# Exploratory Data Analysis with R

## Introduction to R - Part I

Xuemao Zhang
East Stroudsburg University

September 7, 2022

# Outline

- R console
- R data types
  - Numeric
  - Integer
  - Character
  - Factor
  - Date
  - Logical
  - Complex
- R data structures
  - Vector
  - Matrix
  - List
  - Data Frame
- R data operators

# R console

- You can enter commands one at a time at the command prompt (>). For example,

```
3+5
```

```
## [1] 8
```

The above is the form of code in our presentations: The first line is what I typed into the console; the second line is my result.

**Note:** # is the comment symbol in R.

When you run the code 3+5 in your local console, you type after the command prompt > and it will look like this:

```
> 3+5
[1] 8
```

# R console

- But if we want to do more than one thing with the same data, it's best to store that data in a variable.

```
x=3; y=7;
x;      #print(x)
```

```
## [1] 3
```

```
y;      #print(y)
```

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

```
X;      # R is case sensitive.
```

- R can be used as a caculator. Try the following after class.

```
>3**2
>3^2
>exp(1)
>log(3)    #the base of the log function is e
>pi
>sin(pi)   # sin(pi) is supposed to be zero
>round(sin(pi),4)   #Round sin(pi) to four decimal places
```

# R console: need help?

- use google!
- use help() or ? to seek help.

```
help(log); #This asks for information about the log function
```

```
## starting httpd help server ... done
```

```
?log; #a  shorter way of asking for help
example(log); #This will give some  examples
```

```
##
## log> log(exp(3))
## [1] 3
##
## log> log10(1e7) # = 7
## [1] 7
##
## log> x <- 10^-(1+2*1:9)
##
## log> cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
##              x
##  [1,] 1e-03 9.995003e-04 9.995003e-04 1.000500e-03 1.000500e-03
##  [2,] 1e-05 9.999950e-06 9.999950e-06 1.000005e-05 1.000005e-05
```

# R console: clear work space

- `ls()` lists all variables in the worksapce
- `rm()` removes a variable
- `rm(list=ls(all=TRUE))` deletes all variables in the worksapce

```
ls();
```

```
## [1] "x" "y"
```

```
rm(x);  # x;
y;
```

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

```
rm(list=ls(all=TRUE)); # y;
```

# R data types: Numeric

- Decimal values are called **numerics**.

```
x=3.2;
x;
```

```
## [1] 3.2
```

```
class(x);
```

```
## [1] "numeric"
```

# R data types: Integer

- Decimal values are called **numerics**.

```
x=3;
is.integer(x);  # is x an integer?
```

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

```
class(x);
```

```
## [1] "numeric"
```

```
x=as.integer(x); #create an integer variable using as.integer
class(x);
```

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

# R data types: Character

- A **character** object is used to represent string values.

```
myString <- "Hello, World!"
print(myString);
```

```
## [1] "Hello, World!"
```

```
x = as.character(3.14); #convert an object into character values
x;
```

```
## [1] "3.14"
```

```
class(x);  #check data type
```

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

# R data types: Character

- Two character values can be concatenated with the paste function.

```
fname = "John"; lname ="Smith";
paste(fname, lname);
```

```
## [1] "John Smith"
```

-To extract a substring, we apply the substr function.

```
substr("Mary has a little lamb.", start=3, stop=12)
```

```
## [1] "ry has a l"
```

# R data types: Factor

A `factor` is a special `character` vector where the elements have pre-defined groups or 'levels'. You can think of these as qualitative or categorical variables.

```
x = factor(c("boy", "girl", "girl", "boy", "girl"));
x ;
```

```
## [1] boy  girl girl boy  girl
## Levels: boy girl
```

```
class(x)
```

```
## [1] "factor"
```

Note that the levels are **alphabetically ordered by default**.

# R data types: Factor

- Factors are used to represent categorical data, and can also be used for ordinal data (ie categories have an intrinsic ordering)
- The function `factor` is used to encode a vector as a factor (the terms 'category' and 'enumerated type' are also used for factors). If argument ordered is TRUE, the factor levels are assumed to be ordered.

```
factor(x = character(), levels, labels = levels,
       exclude = NA, ordered = is.ordered(x), nmax = NA)
```

```
x = factor(c("boy", "girl", "girl", "boy", "girl"),
           levels=c("girl","boy"));
x ;
```

```
## [1] boy  girl girl boy  girl
## Levels: girl boy
```

# R data types: Factor

Factors can be converted to numeric or character very easily

```
casecontrol = c("case","case","case","control",
                "control","control");
x = factor(casecontrol, levels = c("control","case"), ordered=TRUE);
x;
```

```
## [1] case    case    case    control control control
## Levels: control < case
```

```
as.character(x);
```

```
## [1] "case"    "case"    "case"    "control" "control" "control"
```

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

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

# R data types: Factor

However, you need to be careful modifying the labels of existing factors, as its quite easy to alter the meaning of the underlying data.

```
xCopy = x;
levels(xCopy) = c("case", "control") # wrong way!
xCopy;
```

```
## [1] control control control case    case    case
## Levels: case < control
```

```
as.character(xCopy); # labels switched
```

```
## [1] "control" "control" "control" "case"    "case"    "case"
```

```
as.numeric(xCopy);
```

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

# R data types: Date

- Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates.

```
Sys.Date( );  # it returns today's date
```

```
## [1] "2022-09-01"
```

```
date();       # it returns the current date and time
```

```
## [1] "Thu Sep  1 09:34:44 2022"
```

# R data types: Date

You can convert date-like strings in the Date class (http://www.statmethods.net/input/dates.html for more info)

- Character to Date Conversion

```
strdates = c("01/05/1965", "08/16/1975");
dates = as.Date(strdates, "%m/%d/%Y");
# convert date info in format mm/dd/yyyy
dates;
```

```
## [1] "1965-01-05" "1975-08-16"
```

- Date to Character Conversion

```
strDates <- as.character(dates);
strDates;
```

```
## [1] "1965-01-05" "1975-08-16"
```

- For more information, help(as.Date) and help(strptime).

# R data types: Date

However, the lubridate package is much easier for generating explicit dates:

```
strdates;
```

```
## [1] "01/05/1965" "08/16/1975"
```

```
library(lubridate); # great for dates!
newDate2 = mdy(strdates);
newDate2;
```

```
## [1] "1965-01-05" "1975-08-16"
```

```
strdates;
```

```
## [1] "01/05/1965" "08/16/1975"
```

```
library(lubridate); # great for dates!
newDate2 = mdy(strdates);
newDate2;
```

```
## [1] "1965-01-05" "1975-08-16"
```

```
newDate2[1]; newDate2[2];
```

# R data types: Date

- The `POSIXct` class is like a more general date format (with hours, minutes, seconds).

```
theTime = Sys.time();
theTime;
```

```
## [1] "2022-09-01 09:34:44 EDT"
```

```
class(theTime);
```

```
## [1] "POSIXct" "POSIXt"
```

```
theTime + as.period(30, unit = "minutes"); # the future
```

```
## [1] "2022-09-01 10:04:44 EDT"
```

# R data types: Logical

- A **logical** value is often created via comparison between variables.

```
x = 1; y = 2;    # sample values
z = x > y;       # is x larger than y?
z;               # print the logical value
```

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

```
class(z);        # print the class name of z
```

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

# R data types: Logical

- Standard logical operations are & (and), | (or), and ! (negation).

```
u = TRUE; v = FALSE
u & v              # u AND v
```

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

```
u | v              # u OR v
```

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

```
!u                 # negation of u
```

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

# R data types: Complex

- A **complex** value in R is defined via the pure imaginary value `i`(skip this data type).

```
z = 1+2i;          # create a complex number
z;                 # print the value of z
```

```
## [1] 1+2i
```

```
class(z);          # print the class name of z
```

```
## [1] "complex"
```

```
sqrt(-1);          # square root of -1 which is not a complex value
```

```
## Warning in sqrt(-1): NaNs produced
```

```
## [1] NaN
```

```
sqrt(-1+0i);       # square root of -1+0i
```

```
## [1] 0+1i
```

- An alternative is to coerce -1 into a complex value.

```
sqrt(as.complex(-1));
```

```
## [1] 0+1i
```

# R data strctures: Vector

- The basic data structure in R is the vector, a sequence of data elements of the same basic type. In order to create a vector in R Programming, c() function is used. We have seen several examples.

```
A=c(2, 3, 5);  A;
```

```
## [1] 2 3 5
```

```
B=c(TRUE, FALSE, TRUE, FALSE, FALSE); B;
```

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

```
C=c("aa", "bb", "cc", "dd", "ee"); C;
```

```
## [1] "aa" "bb" "cc" "dd" "ee"
```

```
length(C);  #how many elements it contains
```

```
## [1] 5
```

# R data strctures: Vector

- Refer to elements of a vector using subscripts.

```
A=c(2, 3, 5);  A[1]; A[c(1,3)];
```

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

```
## [1] 2 5
```

- Combining Vectors

```
A=c(2, 3, 5);
B = c("aa", "bb", "cc", "dd", "ee");
c(A, B);
```

```
## [1] "2"  "3"  "5"  "aa" "bb" "cc" "dd" "ee"
```

**Note**: In the above, numeric values are being coerced into character strings when the two vectors are combined. This is necessary so as to maintain the same primitive data type for elmentss in the same vector.

# R data strctures: Matrix

- Matrix is a two-dimensional structure where all values are of the **same type**. There are various ways to construct a matrix.

```
B = matrix( c(2, 4, 3, 1, 5, 7),  nrow=3, ncol=2); B;
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    4    5
## [3,]    3    7
```

```
C = matrix( c(2, 4, 3, 1, 5, 7),  nrow=3, ncol=2,byrow=TRUE); C;
```

```
##      [,1] [,2]
## [1,]    2    4
## [2,]    3    1
## [3,]    5    7
```

# R data strctures: Matrix

- Combining Matrices using row bind: rbind() or column bind: cbind().

```
B = matrix( c(2, 4, 3, 1, 5, 7),  nrow=3, ncol=2);
D=matrix( c(7, 4, 2), nrow=3,  ncol=1);  # D has 3 rows
cbind(B, D);
```

```
##      [,1] [,2] [,3]
## [1,]    2    1    7
## [2,]    4    5    4
## [3,]    3    7    2
```

```
E= matrix( c(6, 2), nrow=1, ncol=2); #E has 2 columns
rbind(B, E);
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    4    5
## [3,]    3    7
## [4,]    6    2
```

# R data strctures: List

- A list is a generic vector containing **other objects**. The following example creats a list containing a vector and a matrix.

```
T =list();
A=c("aa", "bb", "cc", "dd", "ee");
B = matrix( c(2, 4, 3, 1, 5, 7),  nrow=3, ncol=2);
T[[1]]=A;  T[[2]]=B;
T;       #print T
```

```
## [[1]]
## [1] "aa" "bb" "cc" "dd" "ee"
##
## [[2]]
##      [,1] [,2]
## [1,]    2    1
## [2,]    4    5
## [3,]    3    7
```

# R data strctures: List

- We can assign names to list members, and reference them by names instead of numeric indexes.

```
V = list(bob=c(2, 3, 5), john=c("aa", "bb")) ;
V["bob"];
```

```
## $bob
## [1] 2 3 5
```

```
V[c("john", "bob")];
```

```
## $john
## [1] "aa" "bb"
##
## $bob
## [1] 2 3 5
```

# R data strctures: Data Frame

- A data frame is used for storing data tables. It is a collection of vectors that all have the same length. This is like a matrix, except that different columns can different data types.
- The column names should be non-empty.
- The column names should be unique.
- Each column should contain same number of data items.
- The data stored in a data frame can be of any type.

# R data strctures: Data Frame

- The data set mtcars is a built-in data frame in R.

```
names(mtcars);   #list the variable names of the data
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gea
## [11] "carb"
```

```
#colnames(mtcars)

rownames(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

# R data strctures: Data Frame

- The data set mtcars is a built-in data frame in R.

```
head(mtcars);   #show the header of the data
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear c
## Mazda RX4        21.0   6  160 110 3.90 2.620 16.46  0  1    4
## Mazda RX4 Wag    21.0   6  160 110 3.90 2.875 17.02  0  1    4
## Datsun 710       22.8   4  108  93 3.85 2.320 18.61  1  1    4
## Hornet 4 Drive   21.4   6  258 110 3.08 3.215 19.44  1  0    3
## Hornet Sportabout 18.7  8  360 175 3.15 3.440 17.02  0  0    3
## Valiant          18.1   6  225 105 2.76 3.460 20.22  1  0    3
```

```
head(mtcars, n=4);
```

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

# R data strctures: Data Frame

```
tail(mtcars);   #show the last several rows
```

```
##                  mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2   26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa    30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L  15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino    19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora   15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E      21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```
tail(mtcars, n=4);
```

```
##                  mpg cyl disp  hp drat   wt qsec vs am gear carb
## Ford Pantera L  15.8   8  351 264 4.22 3.17 14.5  0  1    5    4
## Ferrari Dino    19.7   6  145 175 3.62 2.77 15.5  0  1    5    6
## Maserati Bora   15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
## Volvo 142E      21.4   4  121 109 4.11 2.78 18.6  1  1    4    2
```

# R data strctures: Data Frame

```
str(mtcars);   #display the internal structure
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

# R data strctures: Data Frame

```
nrow(mtcars);     # number of data rows
```

```
## [1] 32
```

```
ncol(mtcars);     # number of columns
```

```
## [1] 11
```

```
dim(mtcars);      # dimensions of the data
```

```
## [1] 32 11
```

# R data strctures: Data Frame

- We can extract specific column from a data frame using column name.

```
names1=c("mpg","cyl","hp","wt");
mtcars1=mtcars[names1]; #extract the variables: mpg, cyl,hp, wt
str(mtcars1);
```

```
## 'data.frame':    32 obs. of  4 variables:
##  $ mpg: num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl: num  6 6 4 6 8 6 8 4 4 6 ...
##  $ hp : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ wt : num  2.62 2.88 2.32 3.21 3.44 ...
```

```
mpg=mtcars$mpg;   #extract the variable mpg using the $ operator
str(mpg);
```

```
##  num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
```

# R data strctures: Data Frame

- We can add a column to a data frame by adding the column vector using a new column name.

```
mtcars0=mtcars1;        # create a copy of mtcars1
wt2=(mtcars0$wt)^2;     # define wt2, the square of wt
mtcars0$wt2=wt2;        #Add a new column wt2
head(mtcars0);
```

```
##                   mpg cyl  hp    wt       wt2
## Mazda RX4        21.0   6 110 2.620  6.864400
## Mazda RX4 Wag    21.0   6 110 2.875  8.265625
## Datsun 710       22.8   4  93 2.320  5.382400
## Hornet 4 Drive   21.4   6 110 3.215 10.336225
## Hornet Sportabout 18.7  8 175 3.440 11.833600
## Valiant          18.1   6 105 3.460 11.971600
```

# R data strctures: Data Frame

- We can remove a column as well.

```
mtcars0$wt2=NULL;
head(mtcars0);
```

```
##                    mpg cyl  hp    wt
## Mazda RX4         21.0   6 110 2.620
## Mazda RX4 Wag     21.0   6 110 2.875
## Datsun 710        22.8   4  93 2.320
## Hornet 4 Drive    21.4   6 110 3.215
## Hornet Sportabout 18.7   8 175 3.440
## Valiant           18.1   6 105 3.460
```

# R data strctures: Data Frame

- We can extract specific rows from a data frame.

```
mtcars2=mtcars1[1:5,]    #extract the first 5 rows
mtcars2;
```

```
##                   mpg cyl  hp    wt
## Mazda RX4          21.0   6 110 2.620
## Mazda RX4 Wag      21.0   6 110 2.875
## Datsun 710         22.8   4  93 2.320
## Hornet 4 Drive     21.4   6 110 3.215
## Hornet Sportabout 18.7   8 175 3.440
```

# R data strctures: Data Frame

- We can add more rows permanently to an existing data frame using the rbind() function.

```
newrow=c(30, 4, 110, 2.578);
mtcars3=rbind(mtcars1, newrow);
tail(mtcars3);
```

```
##                  mpg cyl  hp    wt
## Lotus Europa    30.4   4 113 1.513
## Ford Pantera L 15.8   8 264 3.170
## Ferrari Dino    19.7   6 175 2.770
## Maserati Bora   15.0   8 335 3.570
## Volvo 142E      21.4   4 109 2.780
## 33              30.0   4 110 2.578
```

# R data strctures: Data Frame

- We can remove some rows as well.

```
mtcars4=mtcars3[-c(32,33),];   #remove the last two rows: 32 and 33
tail(mtcars4);
```

```
##                  mpg cyl  hp    wt
## Fiat X1-9       27.3   4  66 1.935
## Porsche 914-2   26.0   4  91 2.140
## Lotus Europa    30.4   4 113 1.513
## Ford Pantera L  15.8   8 264 3.170
## Ferrari Dino    19.7   6 175 2.770
## Maserati Bora   15.0   8 335 3.570
```

# R data strctures: Data Frame

```
class(mtcars1$cyl);
```

```
## [1] "numeric"
```

```
mtcars1$cyl=as.factor(mtcars1$cyl); #mtcars1$cyl=factor(mtcars1$cyl,
str(mtcars1);
```

```
## 'data.frame':    32 obs. of  4 variables:
##  $ mpg: num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl: Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
##  $ hp : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ wt : num  2.62 2.88 2.32 3.21 3.44 ...
```

```
levels(mtcars1$cyl); #list levels of the factor
```

```
## [1] "4" "6" "8"
```

# R operators

**Arithmetic Operators**: +,-, *, ^, /

- The operators act on each element of a vector.

```
v = c(2,5,6);  w = c(8,3,6);
v+w;      #sum of two vectors
```

```
## [1] 10  8 12
```

```
v-w;      #difference of two vectors
```

```
## [1] -6  2  0
```

```
v*w;      #product of two vectors
```

```
## [1] 16 15 36
```

```
v/w;      #quotient
```

```
## [1] 0.250000 1.666667 1.000000
```

```
v^2;      #square of a vector
```

```
## [1]  4 25 36
```

# R operators

Matrix manipulation in R are very useful in Linear Algebra. For more information, see Matrix Operators.

- Transpose – `t`
- Multiplication – `%*%`
- Determinant – `det`
- Inverse – `solve`, or `ginv` of MASS library
- Eigenvalues and Eigenvectors – `eigen`

# R operators

**Relational Operators**: >,<, ==, >=, <=, !=

```
v = c(2,5,6);  w = c(8,3,6); v>w;
```

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

```
v<w;
```

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

```
v==w;
```

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

```
v>=w;
```

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

```
v!=w;
```

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

# R operators

**Assignment Operators**: =, <-, c

```
v1 <- c(3,1,TRUE,2);
v1;
```

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

```
v2 = c(3,1,TRUE,2);
v2;
```

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

**Right Assignment Operator**: ->

```
c(3,1,F,2) -> v3;
v3;
```

```
## [1] 3 1 0 2
```

# R operators

- : Colon operator. It creates the series of numbers in sequence for a vector.

```
v = 2:8; v;
```

```
## [1] 2 3 4 5 6 7 8
```

- :: Double Colon operator. Two functions in two different R packages may have the same name. To specify the package that you want to use, the syntax is `package::function()`

# R operators

- %in% It is used to identify if an element belongs to a vector.

```
v1 = 8; v2 = 12; w = 1:10;
print(v1 %in% w);
```

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

```
print(v2 %in% w);
```

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

# License