Data Acquisition in R

Emanuele Guidotti

Contents

Files

A comma-separated values (CSV) file is a delimited text file that generally uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by the delimiter. CSV is a common data exchange format that is widely supported by consumer, business, and scientific applications. R makes it easy to export and import data in CSV format.

Local Files

Export data to a csv file

```
data("mtcars")                          # load the mtcars dataset
write.csv(mtcars, file = 'mtcars.csv')  # export to file
```

Import data from a csv file

```
x <- read.csv('mtcars.csv')             # read file
head(x)                                 # print data
```

```
##                   X  mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1         Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## 2     Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## 3        Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## 4    Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## 5 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

```
## 6           Valiant 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Remote Files

Some data providers offer data in csv format on their website. The STOXX website, a financial index provider, is one of these. Open this link for the EURO STOXX 50 Index: tab *Data -> Historical Data* provides some open source files for histroical prices. Clicking on *EUR Price* will open this link. The `read.csv()` function can read this file directly from the internet.

```
# read.csv is very flexible. For the full list of arguments type ?read.csv
x <- read.csv('https://www.stoxx.com/document/Indices/Current/HistoricalData/h_3msx5e.txt', sep = ';')
head(x)
```

```
##         Date Symbol Indexvalue  X
## 1 03.03.2025   SX5E    5540.69 NA
## 2 04.03.2025   SX5E    5387.31 NA
## 3 05.03.2025   SX5E    5489.12 NA
## 4 06.03.2025   SX5E    5520.47 NA
## 5 07.03.2025   SX5E    5468.41 NA
## 6 10.03.2025   SX5E    5386.98 NA
```

```
rownames(x) <- as.Date(x[,1], format = '%d.%m.%Y') # assign rownames
x[,c(1,ncol(x))] <- NULL                           # drop the first and last column
head(x)                                            # print data
```

```
##            Symbol Indexvalue
## 2025-03-03   SX5E    5540.69
## 2025-03-04   SX5E    5387.31
## 2025-03-05   SX5E    5489.12
## 2025-03-06   SX5E    5520.47
## 2025-03-07   SX5E    5468.41
## 2025-03-10   SX5E    5386.98
```

R Packages

The 'quantmod' Package

The `quantmod` package provides a very suitable function for downloading **financial data** from the web. This function is called `getSymbols`. The function works with a variety of sources.

```
# install the package
install.packages('quantmod')
```

```
# load the package
require(quantmod)
```

For stocks and shares, the `yahoo` source is used. Symbols can be found here.

```
# retrieve Facebook quotes
x <- getSymbols(Symbols = 'META', src = 'yahoo', auto.assign = FALSE)
tail(x)
```

```
##            META.Open META.High META.Low META.Close META.Volume META.Adjusted
## 2025-05-21    631.79    646.61   630.17     635.50    11464600        635.50
## 2025-05-22    634.05    643.25   630.71     636.57     8228400        636.57
## 2025-05-23    624.00    632.45   622.65     627.06     8454100        627.06
## 2025-05-27    635.41    643.08   632.75     642.32     9508400        642.32
```

```
## 2025-05-28    642.60    650.88   642.55     643.58     9042900        643.58
## 2025-05-29    651.65    653.32   639.50     645.05     8858800        645.05
```

For currencies and metals, the `oanda` source is used. Symbols are the instruments' ISO codes separated by `/`. ISO codes can be found here.

```
# retrieve the historical euro/dollar exchange rate
x <- getSymbols(Symbols = 'EUR/USD', src = 'oanda', auto.assign = FALSE)
tail(x)
```

```
##            EUR.USD
## 2025-05-24 1.13646
## 2025-05-25 1.13655
## 2025-05-26 1.13900
## 2025-05-27 1.13568
## 2025-05-28 1.13078
## 2025-05-29 1.13114
```

For economics series, the `FRED` source is used. Symbols can be found here.

```
# retrieve the historical Gross Domestic Product for Japan
x <- getSymbols(Symbols = 'JPNNGDP', src = 'FRED', auto.assign = FALSE)
tail(x)
```

```
##             JPNNGDP
## 2023-07-01 594243.2
## 2023-10-01 596148.9
## 2024-01-01 595852.8
## 2024-04-01 608723.0
## 2024-07-01 613017.6
## 2024-10-01 619965.5
```

RESTful APIs

An Application Program Interface (API) is basically a messenger that takes a request, tells a system what you want to do and then returns the response back to you. A RESTful API is an API that uses HTTP requests to GET, PUT, POST and DELETE data. The `httr` R package is a useful tool for working with HTTP. Each API has its very specific usage and documentation.

```
# install the package
install.packages('httr')
```

```
# load the package
require(httr)
```

CRAN downloads

The API of the CRAN downloads database. Documentation available here

Example. Which was the most downloaded package of the last month?

```
baseurl <- 'https://cranlogs.r-pkg.org/'         # API base url. See documentation
endpoint <- 'top/'                               # API endpoint. See documentation
period <- 'last-month/'                          # API parameter. See documentation
count <- 1                                       # API parameter. See documentation
url <- paste0(baseurl, endpoint, period, count) # build full url
x <- GET(url)                                    # retrieve url
```

```r
data <- content(x)                                  # extract data
data                                                # print data
```

```
## $start
## [1] "2025-04-28T00:00:00.000Z"
##
## $end
## [1] "2025-05-27T00:00:00.000Z"
##
## $downloads
## $downloads[[1]]
## $downloads[[1]]$package
## [1] "rlang"
##
## $downloads[[1]]$downloads
## [1] "1849909"
```

The most downloaded package between **2025-04-28** and **2025-05-27** was **rlang** with a total of **1849909** downloads.

KuCoin API

The API of KuCoin, cryptocurrency exchange. Documentation available here

Example. Retrieve and plot Bitcoin price every minute in the last 24 hours.

```r
# set GMT timezone. See documentation
Sys.setenv(TZ='GMT')
# API base url. See documentation
baseurl <- 'https://api.kucoin.com'
# API endpoint. See documentation
endpoint <- '/api/v1/market/candles'
# today and yesterday in seconds
today <- as.integer(as.numeric(Sys.time()))
yesterday <- today - 24*60*60
# API parameters. See documentation
param <- c(symbol = 'BTC-USDT', type = '1min', startAt = yesterday, endAt = today)
# build full url. See documentation
url <- paste0(baseurl, endpoint, '?', paste(names(param), param, sep = '=', collapse = '&'))
# retrieve url
x <- GET(url)
# extract data
x <- content(x)
data <- x$data
# formatting
data <- sapply(1:length(data), function(i) {
  # extract single candle
  candle <- as.numeric(data[[i]])
  # formatting. See documentation
  return( c(time = candle[1], open = candle[2], close = candle[3], high = candle[4], low = candle[5]) )
})
# convert to xts
datetime <- as.POSIXct(data[1,], origin = '1970-01-01')
data <- xts(t(data[-1,]), order.by = datetime)
# plot closing values
```

```r
plot(data$close, main = 'Bitcoin price in dollars')
```

Web Scraping

Web scraping is a technique for converting the data present in unstructured format (HTML tags) over the web to the structured format which can easily be accessed and used. The `rvest` package is a useful tool to scrape information from web pages.

```r
# install the package
install.packages('rvest')
```

```r
# load the package
require(rvest)
```

Example. Write a function to retrieve articles from Google Scholar given a generic query string `q`.

```r
getArticles <- function(q){
  # build url
  url <- paste0('https://scholar.google.com/scholar?hl=en&q=', q)
  # sanitize url
  url <- URLencode(url)
  # get results
  res <- read_html(url) %>%           # get url
    html_nodes('div.gs_ri h3 a') %>%  # select titles by css selector
    html_text()                       # extract text
  # return results
  return(res)
}
```

```r
# retrieve articles about web scraping in r
getArticles('web scraping in r')
```

```
##  [1] "Web scraping using R"
##  [2] "Automated data collection with R: A practical guide to web scraping and text mining"
##  [3] "Web scraping in the R language: A tutorial"
##  [4] "R Web Scraping Quick Start Guide: Techniques and tools to crawl and scrape data from websites"
##  [5] "RCrawler: An R package for parallel web crawling and scraping"
##  [6] "Web Scraping With R"
##  [7] "Research note: Scraping financial data from the web using the R language"
##  [8] "Automated data collection with Ra practical guide to web scraping and text mining"
##  [9] "Web scraping in the statistics and data science curriculum: Challenges and opportunities"
## [10] "Challenges in Geocoding: An Analysis of R Packages and Web Scraping Approaches"
```

Code Download

Download the full code to generate this document and reproduce the examples. The file is in R Markdown, format for making dynamic documents with R. An R Markdown document is written in markdown, an easy-to-write plain text format, and contains chunks of embedded R code.
Download: https://storage.googleapis.com/emanueleguidotti/R/data-acquisition.Rmd