Exercises, Examples, and Notes with R

Nith Kosal

May 21, 2021

Introduction to R

Exercise 1: The installation of R and RStudio

An up-to-date version of R may be downloaded from a Comprehensive R Archive Network (CRAN) mirror site. There are links at http://cran.r-project.org/. Installation instructions are provided at the web site for installing R in Windows, Mac, and Linux. At the time, you need use https://www.rstudio.com/products/rstudio/ to download the RStudio Desktop. RStudio is an integrated development environment (IDE) for R programming language for statistical computing and graphics. You can not use R without RStudio. If you are already installed, please move to Exercise 2.

Exercise 2: Using the console window

Now, we start do the exercise with RStudio application. please open RStudio program in your computer. The command line prompt (`>`) is an invitation to type commands or expressions. Once the command or expression is complete, and the **Enter** key is pressed, R evaluates and prints the result in the console window. This allows the use of R as a calculator. For example, type `2+2` and press the **Enter** key. Here is what appears on the screen:

```
> 2021 + 2021
```

```
[1] 4042
```

The first element is labeled `[1]` even when, as here, there is just one element!

```
# Thus, please type in the console like the list blows:
```

1. `23 + 27 + 29`
2. `239 * 39 / 34`
3. `sqrt(2021)` # the suare root of 2021
4. `pi` # R knows about pi
5. `2 * pi * 6378`
6. `34 * 4^4; (34 * 4)^2; 2021 / 21`
7. `"Cambodia"; "Thailand"; "Vietnam"; "Laos"`
8. `1:40`

Anything that follows a `#` on the command line is taken as comment and ignored by R. A continuation prompt, by default `+`, appears following a carriage return when the command is not yet complete. For example, an interruption of the calculation of `23*3^4` by a carriage return could appear as:

```
> 23 * 3^
```

```
+ 4
```

```
[1] 1863
```

Exercise 3: Type and run Excercise 2 from 1 to 8 in a R Script file

By doing so, you need create a new R Scrip file and save it in a specific name that you want to. Anything that follows a `#` on the command line is taken as comment and ignored by R. A continuation prompt, by default `+`, appears following a carriage return when the command is not yet complete.For example, an interruption of the calculation of `23*3^4` by a carriage return could appear as:

```
# Calculate number below:
23 + 27 + 29
```

```
## [1] 79
```

```
23 * 3^
  + 4
```

```
## [1] 1863
```

```
# Please type and run other values from Exercise 3 in R Script file
```

Exercise 4: Create objects in your R Script

The `<-` is a left angle bracket (`<`) followed by a minus sign (`-`). It means "the values on the right are assigned to the name on the left".

```
par(mar = c(4, 4, .1, .1)) # Use to reduce space between figure and caption in Rmd file

# For example
apple <- c(20, 21, 30, 21, 18, 23, 35, 40, 12, 37, 16, 32)
FutureF <- c("Young Researcher", "Junior Researcher", "Researcher", "Editor")
FutureF     # Display the contents of the vector.
```

```
## [1] "Young Researcher"  "Junior Researcher" "Researcher"
## [4] "Editor"
```

```
mango <- c(24:30, 40:44)
tf <- c(T, F, F, F, T, T, F)     # the logical value
tf
```

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

```
a <- c(23, 25, 34, 21, 45, 32)
a > 25
```

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

```
a != 25
```

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

```
a[-c(2,5)]
```

```
## [1] 23 34 21 32
```

```
heights <- c(Andreas=178, John=185, Jeff=183)
heights[c("John","Jeff")]
```

```
## John Jeff
##  185  183
```

```
b <- seq(from=5, to=22, by=4) #  The first value is 5, the final value is <=22.
c <- rep(c(2,3,5), 5) # To repeat the sequence (2, 3, 5) five times over.
```

```
plot(apple) # Panel left
plot(mango ~ apple, pch=16) # Panel right, pch is "plot character": a solid black dot.
```

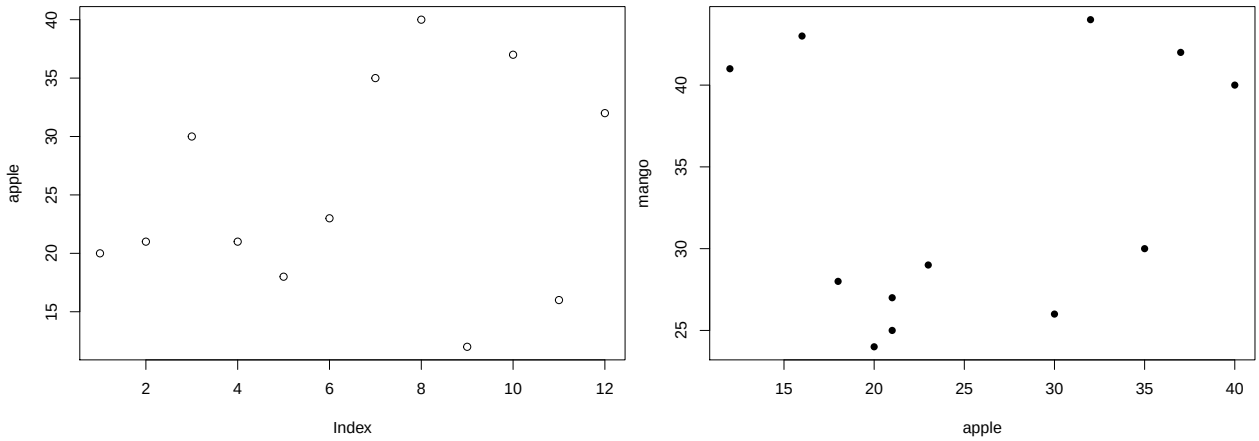

Figure 1: Apple, and Apple vs Mango

```
# 1. Please create object or data frame "year" which values from 2000-2021,
# object "x" which values from 0-21 , and object "y" have values such as 301,
# 320, 230, 240, 302, 400, 340, 260, 491, 350, 230, 360, 325, 452, 298, 345, and 264.

# 2. Plot "y" and plot "x" against "Year" as in Figure 1.

# 3. Create a object with the logical value.
```

Reminder: In R and other programming languages, the case is significant for names of objects or commands. Apple is different from apple. Relational operators are `<`, `<=`, `>`, `>=`, `==`, and `!=`.

Once created, the objects `Year` and `x`, `y` are stored in the Environment (workspace), as part of the user's working collection of R objects.

Exercise 5: Collection of vectors into a data frame

The two vectors `apple` and `mango` created earlier are matched, element for element. It is convenient to group them together into an object that has the name `data.frame`, thus:

```
fruit <- data.frame(Apple = apple, Mango = mango) # Apple is new object get from "apple".
fruit # Display the contents of the data frame.
```

```
##    Apple Mango
## 1     20    24
## 2     21    25
## 3     30    26
## 4     21    27
## 5     18    28
## 6     23    29
## 7     35    30
## 8     40    40
## 9     12    41
## 10    37    42
## 11    16    43
## 12    32    44
```

```
rm(apple, mango)  # The rm() function removes unwanted objects.
plot(Apple ~ Mango, data=fruit) # Plot Apple and Mango from data frame.
```

```
# 1. Please create a data frame that's name "xdata" by using your recent objects name
# "year" and "x".

# 2. Removes your previous objects "y", "x", and "year".

# 3. Please plot your new two objects together, that just getting from data frame "xdata".
```

We have several ways to identify columns by name in data frame. Here, note that the second column can be referred to as `fruit[, 2]`, or as `fruit[, "Mango"]`, or as `fruit$mango`. For example:

```
fruit[, "Mango"]
```

```
##  [1] 24 25 26 27 28 29 30 40 41 42 43 44
```

Data frames are the preferred way to organize data sets that are of modest size. For now, think of data frames as a rectangular row by column layout, where the rows are `observations` and the columns are `variables`. As you see in example in Exercise 5.

```
# 4. Please type your data frame "xdata" like this xdata[1]

# 5. Please type getwd() to display the name of the working directory.

# 6. Please type ls() to list your the workspace contents.
```

Use the `q()` function to quit (exit) from R. There will be a message asking whether to save the workspace image. Clicking **Yes** has the effect that, before quitting, all the objects that remain in the workspace are saved in a file that has the name `.RData`.

Exercise 6: Installation and utilization of packages

R comes with a standard set of packages. Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library. Assuming access to a live internet connection, packages can be installed pretty much automatically. For installation from the command line, enter, for example:

```
install.packages("DAAG")      # Basic method
install.packages(c("magic", "schoolmath"), dependencies=TRUE)  # Advanced method
```

```
# Please install package name: DAAG, magic, schoolmath, tidyverse, broom, coefplot,
# cowplot, drat, fs, gapminder, GGally, ggrepel, ggridges, gridExtra, haven, here,
# interplot, margins, maps, mapproj, mapdata, MASS, quantreg, rlang, scales, survey,
# srvyr, viridis, viridisLite, devtools.
```

If you have already installed it in the previous period, please ignore it and proceed to the next exercise.

To use function and data set from package, we use `library()` to call package.

```
library(tidyverse) # Basic method
sessionInfo()      # To see which packages are currently attached.

# Load multiple packages at once
lapply(c("gganimate", "tidyverse", "gapminder"), require, character.only = TRUE)
```

Help: `?pot` is equivalent to `help(plot)`. `apropos("sort")` and `help.search("sort")`: search for function that perform a desired task. Using the function `example()` to run the examples on the relevant help page.

For example: `example("image")`, and `par(ask=FALSE)` use for turn off the prompts.

Exercise 7: Missing values

The missing value symbol is `NA`. As an example, consider the column `branch` of the data set `rainforest` from library "DAAG":

```
library(DAAG)
```

```
## Warning: package 'DAAG' was built under R version 3.6.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.6.3
```

```
rforest <- subset(rainforest, species=="Acacia mabellae")$branch
rforest
```

```
##  [1] NA 35 41 50 NA NA NA NA NA  4 30 13 10 17 46 92
```

```
sum(rforest)
```

```
## [1] NA
```

```
# Omitted all missing values (NA) before proceeding with the calculation.
sum(rforest, na.rm = TRUE)
```

```
## [1] 338
```

```
rforest[is.na(rforest)] <- 11 # To replace all NAs by 11 use the function is.na().
rforest
```

```
##  [1] 11 35 41 50 11 11 11 11 11  4 30 13 10 17 46 92
```

```
mean(rforest)
```

```
## [1] 25.25
```

```
# 1. Please load library(DAAG), create new object name "rrootsk" by selecting "rootsk"
# variable in the "rainforest" data set and print your new object.

# 2. Replace your missing value in "rrootsk" object (vector) by -100.
```

Example on data frame

```
par(mar = c(2, 4, 1, 0)) # Use to reduce space between figure and caption in Rmd file


# Create character vector
gender <- c(rep("female",200), rep("male",201))
levels(gender)     # For a character vector, this returns NULL
```

```
## NULL
```

```
# From character vector, create factor
gender <- factor(gender)
levels(gender)
```

```
## [1] "female" "male"
```

```
nincome <- c(982,981,984,982,981,983,983,983,983,979,973,979,
             974,981,985,987,986,980,983,983,988,994,990,999)
plot(nincome)  # Panel left
# Converts numeric vectors into time series objects by using function "ts()"
nincome <- ts(nincome, start=1995, frequency = 2)
plot(nincome)  # Penal right

# Use the function "window()" to extract a subset of the time series.
# Extracts the last quarter of 1995 and the first few months of 1996
fnincome <- window(nincome, start=1995.75, end=1996.25)
```

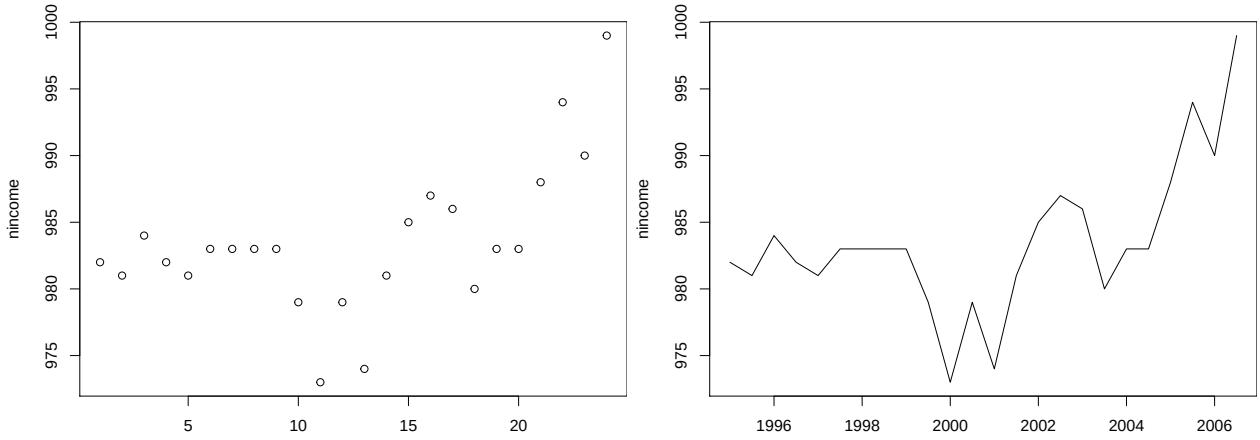

Figure 2: Income of individuals

```
# Use data frame from package "DAAG"
Cars93.summary
```

```
##         Min.passengers Max.passengers No.of.cars abbrev
## Compact              4              6         16      C
## Large                6              6         11      L
## Midsize              4              6         22      M
## Small                4              5         21     Sm
## Sporty               2              4         14     Sp
## Van                  7              8          9      V
```

```
# Function "edit()" use for edit observations.
```

The first three columns are numeric, and the fourth is a factor. Use the function `class()` to check this:

```
class(Cars93.summary$abbrev)
```

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

```
head(Cars93.summary, n=3) # Display the first 3 rows (the default is 6)
```

```
##         Min.passengers Max.passengers No.of.cars abbrev
## Compact              4              6         16      C
## Large                6              6         11      L
## Midsize              4              6         22      M
```

```
rownames(Cars93.summary) # Extract row names
```

```
## [1] "Compact" "Large"   "Midsize" "Small"   "Sporty"  "Van"
```

```
colnames(Cars93.summary) # Extract column names
```

```
## [1] "Min.passengers" "Max.passengers" "No.of.cars"     "abbrev"
```

```
# The functions names() or colnames() and rownames() can also be used to
# assign new names. For example:
names(Cars93.summary)[3] <- "numCars"
names(Cars93.summary) <- c("minPass","maxPass","numCars","code")
Cars93.summary
```

```
##         minPass maxPass numCars code
## Compact       4       6      16    C
## Large         6       6      11    L
## Midsize       4       6      22    M
## Small         4       5      21   Sm
## Sporty        2       4      14   Sp
## Van           7       8       9    V
```

```
# Subsets of data frames
Cars93.summary[1:3, 2:3]   # Rows 1-3 and columns 2-3
```

```
##         maxPass numCars
## Compact       6      16
## Large         6      11
## Midsize       6      22
```

```
Cars93.summary[, 2:3]      # Columns 2-3 (all rows)
```

```
##         maxPass numCars
## Compact       6      16
## Large         6      11
## Midsize       6      22
## Small         5      21
## Sporty        4      14
## Van           8       9
```

```
Cars93.summary[, -c(2,3)]  # omit columns 2 and 3
```

```
##         minPass code
## Compact       4    C
## Large         6    L
## Midsize       4    M
## Small         4   Sm
## Sporty        2   Sp
## Van           7    V
```

```
Cars93.summary[, c("maxPass", "code")]   # Cols 2-3, by name
```

```
##         maxPass code
## Compact       6    C
## Large         6    L
## Midsize       6    M
## Small         5   Sm
## Sporty        4   Sp
## Van           8    V
```

```
Cars93.summary[, -c(2,3)]  # omit columns 2 and 3
```

```
##         minPass code
## Compact       4    C
## Large         6    L
## Midsize       4    M
## Small         4   Sm
## Sporty        2   Sp
## Van           7    V
```

```
subset(Cars93.summary,
       subset=c(TRUE, TRUE, FALSE, FALSE, TRUE, FALSE))
```

```
##         minPass maxPass numCars code
## Compact       4       6      16    C
## Large         6       6      11    L
## Sporty        2       4      14   Sp
```

```
# Data frames are a specialized type of list
# Cities with more than 2.5 million inhabitants
USACanada <- list(USACities=c("NY", "LA", "Chicago"),
                  CanadaCities=c("Toronto", "Montreal"),
                  millionsPop=c(USA=305.9, Canada=31.6))
USACanada
```

```
## $USACities
## [1] "NY"      "LA"      "Chicago"
##
## $CanadaCities
## [1] "Toronto"  "Montreal"
##
## $millionsPop
##    USA Canada
##  305.9   31.6
```

```
# The function with() is often helpful in this connection. Thus, an
# alternative to c(mean(cfseal$weight), median(cfseal$weight)) is:
with(cfseal, c(mean(weight), median(weight)))
```

```
## [1] 54.79 46.25
```

```
with(pair65,       # stretch of rubber bands, from DAAG
  {lenchange <- heated-ambient
   c(mean(lenchange), median(lenchange))
})
```

```
## [1] 6.333333 6.000000
```

```
attach(fossilfuel)  # Attach data frame fossilfuel
year
```

```
## [1] 1800 1850 1900 1950 2000
```

```
detach(fossilfuel)  # Detach data frame

# Aggregation, stacking and unstacking
chickwtAvs <- with(chickwts,
                   aggregate(weight, by=list(feed), mean))
names(chickwtAvs) <- c("Feed Group", "Mean Weight")
chickwtAvs
```

```
##   Feed Group Mean Weight
## 1     casein    323.5833
## 2  horsebean    160.2000
## 3    linseed    218.7500
## 4   meatmeal    276.9091
## 5    soybean    246.4286
## 6  sunflower    328.9167
```

```
head(jobs,3) # A data frame from library "DAAG"
```

```
##     BC Alberta Prairies Ontario Quebec Atlantic     Date
## 1 1752    1366      982    5239   3196      947 95.00000
## 2 1737    1369      981    5233   3205      946 95.08333
## 3 1765    1380      984    5212   3191      954 95.16667
```

```
Jobs <- stack(jobs, select = 3:6)
  # stack() concatenates selected data frame columns into a
  # single column named "values", & adds a factor named "ind"
  # that has the names of the concatenated columns as levels.
head(Jobs,3)
```

```
##   values      ind
## 1    982 Prairies
## 2    981 Prairies
## 3    984 Prairies
```

```
# Data frames and matrices
fossilfuelmat <- matrix(c(1800, 1850, 1900, 1950, 2000,
                          8, 54, 534, 1630, 6611), nrow=5)
colnames(fossilfuel) <- c("year", "carbon")

fossilfuelmat <- cbind(year=c(1800, 1850, 1900, 1950, 2000),
                       carbon=c(8, 54, 534, 1630, 6611))
```

Common useful built-in functions

- `all()` # returns `TRUE` if all values are `TRUE`
- `any()` # returns `TRUE` if any values are `TRUE`
- `args()` # information on the arguments to a function
- `cat()` # prints multiple objects, one after the other
- `cumprod()` # cumulative product
- `cumsum()` # cumulative sum
- `diff()` # form vector of first differences and has one less element than `x`
- `history()` # displays previous commands used
- `is.factor()` # returns TRUE if the argument is a factor
- `is.na()` # returns TRUE if the argument is an `NA` and also `is.logical()`, `is.matrix()`, etc.
- `length()` # number of elements in a vector or of a list
- `ls()` # list names of objects in the workspace
- `mean()` # mean of the elements of a vector
- `median()` # median of the elements of a vector
- `order()` # `x[order(x)]` sorts `x` (by default, NAs are last)
- `print()` # prints a single R object
- `range()` # minimum and maximum value elements of vector
- `sort()` # sort elements into order, by default omitting NAs
- `rev()` # reverse the order of vector elements
- `str()` # information on an R object

- `unique()` # form the vector of distinct values
- `which()` # locates `'TRUE'` indices of logical vectors

- `which.max()` # locates (first) maximum of a numeric vector
- `which.min()` # locates (first) minimum of a numeric vector
- `with()` # do computation using columns of specified data frame