you to test them out
Help	Where you can look up functions to know better how to use them

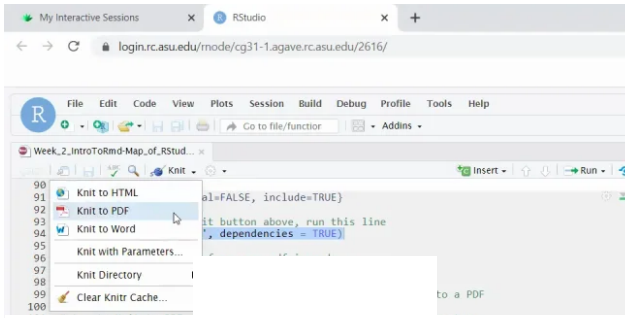
It is worth mentioning that panes can be rearranged and settings modified by clicking the pane button caret (Layout…' Beyond rearranging the panes, this is where you can modify the appearance and customize your expe preferences while working in RStudio.

When you are finished, exit the RStudio Server by clicking the power button ( ) after your ASUrite ID in the top the app. This will prompt you to save your changes if you haven't already clicked the save button ( ). You want and it is recommended to "save as you go" to avoid any unforeseen circumstances that could make you lose you button ( ) will prompt you to save all the files you have open in the source pane. Note: both save options save t your local machine; you can transfer the file to your own system if desired.

Once saved, you can close the tab or window you were working in. In the remaining session tab, delete the activ Interactive Apps browser and click 'Confirm' when prompted. The active session should disappear on the Interact you can select the RStudio Server button on the left to start a new session.

Knit Rmd Reports

By writing your R code in an R Markdown format, not only can you include important descriptive information to he understand your code and you can also easily print a report that can be shared. This is done using the Knitr pack specific rendering packages to help print images of the outputs you generate. Knitr runs all the code in your Rmd description, code, and output to a report. Knitr can print reports in many file formats, which can be specified in the header.



Video. Knitting Reports in RMarkdown.

This video shows how to knit a RMarkdown to save in different formats.

View Transcripts. (https://canvas.asu.edu/courses/122165/files/54792243?wrap=1)

(https://canvas.asu.edu/courses/122165/files/54792243/download?download_frd=1)

The video above shows that you can click the Knit button (Knit) at the top of the tab and then select the desi most common are pdf, html, and Word document. For this module, you can knit .html files that should pop-up in a version of your work in a new tab or window. If you look in your Files tab, you should see the report saved in the s Rmd file with the same name as the Rmd file but with the file suffix changed to indicate the output format (.html, . can turn in or share this report with others.

If you get an error printing to PDF, it could be because you do not have the underlying functions needed to gener properly. There is a section in `Week_2_IntroToRmd-Map_of_RStudio.Rmd` called `InstallKnitr` that cont install `tinytex`, a package that allows you to create PDFs. Running that chunk should take care of the problem

Lastly, if you change your code and press Knit to print a new report, the same output type will be selected autom report will be overwritten. If you want to save multiple versions of a printed report, you can rename the current re again using the button with a blue arrow in the Files tab to rename it so it is not overwritten.

Coding Errors

As you run code inside the Rmd, either by running selected lines or by running a whole chunk, you will get errors something doesn't make sense to the R interpreter. If you are running line by line you will see errors in the Cons knitting a report you will see the errors in the R Markdown tab (image below). If you have any errors in your code from being able to export the document. We will talk more about debugging errors in the next modules, but for no the error carefully to see if you can figure out what might be the problem and fix it.

Please post to Slack to ask your classmates as they should be running this code too.

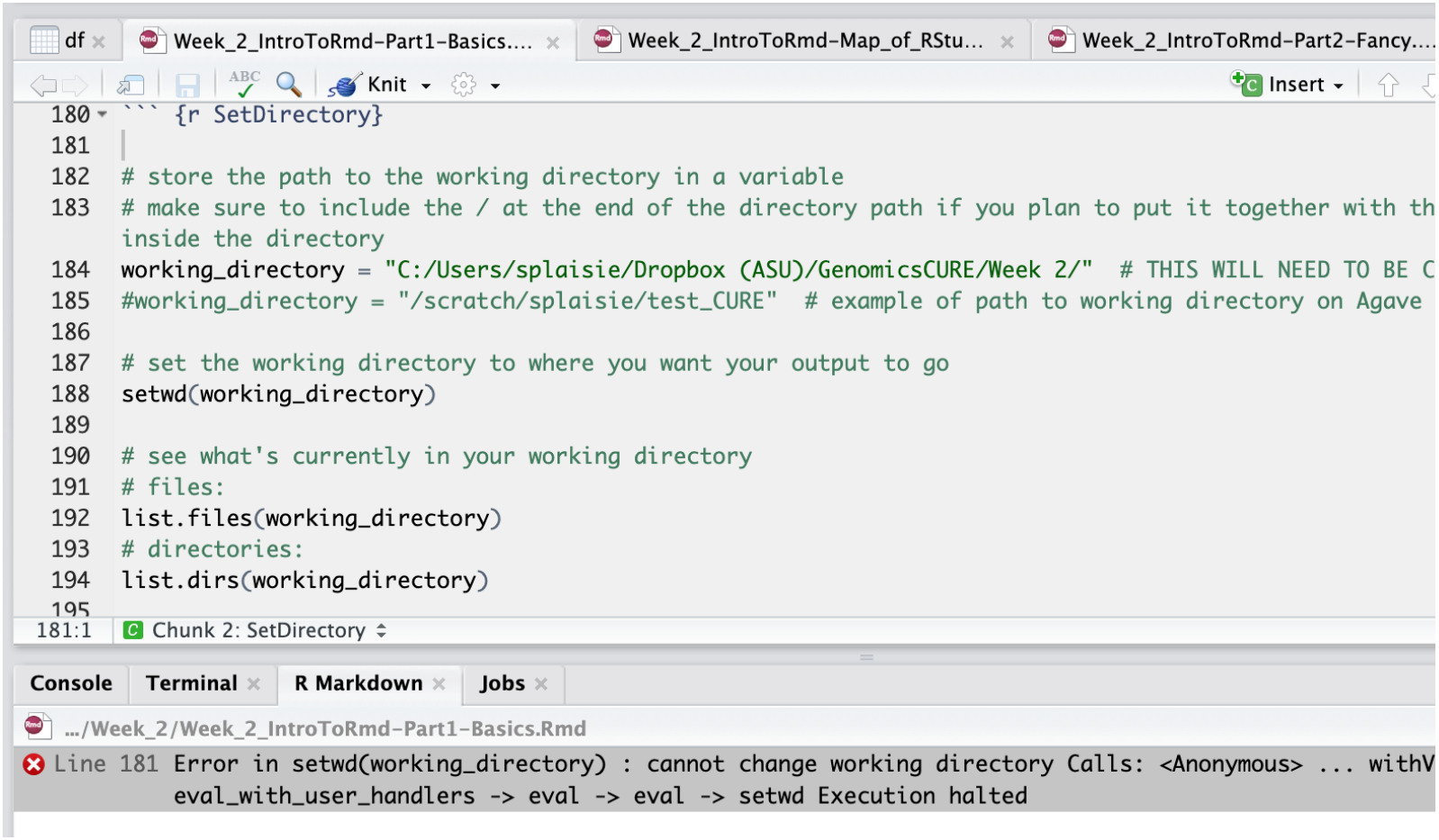


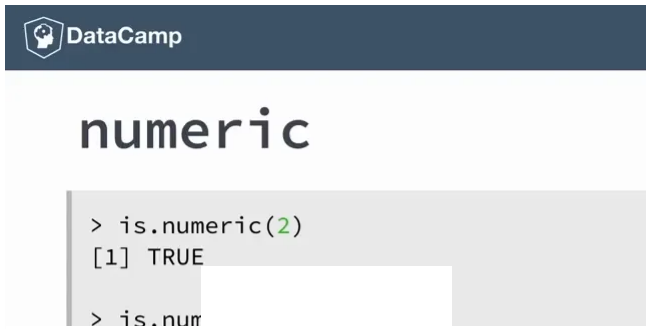
Figure. Error code while trying to knit.

In this case, the R Markdown tab in the bottom pane directs you to the `setwd(working_directory)` comman with line 181. Do you see what the error could be? That's right, we were supposed to change the (`working_dir` 184 to our `/CURE_2022_Scripts` directory because the current path is a local directory on Dr.Plaisier's comput doesn't exist in your data on Agave. Once the error is fixed, you should be able to successfully knit your code to a

Data types in R

When working with different types of data in R, you will store them in a named space in memory called a variable require you to specify what kind of data you are storing in a variable upfront, a data type will be assigned as soor

This video outlines the different data types we will be working with in R:



Video. Basic data types in R.

This video will introduce you to the main data types used in R: numerics, logicals, characters, and

View Transcripts. (https://canvas.asu.edu/courses/122165/files/54792242?wrap=1) **(https://canvas.asu.edu/courses/122165/files/54792242/download?download_frd=1)**

Table. Basic data types in R.

Data type	Description	Example
Doubles/numerics `<dbl>`	(1) whole numbers (2) fractions or decimals	(1) 123 (2) 123.5

Logical (aka “Boolean”) `<lgl>`	This is when you have two categories to choose from	True/False
Character `<chr>`	(1) A single letter or (2) a string of text	(1) “W” (2) “Word”
Factors `<fctr>`	Categorical variables	Groups/lev

You can check the data type of an object using the `typeof` command.

```
typeof(my_variable_name)
```

We can also store a collection of values in data types called data structures. Different data structures will have d
and have features like indices to allow you to access specific elements in the data structure as needed.

Table. Data Structures and dimensions.

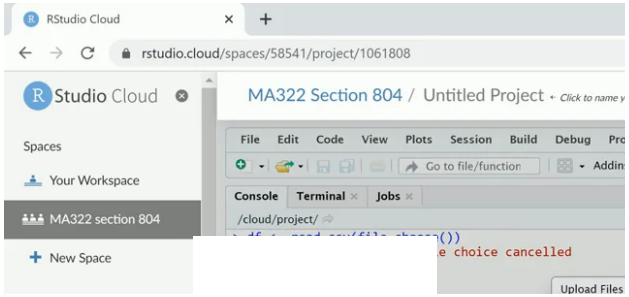
Object type	**Description**
Vector	1-dimensional collection of elements of one type, usually character, logical, inte numeric
Matrix	multi-dimensional collection of elements of one type (specify num rows and num

List	multi-dimensional collection of elements that can be multiple data types
Data frame	special type of list where every element has the same length ("rectangular" list)
Factors	data objects used to categorize and order repeated elements in a vector or list

This is important because different functions you can use will require specific data types as input. If you enter an is not of the expected data type, you will get an error.

Reading data from files (tables)

We will need to upload data tables stored in files such as count matrices or sample metadata that are saved in .c so it is important to know some functions in R to import files. The video below will walk you through importing file Interactive App.



Video. RStudio Upload.

This video will show you how to upload datasets into RStudio Server.

Note: The narrator uses RStudio Cloud, but like RStudio Server, the Interactive App maintains an interface that is consistent with RStudio as a desktop application.

View Transcript (https://canvas.asu.edu/courses/122165/files/54792245?wrap=1) **(https://canvas.asu.edu/courses/122165/files/54792245/download?download_frd=1)** .

Table. Common dataset formatting types

Extension	Name	Description
.csv	Comma separated value	Spreadsheet data (rows and columns) separated by a com
.tsv	Tab separated value	Spreadsheet data (rows and columns) separated by a tab

You can also import files manually by keying in the correct command. This is often the preferred method for serio keep their hands on the keyboard to save time.

In the video the narrator needed to upload the file into the server first, so let's begin by creating a simple text doc .csv file. You can copy and paste this code into a text editor and then save it as a sample .csv file (let's call it 2-2 your preferred location on your local machine (you can delete this small file after this exercise). Note: the comma value indicate a new column; if they had tabs between them, it would be a .tsv file.

```
"","X.1","X","X.3","ALT"
"1",1,1,1,"G"
"2",2,2,2,"T"
"3",3,3,3,"T"
"4",4,4,4,"CTTTTTTTT"
"5",5,5,5,"CCGCGC"
"6",6,6,6,"T"
"7",7,7,7,"A"
"8",8,8,8,"A"
```

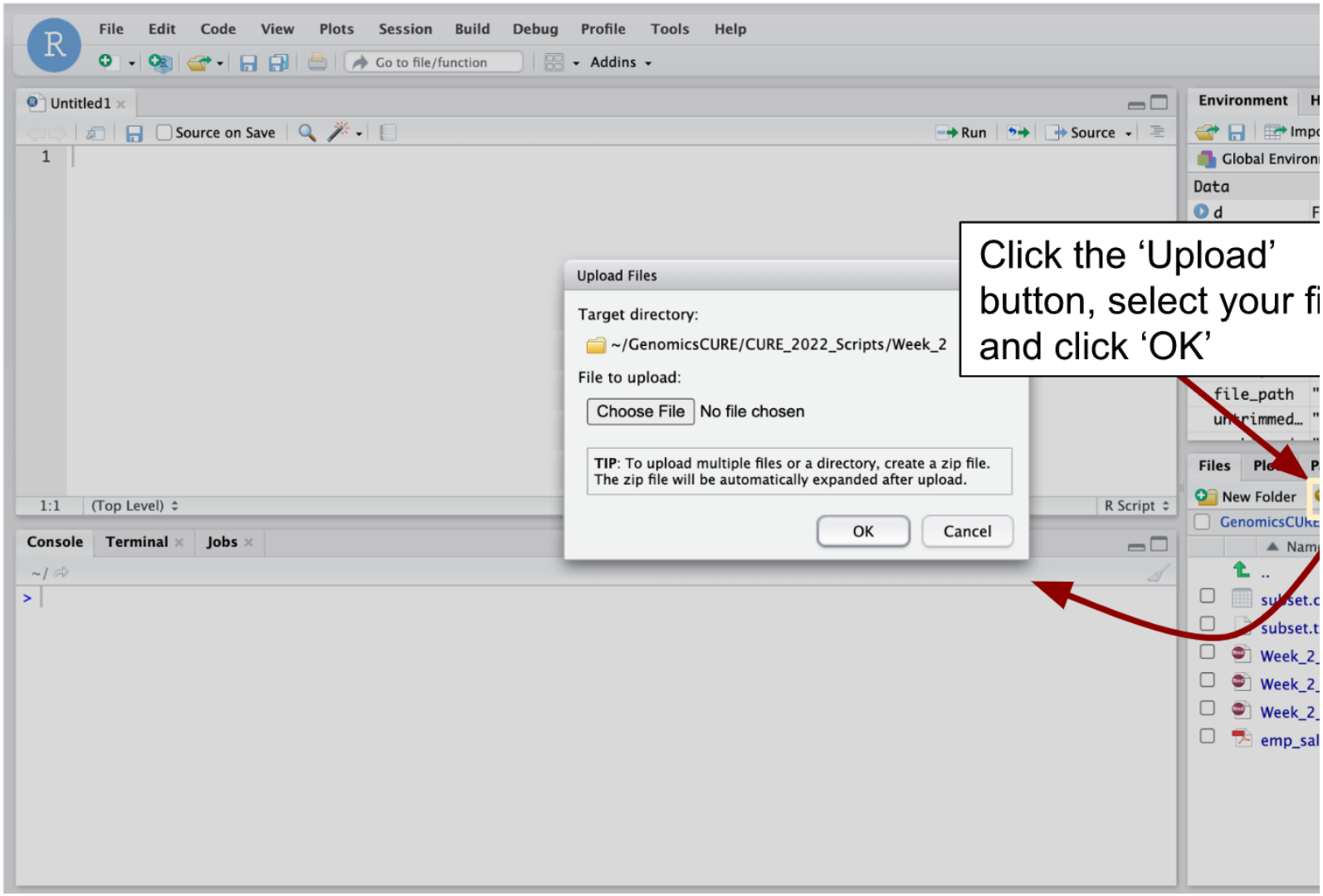


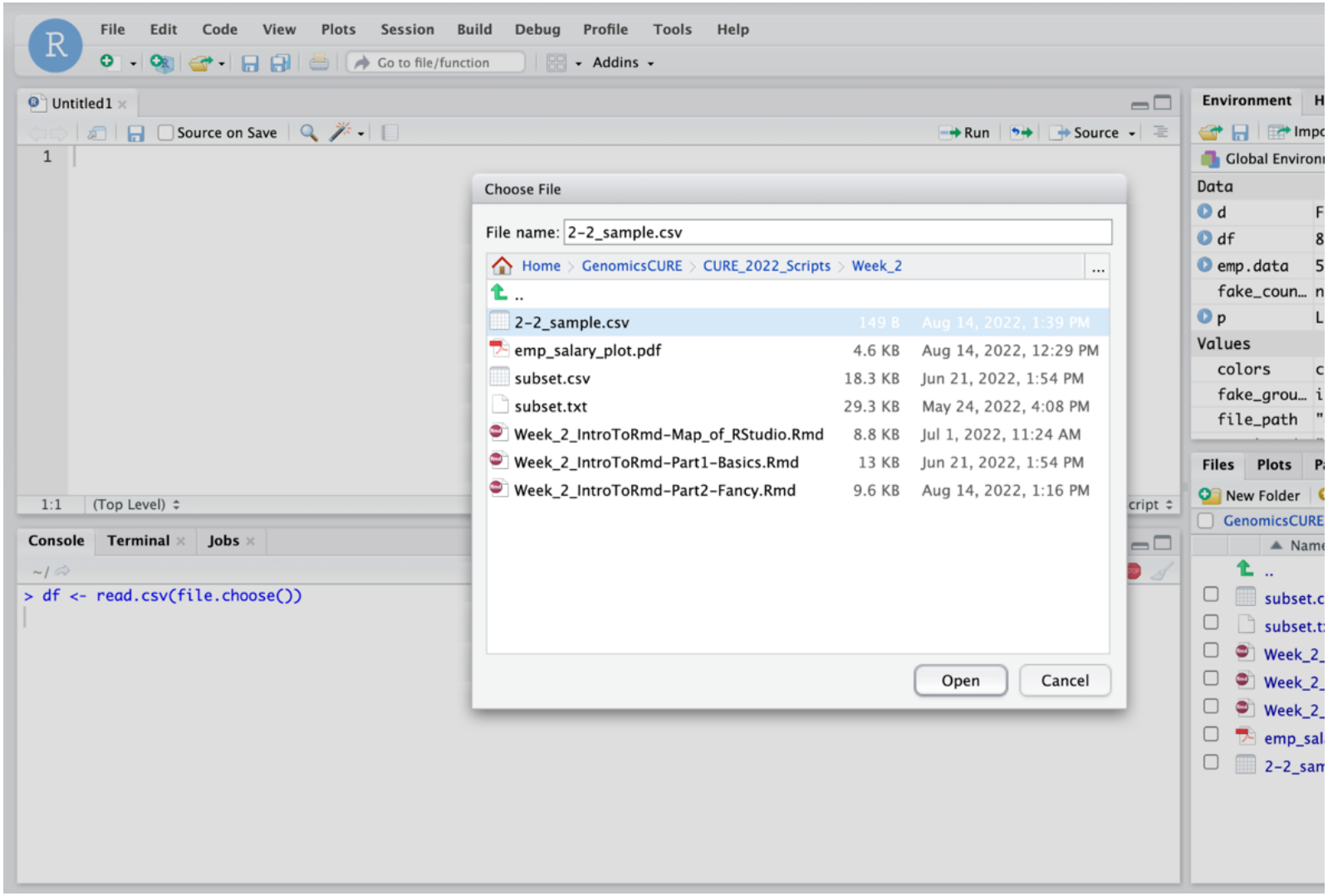
Figure. Importing a dataset from your local machine to RStudio Server. Click the Upload button, select the file should now see the .csv file listed in the Files tab.

You can also import datasets from other directories in the server. To import your data into the data frame `df` from the pathname in quotations.

```
df <- read.csv("/home/<ASUrite_id>/direct/path/to/file.csv")
```

If you do not know the exact path, you can also use the file.choose() function for a graphical interface to locate yo
server. Be sure to leave the braces empty for the pop up, search for the file, select it, and double-click the file or c

```
df <- read.csv(file.choose())
```



You'll know it has been imported by seeing `df` listed in the environment and can expand (right) or open the file (le

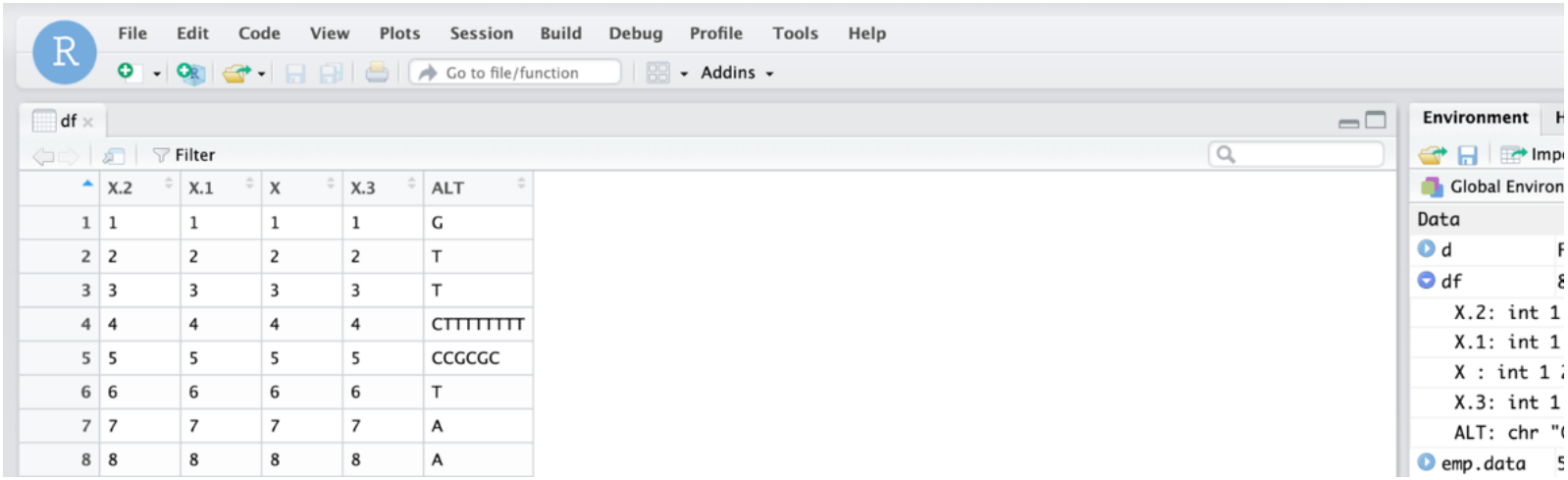


Figure. Data from uploaded and imported .csv file.

Installing and loading packages

Throughout this course we will be using several packages to answer our research question. Packages are simply functions written by other programmers. Large packages with sophisticated functions that were developed with a expertise in a specific area are typically debuted in publications and the best ones are taken up into large reposit Bioconductor. RStudio makes it easier to connect into those repositories to install (download) those packages, a use the functions inside them in your code.

When working with packages, there are two aspects to know about: installing and loading. Installing a package m code for all the functions into your local storage space. This is done with the `install.packages` command or functions like Bioconductor's `BiocManager.install`. This can also be done with the Packages tab in RStudi resources below for more on that). There are so many packages available and not all of them work well together dependencies and not all of them work equally well, so we only install packages that we are interested in using a as best we can before we use them with our own data. We do them one or a few at a time so that we can make as expected before proceeding.

After a package is installed, you can load or activate it for use in your local coding environment. R uses the functi indicate which packages you want to use and which functions should be available. In the template code we prov will see a chunk that checks to see if the packages we need for that Rmd are installed, installs them if they are nc use.

In our case, we will be using RStudio as an instance from Agave. When you install packages, they will be stored directory (default path is something like `/home/<ASUrite_id>/R/x86_64-pc-linux-gnu-library/4.1/` check it out). Once a package is installed, you don't have to install it every time you use it; instead, just use the `l` the name of the package whenever you want to use it in your code.

In all of our template code, we include a chunk at the end that lists all the packages that are loaded when the coc modules, we will talk about how this information can help us to keep track of what exact steps and functions we u and ensure reproducibility when doing that analysis in future studies. Differences in output may be explained by s packages listed as dependencies for other packages.

Activity 2: Using the basic functions of R in action

Time for you to practice using basic data types in R, installing packages, and loading libraries. In the same Week already been working in, you will find two additional Rmds you'll need to run. One builds upon the other, so it is ir all the content and code then run them in the order we have presented them in.

Open this Rmd in RStudio:

```
/home/<ASUrite_id>/CURE_2022_Scripts/Week_2/Week_2_IntroToRmd-Part1_Basics.Rmd
```

Go line by line, running each line of code and exploring around to see what is happening. This Rmd will show you variables, how to view the contents of variables, and how to write out results to files. The code is heavily comme what is being done. If you aren't certain what anything is doing, please ask on Slack so we can help you, as the more complicated moving forward and we want you to have a good grasp on the basics.

Activity 3: Breaking down complex data types and functions

After you have run `Week_2_IntroToRmd-Map_of_RStudio.Rmd` and `Week_2_IntroToRmd-Part1-Basic` Week_2 subdirectory, we would like you to run another Rmd we have prepared that will break down some more c will be using in the rest of the modules. We are using simple examples to help you understand some of the spec complicated functions so that you are not confused when we are using them to process our real data.

Open this Rmd in RStudio:

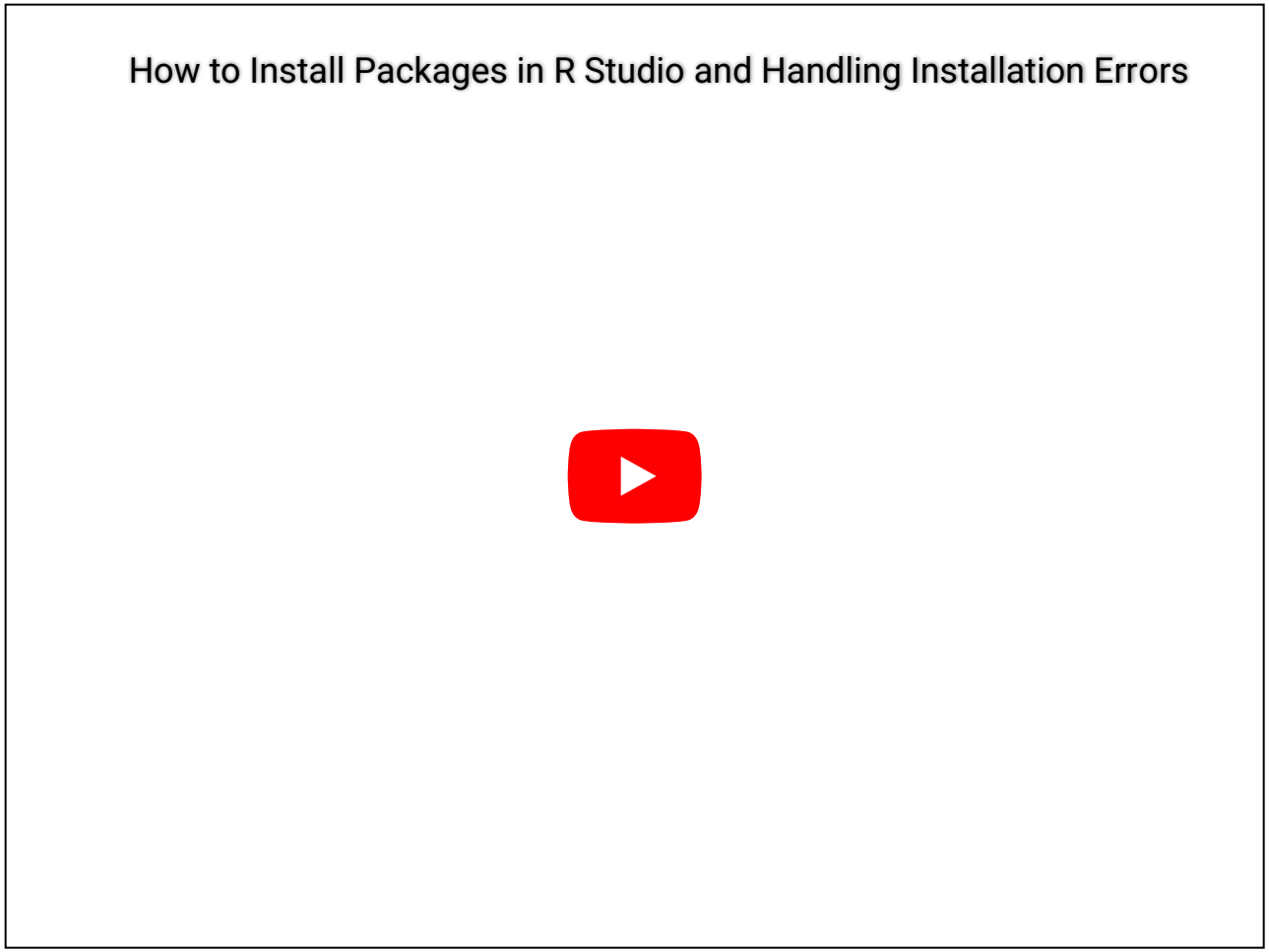
```
/home/<ASUrite_id>/CURE_2022_Scripts/Week_2/Week_2_IntroToRmd-Part2-Fancy.Rmd
```

In the next module, you will be using packages to perform differential gene expression analysis (`edgeR, limma` use a specialized data type called DGEList to store expression data, sample characteristics, normalization factor specialized data type is a combination of basic data types that is standardized which makes it easier to write func expression data. The Rmd will show you how data is filled into this specialized data type, how you can access the objects, and how you can pass DGEList objects to functions to get specific outputs.

The second part of this Rmd will be to show you how to use the `ggplot` function, a complex and powerful functio function is meant to be used in a plug-and-play fashion to try out multiple types of data visualizations (more on th and customize aesthetic features so you get figures that look the way you like.

Module 2.2 Additional Resources

- How to install packages with the Packages tab in RStudio and how to troubleshoot installation errors



- **More details about the specifics of what Rmd supports (https://rmarkdown.rstudio.com/lesson-1.html)**
- **Modern Dive: Getting Started with R (chapters 1.1-1.3) (https://moderndive.com/1-getting-started.html)**
- **Advanced R Style Guide (http://adv-r.had.co.nz/Style.html)**
- **RStudio Cheatsheets (https://www.rstudio.com/resources/cheatsheets/)**
- **Base R cheat sheet (https://github.com/rstudio/cheatsheets/blob/main/base-r.pdf)**
- **The basics of Rmd (each lesson from introduction to cheatsheet ~45mins) (https://rmarkdown.rstudio**
- **DGEList (https://rdrr.io/bioc/edgeR/man/DGEList.html)**