The objective of this section is to review the syllabus and to introduce the R environment. If there is remaining time, I'll work through some basic code puzzles that will require you to work in R, but will more likely leave them for you to play with on your own. Today may be a little slow for those of you with substantial experience in R, and for those who actually worked through the Google R Tutorial, but I promise we'll speed up soon.

Download R

The download of R will vary by operating system, but it will begin here in any event:

cran.r-project.org

The online documentation and installer routines are comprehensive. The version you choose will depend on your operating system. It is unlikely that the built-in editor will suffice for anything but the simplest purposes, so I recommend against it. RStudio is a popular cross-platform integrated development environment (IDE) that provides a very user-friendly interface, comes with many convenient features, and I recommend it to almost anyone. For the highly tech-oriented, the Linux distribution is very flexible; and I'd use Emacs with the ESS package for editing. Setting this up can be time consuming, so think about exactly which features you'd be after before going down this route.[1]

Working in R

R is all about packages; these are extensions of the base R code and are the main reason why R is so popular and powerful. In order to download these specific packages, such as the **foreign** package, you'll enter the following commands:

```
install.packages("foreign")
library(foreign)
```

Once **foreign** is loaded, you'll have access to all of its constituent functions, such as the **read.dta** function which will convert any **.dta** file (from STATA) to a **data.frame** [2]. We will do that now, loading into memory the **auto.dta** into a data frame called **mydata**.

```
mydata <- read.dta("auto.dta")
```

We can read the names from the data set; but they aren't much help.

```
names(mydata)

## [1] "V1" "V2" "V3"
```

[1] I have personally tried Emacs-ESS and RStudio on Windows, and RStudio on Linux. Some techie people seem to like Emacs-ESS with Linux, but I haven't come across a good argument for using it personally, even when I'm using Linux. Emacs-ESS on Windows worked fine, but was awful to set up and has some basic behavior that will be unfamiliar to anyone who hasn't used it before. Many of the nice features that are available on Emacs-ESS (debugging, autocompletion, autodisplay documentation) are now available in RStudio, along with more, such as project support, package development support, packrat, and environment/data summaries.

[2] Note that it is also possible to read in **xls**, **xlsx**, comma-delimited, tab-delimited, and many other types of data using similar functions.

We can replace the column headers with more descriptive variable names.

```
names(mydata) <- c("price", "mpg", "weight")
```

To get a sense of the data, list the first six observations:

```
head(mydata)
```

```
##   price mpg weight
## 1  4099  22   2930
## 2  4749  17   3350
## 3  3799  22   2640
## 4  4816  20   3250
## 5  7827  15   4080
## 6  5788  18   3670
```

With the columns appropriately named, we can refer to particular variables within the data set using the unique indexing in R, where data objects tend to be variants of lists and nested lists.

```
head(mydata$mpg)
```

```
## [1] 22 17 22 20 15 18
```

RStudio Tricks

RStudio has many non-obvious convenient features that will help you code in R. This part of the section requires that you have RStudio installed and follow along in the program. You can find binaries for your operating system at

`http://www.rstudio.com/products/rstudio/download/`.

There are many features of RStudio which I consider obvious as any normal user will come across them quickly. These include automatic close bracketing/bracing/parentheses/quotes, the environment summary, and the plots viewer. Here are some you may not have come across:

Auto code completion

This is, by far, my favorite feature of RStudio. Let's say I'd like to install a package, but I can't quite remember how the code goes. Maybe it's `installPackage`, or `install.package` or `install.me.a.package.please?`. With RStudio, I can start typing:

```
inst
```

followed by the `TAB` key. Up comes `install.packages`! Also in the list is `installed.packages`, which isn't what we want. Navigating to your desired function using the arrow keys and hitting `TAB` will select the function you want:

```
install.packages
```

This feature works for both functions and objects. Despite being told not to do this as a young boy learning to program, I now use very long descriptive variable names to save going back and forth to find definitions. For example, I have a variable named `sugarcaneStalkAverageYieldTonPerHa` in a current project and just need to type `sug` and `TAB` when I need to use it.

Auto documentation display

RStudio can also see when you're inside a function call, and want to know the arguments you need to provide.

```
install.packages(
```

and hit `TAB` or `CTRL + SPACE`. Up comes all the potential arguments to this function with their documentation, which can be navigated using the up/down arrows. You can then use the `TAB` or `ENTER` key to input the variable name, along with the `=` sign for input:

```
install.packages(pkgs = "data.table")
```

Recall that the names of the arguments are not usually necessary, but do often aid readability.

File navigation

You want to read in a dataset and you can't exactly remember what it's called, or where it is, or where the current working directory is. Start typing:

```
read.dta("
```

and hit `TAB`. Up comes all the files and folders in the current working directory. You can navigate into subfolders using the `TAB` key, and you can even navigate upwards using "../", followed by `TAB`, as in:

```
read.dta("../
```

followed by `TAB`.

Debugging

This is a relatively new feature of RStudio and won't be better introduced than on the RStudio website:

`https://support.rstudio.com/hc/en-us/articles/200713843-Debugging-with-RStudio`

Have a go with setting a breakpoint and messing around.

Linear algebra puzzles

These notes will provide a code illustration of the Linear Algebra review in Chapter 1 of the lecture notes. Don't worry if you can't solve these puzzles. Come back to them later, once we have gone over R code in more detail. There are many correct ways to solve these puzzles. We may go over a few solutions in section.

1. Let $\mathbf{I}_5$ be a 5×5 identity matrix. Demonstrate that $\mathbf{I}_5$ is symmetric and idempotent using simple functions in R.

2. Generate a 2×2 idempotent matrix $\mathbf{X}$, where $\mathbf{X}$ is not the identity matrix. Demonstrate that $\mathbf{X} = \mathbf{XX}$.

3. Generate two random variables, $\mathbf{x}$ and $\mathbf{e}$, of dimension $n = 100$ such that $\mathbf{x}, \mathbf{e} \sim N(0, 1)$. Generate a random variable $\mathbf{y}$ according to the data generating process $y_i = x_i + e_i$. Show that if you regress $\mathbf{y}$ on $\mathbf{x}$ using the canned linear regression routine `lm()`, then you will get an estimate of the intercept β_0 and the coefficient on $\mathbf{x}$, β_1, such that $\beta_0 = 0$ and $\beta_1 = 1$.

4. Show that if $\lambda_1, \lambda_2, \ldots, \lambda_5$ are the eigenvectors of a 5×5 matrix $\mathbf{A}$, then $\text{tr}(\mathbf{A}) = \sum_{i=1}^{5} \lambda_i$.