To repeat a line of code, simply press the Up Arrow key and hit Enter again. All previous commands are saved and can be accessed by repeatedly using the Up and Down Arrow keys to cycle through them.

Interrupting a command is done with Esc in Windows and Mac and Ctrl-C in Linux.

Often when working on a large analysis it is good to have a file of the code used. Until recently, the most common way to handle this was to use a text editor\footnote{This means a programming text editor as opposed to a word processor such as Microsoft Word. A text editor preserves the structure of the text whereas word processors may add formatting that makes it unsuitable for insertion into the console.} such as TextPad or UltraEdit to write code and then copy and paste it into the R console. While this worked, it was sloppy and led to a lot of switching between programs.

\section*{2.2 RStudio}

While there are a number of IDEs available, the best right now is RStudio, created by a team led by JJ Allaire whose previous products include ColdFusion and Windows Live Writer. It is available for Windows, Mac and Linux and looks identical in all of them. Even more impressive is the RStudio server, which runs an R instance on a Linux server and allows the user to run commands through the standard RStudio interface in a Web browser. It works with any version of R (greater than 2.11.1) including Revolution R from Revolution Analytics. RStudio has so many options that it can be a bit overwhelming. We will cover some of the most useful or frequently used features.

RStudio is highly customizable but the basic interface looks roughly like Figure 2.3. In this case the lower left pane is the R console, which can be used just like the standard R console. The upper left pane takes the place of a text editor but is far more powerful. The upper right pane holds information about the workspace, command history, files in the current folder and Git version control. The lower right pane displays plots, package information and help files.

There are a number of ways to send and execute commands from the editor to the console. To send one line place the cursor at the desired line and press Ctrl+Enter (Command+Enter on Mac). To insert a selection, simply highlight the selection and press Ctrl+Enter. To run an entire file of code, press Ctrl+Shift+S.

When typing code, such as an object name or function name, hitting Tab will autocomplete the code. If more than one object or function matches the letters typed so far, a dialog will pop up giving the matching options as shown in Figure 2.4.

Typing Ctrl+1 moves the cursor to the text editor area and Ctrl+2 moves it to the console. To move to the previous tab in the text editor, press Ctrl+Alt+Left in Windows, Ctrl+PageUp in Linux and Ctrl+Option+Left on Mac. To move to the next tab in the text editor, press Ctrl+Alt+Right in Windows, Ctrl+PageDown in Linux and Ctrl+Option+Right on Mac. For a complete list of shortcuts click Help &gt; Keyboard Shortcuts.
2.2.1 RStudio Projects

A primary feature of RStudio is projects. A project is a collection of files—and possibly data, results and graphs—that are all related to each other. Each package even has its own working directory. This is a great way to keep organized.

The simplest way to start a new project is to click File >> New Project as in Figure 2.5.

---

2. This is different from an R session, which is all the objects and work done in R and kept in memory for the current usage period, which usually resets upon restarting R.
2.2 RStudio

R is a programming language, and it easily handles all the basic operations of statistics, including drawing random numbers, generating random numbers, finding minimums, maximums, means, variances, standard deviations, correlation, and t-tests (the focus of this chapter). R provides numerous functions for making use of them. These generating random numbers and calculating the distribution and quantile.

Three options are available, shown in Figure 2.6: starting a new project in a new directory, associating a project with an existing directory or checking out a project from a version control repository such as Git or SVN. In all three cases a .Rproj file is put into the resulting directory and keeps track of the project.

Figure 2.5 Clicking File >> New Project begins the project creation process.

Figure 2.6 Three options are available to start a new project: a new directory, associating a project with an existing directory or checking out a project from a version control repository.
Choosing to create a new directory brings up a dialog, shown in Figure 2.7, that requests a project name and where to create a new directory.

![Figure 2.7 Dialog to choose the location of a new project directory.](image)

Choosing an existing directory asks for the name of the directory, seen in Figure 2.8.

![Figure 2.8 Dialog to choose an existing directory in which to start a project.](image)
Choosing to use version control (we prefer Git) first asks whether to use Git or SVN as in Figure 2.9.

![Figure 2.9](image)

**Figure 2.9** Here is the option to choose which type of repository to start a new project from.

Selecting Git asks for a repository URL, such as `git@github.com:jaredlander/coefplot.git`, which will then fill in the project directory name, as shown in Figure 2.10. As with creating a new directory, this will ask where to put this new directory.

![Figure 2.10](image)

**Figure 2.10** Enter the URL for a Git repository, as well as the folder where this should be cloned to.
2.2.2 RStudio Tools

RStudio is highly customizable with a lot of options. Most are contained in the Options dialog accessed by clicking Tools >> Options, as seen in Figure 2.11.

First are the General options, shown in Figure 2.12. There is a control for selecting which version of R to use. This is a powerful tool when a computer has a number of versions of R. However, RStudio must be restarted after changing the R version. In the
future, RStudio is slated to offer the ability to set different versions of R for each project. It is also a good idea to not restore or save .RData files on startup and exiting.\(^3\)

The Code Editing options, shown in Figure 2.13, control the way code is entered and displayed in the text editor. It is generally considered good practice to replace tabs with spaces, either two or four. Some hard-core programmers will appreciate vim mode. As of now there is no Emacs mode.

![Options](image)

**Figure 2.13** Options for customizing the code editing pane.

Appearance options, shown in Figure 2.14, change the way code looks, aesthetically. The font, size and color of the background and text can all be customized here.

The Pane Layout options, shown in Figure 2.15, simply rearrange the panes that make up RStudio.

---

3. RData files are a convenient way of saving and sharing R objects and are discussed in Section 6.5.
Figure 2.14 Options for code appearance.

Figure 2.15 These options control the placement of the various panes in RStudio.
The Packages options, shown in Figure 2.16, set options regarding packages, although the most important is the CRAN mirror. While this is changeable from the console, this is the default setting. It is best to pick the mirror that is geographically the closest.

![Options](image)

**Figure 2.16** Options related to packages. The most important is the CRAN mirror selection.

Sweave, Figure 2.17, may be a bit misnamed, as this is where to choose between using Sweave or knitr. Both are used for the generation of PDF documents with knitr also enabling the creation of HTML documents. knitr, detailed in Chapter 23, is by far the better option, although it must be installed first, which is explained in Section 3.1. This is also where the PDF viewer is selected.

RStudio contains a spelling checker for writing \LaTeX{} and Markdown documents (using knitr, preferably), which is controlled from the Spelling options, Figure 2.18. Not much needs to be set here.
Figure 2.17  This is where to choose whether to use Sweave or knitr and select the PDF viewer.

Figure 2.18  These are the options for the spelling check dictionary, which allows language selection and the custom dictionaries.
The last option, Git/SVN, Figure 2.19, indicates where the executables for Git and SVN exist. This needs to be set only once but is necessary for version control.

**Figure 2.19** This is where to set the location of Git and SVN executables so they can be used by RStudio.