

MFE R Programming Workshop

Week 1

Brett Dunn

Fall 2016

Goals

- ▶ Learn to program in R.
- ▶ What does programming mean?
 - ▶ Language syntax.
 - ▶ Debugging.
 - ▶ Finding solutions.
 - ▶ Translating math to code.
- ▶ This is just the beginning, you'll develop these skills throughout the program.

R as a language

- ▶ R is object oriented.
 - ▶ Everything is an object and functions operate differently when passed different types of objects.
- ▶ R is functional.
 - ▶ You write fewer loops.
 - ▶ You write cleaner code.

R vs C++

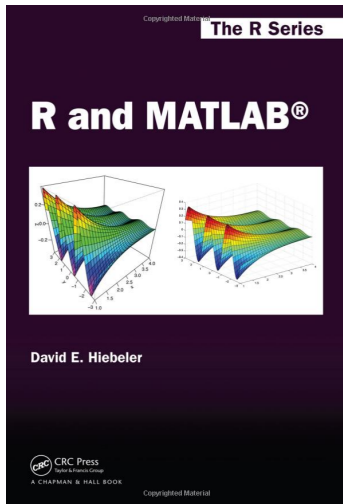
- ▶ Both are useful, and you will use both in the MFE program.
- ▶ R is an interpreted language.
 - ▶ Low programmer time.
 - ▶ A great tool for data munging, statistics, regressions, ect.
 - ▶ However, certain tasks in R can be slow (e.g. loops).
- ▶ C++ is very fast, but it takes longer to write programs.
- ▶ We can use both together!
- ▶ A good workflow:
 1. Write your program in R.
 2. If the program is too slow, benchmark your code.
 3. Try to speedup any bottlenecks in R.
 4. Convert any remaining bottlenecks to C++.

Jack of All Trades, Master of None

- ▶ You are better served by learning R and C++ very well, rather than trying to learn R, C++, MATLAB, Python, Julia, SAS, ect.
- ▶ The MFE program is just too short.
 - ▶ You also need to learn finance!
- ▶ Once you are proficient with R and C++, learning other languages is easy.
- ▶ Don't become a master of none!

MATLAB

- If you want to learn MATLAB after learning R, take a look at **R and MATLAB** by David Hiebeler.



Structure

- ▶ I will talk at the beginning of each class.
- ▶ For the remainder of the time you will break into your study groups and work on programming tasks.
- ▶ Tasks are designed to introduce you to the building blocks that will be used for course assignments throughout the MFE program.
- ▶ This course is a programming course with emphasis on methods for finance:
 - ▶ You will see finance terms and math.
 - ▶ You *may* not understand all of the finance, but you will learn it throughout the program.
- ▶ The key skills will be translating mathematical algorithms into code and developing the ability to find helpful resources.

Questions

Any questions before we start?

R Resources: Books

- ▶ Introductory:

- ▶ R for Everyone by Jared P. Lander
- ▶ R Cookbook by Paul Teetor (free at [UCLA LearnIT](#))

- ▶ Intermediate:

- ▶ The Art of R Programming by Norman Matloff

- ▶ Advanced:

- ▶ Software for Data Analysis by John Chambers
- ▶ Extending R by John Chambers
- ▶ Advanced R by Hadley Wickham

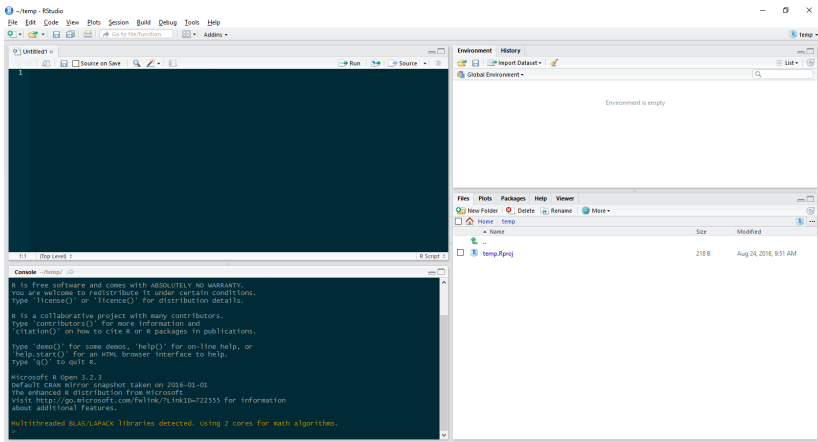
Other Resources

- ▶ Book series:
 - ▶ Use R! Springer series
 - ▶ FYI: Many Springer textbooks are just \$25 through <http://link.springer.com/>. You need to be on campus or signed into the UCLA VPN. You can download the pdfs for free.
 - ▶ O'Reilly R Books (free at [UCLA LearnIT](#))
- ▶ Built in documentation!
 - ▶ `?funcname`
- ▶ [Journal of Statistical Software](#)
- ▶ Data science courses on [Coursera](#)
- ▶ [Data Camp](#)
- ▶ Google, Stack Overflow, ect.

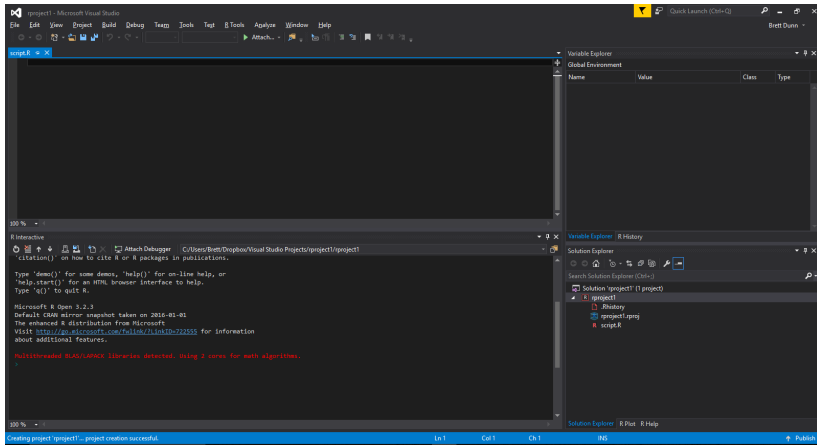
R Environment

- ▶ First, you need an R distribution.
 - ▶ I recommend Microsoft R Open.
 - ▶ <https://mran.revolutionanalytics.com/download/>
- ▶ Second, you need an integrated development environment (IDE) for R.
 - ▶ **R Studio** is a fantastic environment to interact with R.
 - ▶ Other options:
 - ▶ **R Tools for Visual Studio** if you use Visual Studio.
 - ▶ **Emacs Speaks Statistics (ESS)** if you use Emacs.
- ▶ I am going to assume that you have a working installation of R Studio and that you have a basic understanding of how it works.
- ▶ I will show you some Visual Studio.
- ▶ My focus is going to be on R programming.

RStudio

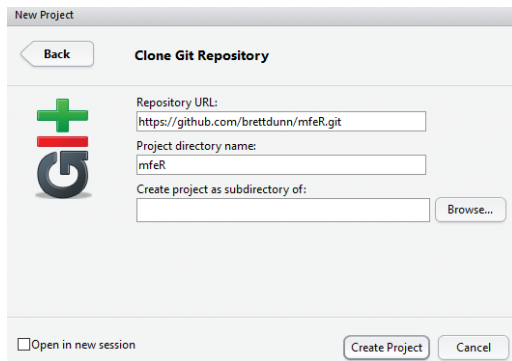


R Tools for Visual Studio



Course Materials

- ▶ <https://github.com/brettdunn/mfeR>
- ▶ The materials for this course were created in RStudio, using R Markdown.
- ▶ To create your own RStudio project:
 - ▶ File / New Project / Version Control / Git
 - ▶ Enter the URL



Command Line Interface

- ▶ To run a command in R, type it into the console next to the > symbol and press the Enter key.

```
2 + 3
```

```
## [1] 5
```

- ▶ Up Arrow + Enter repeats the line of code.
- ▶ Esc (Windows/Mac) or Ctrl-C (Linux) interrupts a command.

RStudio

- ▶ To start, create a new R Script file.
 - ▶ File/New File/R Script
- ▶ You can type your commands in the R Script file and run them on the Console.
 - ▶ Easy way to save your work.
 - ▶ `Ctrl+Enter` sends the line at the cursor to the console.
 - ▶ `Ctrl+Shift+S` runs the entire file.
 - ▶ Help/Keyboard Shortcuts lists all the available shortcuts.
 - ▶ Check out the multiple cursors.
- ▶ For larger tasks with many files, create an R project.
- ▶ Visual Studio is similar.

General Comments

- ▶ Make your code easy to read.
- ▶ Check out [Google's R Style Guide](#)
- ▶ Comment your code!

Google's R Style Guide

R is a high-level programming language used primarily for statistical computing and graphics. The goal of the R Programming Style Guide is to make our R code easier to read, share, and verify. The rules below were designed in collaboration with the entire R user community at Google.

Summary: R Style Rules

1. [File Names](#): end in .R
2. [Identifiers](#): `variable_name` (or `variableName`), `FunctionName`, `kConstantName`
3. [Line Length](#): maximum 80 characters
4. [Indentation](#): two spaces, no tabs
5. [Spacing](#)
6. [Curly Braces](#): first on same line, last on own line
7. [else](#): Surround else with braces
8. [Assignment](#): use `<-`, not `=`
9. [Semicolons](#): don't use them
10. [General Layout and Ordering](#)
11. [Commenting Guidelines](#): all comments begin with `#` followed by a space; inline comments need two spaces before the `#`
12. [Function Definitions and Calls](#)
13. [Function Documentation](#)
14. [Example Function](#)
15. [TODO Style](#): `TODO(username)`

R Packages

- ▶ A package is essentially a library of prewritten code designed to accomplish some task or a collection of tasks.
- ▶ R has a huge collection of user-contributed packages.
 - ▶ Warning: Not all packages are of the same quality.



[CRAN](#)
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

[About R](#)
[R Homepage](#)
[The R Journal](#)

[Software](#)
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

[Documentation](#)
[Manuals](#)
[FAQs](#)
[Contributed](#)

[Bayesian](#)
[ChemPhys](#)
[ClinicalTrials](#)
[Cluster](#)
[DifferentialEquations](#)
[Distributions](#)
[Econometrics](#)
[Environmetrics](#)
[ExperimentalDesign](#)
[ExtremeValueTheory](#)
[Finance](#)
[Genetics](#)
[Graphics](#)
[HighPerformanceComputing](#)
[MachineLearning](#)
[MedicalImaging](#)
[MetaAnalysis](#)
[Multivariate](#)
[NaturalLanguageProcessing](#)
[NumericalMathematics](#)
[OfficialStatistics](#)

CRAN Task Views

Bayesian Inference
Chemometrics and Computational Physics
Clinical Trial Design, Monitoring, and Analysis
Cluster Analysis & Finite Mixture Models
Differential Equations
Probability Distributions
Econometrics
Analysis of Ecological and Environmental Data
Design of Experiments (DoE) & Analysis of Experimental Data
Extreme Value Theory
Empirical Finance
Statistical Genetics
Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
High-Performance and Parallel Computing with R
Machine Learning & Statistical Learning
Medical Image Analysis
Meta-Analysis
Multivariate Statistics
Natural Language Processing
Numerical Mathematics
Official Statistics & Survey Methodology

R Packages

- ▶ Installing a packages:
 - ▶ Ctrl+7 in RStudio accesses the packages pane
 - ▶ You can also type `install.packages("packageName")`
- ▶ Uninstalling a package:
 - ▶ `remove.packages("packageName")`
- ▶ Loading packages:
 - ▶ `require(packageName)` or `library(packageName)` loads a package into R
 - ▶ The difference is that `require` returns `TRUE` if the package loads or `FALSE` if it doesn't.
- ▶ Unloading packages
 - ▶ `detach(package:packageName)`
- ▶ If two packages have the same function name use two colons:
`-package1::func` or `package2::func`

Variables

- ▶ Unlike C++, R does not require variable types to be declared.
- ▶ A variable can take on any data type.
- ▶ A variable can also hold any R object such as a function, the result of an analysis, a plot, ect.
- ▶ Variable assignment is done with `<-`.
 - ▶ `=` works, but there are reasons to prefer `<-`.
- ▶ We can remove variables (e.g. to free up memory) with the `rm` function. `gc()` runs garbage collection.

```
x <- 2  # x is a pointer  
x      # the same output as print(x)
```

```
## [1] 2
```

```
rm(x)  # removes x
```

Data Types

- ▶ There are many different data types in R.
- ▶ The four main types of data most likely to be used are:
 1. numeric
 2. character (string)
 3. Date/POSIXct (time-based)
 4. logical (TRUE/FALSE)
- ▶ The data type can be checked with the `class` function

```
x <- as.Date("2010-12-21")  
class(x)
```

```
## [1] "Date"
```

Casting

```
x <- "2010-12-21"  
class(x)
```

```
## [1] "character"
```

```
x
```

```
## [1] "2010-12-21"
```

```
x <- as.Date(x)  
class(x)
```

```
## [1] "Date"
```

```
x
```

```
## [1] "2010-12-21"
```

More Casting

```
x <- as.numeric(x)  
class(x)
```

```
## [1] "numeric"
```

```
is.numeric(x)
```

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

```
x # number of days since Jan 1, 1970
```

```
## [1] 14964
```

Even More Casting

```
x <- as.integer(x)  # x <- 14964L assigns an integer  
class(x)
```

```
## [1] "integer"
```

```
is.integer(x)
```

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

```
is.numeric(x)  # R promotes int to numeric as needed
```

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

```
4L / 5L
```

```
## [1] 0.8
```


Logicals

```
# TRUE == 1 and FALSE == 0  
x <- TRUE  # TRUE, FALSE, T, F are logicals  
is.logical(x)
```

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

```
5 == 5  # != tests for inequality
```

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

```
"a" < "b"  # works on characters as well
```

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

Vectors

- ▶ A vector is a collection of elements, all of the same type.
- ▶ We will learn about:
 - ▶ Recycling
 - ▶ The automatic lengthening of vectors.
 - ▶ Filtering
 - ▶ The extraction of subsets of vectors.
 - ▶ Vectorization
 - ▶ Where functions are applied element-wise to vectors.