# Introduction to R/RStudio

Jerome Dumortier

26 August 2022

# 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.

# Overview of R/RStudio

# 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

There is also `www.sthda.com/english/`, which is very useful for some more advanced applications like plotting with the package `ggplot2`.

# Two Particularly Useful Online Resources

Two online resources will provide you the solution to the vast majority of your R questions. Getting to those websites is usually the result of a Google search.

- Statistical Data Analysis R: This resource contains the function manual for R/RStudio including all packages:
  - Example for boxplot
  - The most helpful part are the examples at the bottom of the 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.

# 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 $\Rightarrow$ New File $\Rightarrow$ 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

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

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

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

Packages are updated on a regular basis by users.

## Front Matter

The purpose of the hashtag is that R will skip whatever is after. The following command clears all variables from R:

```
rm(list=ls())
```

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. The following will import sample data.

```
honda = read.csv("honda.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,[...])`

# Data Management

# Data in R

The main data types in R are:

- 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 sheet.
  - More on this later.
- 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, product, natural log, or mean is done with the following commands:

```
sum(height)
prod(height)
log(height) # Default is the natural log
meanheight = mean(height)
```

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. They are similar to an Excel sheet.

- 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

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),]`

Referring to a particular column in a data frame is done through the dollar symbol:

- `students$english`
- You will use this functionality very often.

Creating a new column: `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`:

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

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

```
unique(cars2015$VClass)
```

Suppose you are only interested in the variables *ghgScore* and *VClass* for the model year 2015.

- `cars2015 = subset(vehicles,year==2015,select=c("ghgScore","VClass"))`

Get glimpse at the results: `table(cars2015\$VClass,cars2015\$ghgScore)`

# 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. There 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","ghgScore","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: *ghgScore*, *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(cars2015$ghgScore,
                     by=list(cars2015$make,cars2015$VClass),
                     FUN=mean)
```

To write data to the current working directory:

```
write.csv(cars2014,"cars2014.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(gssgun$owngun,gssgun$sex)
CrossTable(gssgun$owngun,gssgun$sex)
```

Note that for almost any R command, you can store the output by assigning it a name:

```
output = CrossTable(gssgun$owngun,gssgun$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 datasets in R:

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

# Plotting and Graphs with R

# 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 "famous" R data set is `faithful`:

```
faithful = 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
```

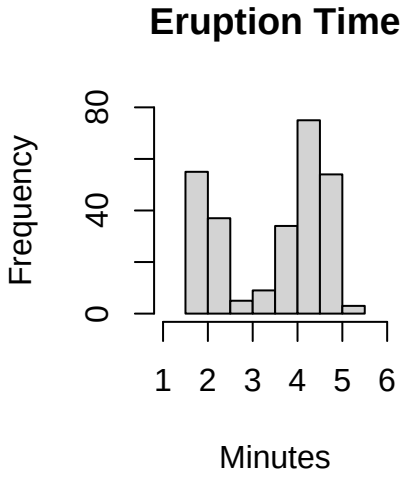
```
cor(faithful$eruptions,faithful$waiting)
```

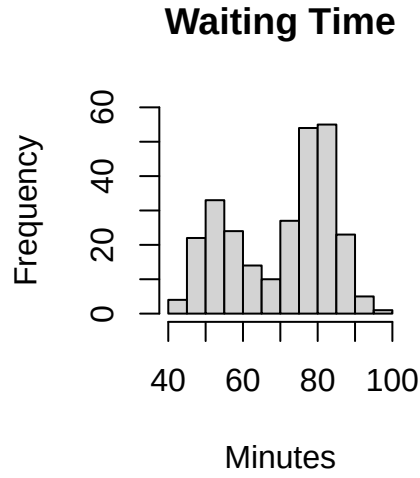
```
## [1] 0.9008112
```

# Faithful Dataset: Histogram Setup

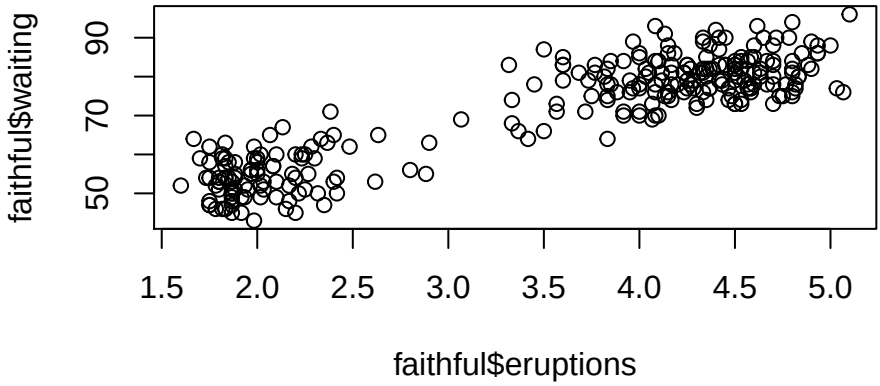
```
par(mfrow=c(1,2))
hist(faithful$eruptions,main="Eruption Time",
    xlab="Minutes",xlim=c(1,6),ylim=c(0,80))
hist(faithful$waiting,main="Waiting Time",
    xlab="Minutes",xlim=c(40,100),ylim=c(0,60))
```

# Faithful Dataset: Histogram Plot



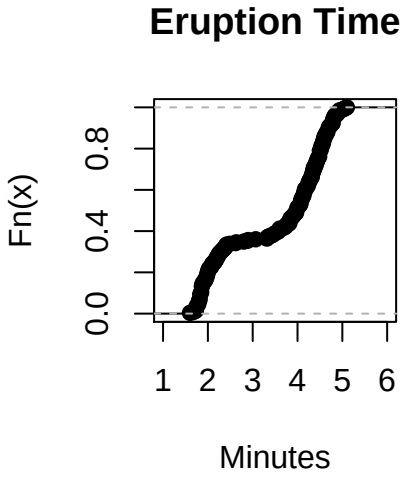
# Faithful Dataset: Correlation

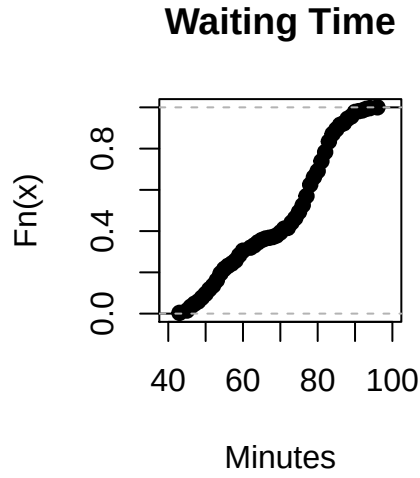


# Faithful Dataset: ECDF Setup

```
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