Introduction to programming in R: Day 1

Kanhu Charan Moharana

November 25, 2019

About me

Currently

- PhD student LQFPP/CBB

Research interests

- Bioinformatics
- Plant genomics

Skills and experience

- Unix and grid computing
- Reproducible research using various languages such as Shell, Perl, R, Python.
- Web development

Connect

- Email: kcm.eid@gmail.com
- Twitter @kc_moharana
- http://lattes.cnpq.br/2585560606421215

I assume

- You have basic understanding of MS Excel
  - Excel Sheets, tables
  - Generating basic plots
  - Using simple formula
- You have installed following software on your laptop
  - R software
  - R studio
  - Excel
- Example Data sets

We will not cover

- Advanced statistical tests
  - hypothesis testing
  - Multivariate tests
  - Machine learning
  - Clustering, Classification
- Bioinformatics analysis

Learning objectives: Day 1

- Introduction to R
  - Basic setup
  - Get help
- Data types and Objects
  - Vectors
  - Factors
  - Matrix
  - List
  - Data frame
  - missing values
- Class Coercion
- Reading and writing Data files
- Sub setting

R

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.

History

- S language (Bell Labs)
- **1991:** Ross Ihka and Robert Gentleman started developing R language.
- **1993:** First public announcement of R
- **1995:** GNU free license to R
- **1996:** R public mailing group was created.
- **1997:** The R Core group was created.
- **2000:** R version 1.0 was released
- Current version 3.6.1: https://cran.r-project.org

R

Pros

- Reproducible data intensive research
- A programming language
- Large community
- Huge number of free to use libraries.
- Easy learning curve.

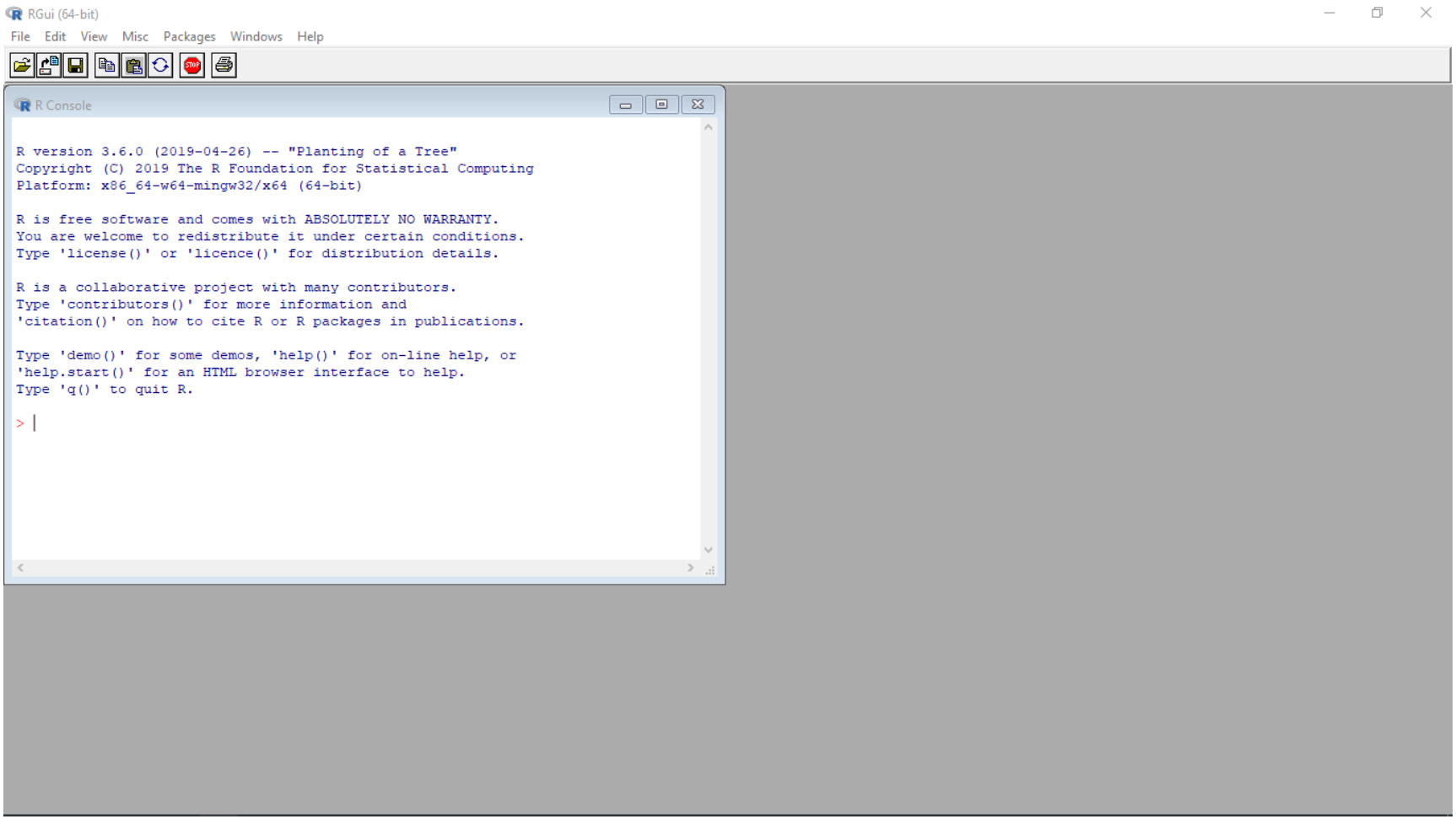
Cons

- Hard to master
- Large data takes large memory.
- https://www.r-bloggers.com/why-r-is-hard-to-learn/

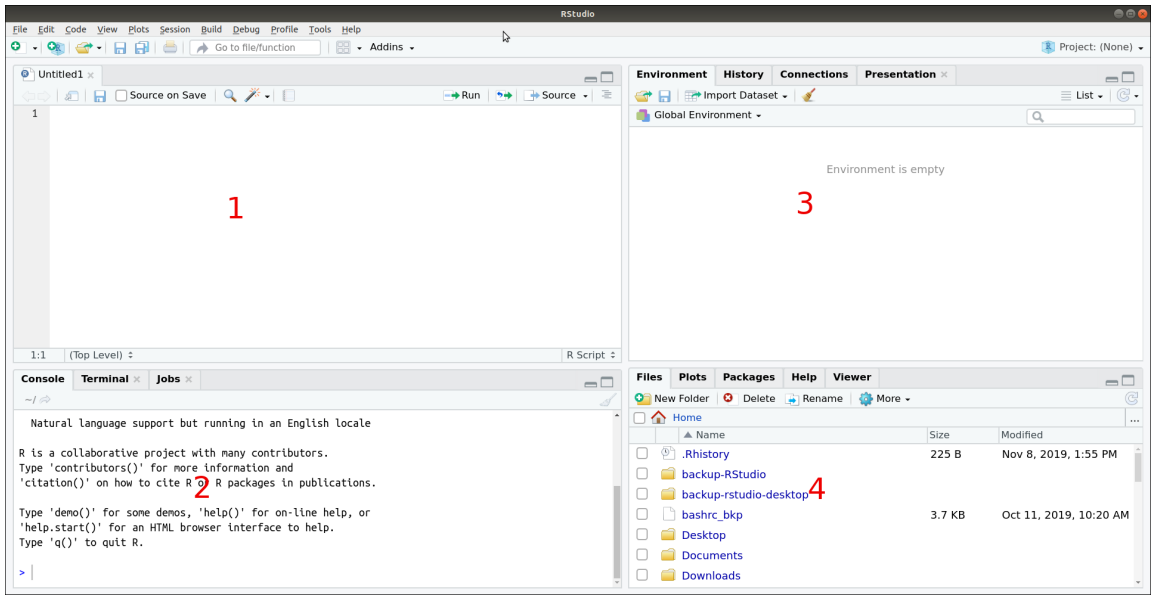
Basic Setup

- Download and install R
  - https://cran.r-project.org/
  - https://mran.microsoft.com/
- Latest version : R 3.6.1
- Integrated Development Environment (IDE)
  - R-studio
- Online editors
  - https://rdrr.io/snippets/

Core R terminal



R Studio: interactive developmental environment



1. Syntax-highlighting code editor
2. R terminal
3. WorkSpace: Variables and function
4. View plots/help

Hello world

- On R terminal type the follwoing

Hello world !

```
# This is a comment line and will not execute
print("hello")
```

```
## [1] "hello"
```

Using Panel-1 to execute code

- type the follwoing on the panel-1 of R-studio
- select all lines and press `Ctrl+Enter`

Print R version

```
r_version <- getRversion()
print("your R version is: ")
```

```
## [1] "your R version is: "
```

```
print(r_version)
```

```
## [1] '3.4.4'
```

Save the script

- Use `File -> save as` to save the script as `print_r_version.R`

R terminal as calculator

- R interface can be used as a simple calculator.

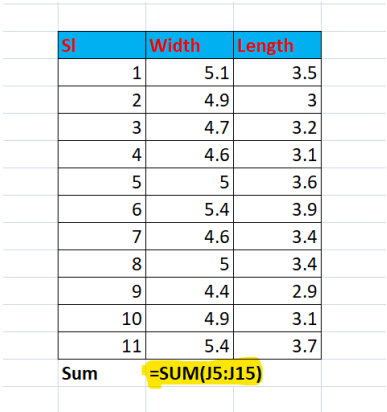

```
# Type on R terminal
1+5
100 * (10+40+33)/200
a  = 10
b  = 34
a + b
```

Assignment operator (<-), left to right

- To store values in a variable (on RAM)
- equals to(=) or <-
  - both can be used for assignment
  - <- is more accepted for variable assignment
  - = used for defining function parameters.

```
## Type on your R terminal
foo <- 10.3
bar <- 89
baz <- foo * bar
```

R built-in functions



Sl	Width	Length
1	5.1	3.5
2	4.9	3
3	4.7	3.2
4	4.6	3.1
5	5	3.6
6	5.4	3.9
7	4.6	3.4
8	5	3.4
9	4.4	2.9
10	4.9	3.1
11	5.4	3.7
Sum	=SUM(J5:J15)	

Figure 1: Excel sum() function

- **functions** are a type of procedure or routine
  - often accept input and parameters
  - performs some operation and may return a value
- use a pair of parenthesis '()'

R built-in functions

```
# Diplays demo plots
demo()

# Get help
help()
# help on any topic, e.g plot
help('plot')
# list all files in current working direcotry
dir()

# Exit R terminal
q()
```

Working directory

- Where to find and save necessary files on your computer.
- Should be defined before beginning any analysis.
- Use menus,
  - `Session -> Set working directory.`
- Using command line `getwd()` and `setwd()`

```
# Get the current working directory
getwd()
# Set the current working directory to PATH
setwd("path/to/address")

# Change \ to  \\ or / for Windows
setwd('C:/Users/Kanhu/Desktop/R_course')
## OR
setwd('C:\\Users\\Kanhu\\Desktop\\R_course')
```

Basic syntax rules

- Each line may have an optional semicolon (;)
  - useful in writing compact functions
- Multi-line instructions should be separated by plus (+)
  - `ggplot2` uses +
- **Comments:** A comment is a way of annotating code.
  - Hash (#)symbol is used at the beginning of the line.
  - These lines will not be executed by R
  - Instruction or documentation purpose
- **Functions:** Executes predefined instructions
  - accepts parameters.
  - Example:

```
x <- 'I love UENF' print(x, ...)
```

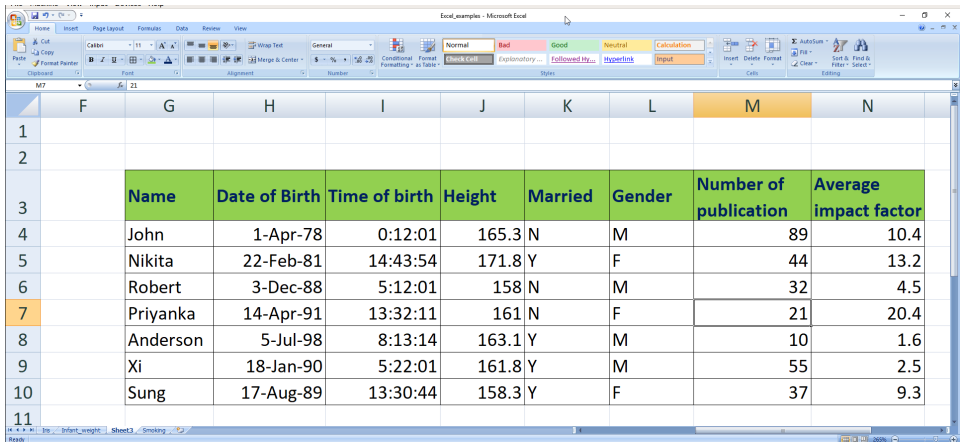
Tabular data

String	Date	Time	Decimal	Binary	Binary	Integer	Decimal
Name	**Date of Birth**	**Time of birth**	**Height**	**Married**	**Gender**	**Number of publication**	**Average impact factor**
John	1-Apr-78	0:12:01	165.3	N	M	89	10.4
Nikita	22-Feb-81	14:43:54	171.8	Y	F	44	13.2
Robert	3-Dec-88	5:12:01	158	N	M	32	4.5
Priyanka	14-Apr-91	13:32:11	161	N	F	21	20.4
Anderson	5-Jul-98	8:13:14	163.1	Y	M	10	1.6
Xi	18-Jan-90	5:22:01	161.8	Y	M	55	2.5
Sung	17-Aug-89	13:30:44	158.3	Y	F	37	9.3

Figure 2: Tables with rows and columns

- A table has rows and columns

Cell address



	Name	Date of Birth	Time of birth	Height	Married	Gender	Number of publication	Average impact factor
4	John	1-Apr-78	0:12:01	165.3	N	M	89	10.4
5	Nikita	22-Feb-81	14:43:54	171.8	Y	F	44	13.2
6	Robert	3-Dec-88	5:12:01	158	N	M	32	4.5
7	Priyanka	14-Apr-91	13:32:11	161	N	F	21	20.4
8	Anderson	5-Jul-98	8:13:14	163.1	Y	M	10	1.6
9	Xi	18-Jan-90	5:22:01	161.8	Y	M	55	2.5
10	Sung	17-Aug-89	13:30:44	158.3	Y	F	37	9.3

Figure 3: Cell address: M7 (Column:M, Row:7)

- Each bit of data is stored in a CELL
- Collection of CELLs make a SERIES
  - Two cells can be used for mathematical operation and the result can be saved in another cell.
  - Row number and column number can be used to point to one cell

Data table

- Many SERIES combine together to form a table
  - A series can be a column or row
  - Useually one column has a distinct datatype

Data types

String	Date	Time	Decimal	Binary	Binary	Integer	Decimal
Name	**Date of Birth**	**Time of birth**	**Height**	**Married**	**Gender**	**Number of publication**	**Average impact factor**
John	1-Apr-78	0:12:01	165.3	N	M	89	10.4
Nikita	22-Feb-81	14:43:54	171.8	Y	F	44	13.2
Robert	3-Dec-88	5:12:01	158	N	M	32	4.5
Priyanka	14-Apr-91	13:32:11	161	N	F	21	20.4
Anderson	5-Jul-98	8:13:14	163.1	Y	M	10	1.6
Xi	18-Jan-90	5:22:01	161.8	Y	M	55	2.5
Sung	17-Aug-89	13:30:44	158.3	Y	F	37	9.3

Figure 4: Each column has distinct datatype

Data types in R

1. **Logical :** `TRUE, FALSE`
2. **Character:** `'a','name','TRUE', "1234", "--NA", "+2"`
3. **Integer:** `10,  20,100, 78`
4. **Double:** `0.0001, 12.898, 10e-10`
5. **Complex:** `3 + 2i`
6. **Raw:**

Variables and R objects

- In programming, a variable is a value that can change, depending on conditions or on information passed to the program.
- Rules to follow:
  - Identifiers can be a combination of letters, digits, period (.) and underscore (_).
  - It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
  - They are case sensitive. e.g `Data_table` NOT equals to `data_table`

Variable names

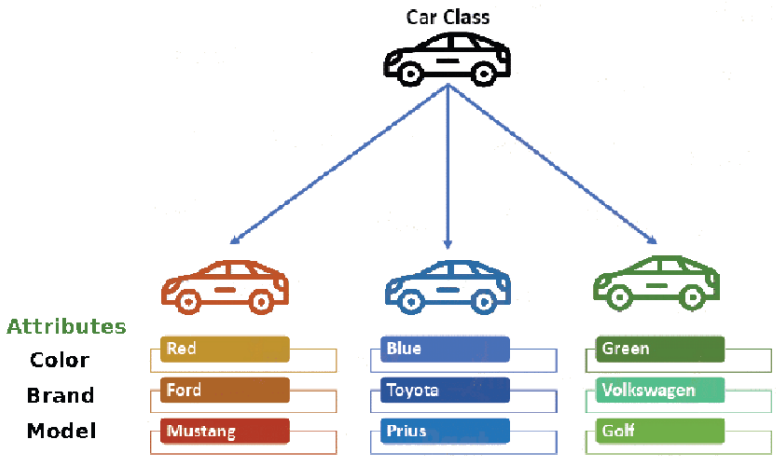
Acceptable variable names

`total, Sum, .fine.with.dot, this_is_acceptable, Number5`

Wrong variable names

- `tot@l, 5um, _fine, TRUE, .0ne`
- Reserved words in R cannot be used as identifiers.
  - Reserved words :
    - `for, if, next`
  - Pre-defined Constants :
    - `pi, LETTERS, letters, month.name`

Objects in real world



Class, attributes and objects

- **Class :** Car has several **attributes**
  - Number of wheels
  - Fuel
  - Color
- Instances of each class is an **object**
  - So each class inherits the attributes of the class

Objects in R

Primary objects

- **Vector:** collection of values
- **Matrix:** 2D-collection of multiple vectors
- **Array :** >2D-collection of multiple vectors

Derived objects

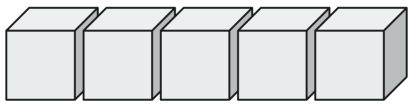
- List
- Data frame

Special Vector

- Factor

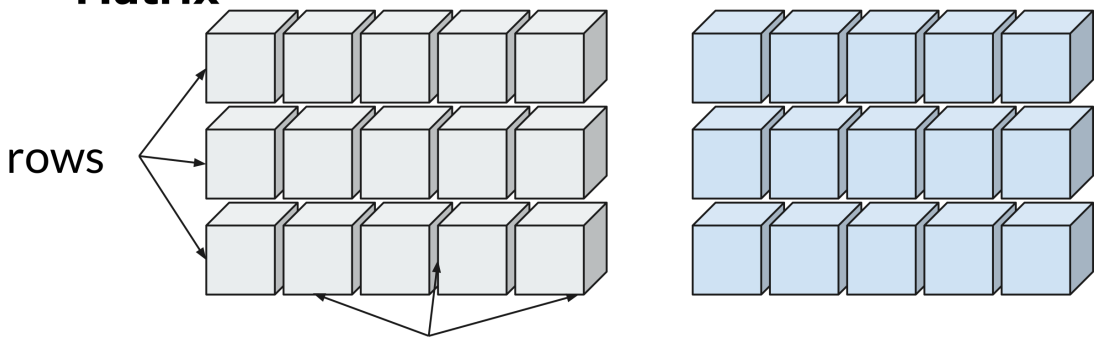
Objects in R

Vectors

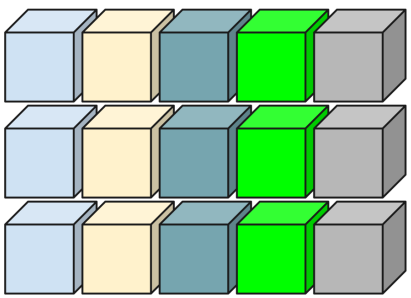


Vector of length 5

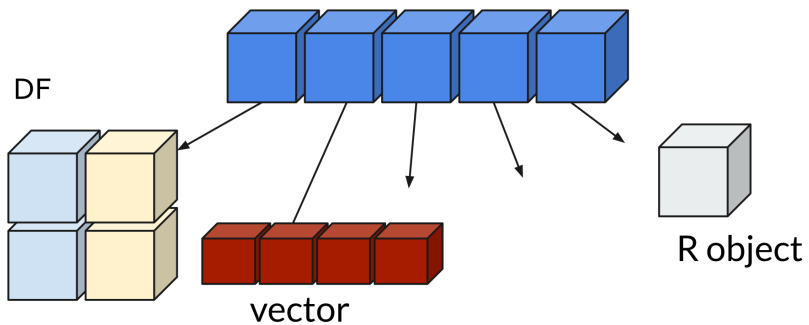
Matrix



Dataframe

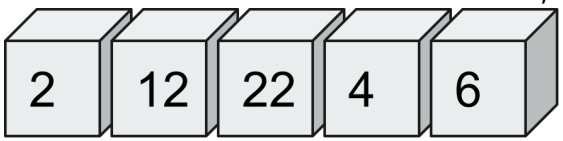


List



Vectors

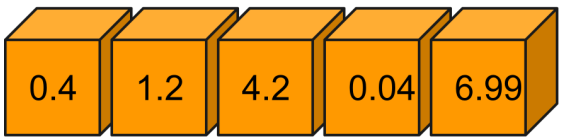
- Basic type of 'R object'
- Only one type of data
- Similar to one column in Excel Table.
- If there is one character data, will convert all to character
- If there is one decimal data, convert all to decimal



Integer vector

```
int_vec <- c(2,12,22,4,6)
```

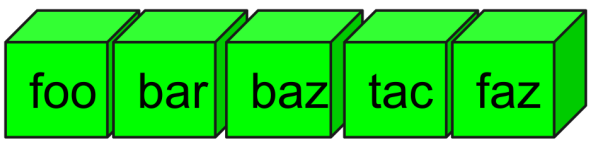
```
x <- vector(mode='integer', length=5)
```



Numeric vector

```
num_vec <- c(0.4,1.2,4.2,0.04,6.99)
```

```
x <- vector(mode='numeric', 5)
```



Character vector

```
char_vec <- c('foo','bar','baz','tac',
'faz')
```

```
x <- vector(mode='character', 5)
```

Vectors

- type on R terminal

```
x <- vector(mode='double',length=5)

# Three 'shortcuts' to create  numerical vector
a <- c(1,2,5,6,7)
b <-  1:5
c <- seq(12,20, by= 2)

dec <- seq(0, 0.5, by=0.025)

products <- c('milk', 'mango', 'good')
```

Factors

- Categorical data
  - Ordered (High, medium, Low)
  - Unordered (Male, Female)
- Can be also integers
- Useful in data modelling problems
- Factors are vectors(), with additional information or attribute called levels, representing that order

```
## Create a vector
x <-  factor(
  c('yes', 'no', 'no', 'yes', 'no'),
  levels = c('yes', 'no')
)
```

Factors

- Some functions are useful to create factors

gl()

- Generate factors by specifying the pattern of their levels.

```
# 2 conditions and 8 replication each
 gl(2, 8, labels = c("Control", "Treat"))
```

```
##  [1] Control Control Control Control Control Control Control Control
##  [9] Treat   Treat   Treat   Treat   Treat   Treat   Treat   Treat
## Levels: Control Treat
```

rep()

- Replicate Elements of Vectors

```
 rep(x=1:2, each=2, times=3, length.out=6)
```

```
## [1] 1 1 2 2 1 1
```

Matrix

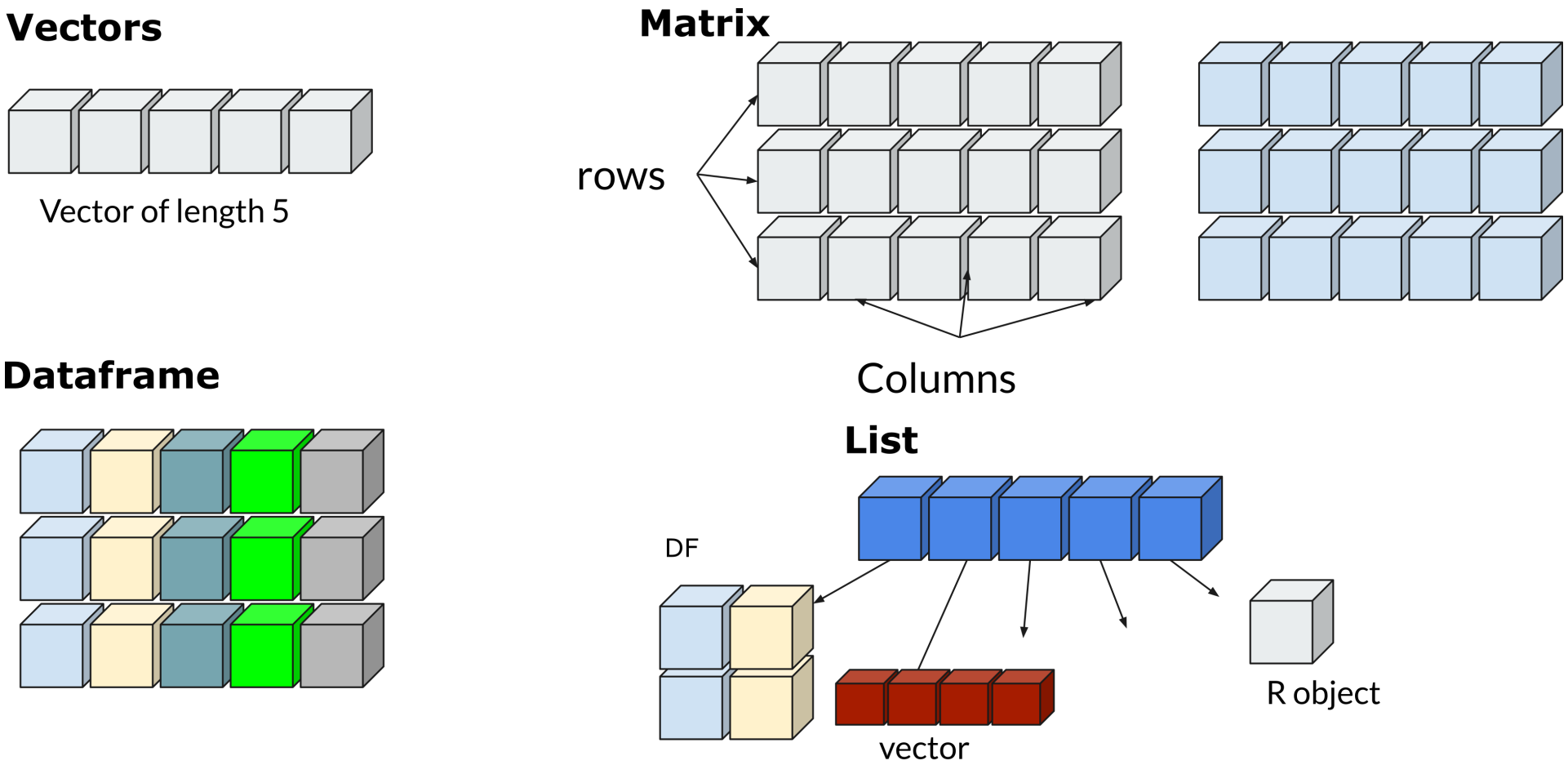


Figure 5: R objects: matrix

Matrix

- Are vectors with dimension attribute (# of rows, # of columns)
- Only one type of data

```
# Empty matrix
m <- matrix()

m1 <- matrix(c(1,2,3,4), byrow=T, nrow=2)
m1
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
m2 <- matrix(c(1,2,3,4), byrow=F, nrow=2)
m2
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
dim(m1)
```

```
## [1] 2 2
```

Matrix for a vector

- We can devide a vector to matrix of specified columns and rows.
- R can interpret the number of rows and columns from your arguments
  - byrow=F, byrow=T

byRow=FALSE

	1	2
1	1	6
2	2	7
3	3	8
4	4	9
5	5	10

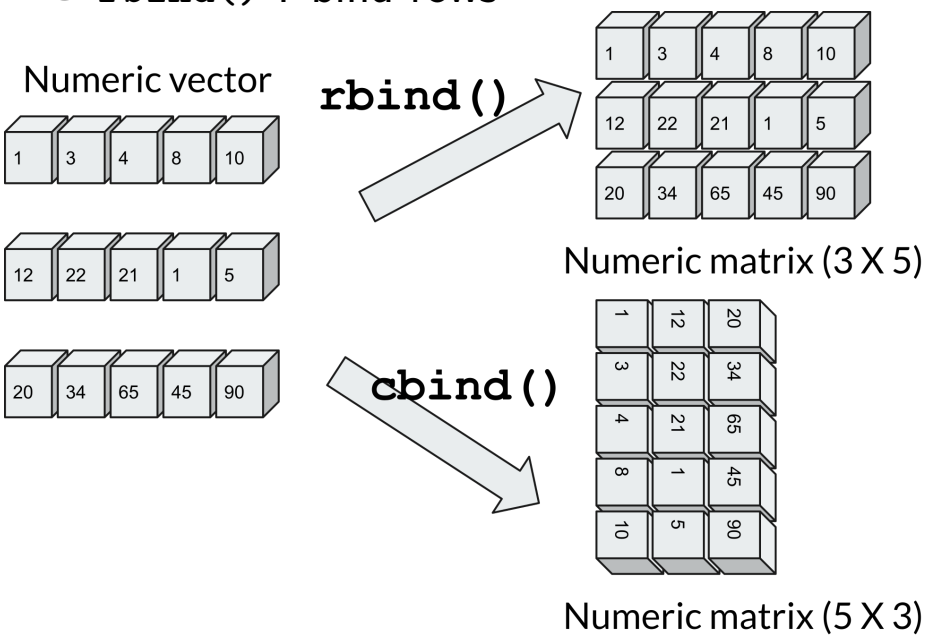
byRow=TRUE

	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	10

Create matrix from vectors using functions

cbind() , rbind()

- `cbind()` : bind columns
- `rbind()` : bind rows



Create matrix from vectors using functions

```
a <- 1:5
b <- 11:15
c <- 21:25

cbind(a,b,c)
```

```
##      a  b  c
## [1,] 1 11 21
## [2,] 2 12 22
## [3,] 3 13 23
## [4,] 4 14 24
## [5,] 5 15 25
```

```
rbind(a,b,c)
```

```
##   [,1] [,2] [,3] [,4] [,5]
## a    1    2    3    4    5
## b   11   12   13   14   15
## c   21   22   23   24   25
```

Mix multiple datatypes in a matrix

```
#Character matrix
mat <- matrix(c('a','b','c','k'), byrow=T, nrow=2)
mat
```

```
##      [,1] [,2]
## [1,] "a"  "b"
## [2,] "c"  "k"
```

```
## Mix an integer type value
mat2 <- matrix(c('a',1,'c','k'), byrow=T, nrow=2)
mat2
```

```
##      [,1] [,2]
## [1,] "a"  "1"
## [2,] "c"  "k"
```

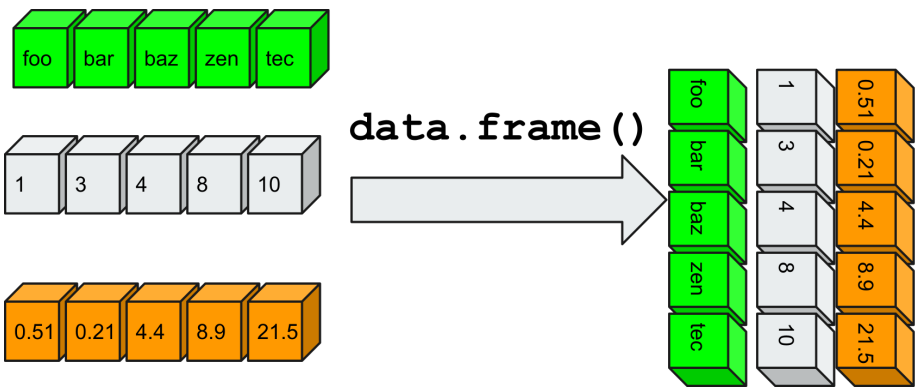
Data frames

- Data frames are collection of *named* vectors.

```
df  <- data.frame(
     Name = c('Robert', 'Priyanka', 'John'),
     Height = c(167.1, 156.9, 162.5)
)
df
```

```
##       Name Height
## 1   Robert  167.1
## 2 Priyanka  156.9
## 3     John  162.5
```

Data frames




```
Df <- data.frame(
      name=c('foo','bar'...),
      score=c(1,3,4, ... ),
      name=c(0.51, 0.21, 4.4,...)
)
```

Figure 6: Data frames

Read tabular files to create DF

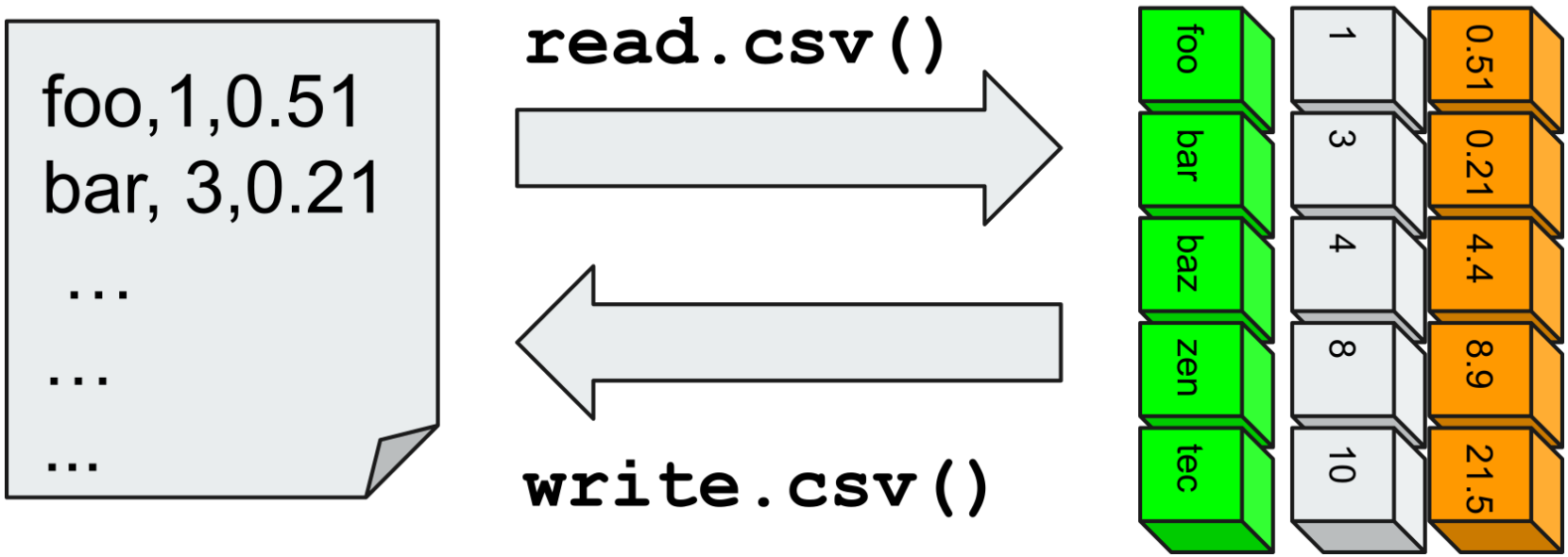


Figure 7: Data frames from *.csv file

List

- Collection of R objects
  - Objects can be of different class

```
new_list  <-list(
     name = c('Robert', 'Priyanka', 'John'),
     Height = c(167.1, 156.9, 162.5),
     num_elements=2
)
new_list
```

```
## $name
## [1] "Robert"   "Priyanka" "John"
##
## $Height
## [1] 167.1 156.9 162.5
##
## $num_elements
## [1] 2
```

Missing Values

- Undefined values
- Denoted by NA or NaN
  - **NaN:** for mathematical operations (e.g: 0/0)
  - **NA:** everything else
    - character NA
    - Integer NA
- **NaN** can be equal to **NA**, but reverse is not true
- *is.na()* used to check **NA** values in a vector
- *is.nan()* used to check **NaN** values

```
x <- c(5,6,NA,9)
is.na(x)
```

```
## [1] FALSE FALSE  TRUE FALSE
```

```
x <- c(5,6,NaN,9, NA)
is.na(x)
```

```
## [1] FALSE FALSE  TRUE FALSE  TRUE
```

```
is.nan(x)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

Class Coercion

1. Implicit coercion

- sometimes different classes of R objects get mixed together
- vectors and matrices cannot have values of different datatypes
- coercion occurs so that every element in the vector is of the same class.
- lower ranking type will be coerced into the higher ranking type
  - logical $<$ integer $<$ numeric $<$ character

```
y <- c(1.7, "a")   ## character
y
```

```
## [1] "1.7" "a"
```

```
p <- c(TRUE, "Hello")
p
```

```
## [1] "TRUE"  "Hello"
```

```
q <- c(TRUE, 12)
q
```

```
## [1]  1 12
```

Class Coercion

2. Explicit Coercion

- Objects can be explicitly coerced from one class to another using the 'as.*' functions.

```
x <- 0:6
class(x)
```

```
## [1] "integer"
```

```
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5 6
```

```
as.logical(x)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Class Coercion

Data frame to matrix

- often many operations require the input as matrix

```
df <- data.frame(foo=1:5,bar=11:15)
df
```

```
##   foo bar
## 1   1  11
## 2   2  12
## 3   3  13
## 4   4  14
## 5   5  15
```

```
class(df)
```

```
## [1] "data.frame"
```

```
as.matrix(df)
```

```
##      foo bar
## [1,]   1  11
## [2,]   2  12
## [3,]   3  13
## [4,]   4  14
## [5,]   5  15
```

Class Coercion

List to data frame

```
l <- list(gene=c('APO','ANN5','SPT'),
          sample1=c(1.4,0.5,2.5),
          sample2=c(2.4,0.3,0.5)
          )
class(l)
as.data.frame(l)
```

generic functions and operations

Function	Description
ls()	list R objects in the current environment
class()	a character vector giving the names of the classes from which the object
length()	the number of elements in a vector or factor.
dim()	a vector of length 2, representing rows and columns
head()	displays first 10 lines of a vector, matrix, df.
tail()	displays last 10 lines of a vector, matrix, df.
str()	compactly display the internal *str*ucture of an R object
print()	generic print an R object to screen or output devices.
rm()	remove/delete given list of R objects from current environment
attributes()	to access an object's attributes.

Figure 8: Some important R functions.

Name attribute

- Naming column and rows makes it easier to data manipulation and human readable.
- All R objects can have names attribute
  ## Named vectors
- each element in a vector can be named.

```
a <- c('a'=5, 'b'=10, 'c'= 2)
a
```

```
##  a  b  c
##  5 10  2
```

```
names(a)
```

```
## [1] "a" "b" "c"
```

Name attribute: Matrix

Matrices

- Row and column of a matrix can be named
- dimnames() : for matrices
  - colnames() : column names
  - rownames() : row names

```
m = matrix(c(1,2,3,4), byrow=F, nrow=2)
dimnames(m) =list( c('p','q'),c('r','s') )
m
```

```
##   r s
## p 1 3
## q 2 4
```

```
colnames(m)
```

```
## [1] "r" "s"
```

```
rownames(m)
```

```
## [1] "p" "q"
```

Name attribute : data frames

data frames

- row.names(): get/set row names
  - row.names() : always unique for each row
- names(): get/set column names
  - names() : always unique for each column

```
df = data.frame(
  k=c('q','a','r'),
  l=c(1,2,3)
  )
df
```

```
##   k l
## 1 q 1
## 2 a 2
## 3 r 3
```

```
row.names(df)
```

```
## [1] "1" "2" "3"
```

Name attribute : data frames

data frames

```
names(df)
```

```
## [1] "k" "l"
```

```
# Assign new row names
row.names(df) <- c('a','b','c')
# assign new column names
names(df) <- c('foo','baz')

df
```

```
##   foo baz
## a   q   1
## b   a   2
## c   r   3
```

Create file by input from keyboard

Enter data using editor

```
mydata <- data.frame(
  age=numeric(0),
  gender=character(0),
  weight=numeric(0)
 )

mydata <- edit(mydata)
```

Contd. . .

Create file by input from keyboard

Enter data using editor

Read/Write data from text file

Read

- Two important functions to read tabular data
  - `read.table()`, `read.csv()`
- Read R script (*.R)
  - `source()`
- Read saved R data objects (*.RData)
  - `load()`

Read/Write data from text file

Write

- Two important functions to write tabular data
  - `write.table()`,`write.csv()`
- Save the R command to a text file (*.R)
  - `savehistory()`
- Read current R data objects to a binary file (*.RData)
  - `save()`

Iris Flower data set

Tabular data

Table 1: First 6 lines

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

read.table()

- Reads a file in table format and creates a data frame from it.

```
df <- read.table(file="iris_dataset.csv",
  header = FALSE,  ## TRUE, if 1st row repsresnts the header

  sep = "", ## Column separator

  skip = 0, ## Skip top 'n' number of lines

  check.names = TRUE, ## Check column names for characters like space, comma, @
  fill = !blank.lines.skip,
  strip.white = FALSE,
  blank.lines.skip = TRUE
)
```

read.csv()

- tweaked version of read.table()
  - some parameters are predefined
  - recommended

```
df <- read.csv(file="iris_dataset.csv",
  header = TRUE,
  sep = ",",
  quote = "\"",
  dec = ".",
  fill = TRUE,
  comment.char = "",
 ...
)
```

write.table() / write.csv()

```
write.csv(data_frame, file="output_dataframe.csv",
  header = TRUE,
  quote = FALSE,
  row.names = TRUE,
  col.names = TRUE
)
```

write R code and read pre-defined codes

savehistory()

- Write the R command history to a text file
- Use extension .R

source()

- read and execute codes written in a plain text file
- usually .R extension

`save` and `load` R objects

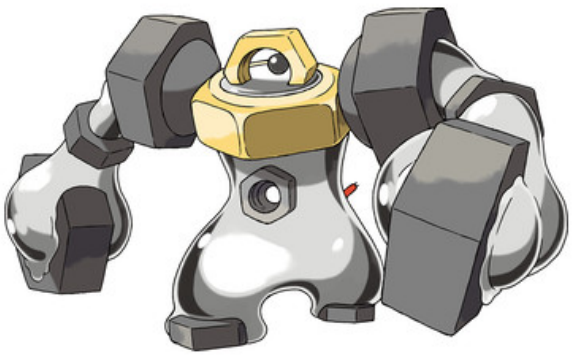
- writes an external representation of R objects to the specified file.
- The objects can be read back from the file at a later date by using the function 'load' or 'attach'
- Use .RData extension

```
## Save the current environment (all objects) to one file
save(data, vector_a, file= "analysis_results.RData")


## Reload all objects) from .RData file
load("analysis_results.RData")
```

Real life example of importing tabular data

Note: *Meltan* has an ability assigned in the code for Pokémon Let's Go, even though abilities are not available.



Additional artwork

Pokédex data

National №	**809**
Type	STEEL
Species	Hex Nut Pokémon
Height	2.5 m (8′02″)
Weight	800.0 kg (1763.7 lbs)
Abilities	1. Iron Fist
Local №	153 (Let's Go Pikachu/Let's Go Eevee)

Training

EV yield	—
Catch rate	—
Base Friendship	—
Base Exp.	—
Growth Rate	—

Breeding

Egg Groups	Undiscovered
Gender	—
Egg cycles	—

Base stats

HP	135	380	474
Attack	143	261	423
Defense	143	261	423
Sp. Atk	80	148	284
Sp. Def	65	121	251
Speed	34	65	183
Total	**600**	Min	Max

Type defenses

The effectiveness of each type on *Melmetal*.

NOR	FIR	WAT	ELE	GRA	ICE	FIG	POI	GRO
½	2			½	½	2	0	2

FLY	PSY	BUG	ROC	GHO	DRA	DAR	STE	FAI
½	½	½	½		½		½	½

Real life example of importing tabular data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Pokedex number	Name	percentage male	type1	type2	weight_kg	height_m	generation	classfication	defense	hp	attack	special attack	special defense	speed	Is legendar
2	1	Bulbasaur	88.1	grass	poison	6.9	0.7	1	Seed Pokémon	49	45	49	65	65	45	0
3	2	Ivysaur	88.1	grass	poison	13	1	1	Seed Pokémon	63	60	62	80	80	60	0
4	3	Venusaur	88.1	grass	poison	100	2	1	Seed Pokémon	123	80	100	122	120	80	0
5	4	Charmande	88.1	fire		8.5	0.6	1	Lizard Pokémon	43	39	52	60	50	65	0
6	5	Charmeleo	88.1	fire		19	1.1	1	Flame Pokémon	58	58	64	80	65	80	0
7	6	Charizard	88.1	fire	flying	90.5	1.7	1	Flame Pokémon	78	78	104	159	115	100	0
8	7	Squirtle	88.1	water		9	0.5	1	Tiny Turtle Pokémo	65	44	48	50	64	43	0
9	8	Wartortle	88.1	water		22.5	1	1	Turtle Pokémon	80	59	63	65	80	58	0
10	9	Blastoise	88.1	water		85.5	1.6	1	Shellfish Pokémon	120	79	103	135	115	78	0
11	10	Caterpie	50	bug		2.9	0.3	1	Worm Pokémon	35	45	30	20	20	45	0
12	11	Metapod	50	bug		9.9	0.7	1	Cocoon Pokémon	55	50	20	25	25	30	0
13	12	Butterfree	50	bug	flying	32	1.1	1	Butterfly Pokémon	50	60	45	90	80	70	0
14	13	Weedle	50	bug	poison	3.2	0.3	1	Hairy Pokémon	30	40	35	20	20	50	0
15	14	Kakuna	50	bug	poison	10	0.6	1	Cocoon Pokémon	50	45	25	25	25	35	0
16	15	Beedrill	50	bug	poison	29.5	1	1	Poison Bee Pokém	40	65	150	15	80	145	0
17	16	Pidgey	50	normal	flying	1.8	0.3	1	Tiny Bird Pokémon	40	40	45	35	35	56	0
18	17	Pidgeotto	50	normal	flying	30	1.1	1	Bird Pokémon	55	63	60	50	50	71	0
19	18	Pidgeot	50	normal	flying	39.5	1.5	1	Bird Pokémon	80	83	80	135	80	121	0
20	19	Rattata	50	normal	dark			1	Mouse Pokémon	35	30	56	25	35	72	0
21	20	Raticate	50	normal	dark			1	Mouse Pokémon	70	75	71	40	80	77	0
22	21	Spearow	50	normal	flying	2	0.3	1	Tiny Bird Pokémon	30	40	60	31	31	70	0
23	22	Fearow	50	normal	flying	38	1.2	1	Beak Pokémon	65	65	90	61	61	100	0
24	23	Ekans	50	poison		6.9	2	1	Snake Pokémon	44	35	60	40	54	55	0
25	24	Arbok	50	poison		65	3.5	1	Cobra Pokémon	69	60	95	65	79	80	0
26	25	Pikachu	50	electric		6	0.4	1	Mouse Pokémon	40	35	55	50	50	90	0
27	26	Raichu	50	electric	electric			1	Mouse Pokémon	50	60	85	95	85	110	0
28	27	Sandshrew	50	ground	ice			1	Mouse Pokémon	90	50	75	10	35	40	0

Source: https://www.kaggle.com/rounakbanik/pokemon/data

Real life example of importing tabular data

Prepare your dataset

- if your data is in excel export it as csv file.
  - or tab separated file (*.tsv)

```
# Read the *.csv file
pokemon_data <- read.csv(
  file='Data/pokemon_data.csv',
  sep=',',
  header=TRUE
)

# Get an over view of the data
dim(pokemon_data)
```

```
## [1] 801  16
```

Get overview of data frame

```
# See top 10 lines
head(pokemon_data)
```

```
##   Pokedex.number       Name percentage.male type1  type2 weight_kg
## 1              1  Bulbasaur            88.1 grass poison       6.9
## 2              2    Ivysaur            88.1 grass poison      13.0
## 3              3   Venusaur            88.1 grass poison     100.0
## 4              4 Charmander            88.1  fire              8.5
## 5              5 Charmeleon            88.1  fire             19.0
## 6              6  Charizard            88.1  fire flying      90.5
##   height_m generation classfication defense hp attack special.attack
## 1      0.7          1          Seed      49 45     49             65
## 2      1.0          1          Seed      63 60     62             80
## 3      2.0          1          Seed     123 80    100            122
## 4      0.6          1        Lizard      43 39     52             60
## 5      1.1          1         Flame      58 58     64             80
## 6      1.7          1         Flame      78 78    104            159
##   special.defense speed Is.legendary
## 1              65    45            0
## 2              80    60            0
## 3             120    80            0
## 4              50    65            0
## 5              65    80            0
## 6             115   100            0
```

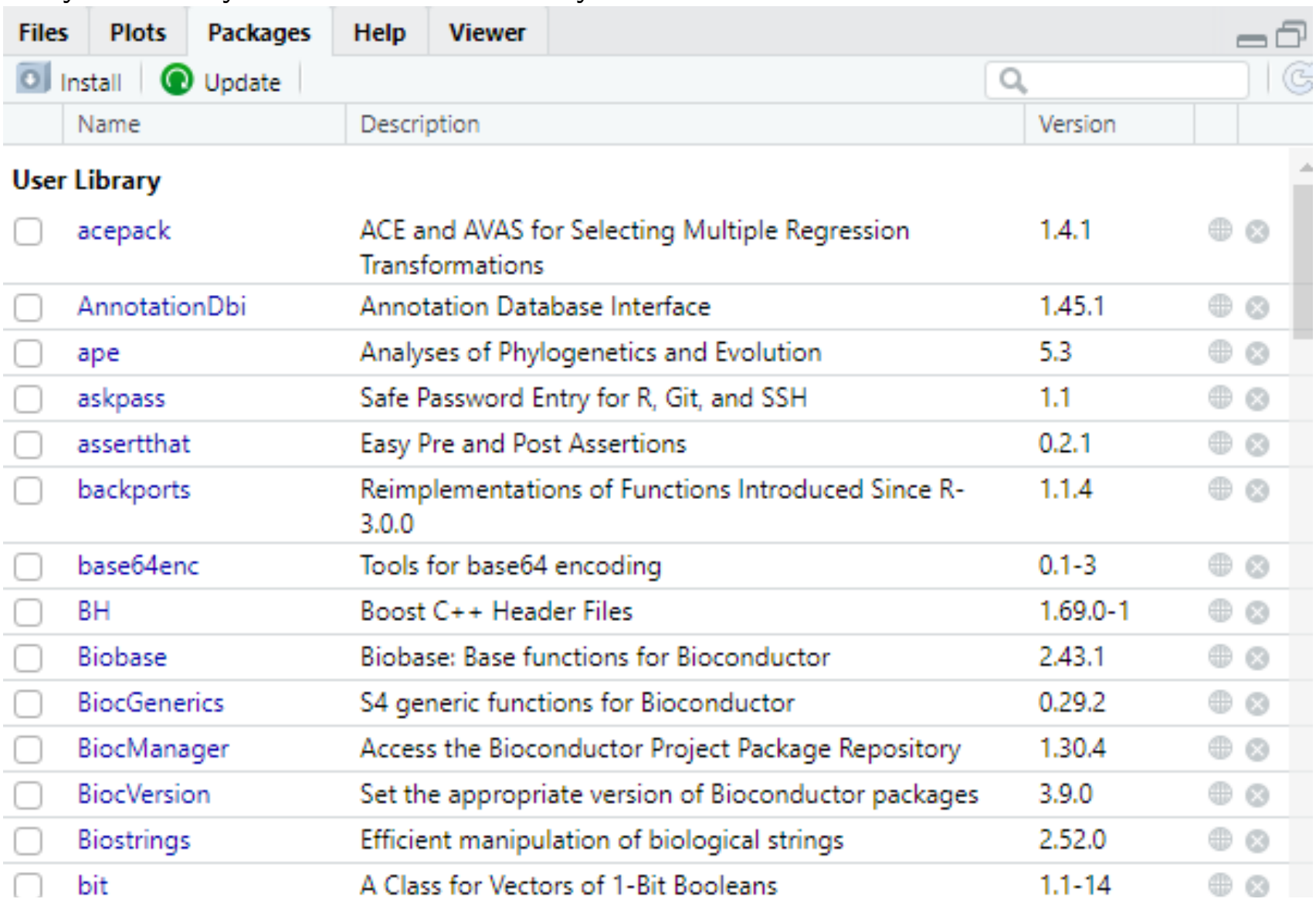
Get overview of data frame

```
# See datatypes
str(pokemon_data)
```

```
## 'data.frame':    801 obs. of  16 variables:
##  $ Pokedex.number : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Name           : Factor w/ 801 levels "Abomasnow","Abra",..: 73 321 745 95 96
##  $ percentage.male: num  88.1 88.1 88.1 88.1 88.1 88.1 88.1 88.1 88.1 50 ...
##  $ type1          : Factor w/ 18 levels "bug","dark","dragon",..: 10 10 10 7 7 7
##  $ type2          : Factor w/ 19 levels "","bug","dark",..: 15 15 15 1 1 9 1 1 1
##  $ weight_kg      : num  6.9 13 100 8.5 19 90.5 9 22.5 85.5 2.9 ...
##  $ height_m       : num  0.7 1 2 0.6 1.1 1.7 0.5 1 1.6 0.3 ...
##  $ generation     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ classfication  : Factor w/ 588 levels "Abundance","Acorn",..: 449 449 449 299
##  $ defense        : int  49 63 123 43 58 78 65 80 120 35 ...
##  $ hp             : int  45 60 80 39 58 78 44 59 79 45 ...
##  $ attack         : int  49 62 100 52 64 104 48 63 103 30 ...
##  $ special.attack : int  65 80 122 60 80 159 50 65 135 20 ...
##  $ special.defense: int  65 80 120 50 65 115 64 80 115 20 ...
##  $ speed          : int  45 60 80 65 80 100 43 58 78 45 ...
##  $ Is.legendary   : int  0 0 0 0 0 0 0 0 0 0 ...
```

Packages available in R

- a packages is a free libraries of code written by R's active user community.
- They add many useful R functions to your R environment



Install external packages

- To install an R package, open an R session and type at the command line `install.packages("<the package's name>")`
- install an package to read Excel files
- search on internet or r-cran.org

```
## Install many packages as a vector
install.packages(
  c('readxl','gplots')
  )
```

- usually installs in "My Documents"

Thank you.