Getting started with R, Essentials of the R language

Dhafer Malouche

كلية الآداب والعلوم
College of Arts and Sciences
QATAR UNIVERSITY جامعة قطر

Outline

Downloading and Installing `R` and `RStudio`

R objects

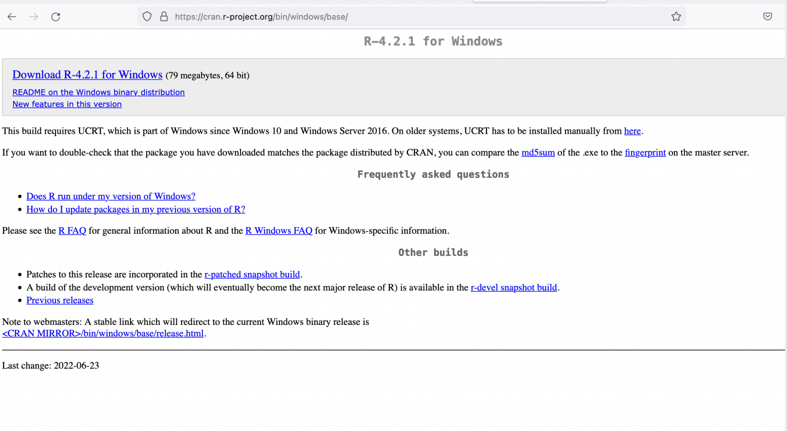
Importing/Exporting Data

Downloading and Installing R and RStudio

R and RStudio

- Downloading R:
  https://cran.r-project.org/bin/windows/base/
- Downloading RStudio:
  https://www.rstudio.com/products/rstudio/download/.
  Click on “free version”.

R for Windows users



R-4.2.1 for Windows

Download R-4.2.1 for Windows (79 megabytes, 64 bit)

README on the Windows binary distribution

New features in this version

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from here.

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

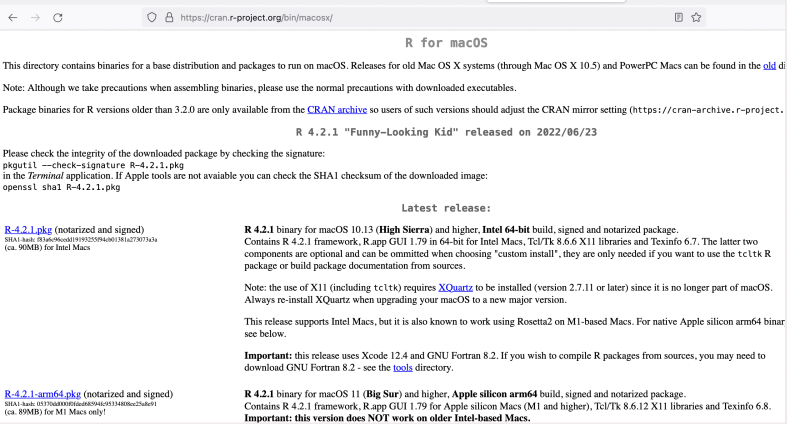
Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.html.

Last change: 2022-06-23

R for Mac users


https://cran.r-project.org/bin/macosx/

R for macOS

This directory contains binaries for a base distribution and packages to run on macOS. Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the old di

Note: Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

Package binaries for R versions older than 3.2.0 are only available from the CRAN archive so users of such versions should adjust the CRAN mirror setting (`https://cran-archive.r-project.`

R 4.2.1 "Funny-Looking Kid" released on 2022/06/23

Please check the integrity of the downloaded package by checking the signature:
`pkgutil --check-signature R-4.2.1.pkg`
in the *Terminal* application. If Apple tools are not avaiable you can check the SHA1 checksum of the downloaded image:
`openssl sha1 R-4.2.1.pkg`

Latest release:

R-4.2.1.pkg (notarized and signed)
SHA1-hash: f83a6c96cedd19193255f94cb01381a273073a3a
(ca. 90MB) for Intel Macs

R 4.2.1 binary for macOS 10.13 (**High Sierra**) and higher, **Intel 64-bit** build, signed and notarized package.
Contains R 4.2.1 framework, R.app GUI 1.79 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 6.7. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires XQuartz to be installed (version 2.7.11 or later) since it is no longer part of macOS. Always re-install XQuartz when upgrading your macOS to a new major version.

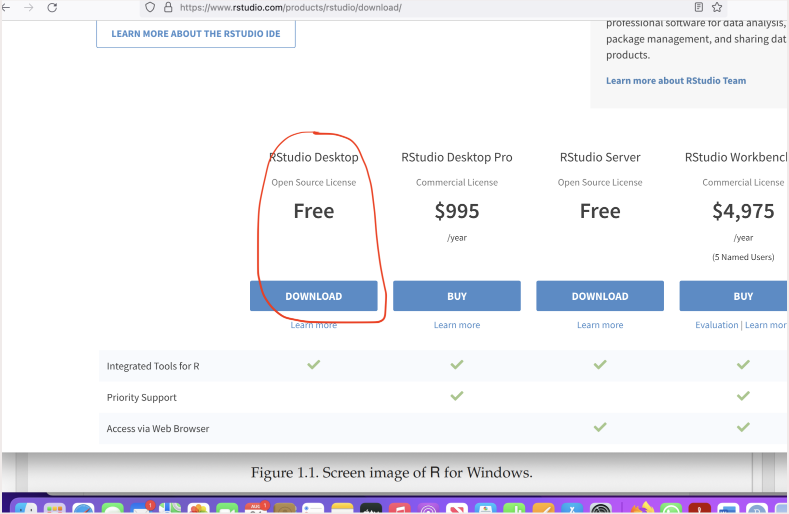
This release supports Intel Macs, but it is also known to work using Rosetta2 on M1-based Macs. For native Apple silicon arm64 binar see below.

Important: this release uses Xcode 12.4 and GNU Fortran 8.2. If you wish to compile R packages from sources, you may need to download GNU Fortran 8.2 - see the tools directory.

R-4.2.1-arm64.pkg (notarized and signed)
SHA1-hash: 05370dd000f0fded68594fc95334808ee25a8e91
(ca. 89MB) for M1 Macs only!

R 4.2.1 binary for macOS 11 (**Big Sur**) and higher, **Apple silicon arm64** build, signed and notarized package.
Contains R 4.2.1 framework, R.app GUI 1.79 for Apple silicon Macs (M1 and higher), Tcl/Tk 8.6.12 X11 libraries and Texinfo 6.8.
Important: this version does NOT work on older Intel-based Macs.

R for Mac users



https://www.rstudio.com/products/rstudio/download/

LEARN MORE ABOUT THE RSTUDIO IDE

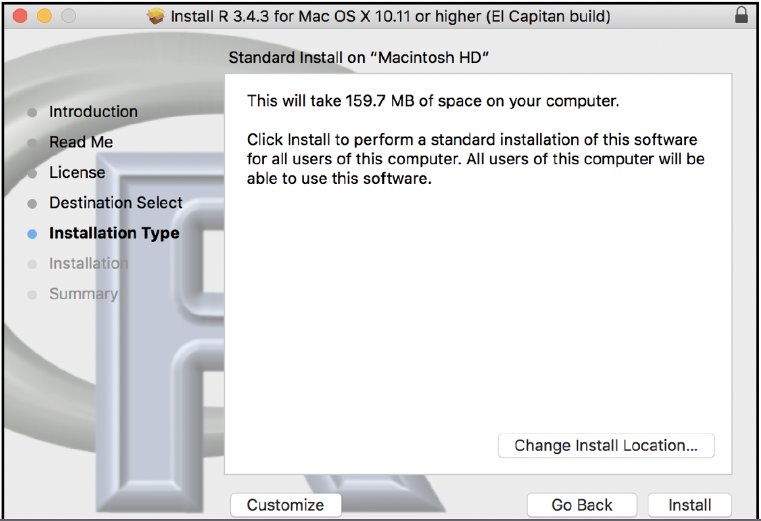
professional software for data analysis, package management, and sharing dat products.

Learn more about RStudio Team

	RStudio Desktop	RStudio Desktop Pro	RStudio Server	RStudio Workbencl
	Open Source License	Commercial License	Open Source License	Commercial License
	Free	$995 /year	Free	$4,975 /year (5 Named Users)
	DOWNLOAD Learn more	BUY Learn more	DOWNLOAD Learn more	BUY Evaluation | Learn mor
Integrated Tools for R	✓	✓	✓	✓
Priority Support		✓		✓
Access via Web Browser			✓	✓

Figure 1.1. Screen image of R for Windows.

R installation



Standard Install on "Macintosh HD"

This will take 159.7 MB of space on your computer.

Click Install to perform a standard installation of this software for all users of this computer. All users of this computer will be able to use this software.

R objects

R as calculator

- One of the simplest possible tasks in R is to enter an arithmetic expression and receive a result. (The second line is the answer from the machine.)

  ```
  > 2 + 2
  [1] 4
  > exp(-2)
  [1] 0.1353353
  ```

- Generating 4 random numbers from a normal distribution

  ```
  > rnorm(4)
  [1]  1.3507720  1.0938817 -0.5241599 -0.6047982
  ```

Assignments

```
> x<-2
> x
[1] 2
> x<-A
Error: object ’A’ not found
> x<-’A’
> x
[1] "A"
```

Vectors

Vector Objects

- The construct c(...) is used to define vectors

```
> weight <- c(60, 72, 57, 90, 95, 72)
> weight
[1] 60 72 57 90 95 72
```

- You can do calculations with vectors as long as they are of the same length:

```
> height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
> bmi <- weight/height^2
> bmi
[1] 19.59184 22.22222 20.93664 24.93075 31.37799
[2] 19.73630
```

Vector Objects

- Computing the mean

```
> sum(weight)
[1] 446
> sum(weight)/length(weight)
[1] 74.33333
```

Vector Objects

- Computing the Standard deviation

```
> xbar <- sum(weight)/length(weight)
> weight - xbar
[1] -14.333333 -2.333333 -17.333333 15.666667 20.666667
[6] -2.333333
> (weight - xbar)^2
[1] 205.444444 5.444444 300.444444 245.444444 427.111111
[6] 5.444444
> sum((weight - xbar)^2)
[1] 1189.333
> sqrt(sum((weight - xbar)^2)/(length(weight) - 1))
[1] 15.42293
```

Vector Objects

- Using mean and sd functions

```
> mean(weight)
[1] 74.33333
> sd(weight)
[1] 15.42293
```

Vector Objects

- A character vector is a vector of text strings

```
> c("Huey","Dewey","Louie")
[1] "Huey" "Dewey" "Louie"
> c(0,2,3,"A")
[1] "0" "2" "3" "A"
```

- A logical vector

```
> c(T,T,F,T)
[1] TRUE TRUE FALSE TRUE
> c(T,F,0,T)
[1] 1 0 0 1
> c(T,F,"A")
[1] "TRUE"  "FALSE" "A"
```

Manipulating Vectors

- Concatenate vectors

```
> x <- c(1, 2, 3)
> y <- c(10, 20)
> c(x, y, 5)
[1] 1 2 3 10 20 5
```

- Assign names to the elements

```
> x <- c(red="Huey", blue="Dewey", green="Louie")
> x
   red    blue   green
"Huey" "Dewey" "Louie"
```

Creating Vectors

- Creating a sequence of numbers from 4 to 9

  ```
  > seq(4,9)
  [1] 4 5 6 7 8 9
  > 4:9
  [1] 4 5 6 7 8 9
  ```

- Creating a sequence of numbers from 4 to 9 with jumps of 2

  ```
  > seq(4,10,2)
  [1] 4 6 8 10
  ```

- Repeating a vector

  ```
  > oops <- c(7,9,13)
  > rep(oops,3)
  [1] 7 9 13 7 9 13 7 9 13
   rep(1:2,c(2,4))
  [1] 1 1 2 2 2 2
  ```

Built-in Functions

Built-in Functions

All the mathematical functions are here in R

- log function

  ```
  > log(10)
  [1] 2.302585
  ```

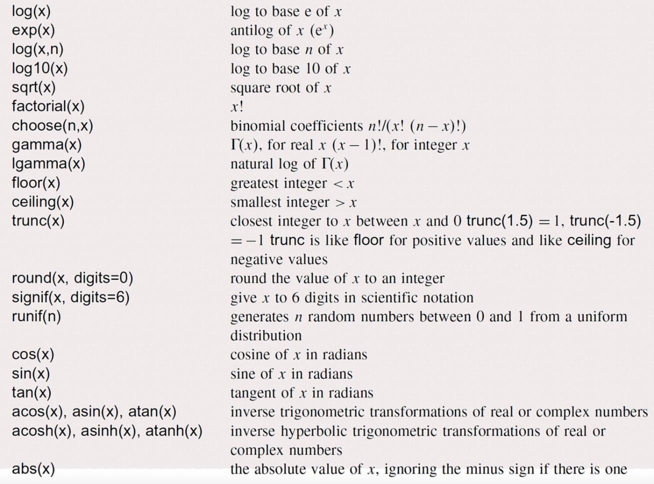
- log to the base 10

  ```
  > log10(6)
  [1] 0.7781513
  > log(6)/log(10)
  [1] 0.7781513
  ```

- log to the base 3

  ```
  > log(10,3)
  [1] 2.095903
  > log(10)/log(3)
  [1] 2.095903
  ```

Built-in Functions



log(x)	log to base e of x
exp(x)	antilog of x (e^x)
log(x,n)	log to base n of x
log10(x)	log to base 10 of x
sqrt(x)	square root of x
factorial(x)	$x!$
choose(n,x)	binomial coefficients $n!/(x!\ (n-x)!)$
gamma(x)	$\Gamma(x)$, for real x $(x-1)!$, for integer x
lgamma(x)	natural log of $\Gamma(x)$
floor(x)	greatest integer $< x$
ceiling(x)	smallest integer $> x$
trunc(x)	closest integer to x between x and 0 trunc(1.5) $= 1$, trunc(-1.5) $= -1$ trunc is like floor for positive values and like ceiling for negative values
round(x, digits=0)	round the value of x to an integer
signif(x, digits=6)	give x to 6 digits in scientific notation
runif(n)	generates n random numbers between 0 and 1 from a uniform distribution
cos(x)	cosine of x in radians
sin(x)	sine of x in radians
tan(x)	tangent of x in radians
acos(x), asin(x), atan(x)	inverse trigonometric transformations of real or complex numbers
acosh(x), asinh(x), atanh(x)	inverse hyperbolic trigonometric transformations of real or complex numbers
abs(x)	the absolute value of x, ignoring the minus sign if there is one

Numbers

Numbers with Exponents

For very big numbers or very small numbers R uses the following scheme:

- `1.2e3` means 1200 because the e3 means ‘move the decimal point 3 places to the right’
- `1.2e-2` means 0.012 because the e-2 means ‘move the decimal point 2 places to the left’
- `3.9+4.5i` is a complex number with real (3.9) and imaginary (4.5) parts, and i is the square root of 1.

Modulo and Integer Quotients

- Suppose we want to know the integer part of a division: say, how many 13s are there in 119 (quotient):

```
> 119 %/% 13
[1] 9
```

- Now suppose we wanted to know the remainder (what is left over when 119 is divided by 13): in maths this is known as modulo:

```
> 119%%13
[1] 2
```

- Question: How can we test whether a number is odd or even?

Rounding

- The ‘greatest integer less than’ function is `floor`

```
> floor(5.7)
[1] 5
```

- The ‘next integer’ function is `ceiling`

```
> ceiling(5.7)
[1] 6
```

- Rounding to the closest number with a given number of decimals

```
> round(5.75,21)
[1] 5.8
```

Infinity, Missing values, and others

Infinity

```
> 3/0
[1] Inf
> -5/0
[1] -Inf
> exp(-Inf)
[1] 0
> log(Inf)
[1] Inf
> (0:3)
[1] 0 1 2 3
> (0:3)^Inf
[1]   0   1 Inf Inf
> is.infinite(4)
[1] FALSE
> is.infinite(Inf)
[1] TRUE
```

Missing values

```
> x<-c(1:5,NA)
> is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
> x
[1]  1  2  3  4  5 NA
> mean(x)
[1] NA
> mean(x,na.rm = T)
[1] 3
> ifelse(is.na(x),0,x)
[1] 1 2 3 4 5 0
> which(is.na(x))
[1] 6
```

Min, Max, pmax, pmin

```
> x=sample(1:10,3)
> x
[1] 1 7 6
> y=sample(1:10,3)
> y
[1]  8 10  5
> max(x)
[1] 7
> min(y)
[1] 5
> pmax(x,y)
[1]  8 10  6
> pmin(x,y)
[1] 1 7 5
```

Matrices

Matrices

- A matrix in mathematics is just a two-dimensional array of numbers

```
> x <- 1:12
> dim(x) <- c(3,4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

- Or

```
> matrix(1:12,nrow=3,byrow=T)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Matrices

- Give names to the rows

```
> x <- matrix(1:12,nrow=3,byrow=T)
> rownames(x) <- LETTERS[1:3]
> x
  [,1] [,2] [,3] [,4]
A    1    2    3    4
B    5    6    7    8
C    9   10   11   12
```

- Transpose

```
> t(x)
      A  B  C
[1,]  1  5  9
[2,]  2  6 10
[3,]  3  7 11
[4,]  4  8 12
```

Matrices

```
> x <- matrix(1:12,nrow=3,byrow=T)
> class(x)
[1] "matrix" "array"
> attributes(x)
$dim
[1] 3 4
> dim(x)
[1] 3 4
> is.matrix(x)
[1] TRUE
> x[,2]
[1]  2  6 10
> x[1,]
[1] 1 2 3 4
> x[2,2]
[1] 6
```

Matrices: functions

```
> colSums(x)
[1] 15 18 21 24
> rowMeans(x)
[1]  2.5  6.5 10.5
> apply(x,2,mean)
[1] 5 6 7 8
> apply(x,1,function(z) sum(z^2))
[1]  30 174 446
```

Matrices: adding rows and columns

```
> y=matrix(1:6,ncol=2)
> y
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> cbind(x,y)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    1    4
[2,]    5    6    7    8    2    5
[3,]    9   10   11   12    3    6
```

Arrays

```
> array<-1:25
> is.matrix(array)
[1] FALSE
> dim(array)<-5,5
Error: unexpected ’,’ in "dim(array)<-5,"
> dim(array)<-c(5,5)
> array
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
> is.matrix(array)
[1] TRUE
```

Arrays

Arrays

```
A<-letters[1:24]
> dim(A)<-c(4,2,3)
> A
, , 1

     [,1] [,2]
[1,] "a"  "e"
[2,] "b"  "f"
[3,] "c"  "g"
[4,] "d"  "h"
.... Truncated output
> A[1,2,2]
[1] "m"
> A[1,2,]
[1] "e" "m" "u"
> A[,2,]
     [,1] [,2] [,3]
[1,] "e"  "m"  "u"
[2,] "f"  "n"  "v"
[3,] "g"  "o"  "w"
```

Boolean objects

Boolean objects and logic instructions Logic

- Logic operations: <, >, <=, >=, != [different], == [equal] return `TRUE` or `FALSE`
- The comparison between 2 vectors is done term by term
- If vectors do not have the same length, the shortest is completed automatically.

```
> a = 1:5; b = 2.5
> a<b
[1]  TRUE  TRUE FALSE FALSE FALSE
```

Boolean objects and logic instructions Logic

- Extract elements in a vector according to specific condition

```
> a[a>3]
[1] 4 5
> a<-1:10
> a[a<=4 | a>=8]
[1]  1  2  3  4  8  9 10
> a[a<=4 & a>=8]
integer(0)
> a[a>4 & a<=8]
[1] 5 6 7 8
```

List

Definition

- A `list` is a structure containing objects (not necessarily of same type). A list is created using the function `list`
- Example: A list named `rnd` contining 3 objects
  - a vector in a vector called `serie`
  - a scalar in a variable called `length`
  - a sequence of characters in a variable called `type`
- The code

```
> rnd = list(serie=c(1:100), length = 100, type=’arithm’
```

- Remark: A list might be created without giving a name to variables

```
> rnd = list(c(1:100), 100, "arithm")
```

Operations on lists

- To display the list of elements in a list

```
> names(rnd)
[1] "serie"  "length" "type"
```

- length of a list

```
> length(rnd)
[1] 3
```

- Summary of a list

```
> summary(rnd)
       Length Class  Mode
serie  100    -none- numeric
length   1    -none- numeric
type     1    -none- character
```

Operations on lists

- To extract an elements in a list

```
[1] 100
> rnd[[2]]
[1] 100
> rnd[2:3]
$length
[1] 100

$type
[1] "arithm"
```

Dataframes

Definition

- A `dataframe` is a matrix where columns are not necessarily of a same type: scalar, boolean, character. But the elements in the same column should be with the same type.
- Example:

```
> data1 = data.frame(x1=1,x2=1:5, letter=letters[1:5])
> data1
  x1 x2 letter
1  1  1      a
2  1  2      b
3  1  3      c
4  1  4      d
5  1  5      e
```

Operations on dataframes

- First rows

```
> head(data1,2)
  x1 x2 letter
1  1  1      a
2  1  2      b
```

- Last rows

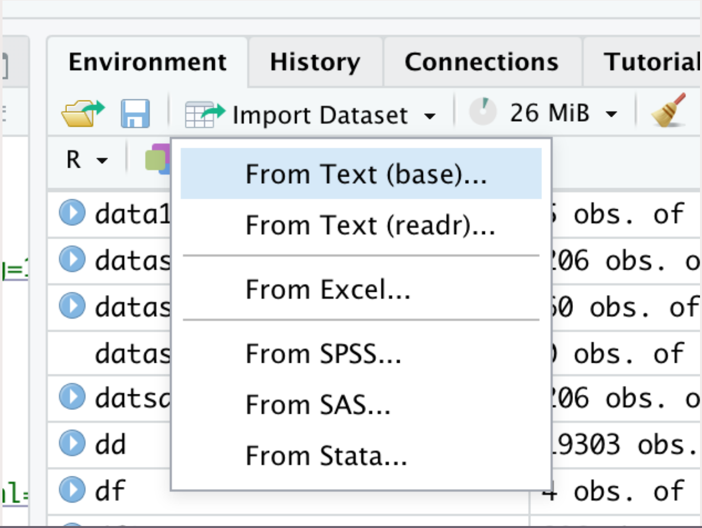
```
> tail(data1,2)
  x1 x2 letter
4  1  4      d
5  1  5      e
```

- Number of rows and columns

```
> dim(data1)
[1] 5 3
```

Importing/Exporting Data

Import interactively using RStudio



Types of data files

- For .txt file use `From Text (base)...`
- For .csv file use `From Text (readr)...`
- For .xls and xlxs files use `From Excel...`
- For spss files use `From SPSS...`