Introductory Exercise - Markdown Documentation Assignment

Exercise 1

The good student

10/03/2021

Contents

Tutorium in R

Basic Arithematic Operations in R

Arithematic operations of vectors are performed member-by-member in R.

Some of the common arithematic operators in R are...

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Exponent (^)

Vector Addition in R

If 2 vectors are of equal length, they are added member-by-member. But if the 2 vectors are of unequal length, their sum is computed by recycling values of the shorter vector. This recycling process will throw a warning if the longer vector is not an integral multiple of the shorter one.

For example....

```
c(3,4,5,6,7)+c(4,5)
```

```
## Warning in c(3, 4, 5, 6, 7) + c(4, 5): longer object length is not a multiple of
## shorter object length
```

```
## [1]  7  9  9 11 11
```

If a scalar value is added to a vector, the operation is performed member-by-member.

For example....

```
c(3,4,5,6,7)+2
```

```
## [1] 5 6 7 8 9
```

Vector Multiplication in R

If a vector is multiplied by a scalar, we would get a vector with each of it's members multiplied by the scalar value.

For example....

```
c(3,4,5,6,7)*2
```

```
## [1]  6  8 10 12 14
```

Vector Exponentiation in R

Raising a vector to the power of x will return a vector with each member raised to the power of x.

For example....

```
c(3,4,5,6,7)^2
```

```
## [1]  9 16 25 36 49
```

Functions in R

Trigonometric functions in R

All trigonometric functions are available in R: the sine, cosine, and tangent functions and their inverse functions. `sin()` function is used to calculate the sine value of the numeric value passed to it as argument. R returns the folowing results for `sin(3.14)` and `sin(pi)`.

```
sin(3.14)
```

```
## [1] 0.001592653
```

```
sin(pi)
```

```
## [1] 1.224647e-16
```

The command `pi` can be used to output the value of pi.

```
pi     # Value of pi
```

```
## [1] 3.141593
```

Exponential function in R

The exponential function, `exp(x)`, calculates the value of e to the power of x (e^x). e is called Euler's number and is the base of the natural logarithms ($\log_e$). It's value is approximately equal to 2.7182818.

```
exp(1)  # Euler's number
```

```
## [1] 2.718282
```

R is not a Computer Algebra System (CAS), it requires numerical values to evaluate. But you can use the library "Ryacas" in order to work with symbolic math.

The code below generates an error in R.

```
exp(-2*t)
```

```
## Error in -2 * t: non-numeric argument to binary operator
```

Generating Artificial Data in R

There are several ways to generate data in R.

Generating data using sequences

The function `seq()` generates a sequence.

The statement below, generates a vector which consists of a sequence of numbers starting from 0 until 5 with a step size of 0.5.

```
t<-seq(from=0,to=5,by=0.5)
```

Executing the code below to produces the following results.

```
exp(-2*t)
```

```
##  [1] 1.000000e+00 3.678794e-01 1.353353e-01 4.978707e-02 1.831564e-02
##  [6] 6.737947e-03 2.478752e-03 9.118820e-04 3.354626e-04 1.234098e-04
## [11] 4.539993e-05
```

Generating data thorough replication

The function `rep()` replicates an object.

`rep(1:3, c(30,40,30))` generates a vector with numbers from 1 to 3 (1:3 indicates the sequence 1,2,3), with 1 being repeated 30 times, 2 being repeated 40 times and 3 being repeated 30 times.

Factors in R are data objects which are used to categorize data and store it as levels. R provides both ordered and unordered factors. Factors are created using the `factor()` function. The character vector of labels are mapped to the levels.

```
group <- factor(rep(1:3, c(30,40,30)), labels = c("group 1", "group 2", "group 3"))
```

Generating random sample data

The `runif()` function generates random deviates of the uniform distribution and is written as `runif(n, min, max)` where n is the number of random samples generated within any interval, defined by the min and the max argument.

The example below generates 100 random samples between the interval 20 and 30.

```
x <- runif(100, 20, 30)
```

`rnorm()` is the R function that simulates random variates having a specified normal distribution. It is written as `rnorm(n,mean,var)` where n is the number of datasets to be simulated. Mean is the mean of the dataset to be simulated and var is the variance covariance matrix.

```
eps <- rnorm(100, 0, 2 )
```

`as.numeric` converts the factor into a numeric or integer.

```
y <- 3 * x + 4 * as.numeric(group) + eps
```

Analysing Data

Joining the variables into a dataset

A data frame is the most common way of storing data in R. A data frame is a table or a two-dimensional array-like structure. A data frame is created using the `data.frame` function. Here a data frame is created with y as column 1, x as column 2 and group as column 3.

```
data.ex <- data.frame(y=y, x=x, group=group)
```

Looking at the data

The `str()` function compactly displays the internal structure of an R object.

```
str(data.ex)
```

```
## 'data.frame':    100 obs. of  3 variables:
##  $ y    : num  78.2 90.3 86.4 93.2 89.3 ...
##  $ x    : num  25 27.8 27.1 29.8 27.3 ...
##  $ group: Factor w/ 3 levels "group 1","group 2",..: 1 1 1 1 1 1 1 1 1 1 ...
```

The `names()` function will return a string vector with the column names of the data frame.

```
names(data.ex)
```

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

The `summary()` function summarizes the data in a data frame.

```
summary(data.ex)
```

```
##        y                x             group
##  Min.   : 63.83   Min.   :20.06   group 1:30
##  1st Qu.: 76.95   1st Qu.:22.65   group 2:40
##  Median : 85.42   Median :25.71   group 3:30
##  Mean   : 84.13   Mean   :25.37
##  3rd Qu.: 90.55   3rd Qu.:27.89
##  Max.   :101.97   Max.   :29.94
```

```
library("xtable")
options(xtable.comment = FALSE)
xtable::xtable(summary(data.ex), type = "latex", comment = FALSE, caption = "Summary of the data set")
```

	y	x	group
X	Min. : 63.83	Min. :20.06	group 1:30
X.1	1st Qu.: 76.95	1st Qu.:22.65	group 2:40
X.2	Median : 85.42	Median :25.71	group 3:30
X.3	Mean : 84.13	Mean :25.37	
X.4	3rd Qu.: 90.55	3rd Qu.:27.89	
X.5	Max. :101.97	Max. :29.94	

Table 1: Summary of the data set

We see that the table generated with the `xtable()` function is nicely formatted and more readable than the one created directly by the `summary()`.

Plotting and Data Visualization in R

Plotting variables directly

The default `plot()` function plots the data as a simple scatter plot. The `lines()` function adds information to a graph. It can not produce a graph on its own. Usually it follows a `plot(x,y)` command that produces a graph.

```
plot(t,exp(-2*t))   # Plot automatically plots points
lines(t,exp(-2*t))  # Add lines manually
```

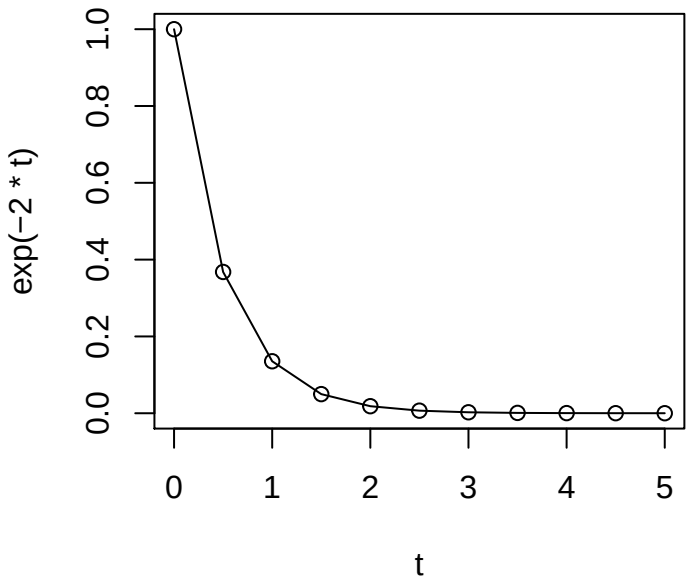


Figure 1: Plotting variables directly

Plotting Data Frames

The `plot()` function can plot data frames. For more than two columns it first calls data.matrix to convert the data frame to a numeric matrix and then calls pairs to produce a scatterplot matrix. In plot 1, a data frame is plotted.

```
plot(data.ex)     # Plot 1
```

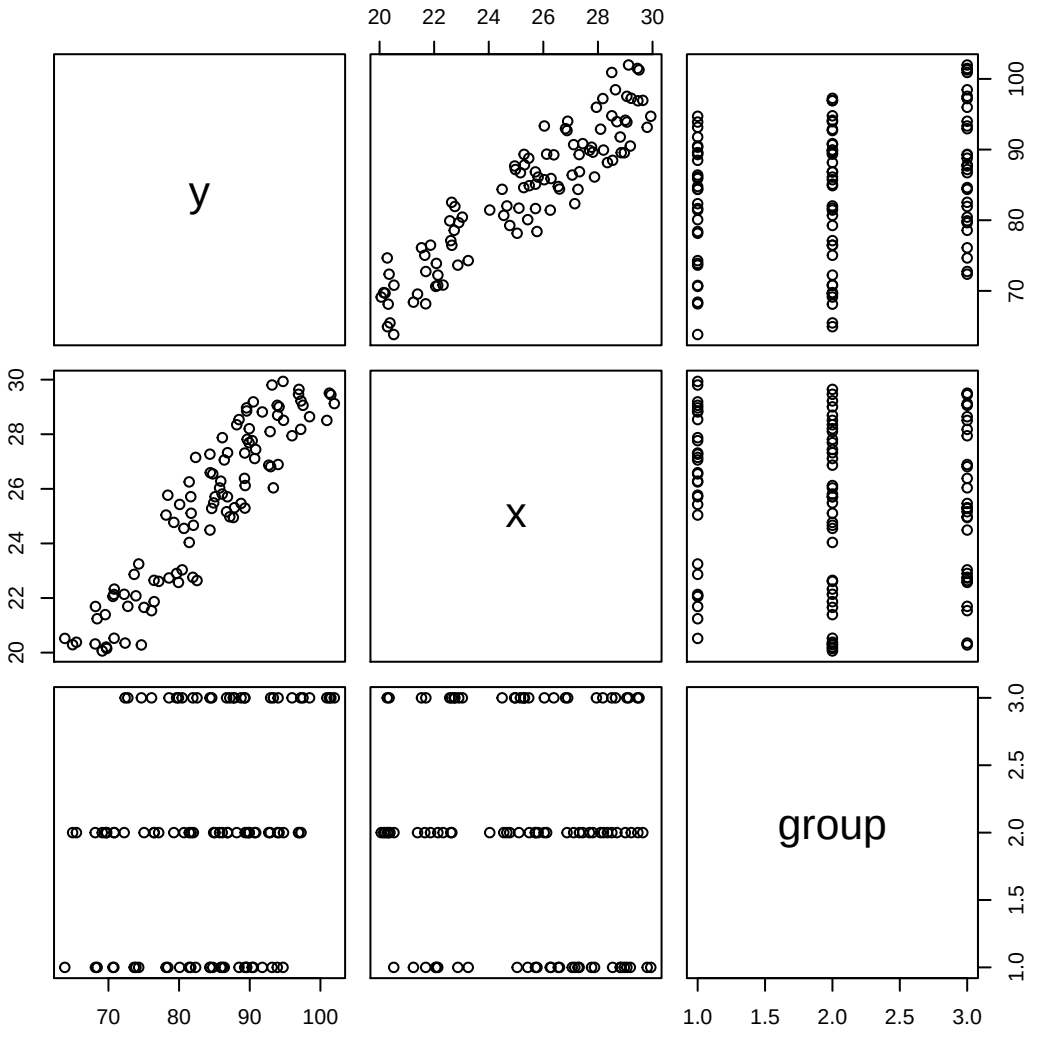


Figure 2: Plot 1

Here categorical data is plotted using the `plot()` function.

```
plot(y ~ group) # Plot 2
```

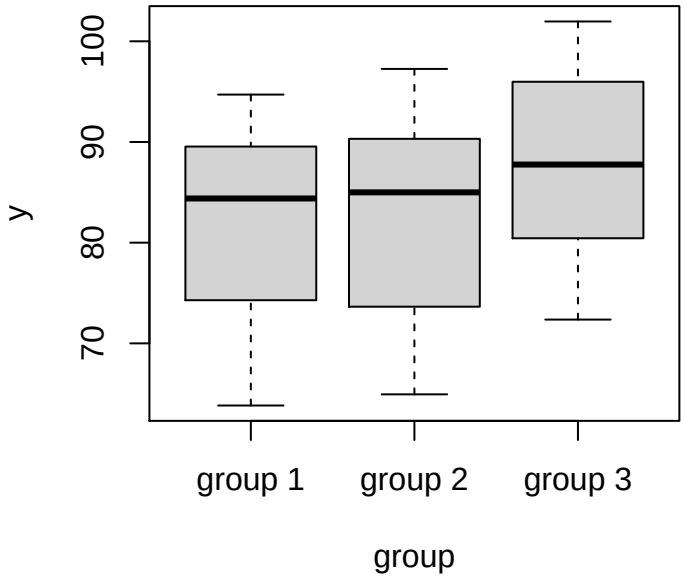


Figure 3: Plot 2

Fitting a Linear Model and Looking at it

lm is used to fit linear models.

```
lm.fit <- lm(y~x+group, data=data.ex)
```

Looking at the model matrix

`model.matrix` creates a design matrix from the description given.

```
head(model.matrix(lm.fit))
```

```
##   (Intercept)        x groupgroup 2 groupgroup 3
## 1           1 25.03999            0            0
## 2           1 27.76622            0            0
## 3           1 27.05785            0            0
## 4           1 29.80414            0            0
## 5           1 27.31158            0            0
## 6           1 22.13051            0            0
```

From the above model matrix we see that the model is...
$y = b0 + b1 * x + b2 * grp2 + b3 * grp3 + error$

Thus: $b0$ for grp1 effect, $b0 + b2$ for grp2 effect, $b0 + b3$ for grp3 effect
Or: $b2$ for balance between grp2 and grp1, $b3$ for balance between grp3 and grp1

Have a look at the fit

```
lm.fit
```

```
##
## Call:
## lm(formula = y ~ x + group, data = data.ex)
##
## Coefficients:
##  (Intercept)             x  groupgroup 2  groupgroup 3
##        2.791         3.054         3.625         8.046
```

Summarize the fit

`summary()` creates a summary of the fit.

```
summary(lm.fit)
```

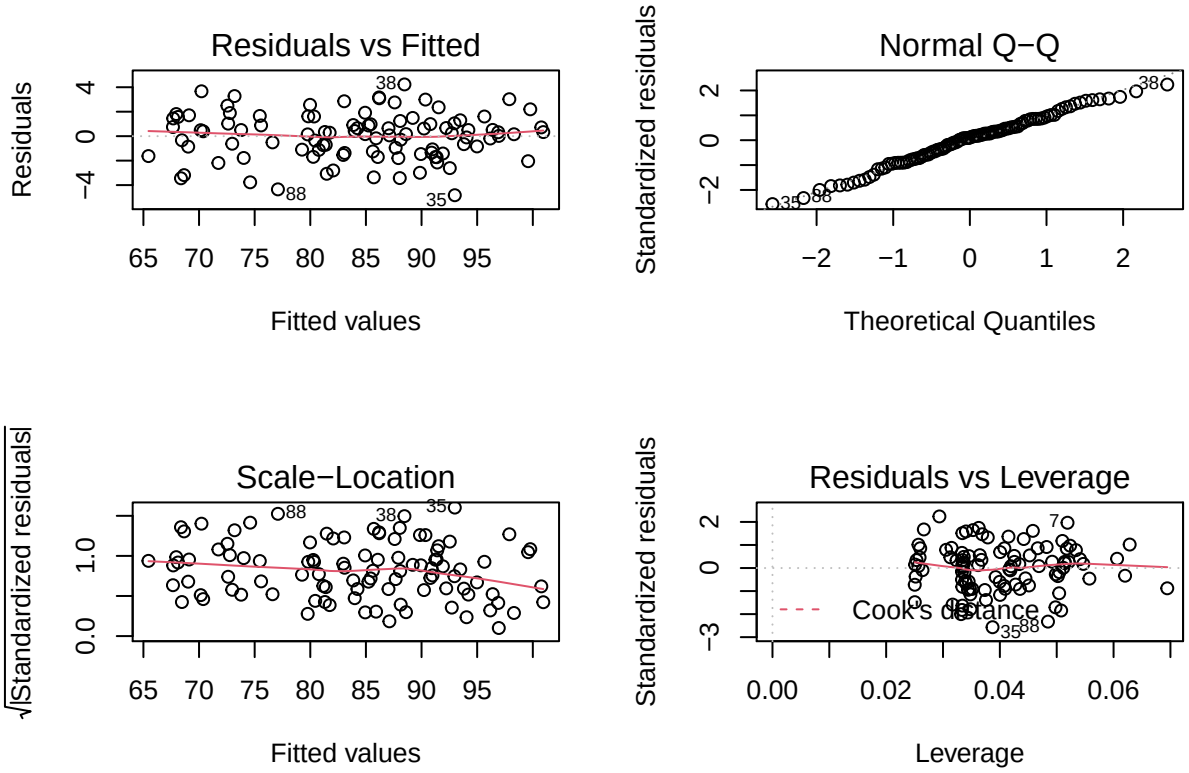
```
##
## Call:
## lm(formula = y ~ x + group, data = data.ex)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8279 -1.3531  0.2024  1.2548  4.2272
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.79084    1.75522   1.590    0.115
## x             3.05382    0.06606  46.225  < 2e-16 ***
## groupgroup 2  3.62497    0.46856   7.736 1.02e-11 ***
## groupgroup 3  8.04621    0.49762  16.169  < 2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.917 on 96 degrees of freedom
## Multiple R-squared:  0.9601, Adjusted R-squared:  0.9589
## F-statistic: 770.4 on 3 and 96 DF,  p-value: < 2.2e-16
```

Diagnostic plots

The `par()` function allows you to place multiple graphs in a single plot by setting some graphical parameters.

```
par(mfrow = c(2,2))
plot(lm.fit)
```



Save Data Output

Check working directory

The `getwd()` function returns an absolute filepath representing the current working directory of the R process.

```
getwd()
```

Manually set working directory

The `setwd()` function sets the current working directory to the path provided.

```
setwd("C:/Masters in Data Science")
```

Save the plot as a pdf

The `pdf()` function saves the plot to a pdf file.

```
pdf(file = "diagnostic_plots.pdf")
par(mfrow = c(2,2))
plot(lm.fit)
dev.off()
```

```
## pdf
##   2
```

Save the plot as a png

The `png()` function saves the plot to a png.

```
png(file = "diagnostic_plots.png",width=1000,height=1000)
par(mfrow = c(2,2))
plot(lm.fit)
dev.off()
```

```
## pdf
##   2
```

Embed the quality plot in the report

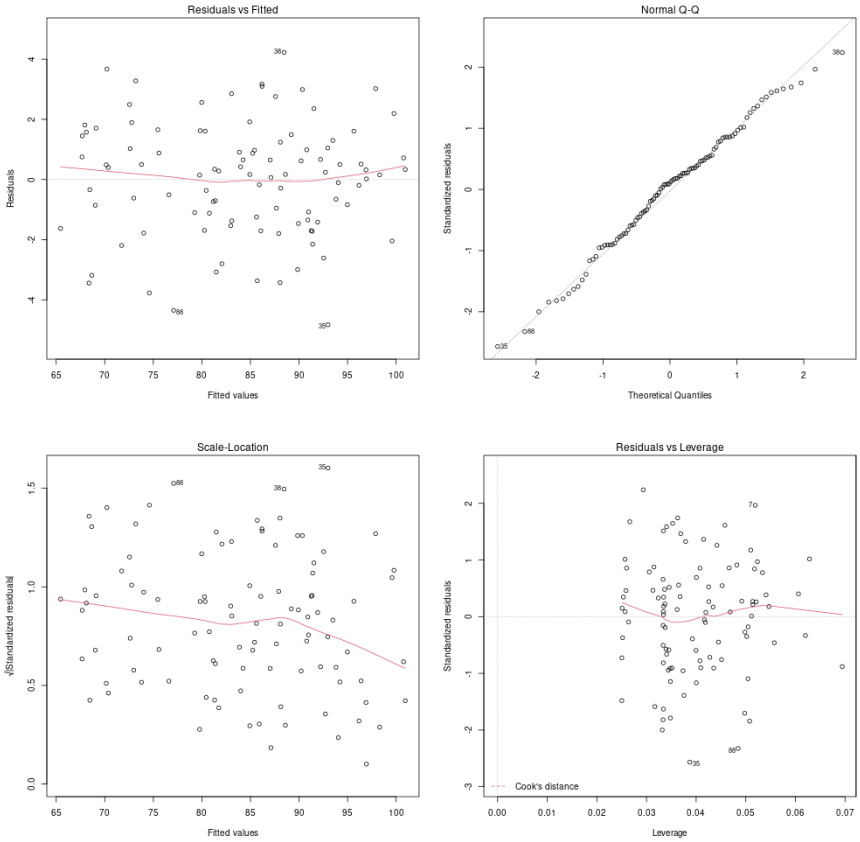


Figure 4: Quality Plot of Linear Model Fit

Most Relevant Commands in R

These are some of the most relevant commands in R.

`help()` - the help() function provides access to the documentation pages for R functions, data sets, and other objects.
`print()` - the print() function prints output in R.
`sum()` - the sum() returns the sum of all the values present in its arguments.
`plot()` - the plot() function plots the data.
`typeof()`- the typeof() can be used to assess the type of an object.
`class()` - the class() can be used to assess the class attribute of an object.
`names()` - the names() function is used to get or set the name of an Object.
`str()` - the str() function is used for compactly displaying the internal structure of a R object.
`head()` - the head() function returns the first n rows of the dataset.
`tail()` - the tail() function returns the last n rows of the dataset.

RMarkdown

RMarkdown contains 3 code chunks.

1) An optional YAML header

```
---
title: "Introductory Exercise - Markdown Documentation Assignment"
subtitle: "Exercise 1."
author: "Binu Cyriac"
date: "10/03/2021"
output:
  pdf_document:
    toc: true
    toc_depth: 3
    highlight: tango
---
```

2) R code chunks

```
```{r}
```
```

3) Text mixed with simple text formatting

```
*Tutorium in R*
```

Chunk output can be customized with knitr options and arguments set in the {} of a chunk header. Some of the arguments are....

- include : Whether to include anything from a code chunk in the output document.
- echo : Whether to echo the source code in the output document
- eval : Whether to evaluate a code chunk.
- results : When set to 'hide', text output will be hidden; when set to 'asis', text output is written.
- message : Whether to show messages.
- warning : Whether to show warnings.
- error : Whether to show errors.
- fig.cap : Adds a caption to graphical results.