

Experiment 4

Name	Ameya S. Daddikar
College I.D.	161070015
Course	Btech. Computer Engineering

Aim

To study the fundamentals and applications of R programming language in data science and data warehousing.

Theory

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

The R Environment

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes:

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hardcopy
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

The term “environment” is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software.

R, like S, is designed around a true computer language, and it allows users to add additional functionality by defining new functions. Much of the system is itself written in the R dialect of S, which makes it easy for users to follow the algorithmic choices made. For computationally-intensive tasks, C, C++ and Fortran code can be linked and called at run time. Advanced users can write C code to manipulate R objects directly.

Many users think of R as a statistics system. The creators of R prefer to think of it as an environment within which statistical techniques are implemented. R can be extended (easily) via packages. There are about eight packages supplied with the R distribution and many more are available through the CRAN family of Internet sites covering a very wide range of modern statistics.

R is very much a vehicle for newly developing methods of interactive data analysis. It has developed rapidly, and has been extended by a large collection of packages. However, most programs written in R are essentially ephemeral, written for a single piece of data analysis.

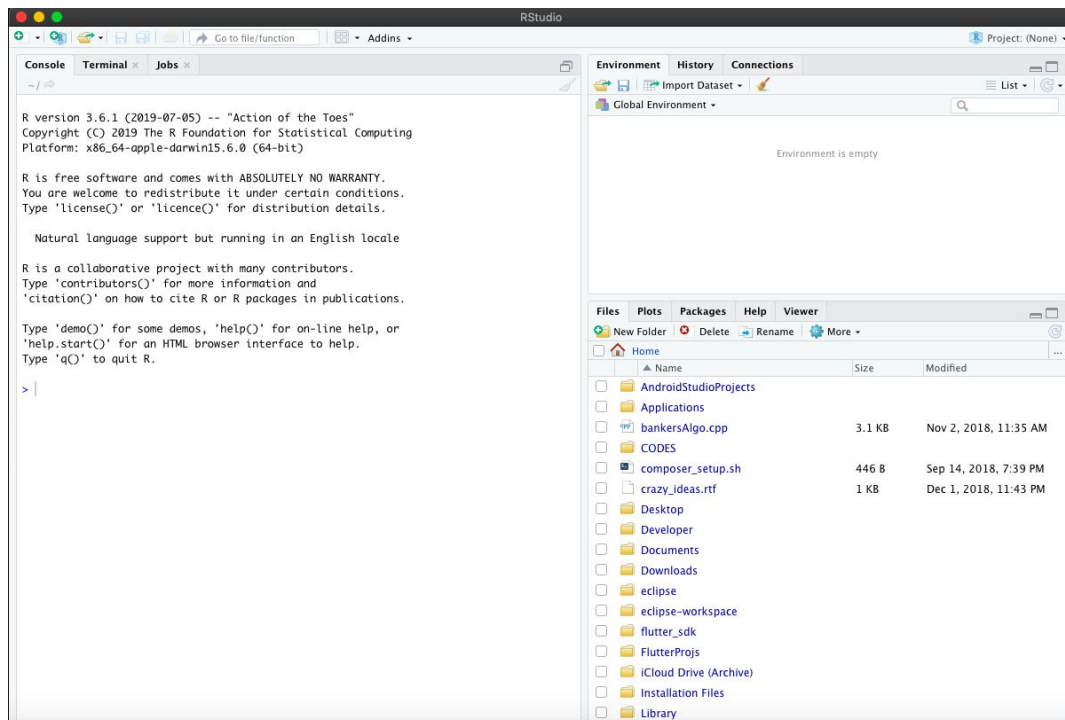
R has its own LaTeX-like documentation format, which is used to supply comprehensive documentation, both on-line in a number of formats and in hardcopy.

RStudio

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, Red Hat/CentOS, and SUSE Linux).

R can be operated using the CLI, Rstudio acts as an IDE that helps you maintain environments and projects.



Screenshot of RStudio Desktop GUI

R Language Fundamentals

Given below are few fundamental concepts obtained from the RStudio Cheat Sheet to get one up to speed with understanding R programs.

Types		
Converting between common data types in R. Can always go from a higher value in the table to a lower value.		
as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
as.numeric	1, 0, 1	Integers or floating point numbers.
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Variable Assignment	
<pre>> a <- 'apple' > a [1] 'apple'</pre>	

Conditions	a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
	a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null


Programming	
For Loop <pre>for (variable in sequence){ Do something }</pre> <p>Example</p> <pre>for (i in 1:4){ j <- i + 10 print(j) }</pre>	While Loop <pre>while (condition){ Do something }</pre> <p>Example</p> <pre>while (i < 5){ print(i) i <- i + 1 }</pre>
If Statements <pre>if (condition){ Do something } else { Do something different }</pre> <p>Example</p> <pre>if (i > 3){ print('Yes') } else { print('No') }</pre>	Functions <pre>function_name <- function(var){ Do something return(new_variable) }</pre> <p>Example</p> <pre>square <- function(x){ squared <- x*x return(squared) }</pre>


As we saw above, the language shares common traits with the interpreted language Python, with some syntax rules borrowed from other sources. Next, we see how R has a rich set of functionalities for handling complex multi-dimensional data types.


Vectors			Selecting Vector Elements	
Creating Vectors			By Position	
c(2, 4, 6)	2 4 6	Join elements into a vector	x[4]	The fourth element.
2:6	2 3 4 5 6	An integer sequence	x[-4]	All but the fourth.
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence	x[2:4]	Elements two to four.
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector	x[-(2:4)]	All elements except two to four.
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector	x[c(1, 5)]	Elements one and five.
Vector Functions			By Value	
sort(x)	Return x sorted.	rev(x)	x[x == 10]	Elements which are equal to 10.
table(x)	See counts of values.	unique(x)	x[x < 0]	All elements less than zero.
			x[x %in% c(1, 2, 5)]	Elements in the set 1, 2, 5.
			Named Vectors	
			x['apple']	Element with name 'apple'.


Matrixes

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

 `m[2,]` - Select a row

 `m[, 1]` - Select a column

 `m[2, 3]` - Select an element

 `t(m)`
Transpose
`m %*% n`
Matrix Multiplication
`solve(m, n)`
Find x in: $m \cdot x = n$


Data Frames


Also see the **dplyr** library.

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

 `df$x`


 `df[[2]]`


Understanding a data frame


`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

Matrix subsetting

 `df[, 2]`

 `df[2,]`


 `df[2, 2]`

`nrow(df)`
Number of rows.


`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

cbind - Bind columns.



rbind - Bind rows.



The main difference between list and vector is that list in R allows you to gather a variety of objects under one name (that is, the name of the list) in an ordered way. These objects can be matrices, vectors, data frames, even other lists, etc. It is not even required that these objects are related to each other in any way... while the elements in a vector all have the same data type.

Dataframes can be correlated with the DataFrame object used on Pandas library in Python. This data type is very easy to use for structured data like CSVs, TSVs, JSONs, etc. Below is the basic way to use/ store DataFrames.

Reading and Writing Data		
Input	Ouput	Description
<code>df <- read.table('file.txt')</code>	<code>write.table(df, 'file.txt')</code>	Read and write a delimited text file.
<code>df <- read.csv('file.csv')</code>	<code>write.csv(df, 'file.csv')</code>	Read and write a comma separated value file. This is a special case of read.table/ write.table.
<code>load('file.RData')</code>	<code>save(df, file = 'file.Rdata')</code>	Read and write an R data file, a file type special for R.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Poisson	rpois	dpois	ppois	qpois
Binomial	rbinom	dbinom	pbinom	qbinom
Uniform	runif	dunif	punif	qunif

Plotting

plot(x)
Values of x in
order.

plot(x, y)
Values of x
against y.

hist(x)
Histogram of
x.

Statistics

lm(x ~ y, data=df)
Linear model.

glm(x ~ y, data=df)
Generalised linear model.

summary
Get more detailed information
out a model.

t.test(x, y)
Perform a t-test for
difference between
means.

pairwise.t.test
Perform a t-test for
paired data.

prop.test
Test for a
difference
between
proportions.

aov
Analysis of
variance.

Also see the **ggplot2** library.

See the **lubridate** library.

As stated earlier, R's base library comes with a bunch of statistical analysis capabilities and graph libraries for data analysis and visualization.

Example Code in R - Linear Regression

For demonstration purposes, I have created an R notebook that runs a linear regression model to a dummy data.

Linear Regression Example in R

This is an R Markdown (<http://rmarkdown.rstudio.com>) Notebook. When you execute code within the notebook, the results appear beneath the code.

Below is a example Linear Regression code.

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The response vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)
```

The above code creates a regression relation of items in vector y w.r.t. items in vector x.

Below is the summary of the relation formed

```
print(summary(relation))
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -6.3002 -1.6629  0.0412  1.8944  3.9775
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -38.45509     8.04901  -4.778  0.00139 **
## x             0.67461     0.05191  12.997 1.16e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.253 on 8 degrees of freedom
## Multiple R-squared:  0.9548, Adjusted R-squared:  0.9491
## F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

The relation object can be used to predict values.

```
# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <- predict(relation,a)
print(result)
```

```
##           1
## 76.22869
```

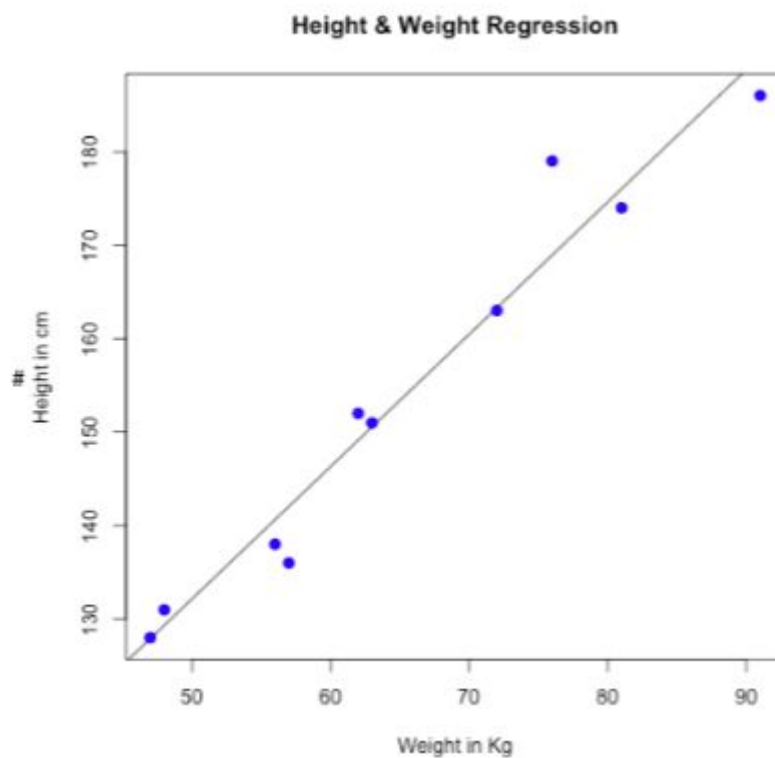
We can also visualize our model by plotting a graph.

```
# Give the chart file a name.
png(file = "linearregression.png")

# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x-y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.
dev.off()
```

```
## quartz_off_screen
##                2
```



Conclusion

Thus we studied the R programming language fundamentals and implemented those in a sample regression program.