**Week 1 – Introduction**

This week we will focus on the following:

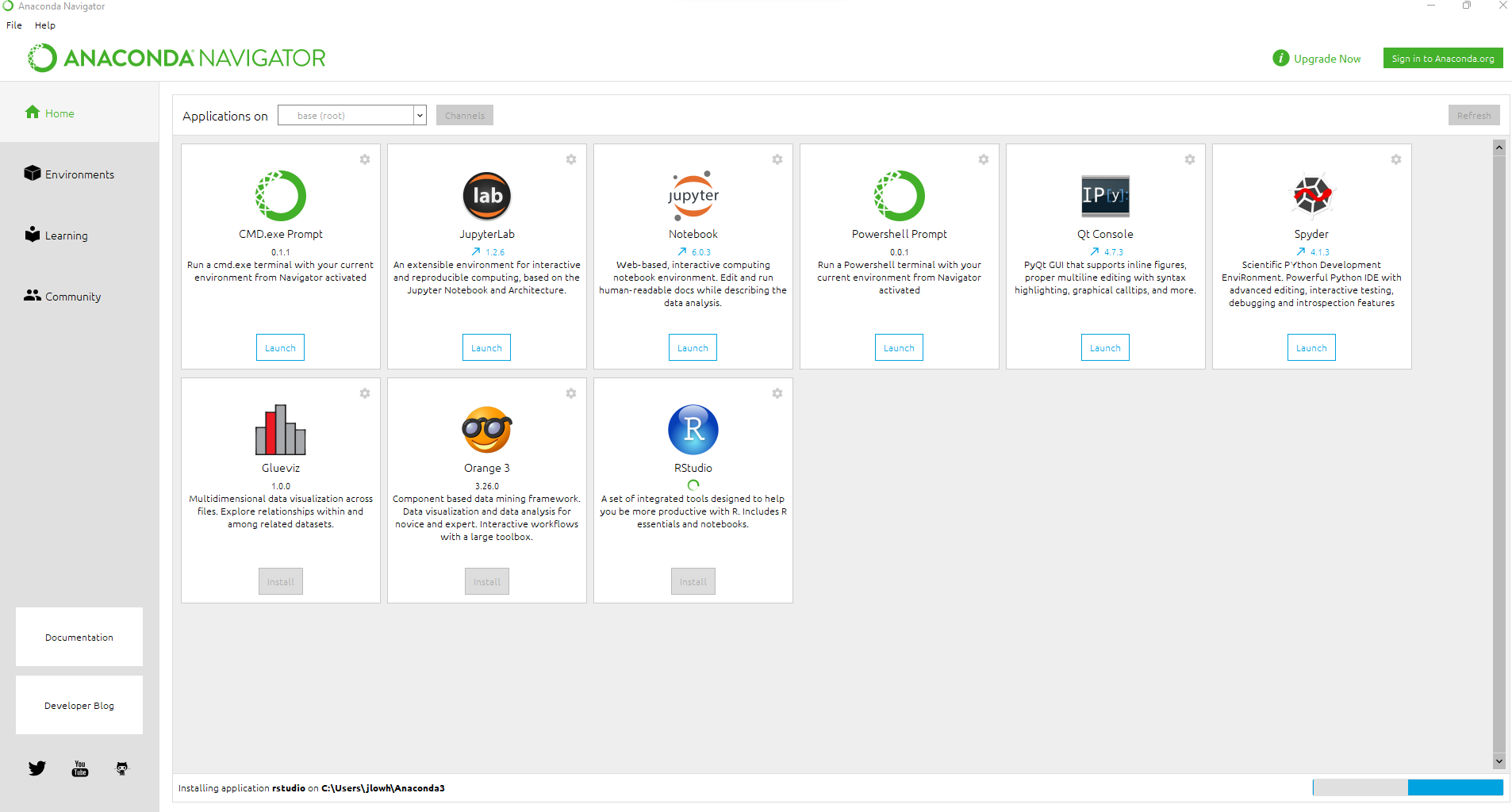
1. Installing R & RStudio
2. Version Control & GitHub
3. Working Directories
4. R Markdown
5. R Environments & scoping

**Installing R & RStudio**

In the first four weeks of class we will be learning the R programming language. We will first need to download the language itself and then download the most widely used IDE for R development, RStudio.

There are several ways to do this, but the most efficient for this class will be to install Anaconda which ships with everything we will need for both R and Python. Anaconda can be installed from here: <https://www.anaconda.com/products/distribution>. They provide installers for Windows, MacOS, and Linux based systems so this should work for everyone.

Once you have Anaconda installed you can simply install RStudio which will ship with R (the language itself)



If you wish to do your own installation of R and RStudio you will first need to install R from CRAN. <https://cran.r-project.org/>

And then install RStudio Desktop from: <https://www.rstudio.com/products/rstudio/download/>

**Mac Specific:**

I have seen folks have issues with Mac installations due to not having xcode-select.

To install this prerequisite, you will need to open your Mac terminal and run: xcode-select –install

If you don’t know how to open your terminal, you can do either of the following:

* Click the Launchpad icon  in the Dock, type Terminal in the search field, then click Terminal.
* In the Finder , open the /Applications/Utilities folder, then double-click Terminal.

**Version Control & GitHub**

We will be using GitHub in this course to organize and store all your work. Most professional organizations use GitHub enterprise to version control their software and store their code off-prem. For this reason it’s a good idea to begin learning the Git paradigm and structing your work in organized directory structures.

To begin create an account at: [github](https://github.com/)

Once you have a GitHub account you will need to install GitHub Desktop from here: [GitHub Desktop](https://desktop.github.com/)

**Working Directories**

**R Markdown**

In this course and subsequent courses, you will be required to provide markdown documents knitted to PDF for your assignments. R Markdown is a file format for making dynamic documents with R. An R Markdown document is written in markdown (an easy-to-write plain text format) and contains chunks of embedded R code. To convert R Markdown documents (Knit) to PDF you will need a software packaged called MiKTeX . You can download MiKTex from here: [Getting MiKTeX](https://miktex.org/download).

**R Environments & Scoping**

To begin working in R you will need a basic understanding of the IDE, RStudio and how it is broken out into the four components shown in the below picture.

**Graphical user interface

Description automatically generated**

What are these components? Let’s look at what each is, and how you can use it.

1. When you have a script, Markdown file, or text file, or any other file open in R it is displayed in this panel. Typically, this is where you write your code/scripts. In the screen shot above you see some code I have written to complete a principal components analysis of the mtcars data set. When ran, this code is executed in the second panel.
2. This is the console, or where your code executes. As mentioned previously, when I run the code from panel 1 it will execute the code and print any results that I have asked it to return. Here we see the correlation of the first principal component of our analysis to each of the individual variables in mtcars. There is also a terminal to the local filesystem, although the usage of that is outside of the scope of this class. The variables or objects that we defined in our script can be viewed in our global environment (because we haven’t used any special scoping) in panel 3.
3. Panel 3 includes any objects you have in memory (from executed code) and displays their type, size, and in some cases other special metadata about the objects and classes. The broom looking button with remove ALL objects from your global environment.
4. Panel 4 contains several different options for you to view Files, Plots, Packages, get Help, or use the Viewer. If you are looking to install a package, this is the easiest place to do it. The file browser will give you an easy way to browse your filesystem and the Help tab will allow you to search for functions and determine their usage patterns.

So, what’s an environment and what’s scoping?

An environment **is a collection of objects (functions, variables, etc.)**. An environment is created when we first start the R interpreter (open RStudio and have a running console).

The top-level environment is called the global environment and that is what is shown in panel 3. There can be local environments that are scoped within functions and classes and whose objects are not available in the global environment. Typically, when you write a function the object that is returned is contained in a return statement. All other variables will not be available unless you **assign()** them.

Example 1:

Let’s look at the following example:A screenshot of a computer

Description automatically generated

In the above picture we have declared a function named myf that takes an input, x. We can see on the 3rd panel that this function is available in our global environment, but we also have another value that we defined in our function. The inner function takes whatever input we give to myf, squares it, and assigns it to the global environment naming it Global.res. Notice that innerf is not available in the global environment, but the value that is created with the **assign()** function is. This is a special case as most programming languages will not allow you define functions inside functions.

Example 2:

What is lexical scoping? Lexical scoping is special to R and python and allows you to inherit global variables into your functions. This means you can define a variable outside of a function and use it within the function. In the below example we define the value y and two functions that take x as an input. In the function g(x) we inherit the value y from the global environment and multiply it by x. If you look at panel 2 we set x to be the value of y so 10 \* 10 = 100.

The second example f(x) sets the value of y within the function. The output from g(x) is 100 and since y = 2 we get the result 104. Notice how the value of y did not change in the global environment, it is still 10, even though we set it to 2 in f(x).

A screenshot of a computer

Description automatically generated with medium confidence