Applied Economic Forecasting

Tutorial on R Studio IV

Spring 2020

Creating Directory and Set working directory

You can use the

```r
getwd()
```

command to obtain the current directory `R` is using.

It is good practice to set the working directory location to where the files and data are stored.

- Consider setting your working directory to a folder called AAEC4984 or AAEC5984 on your desktop (for example).

Creating Directory and Set working directory

Windows

```r
setwd("C:/users/[your user name]/Desktop/AAEC4984/")
# OR
setwd("C:\\users\\[your user name]\\Desktop\\AAEC4984\\")
# notice the double backslashes
```

Mac

```r
  setwd("~/Desktop/AAEC4984")
```

- To check whether the wd is correct, we again use

```
getwd()
```

- To obtain a list of the names of files or folders in the working directory, we can use

```
dir()
```

- To create a new folder in your directory we can use

```
dir.create("[Folder name]")
```

Importing data

`R` allows us to import several file types. I will discuss 3 that we are most likely to use in this course.

1. Text files: Data sometimes come with headers (the first row is variable names, not actual data!) You need to tell R that!

```
textdata<-read.table("examples/hogsdata.txt",header=T)
```

2. CSV files

```
csvdata<-read.csv("examples/hogsdata.csv",header=T)
```

3. xlsx files (requires openxlsx package)

```
xlsxdata<-read.csv("examples/hogsdata.xlsx", ... )
```

VIRGINIA TECH.

Using Functions & Packages

Functions are "canned scripts" that automate more complicated sets of commands including operations assignments, etc. For the purpose of this course, we will use a lot of functions that are built both in base R (that is, they are predfined) or available through R packages (discuss below).

A function usually takes one or more inputs called *arguments*, and often (but not always) return a *value*.

Consider for example, taking the average of a set of random numbers (x).

```r
set.seed(124)
x <- rnorm(6) * 100
(round(x, digits=2)) # round function => 2dp
```

```
## [1] -138.51    3.83  -76.30   21.23  142.55   74.45
```

If we were to do this manually, we would:

1. Sum up the values

```r
sumx <- sum(x)
```

2. Get the number of observations

```r
nx <- length(x)
```

3. Divide sum by total number of observations

```r
meanx <- sumx/nx
```

VIRGINIA TECH.

Using R's built in mean functions we can do all three steps internally and cross check against our manual calculations.

```r
mean(x)
```

```
## [1] 4.542439
```

```r
meanx == mean(x) # cross validation
```

```
## [1] TRUE
```

Installing Packages

Since R is an Open Source software program, thousands of people contribute to the software. They do this by writing commands (called functions) to make a particular analysis easier, or to make a graphic prettier.

When you download R, you get access to a lot of functions that we will use. However the other user-written packages we use for our analyses will make our lives much easier.

For example, though we can use the `plot` command for standard graphics, you will quickly see that we can get much better looking time graphs using the `fpp2` package (which also uses another package called ggplot2).

Installing Packages

To install the fpp2 package, we can use the command

```
install.packages("fpp2")
```

We will need to install a package only once in R.

Now that you have the fpp2 package installed, we can check to see if it is in use

```
search()
```

Lastly, in order to use the package, we will need to load the library

```
library(fpp2)
```

Using libraries

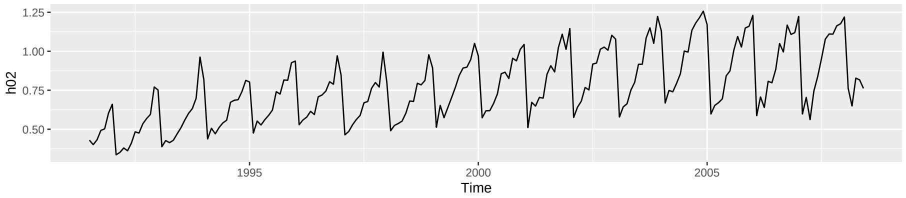
The fpp2 package contains a number of useful datasets. One such data set is h02.

Use the help() function to get a decription of this data. Try

```
help(h02)
```

Now let us create a nice plot of the h02 data

```
autoplot(h02)
```



Let us leave it there for now!