Introducing R

- What is R?
- The logic behind R
- Conventions and good practices
- Useful objects

What is R?

- The R statistical programming language is a free open source package based on the S language developed by Bell Labs.
- The language is very powerful for writing programs.
- Many statistical functions are already built in.
- Contributed packages expand the functionality to cutting edge research.
- Since it is a programming language, generating computer code to complete tasks is required.

Getting Started

- Where to get R?
- Go to

www.r-project.org

- Downloads:

CRAN

[Home]

Download

CRAN

R Project

About R
Logo
Contributors
What's New?
Mailing Lists
Bug Tracking
Conferences
Search

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News

- The **R Logo** is available for download in high-resolution PNG or SVG formats.
- **useR! 2016**, will take place at Stanford University, CA, USA, June 27 - June 30, 2016.
- **The R Journal Volume 7/2** is available.
- **R version 3.2.3 (Wooden Christmas-Tree)** has been released on 2015-12-10.
- **R version 3.1.3 (Smooth Sidewalk)** has been released on 2015-03-09.

Getting Started

- Set your Mirror: Anyone is fine.
- Select Windows or other OS.
- Select base.
- Select R-3.2.3-win32.exe
  - The others are if you are a developer and wish to change the source code.

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals

R-3.2.3 for Windows (32/64 bit)

Download R 3.2.3 for Windows (62 megabytes, 32/64 bit)

Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded exactly matches the package distributed by R the .exe to the true fingerprint. You will need a version of md5sum for windows: both graphical and command lir

Frequently asked questions

- How do I install R when using Windows Vista?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

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

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
- Previous releases

Stata or Spss logic

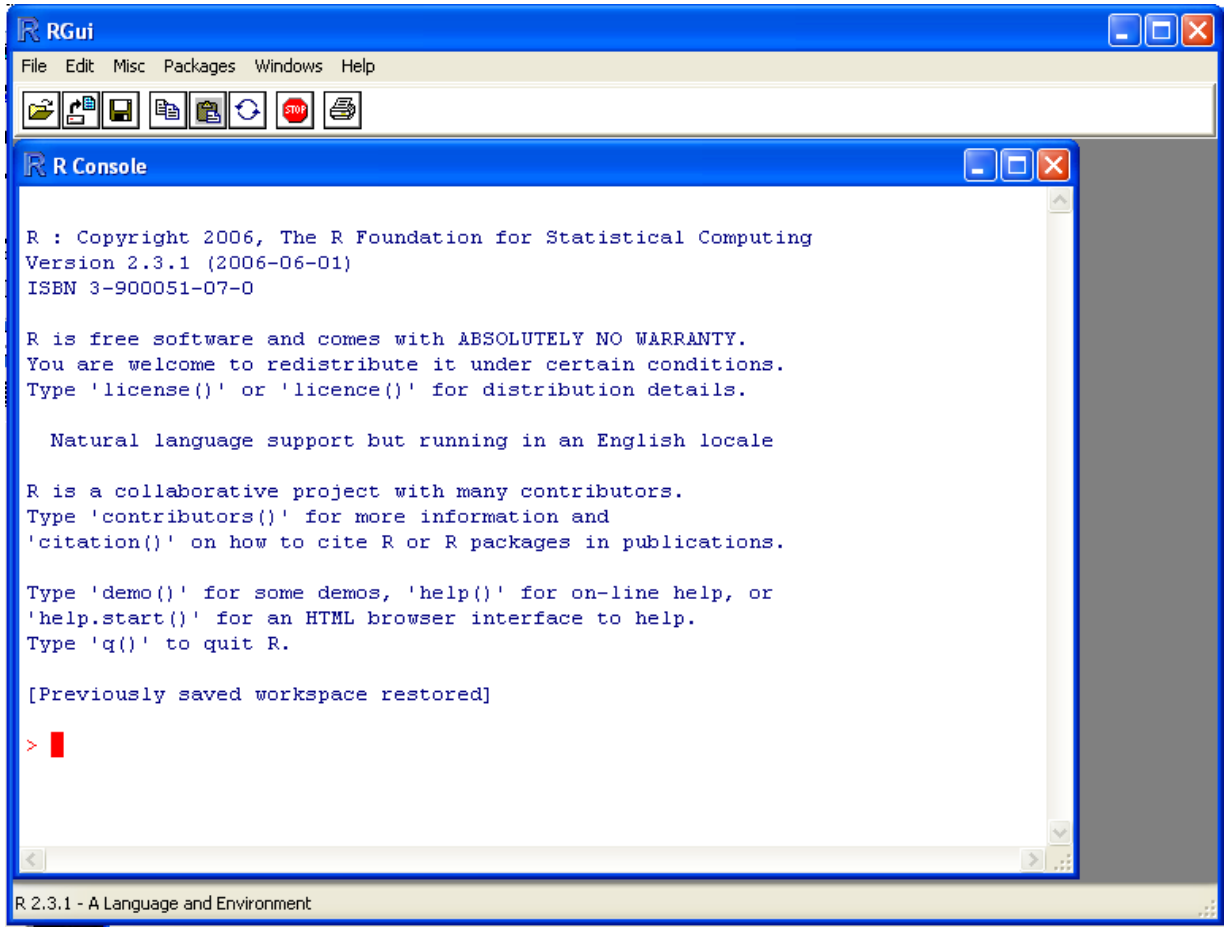
- One single data set at a time.
- Data sets have a row-by-column structure.
- Each row represents a case.
- Each column is a variable.
- Each variable may have various attributes, like a variable label, value labels, user missing values, a data type, etc.
- Most data management and statistical procedures operate on a `row after row' basis.

R logic

- All entities that R creates are known as objects
- Data structures are contained in objects of various sizes, shapes, and types.
- Objects have names. If you want to interact with an object, you have to call it by its name.
- Functions take objects as arguments and give results.
- Objects are resident in the computer's main memory and will disappear when R is stopped, unless you save them.
- Results from statistical calculations are themselves objects that can be further processed.
- R's objects bear much similarity to the `variables' of a typical programming languages

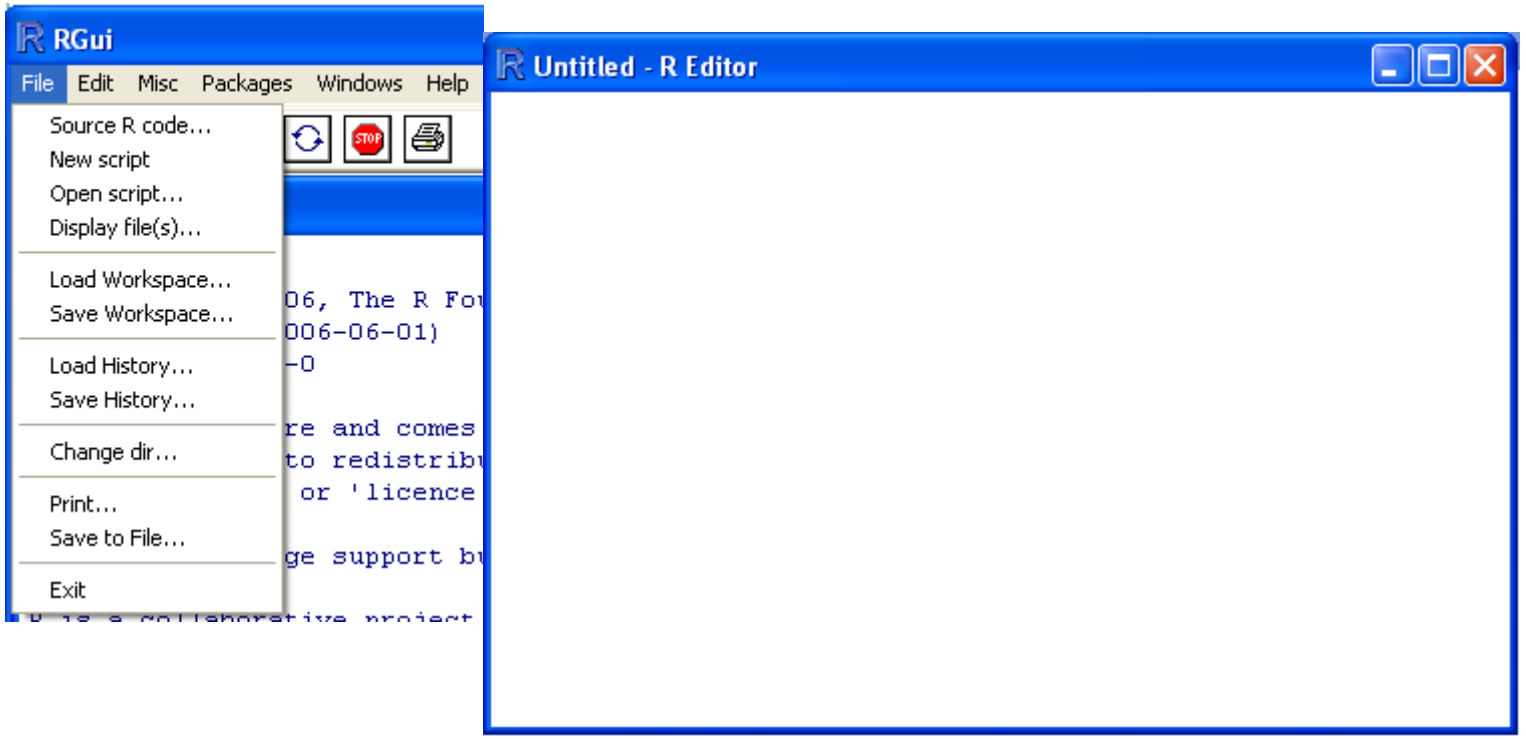
Getting Started

- At start a window with a drop-down menu opens. This is your working environment.
- Inside it a sub-window entitled `R Console' is your interface with the statistical engine.
- More windows with graphical output or data editors can be opened in your working environment.
- The objects you create form a workspace.



Getting Started

- Opening a script.
- This gives you a script window.



How to input commands

- You can enter commands one at a time at the command prompt (>) or run a set of commands from a source file.
- There is a wide variety of data types, including vectors (numerical, character, logical), matrices, dataframes, and lists.
- To quit R, use >q()

How to input commands

- Using R as a calculator

```
> sqrt(534)
[1] 23.10844
> sqrt(534)+115
[1] 138.1084
> sqrt(534)/16
[1] 1.444278
> (sqrt(534)/16)^2
[1] 2.085938
>
```

How to input commands

- Generating and calling objects:

```
> sqrt(534)
[1] 23.10844
> sqrt(534)+115
[1] 138.1084
> sqrt(534)/16
[1] 1.444278
> (sqrt(534)/16)^2
[1] 2.085938
>
```

- Listing all user-defined objects in the active R session:

```
> ls()
[1] "first.object"  "second.object" "third.object"
```

A few conventions and good practices

- R language is case sensitive
- This assignment syntaxes are (almost) equivalent, but the first one is preferred:

```
> first.assign<-1
> 2->second.assign
> third.assign=3
```

- Both double and single quotes can be used, but the firsts are preferred:

```
> third.object<-c("One","Two")
```

- Long objects names should preferably be in the form:

```
> long.object.name
```

Session folders and Saving data

- The golden rule: one project, one directory
  - Control it through the `menu: File > Change Directory`
- All user-created objects are saved in `.RData`
- All commands issued are saved in `.Rhistory`. Objects can also be saved individually

```
> ls()
[1] "first.assign"  "first.object"  "second.assign" "second.object" "third.assign"
[6] "third.object"
> save(first.object, third.object, file="example.RData")
> load("example.RData")
```

Calling for help

- Help function can be used to get information, defaults and examples

```
> ?save
starting httpd help server ... done
> help(save)
> example(save)

save> x <- stats::runif(20)

save> y <- list(a = 1, b = TRUE, c = "oops")

save> save(x, y, file = "xy.RData")
```

R objects: Vectors

- Vectors are the simplest data structure: an unordered collection of elements

```
> num.vect<-c(1:30)
> num.vect
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
[25] 25 26 27 28 29 30
> logic.vect<-num.vect<15
> logic.vect
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE
> char.vect<-c("One","Two","Three")
> char.vect
[1] "One"   "Two"   "Three"
```

R objects: Vectors

- Vectors can be manipulated and modified:

```
> num.vect*c(1,2)
 [1]  1  4  3  8  5 12  7 16  9 20 11 24 13 28 15 32 17 36 19 40 21 44 23 48
[25] 25 52 27 56 29 60
> length(num.vect)
[1] 30
```

R objects: Matrices

- Matrices are multidimensional generalizations of vectors.
- Matrices are sequences of values of the same type (numeric, logical, character) with one additional attribute, the dimensionality.

  The dimensionality can be recalled from and assigned to an object via `dim()`.
  - Rows and columns of matrices can have names. These are set and obtained by the functions `rownames()` and `colnames()` - the same way as `names()` does for names of elements of a vector.
  - By default a matrix is filled with the rows increasing fastest (so called column-major order)

Creating Matrices

```
> first.matrix<-matrix(1:12,nrow=4,ncol=4)
> first.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    5    9    1
[2,]    2    6   10    2
[3,]    3    7   11    3
[4,]    4    8   12    4
> second.matrix<-matrix(1:12,nrow=4,ncol=4,byrow=TRUE)
> second.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]    1    2    3    4
```

Manipulating Matrices

- Row vectors of a matrix M can be accessed (i.e. extracted and assigned to) by e.g. `M[1,], M[2,], M[3,]`, etc. while column vectors can be accessed by e.g. `M[,1], M[,2], M[,3]`, etc.

- Single cells can be accessed by `M[1,1], M[1,2], M[2,1], M[2,2], M[1,3],` etc.

- Submatrices can be accessed by e.g. `M[1:2,1:3]`. In this example, `M[1:2,1:3]` is the submatrix composed of the first two rows and the first three columns of matrix M.

Matrix Operations

- Usually, operators that work elementwise like `+, -, *, /`, can also applied to matrices. Recycling works only if the recycled operator is not a matrix
- R also provides special matrix algebra operators:
  - Matrix product: `%*%`
- Matrix transpose: `t(A)` gives the transpose of A
- Cross products: `crossprod(X,Y)` is a shorthand and computationally efficient version of `t(X) %*% Y`

How to import matrices from files

- From .txt files

```
> dataset1<-as.matrix(read.table("C:/Users/user/Desktop/dataset1.txt"))
```

- From .xlsx files

```
> library(xlsx)
> dataset1<-as.matrix(read.xlsx("C:/Users/user/Desktop/dataset1.xlsx"))
```

- From other file formats

```
> library(foreign)
> dataset1<-as.matrix(read.spss("C:Users/user/Desktop/dataset1.spss"))
```

Introducing SNA with R

- R packages for SNA
- Importing SNA data in R
- Descriptive SNA in R
- Exporting SNA data

Available tools

- The statnet suite
  - R packages for SNA
- Importing SNA data in R
- Descriptive SNA in R
- Exporting SNA data

Introducing SNA with R

- The statnet suite (see http://csde.washington.edu/statnet/):
  - Network
  - sna (see http://erzuli.ss.uci.edu/R.stu/)
  - degreenet
  - Latentnet
  - Networksis
  - ergm
- igraph (see http://igraph.sourceforge.net/)
- tnet
- RSiena

Statnet versus igraph

- The statnet suite
  - use the same data structure, the network object, for both
  - descriptive analysis and modelling
  - focusses specifically on social networks
  - Has a very supportive community

- The igraph package
  - is meant for speed
  - appears to have superior graphical capabilities
  - appears to be better equipped for community detection (it does cohesive blocking)
  - Has an equally supportive community

Statnet versus igraph

- Both igraph and the statnet suite can be used at the same time, however some functions share the same name and are masked from the first package you load

- In this case you can use `some.package.name::some.function.name` to specify the package you want to use

- A simple two-way conversion between the two data structures they use, network and graph, does not exist

Install and load the Statnet suite

- On a computer on which you have administrative privileges either
  - `install.packages("statnet")`
  - Or navigate through the menus: `Packages > Install packages`... And choose a near server
- In order to load statnet use `library(statnet)`
- Note that the statnet suite, like any other contributed packages, can be automatically loaded at start-up by modifying R's configuration scripts

Statnet suite

- An object class specifically suited for network data
- Functions that will work on network objects, but also on matrices, and arrays or list of them (at least in most cases)
- Functions that will work on actor-level measures recursively
- Specific plotting functions for network visualisation

Matrices and network objects

```
library (statnet)
data (flo)

class (flo)
attributes (flo)
summary (flo)

nflo <- network (flo, directed = FALSE )
class (nflo)
summary (nflo)
plot (nflo, displaylabels =TRUE , boxed.labels = FALSE )
```

The network class

The network class data structure has two main advantages

- it can store meta-information on vertices, edges, and the network as a whole
- it allows more efficient computations

It accepts as inputs

- Adjacency matrices
- Edge lists
- Incidence matrices
- Bipartite adjacency matrices

Vertex IDs and Vertex names

- Vertices and edges are labelled internally by the `network()` function in order of entry.
- It is assumed that this is maintained for vertices.
- Removing vertices therefore requires internal relabeling of vertices to keep the ID list continuous.
- It is preferable to set a vertex attribute with meaningful names for your vertices. This attribute will remain in your control and avoid confusion.
- The network class has a reserved vertex attribute, `vertex.names`, for this exact purpose.

Vertex IDs and Vertex names

- If vertex names are not specified, they default to the internal vertex IDs
- In the case of matrices with named dimensions, e.g. flo, vertex names are automatically stored
- Vertex names can be also manually stored and manipulated

Vertex IDs and Vertex names

```
list.vertex.attributes (nflo)
get.vertex.attribute (nflo, "vertex.names")
```

```
 [1] " Acciaiuoli " " Albizzi " " Barbadori " " Bischeri " " Castellani «
 [6] " Ginori " " Guadagni " " Lamberteschi " " Medici " " Pazzi "
[11] " Peruzzi " " Pucci " " Ridolfi " " Salviati " " Strozzi «
[16] " Tornabuoni "
```

```
nflo %v% " vertex.names " <- letters [1:16]
nflo %v% " vertex.names
```

```
 [1] "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" "a"
```

Vertex attributes

Generalising from `vertex.names`

```
list.vertex.attributes (NET _ NAME )
get.vertex.attribute (NET_NAME , " ATTR _ NAME ")
set.vertex.attribute (NET_NAME , " ATTR _ NAME ", ATTR _VALUE ,
VERTEX _ID)
```

or use the operator %v%

Edge attributes

```
list.edge.attributes (NET _ NAME )
get.edge.attribute (NET_NAME , " ATTR _ NAME ")
set.edge.attribute (NET_NAME , " ATTR _ NAME ", ATTR _VALUE ,
EDGE_MATRIX _ID)
```

Using `set.edge.value` allows edges to be specied in `matrix[,]` notation

Also the operator `%e%` is available

Importing network data into R

Various data formats are allowed:

Adjacency

- a square matrix or two-dimensional array, whose i, jth cell contains the value of the edge from i to j
- only for dyadic networks

use `ignore.eval=FALSE, names.eval="EDGE_WEIGHT_NAME"` for valued matrices

Edge list

- a rectangular matrix or two-dimensional array whose row elements represent edges
- additional columns are taken to contain edge attribute values

Its `matrix.type` is `"edgelist"`

Importing matrices from text files

```
klas.mat <-as.matrix (read.table("dataset/klas12b-net-1.dat"))
klas.net <- network (klas.mat, ignore.eval =FALSE,
      names.eval ="weights")
summary(klas.net)

klas.net %e% "weights"
Klas.net %v% "vertex.names“  # or network.vertex.names(klas.net)

plot(klas.net)
```

Adding attributes from text files

```
klas.dem.mat <- as.matrix (read.table ("dataset/dem.dat",col.names =
                c("gender","age","ethnicity","religion")))

klas.net %v% colnames(klas.dem.mat)[2] <- klas.dem.mat[,2]
klas.net %v% colnames(klas.dem.mat)[3] <- klas.dem.mat[,3]

list.vertex.attributes(klas.net)
```

Adding attributes from text files

```
klas.dem.mat <- as.matrix (read.table("dataset/dem.dat",col.names =
                c(" gender ","age"," ethnicity "," religion ")))

klas.net %v% colnames(klas.dem.mat)[2] <- klas.dem.mat[,2]

klas.net %v% colnames(klas.dem.mat)[3] <- klas.dem.mat[,3]

list.vertex.attributes(klas.net)
```

Computing simple descriptive statistics

Network plotting

```
gplot (klas,g=1)
```

Network-level indices

1. density

```
gden(klas,g=1) #density of network 1 only
```

2. reciprocity

```
grecip(klas)
```

3. transitivity

```
gtrans(klas)
```

3. centralization

```
lapply(klas, centralitazion, evcent)
```

Computing simple descriptive statistics

Actor-level indices

1. degree centrality

```
degree(klas,1) #degree centrality of undirected network 1 only
```

2. in/outdegree centrality

```
degree(klas,1, cmode=«outdegree») #outgoing ties

Hist(degree(klas,1, cmode=«outdegree»)) #plotting outgoing
                                        distribution
```

3. Betweenness centrality

```
betweenness(klas,g=c(1:4))   #betweenness centrality of 4 networks
included in klas
```

Computing simple descriptive statistics

Subgraph indices

1. Geodesic distance

```
geodist(klas,g=1) #geodesic distance in network 1 only
```

2. Network components

```
components(klas)
```