Getting started with R

Abhijit Dasgupta

March 25-27, 2019

R is a language

A programming language

- You learn and use languages to communicate with people and cultures
    - French, Farsi, Korean, Hindi, Swahili
- A programming language is a **language** first and foremost
    - Meant to communicate with a computer
    - Has logical structure
    - Is typically precise (since computers are literal)
- R is a *programming language* meant to interface with data
    - Domain Specific Language

R has a grammar

- **Objects**: These are the *nouns*. We will act on objects to create new objects. Each object has a *name* which we will treat as the nouns in our code.
- **Functions**: These are the *verbs*. Functions will act on objects to create new objects.
- **The `%>%` operator**: This acts like the conjunction "then" to create "sentences" called *pipes* or *chains*.
- **Optional function arguments**: These are adverbs which modify the action of the function (verb).

> Start with an object (noun), successively act upon it with functions (verbs) to create another object (noun) that is useful to you in some way.

R is modular

- R has a base set of functions that you install
- You add on to this with other modules called *packages* to enhance functionality
- After you've installed a package, you have to activate it when you need it with a function called `library`

```
library(tidyverse)
```

R is case-sensitive

Humans can read this:

> Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteers be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

Computers cannot!!

Computers are extremely literal

R is case-sensitive

- Spelling and case matter
- White space doesn't matter
- R doesn't have a signal to end a statement (C and Java have `;`)
    - Bit more forgiving

Let's see some code

```
library(tidyverse)

mtcars1 <- mtcars %>% as_tibble() %>% rownames_to_column(var = 'cars')
head(mtcars1,5)
```

```
## # A tibble: 5 x 12
##   cars    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1      21       6   160   110  3.9   2.62  16.5     0     1     4     4
## 2 2      21       6   160   110  3.9   2.88  17.0     0     1     4     4
## 3 3      22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
## 4 4      21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
## 5 5      18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
```

```
cars_summary <- mtcars1 %>%
  group_by(cyl) %>%
  summarize(mpg = mean(mpg)) %>%
  arrange(desc(mpg))
cars_summary
```

```
## # A tibble: 3 x 2
##     cyl   mpg
##   <dbl> <dbl>
## 1     4  26.7
## 2     6  19.7
## 3     8  15.1
```

```r
library(tidyverse)

mtcars1 <- mtcars %>% as_tibble() %>% rownames_to_column(var = 'cars')
```

- start with a noun (`mtcars`)
- apply a verb to it (`as_tibble`)
    - this creates a new noun
- apply a verb to the new noun (`rownames_to_columns`)
    - modify the verb by an optional argument (`var = 'cars'`)
    - this creates a new noun
- assign a name to this noun (`mtcars1`)

```
mtcars1
```

- Call an object (noun) to see it

```
cars_summary <- mtcars1 %>%
  group_by(cyl) %>% # split by levels of cyl
  summarize(mpg = mean(mpg)) %>% # compute the mean mpg at each level
  arrange(desc(mpg)) # re-arrange in descending order of mpg
cars_summary
```

```
## # A tibble: 3 x 2
##     cyl   mpg
##   <dbl> <dbl>
## 1     4  26.7
## 2     6  19.7
## 3     8  15.1
```

- The actual arrangement doesn't matter as long as `%>%` is at the end
- Any text on a line beginning with `#` is ignored as a comment

Try to make your code human-readable

- The functions here, from the `dplyr` package, are English-comprehensible
- Not all functions are, but this kind of attention to detail is very nice

Naming things

Conventions

- A syntactically valid name consists of letters, numbers and the dot or underline characters and starts with a letter or the dot not followed by a number
- Don't use `$`, `@`, `|`, and math symbols as they have other uses
- Make your names expressive, but not complicated
  - Don't use `data1`, `model2`
  - Do use `staffing_data`, `linear_model_height`
- Remember, the next person to see your code will probably be you
  - You can't "phone a friend", since that friend is your past self
  - You'll be left scratching your head about what you wrote
  - Been there, done that.

Conventions

- I like `pothole_case`, i.e. joining words with underscores
    - Traditionally you'd join with `.`, but this is more readable to me
- Some people like `CamelCase`
- You can make different objects just by changing case
    - `staffing_data`, `Staffing_data`, `Staffing_Data`,`StaffingData`, `staffing.data`, `staffing_Data` can all be unambiguously different objects
    - Create your own system to figure out how to name things

> Finding a good name is hard, but often worth the effort

Writing code

- You want to create a "story" for the data using R
- Scripts make the story reproducible and verifiable and transferable to other data
- Your first scripts will be sloppy
- Think about the writer; lot's of things in the trash
- With practice, this will get smoother

R Objects

Objects

- Everything in R is an object
- There are two broad classes of objects: data objects (nouns) and functions (verbs)

Data Objects

- `data.frame` or `tibble`: These are rectangular data sets much like you would see in a spreadsheet
- `vector`: This is a 1-dimensional list of numbers or strings (words in the language sense), but all must be of the same kind (number or string)
- `matrix`: This is a 2-dimensional list of numbers or strings, once again all of the same type
- A single number or word
- `list`: This is a catch-all bucket. Each element of a list can be literally any valid R object. So they could be `tibble`'s, or functions, or matrices, and different elements can be of different types.

Data objects

- Most objects we'll use in this workshop are going to be `data.frame` or `tibble` objects.
    - In case you're wondering, they're basically the same thing, but `tibble`'s have some modest additional functionality
- R comes with a bunch of built-in datasets stored as `data.frames`.

Data Frames

```
as_tibble(mtcars)
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
##  2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
##  3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
##  4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
##  5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
##  6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
##  7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
##  8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
##  9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
## 10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
## # … with 22 more rows
```

- We have columns which are variable
- Rows are observations
- You can see what kind of variable each column is (in a `tibble`)

Characteristics of data frames

```
dim(mtcars)
```

```
## [1] 32 11
```

```
rownames(mtcars)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28"
## [29] "29" "30" "31" "32"
```

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Characteristics of data frames

- Each of the calls on the previous slide produce *bona fide* objects in R.
- You can assign names to these objects to store them for future use.

```
car_names <- rownames(mtcars)
```

Extracting elements

Data frames act like matrices

```
mtcars[3, 4] # extracts from 3rd row, 4th column
```

```
mtcars[,4] # extracts 4th column
```

```
mtcars[3,] # extracts 3rd row
```

Each of these are, in turn, R objects, so you can assign names to them to store.

Extracting elements

We can see the overall structure of the data frame

```
str(mtcars)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

- `data.frame` with 32 rows and 11 colums
- each column is a variable
- each variable is numeric

Extracting elements by name

You can extract columns out by name in 3 ways

- `mtcars[,'mpg']` (matrix notation)
- `mtcars$mpg` (a shortcut, allows tab-completion)
- `mtcars[['mpg']]` (list notation)

A `data.frame` is really a `list`, so list extractions using `[[]]` work, either by index or by name.

The first and third options allow for extracting more than one column

```r
mtcars[,c('mpg','hp')]
mtcars[[c('mpg','hp')]]
```

The `c()` function stands for *concatenate*, and creates vectors.

Exercise

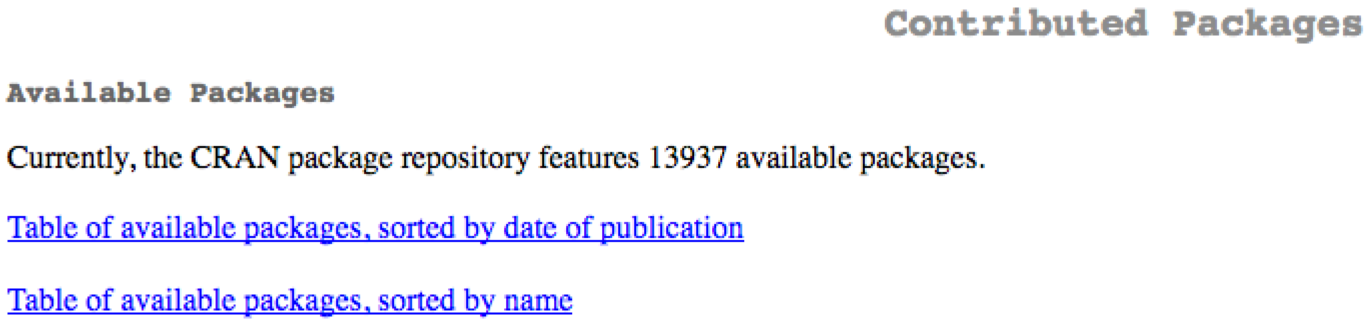
Fisher's iris dataset is in-built in R with the name `iris`.

1. Determine how many observations and variables are in the dataset
2. What are the variable names?
3. What are the row names?
4. Extract the sepal length and petal widths out and save them in new objects

R packages

Packages

- The power of the R ecosystem comes from packages



Contributed Packages

Available Packages

Currently, the CRAN package repository features 13937 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

CRAN is the Comprehensive R Archive Network, the central repository of R packages

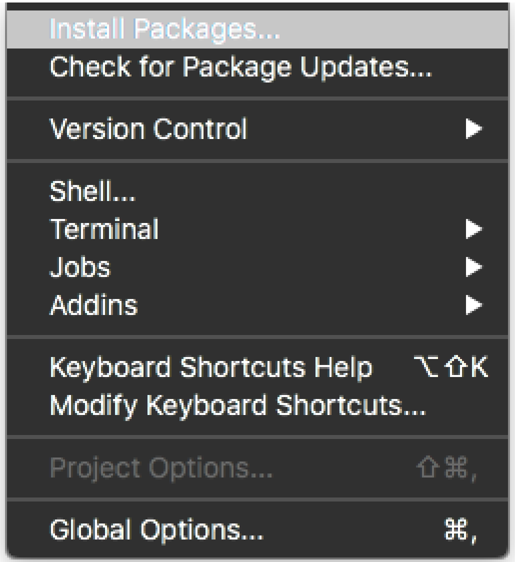
- CRAN has strict software criteria and testing to ensure usability (though not correctness)
- Packages may also reside on Github, or other curated repositories like Bioconductor

Finding packages

Finding packages

- R-Bloggers
- Twitter #rstats
- RSeek

Installing R packages



```
install.packages(<package name>, repos='https://cran.rstudio.com')
```

You can set the default repository in RStudio using `Tools > Global Options`.

Installing R packages

The Packages pane

Exercise

Install the `rio` package using any of the methods mentioned

The `tidyverse` meta-package

Includes

- readr (reading data from text files)
- tidyr (Manipulation, pivoting)
- dplyr (summarize, aggregate, filter)
- ggplot2 (visualization)
- purrr (functions applied across data structures, meta-programming)
- stringr (string manipulation)
- forcats (categorical data)

Importing data

Data

R can access data files from a wide variety of sources. These include

1. Text files (csv, tsv, fixed-width)
2. Microsoft Excel files
3. Microsoft Access databases
4. SQL-based databases (MySql, Postgresql, SQLite, Amazon Redshift)
5. Enterprise databases (SAP, Oracle)

The `rio` package

The `rio` package wraps many other packages to make importing and exporting data easy

It is great for importing and exporting non-database files that sit either on your computer or on the internet

Importing the data will create an object called a data.frame, but if you just import data, it is not saved since it doesn't yet have a name.

```r
library(rio) # activate the package
import('Data/FSI/HR_Data.csv') # can use single or double quotes
```

So every time you import data, you have to name it. You do this using the <- operator.

```r
library(rio)
hr_data <- import('Data/FSI/HR_Data.csv')
```

Checking out the data

```
head(hr_data)
```

```
##                                                  Bureau Gender Grade
## 1    Comptroller and Global Financial Services (CGFS) female   N/A
## 2                East Asian and Pacific Affairs (EAP) female   N/A
## 3                 Overseas Buildings Operations (OBO)   male  FS-5
## 4         Conflict and Stabilization Operations (CSO)   male   N/A
## 5                               Consular Affairs (CA) female  FS-5
## 6 Management Policy, Rightsizing and Innovation (PRI) female  FS-2
##             Name
## 1  Katrina Lilly
## 2          Keene
## 3 Garrett Murphy
## 4     Jim Rhodes
## 5    Anita Myers
## 6 Vivian Einhorn
##                                                                                            Skills
## 1                                   Hydrology, Research, Design, human resources, Administration
## 2                                                                           Sharepoint, Planning
## 3                interagency, Portuguese, Management, Foreign Policy, Economics, Human Resources
## 4                   education, seo, German, Finance, design, portuguese, disease response, Excel
## 5 Healthcare, training, German, french, Sharepoint, Marketing, Data Analysis, Economics, spanish
## 6                 data analysis, Web Development, Hydrology, IT, SEO, Disease Response, Japanese
##   YearsService
## 1           16
## 2           21
## 3            5
## 4            4
```

Checking out the data

```
View(hr_data)
```

Finer control of CSV imports

```
hr_data <- import('Data/FSI/HR_Data.csv', check.names = TRUE)
```

This ensures that the names of the variables are proper

```
hr_data <- import('Data/FSI/HR_Data.csv', check.names = TRUE, dec = ',')
```

This allows European data to be correctly entered.

Finer control of Excel imports

You can specify sheet names or sheet positions for import from an Excel file. If you know the sheet name, you can specify it using the `which` option:

```
dos_data <- import('Data/FSI/simulatedDOS.xlsx', which='Staffing_by_Bureau')
```

You can also grab the same sheet by position:

```
dos_data <- import('Data/FSI/simulatedDOS.xlsx', which = 2)
```

> See the help file for `import` by typing `?import` in the console or searching in the Help pane

Importing from databases

Access databases

```
library(RODBC) # activate package, case-sensitive
channel <- odbcConnectAccess('C:/Documents/Name_of_Access_Database') # change to your
mydata <- sqlQuery(channel, paste("select * from Name_of_table_in_database"))
```

SQL-based databases

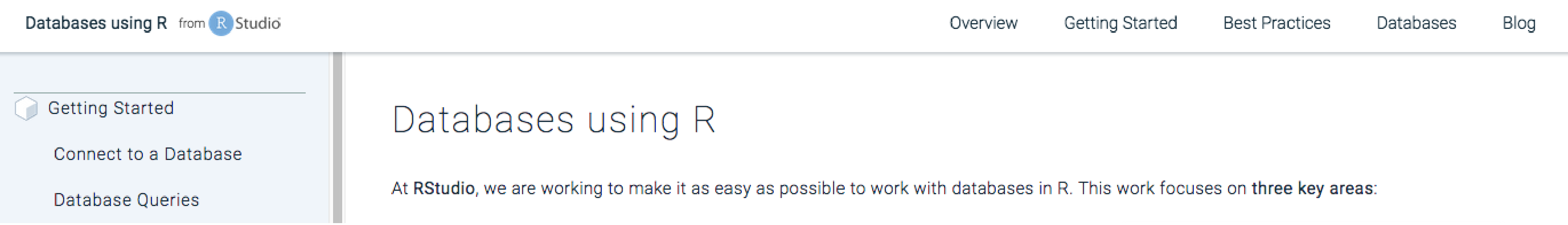
```
library(odbc)
con <- dbConnect(odbc(),
                 Driver   = "[your driver's name]",
                 Server   = "[your server's path]",
                 Database = "[your database's name]",
                 UID      = rstudioapi::askForPassword("Database user"),
                 PWD      = rstudioapi::askForPassword("Database password"),
                 Port     = 1433)
```

and you can load a table into R using

```
dat <- dbGetQuery(con, 'select * from <table name>')
```

Reading from databases

RStudio's tutorial



Databases using R

At **RStudio**, we are working to make it as easy as possible to work with databases in R. This work focuses on **three key areas**: