Data Skills Workshop

Matthias Haber

3-4 February 2020

Introduction

Survey results

```
## pdf
##   2
```

About myself

- Head of Data at Looping Studios since 2018
- Postdoc at Hertie 2015-2017 (Governance Report)
- PhD in PolSci (University of Mannheim)
- Research on parties, legislative politics, electoral behavior
- First started programming in 2011

Contact:

- haber.matthias@gmail.com

About yourself

- Who are you?
- Why did you take this class?
- What data/programming skills would make you work life easier?

Workshop structure

Day 1 Session 1: A basic introduction to R Session 2: Programming Session 3: Tidyverse Session 4: Visualizing trends and relationships

Day 2 Session 1: Creating our own dataset Session 2: Dashboard fundamentals Session 3: Building our own dashboard Session 4: Moving deeper and further

Why data skills?

- Data skills are increasingly important for research and industry projects
- With complex data projects, however, come complex needs for understanding and communicating processes and results

The 80-20 rule

- Most data are messy
  - You spent most of your time cleaning/preparing data
  - You learn lot about the structure of your data

R

- Based on the statistical programming language S (1976)
- R was developed by Ross Ihaka and Robert Gentleman (1995)
- R was intentionally developed to be a data analysis language

Why R

- Open source: makes it highly customizable and easily extensible
- Over 7,500 packages and counting
- Used by many social scientists interested in data analysis
- Powerful tool to generate elegant and effective plots
- Command-line interface and scripts favors reproducibility
- Excellent documentation and online help resources

We will work in RStudio

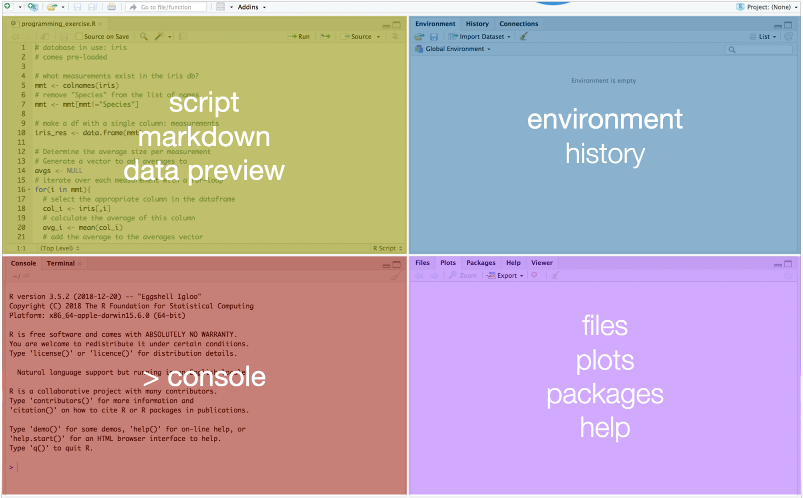
- RStudio is an Integrated Developer Environment (IDE) and serves as:
- Code editor
    - Code highlighting/completion, indentation, . . .
    - Feed code from editor to R-console
- Project manager
- Workspace viewer
- Data browser
- Enhanced output viewer
- Help browser

Install software

- R:
    - R: http://cran.rstudio.com/
    - RStudio: http://www.rstudio.org/download/daily/desktop/
- R packages
    - tidyverse (R packages designed for data science)

Session 1: A basic introduction to R

The RStudio Interface



```r
# database in use: iris
# comes pre-loaded

# what measurements exist in the iris db?
mmt <- colnames(iris)
# remove "Species" from the list of names
mmt <- mmt[mmt!="Species"]

# make a df with a single column: measurements
iris_res <- data.frame(mmt)

# Determine the average size per measurement
# Generate a vector to add averages to
avgs <- NULL
# iterate over each measurement with a for-loop
for(i in mmt){
  # select the appropriate column in the dataframe
  col_i <- iris[,i]
  # calculate the average of this column
  avg_i <- mean(col_i)
  # add the average to the averages vector
```

```
R version 3.5.2 (2018-12-20) -- "Eggshell Igloo"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Basic workflow

- Edit in code editor (.r-file)
- Paste to console
- Save Workspace/Datasets (.Rdata-file)
- Save code routinely (no auto-save!)
- Press TAB to use RStudio's autocompletion feature

Shortcuts

- Run code from editor: Select line and ctrl+Enter
- Switch between source and console: ctrl+1, ctrl+2
- Clear console: ctrl+L
- ‘Arrow up’ gives you the last line of code in the console
- Press Alt+Shift+K to see all keyboard shortcuts

Fundamentals of the R language

- Use # to comment code (will not be run)
- Case-sensitivity: `data` vs `Data`
- Assigning objects: <- and =

```r
# Assign the number 5 to an object called number
number <- 5
number
```

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

```r
# Assign the character string Hello World
string <- "Hello World"
string
```

```
## [1] "Hello World"
```

Naming

- object names must start with a letter, and can only contain letters, numbers, _ and ..
- object names should be descriptive
- Each object name must be unique in an environment.
    - Assigning something to an object name that is already in use will overwrite the object's previous contents.

```
i_use_snake_case
otherPeopleUseCamelCase
some.people.use.periods
And_aFew.People_RENOUNCEconvention
```

Functions

- Functions perform operations on the input given and end in ()
- R has a large collection of built-in functions that are called like this:

Functions

- For example, `seq()` which makes regular **seq**uences of numbers

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Operations on scalars

You can use R as a calculator:

```
2 + 3
2 - 3
2 * 3
2 / 3
```

Functions on scalars:

```
a <- 5
factorial(a)
```

```
## [1] 120
```

Exercise 1

1. Do the following calculation in R: $\frac{1+5}{9}$
2. Assign the results to a variable
3. Bonus: Round off the results to the 1 decimal

Special values in R

- `NA`: not available, missing
- `NULL`: does not exist, is undefined
- `TRUE`, `T`: logical true
- `FALSE`, `F`: logical false

Finding special values

Function	Meaning
`is.na`	Is the value `NA`
`is.null`	Is the value `NULL`
`isTRUE`	Is the value `TRUE`
`!isTRUE`	Is the value `FALSE`

```r
absent <- NA
is.na(absent)
```

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

Operations

Operator	Meaning
<	less than
>	greater than
==	equal to
<=	less than or equal to
>=	greater than or equal to
!=	not equal to
a | b	a or b
a & b	a and b

R is object-oriented

Objects are R's nouns and include (not exhaustive):

- character strings
- numbers
- vectors of numbers or character strings
- matrices
- data frames
- lists

Vectors

A vector is a container of objects put together in an order.

```
# Define a vector
a <- c(1,4,5)
b <- c(3,6,7)
```

```
# Join multiple vectors
ab <- c(a,b)
```

```
# Find vector length (number of its elements)
length(a)
```

Operations on vectors

Operation	Meaning
`sort(x)`	sort a vector
`sum(x)`	sum of vector elements
`mean(x)`	arithmetic mean
`median(x)`	median value
`var(x)`	variance
`sd(x)`	standard deviation
`factorial(x)`	factorial of a number

Exercise 2

1. Create a character vector with the names of the three people sitting closest to you. Save the vector as `name`
2. Create a numeric vector with their respective ages and save it as `age`
3. Use a funtion in R to calculate their average age?

Matrices

A Matrix is a square 2 dimensional container, i.e. vectors combined by row or column

- Must specify number or rows and columns

  ```
  matrix(x,nrow,ncol,byrow)
  ```

  - x: vector of length nrow*ncol
  - nrow: number of rows
  - ncol: number of columns
  - byrow: TRUE or FALSE, specifies direction of input

Exercise 3

Assign a 6 x 10 matrix with the sequence 1,2,3,. . . ,60 as the data. Save the matrix as `m`

Data frames

Data frames are a two-dimensional container of vectors with the same length. Each column (vector) can be of a different class and can be referenced or created with $. You can use functions like nrow(), ncol(), dim(), colnames(), or rownames() on your df.

```r
# Combine two vectors into a data frame
number <- c(1, 2, 3, 4)
name <- c('John', 'Paul', 'George', 'Ringo')
df <- data.frame(number, name, stringsAsFactors = FALSE)
df
```

```
##   number   name
## 1      1   John
## 2      2   Paul
## 3      3 George
## 4      4  Ringo
```

Exercise 4

1. Create a vector called `country` containing the names of the countries from the three people whose names you used earlier.
2. Create a data frame combining `name`, `age`and `country` and save it as `my_first_df`

Lists

A list is an object containing other objects that can have different lengths and classes.

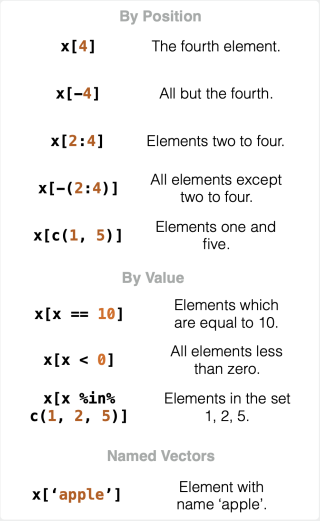
```r
# Create a list with three objects of different lengths
list1 <- list(beatles = c('John', 'Paul', 'George', 'Ringo
              alive = c('Paul', 'Ringo'), albums = 1:13)
list1
```

```
## $beatles
## [1] "John"   "Paul"   "George" "Ringo"
##
## $alive
## [1] "Paul"  "Ringo"
##
## $albums
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13
```

Exercise 5

1. Add one more person's name to `name` vector
2. Try to create a data frame called `my_second_df` and store the new `name` vector, `age` and `country` in it. See what happens and why.
3. Create a list instead of a data frame with the three objects and name it `my_first_list`

Indexing vectors



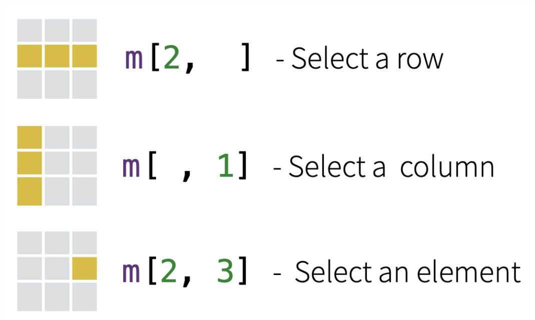
By Position	
`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.
By Value	
`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.
Named Vectors	
`x['apple']`	Element with name 'apple'.

Exercise 6

1. Return the first number in your vector `age`
2. Return the 2nd and 3rd element in your vector `name`
3. Return only ages under 30 from your vector `age`

Referencing matrices

- Like vectors, you can reference matrices by elements
- Can also reference rows/columns, these are vectors



`m[2,   ]` - Select a row

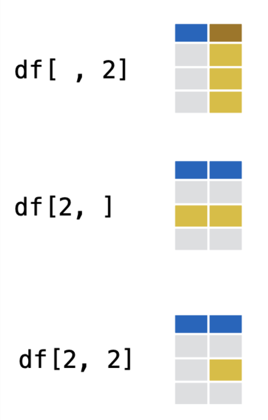
`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

Exercise 7

Extract the 9th column of the matrix from the previous problem.
How can you find the 4th element in the 9th column?

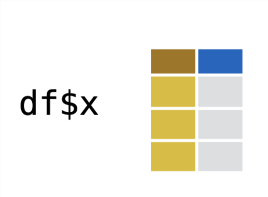
Indexing data frames



```
df[ , 2]
```

```
df[2, ]
```

```
df[2, 2]
```



```
df$x
```

Exercise 8

1. From your data frame `my_first_df`, return the entries for everyone living in a country of your choice.

Indexing lists

```
list1[1] # 1st element of the list
```

```
## $beatles
## [1] "John"   "Paul"   "George" "Ringo"
```

```
list1[[1]] # 1st content of the first element
```

```
## [1] "John"   "Paul"   "George" "Ringo"
```

```
list1[[1]][[2]] # 2nd value in the 1st content
```

```
## [1] "Paul"
```

Role of brackets

- `[]` for indexing vectors, lists, data frames. . .
- `()` for passing arguments to functions
- `{}` for defining content of loops, functions, etc.

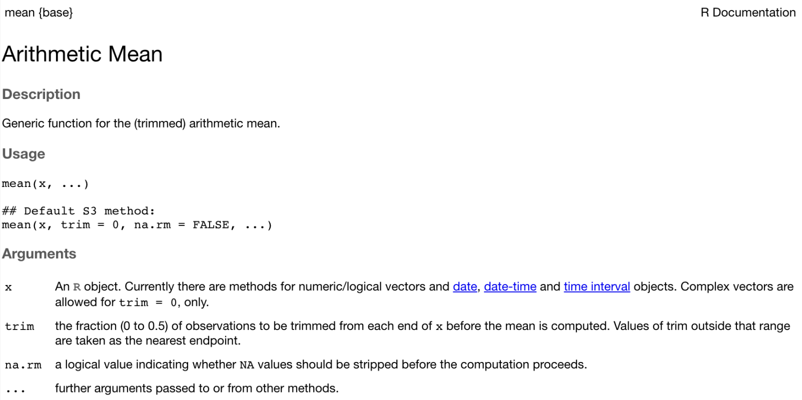
Recap types and structures

- Data types encountered so far:
  - `logical`
  - `numeric`
  - `character`
- Data structures
  - `vector` (1 dimension)
  - matrix‘ (2 dimensions)
  - `data frame` (2 dimensions)
  - `list` (*n* dimensions)
- Absent data
  - `NA` (not available)
  - `NUll` (non-existent)

Recap functions

- Functions encountered so far
  - `c()`
  - `data.frame()`
  - `mean()`
  - ...
- What if you don't know what a functions does?
  - `?mean()` to get help for a function
  - help.search('weighted mean') to get help for a concept

Help



mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

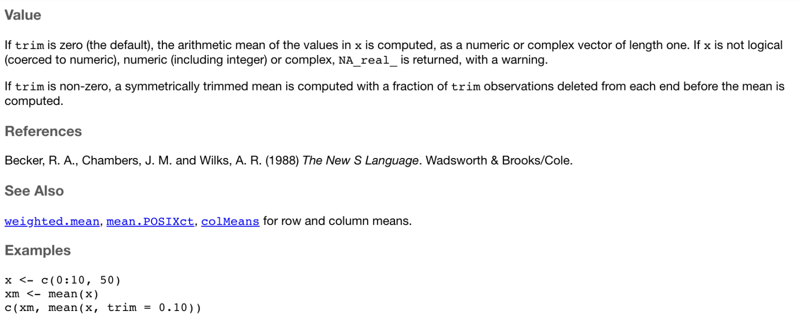
`x` An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.

`trim` the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

`na.rm` a logical value indicating whether `NA` values should be stripped before the computation proceeds.

`...` further arguments passed to or from other methods.

Help



Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

R’s build-in data sets

There are a number of example data sets available within `R`.

```
# List internal data sets:
data()
```

```
# Load swiss data set:
data(swiss)
# Find data description:
?swiss
```

Session 2: Messing with data

Datacamp

- Complete the (free) course on introduction to R: https://www.datacamp.com/courses/free-introduction-to-r

That's it for today. Questions?