Introduction to R/RStudio

Jerome Dumortier

Tutorial Overview

Topics covered in this tutorial

- Overview of R/RStudio
- Data management
- Plotting and graphs with R
- Basic statistics

Exercises will be conducted throughout the tutorial

Online Resources and Help

Very large user community for R

- Google search for “Some topic R” usually leads quickly to the desired help

Here are the links to a few online tutorials

- UCLA OARC Statistical Methods and Data Analytics
- Statmethods
- Statistical Tools for High-Throughput Data Analysis: Very useful for more advanced applications such as plotting with the package `ggplot2`

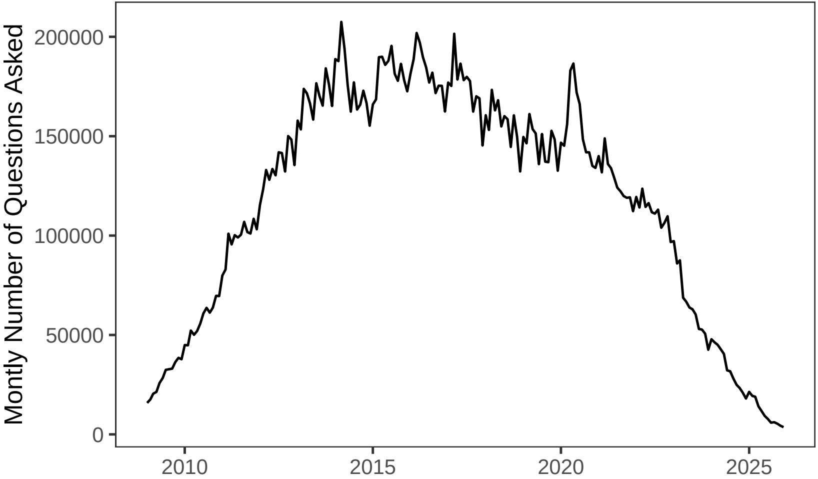
Two Particularly Useful Online Resources

Prior to Large Language Models (LLM), two resources—usually pointed to via a Google search—provided the solution to the vast majority of R questions

- Statistical Data Analysis R: Resource containing the function manual for R/RStudio including all packages
  - Example for boxplot
  - Examples as most helpful part at the bottom of the documentation page
- Stack Overflow: Resources for developers
  - Google search: r ggplot two y axis

Note that all questions on Stack Overflow have to be accompanied by an easily reproducible example

Evolution of Stack Overflow



Using LLMs in Statistics and Regression

LLMs (e.g., ChatGPT)

- Generation of text by predicting likely continuations given input

LLMs as one of the accelerators in the decline of Stack Overflow

- Useful for explaining concepts, drafting code, debugging, summarizing results, and writing intuition
- Strengths: Fast iteration, natural language interface, broad statistical knowledge
- Limitations: Hallucinations, incorrect mathematics, and no inherent understanding of data-generating process

Understanding of statistics and regression models is still required for effective, correct, and efficient use of artificial intelligence

Prompting Tips for Statistical Work

Maybe most important: Spoon-feed

- Do not plug a large paragraph about a statistical problem into AI. Go sentence by sentence and explain data structure (e.g., name of variables in data frame)
- Ask AI to wait until prompted to answer

Other tips

- Be explicit about task (e.g, explanation, derivation, code, critique, or rewrite)
- Specify context and constraints (e.g., assumptions, notation, or data structure)
- Ask for structure (e.g., steps, equations, or checks)
- Request verification (e.g, *Show derivation*, *State assumptions*, *Flag uncertainty*)
- Iterate: Refine prompts based on output rather than expecting a perfect first answer

Good versus Bad Prompts

Bad prompt 1: *Explain regression*

- Good: *Explain OLS regression to a master-level public policy class, include assumptions, intuition, and a simple equation.*

Bad prompt 2: *Fix my R code*

- Good: *Debug this R function, explain the error, and rewrite it using base R only. Here is the code:*

Bad prompt 3: *What is the right model?*

- Good: *Given a panel data with county fixed effects and serial correlation, compare suitable models and their trade-offs.*

Bad prompt 4: *Is this result correct?*

- Good: *Check this derivation step by step, state assumptions, and identify where errors could arise.*

Opening RStudio

Working with RStudio is done in four windows

- Script Window
  - This is were you type your R Script (.R) and where you execute commands
  - Comparable to do-file/editor in Stata
  - This window needs to be opened by File ⇒ New File ⇒ R Script
- Console window
  - Use of R interactively. Should only be used for quick calculations and not part of an analysis
- Environment
  - Lists all the variables, data frames, and user-created functions
  - It is tempting to use the "Import Dataset" function . . . Don't!
- Plots/Packages/Help

Packages

There is a base version of R that allows doing many calculations but the power of R comes through its many packages. To use functions associated with a particular package (e.g., to read data from Excel), click “Install” in the packages window of RStudio and type in the name of the desired package. Or alternatively, use

```r
install.packages("openxlsx")
```

To use a package, you have to activate it by including

```r
library("openxlsx")
```

Packages are updated on a regular basis by users

Front Matter

Hash tags makes R skip whatever is after. The following command clears all variables from R

```
rm(list=ls())
# Syntax after a hash tag is skipped by R
```

To display the current working directory and to set a new one

```
getwd()
setwd("C:/Users/Jerome/Documents/R Lecture")
```

You have to change the part between the quotation marks to the directory you have created. For file paths, replace backslashes with forward slashes

Loading Data

The following should be on one line but is on two lines in the present slide for space purposes. The line loads data from the GitHub data directory automatically

```
load(url("https://github.com/jrfdumortier/DataAnalysis/raw/main/
         DataAnalysisPAData.RData"))
```

R can import data from a wide variety of format (e.g., Excel, comma separated values, SAS, STATA)

- Most basic data import is using `read.csv()`

It is also good practice to save your R-script on a regular basis

Exercise 1

Create a R-script file with the following components

- Two lines for the title and the date (use #)
- Clearing all current contents
- Setting the correct working directory. This should be a folder to which you have downloaded all materials
- Installing and loading the package openxlsx

Functions I

At the core of R are functions that “do things” based on your input. The basic structure is

```
object = functionname(argument1=value,argument2=value,...)
```

Components

- `object`: Output of the function will be assigned to object
- `functionname`: Name of the system function. You can also create and use your own functions. More about this later
- `argument`: Arguments are function specific
- `value`: The value you want a particular argument to take

Functions II

Notes

- If a function is executed without an specific assignment, the output will be displayed in the console window
- Before using a function, read the documentation
- Many functions have default settings. Be aware of default values. In most cases, those defaults are set to values that satisfy most uses

Notation in the help file

- Consider the help file for the function hist

Example about default values

- `t.test(x,y=NULL,[...],mu=0,conf.level=0.95,[...])`

Main Data Types in R

Vectors

```
preselection    = seq(1788,2016,4)
midterm         = seq(by=4,to=2018,from=1790)
```

Matrix (only numerical values are allowed)

```
somematrix      = matrix(8,10,4)
```

Data frames

- By far, the most common data type in R
- Comparable to an Excel spreadsheet

Lists (Collection of objects from of various types)

```
myfirstlist = list(preselection,midterm,somematrix)
```

Using R as a Calculator

Entering heights of people and storing it in a vector named `height`

```
height         = c(71,77,70,73,66,69,73,73,75,76)
```

Calculating the sum and mean is done with the following commands

```
sum(height)                   # Output in console
meanheight     = mean(height) # Output in environment
```

Calculating the height squared (element-wise squaring)

```
height_sq      = height^2
```

Removing (i.e., deleting) unused elements

```
rm(heightsq,meanheight)
```

Creating a Data Frame from Scratch

Data frames are the most commonly used tables in R/RStudio and are similar to spreadsheets

- Column names represent the variables and rows represent observations
- Column names must be unique and without spaces

Suggestion: Use only lower-case variable names and objects

```
studentid      = 1:10
studentnames   = c("Andrew","Linda","William","Daniel",
                   "Gina","Mick","Sonny","Wilbur",
                   "Elisabeth","James")
students       = data.frame(studentid,studentnames,height)
rm(studentid,height,studentnames)
```

Exercises 2

Create a data frame called `students` containing the following information

Name	Economics	English
Mindy	80.0	52.5
Ruiqing	60.0	60.0
Shubra	95.0	77.5
Keith	77.5	30.0
Luisa	97.5	95.0

- Use *name* as the column header for the students' names
- Once you have created the data frame, remove the unused vectors

Indexing I

Indexing refers to identifying elements in your data

- For most objects: `students[row number,coloumn number]`
  - `students[3,2]` returns 95
  - What does `students[3,]` return?
- If you want to select certain columns: `students[c("name")]`
  - Other example: `students[c("name","english")]`

Selecting results based on certain conditions

```
students[which(students$economics>80),]
```

Indexing II

Referring to a particular column in a data frame is done through the dollar symbol (often used functionality)

```r
students$english
```

Creating a new column

```r
students$average = rowMeans(students[c("economics","english")])
```

Basic Handling of Data Frames I

Data on vehicle fuel efficiency for all model years (1984–2020) from DOE and EPA with corresponding documentation of the variables. Sub-setting data is done with the command `subset()`

```r
cars2015 = subset(vehicles,year==2015)
```

Note that the double equal sign conducts a logical test. To list all EPA vehicle size classes (*vclass*)

```r
unique(cars2015$vclass)
```

Suppose you are only interested in the variables *co2tailpipegpm* and *vclass* for the model year 2015

```r
cars2015 = subset(vehicles,year==2015,
                  select=c("co2tailpipegpm","vclass"))
```

Basic Handling of Data Frames II

Suppose you are only interested in *Compact Cars* and *Large Cars* in the column *vclass* for the year 2015. The notation is a bit odd (note that the many line breaks are not necessary to include in R)

```
cars2015 = subset(vehicles,
                  year==2015 &
                  vclass %in% c("Compact Cars","Large Cars"),
                  select=c("make","co2tailpipegpm","vclass"))
```

Exercises 3

From the vehicles data set, extract the GHG Score and the vehicle class from the 2014 model year for the following manufacturers: Toyota, Ford, and Audi. Your new data set should contain the following columns: *co2tailpipegpm*, *make*, and *vclass*. Is the resulting data frame sensible or do you see a problem?

Aggregating Data and Writing .csv-Files

To aggregate data based on a function, e.g., sum or mean

```
cars2015 = aggregate(co2tailpipegpm~make+vclass,
                     FUN=mean,data=cars2015)
```

To write data to the current working directory

```
write.csv(cars2015,"cars2015.csv")
```

Importing Data into R/RStudio

Machine-readable data can be imported as follows

- `read.csv("filename.csv")`: If you have a comma separated value (.csv) file then this is the easiest and preferred way to import data
- `readWorkbook(file="filename.xlsx",sheet="sheet name")`: Requires the package openxlsx. Note that there are many packages reading Excel and this is the most reliable and user-friendly

Importing data from other software packages (e.g., SAS, Stata, Minitab, SPSS) or .dbf (database) files

- Package foreign reads .dta Stata files (Version 5-12) with the command `read.dta`
- Package readstata13 reads files from newer Stata versions

Extending the Basic `table()` Function

Required package

- gmodels

Compare the outputs of `table()` and `CrossTable()`

```
library(gmodels)
table(gss$owngun,gss$sex)
CrossTable(gss$owngun,gss$sex)
```

Note that for almost any R command, you can store the output by assigning it a name (e.g., `output` in the case below)

```
output = CrossTable(gss$owngun,gss$sex)
```

Merging Datasets (ohioscore and ohioincome)

Consider two datasets from school districts in Ohio

- ohioscore which contains an identifier column *IRN* and a score that indicates quality of the school
- ohioincome which contains the same identifier than the previous sheet in addition to median household income and enrollment

One important function to merge data sets in R

```
ohioschool = merge(ohioscore,ohioincome,by=c("irn"))
rm(ohioscore,ohioincome)
```

Overview

Before we start talking about graphics, execute the following command

```
demo(graphics)
```

R has very advanced graphing capabilities that allows you to do any type of visualization. Personally, I use it most often for

- Automatically updating graphs for manuscripts
- Side-by-side plots
- Plotting maps

In almost all cases, vector graphs are preferred over bitmap graphs

Faithful Dataset: Summary

R and some packages include example data sets to facilitate learning the package. A widely-used R data set is `faithful`

```
summary(faithful)
```

```
##    eruptions        waiting    
##  Min.   :1.600   Min.   :43.0  
##  1st Qu.:2.163   1st Qu.:58.0  
##  Median :4.000   Median :76.0  
##  Mean   :3.488   Mean   :70.9  
##  3rd Qu.:4.454   3rd Qu.:82.0  
##  Max.   :5.100   Max.   :96.0  
```

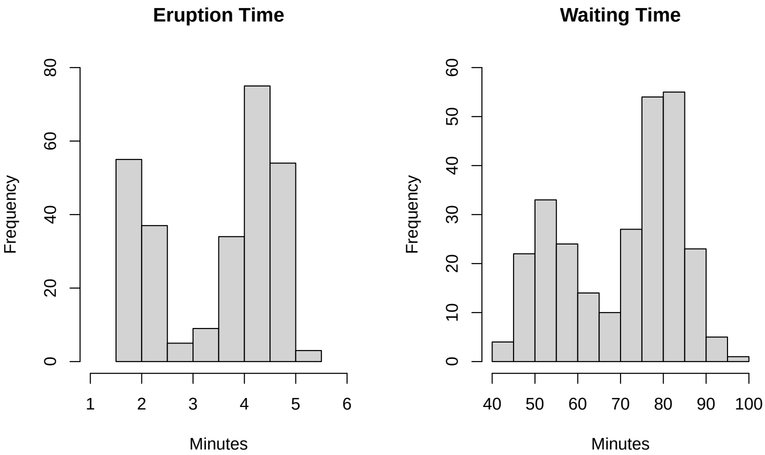
```
cor(faithful$eruptions,faithful$waiting)
```

```
## [1] 0.9008112
```

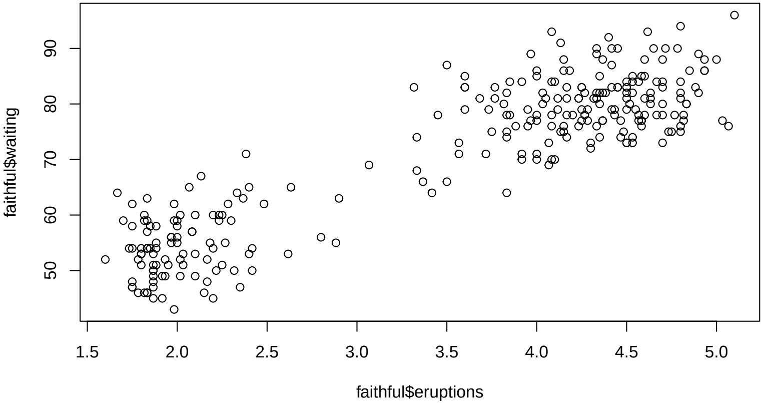
Faithful Dataset: Histogram Setup

```r
par(mfrow=c(1,2))
hist(faithful$eruptions,main="Eruption Time",
    xlab="Minutes",xlim=c(1,6),ylim=c(0,80))
hist(faithful$waiting,main="Waiting Time",
    xlab="Minutes",xlim=c(40,100),ylim=c(0,60))
```

Faithful Dataset: Histogram Plot



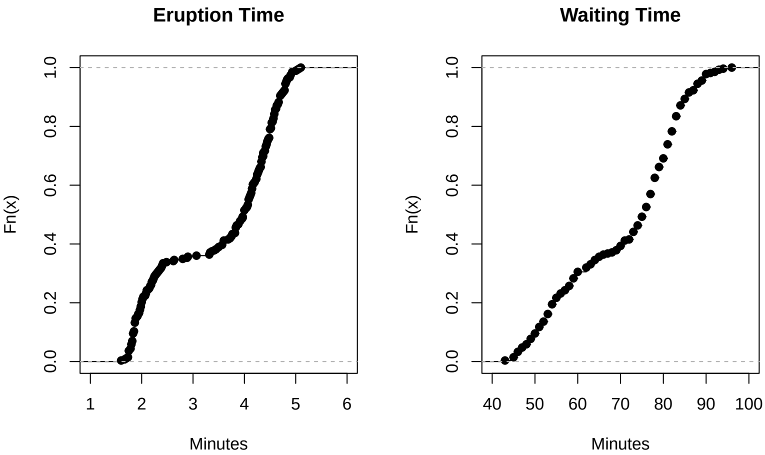
Faithful Dataset: Correlation



Faithful Dataset: ECDF Setup

```
par(mfrow=c(1,2))
plot(ecdf(faithful$eruptions),
   main="Eruption Time",
   xlab="Minutes",xlim=c(1,6))
plot(ecdf(faithful$waiting),
   main="Waiting Time",
   xlab="Minutes",xlim=c(40,100))
```

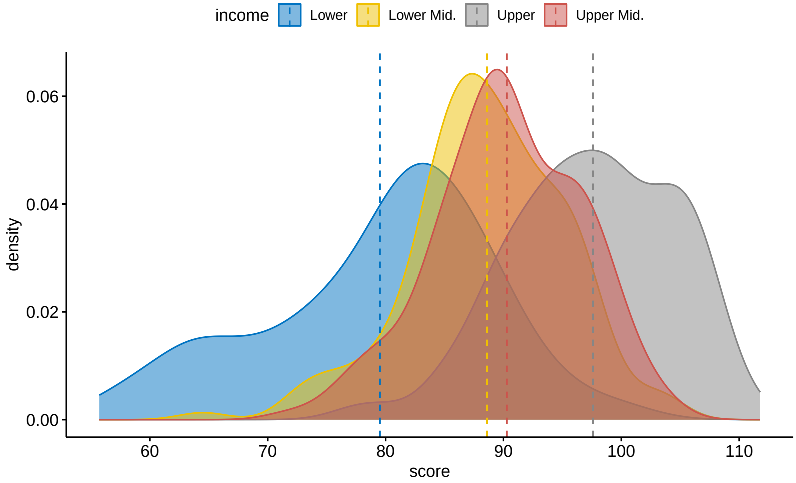
Faithful Dataset: ECDF Plot



Plotting Ohio School Scores: Setup

```
library(ggpubr)
library(ggsci)
topandbottom   = quantile(ohioschool$medianincome,
                          seq(0,1,0.25))
quartiles      = as.integer(cut(ohioschool$medianincome,
                            quantile(ohioschool$medianincome,
                            probs=0:4/4),include.lowest=TRUE))
ohioschool$quartiles                       = quartiles
ohioschool$income                          = NA
ohioschool$income[ohioschool$quartiles==1] = "Lower"
ohioschool$income[ohioschool$quartiles==2] = "Lower Mid."
ohioschool$income[ohioschool$quartiles==3] = "Upper Mid."
ohioschool$income[ohioschool$quartiles==4] = "Upper"
ggdensity(ohioschool,x="score",add="mean",
    color="income",fill="income",palette="jco")
```

Plotting Ohio School Scores: Figure


income
Lower
Lower Mid.
Upper
Upper Mid.
density
score