A Tour of R and RStudio

Downloading R

Base R and packages

Base R and most R packages come from *cran.r-project.org*.

- base R comes with basic data management, analysis and visualization tools

Packages are after-market installations. You **download them once, then call each time you need to use them.**
Let's install and call the `tidyverse` package, which we will be using later in the seminar. This is a highly-used set of tools for data manipulation, analysis and visualization.

Base R and packages

Base R and most R packages come from *cran.r-project.org*.

- base R comes with basic data management, analysis and visualization tools
- On top of that R comes with a universe of packages (more than 11,000 on CRAN!)

Packages are after-market installations. You **download them once, then call each time you need to use them.**
Let's install and call the `tidyverse` package, which we will be using later in the seminar. This is a highly-used set of tools for data manipulation, analysis and visualization.

First, install, the packages with `install.packages`. **Do this once.** Type this into the R console and hit Enter.

```
install.packages("tidyverse")
```

You will be asked which mirror to download from. Select a location near you!

To load the package to your workspace, call it using `library`. **Do this each session you want to use the package.**

```
library(tidyverse)
```

From time to time, remember to check for updates to the packages you are running with 'update.packages()'.

This is really the only time we are going to use R on its own. For the rest of the lesson, we will be working from RStudio.

RStudio

- RStudio is a popular integrated development environment (IDE) that run on top of Base R.

RStudio

- RStudio is a popular integrated development environment (IDE) that run on top of Base R.
- download for FREE at rstudio.com

RStudio

- RStudio is a popular integrated development environment (IDE) that run on top of Base R.
- download for FREE at rstudio.com
- **You still need to download Base R from CRAN!**

RStudio orientation

When you open RStudio for the first time you will see three panes. After writing your first script, you'll see four.

These are, starting clockwise from upper-left:

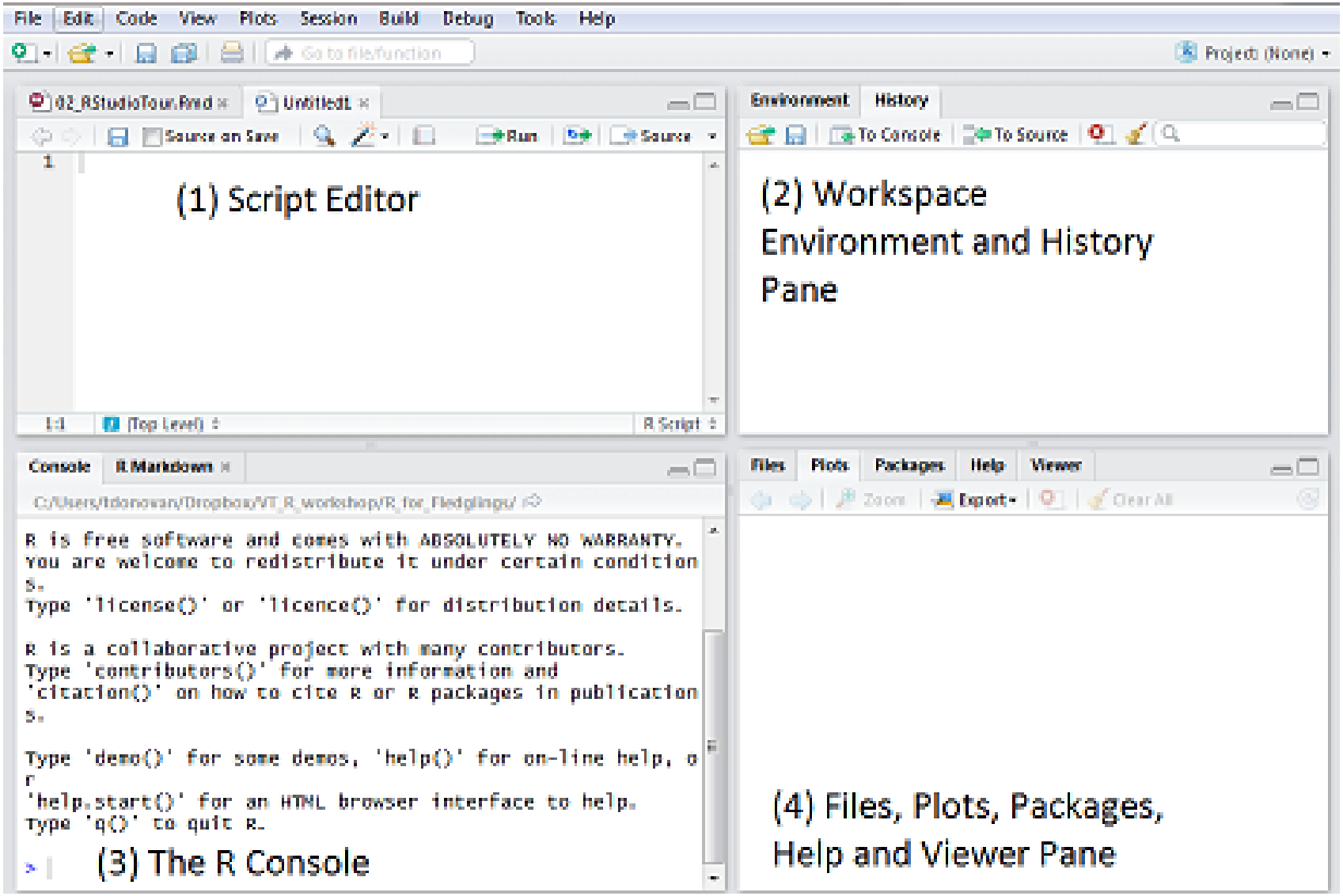


Figure 1: credit: "R for Fledglings," Therese M. Donovan.

These are, starting clockwise from upper-left:

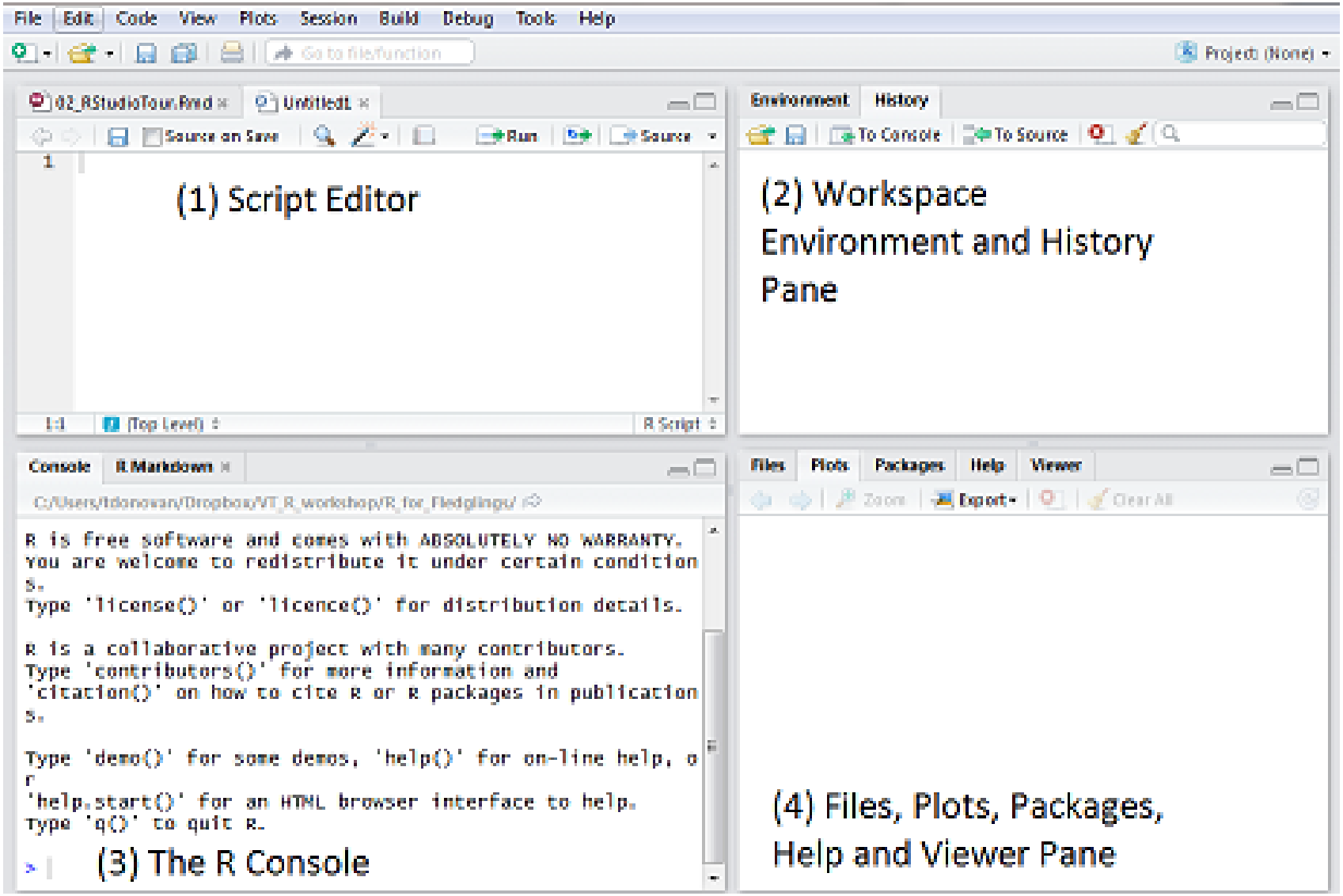


Figure 1: credit: "R for Fledglings," Therese M. Donovan.

These are, starting clockwise from upper-left:

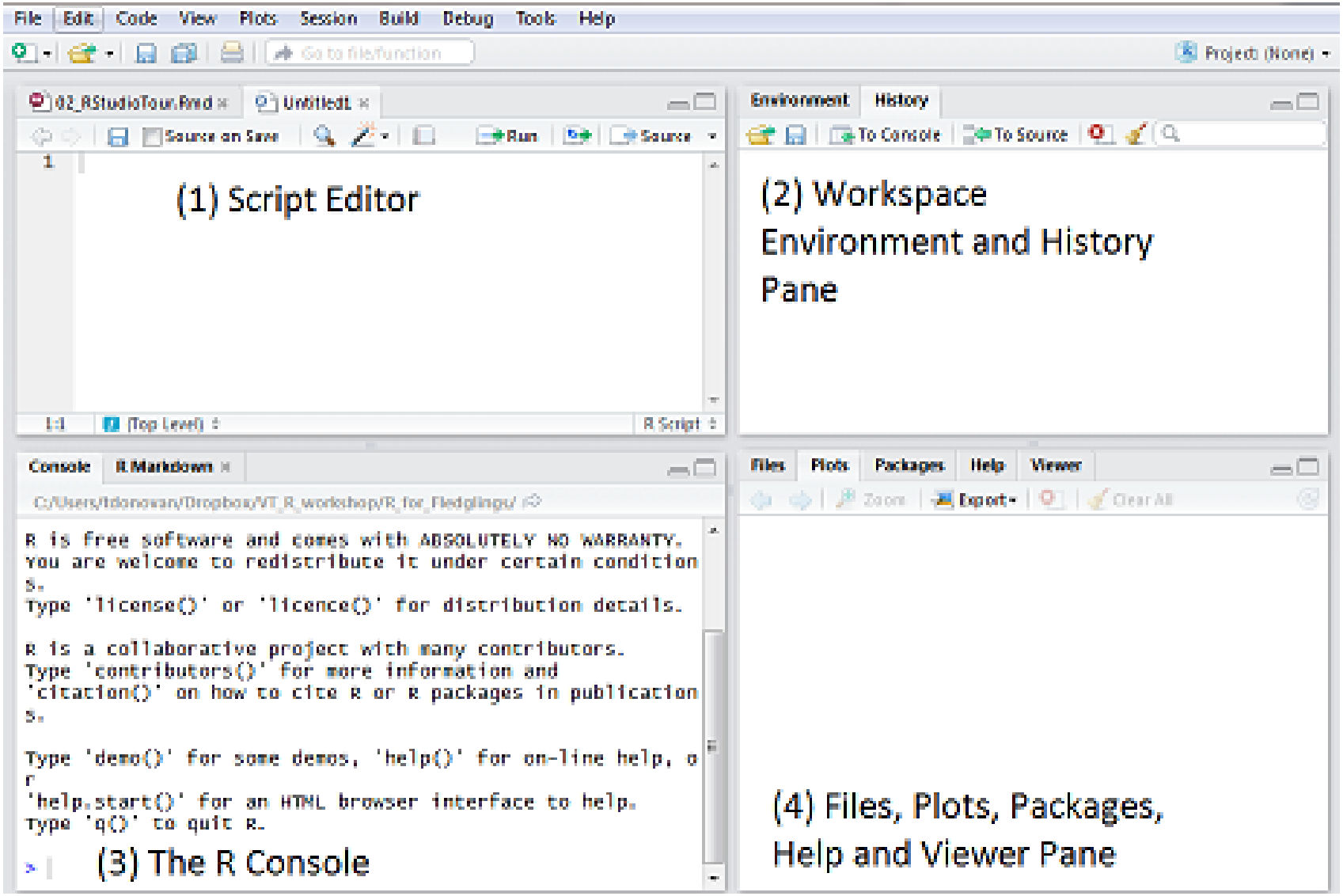


Figure 1: credit: "R for Fledglings," Therese M. Donovan.

These are, starting clockwise from upper-left:

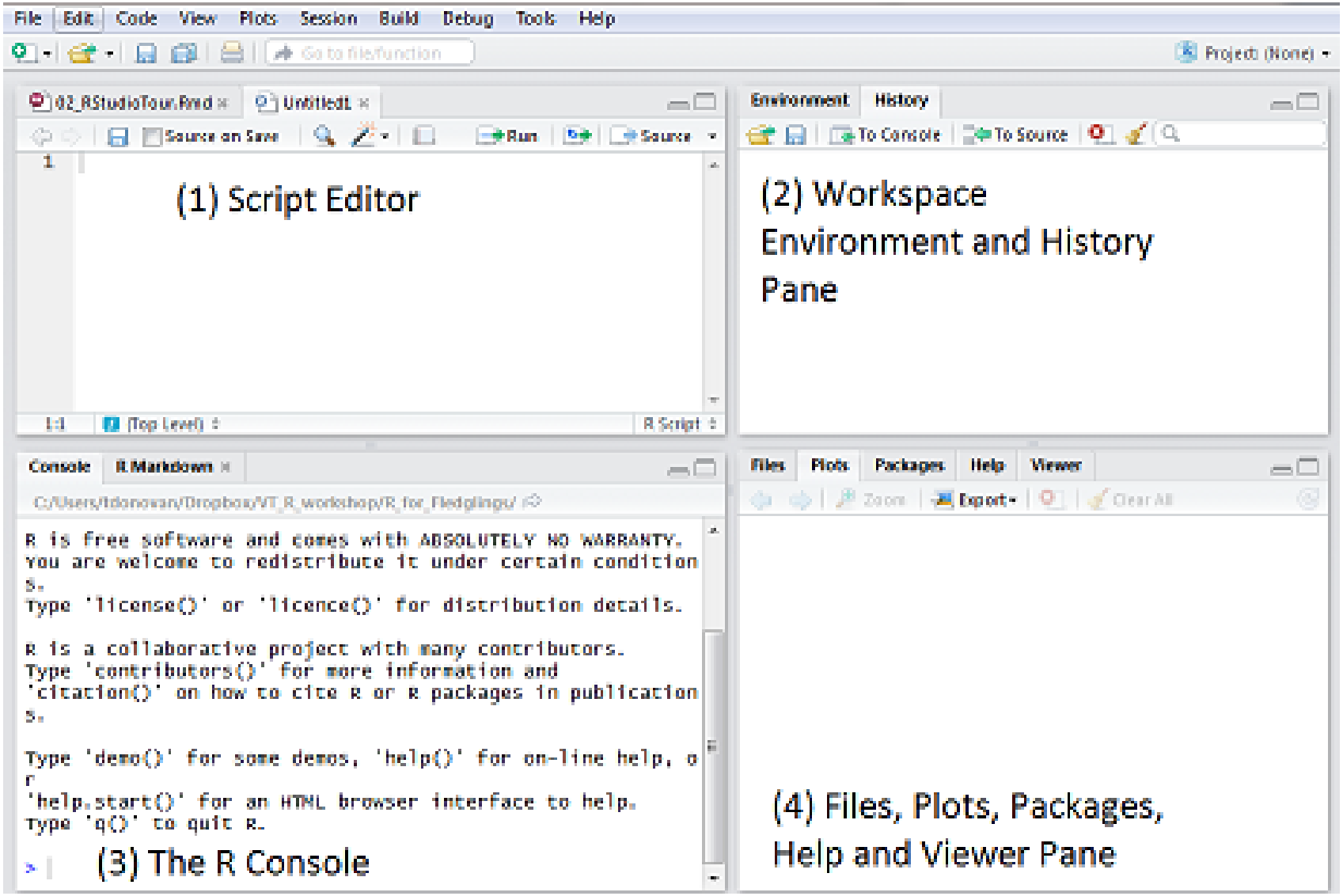


Figure 1: credit: "R for Fledglings," Therese M. Donovan.

We will now walk through the various panes of RStudio which will begin the introduction to R programming.

R Console

This is where commands are submitted to R to execute.

An example... And a caveat.

Type your commands after the prompt symbol, which looks like >.
You should see a vertical, blinking bar to the right of the prompt.
We will start coding here.

In general, there are two types of operations:

1. run a function

We'll be doing the first next.

In general, there are two types of operations:

1. run a function
2. create or modify an object

We'll be doing the first next.

Now it's your turn... in the console type the following:

R can be used as a fancy calculator. Type this directly into the console.

```
1+1
## [1] 2
5*4
## [1] 20
2^3
## [1] 8
```

R follows the order of operations and uses scientific notation:

```
2+3*4/(5+3)*15/2^2+3*4^2
## [1] 55.625
5e4
## [1] 50000
```

Testing Conditions

R also allows you to test conditions of equality among values: + Equality: == (**NOT = - this is important!**) + Not equal: != + Greater / Less than: > or < + Greater / Less than or equal to: >= or <=

```
#does 5 equal 4?
5 == 4
## [1] FALSE
```

```r
#is 10 less than or equal to 11?
10<=11
## [1] TRUE
```

```r
#is the sqrt of 36 equal to 6 squared?
sqrt(36) != 6
## [1] FALSE
```

Now let's make this a little more interesting, by using a function.

```
sqrt(36)
## [1] 6
```

You've just passed the number 36 through the `sqrt` function, just as you might would on a scientific calculator.

Now let's try `Sqrt(10)`:

```
Sqrt(10)
## Error in Sqrt(10): could not find function "Sqrt"
```

Remember, it's the `sqrt` function, NOT the `Sqrt` function.

Moral of the story: R is case sensitive and all-around finnicky!

Editor Pane (upper left pane)

Normally you would not interact directly with the Console. Instead, you will write your code into the script file and then send the code to the console for it to execute.

The script file is your long-term record of the code you ran. The console only keeps a short-term record of it.

In RStudio, choose **File** | **New File** | **R Script** or `Ctrl+N` to open a new R script.

You will see a blank document in the upper left panel of RStudio.

Commenting Code

Type the following lines of code into your R Script.

```
#get the square root of 36
sqrt(36)
## [1] 6
```

Lines of code that are preceded by the# symbol are comments.

Comments are **not executed in R.** Rather, they are **notes** that help the coder understand and remember what the program should do.

Running Code

To send code to R, place your cursor anywhere on the line, then press the Run button. You can also use `Ctrl+Enter` in RStudio.

```
#get the square root of 36
sqrt(36)
## [1] 6
```

Notice that after executing the line your cursor drops to the next line of code.

You can also highlight multiple lines of code at once and press Run.

```
#get the square root of 36
sqrt(36)
## [1] 6
#get the square root of 100
sqrt(100)
## [1] 10
```

Saving your script

To save your R script, go to **File** | **Save** or type `Ctrl+S` in RStudio.

The Files, Plots, Package Help Pane (lower right)

The **help** tab in this window is useful for beginners. It returns a help file when you use `help()` in your command prompt:

```
#call the helpfile for the sqrt funtion
help(sqrt)
## starting httpd help server ... done
```

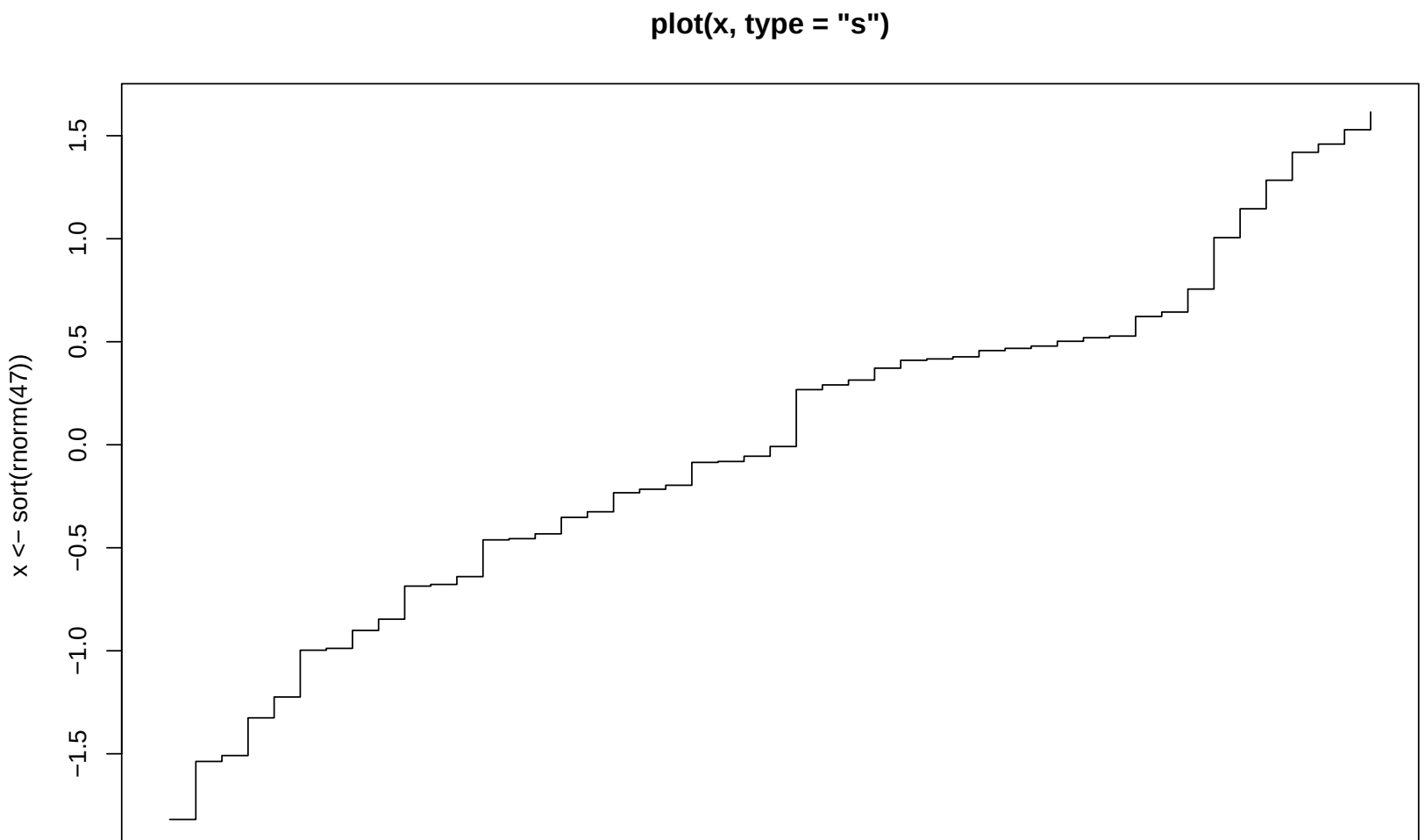
The Plots Tab

This tab will hold any plots you create in your R section.

We will create a plot later, but for now let's go ahead and use the `help()` command on `plot()` itself.

Each helpfile in R contains a section called **Examples**, and there you can copy code from the helpfile and run it in the console to see an example of the function in action.

```
#plot example of plot function
plot(x <- sort(rnorm(47)), type = "s", main = "plot(x, type
```



The Environment and History Pane (upper right)

The History Tab

On the History tab you will see a history of all the commands you have sent to R console in the session.

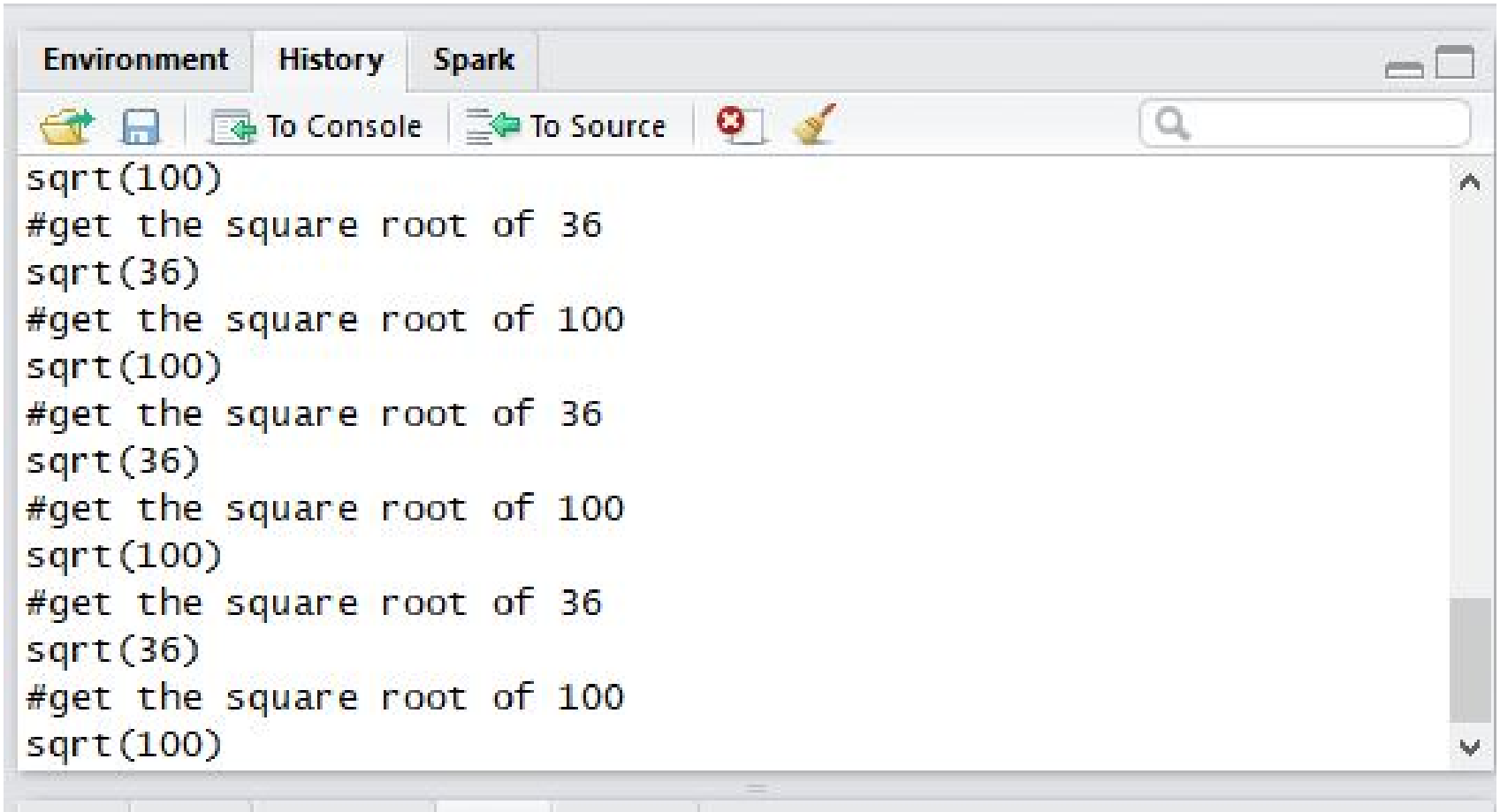


Figure 2:

The Environment Tab

Now let's click on the Environment tab.

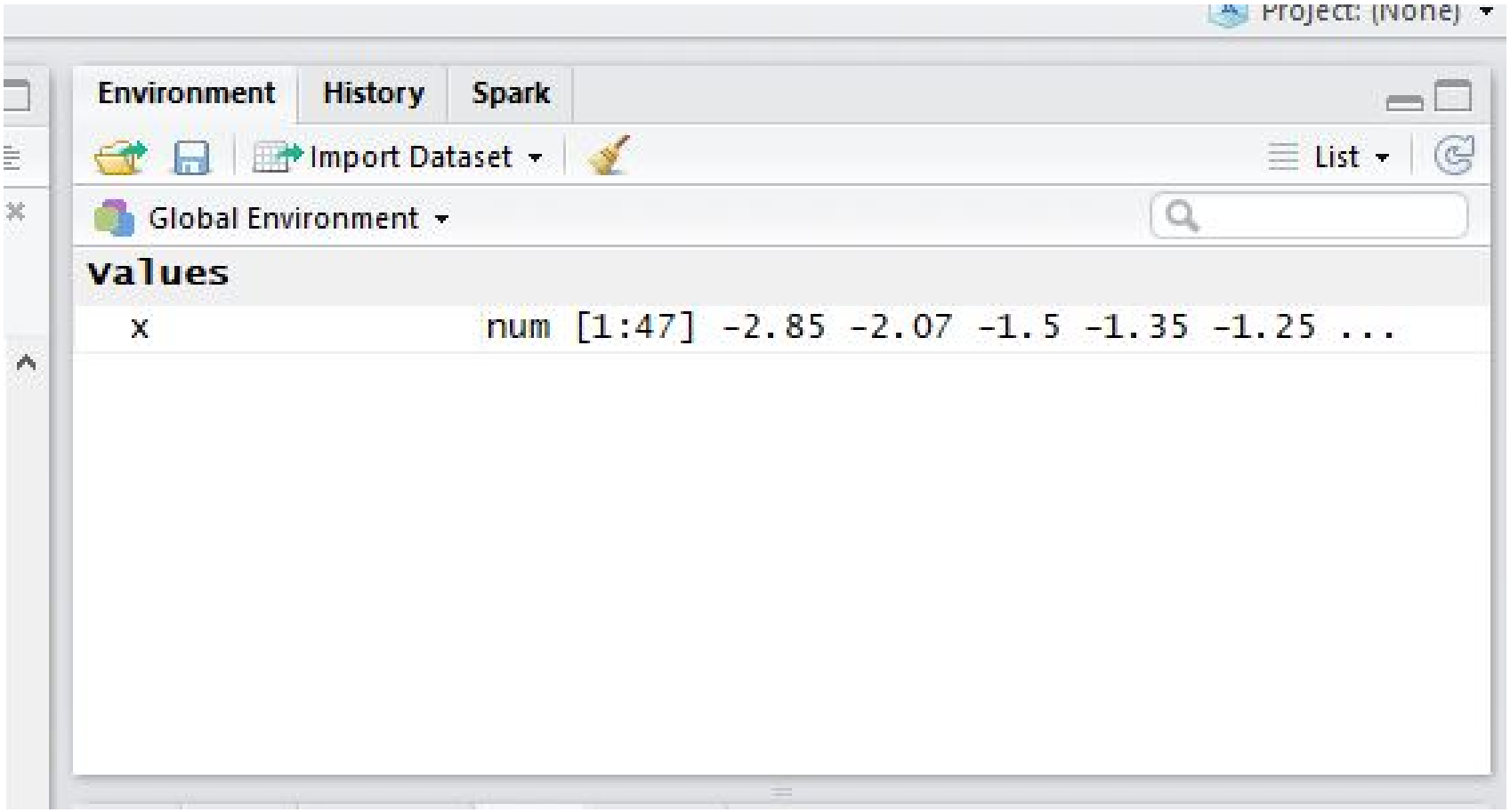


Figure 3:

You will see that R has created an object called "x". Where did this

Earlier we used `sqrt` to calculate the square root of a number. Wouldn't it be nice to save the result to use for later? Let's do that by creating an object, `result`. Type the following into your script:

```
#use sqrt function to calculate sqrt of 10 and store resul
result<-sqrt(10)
```

You should see that the object appears in the section on the Environment tab labeled "Global Environment." A major part of your work involves creating objects.

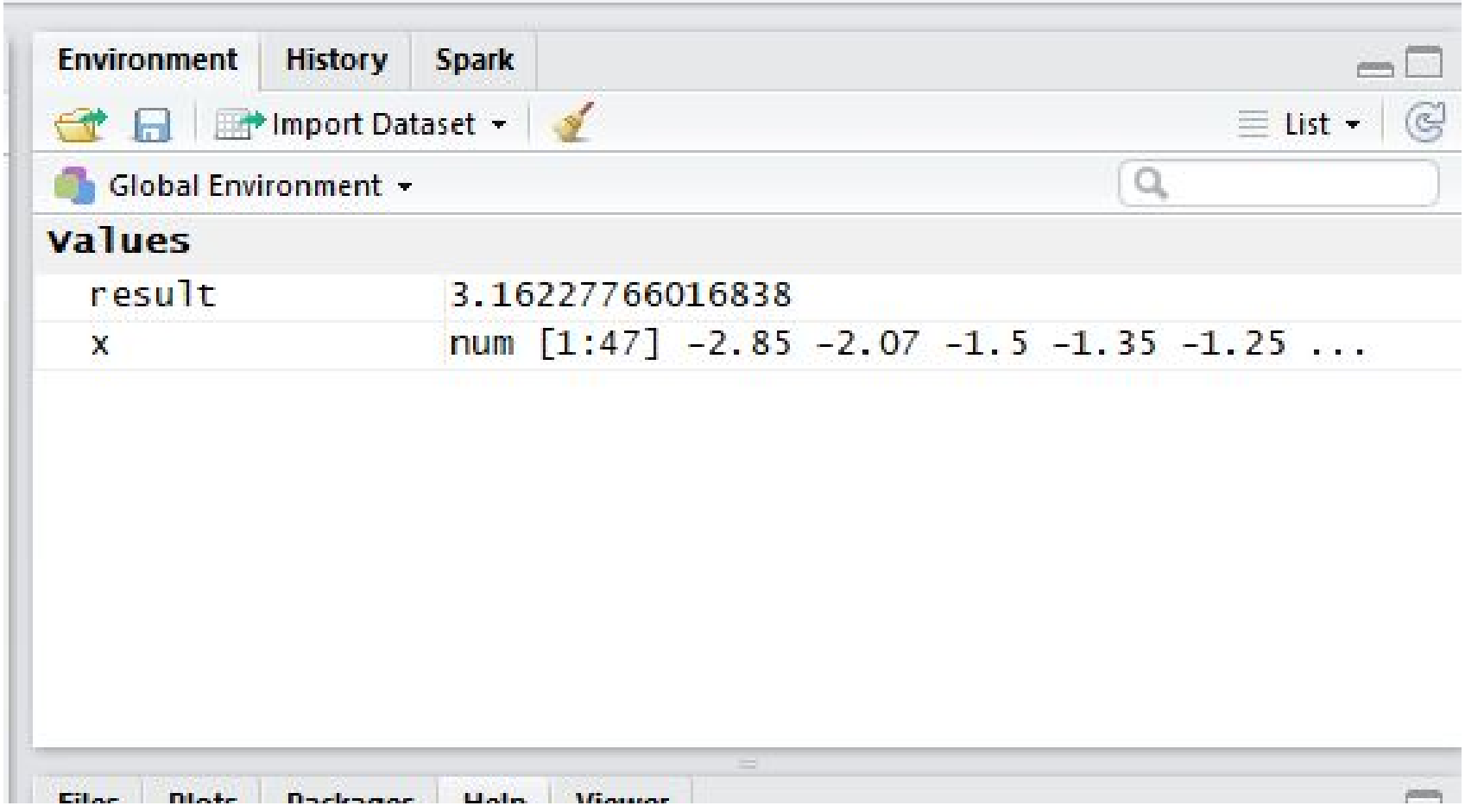


Figure 4:

The objects stored in the Environment and History together make up the **workspace environment.**

When you close R, R brings up a window that asks whether you'd like to save the workspace image (a snapshot of the current contents in the environment).

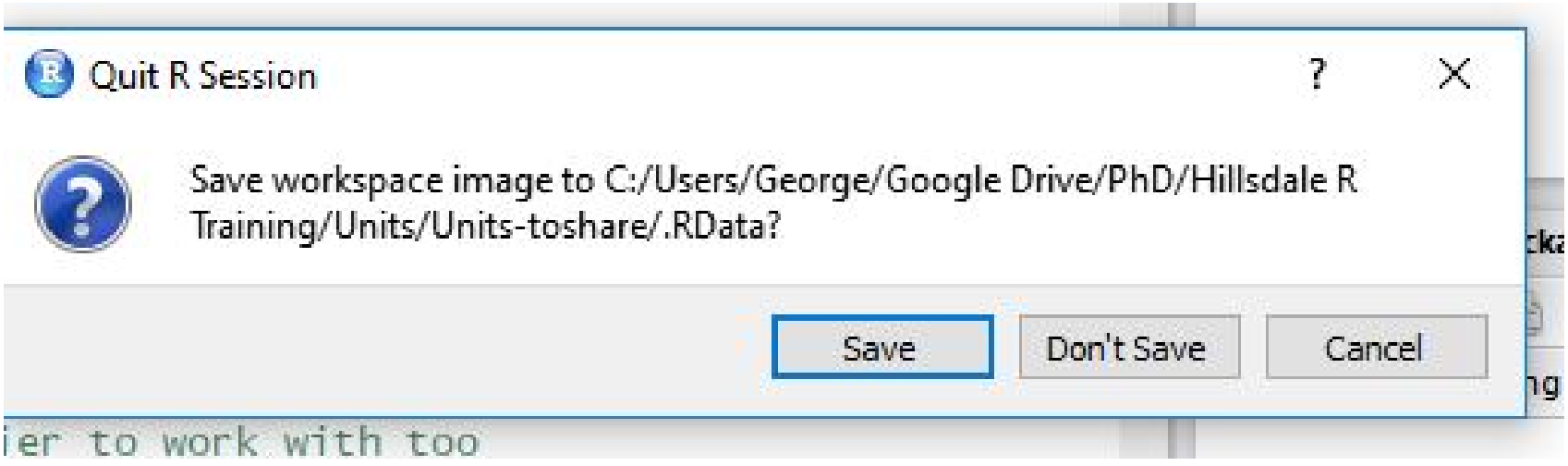


Figure 5:

As a good practice, rarely save the workspace if the code can be quickly re-run. It is easy to accumulate an unmanageable amount of data and objects.

Some other things to note about the envrionment in RStudio:

1. Clear the environment by clicking the "clear" button (with the broom). You can also write `rm(list=ls())` in your script.

Some other things to note about the envrionment in RStudio:

1. Clear the environment by clicking the "clear" button (with the broom). You can also write `rm(list=ls())` in your script.
2. View the environment in your console by typing `ls()`.

Some other things to note about the envrionment in RStudio:

1. Clear the environment by clicking the "clear" button (with the broom). You can also write `rm(list=ls())` in your script.
2. View the environment in your console by typing `ls()`.
3. Remove objects in the environment by using the `rm` function. For exampe, `rm(x)` will remove `x` from your environment.

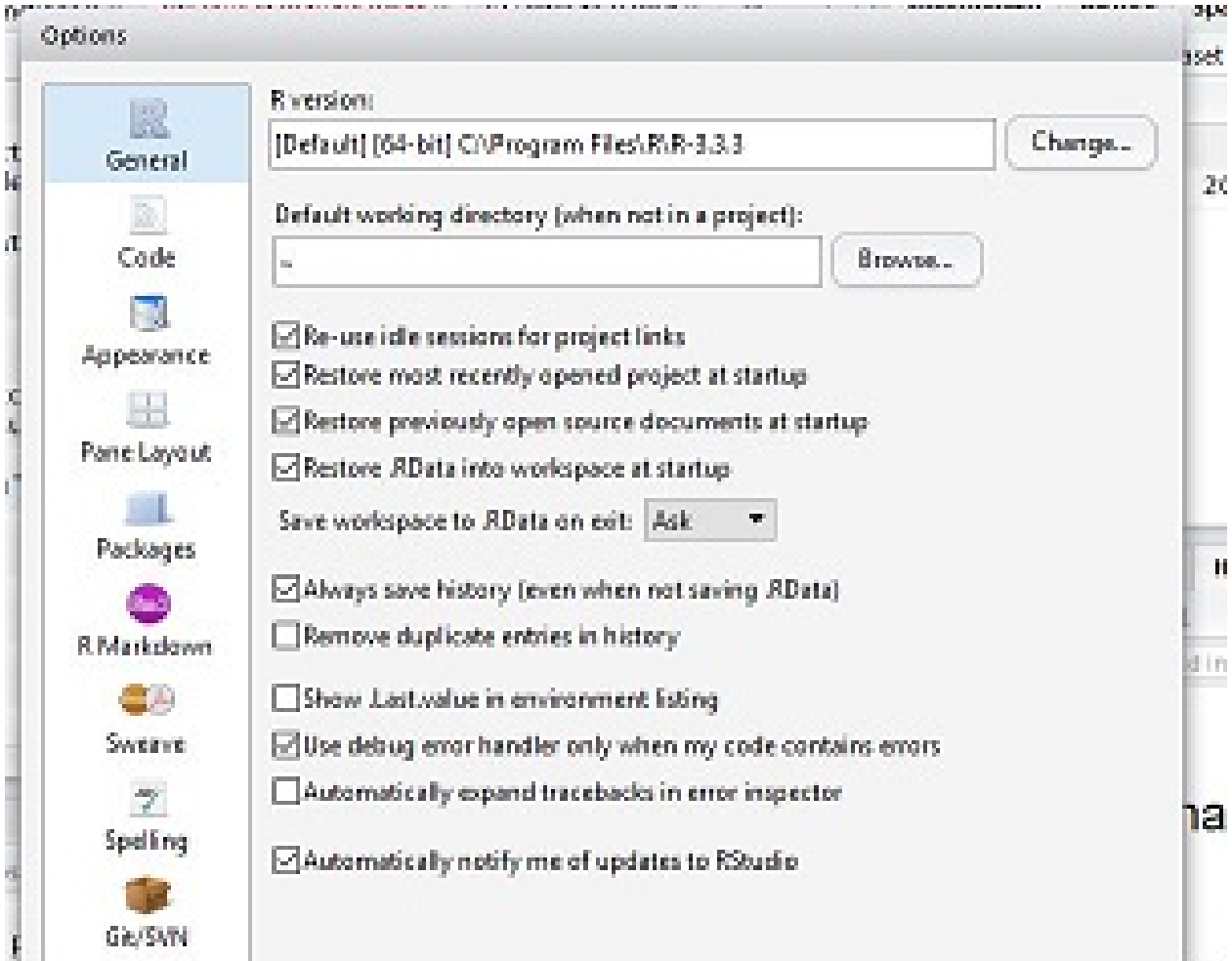
Some other things to note about the envrionment in RStudio:

1. Clear the environment by clicking the "clear" button (with the broom). You can also write `rm(list=ls())` in your script.
2. View the environment in your console by typing `ls()`.
3. Remove objects in the environment by using the `rm` function. For exampe, `rm(x)` will remove `x` from your environment.
4. You can save individual items of the workspace with the `save` function. For example, `save(x)` will save individual objects in the workspace.

RStudio Settings

Now that you have a brief introduction to RStudio's panes let's look at the program's options where you can control settings such as font size, etc.

Go to **Tools** | **Global Options** and you'll see the following box:



There are some cool things here. . . .

- General - look but don't touch. . .

There are some cool things here. . . .

- General - look but don't touch. . .
- Code editing - make sure the soft wrap option is selected; this will ensure that lines of code are automatically wrapped so you don't have to press return when you get to the end of a line.

There are some cool things here. . . .

- General - look but don't touch. . .
- Code editing - make sure the soft wrap option is selected; this will ensure that lines of code are automatically wrapped so you don't have to press return when you get to the end of a line.
- Appearance (*this one is fun*)- choose a color scheme that works for you.

There are some cool things here. . . .

- General - look but don't touch. . .
- Code editing - make sure the soft wrap option is selected; this will ensure that lines of code are automatically wrapped so you don't have to press return when you get to the end of a line.
- Appearance (*this one is fun*)- choose a color scheme that works for you.
- Pane layout - use the default layout for the duration of this primer.

There are some cool things here. . . .

- General - look but don't touch. . .
- Code editing - make sure the soft wrap option is selected; this will ensure that lines of code are automatically wrapped so you don't have to press return when you get to the end of a line.
- Appearance (*this one is fun*)- choose a color scheme that works for you.
- Pane layout - use the default layout for the duration of this primer.
- Packages - make sure "enable packages pane"" is checked. We'll cover packages in the next chapter.

There are some cool things here. . . .

- General - look but don't touch. . .
- Code editing - make sure the soft wrap option is selected; this will ensure that lines of code are automatically wrapped so you don't have to press return when you get to the end of a line.
- Appearance (*this one is fun*)- choose a color scheme that works for you.
- Pane layout - use the default layout for the duration of this primer.
- Packages - make sure "enable packages pane"" is checked. We'll cover packages in the next chapter.
- Spelling - allows a spell-check of your files.

Questions?