A Handbook on R

Freddy Ray Drennan

2021-11-22

Introduction

A Dose of R

Let's solve a problem using R. Suppose we have a friend that is interested in the current trend regarding COVID-19 cases. Where would we start? The first question is always going to be, "How do I obtain data for my analysis?" In the case for COVID-19, the answer is someone has already done that for you in the form of an R package. How do I know? I Googled it `covid19 R package` and found this link COVID19 Datahub. Often it's as easy as a Google search. But there will frequently be times that you need a more customized data aggregation process.

In my own work, I often run across APIs. API stands for Application Programming Interface. An API is nothing more than a way of working with an abstraction of some complicated code that I don't really need to know in order for it to work for me. For example, I worked at an advertising agency. This company creates marketing campaigns for small to medium sized businesses. In order to determine the effectiveness of of our advertising campaings and move money between them to maximize effectiveness, we needed data. So in order for us to sell advertisements, we have to do business with a company that is willing to share data with us. So we at Adtaxi used the Google Adwords API, Trade Desk API, and Facebook API in order to aggregate data about the performance of our ads. We determined what metrics would be similar to measure across the different channels and then ran an optimization that would reallocate budget to the preferred channels.

APIs for other things exist as well. For example, Tensorflow is a library written in C++ for building deep learning models. However, there are certain workflow that are commonly recreated within Tensorflow that it makes sense to abstract that code even more, so that building deep learning models is more plug and play and experimental, than an exercise in deep thought. We like reducing cognitive load.

The first thing I notice when I come to this link is `install.packages("COVID19")` right in the front of the page. `install.packages` is called a function. A

function *does things* given a set of inputs. Remember the equation $y = 2x + 3$ or something like it? What did we say? We said something like - given $x = 2$ then $y = 7$. We put in a 2 for x, $2(2) + 3$ and then say $y = 7$. When we got to calculus, equations became functions with a slightly different look. $f(x) = 2x + 3$. What happened to the y? Well, nothing really except for the way we describe things. We can say $y = f(x)$ or $result = f(x)$ or $z = g(x)$ or anything we like really. Logic allows us to describe things however we want, so long as we are logical. In R, we can describe the relationships above like so -

```
# Same as f(x) = 2x + 3
f = function(x) {
    2 * x + 3
}

f(x = 2)
```

```
## [1] 7
```

The components of the function above are

1. The name of the function f

We now have a way of describing inputs and output a little more clearly. f is the *name* of the rule that we apply to x. f contains the instructions to turn x into something else, often called y. We're just taking note of the fact that x is the only thing that the function needs to apply it's logic. We could have $f(x, y, d) = x^2 + y^2 + d$ and if we wrote $f(2, 4, 5)$ then we would have a shorthand way of describing the whole - $2^2 + 4^2 + 5$. If we can write $f(2, 4, 5)$ instead of $f(2, 4, 5) = 2^2 + 4^2 + 5$ that would be quite convenient.

Functions - * A So in your RStudio console, you can write the following to install the `COVID19` package.

```
install.packages('COVID19')
```

I'm a man of few words. So let's get to it. The two lines of code below consist of three functions. The first two are `library` and `covid19`, but the third is hidden, and I will disclose where shortly. *Functions* are spaces for stuff to happen. Functions help us make common procedures repeatable. By creating a function with a particular name and inputs, we can get some sort of useful (or not useful, the world's your oyster) output.

In this case, `library` loads *packages* from a folder in the R environment called library. You can see which ones your R environment knows about by running the function `.libPaths()`. `covid19` is a function from the `COVID19`

package, and would only be available after executing `library(COVID19)` or if `library(COVID19)` is omitted, by pulling it from the package namespace directly by preceding the function with the package name and two colons like so: `COVID19::covid19`. Generally speaking, you simply use `library` because it reduces the amount of text on the page.

```
library(COVID19)
```

```
## Warning: package 'COVID19' was built under R version 4.1.2
```

```
covid_data <- covid19(
    country = 'United States',
    start = '2021-01-01',
    end = "2021-11-21",
    verbose = FALSE
)
```

How do I know if a function is vectorized

Initial Setup

Book Outline

- Install R
- Install R Studio
- Windows Only: Install RTools
  - When installed, run in the RStudio Console: `write('PATH="${RTOOLS40_HOME}\\usr\\bin;${PATH}"', file = "~/.Renviron", append = TRUE)`
- Windows Only: Install WSL2
  - Computer should be completely updated before install.
- Install Git
- Create Github Account
- Fork r-handbook
- Install Docker and Docker Compose
- Create AWS Account

  - Billing will be discussed in the course, but don't expect to pay much - i.e., 10-20 dollars a month for high course activity.
  - Remember to `stop` EC2 servers when we begin using them. AWS is polite about your first few refund requests.

- Create Reddit Account
  - Follow Instructions here

Make sure you install the `tidyverse` packages. Update to renv later.

```
install.packages('tidyverse')
```

What is R

Types of Problems You Can Solve

Base R, Tidyverse, data.table

Arguments/ Developments within the language

What are Variables

Valid Variable Names

Building Blocks

Vectors

Vectors are containers information of similar type. You can think of them as having $1 * n$ cells where n is any positive integer, and make up the rows and columns of tables. Vectors always contain the same type of value. R has many different types of vectors, but the most common are **numeric**, **character**, and **logical** (**TRUE**/**FALSE**).

Vectors are cool. I like to think of them as boxes that can only be stacked on top of one another.

```
typeof(c(TRUE))
```

```
## [1] "logical"
```

```
typeof(c(TRUE, 1))
```

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

```
typeof(c(TRUE, 1, 'a'))
```

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

Functions

Functions are containers where anything or nothing can happen, but whatever happens, it happens the same way every single time. They allow for generalization of complicated ideas and routines that we wish to repeat over and over again. A function may have an input, but no output. It may have an output, but no input, both or none. If it's something you need to do repeatedly, or

containing code makes your program easier to read, then write a function for that process.

Rule 4: Functions have inputs, outputs, and a body. A function can have multiple outputs, but given a particular set of inputs, the solution should never change assuming you are not developing a function with randomness built in.

R has a built-in constant called `letters`. This means that no matter where you are writing R, `letters` will be available to you. We see that `letters` is a **character vector** in our program below, and use the composition of functions to create a program that describes `letters`.

```
print(letters)
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

Next, we can use some functions which take in pretty much any object that exists in R and spits back information regarding the `letters` data.

```
main <- function() {
  print_information <- function(x) {

    variable_name = deparse1(substitute(x))

    length_x = length(x)
    typeof_x <- typeof(x)
    is_vec_x <- is.vector(x)

    meta_list <- list(
      length = length_x,
      type = typeof_x,
      is_vector = is_vec_x
    )

    cli::cli_alert('Information about {variable_name}')

    cli::cli_alert_info("{variable_name} is a 1x{length_x} dimensional")
    cli::cli_alert_info("")

    purrr::iwalk(meta_list, function(x, index) {
      cli::cli_alert_info(glue::glue('{index} {x} is type {typeof(x)}'))
    })

    return(meta_list)
```

```
  }

  cli::cli_alert_info('Execute print_information')
  output <- print_information(mtcars)
  cli::cli_alert_success('Execute print_information complete')

  print(output)
}

main()
```

```
## i Execute print_information

## > Information about mtcars

## i mtcars is a 1x11 dimensional

## i

## i length 11 is type integer

## i type list is type character

## i is_vector FALSE is type logical

## v Execute print_information complete

## $length
## [1] 11
##
## $type
## [1] "list"
##
## $is_vector
## [1] FALSE
```

Debugging

What is the debugger?

How to learn R without knowing any R

`browser()`

`next`, `continue`

`debug` and `undebug`

`debugonce`

Understanding debugging output

LOTS OF DEBUGGING EXERCISES CANNOT STRESS ENOUGH

Vectors

`c`

`[` and `[[`

- Vectors
  - atomic
- Strings
  - Base R
  - `stringr`
    - Regular Expressions
  - Cheat Sheet
- Numbers
  - Integer
  - Double
- Factors
  - `as.factor` vs. `as_factor`
- Dates
  - Base R
  - `lubridate`

Lists

`list`

`[` and `[[`

- Lists
  - `list()` and `c`
  - `[` and `[[`
  - Connection between lists and json
    - `jsonlite`

Tables

`c`

`[` and `[[`

- Tables
  - matrices
  - `data.frame` vs `tibble`
  - data.frames are lists with equal length, atomic vectors

Functional Programming

2. Functions
    - Sequences
    - Mapping functions
    - pipes
    - void
    - `return`
        - Can a function return nothing?
        - What are side effects?
        - Multiple return statements

Base R

`apply`, `lapply`, `mapply`

Modern R

`purrr` * `map_*` * `map2_*` * `pmap_*` * Iterate over What? * Why are data.frames mapped over columnwise? * A: data.frames are lists, and mapping functions will iterate over each individual item in a list

Tidy Data

- Concept of tidy data
  - Tidy Data Paper
- `tidyr`
  - `pivot_longer`
  - `pivot_wider`

dplyr

- `dplyr` and data manipulation
  - main functions
    - `select`
    - `mutate`
    - `filter`
    - `transmute`
  - summarizing data
    - `group_by`
    - `summarize` - one row per group
    - `mutate` - one or many rows per group will have same value
    - `ungroup` - remove grouping
      - Not everything has to be a `group_by`
      - Solving group problems with vectors

- Joining Tables
  - `inner_join`
  - `full_join`
  - `left_join` / `right_join`

Project Outline

To be expanded over many chapters

1. Windows vs Mac vs Linux
2. Docker Installation
    - Windows needs to set up VM in bios
3. RStudio IDE
    - Cheat Sheet
4. reddit api creds
5. reticulate
    - Enough R to know Python
        - Type Conversions
    - miniconda installation
    - virtual environments
6. Package Structure
    - Defaults for RStudio
        - Rebuild and Restart with Roxygen2
    - `.env`
    - `.gitignore`
    - `.Rprofile`
    - `.Renviron`
    - Packages necessary for efficient development
        - usethis
        - roxygen
        - devtools
            * Cheat Sheet
    - Make and Makefiles
        - Automating Package Build
    - Unit Testing (probably bad location for ut, no code written)

    - testthat

7. Git
    - Github
    - git circle, workflow
8. Retrieving Data from API
    - `praw`
    - `dotenv` and `.env`
    - Old Reddit Code to start with
9. Docker and Docker Compose Introduction
    - `.dockerignore`
10. Create Postgres Database
    - What are Ports?
    - Postgres Credentials
11. Create functions for Storing Reddit Data
12. *Need preferred method for streaming data, i.e., Airflow not a good scheduler for scripts that are always running* and need a kickstart on failure, timeout, etc. Docker with `restart: always` may be sufficient
13. Plumber API
    - Add to docker-compose
    - Functions for ETL, Shiny Application
14. ETL with Airflow and HTTP operator connected to Plumber API
15. Shiny
    - Reactive Graph
        - Order does not matter, the graph does
    - Why Modules?
        - `map` over modules
16. Automating Infrastructure
    - `awscli`
    - `boto3`
        - `biggr`
    - Create EC2 Server from R
        - User Data