Tools and Techniques for Data Science

Sajid Bashir, PhD

26 January, 2020

Contents

Introduction

Agenda

- Course overview
- Introduction to R, RStudio and R Notebooks/R Markdown
- Basics of R Programming

My Understanding

- Basic programming knowledge and skills
- Some stats knowledge e.g.
  - Hypothesis testing (t-tests, confidence intervals)
  - Linear regression
- Correlation between class attendance and performance/ learning
- Class will be *very* cumulative

Logistics

- One 160 minute lecture a week:
  - First 80-100 minutes: concepts, methods, examples
  - Last 80-60 minutes: short labs (if scheduled)
- Marks Distribution
  - Quizzes (15%)
  - Homework (20%) - Bi Weekly
  - Mid Term Exam (20%)
  - Final project (2.5 weeks)
  - Final Exams (25%)
- **Disclaimer:** There will be no late homework submission and grading.

Course Resources

- Class notes, homework etc. will be shared on `class group`. _____
- `TA` will be responsible for creating, managing the group. _____
- Use the group
  - Share queries for dicussion and information of others engaged.
  - All announcements by the teacher.
  - Share audio/ video recordings of the lecture.
- No required textbook, but several are *recommended*:
  - Garrett Grolemund and Hadley Wickham, *R for Data Science*
  - Phil Spector, *Data Manipulation with R*
  - Winston Chang, *The R Graphics Cookbook*

Goal of this class

This class will teach you to use R to:

- Generate graphical and tabular data summaries
- Perform statistical analyses (e.g., hypothesis testing, regression modeling)
- Produce *reproducible* statistical reports using R Markdown and R Notebooks
- Integrate R with other tools (e.g., databases, web, etc.)

Why to learn R?

- Free (open-source)
- Programming language (not point-and-click)
- Excellent graphics
- Offers broadest range of statistical tools
- Easy to generate reproducible reports

- Easy to integrate with other tools

The R Framework

Why to learn R?

- Free (open-source)
- Programming language (not point-and-click)
- Excellent graphics
- Offers broadest range of statistical tools
- Easy to generate reproducible reports
- Easy to integrate with other tools

The R Console

Basic interaction with R is through typing in the **console**. The **terminal** or **command-line** interface is shown in Figure 1.

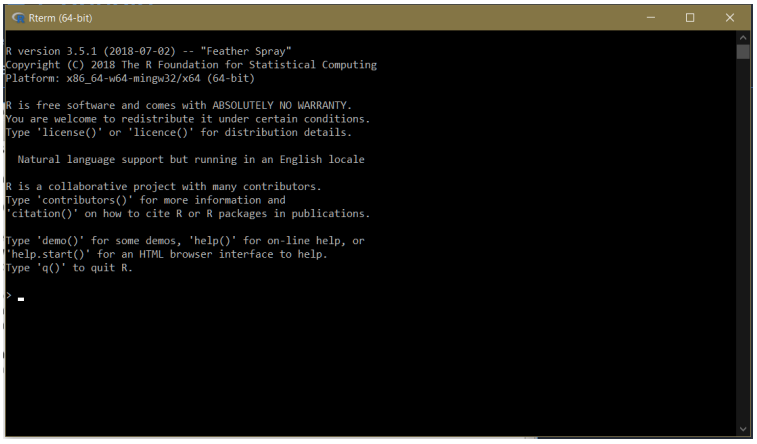

Figure 1: R terminal window or CLI.

- Type in your commands in cosole and R feeds back answers (or errors)
- Menus and other **graphical interfaces** are extras built on top of the console
- We will use **RStudio** in this class

1. Download R: http://lib.stat.cmu.edu/R/CRAN
2. Then download RStudio: http://www.rstudio.com/

RStudio

It is an IDE for R with 4 main windows ('panes') as shown in Figure 2:

1. Console
2. Source
3. Workspace/History
4. Files/Plots/Packages/Help

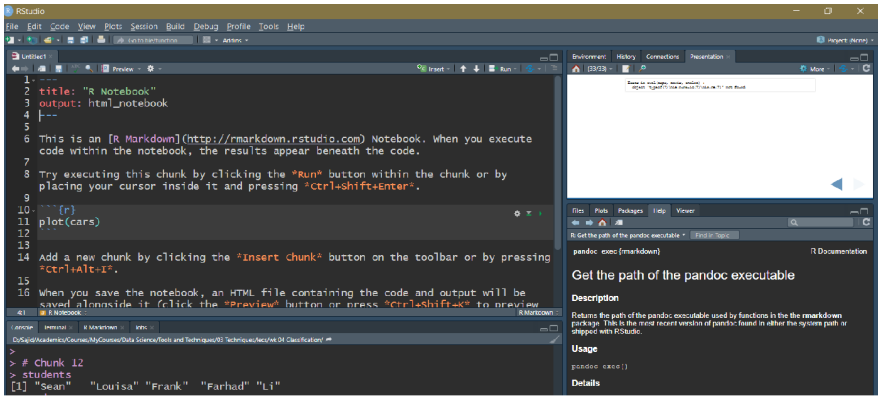

Figure 2: RStudio with it's four panes.

R Console in RStudio

The R Console in the RStudio is as shown in Figure 3.

- Type your command in **R Console** pane to type or paste commands to get output from R
- To look up the help file for a function or data set, type `?function` into the Console
  - e.g., try typing in `?mean`
- Use the `tab` key to auto-complete function and object names

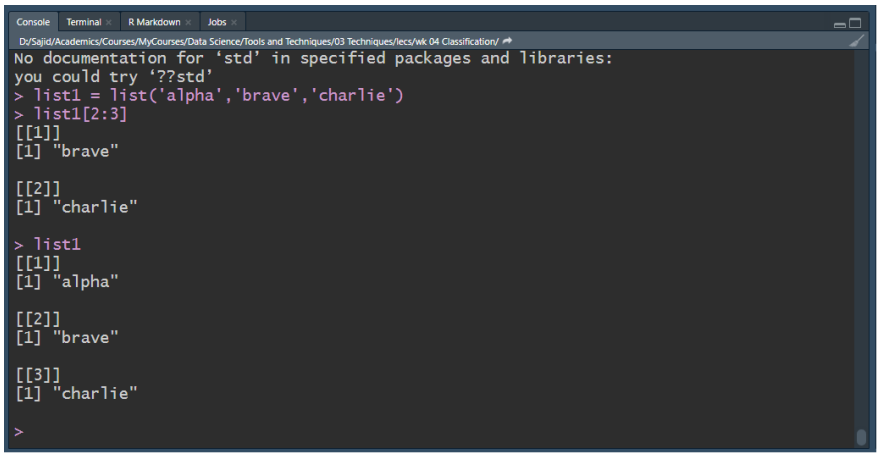

Figure 3: The R Console in RStudio.

Source Pane

The source pane is one of the windows of RStudio GUI as shown in Figure 4.

- Use the **Source** pane to create and edit R and Rmd files
- The menu bar of this pane contains handy shortcuts for sending code to the **Console** for evaluation

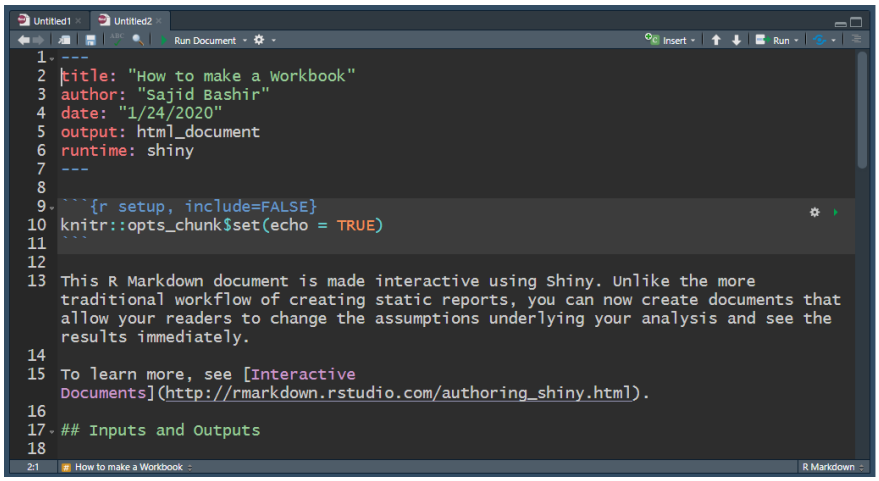

Figure 4: The RStudio Source Pane.

Files/Plots/Packages/Help Pane

One of the panes in RStudio GUI provides a comprehensive view of the Files, plots, list of installed packages and the detailed description of the help. The pane is shown in Figure 5.

- By default, any figure produced in R is displayed in **Plots** tab
- Menu bar allows Zoom, Export, and Navigate back to older plots
- When you request a help (e.g., `?mean`), help documentation appears in **Help** tab

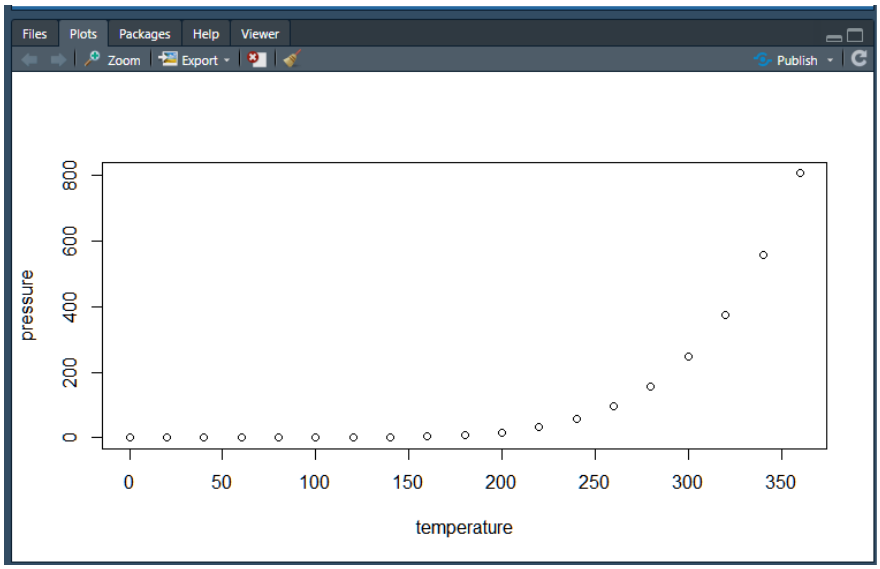

Figure 5: Help and Plot Pane in RStudio.

Operations through RStudio Panes

1. **Console** pane: type or paste in commands to get output from R
2. **Source** pane: create a file that you can save and run later

3. **Workspace/History** pane: see a list of variables or previous commands
4. **Files/Plots/Packages/Help** pane: see plots, help pages, and other items in this window.

Source and Console Panes

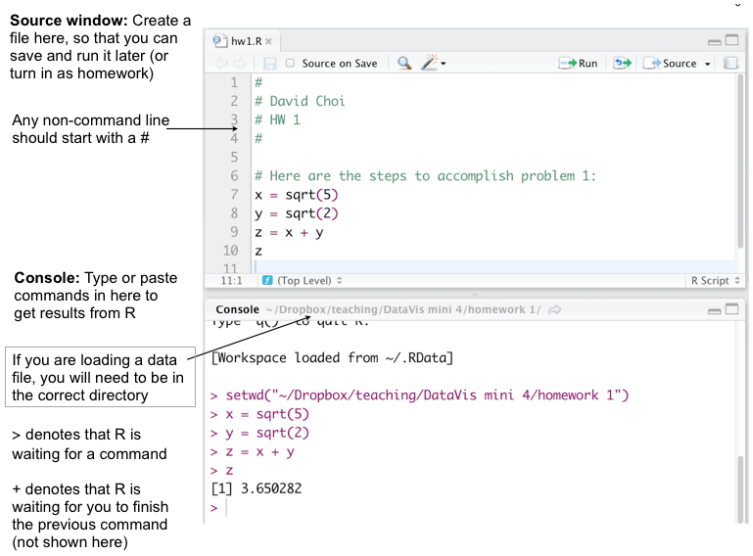

Figure 6: Source and Console Panes

Console Pane

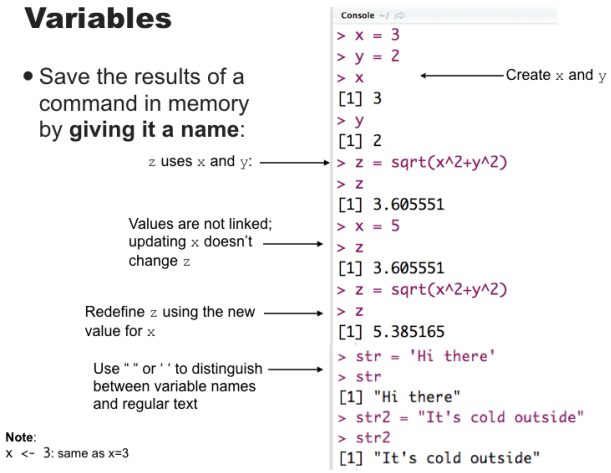

Figure 7: Data Operations in Console Pane

RStudio: Toolbar

R Markdown, R Notebooks

- R Markdown allows user to integrate R code into a report

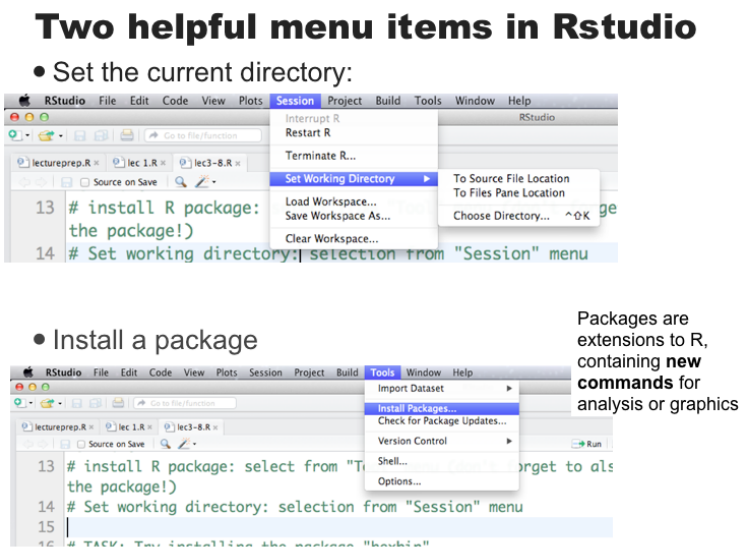

Figure 8: Helpful menu items of RStudio

- When data changes or code changes, so does the report
- No more need to copy-and-paste graphics, tables, or numbers
- Creates **reproducible** reports
  - Anyone who has your R Markdown (.Rmd) file and input data can re-run your analysis and get exactly same results (tables, figures, summaries)
  - R Notebooks are R Markdown documents that allow to execute code interactively and view output in notebook.
- Can output report in HTML (default), Microsoft Word, or PDF

R Markdown

- Example shows an **R Markdown** (.Rmd) file opened Source pane.

- Click **Knit HTML** in `Source pane menu bar` and convert the Rmd file into a report.
- Results appear in a **Preview window**, as shown on the right.
- You can knit into html (default), MS Word, and pdf format

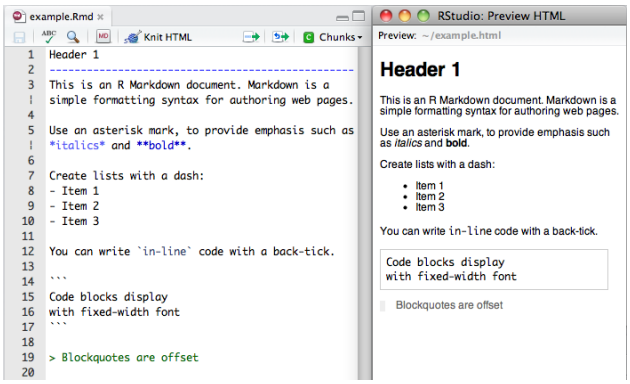

Figure 9: R Markdown document in R Source Pane.

- Integrate R output into report, you need to use R code chunks

- All of the code that appears in between "triple back-ticks" gets executed when Knit

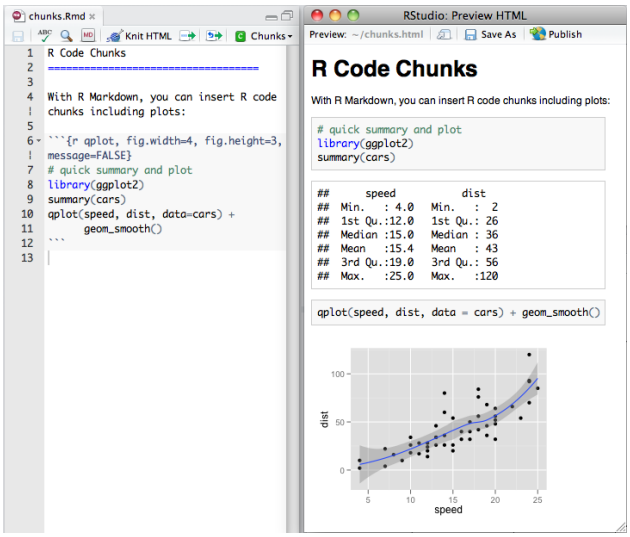

Figure 10: R Code Chunks.

Take-Home Exercise

Hello world!

1. Open **RStudio**
2. File > New File > R Markdown ...
3. Change `summary(cars)` in first code block to `print("Hello world!")`
4. Click `Knit HTML` to produce an HTML file.
5. Save your Rmd file as `helloworld.Rmd`

   All of your Homework assignments and many of your Labs will take the form of a single Rmd file, which you will edit to include your solutions and then submit.

Basics of R Programming

Basics: R in a nutshell

- Everything we'll do comes down to applying **functions** to **data**
- **Data**: things like 9, "nine", 99.000, the matrix $\begin{bmatrix} 9 & 9 & 9 \\ 9 & 9 & 9 \end{bmatrix}$
- **Functions**: things like log , + (two arguments), < (two), mod (two), `mean` (one)

  A function is a machine which turns input objects (**arguments**) into an output object (**return value**), possibly with **side effects**, according to a definite rule

Data building blocks

You'll encounter different kinds of data types

- **Booleans** Direct binary values: `TRUE` or `FALSE` in R
- **Integers**: whole numbers (positive, negative or zero)
- **Characters** fixed-length blocks of bits, with special coding; **strings** = sequences of characters
- **Floating point numbers**: a fraction (with a finite number of bits) times an exponent, like 1.87×10^6
- **Missing or ill-defined values**: `NA`, `NaN`, etc.

Operators (functions)

You can use R as a very, very fancy calculator

Command	Description
`+,-,*,\`	add, subtract, multiply, divide
`^`	raise to the power of
`%%`	remainder after division (ex: `8 %% 3 = 2`)
`( )`	change the order of operations
`log(), exp()`	logarithms and exponents (ex: `log(10) = 2.302`)
`sqrt()`	square root
`round()`	round to the nearest whole number (ex: `round(2.3) = 2`)
`floor(), ceiling()`	round down or round up
`abs()`	absolute value

```
99 + 9 # Addition
```

```
## [1] 108
```

```
9 - 99 # Subtraction
```

```
## [1] -90
```

```
9 * 99 # Multiplication
```

```
## [1] 891
```

```
9 ^ 9 # Exponentiation
```

```
## [1] 387420489
```

```
9 / 99 # Division
```

```
## [1] 0.09090909
```

```
9 %% 5.99 # Modulus
```

```
## [1] 3.01
```

```
18.5 %/% 9 # Integer division
```

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

Operators (Comparison)

Comparisons are also binary operators; they take two objects, like numbers, and give a Boolean

```
9 > 5
```

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

```
9 < 5
```

```
## [1] FALSE
```

```
9 >= 5
```

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

```
9 <= 5
```

```
## [1] FALSE
```

```
9 == 5
```

```
## [1] FALSE
```

```
9 != 5
```

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

Operators (Boolean)

Basically `and` and `or`:

```
(9 < 5) & (6*6 == 36)
```

```
## [1] FALSE
```

```
(9 < 5) | (6*6 == 36)
```

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

(will see special doubled forms, `&&` and `||`, later)

Special Functions

- `typeof()` function returns the type
- `is.`*foo*`()` functions return Booleans for whether the argument is of type *foo*
- `as.`*foo*`()` (tries to) "cast" its argument to type *foo* — to translate it sensibly into a *foo*-type value

Special case: `as.factor()` will be important later for telling R when numbers are actually encodings and not numeric values. (E.g., 1 = Red; 2 = Green; 3 = Blue)

```
typeof(99)
```

```
## [1] "double"
```

```
typeof('99')
```

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

```
is.numeric(99)
```

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

```
is.na(99)
```

```
## [1] FALSE
```