SYLLABAUS

16CSU503B INTRODUCTION TO DATA SCIENCES 4H – 4C

Instruction Hours / week: L: 4 T: 0 P: 0 **Marks:** Int : **40** Ext : **60** Total: **100**

SCOPE

This course gives an introduction to the basics of data sciences and leave armed with practical experience extracting value from big data.

OBJECTIVES

- Building a comprehensive working knowledge and expertise around various analytical and database tools which is a key step to excel in Big Data and Data Science fields.
- The Data Science course covers topics in a comprehensive manner with applications of R programming.

UNIT- I

Data Scientist's Tool Box: Turning data into actionable knowledge, introduction to the tools that will be used in building data analysis software: version control, markdown, git, GitHub, R, and RStudio.

UNIT-II

R Programming Basics: Overview of R, R data types and objects, reading and writing data, Control structures, functions, scoping rules, dates and times, Loop functions, debugging tools, Simulation, code profiling

UNIT-III

Getting and Cleaning Data: Obtaining data from the web, from APIs, from databases and from colleagues in various formats. basics of data cleaning and making data ―tidy.

UNIT-IV

Exploratory Data Analysis: Essential exploratory techniques for summarizing data, applied before formal modeling commences, eliminating or sharpening potential hypotheses about the world that can be addressed by the data, common multivariate statistical techniques used to visualize high-dimensional data.

UNIT-V

Reproducible Research: Concepts and tools behind reporting modern data analyses in a reproducible manner, To write a document using R markdown, integrate live R code into a

literate statistical program, compile R markdown documents using knitr and related tools, and organize a data analysis so that it is reproducible and aCSUessible to others.

Suggested Readings

1. Rachel Schutt., & Cathy O'Neil.(2013). Doing Data Science: Straight Talk from the Frontiline. Schroff/O'Reilly.
2. Foster Provost., & Tom Fawcett.(2013). Data Science for Business What You Need to Know About Data Mining and Data-Analytic Thinking. O'Reilly.
3. John, W. Foreman. (2013). Data Smart: Using data Science to Transform Information into Insight. John Wiley & Sons.
4. Ian Ayres. (2007). Super Crunchers: Why Thinking-by-Numbers Is the New Way to Be Smart (1st ed.). Bantam.
5. Eric Seigel. (2013). Predictive Analytics: The Power to Predict who Will Click, Buy Lie, or Die (1st ed.). Wiley.
6. Matthew, A. Russel. (2013). Mining the Social Web: Data mining Facebook, Twitter, Linkedln, Goole+, GitHub, and More (2nd ed.). O'Reilly Media.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2016 onwards)
DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT : **INTRODUCTION TO DATA SCIENCES**
SEMESTER : **V** **L T P C**
SUBJECT CODE: **16CSU503B** **CLASS** : **III B.Sc.CS A & B** **4 0 0 4**

LECTURE PLAN

STAFF NAME: S.A. SATHYA PRABHA

S.No	Lecture Duration (Hr)	Topics	Support Materials
		UNIT-I	
1.	1	**Data Scientist's Tool Box:** Turning data into actionable knowledge	W1
2.	1	[Cont..] Turning data into actionable knowledge	W1
3.	1	Introduction to the tools that will be used in building data analysis software: version control, markdown, git, GitHub, R, and RStudio	W2,W3
4.	1	[Cont..] Introduction to the tools that will be used in building data analysis software: version control, markdown, git, GitHub, R, and RStudio	W2,W3
5.	1	Recapitulation and discussion of Important questions	
Text Book	**T1→** R Programming for Data Science, 2014 - 2015 Roger D. Peng		
Websites	**W1 →** https://www.edureka.co/blog/data-science-tutorial/ **W2→**https://www.analyticsvidhya.com/blog/2016/02/complete-tutorial-learn-data-science-scratch/ **W3→**https://intellipaat.com/tutorial/data-science-tutorial/introduction-of-data-science/		

Total No of Hours Planned For Unit – I 05			
UNIT-II			
1.	1	**R Programming Basics:** Overview of R, R data types and objects	T1:4-9 W4
2.	1	**[Cont..] R Programming Basics:** Overview of R, R data types and objects	T1:4-9 W4
3.	1	reading and writing data, Control structures	T1:23-25, T1:62
4.	1	functions	T1:70-77
5.	1	scoping rules, dates and times	T1:79-86, T2:45-48
6.	1	Loop functions	T1:64-68
7.	1	debugging tools	T1:108-114
8.	1	Simulation, code profiling	T1: 123-129,T1:116-120
9.	1	Recapitulation and discussion of Important questions	
Text Book	**T1→** R Programming for Data Science, 2014 - 2015 Roger D. Peng		
Websites	**W4→** http://www.tutorialspoint.com/r/r_tutorial.pdf		
Total No of Hours Planned For Unit – II 09			
UNIT-III			
1.	1	Introduction: Getting and Cleaning data	
2.	1	Significance of clear data	W6
3.	1	Obtaining data from web	W6
4.	1	Obtaining data from API's	W7
5.	1	Obtaining data from databases	W8
6.	1	Obtaining data from colleagues	W9
7.	1	Obtaining data in various formats	W6
8.	1	Basics of data cleaning	W6
9.	1	Missing data prediction	W6
10.	1	The purpose of tidy data for data science	W10
11.	1	Strategies for making data tidy	W10

12.	1	Recapitulation and discussion of Important questions	
Text Book	**T1→** R Programming for Data Science, 2014 - 2015 Roger D. Peng		
Journal	**J1→ Discussion Paper:** An introduction to data cleaning with R, Edwin de Jonge Mark van der Loo (https://cran.rproject.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf)		
Websites	**W5→** https://ramnathv.github.io/pycon2014-r/explore/tidy.html **W6→** https://towardsdatascience.com/big-data-what-is-web-scraping-and-how-to-use-it-74e7... **W7→** https://www.earthdatascience.org/courses/...analytics/get-data...apis/API-data-access-r **W8→** https://segment.com/blog/choosing-a-database-for-analytics **W9→** https://www.analyticsvidhya.com/.../an-introduction-to-apis-application-programming. **W10→**https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf		
Total No of Hours Planned For Unit – III 12			
UNIT-IV			
1.	1	**Introduction:** Exploratory Data Analysis	W6,W7
2.	1	Essential techniques for summarizing data	W8
3.	1	Exploratory techniques before formal modeling	W9
4.	1	Exploratory techniques after formal modeling	W9
5.	1	Eliminating or sharpening potential hypotheses	W10
6.	1	Boxplot using R	T2
7.	1	Ggplot using R	T2
8.	1	Scatter plot using R	T2
9.	1	Recapitulation and discussion of Important questions	

Text Book Websites	**T2→** G.Sudhamathy and C.Jothi Venkateswaran, "R Programming An Approach to Data Analytics", MJP Publishers **W6 →** https://bookdown.org/rdpeng/exdata/exploratory-data-analysis-checklist.html#formulate-your-question **W7→** http://r4ds.had.co.nz/exploratory-data-analysis.html **W8→**ttps://pdfs.semanticscholar.org/.../1ddd9aca1f2f5e2d8ff683a55089b8d4b162.pdf **W9 →** assets.press.princeton.edu/chapters/s02_8709.pdf **W10→**https://pdfs.semanticscholar.org/.../1ddd9aca1f2f5e2d8ff683a55089b8d4b162.pdf		
Total No of Hours Planned For Unit – IV 09			
UNIT-V			
1.	1	Introduction	W11
2.	1	Reproducible research	W11
3.	1	Concepts and tools for reporting	W11
4.	1	Importance of mark down in data science	W11
5.	1	To write a document using R mark down	W11
6.	1	Integrate live R code into a statistical program	W11,W14
7.	1	Compiling R mark down documents	W12
8.	1	Compiling R mark down using knitr and tools	W12
9.	1	Knitr commands	W12,W13
10.	1	Recapitulation and discussion of Important questions	
11.	1	**Discussion of previous ESE Question papers**	
12.	1	**Discussion of previous ESE Question papers**	
13.	1	**Discussion of previous ESE Question papers**	
Websites	**W11→** https://web.stanford.edu/~vcs/papers/RoundtableDeclaration2010.pdf1 **W12→**https://moodle.epfl.ch/.../Reproducible_Research_in_Computational_Science-Science-... **W13→** https://www.jstatsoft.org/article/view/v061b02/v61b02.pdf **W14→** https://www.biostat.wisc.edu/~kbroman/.../repro_research_JSM2016_withnotes.pdf7		
Total No of Hours Planned For Unit – V 13			
Total No. of Hours Planned: 48			

Text Book:

T1→ R Programming for Data Science, 2014 - 2015 Roger D. Peng
T2→ G.Sudhamathy and C.Jothi Venkateswaran, "R Programming An Approach to Data Analytics", MJP Publishers

Journal:

J1→ Discussion Paper: An introduction to data cleaning with R, Edwin de Jonge Mark van der Loo (https://cran.rproject.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf)

WEBSITES

W1 → https://www.edureka.co/blog/data-science-tutorial/

W2→https://www.analyticsvidhya.com/blog/2016/02/complete-tutorial-learn-data-science-scratch/

W3→https://intellipaat.com/tutorial/data-science-tutorial/introduction-of-data-science/

W4→ http://www.tutorialspoint.com/r/r_tutorial.pdf

W5→ https://ramnathv.github.io/pycon2014-r/explore/tidy.html
W6→ https://towardsdatascience.com/big-data-what-is-web-scraping-and-how-to- use-it-74e7...

W7→ https://www.earthdatascience.org/courses/...analytics/get-data...apis/API-data-access-r

W8→ https://segment.com/blog/choosing-a-database-for-analytics

W9→ https://www.analyticsvidhya.com/.../an-introduction-to-apis-application-programming.

W10→https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf

W11→ https://web.stanford.edu/~vcs/papers/RoundtableDeclaration2010.pdf1

W12→https://moodle.epfl.ch/.../Reproducible_Research_in_Computational_Science-Science-...

W13→ https://www.jstatsoft.org/article/view/v061b02/v61b02.pdf

W14→ https://www.biostat.wisc.edu/~kbroman/.../repro_research_JSM2016_withnotes.pdf7

UNIT- I

Data Scientist's Tool Box: Turning data into actionable knowledge, introduction to the tools that will be used in building data analysis software: version control, markdown, git, GitHub, R, and RStudio.

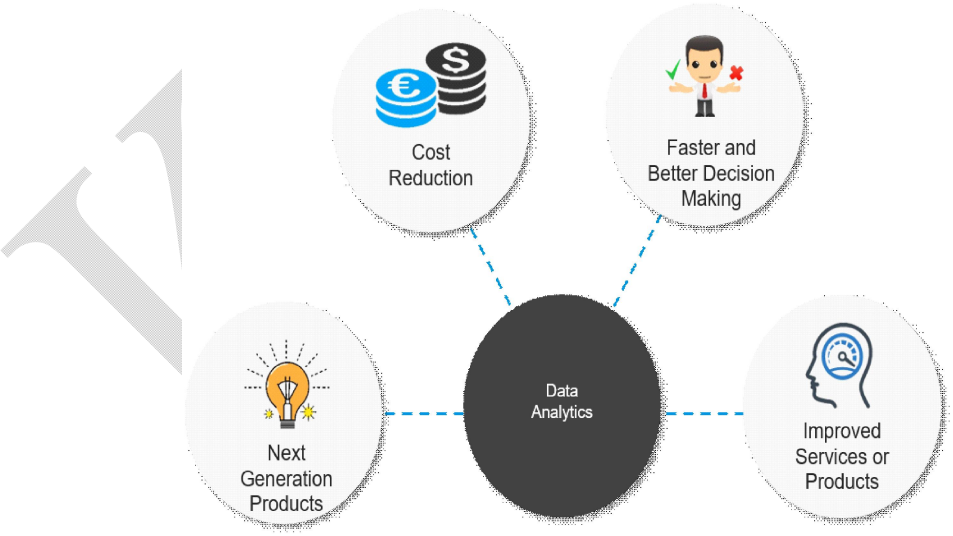
DATA SCIENTIST'S TOOL BOX

Programming is an integral part of data science. Among other things, it is acknowledged that a person who understands programming logic, loops and functions has a higher chance of becoming a successful data scientist.

What is Data science?

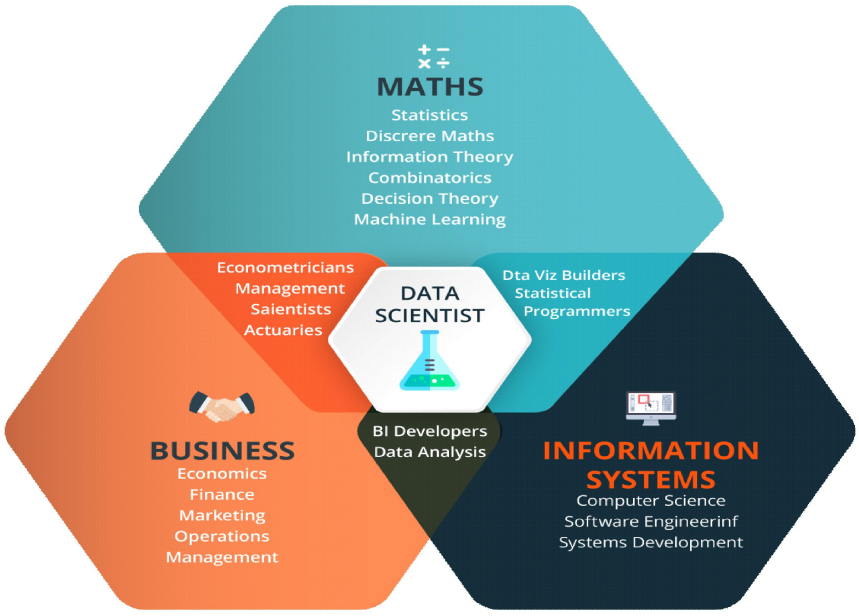
Definition 1: Data science is a technique to change the raw data into information. It is the study of where the valuable data comes from, what it represents and how it can be turned into a valuable resource in the creation of business and IT strategies.

Definition 2: Data Science, it is also known as data driven science, which makes use of scientific methods, processes and systems to extract knowledge or insights from data in various forms, i.e either structured or unstructured.



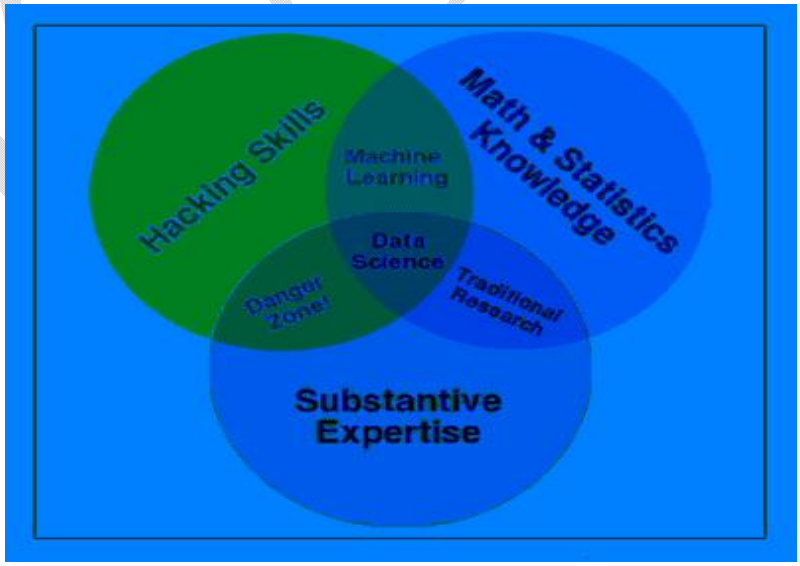
Moving forward, who does all this brain storming, or who practices Data Science? A **Data Scientist**.

Who is a Data Scientist?



Data Scientist = Programmer + Computer Scientist + Mathematician + Story teller + Domain Expert

It contains three components which are organizing, packaging and delivering data (OPD).



OPD Data Science Process

Step 1: Organize Data

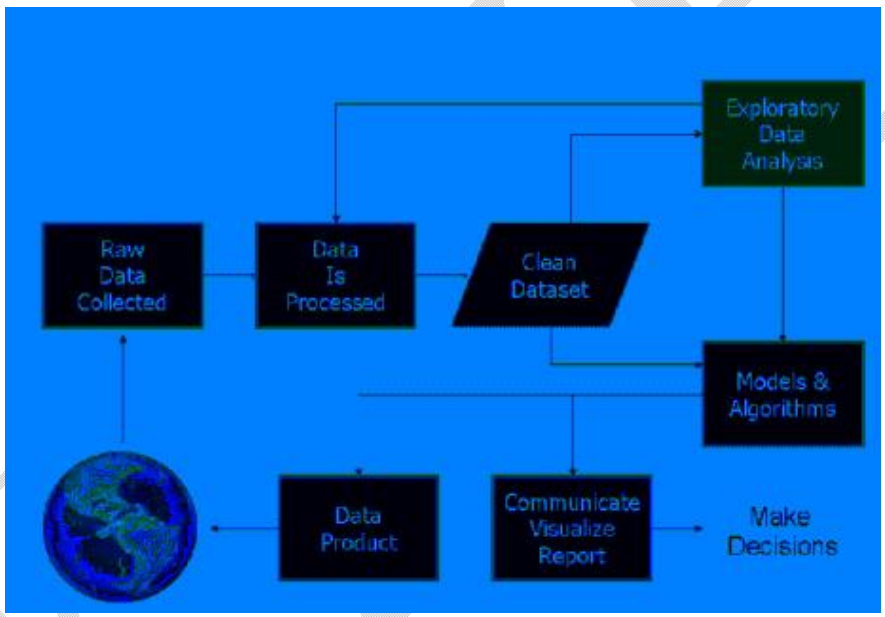
It includes the physical storage and formatting of data and integrated finest practices in data management.

Step 2: Package Data

In this the prototypes are created, the visualization is built and also statistics is performed. It includes logically joining and manipulating the raw data into a new representation and package.

Step 3: Deliver Data

In this process data is delivered to those who need that data.



Data Science vs. Data Analysis

It's very important to know that data science and Data analysis are little similar but, there so many differences between them. Let's check out the differences

Data Science	Data Analysis
Providing strategic actionable insights into the world	Providing operational observations into issues
Mathematical, technical and strategic knowledge are mandatory	Data analysis and visualization skills required
Deal with big data	Not necessarily deal with big data

In brief, with the help of data science, we will be able to analyze datasets, understand it and by performing necessary operations, we will get the resultant which will be beneficial for users.

Data Science Components

1. Datasets

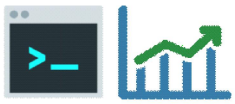
What will you analyze on? Data, right? You need a lot of data which can be analyzed; this data is fed to your algorithms or analytical tools. You get this data from various researches conducted in the past.

2. R Studio

R is an open source programming language and software environment for statistical computing and graphics that is supported by the R foundation. The R language is used in an IDE called R Studio.

Why is it used?

- **Programming and Statistical Language**



  - Apart from being used as a statistical language, it can also be used a programming language for analytical purposes.

- **Data Analysis and Visualization**

  - Apart from being one of the most dominant analytics tools, R also is one of the most popular tools used for data visualization.

- **Simple and Easy to Learn**

  - R is a simple and easy to learn, read & write

- **Free and Open Source**

  - R is an example of a FLOSS (Free/Libre and Open Source Software) which means one can freely distribute copies of this software, read it's source code, modify it, etc.

R Studio was sufficient for analysis, until our datasets became huge, also unstructured at the same time.

Introduction to the tools that will be used in building data analysis software:

Data Science: Productivity Tools

Data science projects involve keeping track of many data files and analysis scripts. Learn GitHub, git, Unix/Linux and RStudio to keep your projects organized and produce reproducible reports.

1. Version control

VCS are sometimes known as **SCM** (Source Code Management) tools or **RCS** (**Revision Control System**). Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Types:

A) Local Version Control Systems

B) Centralized Version Control Systems

C) Distributed Version Control Systems

Local Version Control Systems

Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

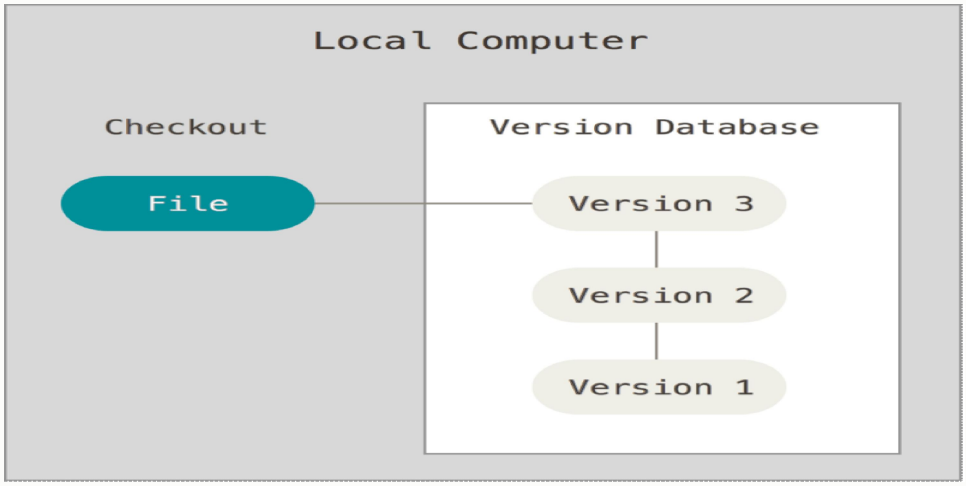


Figure 1. Local version control.

One of the more popular VCS tools was a system called RCS, which is still distributed with many computers today. RCS works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

Centralized Version Control Systems

The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems, such as CVS, Subversion, and Perforce, have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.

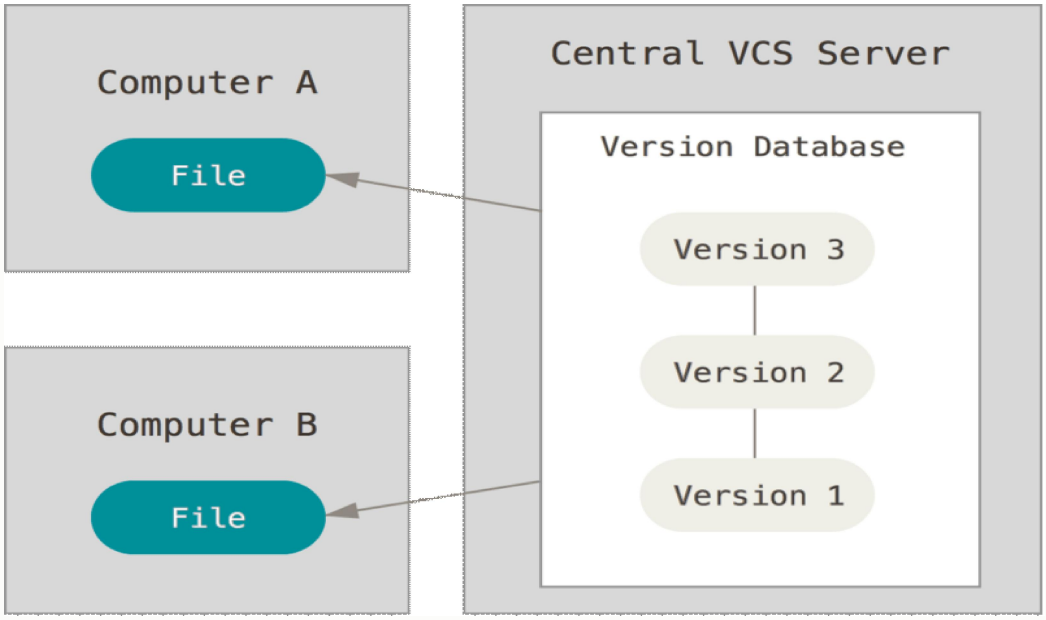


Figure 2. Centralized version control.

This setup offers many advantages, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what, and it's far easier to administer a CVCS than it is to deal with local databases on every client.

However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on. If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything—the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCS systems suffer from this same problem—whenever you have the entire history of the project in a single place, you risk losing everything.

Distributed Version Control Systems

This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

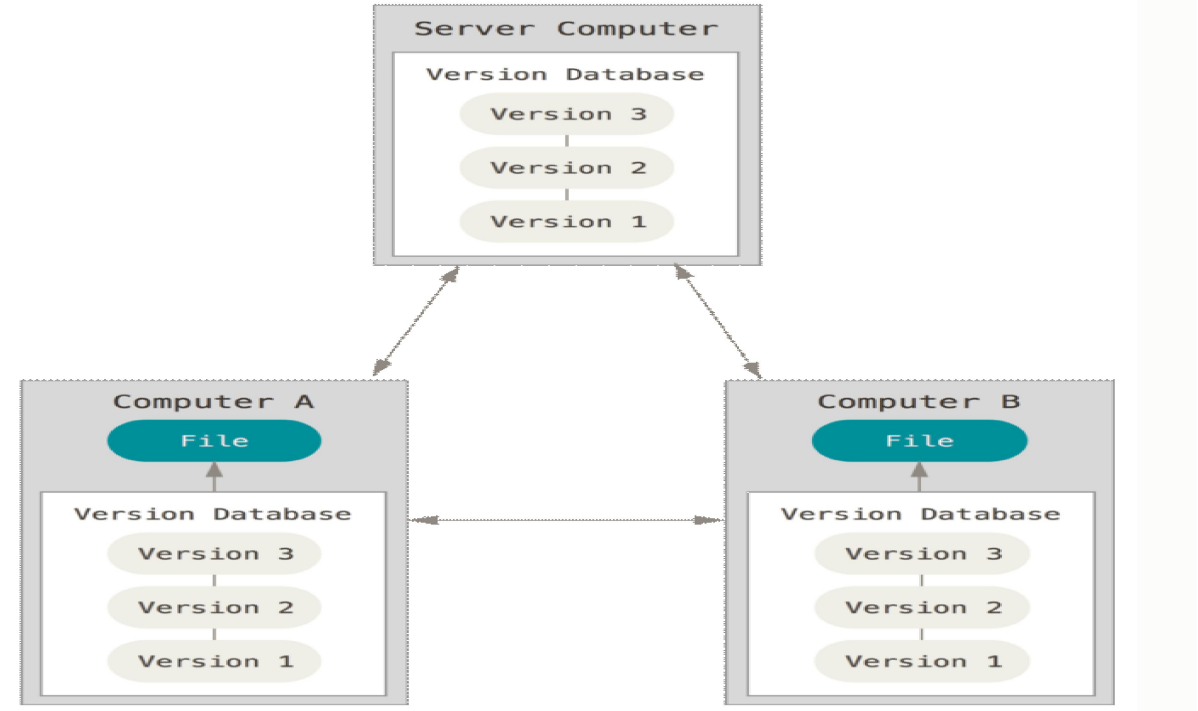


Figure 3. Distributed version control.

Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren't possible in centralized systems, such as hierarchical models.

2. Markdown

Markdown is a lightweight markup language with plain text formatting syntax. It is designed so that it can be converted to HTMLand many other formats using a tool by the same name.Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor.

3. git

Git is considered to be a newer and faster emerging star when it comes to version control systems. First developed by the creator of Linux kernel, Linus Torvalds, Git has begun to take the community for web development and system administration by storm, offering a largely different form of control. Here, there is no singular centralized code base that the code can be pulled from, and different branches are responsible for hosting different areas of the code. Other version control systems, such as VCS and SVN (Subversion control system), use a centralized control, so that only one master copy of software is used.
As a fast and efficient system, many system administrators and open-source projects use Git to power their repositories. However it is worth noting that Git is not as easy to learn as SVN or VCS is, which means that beginners may need to steer clear if they're not willing to invest time to learn the tool.

4. GitHub

GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

5. 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.

6. RStudio.

RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. RStudio was founded by JJ Allaire, creator of the programming language ColdFusion. Hadley Wickham is the Chief Scientist at RStudio.

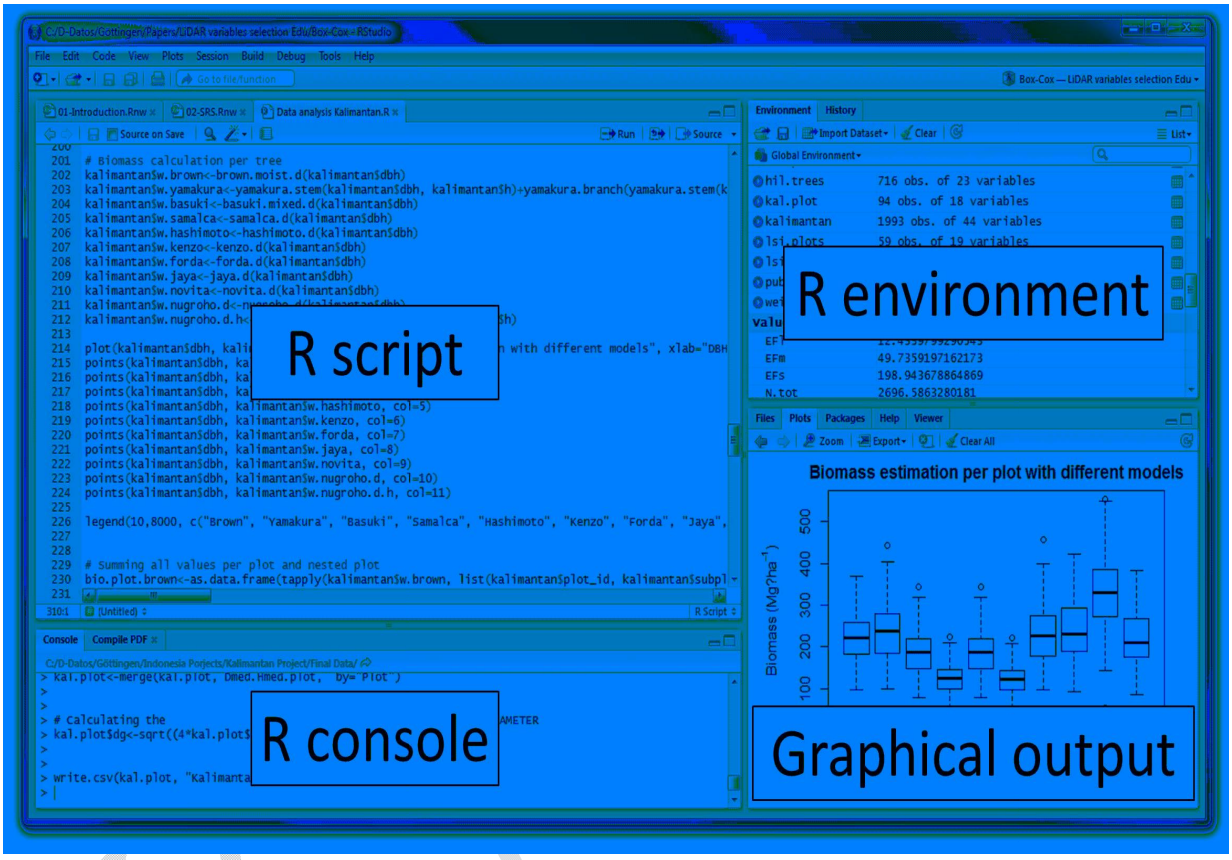
RStudio is available in two editions: RStudio Desktop, where the program is run locally as a regular desktop application; and RStudio Server, which allows accessing RStudio using a web browser while it is running on a remote Linux server. Prepackaged distributions of RStudio Desktop are available for Windows, macOS, and Linux.

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

RStudio is written in the C++ programming language and uses the Qt framework for its graphical user interface.

Work on RStudio started around December 2010,https://en.wikipedia.org/wiki/RStudio_-cite_note-9 and the first public beta version (v0.92) was officially announced in February 2011.Version 1.0 was released on 1 November 2016. Version 1.1 was released on 9 October 2017.

In April 2018 it was announced RStudio will providing operational and infrastructure support for Ursa Labs. Ursa Labs will focus on building a new data science runtime powered by Apache Arrow.



Let's quickly understand the interface of R Studio:

1. **R Console:** This area shows the output of code you run. Also, you can directly write codes in console. Code entered directly in R console cannot be traced later. This is where R script comes to use.
2. **R Script:** As the name suggest, here you get space to write codes. To run those codes, simply select the line(s) of code and press Ctrl + Enter. Alternatively, you can click on little 'Run' button location at top right corner of R Script.
3. **R environment:** This space displays the set of external elements added. This includes data set, variables, vectors, functions etc. To check if data has been loaded properly in R, always look at this area.
4. **Graphical Output:** This space display the graphs created during exploratory data analysis. Not just graphs, you could select packages, seek help with embedded R's official documentation.

POSSIBLE QUESTIONS

2 MARKS

1. What is Data science?
2. Who is a Data Scientist?
3. Define Data Science Process?
4. Compare Data Science vs. Data Analysis?
5. Name the Data Science Components?
6. Classify the Version control system?
4. Engrave GitHub?
5. Write down R?
6. Note down RStudio?

6 MARKS

1. Explain in detail about the tools that will be used in building data analysis software.
2. Write a brief description about data science.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

UNIT - I

S.no	Questions	Opt1	Opt2	Opt3	Opt4	Answer
1	_____ programming language is a dialect of S.	B	C	R	K	R
2	In 2004, _____ purchased the S language from Lucent for $2 million	Insightful	Amazon	IBM	Google	Insightful
3	In 1991, R was created by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of _____	John Hopkins	California	Harvard	Auckland	Auckland
4	Finally, in _____ R version 1.0.0 was released to the public.	2000	2005	2010	2012	2000
5	R is technically much closer to the Scheme language than it is to the original _____ language.	B	C	C++	S	S
6	The R-help and _____ mailing lists have been highly active for over a decade now	R-mail	R-devel	R-dev	Rcell	R-devel
7	Which of the following describes R language ?	Free	Paid	Available for free trial only	Trail	Free
8	The copyright for the primary source code for R is held by the _____ Foundation.	A	S	C++	R	R
9	They primary R system is available from the _____	CRAN	CRWO	GNU	RAN	CRAN
10	R functionality is divided into a number of _____	Packages	Functions	Domains	Library	Packages
11	The _____ R system contains, among other things, the base package which is required to run R and	root	child	base	private	base
12	Which of the following is a base package for R language ?	util	lang	tools	stats	tools
13	Which of the following is “Recommended” package in R ?	util	lang	stats	spatial	spatial
14	How many packages exist in R language for statistics ?	2000	3000	4000	5000	4000

15	Advanced users can write ___ code to manipulate R objects directly.	C	C++	Java	PHP	C
16	Which of the following is used for Statistical analysis in R language ?	RStudio	Studio	Heck	Rstat	RStudio
17	R has how many atomic classes of objects ?	1	3	5	2	5
18	Numbers in R are generally treated as _____ precision real numbers.	single	double	real	integer	double
19	If you explicitly want an integer, you need to specify the _____ suffix.	D	R	L	T	L
20	R objects can have attributes, which are like _____ for the object.	metadata	features	expression	data	metadata
21	What would be the result of following code ? > x<- 2 class(a)	"integer"	"numeric"	"logical"	"real"	"numeric"
22	Which of the following statement would print “0” “1” “2” “3” “4” “5” “6” for the following code ?	as.character(x)	as.logical(x)	as.numeric(x)	as.integer(x)	as.character(x)
23	Point out the wrong statement :	The grammar of the language determines whether an expression is complete or not	The <- symbol is the assignment operator in R	The ## character indicates a comment	The = symbol is also the assignment operator in R	The ## character indicates a comment
24	Files containing R scripts ends with extension :	.S	.R	.Rp	.RR	.R
25	Point out the wrong statement :	: operator is used to create integer sequences	The numbers in the square brackets are part of the vector itself	The numbers in the paranthesis are part of the vector itself	There is a difference between the actual R object and the manner in which that R object is printed to the console	The numbers in the square brackets are part of the vector itself
26	The entities that R creates and manipulates are known as _____	objects	task	container	function	objects

27	Which of the following can be used to display the names of (most of) the objects which are currently stored within R ?	object()	objects()	list()	data.frame()	objects()
28	Collection of objects currently stored in R is called as :	package	workspace	list	objects	workspace
29	R objects can have attributes, which are like _____ for the object	data	metadata	list	package	metadata
30	Matrices can be created by column-binding or row-binding with the _____ and _____ functions.	rowbind() and columnbind()	r_bind() and c_bind()	rbind() and cbind()	rowbind() and colbind()	rbind() and cbind()
31	_____ are a special type of vector that can contain elements of different classes	factors	matrices	data frames	list	list
32	_____ are used to represent categorical data and can be unordered or ordered	factors	matrices	data frames	list	factors
33	_____ is used to test objects if they are NA	is.nan()	is.na()	na()	as.na()	is.na()
34	_____ is used to test objects if they are NAN	is.nan()	is.na()	na()	as.na()	is.nan()
35	R objects can have _____ , which is very useful for writing readable code and self-describing objects.	list	matrices	attributes	names	names
36	Column names and row names can be set separately using the _____ and _____ functions.	colnames() and rownames()	cnames() and rnames()	col_names() and row_names(	columnnames() and rownames()	colnames() and rownames()
37	A _____ can only contain objects of the same class.	list	vector	data frames	factor	vector
38	Point out the wrong statement :	Key feature of R was that its syntax is very similar to S	R runs only on Windows computing platform and operating system	R has been reported to be running on modern tablets, phones, PDAs, and		R runs only on Windows computing platform and operating system
39	Who developed S?	Dennis Ritchie	Bjarne Stroustrup	James Gosling	John Chambers	John Chambers
40	R is an Interpreted Language so it can access through _____	Disk Operating System	User Interface Operating System	Operating System	Command Line Interpreter	Command Line Interpreter
41	R supports _____ arithmetic	logical	basic	matrix	vector	matrix

42	The sequence and number of observations in the vectors must be the same for each vector in the Data Frame to represent a _____	Record	Data object	Data	Data Sets	Data Sets
43	Matrices must have every element be the ____ class	same	different	literal	unique	same
44	Data frames can be converted to a matrix by calling _____	data.frame()	data()	data.matrix()	frame()	data.matrix()
45	Matrices are vectors with a _____ attribute	type	nrow	dimension	ncol	dimension
46	The <- Symbol is the _____ operator	Comparison Operator	Assignment Operator	Logical Operator	Boolean Operator	Assignment Operator
47	_____ can store different classes of objects in each column	data frames	matrices	lists	factors	data frames
48	Factor objects can be created with the _____ function.	data()	factors()	fact()	factor()	factor()
49	Missing values are denoted by __ or ___ for q undefined mathematical operations.	NA or NaN	NA or AS	Naan or No	N or Naa	NA or NaN
50	Objects can be explicitly coerced from one class to another using the _____ functions	.(datatype)	as.*	.(datatype)as	as()	as.*
51	R does not support _____ comments or comment blocks.	single line	*	multi line	//	multi line
52	Attributes of an object (if any) can be accessed using the _____ function	attrib()	att()	attr()	attributes()	attributes()
53	Numbers in R are generally treated as _____ objects	integer	real	numeric	number	numeric
54	>m <- matrix(nrow = 2, ncol = 3) >m > attributes(m)	2 3	3 2	dim	NA	dim
55	_____ function to find the data type of the variable	datatype()	class()	type()	cls()	class()
56	The _____ Function get the current working directory	get()	getwd()	getw()	wd()	getwd()
57	To change current working directory use _____ function	set()	setw()	swd()	setwd()	setwd()
58	A _____ is a vector object used to specify a discrete classification (grouping) of the components of other vectors of the same length	data frames	list	factor	vector	factor
59	_____ replicates the value	repl	rep	replicate	rep_c	rep

60	Which function is used to transpose data frame?	t()	ti()	transpose()	trans()	t()
61	Which of the following are courses in the Data Science Specialization?	Business Analytics	Python Programming	Machine Learning	R programming	R programming
62	Which of the following will initiate a git repository locally?	git merge	git pull	git init	git push	git init
63	Suppose you have forked a repository called datascientist on Github but it isn't on your local computer yet. Which of the following is the command to bring the directory to your local computer?	git pull	git clone	git init	git pull datascientist master	git clone
64	R is an interpreted language so it can access through_____?	disk operating system	user interface operating system	operating system	Command Line Interpreter	Command Line Interpreter
65	Descriptive analysis tell about_____?	past	present	future	any period	past
66	R was named partly after the first names of____R authors.?	one	two	three	four	two
67	Many quantitative analysts use R as their____tool?	leading tool	programming tool	both	none	both
68	Vectors come in two parts:_____ and _____.	atomic vectors and matrix	atomic vectos and array	atomic vectors and	atomic vectors and functions	atomic vectors and list
69	Which of the following is a base package for R language ?	util	lang	tools	math	tools
70	_____ is proprietary tool for predictive analytics.	R	SAS	SSAS	all the above mentioned	all the above mentioned
71	R functionality is divided into a number of _____.	packages	functions	domains	classes	packages
72	Which of the following is a base package for the R language?	utils	lang	tools	both a and b	tools
73	Which statement is correct about Dissimilarity?	Dissimilarity is a measure that ranges from 0 to INF [0, Infinity]	Dissimilarity is a measure that ranges from negative infinity to positive	Dissimilarity is a measure that ranges from -1 to 1.	All of the above statements are correct	Dissimilarity is a measure that ranges from 0 to INF [0, Infinity]
74	A collection of objects currently stored in R is called _____.	package	workspace	list	group	workspace

75	The R-help and _____ mailing lists have been highly active for over a decade now.	R-mail	R-devel	R-dev	R-devl	R-devel
76	Point out the correct statement:	Raw data is original source of data	Preprocessed data is original source of data	Raw data is the data obtained after	Reaw data is a clean data	Raw data is original source of data
77	Which of the following is performed by Data Scientist ?	Define the question	Create reproducible code	Challenge results	All of the Mentioned	All of the Mentioned
78	Which of the following is most important language for Data Science?	Java	Ruby	R	C/C++	R
79	Which of the following is one of the key data science skill?	Statistics	Machine Learning	Data Visualization	All of the Mentioned	All of the Mentioned
80	The entities that R creates and manipulates are known as _____	objects	task	container	data	objects

UNIT-II

R Programming Basics: Overview of R, R data types and objects, reading and writing data, Control structures, functions, scoping rules, dates and times, Loop functions, debugging tools, Simulation, code profiling

R Programming Basics:

R is a programming language and software environment used for statistical analysis, data modeling, and graphical representation and reporting. R is best tooling for software programmers, statisticians and data miners who looking forward for to easily manipulate and present data in compelling ways.

R was first created and developed by Ross Ihaka and Robert Gentleman in the University of Auckland New Zealand in 1993. And now "R Development Core Team" is developing it.

Overview of R:

R is an interpreted programming language (hence also called a scripting language), that means that your code does not have to be compiled before running it. This is a high-level language in which you do not have access to the inner workings of computer where you are running your code; everything is leaning toward helping you analyze data which is advantageous.

Uses of R

- Weather Service uses R to predict severe flooding.
- Social networking companies are using R to monitor their user experience.
- Newspapers companies are using R to create infographics and interactive data journalism applications.

R is adopted by the major companies because their data scientists prefer to use it.

Features of R

As described earlier, R programming language is versatile and can be used for software development environment for statistical analysis or for graphics representation and reporting purposes.

The below mentioned are the significant features of R language:

- R is simple and effective programming language which has been well-developed, as well as R is data analysis software.
- R is a well - designed, easy and effective language that has the concepts of conditionals, looping, user-defined recursive procedures and various I/O facilities.
- R has a large, consistent and incorporated set of tools used for data analysis.
- R contains suite of operators for different types of calculations on arrays, lists and vectors.
- R provides highly extensible graphical techniques.
- R graphical techniques for data analysis output either directly display to the computer, or can be print on paper.
- R has an effective data handling and storage facility.
- R is an online vibrant community.
- R is free, open source, powerful and highly extensible.

Evolution of R

In most of the time, you should be clear from the context that R is being referred to. R (which is the language) was developed in the early 1990's by Ross Ihaka and Robert Gentleman, when they both work at Department of Statistics at the University of Auckland, New Zealand. R made its first appearance in the year 1993. This programming language is based upon the S language which was developed in 1970s at Bell Laboratories, mainly by John Chambers. R (which is the software) is a GNU based project that reflects its status as important free along with open source software. Both the language along with the software is now developed by a group of 20 people approx. known as R Core Team.

R is the most widely used statistical programming language because of various reasons.

- R is free and an open source software project.
- R allows integrating with other languages, like C/C++, Java, and Python etc.
- R has an online vibrant growing community of users.
- The CRAN (The Comprehensive R Archive Network) package repository features have more than 8270 available packages.
- R is platform-independent, so you can use it on any operating system.

Basic Program

Once you have setup the environment for R, it is easy to start R command prompt by simply typing the command mentioned below at your command prompt:

$ R

This command will let you launch the R interpreter with a symbol like this '>' and you start writing the program using command prompt:

>newStr<- "Hello - World!"

> print (newStr)

Output:

[1] "Hello - World!"

R Data types:

There are several basic data types in R which are of frequent occurrence in coding R calculations and programs. Though seemingly in the clear, they can at a halt deliver surprises. Here you will try to understand all of the different forms of data type well by direct testing with the R code.

Here is the list of all the data types provided by R:

- Numeric
- Integer
- Complex
- Logical
- Character

Numeric Data type

Decimal values are referred as numeric data types in R. This is the default working out data type. If you assign a decimal value for any variable x like given below, x will become a numeric type.

> g = 62.4 # assign a decimal value to g

>g # print the variable's value - g

Integer Data type

If you want to create any integer variable in R, you have to invoke the as.integer() function to define any integer type data. You can be certain that y is definitely an integer by applying the is.integer() function.

```
> s = as.integer(3)

>s          # print the value of s
```

Fortunately, you can drive in a numeric value into an integer with this above mentioned as.integer() function like this:

```
>as.integer(3.14)    # drives in a numeric value
```

But it will work like type casting where the value of 3.14 gets changed to 3.

Complex Data type

A complex value for coding in R can be defined using the pure imaginary values 'i'.

```
> k = 1 + 2i     # creating a complex number

>k              # printing the value of k
```

The below mentioned example gives an error since −1 is not a complex value.

```
>sqrt(−1)       # square root of −1
```

And the error message will be something like this:

Warning message:

In sqrt(−1) : NaNs produced

Logical Data type

A logical value is mostly created when comparison between variables are done. Example will be like:

```
>   a = 4; b = 6              # sample values

>   g = a > b             # is a larger than b?

>   g              # print the logical value
```

Output:

[1] False

Character data type

A character object can be used for representing string values in R. You have to convert objects into character values using the as.character() function within your code like this:

```
> g = as.character(62.48)
>g            # prints the character string
```

Output:

[1] "3.14"

```
> class(s)      # print the class name of s
```

Output:

[1] "character"

R has a wide variety of data types including **scalars, vectors (numerical, character, logical), matrices, data frames, and lists.**

Vectors

```
a <- c(1,2,5.3,6,-2,4)                          # numeric vector
b <- c("one","two","three")                     # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)         #logical vector
```

Refer to elements of a vector using subscripts.

```
a[c(2,4)]                                       # 2nd and 4th elements of vector
```

Matrices

All columns in a matrix must have the same mode(numeric, character, etc.) and the same length. The general format is

```
mymatrix<- matrix(vector, nrow=r, ncol=c, byrow=FALSE,
  dimnames=list(char_vector_rownames, char_vector_colnames))
```

byrow=TRUE indicates that the matrix should be filled by rows. **byrow=FALSE** indicates that the matrix should be filled by columns (the default). **dimnames** provides optional labels for the columns and rows.

 # generates 5 x 4 numeric matrix

```
y<-matrix(1:20, nrow=5,ncol=4)
```

 # another example

```
cells <- c(1,26,24,68)
rnames<- c("R1", "R2")
cnames<- c("C1", "C2")
mymatrix<- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
dimnames=list(rnames, cnames))
```

Identify rows, columns or elements using subscripts.

```
x[,4]                    # 4th column of matrix
x[3,]                    # 3rd row of matrix
x[2:4,1:3]               # rows 2,3,4 of columns 1,2,3
```

Arrays

Arrays are similar to matrices but can have more than two dimensions. See **help(array)**for details.

Data Frames

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.). This is similar to SAS(Statistical Analysis System) and SPSS (Software package for statistical analysis)datasets.

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata<- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed")          # variable names
```

There are a variety of ways to identify the elements of a data frame.

```
myframe[3:5]                    # columns 3,4,5 of data frame
myframe[c("ID","Age")]          # columns ID and Age from data frame
myframe$X1                      # variable x1 in the data frame
```

Lists

An ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name.

 # example of a list with 4 components - a string, a numeric vector, a matrix, and a scaler

```
w <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)
```

 # example of a list containing two lists

```
v <- c(list1,list2)
```

Identify elements of a list using the [[]] convention.

```
mylist[[2]] # 2nd component of the list
mylist[["mynumbers"]] # component named mynumbers in list
```

Object in R

In R, all types of data are treated as objects. However, objects are not simply collections of data. They are particular instances (instantiations) of particular classes. Operations, or functions, are defined for specific classes. Let's try working on something such as a point pattern.

 # This time I will not show R outputs with codes. Just type or paste these lines into R and see what you get.

```
x <- rnorm(50, 10, 3)                  # creates 50 random x values from a normal distribution
y <- rnorm(50, 10, 4)                  # creates 50 random y values
mypoints<- as.data.frame(cbind(x,y))   # makes a data frame
class(mypoints)
mypoints
summary(mypoints)
plot(mypoints)                         # Gee, it looks like a point pattern...
box <- bbox(mypoints)      # Type in library(splancs) first. Bounding Box - did this work? Why not?
```

It seems that most functions above work well with this data frame but "bbox" does not. See help(bbox). It didn't work because "bbox" doesn't work on objects of class data.frame.

"bbox" operates on objects of class points (or a matrix of x and y values). Therefore you need to change the class accordingly. The following four approaches all work (try each one separately):

```
box <- bbox(cbind(x,y))
box <- bbox(as.matrix(mypoints))
box <- bbox(as.points(x,y))
box <- bbox(as.points(mypoints))
```

READING AND WRITING DATA TO AND FROM R

Functions for Reading Data into R:

There are a few very useful functions for reading data into R.

1. **read.table()** and **read.csv()** are two popular functions used for reading tabular data into R.
2. **readLines()** is used for reading lines from a text file.
3. **source()** is a very useful function for reading in R code files from a another R program.
4. **dget()** function is also used for reading in R code files.
5. **load()** function is used for reading in saved workspaces
6. **unserialize()** function is used for reading single R objects in binary format.

Functions for Writing Data to Files:

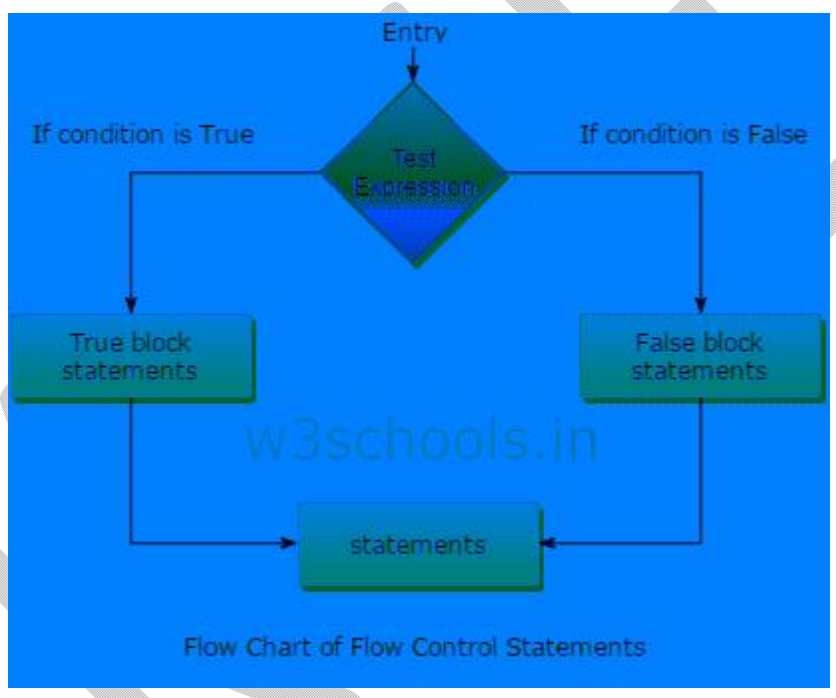
There are similar functions for writing data to files

1. **write.table()** is used for writing tabular data to text files (i.e. CSV).
2. **writeLines()** function is useful for writing character data line-by-line to a file or connection.
3. **dump()** is a function for dumping a textual representation of multiple R objects.
4. **dput()** function is used for outputting a textual representation of an R object.
5. **save()** is useful for saving an arbitrary number of R objects in binary format to a file.
6. **serialize()** is used for converting an R object into a binary format for outputting to a connection (or file).

Control Structures:

In R programming like that with other languages, there are several cases where you might wish for conditionally execute any code. Here 'if' and 'switch' functions of R language can be implemented if you already programmed condition based code in other languages, Vectorized conditional implementation via the ifelse() function is also a characteristics of R.

There are a lot of situations where you do not just want to execute one statement after another: in fact you have to control the flow of execution also. Usually this means that you merely want to execute some code if a condition is fulfilled. In that case control flow statements are implemented within R Program.



Here is the general structure of how control flow can be handled using the conditional statements within R programming.

Types of Flow Control Statements in R Programming

R programming provides three different types if statements that allows programmers to control their statements within source code. These are:

- if statement

- if….else statement
- switch statement

The 'if' Statement

The simplest form of decision controlling statement for conditional execution is the 'if' statement. The 'if' produces a logical value (more exactly, a logical vector having length one) and carries out the next statement only when that value becomes TRUE. In other words, an 'if' statement is having a Boolean expression followed by single or multiple statements.

```
if (TRUE) print ("One line executed")

## One line executed

if (FALSE) print ("Line not executed")

## Line not executed

if (NA) print ("Don't know whether true or not!")

## Error: missing value where TRUE/FALSE needed
```

The 'if….else' statements

In this type of statements the 'if' statement is usually followed by an optional 'else' statement that gets executes when the Boolean expression becomes false. This statement is used when you will be having multiple statements with multiple conditions to be executed.

Example:

```
if (TRUE)

{

print ("This will execute...")

} else

{

print ("but this will not.")

}

## This will execute...
```

The 'switch' Statement in R

A switch statement permits a variable to be tested in favor of equality against a list of case values. In the switch statement, for each case the variable which is being switched is checked. This statement is generally used for multiple selection of condition based statement.

The basic syntax for programming a switch based conditional statements in R is:
Syntax:

switch (test_expression, case1, case2, case3 .... caseN)

Here is a simple example of how to make use of switch statement in R:

Example:

```
gk<-  switch (
2,
"First",
"Second",
"Third",
"Fourth"
)
print (gk)
## [1] "Second"
```

R Functions:

All the variables that we use within a program needs to be stored somewhere. That somewhere can be called as an environment in R language. They are closely related to lists in which they are used to store diverse types of variables together. There may arise two situations where you might encounter environments. 1st, whenever a function is called, all the variables described by the function are stored in an environment belonging to that specific function (a function along with its environment is sometimes referred to as closure). 2nd, when you load a package, the functions in that package gets stored in an environment on the search path.

What is a function in R programming?

A function can be defined as a collection of statements structured together for carrying out a definite task. R provides a huge number of in built functions and also user can create their own functions.

In R, a function is treated as object so the R interpreter is capable of passing control to the function, along with arguments which may be essential to the function for achieving the actions. The function has the capability to turn its performance and returns control to the interpreter that may be stored in other objects.

The keyword 'function' is used to create a function in R.

The basic structure of a function can be:

function_name<- function (argu_1, argu_2, ...argu_N)

{

#Function body

}

Various Components of a Function:

The function in R is having various parts and each of them is having its own characteristics. These are:

- Function Name: is the real name of the function with which you can call it in some other part of the program. It is stored as an object with this name given to it.
- Arguments: is a placeholder for that specific function. As a function gets invoked, you can pass a value to the argument. Arguments are not mandatory to be used within the function; i.e. a function may not contain any arguments. Arguments can contain default values also.
- Function Body: It may contain a set of statements which specifies what the function does and how it will work along with its use.
- Return Value: Return value of any function is the last expression in the function which tells what that function is able to return.

Built in Functions:

Built in functions are those functions whose meaning and working is already defined within the function's body and they are kept somewhere within the packages or libraries of R language. These pre- defined functions make programmers task easier.

Some common examples of in built functions are:

seq(),max(), mean(), sum(x), paste(...) etc.

They are directly called and used by programmers who are writing programs.

Example:

```
print (seq (12,30))
```

This creates a sequence of number from 12 to 30 using the predefined function seq().

```
print (mean (4:26))
```

This calculates the mean of all the numbers ranging from 4 to 26

Here is a

```
hypotenuse<- function (x, y)
{
sqrt (x ^ 2 + y ^ 2)
}
```

You can now call this function as like this:

```
hypotenuse (3, 4)
## [1] 5
hypotenuse (y = 24, x = 7)
## [1] 25
```

Scoping Rules:

Variable Scope :

• A variable is pairing of a name and a value.

• Variables can be created by assignment.Theassigment

```
x = 10
```

Indicates that the value "10" and the name "x" are to be paired.

• R has both local and global variables.

• Global variables are created by assignments at top level.

• Local variables are created by assigning values within functions.

Example: Same Name, Different Variables

There are two "x" variables in the following code.

```
x = 10                  # <-- global
fun = function()
{ x = 20                # <-- local to fun
}
```

Inside "fun," the value of "x" is "20."

Outside "fun," the value of "x" is"10."

Changes made to one "x" do not affect the other.

Visibility of Global Variables:

Inside a function, the values of both local and global variables are visible.

```
> y = 20
>fun =
function()
{
x = 30
x + y
}
>fun()
```

```
[1] 50
```

Here, "fun" combines the value of a local variable, "x," and the value of a global variable, "y

Nested Functions

When R functions are nested, the variables of the outer functions are visible within the nested functions.

```
>outer =
function() {
inner =
function()
 {
 y = 20                    # y is local to inner
 x + y
}
x = 10                  # x is local to outer
inner()
}
>outer()
[1] 30
```

Scoping Rules:

• The scoping rules of a (computer) language govern how the value of variables can be determined.

• The scope of a variable is the region of code where that variable has meaning.

• In R, a local variable has meaning only within the function it is local to (including any functions that are nested within that function).

• When a value is sought for a given name, the local scope is inspected for a value, then any nesting scopes (i.e. functions) and finally a global value is sought. Function Parameters, Arguments and Variables

• Assignment is not the only way that variables are created.

• The association of function's arguments with the function's formal parameters also creates variables.

• In this case the variable name is the formal parameter name and the (initial) value of the variable is corresponding function argument.

Example: Parameters and Arguments

1. In the function definition

fun =

function(a, b)

a^2 + b^2

The formal parameters of the function are "a" and "b."

2. In the function call fun(10, 20) the arguments to the function are "10"and "20." The arguments and parameters are paired as variables.

A Simple Example

The following function adds a local variable "u" to a global variable "x."

> add.x.to = function(u) x + u

> x = 10

>add.x.to(20)

Output:

[1] 30

> x = 20

>add.x.to(20)

Output:

[1] 40

R Date and Time:

Dates and times are very frequently used concept in data analysis — not least for time - series study. The terrible fact is that with different numbers of days for each month, leap years, leap seconds along with time zones, they can be quite awful for dealing with programmatically.

The good news is that R provides a broad range of capabilities to deal with times and dates. While these ideas are quite fundamental to R programming, they have been left until now as some of the best ways of using them comes with in add-on packages.

Date and Time Classes:

There are three types of date and time classes which arrive with R programming. These are:

1. POSIXct

2. POSIXlt and

3. Date.

The sub categories are explained below:

POSIX Dates and Times Classes

POSIX dates and times are classic R: brightly thorough in their implementation, navigating all sorts of obscure technical issues, but with awful Unixy names that make everything seem more complicated than it really is.

The function Sys.time() is used to return the current date and time in POSIXct notation:

(now_ct<- Sys.time ())

[1] "2016-10-28 20:48:02 BST"

Here, ct is the short form for calendar time.

Again, when the date needs to be printed, you just see a formatted version of it, so it won't go obvious how the date is stored. By using 'unclass', you can see where it is indeed just a number:

unclass (now_ct)

[1] 1.374e+09

The Date Class

The third is the 'date' class in base R which is better named as the 'Date' class. It keeps dates as if the number of days since the starting of 1970. The 'Date' class is finely used in cases where programmers' do not bother about the time of day. Fractional days are probable which can get generated by computing a mean Date (suppose), but the POSIX classes are better for those situations like:

(now_date<- as. Date (now_ct))

[1] "2016-10-28"

class (now_date)

[1] "Date"

unclass(now_date)

[1] 15903

Other classes for date and time have add-on packages which include date, dates, chron, year mon, yearqtr, timeDate, ti, and jul.

Lubridate

If you have become sad with dates and you have considered to skip the using this class, do not worry about. 'Lubridate', as the name implies, put in some much needed lubrication to the practice of date manipulation. It does not include many new features over base R, but makes your code more readable and facilitates you to avoid thinking too much.

The real beauty is dissimilar elements in the same vector be able to have different formats (as long as the year is followed by the month that is followed by the day):

library (lubridate)

Attaching the package: 'lubridate' in your program

The following object gets masked from 'package:chron':

for - days, hours, minutes, seconds, years

karlos_rays_birth_date<- c(

"1994 - 08 - 28",

"1993/08\\28",

"Saturday + 1993.08*28"

)

ymd (karlos_rays _birth_date)

[1] "1993 - 08 - 28 UTC" "1993-08-28 UTC" "1993-08-28 UTC"

Loop functions:

Loops are used in programming to repeat a specific block of code.

- For loop
- While loop
- Break and next
- repeat loop

For loop:

A for loop is used to iterate over a vector in R programming.

Syntax of for loop

```
for (val in sequence)
{
statement
}
```

Here, sequence is a vector and val takes on each of its value during the loop. In each iteration, statement is evaluated.

Flowchart of for loop

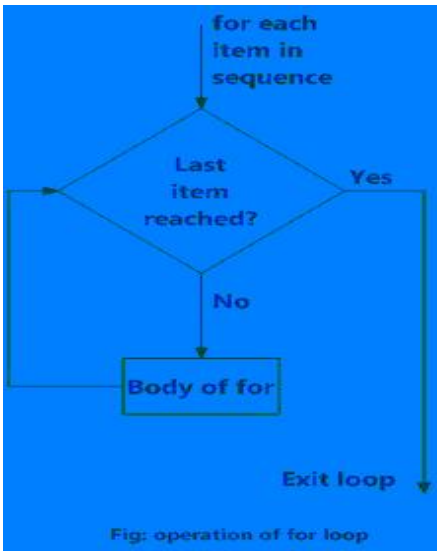


Fig: operation of for loop

Example: for loop

Below is an example to count the number of even numbers in a vector.

```
x <- c(2,5,3,9,8,11,6)
count<- 0
for (val in x) {
if(val %% 2 == 0)  count = count+1
}
print(count)
```

Output

[1] 3

In the above example, the loop iterates 7 times as the vector x has 7 elements.

In each iteration, val takes on the value of corresponding element of x.

We have used a counter to count the number of even numbers in x. We can see that x contains 3 even numbers.

While loop:

In R programming, while loops are used to loop until a specific condition is met.

Syntax of while loop

```
while (test_expression)
{
statement
}
```

Here, test_expression is evaluated and the body of the loop is entered if the result is TRUE.

The statements inside the loop are executed and the flow returns to evaluate the test_expressionagain.

This is repeated each time until test_expression evaluates to FALSE, in which case, the loop exits.

Flowchart of while Loop

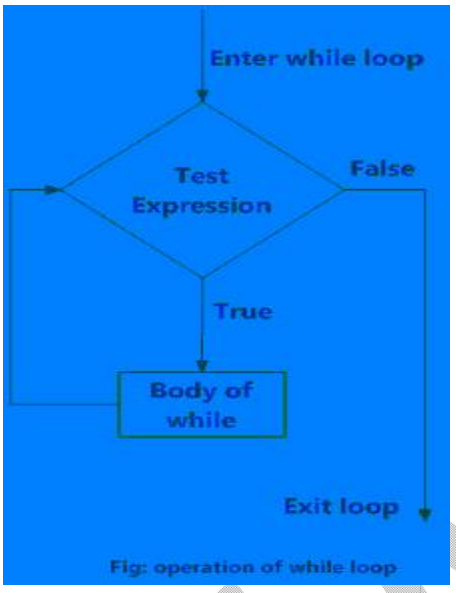


Fig: operation of while loop

Example of while Loop

```
i <- 1
while (i < 6) {
print(i)
i = i+1
}
```

Output

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

In the above example, i is initially initialized to 1.

Here, the test_expression is i < 6 which evaluates to TRUE since 1 is less than 6. So, the body of the loop isentered and i is printed and incremented.

Incrementing i is important as this will eventually meet the exit condition. Failing to do so will result into an infinite loop.

In the next iteration, the value of i is 2 and the loop continues.

This will continue until i take the value 6. The condition 6 < 6 will give FALSE and the while loop finally exits.

R break and next statements:

In R programming, a normal looping sequence can be altered using the break or the next statement.

break statement:

A break statement is used inside a loop (repeat, for, while) to stop the iterations and flow the control outside of the loop.

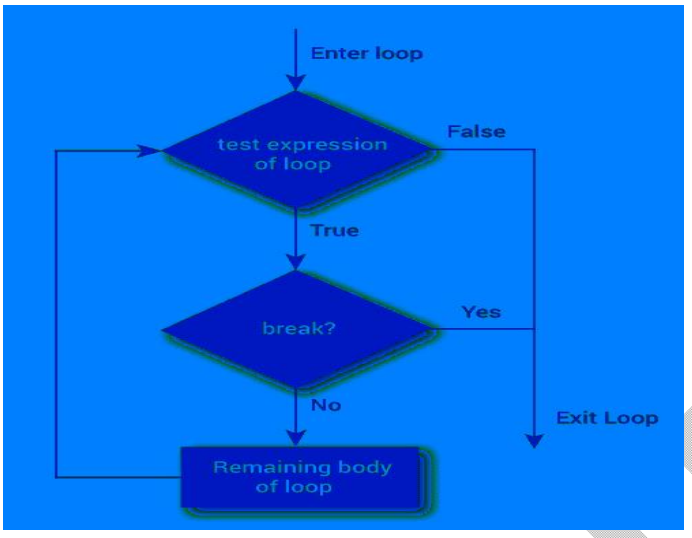
In a nested looping situation, where there is a loop inside another loop, this statement exits from the innermost loop that is being evaluated.

The syntax of break statement is:

```
if (test_expression) {
break
}
```

Note: the break statement can also be used inside the else branch of if...else statement.

Flowchart of break statement



Example 1: break statement

```
x <- 1:5
for (val in x) {
if (val == 3){
break
}
print(val)
}
```

Output

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

In this example, we iterate over the vector x, which has consecutive numbers from 1 to 5.

Inside the for loop we have used a if condition to break if the current value is equal to 3.

As we can see from the output, the loop terminates when it encounters the break statement.

Next statement

A next statement is useful when we want to skip the current iteration of a loop without terminating it. On encountering next, the R parser skips further evaluation and starts next iteration of the loop.

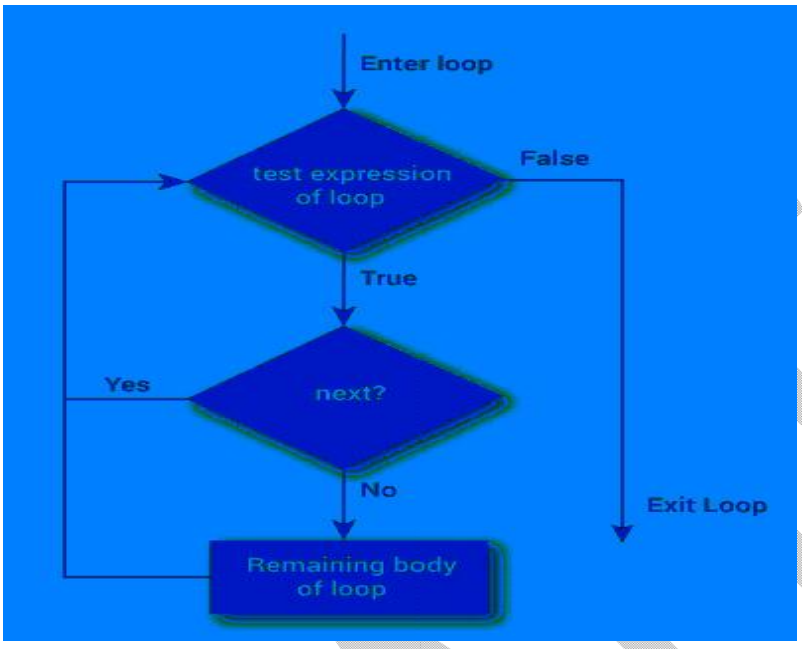
The syntax of next statement is:

```
if (test_condition) {
next
```

}

Note: the next statement can also be used inside the else branch of if...else statement.

Flowchart of next statement



Example 2: Next statement

```
x <- 1:5
for (val in x) {
if (val == 3){
next
}
print(val)
}
```

Output

```
[1] 1
[1] 2
[1] 4
[1] 5
```

In the above example, we use the next statement inside a condition to check if the value is equal to 3.

If the value is equal to 3, the current evaluation stops (value is not printed) but the loop continues with the next iteration.

The output reflects this situation.

Repeat loop:

A repeat loop is used to iterate over a block of code multiple numbers of times.

There is no condition check in repeat loop to exit the loop.

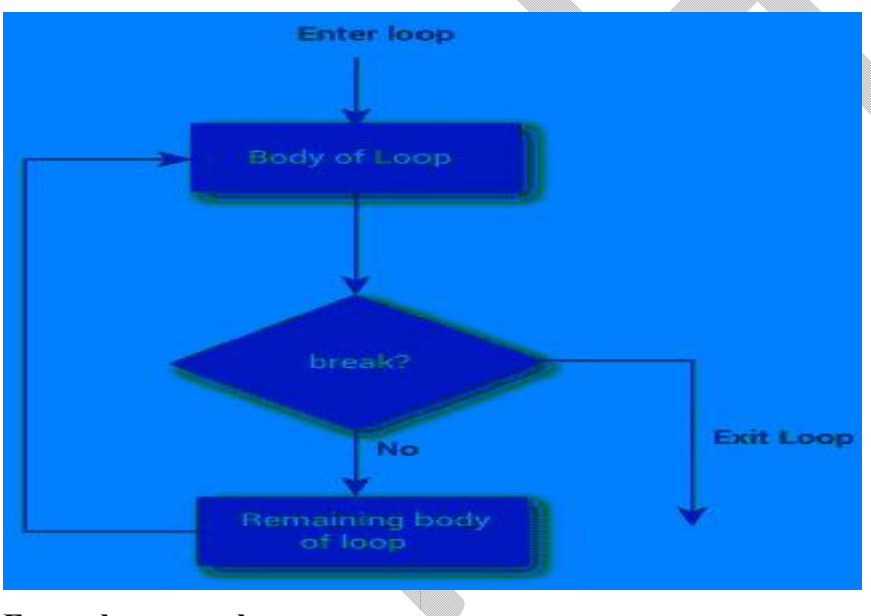
We must ourselves put a condition explicitly inside the body of the loop and use the break statement to exit the loop. Failing to do so will result into an infinite loop.

Syntax of repeat loop

```
repeat {
statement
}
```

In the statement block, we must use the break statement to exit the loop.

Flowchart of repeat loop



Example: repeat loop

```
x <- 1
repeat {
print(x)
x = x+1
if (x == 6){
```

break

}

}

Output

[1] 1

[1] 2

[1] 3

[1] 4

[1] 5

In the above example, we have used a condition to check and exit the loop when x takes the value of 6.

Hence, we see in our output that only values from 1 to 5 get printed.

Debugging tools:

What is R Debug?

A grammatically correct program may give us incorrect results due to logical errors. In case, if such errors (i.e. bugs) occur, we need to find out why and where they occur so that you can fix them. The procedure to identify and fix bugs is called "**debugging**".

There are number of R debug Functions, such as:

- traceback()
- debug()
- browser()
- trace()
- recover()

R Debug Functions

traceback()

If our code has already crashed and we want to know where the offending line is, try **traceback()**. This will (sometimes) show where about in the code the problem occurred.

When an **R function** fails, an error is printed to the screen. Immediately after the error, you can call traceback() to see in which function the error occurred. The traceback() function prints the list of functions that were called before the error occurred. The functions are printed in reverse order.

For Example:

```
f<-function(x) {
r<-x-g(x)
r
}
g<-function(y) {
r<-y*h(y)
r
}
h<-function(z) {
r<-log(z)
if(r<10)
r^2
else
r^3
}
>f(-1)
Error in if (r < 10) r^2 else r^3: missing value where TRUE/FALSE needed
In addition: Warning message:
In log(z) : NaNs produced
>traceback()
3: h(y)
2: g(x)
1: f (-1)
```

debug()

The function **debug() in R** allows one to step through the execution of a function, line by line. At any point, we can print out values of variables or produce a graph of the results within the function. While debugging, we can simply type "c" to continue to the end of the current section of code. traceback() does not tell us where in the function the error occurred. In order to know which line causes the error, we may want to step through the function using debug().

For Example:

Compute the sum of squared error SS = $\sum_{i=1}^{n}(x_i-\mu)$

```
## compute sum of squares
SS<-function(mu,x) {
d<-x-mu
d2<-d^2
ss<-sum(d2)
```

```
ss
}
# set seed to get reproducible results
>set.seed(100)
> x<-rnorm(100)
>SS(1,x)
[1] 202.5615
## now start debugging
>debug(SS)
>SS(1,x)
debugging in: SS(1, x)
debug: {
d <- x - mu
d2 <- d^2
ss<- sum(d2)
ss
}
attr(,"srcfile")
C:\Users\jihk\doc\courses\StatisticalComputing\lecture5\debugex2.R
Browse[1]>
```

After you see the "Browse[1]>" prompt, you can do different things:

- Typing n executes the current line and prints the next one;
- By typing Q, we can quit the debugging;
- Typing where tells where you are in the function call stack;
- By typing ls(), we can list all objects in the local environment;

Typing an object name or print(<object name>) tells us current value of the object. If your object has name n, c or Q, we have to use print() to see their values.

browser()

The R debug function '**browser()'** stops the execution of a function until the user allows it to continue. This is useful if we don't want to step through all the code, line-by-line, but we want it to stop at a certain point so we can check out what is going on. Inserting a call to the browser() in a function will pause the execution of a function at the point where the browser() is called. Similar to using debug() except we can control where execution gets paused.

For Example:

```
h<-function(z) {
```

```
browser() ## a break point inserted here
r<-log(z)
if(r<10)
r^2
else
r^3
}
>f(-1)
Called from: h(y)
Browse[1]>ls()
[1] "z"
Browse[1]> z
[1] -1
Browse[1]> n
debug: r <- log(z)
Browse[1]> n
debug: if (r < 10) r^2 else r^3
Browse[1]>ls()
[1] "r" "z"
<strong>Warning message:</strong>
In log(z) : NaNs produced
Browse[1]> r
[1] NaN
Browse[1]> c
Error in if (r < 10) r^2 else r^3 : missing value where TRUE/FALSE needed
>
```

trace()

Calling **trace()** on a function allows the user to insert bits of code into a function. The syntax for R debug function trace() is a bit strange for first time users. It might be better off using debug().

For Example:

```
>as.list(body(h))
[[1]]
`{`
[[2]]
r <- log(z)
[[3]]
if (r < 10) r^2 else r^3
attr(,"srcfile")
```

```
C:\Users\jihk\doc\courses\StatisticalComputing\lecture5\debugex1.R
>trace("h",quote(if(is.nan(r)) {browser()}), at=3, print=FALSE)
>f(1)
[1] 1
>f(-1)
Called from: eval(expr, envir, enclos)
Browse[1]>ls()
[1] "r" "z"
<strong>Warning message:</strong>
In log(z) : NaNs produced
Browse[1]> r
[1] NaN
Browse[1]> z
[1] -1
Browse[1]> c
...
>
>trace("h",quote(if(z<0) {z<-1}), at=2, print=FALSE)
[1] "h"
>f(-1)
[1] -1
```

recover()

When we are debugging a function, **recover()** allows us to check variables in upper level functions.

For Example:

```
>trace("h",quote(if(is.nan(r)) {recover()}), at=3, print=FALSE)
[1] "h"
>f(-1)
Enter a frame number, or 0 to exit
1: f(-1)
2: g(x)
3: h(y)
Selection: 1
Called from: eval(expr, envir, enclos)
Browse[1]>ls()
[1] "x"
Warning message:
In log(z) : NaNs produced
Browse[1]> x
```

```
[1] -1
Browse[1]> c
Enter a frame number, or 0 to exit
1: f(-1)
2: g(x)
3: h(y)
Selection: 2
Called from: eval(expr, envir, enclos)
Browse[1]>ls()
[1] "y"
Browse[1]> y
[1] -1
Browse[1]> c
Enter a frame number, or
Enter 0 to exit
1: f(-1)
2: g(x)
3: h(y)
Selection: 3
Called from: eval(expr, envir, enclos)
Browse[1]>ls()
[1] "r" "z"
Browse[1]> r
[1] NaN
Browse[1]> z
[1] -1
Browse[1]> c
Enter a frame number, or
Enter 0 to exit
1: f(-1)
2: g(x)
3: h(y)
Selection: 0
Error in if (r < 10) r^2 else r^3: missing value where TRUE/FALSE needed
```

- recover() can be used as an error handler, set using options() (e.g.options(error=recover)).
- When a function throws an error, execution is halted at the point of failure. We can browse the function calls and examine the environment to find the source of the problem.

Simulation

Simulations provide a powerful technique for answering a broad set of methodological and theoretical questions and provide a flexible framework to answer specific questions relevant to one's own research.

For example, simulations can evaluate the robustness of a statistical procedure under ideal and non-ideal conditions, and can identify strengths (e.g., accuracy of parameter estimates) and weaknesses (e.g., type-I and type-II error rates) of competing approaches for hypothesis testing.

Simulations can be used to estimate the statistical power of many models that cannot be estimated directly through power tables and other classical methods (e.g., mediation analyses, hierarchical linear models, structural equation models, etc.). The procedures used for simulation studies are also at the heart of bootstrapping methods, which use resampling procedures to obtain empirical estimates of sampling distributions, confidence intervals, and *p*-values when a parameter sampling distribution is non-normal or unknown.

Simulation studies typically are designed according to the following steps:

1. A set of assumptions about the nature and parameters of a dataset are specified.
2. A dataset is generated according to these assumptions.
3. Statistical analyses of interest are performed on this dataset, and the parameter estimates of interest from these analyses (e.g., model coefficient estimates, fit indices, *p*-values, etc.) are retained.
4. Steps 2 and 3 are repeated many times with many newly generated datasets (e.g., 1000 datatsets) in order to obtain an empirical distribution of parameter estimates.
5. Often, the assumptions specified in step 1 are modified and steps 2–4 are repeated for datasets generated according to new parameters or assumptions.
6. The obtained distributions of parameter estimates from these simulated datasets are analyzed to evaluate the question of interest.
7. Common R commands for simulation studies.

Commands for working with vectors

Command	Description	Examples
c	Combines arguments to make vectors	#create vector called a which contains the values 3, 5, 4

Commands for working with vectors

Command	Description	Examples
		a = c(3,5,4)
		#identical to above, uses <- instead of =
		a <- c(3,4,5)
		#return the second element in vector a, which is 5
		a[2]
		#remove the contents previously stored in vector a
		a = NULL

Commands for working with vectors

Command	Description	Examples
length	Returns the length of a vector	#return length of vector a, which is 3 a = c(3,5,4) length(a)
rbind and cbind	Combine arguments by rows or columns	#create matrix d that has vector a as row 1 and vector b as row 2. a = c(3,5,4) b = c(9,8,7) d = rbind(a,b) #create matrix e that has two copies

Commands for working with vectors

Command	Description	Examples
		of matrix d joined by column e = cbind(d,d)

Commands for generating random values

Command	Description	Examples
rnorm	Randomly samples values from normal distribution with a given population *M* and *SD*	#randomly sample 100 values from a normal distribution with a population *M* = 50 and *SD* = 10 x = rnorm(100, 50, 10)
sample	Randomly sample values from another vector	#randomly sample 8 values from vector a, with replacement

Commands for generating random values

Command	Description	Examples
		a = c(1,2,3,4,5,6,7,8)
		sample(a, size=8, replace=TRUE)
		#e.g., returns 3 1 3 6 5 4 2 2
set.seed	Allows exact replication of randomly-generated numbers between simulations	#The same 5 random numbers returned each time the following lines are run
		set.seed(12345)
		rnorm(5, 50, 10)

Command for statistical modeling

Command	Description	Examples

lm	fits linear ordinary least squares models	#Regress y onto x1 and x2
		y = c(2,2,5,4,3,6,4,6,5,7)
		x1 = c(1,2,3,1,1,2,3,1,2,2)
		x2 = c(0,0,0,0,0,0,1,1,1,1,1)
		mymodel = lm(y ~ x1 + x2)
		summary(mymodel)
		#retrieve fixed effect coefficients from a lm object
		mymodel$coefficients

Commands for programming

Command	Description	Examples

function	generate customized function	# function that returns the sum of x1 and x2
		myfunction = function(x1, x2){
		mysum = x1 + x2
		return(mysum)
		}

for	create a loop, allowing sequences of commands to be executed a specified number of times	#Create vector of empirical sample means (stored as mean_vector) from 100 random samples of size $N = 20$, sampled from a population $M = 50$ and $SD = 10$.
		mean_vector = NULL

Commands for programming

Command	Description	Examples

```
for (i in 1:100){

  x = rnorm(20, 50, 10)

  m = mean(x)

  mean_vector = c(mean_vector, m)

}
```

Code profiling

Profiling R code gives you the chance to identify bottlenecks and pieces of code that needs to be more efficiently implemented.

Profiling R code is usually the last thing I do in the process of package (or function) development. In my experience we can reduce the amount of time necessary to run an R routine by as much as 90% with very simple changes to our code. Just yesterday I reduced the time necessary to run one of my functions from 28 sec. to 2 sec. just by changing one line of the code from

```
x = data.frame(a = variable1, b = variable2)
```

to

```
x = c(variable1, variable2)
```

This big reduction happened because this line of code was called several times during the execution of the function.

Rprof and summaryRprof approach

The standard approach to profile R code is to use the Rprof function to profile and the summaryRprof function to summarize the result.

```
Rprof("path_to_hold_output")
## some code to be profiled
Rprof(NULL)
## some code NOT to be profiled
Rprof("path_to_hold_output", append=TRUE)
## some code to be profiled
Rprof(NULL)


# summarize the results
summaryRprof("path_to_hold_output")
```

Rprof works by recording at fixed intervals (by default every 20 msecs) which R function is being used, and recording the results in a file. summaryRprof will give you a list with four elements:

- by.self: time spent in function alone.
- by.total: time spent in function and callees.
- sample.interval: the sampling interval, by default every 20 msecs.
- sampling.time: total time of profiling run. Remember that profiling does impose a small performance penalty.

Profiling short runs can be misleading, so in this case I usually use the replicate function

```
1  # Evaluate shortFunction() for 100 times
2  replicate(n = 100, shortFunction())
```

R performs garbage collection from time to time to reclaim unused memory, and this takes an appreciable amount of time which profiling will charge to whichever function happens to provoke it. It may be useful to compare profiling code immediately after a call to gc() with a profiling run without a preceding call to gc.

Example

A short default example collected from the help files is

```
1   Rprof(tmp<- tempfile())
2   example(glm)
3   Rprof()
4   summaryRprof(tmp)
```

which returns the following output:

```
$by.self
self.time self.pct total.time total.pct
"print.default"     0.04    18.18      0.04     18.18
"glm.fit"           0.02     9.09      0.04     18.18
"all"               0.02     9.09      0.02      9.09
""        0.02     9.09      0.02      9.09
...

$by.total
total.time total.pct self.time self.pct
"example"                0.22    100.00      0.00     0.00
"source"                 0.20     90.91      0.00     0.00
"eval"                   0.12     54.55      0.00     0.00
"print"                  0.12     54.55      0.00     0.00
...

$sample.interval
[1] 0.02

$sampling.time
[1] 0.22
```

POSSIBLE QUESTIONS

2 MARKS

1. What is the use of R?
2. Define data type.
3. Mention the loops in R.
4. What is a function?
5. Define control structures.
6. Define Matrices with example.
7. What are data frame?
8. Define List with example.
9. Define object in R.
10. Name the functions used to read data from R.
11. Name the functions used to write data to R.
12. Name some built in functions with example.
13. What are the components of function?
14. Define control structures.
15. What is scope of variables?
16. Mention any two scoping rules in R.
17. What is lubridate?
18. Define Date class.
19. Define break and next statements in R.
20. What is debugging?

6 MARKS

1. Explain R data types with examples.
2. Write about control structures in R with examples.
3. Explain in detail about functions.
4. Enlighten date and time class with example.
5. Explain loop functions with example.
6. Explain the following:
    a) Debugging tools
    b) Simulation
    c) Code profiling

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

UNIT - II

sno	Questions	Opt1	Opt2	Opt3	Opt4	Answer
1	Which of the following is used for reading tabular data	read.csv	dget	readLines	get	read.csv
2	Which of the following is used for reading in saved workspaces ?	unserialize	load	get	read	load
3	Which of the following statement would read file “foo.txt”	data <- read.table(“foo.txt”)	read.data <- read.table(“foo.txt”)	data <- read.data(“foo.txt”)	data.read <- read(“foo.txt”)	data <- read.table(“foo.txt”)
4	Which of the following function is identical to read.table	read.csv	read.data	read.tab	read.table	read.csv
5	Which of the following code would read 100 rows	initial <- read.table(“datatable.txt”, *nrows = 100)*	tabAll <- read.table(“datatable.txt”, colClasses = classes)	initial <- read.table(“datatable.txt”, *nrows = 99)*	initial <- read.table(“datatable.txt”, ncols= 99)	initial <- read.table(“datatable.txt”, *nrows = 100)*
6	Which of the following code opens a connection to the file foo.txt, reads from it, and closes the connection when its done ?	data <- read.csvo(“foo.txt”)	data <- read.csv(“foo.txt”)	data <- readonly.csv(“foo.txt”)	data <- readcsv(“foo.txt”)	data <- read.csv(“foo.txt”)
7	Which of the following extracts first element from the following vector ? > x <- c("a", "b", "c", "c", "d", "a")	x[10].	x[1].	x[0].	x[11].	x[1].
8	Point out the correct statement :	There are three operators that can be used to extract subsets of R objects	The [operator is used to extract elements of a list or data frame by literal name	The [[operator is used to extract elements of a list or data frame by string name	The ((operator is used to extract elements of a list or data frame by string name	There are three operators that can be used to extract subsets of R objects
9	Which of the following extracts first four element from the following vector ? > x <- c("a", "b", "c", "c", "d", "a")	x[0:4].	x[0:3].	x[1:4].	x[1:3].	x[1:4].
10	What would be the output of the following code ? x <- c("a", "b", "c", "c", "d", "a") > x[c(1, 3, 4)]	“a” “b” “c”	“a” “c” “c”	“a” “c” “b”	“a” “b” “b”	“a” “c” “c”

11	Point out the wrong statement :	$ operator semantics are similar to that of [[	The [operator always returns an object of the same class as the original	The $ operator is used to extract elements of a list or a data frame	The [[operator is used to extract elements of a list or a data frame	The $ operator is used to extract elements of a list or a data frame
12	What would be the output of the following code ? > x <- matrix(1:6, 2, 3) > x[1, 2]	3	2	1	0	3
13	What would be the output of the following code ? > x <- matrix(1:6, 2, 3) > x[1,]	1 3 5	2 3 5	3 3 5	file	1 3 5
14	Which of the following code extracts the second column for the following matrix ? > x <- matrix(1:6, 2, 3)	x[2,].	x[1, 2].	x[, 2].	x[2, 2].	x[, 2].
15	Point out the wrong statement :	$ operator semantics are similar to that of [[	The [[operator can take an integer sequence if you want to extract a nested element of a list	The $ operator can be used to extract multiple elements from a list	There are three operators that can be used to extract subsets of R objects	The $ operator can be used to extract multiple elements from a list
16	Which of the following code extracts 1st element of the 2nd element ? > x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))	x[[c(2, 1)]].	x[[c(1, 2)]].	x[[c(2, 1,1)]].	x[[c(2, 0,1)]].	x[[c(2, 1)]].
17	_____ , for dumping a textual representation of multiple R objects	dput	save	dump	serialize	dump
18	_____ , for outputting a textual representation of an R object	dput	save	dump	serialize	dput
19	_____ , for saving an arbitrary number of R objects in binary format (possibly compressed) to a file.	dput	save	dump	serialize	save
20	_____ , for converting an R object into a binary format for outputting to a connection (or file).	dput	save	dump	serialize	serialize
21	_____ string indicating how the columns are separated	sep	colClasses	nrows	file	sep
22	_____ character vector indicating the class of each column in the dataset	sep	colClasses	nrows	file	colClasses

23	_____ the number of rows in the dataset. By default read.table() reads an entire file	sep	colClasses	nrows	file	nrows
24	_____ logical indicating if the file has a header line	sep	colClasses	nrows	header	header
25	_____ character string indicating the comment character	sep	colClasses	comment.char	header	comment.char
26	Partial matching of names is allowed with ____ and __	[and $	[[and [	[[and [$	[[and $	[[and $
27	The _____ operator can take an integer sequence if you want to extract a nested element of a list.	$	[[	[	((	[[
28	The _____ operator can be used to extract single elements from a list	$	[	[[	((	[[
29	The ____ operator to extract elements by name	$	[	[[	((	$
30	The _____ function can be useful for reading in lines of webpages	Load()	readLines()	read()	readpage()	readLines()
31	Text files can be read line by line using the _____function.	Load()	readpage()	read()	readLines()	readLines()
32	The _____ package is recently developed by Hadley Wickham to deal with reading in large flat files quickly.	readr	dplyr	read	dr	readr
33	The _____ and _____ functions are useful because the resulting textual format is editable, and in the case of corruption, potentially recoverable.	dump() and dget()	dump() and dput()	dget() and dput()	dump() and dp()	dump() and dput()
34	_____ opens a connection to a file	file	gzfile	bzfile	url	file
35	_____ opens a connection to a file compressed with gzip	file	gzfile	bzfile	url	gzfile
36	_____ opens a connection to a file compressed with bzip2	file	gzfile	bzfile	url	bzfile
37	_____ opens a connection to a webpage	file	gzfile	bzfile	url	url
38	The ____ function has a number of arguments that are common to many other connection	f()	close()	file()	open()	file()
39	_____ open file in read only mode	“r”	“a”	“w”	"ab"	“r”
40	_____ open a file for writing (and initializing a new file)	“r”	“a”	“w”	"ab"	“w”

41	_____ open a file for appending	“r”	“a”	“w”	"ab"	“a”
42	The_____ operator can be used to extract multiple elements of a vector by passing the operator an integer sequence	$	[	[[	((	[
43	What would be the output of the following code ? > x <- list(foo = 1:4, bar = 0.6, baz = "hello") > name <- "foo" > x[[name]]	1 2 3 4	0 1 2 3	1 2 3 4 5	1 2 3 5	1 2 3 4
44	What would be the output of the following code ? > x <- list(aardvark = 1:5) > x$a	2 3 5	1 3 3 5	1 2 3	1 2 3 4 5	1 2 3 4 5
45	What would be the output of the following code ? > x <- list(foo = 1:4, bar = 0.6, baz = "hello") > name <- "foo" > x$name	1	3	2	4	2
46	What would be the output of the following code ? > x <- list(a = list(10, 12, 14), b = c(3.14, 2.81)) > x[[c(1, 3)]]	13	14	15	16	14
47	The _____ function is used to convert individual R objects into a binary format that can be communicated across an arbitrary connection.	dput()	save()	serialize()	dump()	serialize()
48	Matrices can be subsetted in the usual way with (i,j) type _____	subset	subsetting	indices	sets	indices
49	The main functions for converting R objects into a binary format are	save(), save.image(), and unserialize()	save(), save.image(), and serialize()	save(), unserialize, and serialize()	unserialize(), save.image(), and serialize()	save(), save.image(), and serialize()
50	The _____ function is one of the most commonly used functions for reading data in R	read.csv()	read.table()	read.data()	read()	read.table()
51	_____, a character vector indicating the class of each column in the dataset	sep	header	file	colClasses	colClasses
52	The inverse of dump() is _____ function	file()	dput()	source()	dum()	source()

53	Vectors are basic objects in R and they can be subsetted using the _____ operator	((	[	[]	[[	[
54	The _____ function is identical to read.table except that some of the defaults are set differently	read.csv()	read.table()	read()	read.data()	read.csv()
55	Factors are important in statistical modeling and are treated specially by modelling functions like ___ and _____.	l() and gl()	lm() and glm().	lme() and glme()	m() and gm()	lm() and glm().
56	We can also create an empty list of a prespecified length with the_____function	create()	file()	vector()	list()	vector()
57	The sequence does not have to be in order; you can specify any _____ integer vector.	specified	legel	unarbitrary	arbitrary	arbitrary
58	The [[operator can be used to extract _____ elements from a list.	no	all	single	double	single
59	The $ operator can only be used with _____ names.	different	literal	same	unique	literal
60	A common task in data analysis is removing _____	missing values	segments	changing values	names	missing values
61	In R every operation has a _____call?	system	function	compiler	interprter	function
62	How many types of R objects are present in R data type?	4	5	6	7	6
63	R cosiders as an alternate execution of ___?	SAS	SPSS	S	SASS	S
64	The _____ in R is a vector.	basic data structure	basic datatypes	class	object	basic data structure
65	How many types of vetrices functions are peresent?	1	2	3	4	2
66	_____and_____ are types of matrices functions?	apply and sapply	**apply and lapply**	both	apply	apply and sapply
67	What will be the output of the following code? centre <-function(x, type){ switch(type, mean=mean(x), median=median(x), trimmed =mean(x, trim = .1)) } x <-rcauchy(10) centre(x, "mean")	0.8760325	0.5360891	0.6086504	Random Value	Random Value

68	_____ is used to skip an iteration of a loop.	next	skip	group	break	next
69	Which of the following code snippets stops a loop after 20 iterations?	for(i in 1:100){ if(i >20){ break } print(i) }	for(i in 1:100){ if(i >19){ break } print(i) }	for(i in 1:100){ if(i <20){ break } print(i) }	for(i in 1:100){ if(i <19){ break } print(i)	for(i in 1:100){ if(i >20){ break } print(i) }
70	Which of the following true about loops in R?	The only way to exit a repeat loop is to call break.	Infinite loops should be avoided.	Neither a nor b is correct.	Both a and b are correct.	Both a and b are correct.
71	R functionality is divided into a number of _____.	packages	functions	domains	classes	packages
72	_____ initiates an infinite loop right from the start.	never	repeat	break	set	repeat
73	_____ is used to skip an iteration of a loop.	next	skip	group	break	next
74	What will be the output of the following code? x <-2 switch(2, 2+2, mean(1:10), rnorm(5))	5	5.5	NULL	both a and c	5.5
75	Identify the correct statement about lists in R.	The search list can be found by using the searchlist() function	The search list can be found by using the search() function.	The global environment or the user's workspace is always the second element of the search list.	Both a and c are correct	The search list can be found by using the search() function.
76	The _____ feature of R is the main feature that makes it different from any original S language.	scoping rules	closure rules	environment rules	return rules	scoping rules
77	_____ extracts a subset of rows from a data frame based on the logical conditions.	Rename	filter	set	subset	filter
78	_____adds new variables/columns or transforms existing variables.	mutate	add	append	arrange	mutate

79	To get the current date, the _____ function will return a Date object which can be converted to a different class if necessary.	Sys.Time	Sys.Date	Sys.DateTime	All of the mentioned	Sys.Date
80	What would be the value of following expression ? > log(-2.3)	Warning in log(-2.3): NaNs produced	1	Null	0	Warning in log(-2.3): NaNs produced
81	_____ allows you to insert debugging code into a function a specific places	debug()	trace()	browser()	all of the mentioned	debug()

UNIT-III

Syllabus

Getting and Cleaning Data: Obtaining data from the web, from APIs, from databases and from colleagues in various formats. Basics of data cleaning and making data ―tidy.

Overview

- finding and extracting raw data
- today any how to make data tiny
- Raw data →→ processing script →→ tidy data →→ data analysis →→ data communication

Raw and processed data

- **Data** = values of qualitative/quantitative, variables, belonging to a set of items
  - **variables** = measurement of characteristic of an item
- **Raw data** = original source of data, often hard to use, processing must be done before analysis, may need to be processed only once
- **Processed data** = ready for analysis, processing done (merging, transforming, etc.), all steps should be recorded
- Sequencing DNA: $1B for Human Genome Project →→ $10,000 in a week with Illumina

Tidy Data

1. **Raw Data**
   - no software processing has been done
   - did not manipulate, remove, or summarize in anyway
2. **Tidy data set**
   - end goal of cleaning data process
   - each variable should be in one column
   - each observation of that variable should be in a different row
   - one table for each kind of variable

        - if there are multiple tables, there should be a column to link them
    - include a row at the top of each file with variable names (variable names should make sense)
    - in general data should be save in one file per table

3. **each variable and its values in the tidy data set**
    - information about the variables (w/ units) in dataset ***NOT*** contained in tidy data
    - information about the summary choice that were made (median/mean)
    - information about experimental study design (data collection methods)
    - common format for this document = markdown/Word/text
        - ***"study design"*** section = thorough description of how data was collected
        - ***"code book"*** section = describes each variable and units
4. **Explicit steps and exact recipe to get through 1 - 3 (instruction list)**
    - ideally a computer script (no parameters)
    - output = processed tidy data
    - in addition to script, possibly may need steps to run files, how script is run, and explicit instructions

Download files

- **Set working directory**
    - ***Relative***: setwd("./data"), setwd("../") = move up in directory
    - ***Absolute***: setwd("/User/Name/data")
- **Check if file exists and download file**
    - if(!file.exists("data"){dir.create("data")}
- **Download file**
    - download.file(url, destfile= "directory/filname.extension", method = "curl")
        - method = "curl" [mac only for https]
    - dateDownloaded <- date() = record the download date
- **Read file and load data**

- read.table() = need to specify file, header, sep, row.names, nrows
  - read.csv() = automatically set sep = "," and header = TRUE
- quote = "" = no quotes (extremely helpful, common problem)
- na.strings = set the character that represents missing value
- nrows = how many rows to read
- skip = how many lines to skip
- col.names = specifies column names
- check.names = TRUE/FALSE = If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names and are not duplicated. If necessary they are adjusted appropriately to be syntactically valid names

Reading Excel files

- xlsx package: read.xlsx(path, sheetIndex = 1, ...)
  - colIndex, rowIndex = can be used to read certain rows and columns
- write.xlsx() = write out excel file
- read.xlsx2() = faster than read.xlsx() but unstable for reading subset of rows
- XLConnect package has more options for writing/manipulating Excel files
- generally good to store data in database/csv/tab separated files (.tab/.txt), easier to distribute

Reading XML

- **XML** = extensible markup language
- frequented used to store structured data, widely used in Internet apps
- extracting XML = basis for most of web scraping
- components
  - ***markup*** = labels that give text structure
  - ***content*** = actual text of document
- tags = <section>, </section>, <line-break />
- elements = <Greeting> test </Greeting>

- attributes = <image src ="a.jpg" alt = "b">
- reading file into R
    - library(XML)
    - doc <- xmlTreeParse(fileUrl, useInternal = TRUE) = loads data
    - rootNode <- xmlRoot(doc) = wrapper element for entire document
    - xmlName(rootNode) = returns name of the document
    - names(rootNode) = return names of elements
    - rootNode[[1]] = access first elements, similar to list
    - rootNode[[1]][[1]] = first sub component in the first element
    - xmlSApply(rootNode, xmlValue) = returns every single tagged element in the doc
- **XPath**
    - get specific elements of document
    - /node = top level node
    - //node = node at any level
    - node[@attr-name = 'bob'] = node with attribute name
        - xpathSApply(rootNode, "//name", xmlValue) = get the values of all elements with tag “name”
        - xpathSApply(rootNode, "//price", xmlValue) = get the values of all elements with tag “price”
- **extract content by attributes**
    - doc <- htmlTreeParse(url, useInternal = True)
    - scores <- xpathSApply(doc, "//li@class='score'", xmlvalue) = look for li elements with class = "score" and return their value

Reading JSON

- **JSON** = JavaScript Object Notation
- lightweight data storage, common format for data from application programming interfaces (API)
- similar to XML in structure but different in syntax/format

- data can be stored as:
  - numbers (double)
  - strings (double quoted)
  - boolean (true/false)
  - array (ordered, comma separated enclosed in [])
  - object (unordered, comma separated collection of key/value pairs enclosed in {})
- jsonlite package (json vignette can be found in help)
  - library(jsonlite) = loads package
  - data <- fromJSON(url) = strips data
    - names(data$owner) = returns list of names of all columns of owner data frame
    - data$owner$login = returns login instances
  - data <- toJSON(dataframe, pretty = TRUE) = converts data frame into JSON format
    - pretty = TRUE = formats the code nicely
  - cat(data) = prints out JSON code from the converted data frame
  - fromJSON() = converts from JSON object/code back to data frame

data.table

- inherits from data.frame (external package) →→ all functions that accept data.frame work on data.table
- can be much faster (written in C), ***much much faster*** at subsetting/grouping/updating
- **syntax**: dt <- data.table(x = rnorm(9), y = rep(c(a, b, c), each = 3), z = rnorm(9)
- tables() = returns all data tables in memory
  - shows name, nrow, MB, cols, key
- some subset works like before = dt[2,], dt[dt$y=="a",]
  - dt[c(2, 3)] = subset by rows, rows 2 and 3 in this case
- **column subsetting** (modified for data.table)

- argument after comma is called an ***expression*** (collection of statements enclosed in {})
- dt[, list(means(x), sum(z)] = returns mean of x column and sum of z column (no "" needed to specify column names, x and z in example)
- dt[, table(y)] = get table of y value (perform any functions)

- **add new columns**
  - dt[, w:=z^2]
    - when this is performed, a new data.table is created and data copied over (not good for large datasets)
  - dt2 <- dt; dt[, y:= 2]
    - when changes are made to dt, changes get translated to dt2
    - ***Note****: if copy must be made, use the copy() function instead*
- **multiple operations**
  - dt[, m:= {temp <- (x+z); log2(temp +5)}] →→ adds a column that equals log2(x+z + 5)
- **plyr like operations**
  - dt[,a:=x>0] = creates a new column a that returns TRUE if x > 0, and FALSE other wise
  - dt[,b:=mean(x+w), by=a] = creates a new column b that calculates the aggregated mean for x + w for when a = TRUE/FALSE, meaning every b value is gonna be the same for TRUE, and others are for FALSE
- **special variables**
  - .N = returns integer, length 1, containing the number (essentially count)
    - dt <- data.table (x=sample(letters[1:3], 1E5, TRUE)) = generates data table
    - dt[, .N by =x] = creates a table to count observations by the value of x
- **keys** (quickly filter/subset)
  - *example*: dt <- data.table(x = rep(c("a", "b", "c"), each 100), y = rnorm(300)) = generates data table
    - setkey(dt, x) = set the key to the x column

- dt['a'] = returns a data frame, where x = 'a' (effectively filter)

- **joins** (merging tables)
  - *example*: dt1 <- data.table(x = c('a', 'b', ...), y = 1:4) = generates data table
    - dt2 <- data.table(x= c('a', 'd', ...), z = 5:7) = generates data table
    - setkey(dt1, x); setkey(dt2, x) = sets the keys for both data tables to be column x
    - merge(dt1, dt2) = returns a table, combine the two tables using column x, filtering to only the values that match up between common elements the two x columns (i.e. 'a') and the data is merged together

- **fast reading of files**
  - *example*: big_df <- data.frame(norm(1e6), norm(1e6)) = generates data table
    - file <- tempfile() = generates empty temp file
    - write.table(big.df, file=file, row.names=FALSE, col.names = TRUE, sep = "\t". quote = FALSE) = writes the generated data from big.df to the empty temp file
    - fread(file) = read file and load data = much faster than read.table()

Reading from MySQL

- install.packages("RMySQL"); library(RMySQL) = load MySQL package
- free/widely used open sources database software, widely used for Internet base applications
- each row = record
- data are structured in databases →→ series tables (dataset) →→ fields (columns in dataset)
- dbConnect(MySQL(), user = "genome", db = "hg19", host = "genome-mysql.cse.ucsc.edu) = open a connection to the database
  - db = "hg19" = select specific database
  - MySQL() can be replaced with other arguments to use other data structures
- dbGetQuery(db, "show databases;") = return the result from the specified SQL query executed through the connection
  - any SQL query can be substituted here
- dbDisconnect(db) = disconnects the open connection

- crucial to disconnect as soon as all queries are performed

- dbListFields(db, "name") = returns the list of fields (columns) from the specified table
- dbReadTable(db, "name") = returns the the specified table
- query <- dbSendQuery(db, "query") = send query to MySQL database and store it remotely
- fetch(query, n = 10) = executes query and returns the result
  - n = 10 = returns the first 10 rows
- dbClearResult(query) = clears query from remote database, important
- sqldf package example

Web Scraping

- **webscraping** = programmatically extracting data from the HTML code of websites
- con = url("website") = opens connection from URL
- htmlCode = readLines(con) = reads the HTML code from the URL
  - always remember to close(con) after using it
  - the htmlCode return here is a bit unreadable
- **Parsing with XML**
  - library(XML)
  - url <- "http://..." = sets the desired URL as a character variable
  - html <- htmlTreeParse(url, useInternalNodes = T) = reads and parses the html code
  - xpathSApply(html, "//title", xmlValue) = returns the value of the //title node/element
  - xpathSApply(html, "//td[@id='col-citedBy']", xmlValue) = returns the value of the //td element where the id = 'col-citedBy' in the html code
- **Parsing with httr package**
  - library(httr)
  - html2 <- GET(url) = reads the HTML code from the URL
  - cont = content(html2, as = "text") = extracts the HTML code as a long string
  - parsedHtml = htmlParse(cont, asText = TRUE) = parses the text into HTML (same output as the XML package function htmlTreeParse)

- xpathSApply(html, "//title", xmlValue) = returns the value of the //title node/element
- accessing websites with passwords
  - pg = GET("url") = this would return a status 401 if the website requires log in without authenticating
  - pg2 = GET("url", authenticate("username", "password")) = this authenticates before attempting to access the website, and the result would return a status 200 if authentication was successful
  - names2(pg2) = returns names of different components
- using handles (username/login information)
  - using handles allows you to save authentication across multiple parts of the website (only authenticate once for different requests)
  - *example*: google = handle("http://google.com")
  - pg1 = GET(handle = google, path = "/")
  - pg2 = GET(handle = google, path = "search")

Working with API

- load http package first: library(httr)
  - allows GET, POST, PUT, DELETE requests if you are authorized
- myapp = oath_app("app", key = "consumerKey", secret = "consumerSecret") = start authorization process for the app
- sig = sign_oauth1.0(myapp, token = "tokenGenerated", token_secret = "tokenSecret") = login using the token information (sets up access so you can use it to get data)
- homeTL = get("url", sig) = use the established authentication (instead of username/password) to get the data (usually in JSON format)
  - use the url to specify what data you would like to get
  - use the documentation to get information and parameters for the url and data you have access to
- json1 = content(homeTL) = recognizes the data in JSON format and converts it to a structured R object [a bit hard to read]
- json2 = jsonlite::fromJSON(toJSON(json1)) = converts data back into JSON format and then use the fromJSON function from the jsonlite package to read the data into a data frame

- each row corresponds to a line of the data you received

- **GitHub example**
  - library(httr)
  - myapp <- oauth_app("github", key = "clientID", secret = "clientSecret")
    - an application must be registered with GitHub first to generate the client ID and secrets
  - github_token <- oauth2.0_token(oauth_endpoints("github"), myapp)
    - oauth_endpoints() = returns the the authorize/access url/endpoints for some common web applications (GitHub, Facebook, google, etc)
    - oauth2.0_token(endPoints, app) = generates an oauth2.0 token with the credentials provided
  - gtoken <- config(token = github_token) = sets up the configuration with the token for authentication
  - req <- with_config(gtoken, GET("https://api.github.com/rate_limit")) = executes the configuration set to send a get request from the specified URL, and returns a response object
  - library(jsonlite); json1 <- fromJSON(toJSON(content(req))) = converts the content of the response object, to JSON format, and converts it again to data frame format
  - names(json1) = returns all the column names for the data frame
  - json1[json1$name == "datasharing",]$created_at = returns the create date for the data sharing repo

Reading from Other Sources

- **interacting directly with files**
  - file = open a connection to a text file
  - url = opens a connection to a URL
  - gzfile/bzfile = opens a connection to a .gz/.bz2 file
  - ?connections = for more information about opening/closing connections in R
- **foreign package**
  - loads data from Minitab/S/SAS/SPSS/Stat/Systat

- basic functions
  - read.arff (Weka)
  - read.dta (Stata)
  - read.mtp (Minitab)
  - read.octave (Octave)
  - read.spss (SPSS)
  - read.xport (SAS)
  - read.fwf (fixed width files, [.for])
  - *example*: data <- read.fwf(file = "quiz02q5.for", skip = 4, widths = c(-1, 9,-5, 4, 4, -5, 4, 4,-5, 4, 4,-5, 4, 4))
  - widths = c() = specifies the width of each variable
  - the negative numbers indicate the space to disregard/take out

- **Other packages/functions**
  - RPostresSQL = provides DBI-compliant database connection from R
  - RODBC = provides interfaces to multiple databases including PostgreQL, MySQL, Microsoft Access, SQLite
  - RMongo/rmongodb = provides interfaces to MongoDb
    - similar to MySQL, except send queries in the database's syntax
  - reading images (functions)
    - jpeg, readbitmap, png, EBImage (Bioconductor)
  - reading (GIS Geographical Information Systems) data (packages)
    - rdgal, rgeos, raster
  - reading music data (packages)
    - tuneR, seewave

dplyr

- external package, load by library(dplyr)
  - developed by Hadley Wickham of RStudio

- optimized/distilled version of the plyr package, does not provide new functionality but greatly simplifies existing R functionality
- very fast, many key operations coded in C++
- dplyr package also works on data.table and SQL interface for relational databases (DBI package)

- load data into tbl_df (data frame table) by data <- tbl_df(rawData)
  - main advantage to using a tbl_df over a regular data frame is printing
  - more compact output/informative = similar to a combination of head/str
    - displays class, dimension, preview of data (10 rows and as many columns as it can fit), undisplayed variables and their class

- **functions**
  - ***Note****: for all functions, first argument always the data frame, and result is always a data frame*
  - select()
    - *example*: select(dataFrameTable, var1, var2, var3) = returns a table (similar in format as calling the actual data frame table)
    - no need to use $ as we would normally, since select() understands that the variables are from the dataFrameTable
    - columns are returns in order specified
    - : operator (normally reserved for numbers) can be used to select a range of columns (from this column to that column), works in reverse order as well = select(dataFrameTable, var1:var5)
    - "-column" can be used to specify columns to throw away = select(dataFrameTable, -var1) = but this does not modify original dataFrameTable
    - -(var1:size) = eliminate all columns
    - normally this can be accomplished by finding the indices of names using the match("value", vector) function
  - filter()

- *example*: filter(cran, package == "swirl") = returns a table (similar in format as calling the actual data frame table)
- returns all rows where the condition evaluates to TRUE
- automatically recognized that package is a column without $
- able to specify as many conditions as you want, separated by ,, | and & work here as well
- multiple conditions specified by , is equivalent to &
- is.na(var1) also works here
- ***Note**: ?Comparison brings up relevant documentation for relational comparators*

- arrange()
  - *example*: arrange(dataFrameTable, var) = order the data frame table by specified column/variable
  - desc(var) = arrange in descending order by column value
  - can specify multiple values to sort by by using ,
  - order listed in the call will be the order that the data is sorted by (can use in conjunction with desc())
- rename()
  - *example*: rename(dataFrameTable, newColName = colName) = renames the specified column with new name
  - capable of renaming multiple columns at the same time, no quotes needed
- mutate()
  - create a new variable based on the value of one or more existing variables in the dataset
  - capable of modifying existing columns/variables as well
  - *example*: mutate(dataFrameTable, newColumn = size / 2^20) = create a new column with specified name and the method of calculating
  - multiple columns can be created at the same time by using , as separator, new variables can even reference each other in terms of calculation

- summarize()
  - collapses the dataset into a single row
  - *example*: summarize(dataFrameTable, avg = mean(size)) = returns the mean from the column in a single variable with the specified name
  - summarize(can return the requested value for each group in the dataset
- group_by()
  - *example*: by_package <- group_by(cran, package) = creates a grouped data frame table by specified variable
  - summarize(by_package, mean(size)) = returns the mean size of each group (instead of 1 value from thesummarize() example above)
  - ***Note**: n() = counts number of observation in the current group*
  - ***Note**: n_distinct() = efficiently count the number of unique values in a vector*
  - ***Note**: quantile(variable, probs = 0.99) = returns the 99% percentile from the data*
  - ***Note**: by default, dplyr prints the first 10 rows of data if there are more than 100 rows; if there are not, it will print everything*
- rbind_list()
  - bind multiple data frames by row and column
  - *example*: rbind_list(passed, failed)

- **Chaining/Piping**
  - allows stringing together multiple function calls in a way that is compact and readable, while still accomplishing the desired result
    - ***Note**: all variable calls refer to the tbl_df specified at the same level of the call*
  - %>% = chaining operator
    - ***Note**: ?chain brings up relevant documentation for the chaining operator*
    - Code on the right of the operator operates on the result from the code on the left
    - exp1 %>% exp2 %>% exp3 ...

- exp1 is calculated first
- exp2 is then applied on exp1 to achieve a result
- exp3 is then applied to the result of that operation, etc.

- ***Note**: the chaining aspect is done with the data frame table that is being passed from one call to the next*
- ***Note**: if the last call has no additional arguments, print() for example, then it is possible to leave() off*

tidyr

- gather()
  - gather columns into key value pairs
  - *example*: gather(students, sex, count, -grade) = gather each key (in this case named sex), value (in this case count) pair into one row
    - effectively translates to (columnName, value) with new names imposed on both = all combinations of column name and value
    - -grade = signifies that the column does not need to be remapped, so that column is preserved
    - class1:class5 = can be used instead to specify where to gather the key values
- separate()
  - separate one column into multiple column
  - *example*: separate(data = res, col = sex_class, into = c("sex", "class") = split the specified column in the data frame into two columns
    - ***Note**: the new columns are created in place, and the other columns are pushed to the right*
    - ***Note**: separate() is able to automatically split non-alphanumeric values by finding the logical separator; it is also possible to specify the separator by using the sep argument*
- spread()
  - spread key-value pairs across multiple columns = turn values of a column into column headers/variables/new columns

- *example*: spread(students3, test, grade) = splits "test" column into variables by using it as a key, and "grade" as values
  - ***Note**: no need to specify what the columns are going to be called, since they are going to be generated using the values in the specified column*
  - ***Note**: the value will be matched and split up according their alignment with the key ("test") = midterm, A*

- extract_numeric()
  - extract numeric component of variable
  - *example*: extract_numeric("class5") = returns 5
  - *example*: mutate(class = extract_numeric(class)) = changes the class name to numbers only
- unique() = general R function, not specific to tidyr
  - returns a vector with the duplicates removed
- ***Note**: when there are redundant information, it's better to split up the info into multiple tables; however, each table should also contain primary keys, which identify observations and link data from one table to the next*

lubridate

- consistent, memorable syntax for working with dates
- wday(date, label = TRUE) = returns number 1 - 7 representing Sunday - Saturday, or returns three letter day of the week if label = TRUE
- today(), now() = returns the current date and time, with extractable parts (hour(), month())
- tzone = "America/New_York" = used to specify time zones
- ymd("string") = converts string in to year month day format to a POSIXct time variable
  - mdy("string") = parses date in month day year format
  - dmy(2508195) = parses date in day month year format using a number
  - ymd_hms("string") = parses the year month day, hour minute second
  - hms("string") = parses hour minute second
    - tz = "" = can use the "tz" argument to specify time zones

- ***Note**: there are a variety of functions that are available to parse different formats, all of them are capable of converting the correct information if the order of month year day is correct*
- ***Note**: when necessary, // or — should be added to provide clarity in date formatting*

- update(POSIXct, hours = 8, minutes = 34, seconds = 55) = updates components of a date time
  - ***Note**: does not alter the date time passed in unless explicitly assigned*
- arithmetic can be performed on date times by using the days() hours() minutes(), etc. functions
  - *example*: now() + hours(5) + minutes(2) = returns the date time for 5 hours and 2 minutes from now
- with_tz(time, tone ="") = return date-time in a different time zone
- as.period(new_interval(last_time, arrive)) = return the properly formatted difference between the two date times

Missing values

Changing the representation of a dataset brings up an important subtlety of missing values. Surprisingly, a value can be missing in one of two possible ways:

- **Explicitly**, i.e. flagged with NA.
- **Implicitly**, i.e. simply not present in the data.

Let's illustrate this idea with a very simple data set:

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

There are two missing values in this dataset:

- The return for the fourth quarter of 2015 is explicitly missing, because the cell where its value should be instead contains NA.

- The return for the first quarter of 2016 is implicitly missing, because it simply does not appear in the dataset.

One way to think about the difference is with this Zen-like koan: An explicit missing value is the presence of an absence; an implicit missing value is the absence of a presence.

The way that a dataset is represented can make implicit values explicit. For example, we can make the implicit missing value explicit by putting years in the columns:

```
stocks %>%
  spread(year, return)
#> # A tibble: 4 x 3
#>     qtr `2015` `2016`
#>   <dbl>  <dbl>  <dbl>
#> 1     1   1.88  NA
#> 2     2   0.59   0.92
#> 3     3   0.35   0.17
#> 4     4  NA      2.66
```

Because these explicit missing values may not be important in other representations of the data, you can set na.rm = TRUE in gather() to turn explicit missing values implicit:

```
stocks %>%
  spread(year, return) %>%
  gather(year, return, `2015`:`2016`, na.rm = TRUE)
#> # A tibble: 6 x 3
#>     qtr year  return
#> * <dbl> <chr>  <dbl>
#> 1     1 2015    1.88
#> 2     2 2015    0.59
#> 3     3 2015    0.35
#> 4     2 2016    0.92
#> 5     3 2016    0.17
```

```
#> 6    4 2016    2.66
```

Another important tool for making missing values explicit in tidy data is complete():

```
stocks %>%
  complete(year, qtr)
#> # A tibble: 8 x 3
#>    year   qtr return
#>   <dbl> <dbl>  <dbl>
#> 1  2015     1   1.88
#> 2  2015     2   0.59
#> 3  2015     3   0.35
#> 4  2015     4  NA
#> 5  2016     1  NA
#> 6  2016     2   0.92
#> # ... with 2 more rows
```

complete() takes a set of columns, and finds all unique combinations. It then ensures the original dataset contains all those values, filling in explicit NAs where necessary.

There's one other important tool that you should know for working with missing values. Sometimes when a data source has primarily been used for data entry, missing values indicate that the previous value should be carried forward:

```
treatment <- tribble(
  ~ person,           ~ treatment, ~response,
  "Derrick Whitmore", 1,           7,
  NA,                 2,           10,
  NA,                 3,           9,
  "Katherine Burke",  1,           4
)
```

You can fill in these missing values with fill(). It takes a set of columns where you want missing values to be replaced by the most recent non-missing value (sometimes called last observation carried forward).

```
treatment %>%
  fill(person)
#> # A tibble: 4 x 3
#>   person           treatment response
#>   <chr>                <dbl>    <dbl>
#> 1 Derrick Whitmore         1        7
#> 2 Derrick Whitmore         2       10
#> 3 Derrick Whitmore         3        9
#> 4 Katherine Burke          1        4
```

The purpose of tidy data for data science

Definition:

"Happy families are all alike; every unhappy family is unhappy in its own way." — Leo Tolstoy

"Tidy datasets are all alike, but every messy dataset is messy in its own way." — Hadley Wickham

Tidy data

You can represent the same underlying data in multiple ways. The example below shows the same data organized in four different ways. Each dataset shows the same values of four variables *country*, *year*, *population*, and *cases*, but each dataset organizes the values in a different way.

table1

```
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
```

#> 4 Brazil 2000 80488 174504898

#> 5 China 1999 212258 1272915272

#> 6 China 2000 213766 1280428583

table2

#> # A tibble: 12 x 4

#> country year type count

#> <chr> <int> <chr> <int>

#> 1 Afghanistan 1999 cases 745

#> 2 Afghanistan 1999 population 19987071

#> 3 Afghanistan 2000 cases 2666

#> 4 Afghanistan 2000 population 20595360

#> 5 Brazil 1999 cases 37737

#> 6 Brazil 1999 population 172006362

#> # ... with 6 more rows

table3

#> # A tibble: 6 x 3

#> country year rate

*#> * <chr> <int> <chr>*

#> 1 Afghanistan 1999 745/19987071

#> 2 Afghanistan 2000 2666/20595360

#> 3 Brazil 1999 37737/172006362

#> 4 Brazil 2000 80488/174504898

#> 5 China 1999 212258/1272915272

#> 6 China 2000 213766/1280428583

 # Spread across two tibbles

table4a # *cases*

```
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
```

table4b # *population*

```
#> # A tibble: 3 x 3
#>   country      `1999`     `2000`
#> * <chr>         <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

These are all representations of the same underlying data, but they are not equally easy to use. One dataset, the tidy dataset, will be much easier to work with inside the tidyverse.

There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

Figure: 1 shows the rules visually.

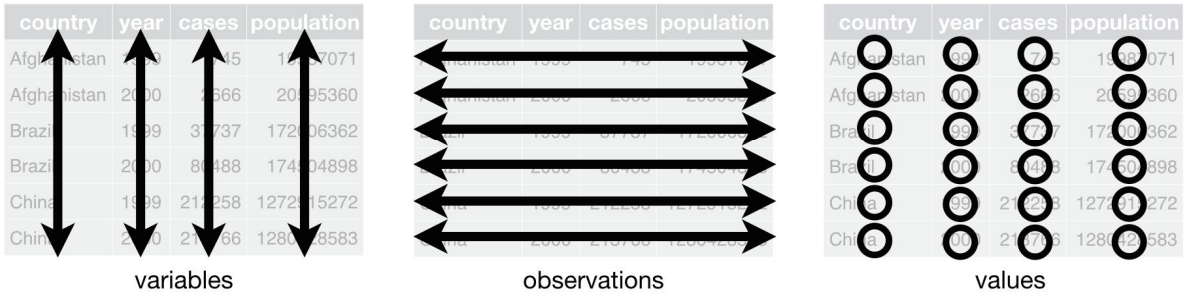


Figure: 1 following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

These three rules are interrelated because it's impossible to only satisfy two of the three. That interrelationship leads to an even simpler set of practical instructions:

1. Put each dataset in a tibble.
2. Put each variable in a column.

In this example, only table1 is tidy. It's the only representation where each column is a variable.

Why ensure that your data is tidy? There are two main advantages:

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.
2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

dplyr, ggplot2, and all the other packages in the tidyverse are designed to work with tidy data. Here are a couple of small examples showing how you might work with table1.

 # Compute rate per 10,000

table1 %>%

mutate(rate = cases / population * 10000)

#> # A tibble: 6 x 5

#> country year cases population rate

#> <chr> <int> <int> <int> <dbl>

#> 1 Afghanistan 1999 745 19987071 0.373

#> 2 Afghanistan 2000 2666 20595360 1.29

#> 3 Brazil 1999 37737 172006362 2.19

#> 4 Brazil 2000 80488 174504898 4.61

#> 5 China 1999 212258 1272915272 1.67

#> 6 China 2000 213766 1280428583 1.67

 # Compute cases per year

table1 %>%

count(year, wt = cases)

#> # A tibble: 2 x 2

#> year n

#> <int> <int>

#> 1 1999 250740

#> 2 2000 296920

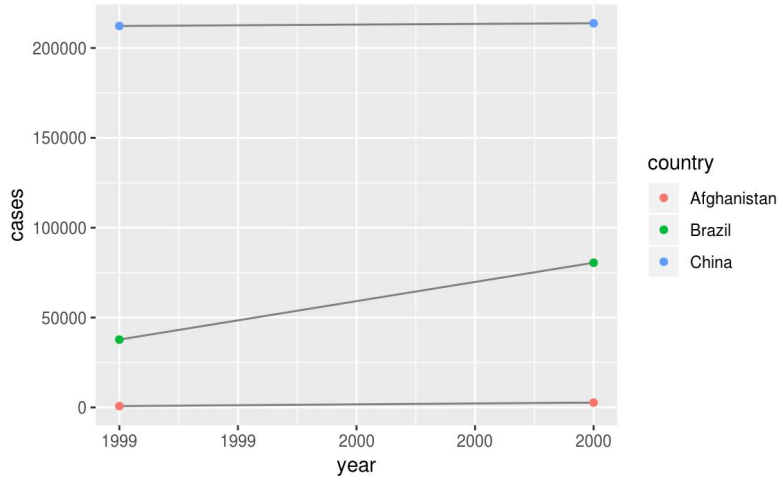
 # Visualise changes over time

library(ggplot2)

ggplot(table1, **aes**(year, cases)) +

geom_line(**aes**(group = country), colour = "grey50") +

geom_point(**aes**(colour = country))



POSSIBLE QUESTIONS

2 MARKS

1. Define raw data.
2. Define processed data.
3. What is tidy data?
4. What is web scraping?
5. Define mutate( ), and arrange( ) functions.
6. Define chaining/piping?
7. What is tidyr? Mention its function.
8. Define lubridate.
9. What is missing values? Mention its types.
10. What is dplyr?

6 MARKS

1. Write a detailed note on getting data from various sources.
2. Illustrate the purpose of tidy data in data science.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

UNIT - III

sno	Questions	Opt1	Opt2	Opt3	Opt4	Answer
1	Which of the following is example of vectorized operation as far as subtraction is concerned ? > x <- 1:4 > y <- 6:9	x+y	x-y	x*y	x/y	x-y
2	Point out the wrong statement :	Very less operations in R are vectorized	Vectorization allows you to write code that is efficient, concise, and easier to read than in non-vectorized languages	vectorized means that operations occur in parallel in certain R objects	Many operations in R are vectorized, meaning that operations occur in parallel in certain R objects.	Very less operations in R are vectorized
3	What would be the output of the following code ? > x <- 1:4 > y <- 6:9 > z <- x + y > z	7 9 11 13	7 9 11 13 14	9 7 11 13	7 9 11 14	7 9 11 13
4	Which of the followin code represents internal representation of a Date object ?	class(as.Date(“1970-01-02”))	unclass(as.Date(“1970-01-02”))	unclassint(as.Date(“1970-01-02”))	classint(as.Date(“1970-01-02”))	unclass(as.Date(“1970-01-02”))
5	What would be the output of the following code ? > x <- Sys.time() > class(x)	“POSIXct” “POSIXt”	“POSIXXt” “POSIXt”	“POSIXct” “POSIct”	“POSIXct” “POSIXXct”	“POSIXct” “POSIXt”
6	Which of the following function gives the day of the week ?	weekdays	months	quarters	years	weekdays
7	What would be the output of the following code ? > p <- as.POSIXlt(x) > names(unclass(p)) > p$wday	1	2	3	0	1

8	What would be the output of the following code ? > x <- as.Date("2012-03-01") > y <- as.Date("2012-02-28") > x-y	Time difference of 3 days	Time difference of 2 days	Time difference of 1 day	Time difference of 4 days	Time difference of 2 days
9	Which of the following return a subset of the columns of a data frame ?	select	retrieve	get	hold	select
10	_____ extract a subset of rows from a data frame based on logical conditions.	rename	filter	set	subset	rename
11	_____ generate summary statistics of different variables in the data frame, possibly within strata	rename	summarize	set	subset	summarize
12	Point out the wrong statement :	The dplyr package was developed by Hadley Wickham of RStudio	The dplyr packageis an optimized and distilled version of his plyr package	The dplyr package provideS any “new” functionality to R	The dplyr package does not provide any “new” functionality to R	The dplyr package provideS any “new” functionality to R
13	_____ add new variables/columns or transform existing variables	mutate	add	apped	arrange	mutate
14	The _____ operator is used to connect multiple verb actions together into a pipeline	pipe	piper	start	end	pipe
15	The dplyr package can be installed from GitHub using the _____ package	dev	devtools	devtool	dtool	devtools
16	The dplyr package can be installed from CRAN using :	installall.packages(“dplyr”)	install.packages(“dplyr”)	installed.packages(“dplyr”)	installed.package(“dplyr”)	install.packages(“dplyr”)
17	Which of the following object is masked from ‘package:stats’ ?	difference	setdifference	union	filter	filter
18	The _____ function can be used to select columns of a data frame that you want to focus on.	filter	get	rename	select	select

19	Point out the correct statement :	You can also omit variables using the select() function by using the negative sign	The arrange() function also allows a special syntax that allows you to specify variable names based on patterns	Reordering rows of a data frame is normally easier to do in R	The rename() function is designed to make this process difficult.	You can also omit variables using the select() function by using the negative sign
20	_____ function is similar to the existing subset() function in R but is quite a bit faster.	rename	filter	set	subset	filter
21	Columns can be arranged in descending order too by using the special ____ operator.	asc()	desc()	descending()	subset	desc()
22	Point out the wrong statement :	Renaming a variable in a data frame in R is surprisingly hard to do	The mutate() function exists to compute transformations of variables in a data frame	mute() function, which does the same thing as mutate() but then drops all non-transformed variables	The rename() function is designed to make this process easier.	mute() function, which does the same thing as mutate() but then drops all non-transformed variables
23	The _____ function is used to generate summary statistics from the data frame within strata defined by a variable.	groupby()	group()	group_by()	arrange	group_by()
24	The _____ operator allows you to string operations in a left-to-right fashion.	%>%>	%>%	>%>%	>%>%>	%>%
25	There is an SQL interface for relational databases via the _____ package.	DIB	DB2	DBI	DB	DBI
26	dplyr can be integrated with the _____ package for large fast tables.	data.table	read.table	data.data	read.data	data.table
27	Which of the following function is similar to summarize ?	arrange_by()	group()	group_by()	arrange	group_by()

28	Which of the following is valid syntax for if else statement in R ?	if(<condition>) { ## do something } else { ## do something else }	if(<condition>) { ## do something } elseif { ## do something else }	if(<condition >) { ## do something } else if { ## do something else }	if(<condition >) { ## do something } elsif{ ## do something else }	if(<condition>) { ## do something } else { ## do something else }
29	Point out the correct statement :	Blocks are evaluated until a new line is entered after the closing brace	Single statements are evaluated when a new line is typed at the start of the syntactically complete statement	The if/else statement conditionally evaluates two statements	The jump statement conditionally evaluates two statements	The if/else statement conditionally evaluates two statements
30	Which of the following syntax is correct for while loop ?	while (statement1) statement2	while (statement1) else statement2	while (statement1) do statement2	while (statement1) else if statement2	while (statement1) statement2
31	_____ is used to break the execution of a loop.	next	skip	break	if	break
32	Which of the following statement can be used to explicitly control looping ?	if	while	break	next	break
33	Which of the following should be preferred for evaluation from list of alternatives ?	subsett	eval	switch	if	eval
34	_____ initiates an infinite loop right from the start.	never	repeat	break	set	repeat
35	Which of the following code snippet stops loop after 20 iterations ?	for(i **in** 1:100) { print(i) if(i>20){ break }}	for(i **in** 1:100) { print(i) if(i>19){ break }}	for(i **in** 1:100) { print(i) if(i<19){ break }}	for(i **in** 1:100) { print(i) if(i<20){ break }}	for(i **in** 1:100) { print(i) if(i>20){ break }}

36	Point out the wrong statement :	Statements cannot be grouped together using braces ‘{’ and ‘}’	Computation in R consists of sequentially evaluating statements	Computation in R consists of sequentially evaluating statements	Control structures in R allow you to control the flow of execution of a series of R expressions.	Statements cannot be grouped together using braces ‘{’ and ‘}’
37	_____ is used to skip an iteration of a loop.	group by	group	skip	next	next
38	R has _____ statements that provide explicit looping.	two	three	four	five	three
39	The syntax of the repeat loop is :	rep statement	repeat statement	repeat else	else statement	repeat statement
40	What will be the output of the following code ? > x <- 3 > switch(2, 2+2, mean(1:10), rnorm(5))	5	5.5	0	5.3	5.5
41	Point out the correct statement :	The next statement causes an exit from the innermost loop that is currently being executed	There are two statements that can be used to explicitly control looping	The break statement immediately causes control to return to the start of the loop	There are two statements that can be used to implicitly control looping	There are two statements that can be used to explicitly control looping
42	What will be the output of the following code ? > y <- "fruit" > switch(y, fruit = "banana", vegetable = "broccoli", "Neither")	“banana”	“Neither”	“broccoli”	"fruit"	“banana”
43	R has _____ basic indexing operators.	two	three	four	five	three
44	The syntax of the for loop is :	for ($name in vector) statement1	for loop(name in vector) statement1	for (name **in vector**) statement1	for loop ($name **in vector**) statement1	for (name **in vector**) statement1

45	What would be the output of the following code ? > x <- matrix(1:4, 2, 2) > y <- matrix(rep(10, 4), 2, 2) > x * y	[,1] [,2] [1,] 10 30 [2,] 20 40	[,1] [,2] [1,] 10 30 [2,] 30 40	[,1] [,2] [1,] 20 30 [2,] 20 40	[,1] [,2] [1,] 10 30 [2,] 30 40	[,1] [,2] [1,] 10 30 [2,] 20 40
46	What would be the output of the following code ? > x <- 1:4 > y <- 6:9 > x/y	0.1666667 0.2857143 0.4444444	0.1666667 0.2857143 0.3750000 0.4444444	0.2857143 0.3750000 0.4444444	0.2857143 0.3750000 0.1666667	0.1666667 0.2857143 0.3750000 0.4444444
47	What would be the output of the following code ? > x <- as.Date("1970-01-01") > x	“1970-01-01”	“1970-01-02”	“1970-02-01”	“1970-02-02”	“1970-01-01”
48	What would be the output of the following code ? > x <- as.Date("2012-01-01") > y <- strptime("9 Jan 2011 11:34:21", "%d %b %Y %H:%M:%S") > x-y	Time difference of 356.3095 days	Warning	NULL	Error	Warning
49	What would be the output of the following code ? > x <- as.POSIXct("2012-10-25 01:00:00") > y <- as.POSIXct("2012-10-25 06:00:00", tz = "GMT") > y-x	Time difference of 10 sec	Time difference of 1 sec	Time difference of 1 min	Time difference of 1 hour	Time difference of 1 hour
50	Which of the following code generate a uniform random number ?	x <- runif(1, 0, 10) if(x > 3) { y <- 10 } else { y <- 0 }	x <- run(1, 0, 10) if(x > 3) { y <- 10 } else { y <- 0 }	x <- random(1, 0, 10) if(x > 3) { y <- 10 } else { y <- 0 }	x <-runn(1, 0, 10) if(x > 3) { y <- 10 } else { y <- 0 }	x <- runif(1, 0, 10) if(x > 3) { y <- 10 } else { y <- 0 }
51	Point out the wrong statement :	for will execute a loop a fixed number of times	break will execute a loop while a condition is true	if and else tests a condition and acting on it	break will execute a loop while a condition is false	break will execute a loop while a condition is true
52	_____ initiates an infinite loop right from the start.	next	for	repeat	while	repeat
53	_____ is used to exit a loop immediately, regardless of what iteration the loop may be on.	next	break	repeat	while	break
54	____ loops begin by testing a condition.	next	break	repeat	while	while

55	The _____ function is commonly used in conjunction with for loops in order to generate an integer sequence based on the length of an object	seq()	seq_long()	seq_along()	seq_alo()	seq_along()
56	The _____ function is used to extract subsets of rows from a data frame.	arrange()	filter()	select()	mutate()	filter()
57	The _____ function is used to reorder rows of a data frame according to one of the variables/- columns	arrange()	filter()	select()	mutate()	arrange()
58	The _____ function is designed to make this process easier.	arrange()	rename()	select()	mutate()	rename()
59	The _____ function is used to generate summary statistics from the data frame within strata defined by a variable.	subset()	summarize()	group_by()	group()	group_by()
60	The _____ package provides a concise set of operations for managing data frames.	summarize	dlyr	dpl	dplyr	dplyr
61	Point out the correct statement :	Each row is an observation in tidy data	Each column is a variable in tidy data	Arranging your data in tidy way makes it easier to work	All of the move mentioned	All of the above mentioned
62	Which of the following is complementary to tidyr ?	geolr	dplyr	d3lr	dplya	dplyr
63	How many functions exist for tidying the data ?	one	two	three	five	three
64	Which of the following function takes multiple columns ?	spread()	gather()	separate()	mutate()	gather()
65	_____ uses regex groups instead of a splitting pattern or position.	spread()	gather()	separate()	arrange()	separate()
66	Which of the following function works similar to separate() ?	extract()	gather()	sep()	split()	extract()
67	Which of the following function is used to view the dataset in spreadsheet like format?	disp()	View()	seq()	display()	View()
68	“dplyr” is one of the most popular package used in R for manipulating data and it contains 5 core functions to handle data. Which of the following is not one of the core functions of dplyr package?	select()	filter()	arrange()	summary()	summary()

69	Which of the following commands will select the rows having “Alpha” values in “Column1” and value less than 50 in “Column4”? The dataframe is stored in a variable named table.	dplyr::filter(table,Column1==’Alpha’, Column4<50)	dplyr::filter(table,Column1==’Alpha’ & Column4<50)	both a and b	dplyr:filter(table,Column1==’Alpha’, Column4<50)	both a and b
70	Which of the following code will sort the dataframe based on “Column2” in ascending order and “Column3” in descending order?	dplyr::arrange(table,desc(Column3),Column2)	table[order(-Column3,Column2),]	both a and b	table(order(-Column3,Column2),)	both a and b
71	dplyr can be integrated with the _____ package for large fast tables	data.table	read.table	data.data	table.read	data.table
72	_____ extract a subset of rows from a data frame based on logical conditions.	rename	filter	set	subset	rename
73	The _____ operator allows you to string operations in a left-to-right fashion	%>%>	%>%	>%>%	%%>%%	%>%

UNIT-IV

Syllabus

Exploratory Data Analysis: Essential exploratory techniques for summarizing data, applied before formal modeling commences, eliminating or sharpening potential hypotheses about the world that can be addressed by the data, common multivariate statistical techniques used to visualize high-dimensional data.

What is Exploratory Data Analysis?

It is an approach to analyze data sets to summarize their main characteristics, often with visual methods. It is most useful while identifying outliers, trends and patterns. It was promoted by John Tukey to encourage statisticians to explore the data. Tukey also invented the term "bit", "software",and extended the "jackknife method".

Exploratory Data Analysis (EDA) is the first step in your data analysis process. Exploratory Data Analysis (EDA) is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to

- maximize insight into a data set;
- uncover underlying structure;
- extract important variables;
- detect outliers and anomalies;
- test underlying assumptions;
- develop parsimonious models; and
- Determine optimal factor settings.

Primary and Secondary Goals:

The primary goal of EDA is to maximize the analyst's insight into a data set and into the underlying structure of a data set, while providing all of the specific items that an analyst would want to extract from a data set, such as:

1. a good-fitting, parsimonious model
2. a list of outliers
3. a sense of robustness of conclusions
4. estimates for parameters
5. uncertainties for those estimates
6. a ranked list of important factors

7. conclusions as to whether individual factors are statistically significant
8. optimal settings

Data Analysis Approaches:

EDA is a data analysis approach. What other data analysis approaches exist and how does EDA differ from these other approaches?

Three popular data analysis approaches are:

- Classical
- Exploratory (EDA)
- Bayesian

Paradigms for Analysis Techniques:

These three approaches are similar in that they all start with a general science/engineering problem and all yield science/engineering conclusions. The difference is the sequence and focus of the intermediate steps.

For classical analysis, the sequence is

Problem => Data => Model => Analysis => Conclusions

For EDA, the sequence is

Problem => Data => Analysis => Model => Conclusions

For Bayesian, the sequence is

Problem => Data => Model => Prior Distribution => Analysis => Conclusions

Techniques:

Classical:

Classical techniques are generally quantitative in nature. They include ANOVA, t tests, chi-squared tests, and F tests.

Exploratory:

EDA Techniques are generally graphical. They include scatter plots, character plots, box plots, histograms, bihistograms, probability plots, residual plots, and mean plots.

Introduction to Exploratory Data Analysis:

To summarize main characteristics of data analysis in R, EDA is the only approach with the help of descriptive statistics and visual methods.

It is not a formal process that contains a strict set of rules. More than anything, EDA is a state of mind. During the initial phases of EDA, you should feel free to investigate every idea that occurs to you.

Why do we use exploratory graphs in data analysis?

- To understand data properties

- For finding patterns in data
- To suggest modeling strategies
- To "Debug" analyses

Terminologies in EDA

Variable

It is a quantity, quality, or property that you can measure.

Value

It is the state of a variable when you measure it. The value of a variable may change from measurement to measurement.

Observation

It is a set of measurements made under similar conditions. An observation will contain several values, each associated with a different variable. I'll sometimes refer to an observation as a data point.

Tabular data

Basically, it is a set of values, each associated with a variable and an observation. Tabular data is tidy if each value is placed in its own "cell", each variable in its own column and each observation in its own row.

Dataset

Following are the components of a data/dataset:

- Basically, a data set is represented as a matrix
- There is a row for each unit
- There is a column for each variable
- A unit is an object which we use to measure, such as a person, or a thing
- A variable is a characteristic of a unit. We use it toassign a number or a category

Dimensionality of Data Sets:

- **Univariate:** Measurement made on one variable per subject
- **Bivariate:** Measurement made on two variables per subject
- **Multivariate:** Measurement made on many variables per subject

Type of variables:

a. Qualitative: Variables take on values that are names or labels.

Ex. The color of a ball (e.g., red, green, blue) or the breed of a dog (e.g., collie, shepherd, terrier.

Types of Qualitative Variables:

i. Nominal: Basically, it displays graphical data — all orderings are equally meaningful.

Ex. a student's religion (Atheist, Christian, Muslim, Hindu, …) is nominal.

ii. Ordinal: A categorical variable whose categories can be meaningfully ordered is called ordinal.

Ex. a student's grade in an exam (A, B, C or Fail) is ordinal.

b. Quantitative: Variables that can measure on a numeric or quantitative scale.

Ex. Age, count of anything etc.

Types of Quantitative Variables:

i. Discrete: A discrete variable is one that cannot take on all values within the limits of the variable.

Ex. The number of children is a discrete numerical variable (a count). The variable cannot have the value 1.7

ii. Continuous: In this, the variable can take on any value between two specified values.

Ex. age of a human: 25 years, 10 months, 2 days, 5 hours

Techniques

Classical	Classical techniques are generally quantitative in nature. They include ANOVA, t tests, chi-squared tests, and F tests.
Exploratory	EDA techniques are generally graphical. They include scatter plots, character plots, box plots, histograms, bihistograms, probability plots, residual plots, and mean plots.

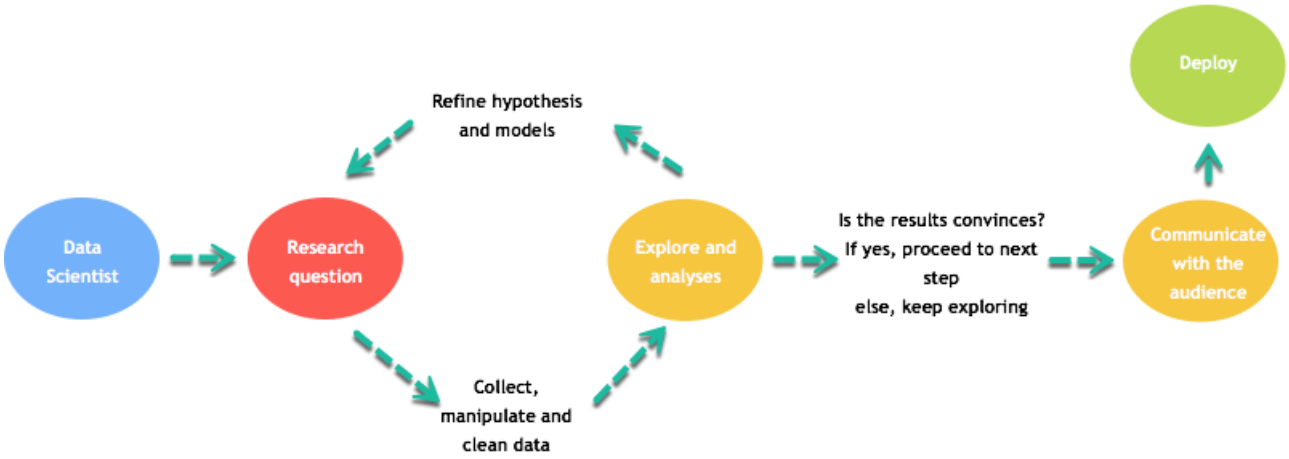
Graphical Data Analysis with R

Graphs are the third part of the process of data analysis. The first part is about **data extraction**, the second part deals with **cleaning and manipulating the data**. At last, the data scientist may need to **communicate his results graphically**.

The job of the data scientist can be reviewed in the following picture

- The first task of a data scientist is to define a research question. This research question depends on the objectives and goals of the project.
- After that, one of the most prominent tasks is the feature engineering. The data scientist needs to collect, manipulate and clean the data

- When this step is completed, he can start to explore the dataset. Sometimes, it is necessary to refine and change the original hypothesis due to a new discovery.



- When the **explanatory** analysis is achieved, the data scientist has to consider the capacity of the reader to **understand the underlying concepts and models**.
- His results should be presented in a format that all stakeholders can understand. One of the best methods to **communicate** the results is through a **graph**.
- Graphs are an incredible tool to simplify complex analysis.

Graphs are useful for non-numerical data, such as colors, flavors, brand names, and more. When numerical measures are difficult or impossible to compute, graphs play an important role.

Statistical computing is done with the aim to produce high-quality graphics.

Various types of plots drawn in R are:

- **Plots with single variables –** You can plot a graph for a single variable.
- **Plots with multiple variables –** You can plot graph with multiple variables
- **Special plots –** R has low and high-level graphics facilities.

Boxplot using R:

Boxplots can be created for individual variables or for variables by group.

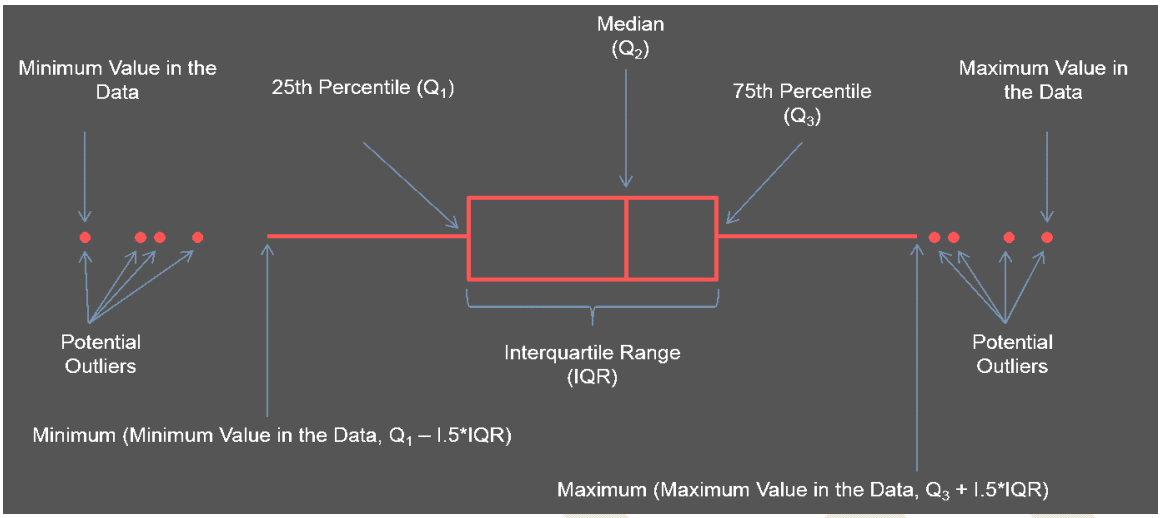
The format is boxplot(x, data=), where x is a formula and data= denotes the data frame providing the data.

A box plot gives a nice summary of one or several numeric variables. It is composed of several elements:

- The line that divides the box into 2 parts represents the median of the data. If the median is 10, it means that there are the same number of data points below and above 10.
- The end of the box shows the upper and lower quartiles. If the third quartile is 15, it means that 75% of the observation are lower than 15.
- The difference between Quartiles 1 and 3 is called the interquartile range (IQR)

- The extreme lines show the highest and lowest value excluding outliers.

Here is a diagram showing the box plot anatomy:



Boxplots are a measure of how well distributed is the data. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots. R Boxplot is created by using the **boxplot()** function.

Syntax:

The basic syntax to create a boxplot in R is:

boxplot(x,data,notch,varwidth,names,main)

Following is the description of the parameters used:

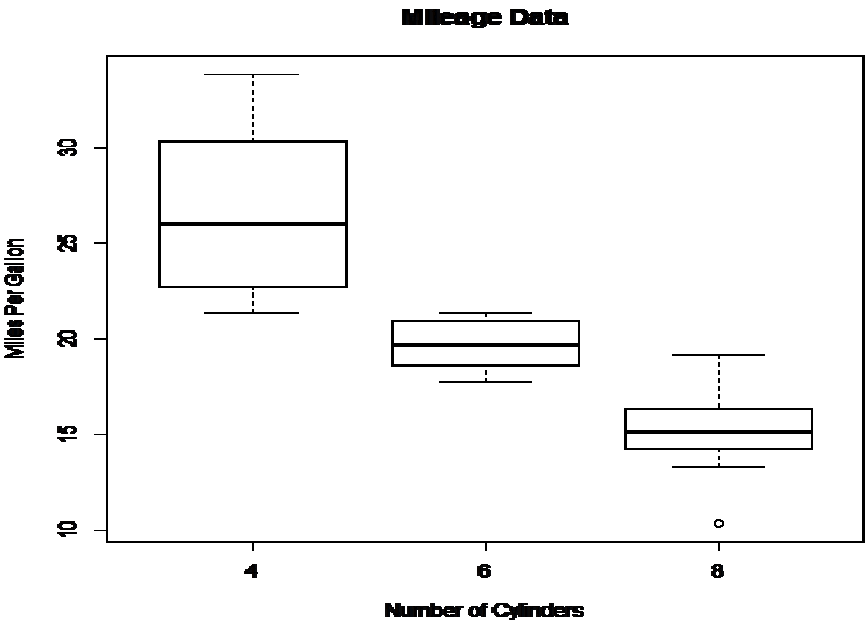
- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

Creating the Boxplot in R

The below script will create a boxplot graph for the relation between mpg(miles per gallon) and cyl (number of cylinders) from the well known mtcars data set.

boxplot(mpg ~ cyl, data=mtcars,xlab="Number of Cylinders",ylab="Miles Per Gallon",main="Mileage Data")

When we execute the above code, it produces the following result



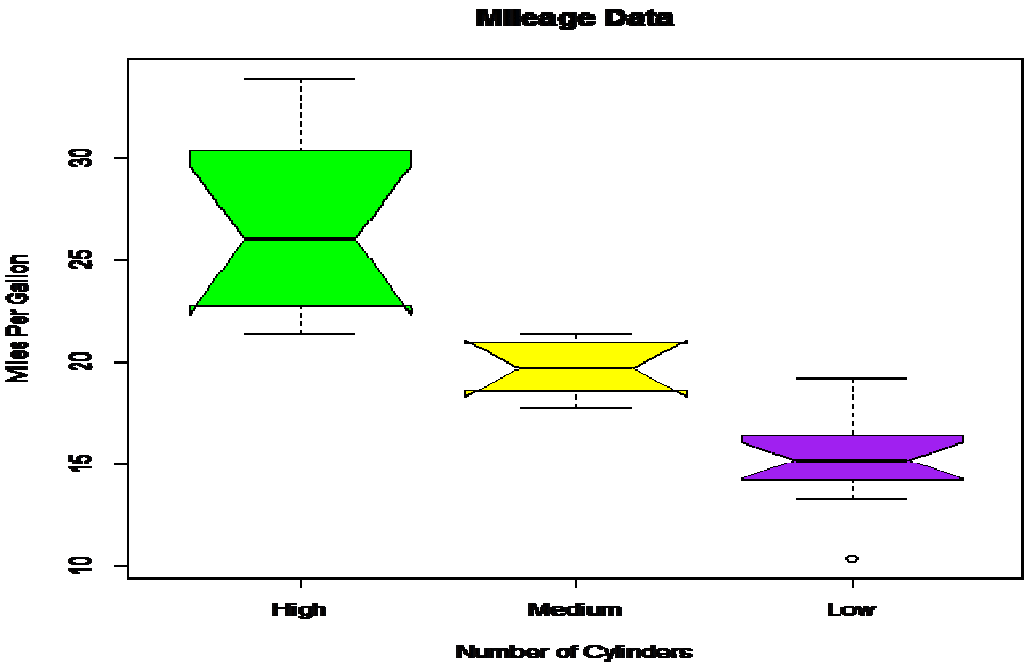
R Boxplot with Notch

We can draw boxplot with notch to find out how the medians of different data groups match with each other. The below script will create a boxplot graph with notch for each of the data group.

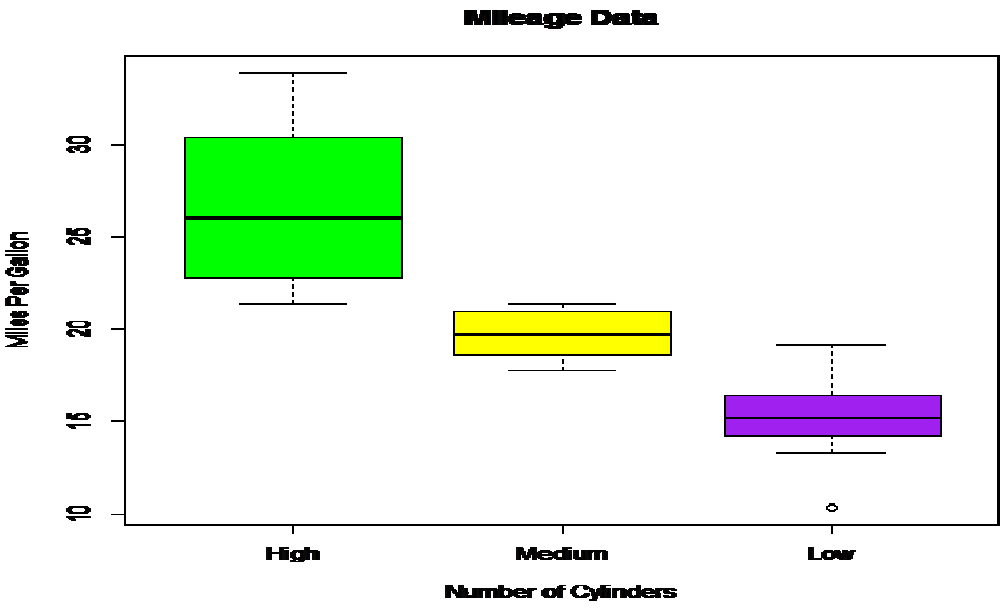
boxplot(mpg ~ cyl, data=mtcars,xlab="Number of Cylinders",ylab="Miles Per Gallon",

main="MileageData",notch=TRUE,col=c("green","yellow","purple"),names=c("High","Medium","Low"))

output of above script is



In the above script if we replace value of notch=FALSE Then the output will be



Graphics with ggplot2

The **ggplot2** package, created by Hadley Wickham, offers a powerful graphics language for creating elegant and complex plots. Its popularity in the **R** community has exploded in recent years. ... There is a helper function called **qplot**() (for quick plot) that can hide much of this complexity when creating standard graphs.

In ggplot2, a graph is composed of the following arguments:

- data
- aesthetic mapping
- geometric object
- statistical transformations
- scales
- coordinate system
- position adjustments
- faceting

The basic syntax of ggplot2 is:

ggplot(data, mapping=aes()) + geometric object

Arguments:

data: Dataset used to plot the graph

mapping: Control the x and y-axis

geometric object: The type of plot you want to show. The most common objects are:

- Point: `geom_point()`
- Bar: `geom_bar()`
- Line: `geom_line()`
- Histogram: `geom_histogram()`

The ggplot2 package, created by Hadley Wickham, offers a powerful graphics language for creating elegant and complex plots. Its popularity in the R community has exploded in recent years. Originally based on Leland Wilkinson's The Grammar of Graphics, ggplot2 allows you to create graphs that represent both univariate and multivariate numerical and categorical data in a straightforward manner. Grouping can be represented by color, symbol, size, and transparency. The creation of trellis plots (i.e., conditioning) is relatively simple.

qplot()

The **qplot()** function can be used to create the most common graph types. While it does not expose **ggplot**'s full power, it can create a very wide range of useful plots.

The format is:

qplot(*x*, *y*, data=, color=, shape=, size=, alpha=, geom=, method=, formula=, facets=, xlim=, ylim= xlab=, ylab=, main=, sub=)

where the options are:

option	**description**
alpha	Alpha transparency for overlapping elements expressed as a fraction between 0 (complete transparency) and 1 (complete opacity)
color, shape, size, fill	Associates the levels of variable with symbol color, shape, or size. For line plots, color associates levels of a variable with line color. For density and box plots, fill associates fill colors with a variable. Legends are drawn automatically.
data	Specifies a data frame
facets	Creates a trellis graph by specifying conditioning variables. Its value is expressed as *rowvar ~ colvar*. To create trellis graphs based on a single conditioning variable, use *rowvar~.* or *.~colvar*)
geom	Specifies the geometric objects that define the graph type. The geom option is expressed as a character vector with one or more entries. geom values include "point", "smooth", "boxplot", "line", "histogram", "density", "bar", and "jitter".
main, sub	Character vectors specifying the title and subtitle
method, formula	If geom="smooth", a loess fit line and confidence limits are added by default. When the number of observations is greater than 1,000, a more efficient smoothing algorithm is employed. Methods include "lm" for regression, "gam" for generalized additive models, and "rlm" for robust regression. The formula parameter gives the form of the fit. For example, to add simple linear regression lines, you'd specify geom="smooth", method="lm", formula=y~x. Changing the formula to y~poly(x,2) would produce a quadratic fit. Note that the formula uses the letters x and y, not the names of the variables. For method="gam", be sure to load the mgcv package. For method="rml", load the MASS package.
x, *y*	Specifies the variables placed on the horizontal and vertical axis. For univariate plots (for example, histograms), omit *y*
xlab, ylab	Character vectors specifying horizontal and vertical axis labels
xlim,ylim	Two-element numeric vectors giving the minimum and maximum values for the horizontal and vertical axes, respectively

Notes:

- At present, ggplot2 cannot be used to create 3D graphs or mosaic plots.
- Use **I(*value*)** to indicate a specific value. For example **size=z** makes the size of the plotted points or lines proporational to the values of a variable z. In contrast, **size=I(3)**sets each point or line to three times the default size.

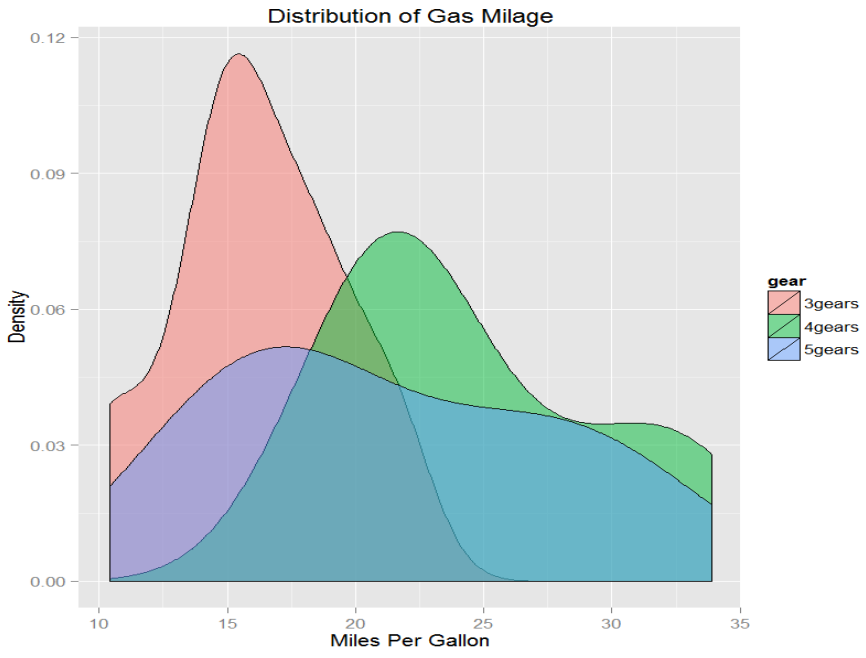
Here are some examples using automotive data (car mileage, weight, number of gears, number of cylinders, etc.) contained in the **mtcars** data frame.

 # ggplot2 examples

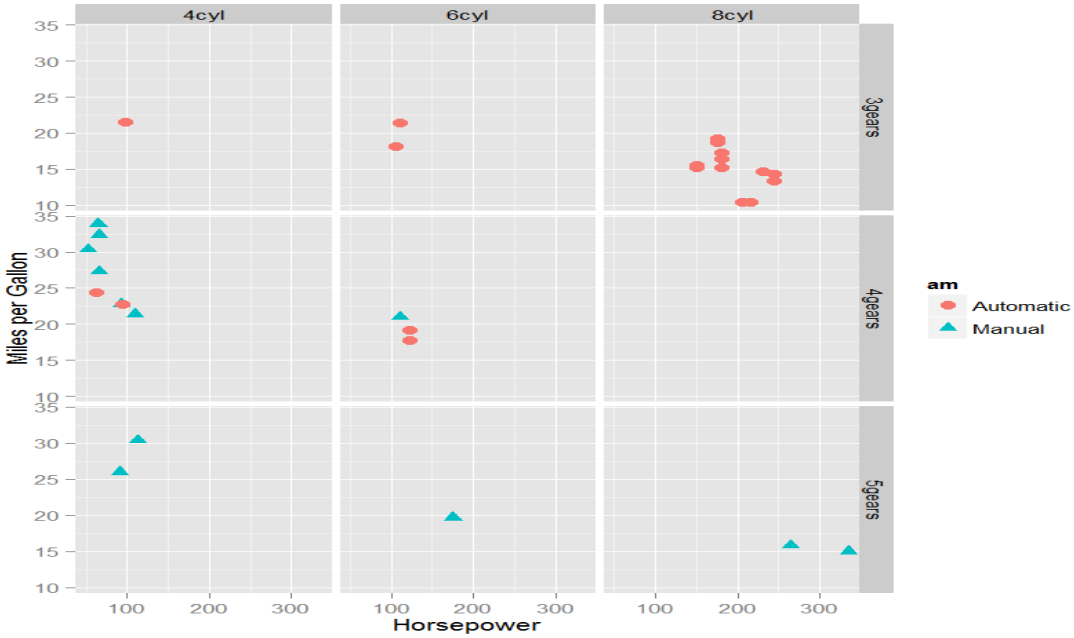
```
library(ggplot2)

# create factors with value labels
mtcars$gear <- factor(mtcars$gear,levels=c(3,4,5),   labels=c("3gears","4gears","5gears"))
mtcars$am <- factor(mtcars$am,levels=c(0,1),   labels=c("Automatic","Manual"))
mtcars$cyl <- factor(mtcars$cyl,levels=c(4,6,8),   labels=c("4cyl","6cyl","8cyl"))

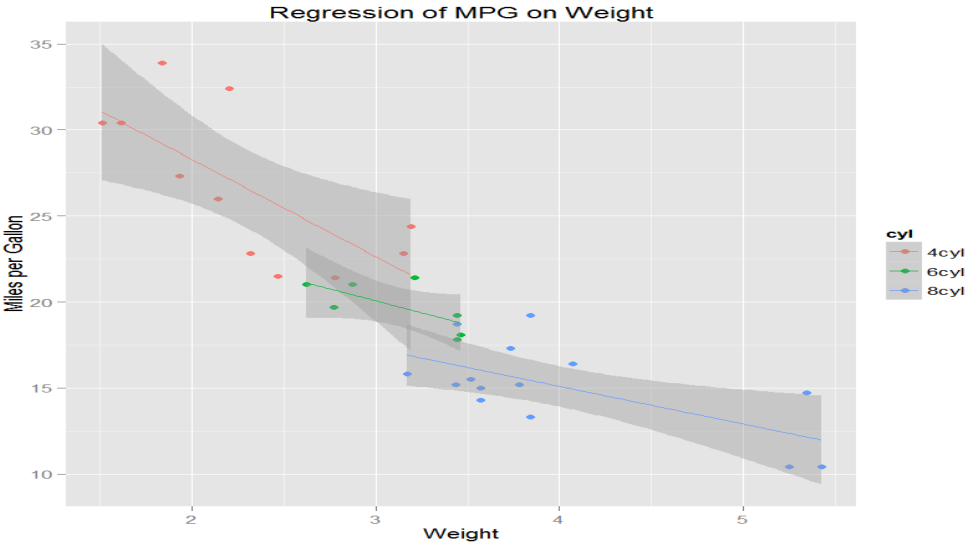
# Kernel density plots for mpg
# grouped by number of gears (indicated by color)
qplot(mpg, data=mtcars, geom="density", fill=gear, alpha=I(.5), main="Distribution of Gas Milage",
xlab="Miles Per Gallon",   ylab="Density")
```



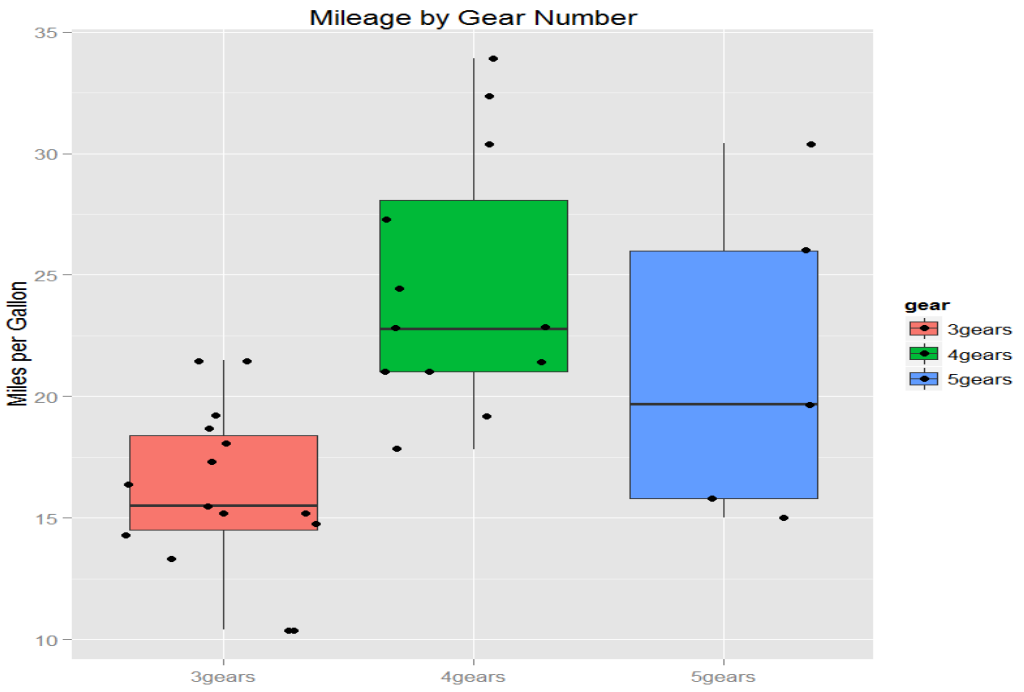
# Scatterplot of mpg vs. hp for each combination of gears and cylinders
# in each facet, transmittion type is represented by shape and color
qplot(hp, mpg, data=mtcars, shape=am, color=am, facets=gear~cyl, size=I(3), xlab="Horsepower", ylab="Miles per Gallon")



# Separate regressions of mpg on weight for each number of cylinders
qplot(wt, mpg, data=mtcars, geom=c("point", "smooth"), method="lm", formula=y~x, color=cyl, main="Regression of MPG on Weight", xlab="Weight", ylab="Miles per Gallon")



```
# Boxplots of mpg by number of gears
# observations (points) are overlayed and jittered
qplot(gear, mpg, data=mtcars, geom=c("boxplot", "jitter"),   fill=gear, main="Mileage by Gear Number",
  xlab="", ylab="Miles per Gallon")
```



Scatterplot

You start by plotting a scatterplot of the mpg variable and drat variable.

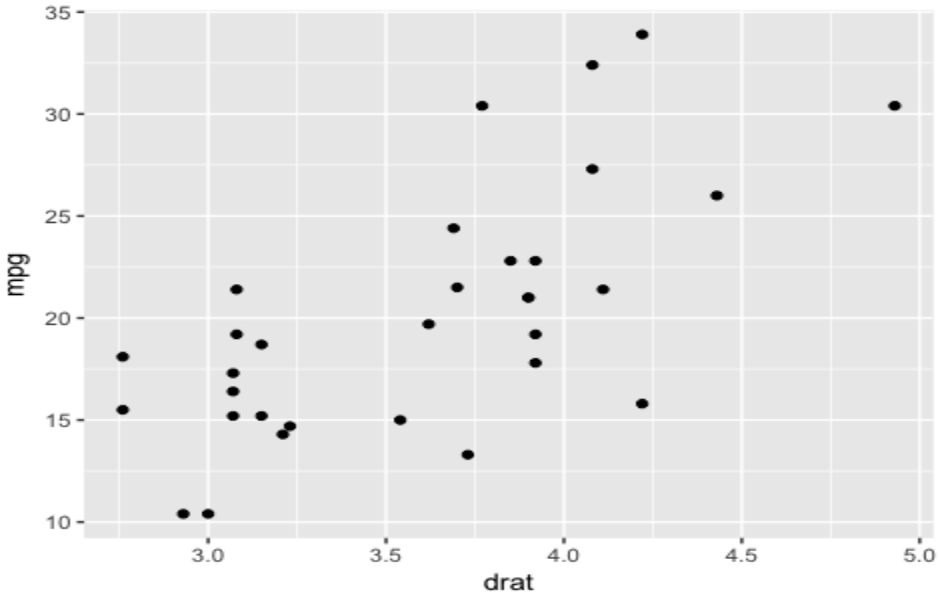
Basic scatter plot

```
library(ggplot2)
ggplot(mtcars, aes(x = drat, y = mpg)) +  geom_point()
```

Code Explanation

- You first pass the dataset mtcars to ggplot.
- Inside the aes() argument, you add the x-axis and y-axis.
- The + sign means you want R to keep reading the code. It makes the code more readable by breaking it.
- Use geom_point() for the geometric object.

Output:



Scatter plot with groups

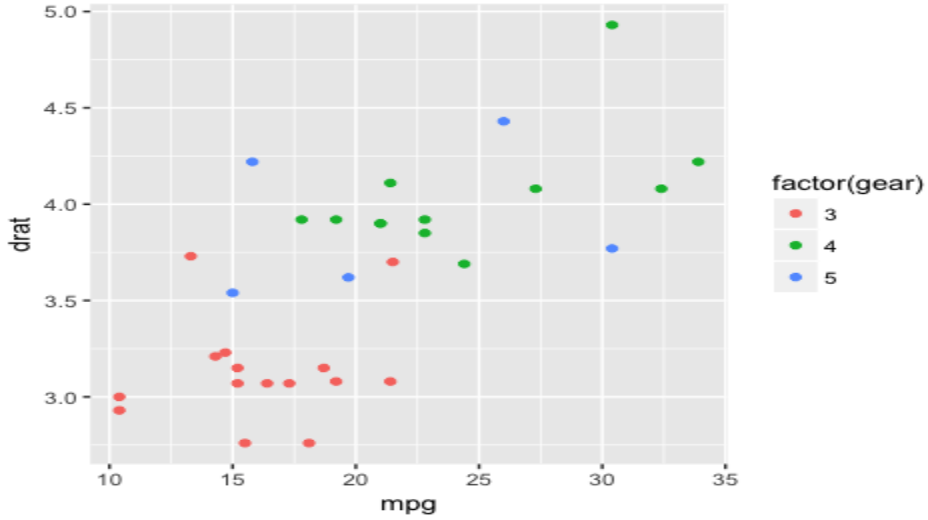
Sometimes, it can be interesting to distinguish the values by a group of data (i.e. factor level data).

ggplot(mtcars, aes(x = mpg, y = drat)) + geom_point(aes(color = factor(gear)))

Code Explanation

- The aes() inside the geom_point() controls the color of the group. The group should be a factor variable. Thus, you convert the variable gear in a factor.
- Altogether, you have the code aes(color = factor(gear)) that change the color of the dots.

Output:



Change axis

Rescale the data is a big part of the data scientist job. In rare occasion data comes in a nice bell shape. One solution to make your data less sensitive to outliers is to rescale them.

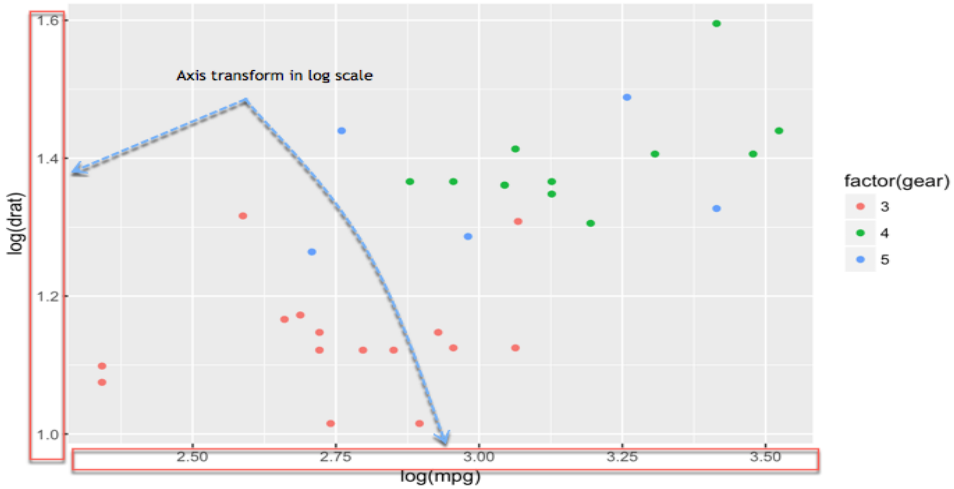
ggplot(mtcars, aes(x = log(mpg), y = log(drat))) + geom_point(aes(color = factor(gear)))

Code Explanation

- You transform the x and y variables in log() directly inside the aes() mapping.

Note that any other transformation can be applied such as standardization or normalization.

Output:



Scatter plot with fitted values

You can add another level of information to the graph. You can plot the fitted value of a linear regression.

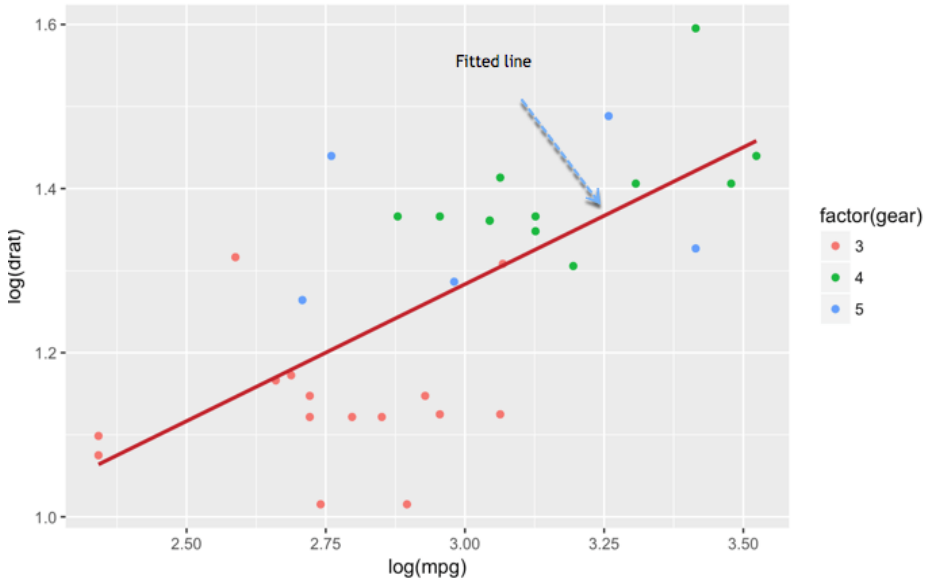
my_graph <- ggplot(mtcars, aes(x = log(mpg), y = log(drat))) + geom_point(aes(color = factor(gear))) + stat_smooth(method = "lm", col = "#C42126", se = FALSE, size = 1)

> my_graph

Code Explanation

- graph: You store your graph into the variable graph. It is helpful for further use or avoid too complex line of codes
- The argument stat_smooth() controls for the smoothing method
- method = "lm": Linear regression
- col = "#C42126": Code for the red color of the line
- se = FALSE: Don't display the standard error
- size = 1: the size of the line is 1

Output:



Note that other smoothing methods are available

- glm
- gam
- loess: default value
- rim

Add information to the graph

So far, we haven't added information in the graphs. Graphs need to be informative. The reader should see the story behind the data analysis just by looking at the graph without referring additional documentation. Hence, graphs need good labels. You can add labels with labs()function.

The basic syntax for lab() is :

lab(title = "Hello Guru99")

argument:

- title: Control the title. It is possible to change or add title with:

- subtitle: Add subtitle below title

- caption: Add caption below the graph

- x: rename x-axis

- y: rename y-axis

Example: lab(title = "Hello Guru99", subtitle = "My first plot")

Add a title

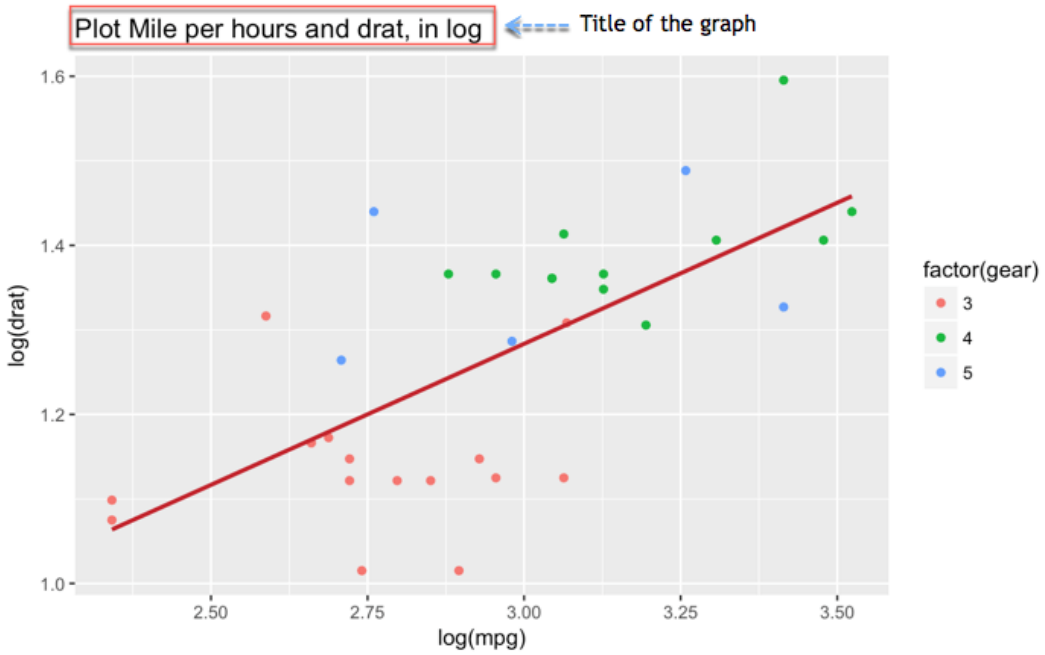
One mandatory information to add is obviously a title.

my_graph + labs(title = "Plot Mile per hours and drat, in log")

Code Explanation

- my_graph: You use the graph you stored. It avoids rewriting all the codes each time you add new information to the graph.
- You wrap the title inside the lab().
- Code for the red color of the line
- se = FALSE: Don't display the standard error
- size = 1: the size of the line is 1

Output:



Add a subtitle

Two additional detail can make your graph more explicit. You are talking about the subtitle and the caption. The subtitle goes right below the title. The caption can inform about who did the computation and the source of the data.

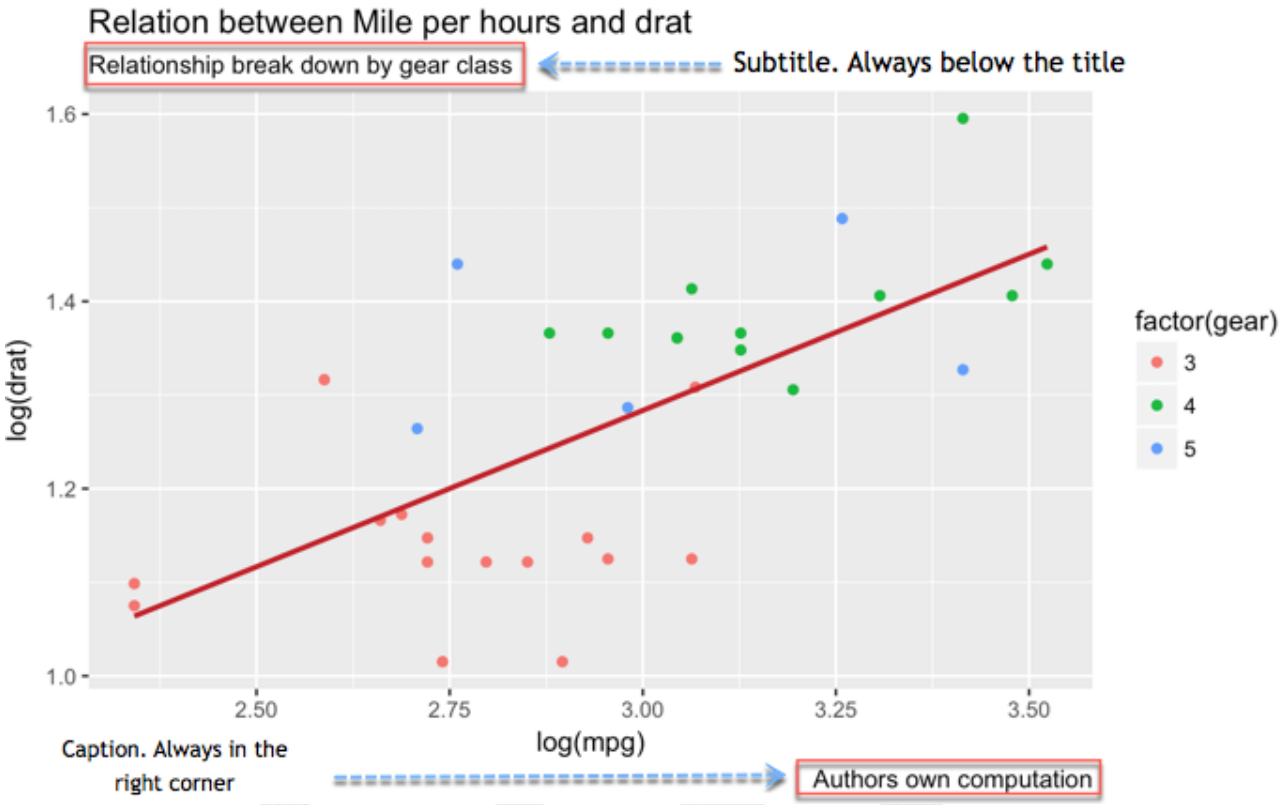
```
my_graph +
  labs(
    title =
    "Relation between Mile per hours and drat",
    subtitle =
    "Relationship break down by gear class",
    caption = "Authors own computation"
  )
```

Code Explanation

- Inside the lab(), you added:
  - title = "Relation between Mile per hours and drat": Add title
  - subtitle = "Relationship break down by gear class": Add subtitle
  - caption = "Authors own computation: Add caption
  - You separate each new information with a comma, ,

- Note that you break the lines of code. It is not compulsory, and it only helps to read the code more easily

Output:



Rename x-axis and y-axis

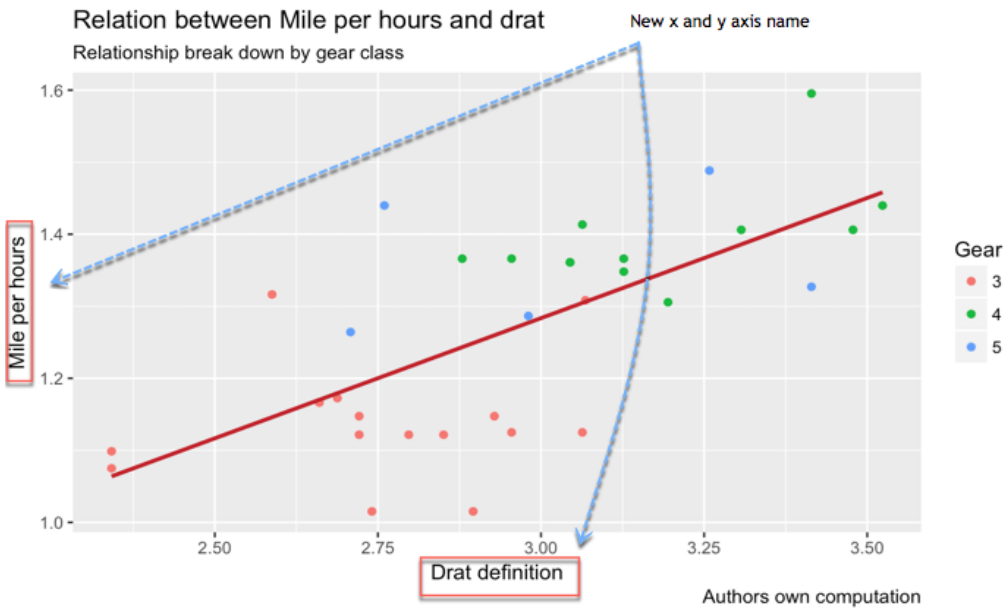
Variables itself in the dataset might not always be explicit or by convention use the _ when there are multiple words (i.e. GDP_CAP). You don't want such name appear in your graph. It is important to change the name or add more details, like the units.

```
my_graph +
  labs(
    x = "Drat definition",
    y = "Mile per hours",
    color = "Gear",
    title = "Relation between Mile per hours and drat",
    subtitle = "Relationship break down by gear class",
    caption = "Authors own computation"
  )
```

Code Explanation

- Inside the lab(), you added:
  - x = "Drat definition": Change the name of x-axis
  - y = "Mile per hours": Change the name of y-axis

Output:



Control the scales

You can control the scale of the axis.

The function seq() is convenient when you need to create a sequence of number. The basic syntax is:

seq(begin, last, by = x)

arguments:

- begin: First number of the sequence

- last: Last number of the sequence

- by= x: The step. For instance, if x is 2, the code adds 2 to `begin-1` until it reaches `last`

For instance, if you want to create a range from 0 to 12 with a step of 3, you will have four numbers, 0 4 8 12

seq(0, 12,4)

Output:

[1] 0 4 8 12

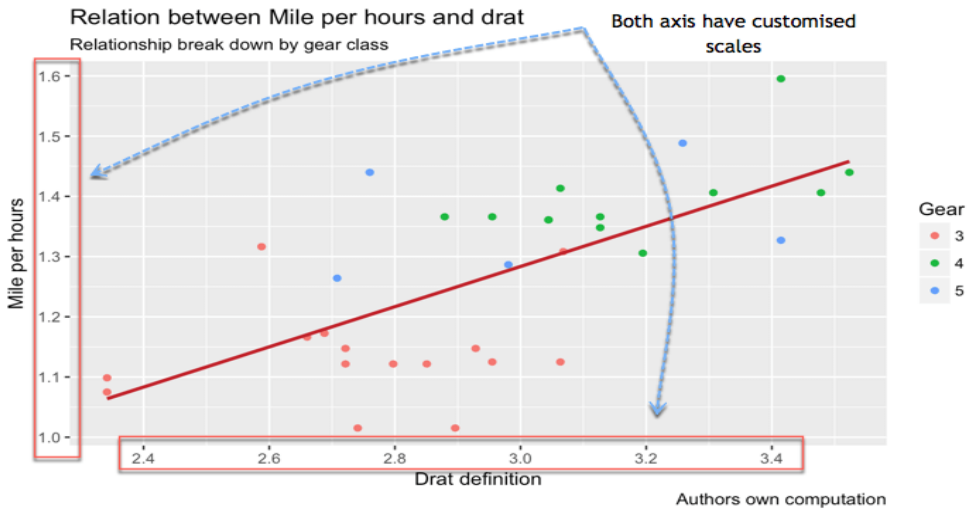
You can control the scale of the x-axis and y-axis as below

```
my_graph +
  scale_x_continuous(breaks = seq(1, 3.6, by = 0.2)) +
  scale_y_continuous(breaks = seq(1, 1.6, by = 0.1)) +
  labs(
    x = "Drat definition",
    y = "Mile per hours",
    color = "Gear",
    title = "Relation between Mile per hours and drat",
    subtitle = "Relationship break down by gear class",
    caption = "Authors own computation"
  )
```

Code Explanation

- The function scale_y_continuous() controls the **y-axis**
- The function scale_x_continuous() controls the **x-axis**.
- The parameter breaks controls the split of the axis. You can manually add the sequence of number or use the seq()function:
  - seq(1, 3.6, by = 0.2): Create six numbers from 2.4 to 3.4 with a step of 3
  - seq(1, 1.6, by = 0.1): Create seven numbers from 1 to 1.6 with a step of 1

Output:



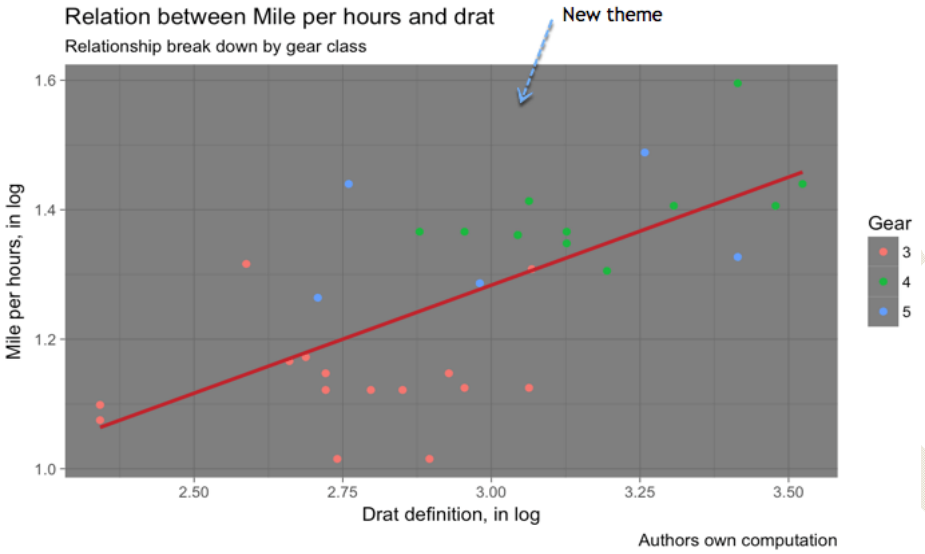
Theme

Finally, R allows us to customize out plot with different themes. The library ggplot2 includes eights themes:

- theme_bw()
- theme_light()
- theme_classis()
- theme_linedraw()
- theme_dark()
- theme_minimal()
- theme_gray()
- theme_void()

```
my_graph +
  theme_dark() +
  labs(
    x = "Drat definition, in log",
    y = "Mile per hours, in log",
    color = "Gear",
    title = "Relation between Mile per hours and drat",
    subtitle = "Relationship break down by gear class",
    caption = "Authors own computation"
  )
```

Output:



Save Plots

After all these steps, it is time to save and share your graph. You add ggsave('NAME OF THE FILE) right after you plot the graph and it will be stored on the hard drive.

The graph is saved in the working directory. To check the working directory, you can run this code:

directory <-getwd()

> directory

ggsave("my_fantastic_plot.png")

POSSIBLE QUESTIONS

2 MARKS

1. What is Exploratory Data Analysis?
2. Mention the Data Analysis Approaches.
3. What is a Classical technique?
4. DefineVariable.
5. What is Value?
6. What is Observation?
7. What is a Dataset?
8. Name Dimensionality of Data Sets in R.
9. What is the Type of variables?

6 MARKS

1. Give a detailed description about EDA.
2. Explain the Scatter plot with example.
3. Elaborates the ggplot2 graph with example.
4. Illustrate Box plot with example.s

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

UNIT - IV

Sno	Questions	Opt1	Opt2	Opt3	Opt4	Answer
1	Which of the following is apply function in R ?	apply()	tapply()	fapply()	sapply()	tapply()
2	Functions are defined using the _____ directive and are stored as R objects	function()	funct()	functions()	func()	function()
3	Point out the wrong statement :	Functions in R are “second class objects”	The writing of a function allows a developer to create an interface to the code, that is explicitly specified with a set of parameters	Functions provides an abstraction of the code to potential users	Functions in R are “first class objects”	Functions in R are “second class objects”
4	What will be the output of the following code ? > f <- function() { + ## This is an empty function + } > class(f)	“data"	“procedure”	“function”	“class”	“function”
5	The _____ function returns a list of all the formal arguments of a function	formals()	funct()	formal()	function()	formals()

6	Point out the wrong statement :	Functions can be nested, so that you can define a function inside of another function	The value returned by the call to function is not a function	A formal argument can be a symbol, a statement of the form 'symbol = expression', or the special formal argument	The first component of the function declaration is the keyword function	A formal argument can be a symbol, a statement of the form 'symbol = expression', or the special formal argument
7	You can check to see whether an R object is NULL with the _____ function.	is.null()	is.nullobj()	null()	is.obj()	is.null()
8	Which of the following code will print NULL ?	> args(pastebin)	> args(paste)	> args(pastebin)	> argc(pastebin)	> args(paste)
9	What will be the output of following code snippet ? > paste("a", "b", sep = ":")	"a+b"	"a=b"	"a:b"	"a-b"	"a:b"
10	What will be the output of following code ? > f <- function(a, b) { + print(a) + print(b) + } > f(45)	32	42	52	45	45
11	_____ is an indication that a fatal problem has occurred and execution of the function stops	message	error	warning	stop	error
12	What would be the value of following expression ? log(-1)	0	Null	Warning in log(-1): NaNs produced	1	Warning in log(-1): NaNs produced
13	Warnings are generated by the _____ function	warning()	error()	run()	runif()	warning()

14	Point out the correct statement :	POSIX represents a portable operating system interface, primarily for UNIX systems	There are different levels of indication that can be used, ranging from mere notification to fatal error	The default input format for POSIX dates consists of the month, followed by the year and day, separated by slashes or dashes	Dates are not stored in the POSIX format are date/time values	POSIX represents a portable operating system interface, primarily for UNIX systems
15	To get the current date, the _____ function will return a Date object which can be converted to a different class if necessary.	Sys.Time	Sys.Date	Sys.DateTime	Sys.TimeDate	Sys.Date
16	Which of the followin code represents internal representation of a Date object ?	class(as.Date(“1970-01-02”))	classint(as.Date(“1970-01-02”))	unclass(as.Date(“1970-01-02”))	unclassint(as.Date(“1970-01-02”))	unclass(as.Date(“1970-01-02”))
17	What will be the output of following code snippet ? > lm <- function(x) { x * x } > lm	function(x) { x * x }	func(x) { x * x }	function(x) { x / x }	funct(x) { x / x }	function(x) { x * x }
18	Point out the correct statement :	The search list can be found by using the searchlist() function	The search list can be found by using the search() function	The global environment or the user’s workspace is always the second element of the search list	The search can be found by using the searchlt() function	The search list can be found by using the search() function
19	A function, together with an environment, makes up what is called a _____ closure.	formal	function	reflective	unformal	function

20	R uses _____ scoping or static scoping.	reflective	transitive	lexical	formal	lexical
21	The only environment without a parent is the _____ environment.	full	half	null	empty	empty
22	The _____ for R are the main feature that make it different from the original S language	scoping rules	closure rules	environment rules	lexical rules	scoping rules
23	The _____ function is a kind of “constructor function” that can be used to construct other functions.	make.pow()	make.power()	keep.power()	keep.pow()	make.power()
24	What will be the output of following code ? > g <- function(x) { + a <- 3 + x+a+y + ## 'y' is a free variable + } > g(2)	9	42	8	Error	Error
25	_____ functions can be “built which contain all of the necessary data for evaluating the function	Objective	reflective	Nested	lexical	Objective
26	_____ require you to pass a function whose argument is a vector of parameters (	optimize()	optimise()	opt()	oplt()	opt()
27	The _____ function is used to plot negative likelihood.	plot()	graph()	graph.plot()	plot.graph()	plot()
28	_____ loop over a list and evaluate a function on each element	apply()	lapply()	sapply()	mapply()	apply()

29	Point out the wrong statement :	Multi-line expressions with curly braces are just not that easy to sort through when working on the command line	lappy() loops over a list, iterating over each element in that list	lapply() does not always returns a list	lapply() always returns a list, regardless of the class of the input.	lapply() does not always returns a list
30	_____ function is same as lapply in R	apply()	lapply()	sapply()	mapply()	sapply()
31	Which of the following is multivariate version of lapply ?	apply()	lapply()	sapply()	mapply()	mapply()
32	Point out the correct statement :	lapply() takes elements of the list and passes them as the first argument of the function you are applying	You can use lapply() to evaluate a function multiple times each with a different argument	Functions that you pass to lapply() may have other arguments	The lapply() function and its friends make heavy use of anonymous functions.	The lapply() function and its friends make heavy use of anonymous functions.
33	_____ applies a function over the margins of an array.	apply()	lapply()	sapply()	mapply()	apply()
34	_____ is used to apply a function over subsets of a vector.	apply()	lapply()	tapply()	mapply()	tapply()
35	lappy functions takes _____ arguments in R language.	two	three	four	five	four

36	Point out the wrong statement :	The sapply() function behaves similarly to lapply()	With multiple factors and many levels, creating an interaction can result in many levels that are empty	apply() can be thought of as a combination of split() and sapply() for vectors only	tapply() can be thought of as a combination of split() and sapply() for vectors only.	apply() can be thought of as a combination of split() and sapply() for vectors only
37	The _____ function takes a vector or other objects and splits it into groups determined by a factor or list of factors.	apply()	lsplit()	split()	mapply()	split()
38	What will be the output of the following code ? > nLL <- make.NegLogLik(normals, c(1, FALSE)) > optimize(nLL, c(1e-6, 10))$minimum	1.217775	1.800596	3.73424	empty	1.800596
39	Point out the correct statement :	An environment is a collection of (symbol, value) pairs, i.e. x is a symbol and 3.14 might be its value	If the value of a symbol is not found in the environment in which a function was defined, then the search is continued in the child environment	After the top-level environment, the search continues down the search list until we hit the parent environment	Every environment has a parent environment and it is not possible for an environment to have multiple “children”.	An environment is a collection of (symbol, value) pairs, i.e. x is a symbol and 3.14 might be its value

40	Point out the wrong statement :	Dynamic scoping turns out to be particularly useful for simplifying statistical computations	Lexical scoping turns out to be particularly useful for simplifying statistical computations	The scoping rules of a language determine how values are assigned to free variables	Free variables are not formal arguments and are not local variables	Dynamic scoping turns out to be particularly useful for simplifying statistical computations
41	What would be the output of the following code ? > printmessage <- function(x) { + if(x > 0) + print("x is greater than zero") + else + print("x is less than or equal to zero") + invisible(x) + } > printmessage(NA)	Error	Warning	Messages	Data	Error
42	Arguments to functions are evaluated _____ , so they are evaluated only as needed in the body of the function.	completely	lazily	directly	inversely	lazily
43	In R the calling environment is known as the _____	data frame	child fram	parent frame	called frame	parent frame
44	_____ turns out to be particularly useful for simplifying statistical computations	scoping rules	Lexical scoping	dynamic scoping	scoping	Lexical scoping
45	Optimization routines in R like _____, _____ and _____ require you to pass a function whose argument is a vector of parameters	opti(), lm(), and optimize()	opt(), nm(), and optimi()	optim(), nlm(), and optimize()	optim(), lmn(), and optimize()	optim(), nlm(), and optimize()
46	Optimization functions in R _____ functions, so you need to use the negative loglikelihood.	minimize	maximize	calling	return	minimize

47	The mapply() function can be use to automatically _____ a function	minimize	maximize	vectorize	calling	vectorize
48	The _____ function can be used to divide an R object in to subsets determined by another variable which can subsequently be looped over using loop functions.	apply()	lsplit()	split()	mapply()	split()
49	_____ expressions with curly braces are just not that easy to sort through when working on the command line	looping	Multi-line	lexical	Single-line	Multi-line
50	we are passing the _____ function as an argument to the lapply() function.	mode()	median()	mean()	split()	mean()
51	The lapply() function and its friends make heavy use of _____ functions.	calling	unanonymous	anonymous	member	anonymous
52	What will be the output of the following code ? > f <- function() { + ## This is an empty function + } > f()	0	No result	NULL	Error	NULL
53	Which of the following code will print “Hello, world!” ?	> f <- function() { cat("Hello, world!\n") } > f()	> f <- function() { cat("Hello, World!\n") } > f()	> f <- function() { cat("Hello world!\n") } > f()	> f <- function() { cat("hello World!\n") } > f()	> f <- function() { cat("Hello, world!\n") } > f()

54	What will be the output of following code ? > f <- function(num) {
	+ for(i in seq_len(num)) {
	+ cat("Hello, world!\n")
	+ }
	+ }
	> f(3)
		Hello, world!
		Hello, world!
			Hello, world!
			Hello, world!
			Hello, world!
				Hello, world!
				Hello, world!
				Hello, world!
				Hello, world!
					Hello, world!	Hello, world!
						Hello, world!
						Hello, world!

55	What will be the output of the following code ? > f <- function(num = 1) {
	+ hello <- "Hello, world!\n"
	+ for(i in seq_len(num)) {
	+ cat(hello)
	+ }
	+ chars <- nchar(hello) * num
	+ chars
	+ }
	> f()
		Hello, world!
		[1] 14
			Hello, world!
			[1] 15
				Hello, world!
				[1] 16
					Hello, world!
					[1] 17
						Hello, world!
						[1] 14

56	What will be the output of following code ? > f <- function(a, b) { + a^2 + } > f(2)	4	3	2	1	4
57	What will be the output of following code ? > f <- function(a, b) { + print(a) + print(b) + } > f(45)	32	42	52	45	45
58	What would be the output of the following code ? > p <- as.POSIXlt(x) > names(unclass(p)) > p$wday	1	2	3	4	1
59	_____ will not simplify the result and will return a list	apply()	lapply()	tapply()	mapply()	tapply()
60	_____ keeps track of the function call stack at regularly sampled intervals and tabulates how much time is spent inside each function	summaryRprof()	Rprof()	system.time()	prof()	Rprof()
61	_____ produces bivariate scatterplots or time-series plots.	xyplot	dotplot	barchart	bwplot	xyplot

62	_____ produces one-dimensional scatterplots.	xyplot	stripplot	barchart	bwplot	stripplot
63	What is ggplot2 an implementation of?	the Grammar of Graphics developed by Leland Wilkinson	3D visualization system	the S language originally developed by Bell Labs	the base plotting system in R	the Grammar of Graphics developed by Leland Wilkinson
64	_____ is used to create a plot to illustrate patterns of missing values.	ggmissplot	ggmissing	ggfluctuation	all the above mentioned	ggmissing
65	Which of the following is lattice command for producing a scatterplot ?	plot()	lm()	xyplot()	barplot()	xyplot()

UNIT-V

SYLLABUS

Reproducible Research: Concepts and tools behind reporting modern data analyses in a reproducible manner, To write a document using R markdown, integrate live R code into a literate statistical program, compile R markdown documents using knitr and related tools, and organize a data analysis so that it is reproducible and aCSUessible to others.

What is **Reproducible research**?

Reproducible research is the idea that data analyses, and more generally, scientific claims, are published with their data and software code so that others may verify the findings and build upon them. ... We also cover structuring and organizing a data analysis to help make it more **reproducible**.

If someone was to carry out the test again using the same method and got the same result, the data is said to be **repeatable**. If a measurement was taken using a **different** method and got the same result, the result is referred to as **reproducible**.

A study can be truly reproducible when it satisfies at least the following three criteria.

– All methods are fully reported.

– All data and files used for the analysis are (publicly) available.

– The process of analyzing raw data is well reported and preserved.

Therefore:

Same data + Same script = Same results

Basic definitions

1. Analysis = software + data + environment + invocations
2. Reproducible analysis = analysis that can be carried out by independent parties
3. Extensible analysis = analysis supporting independent variations
4. Study = Design + implementation + analysis
5. Replicable study = A study that, when executed by independent parties, produces statistically compatible interpretations

Replication

- The ultimate standard for strengthening scientific evidence is replication of findings and conducting studies with independent
  - Investigators
  - Data
  - Analytical methods
  - Laboratories
  - Instruments
- Replication is particularly important in studies that can impact broad policy or regulatory decisions

What's Wrong with Replication?

- Some studies cannot be replicated
  - No time, opportunistic
  - No money
  - Unique
- Reproducible Research: Make analytic data and code available so that others may reproduce findings

Why Do We Need Reproducible Research?

- New technologies increasing data collection throughput; data are more complex and extremely high dimensional
- Existing databases can be merged into new "megadatabases"
- Computing power is greatly increased, allowing more sophisticated analyses
- For every field "X" there is a field "Computational X"

Research Pipeline

The basic issue is when you read a description of a data analysis, such as in an article or a technical report, for the most part, what you get is the report and nothing else. Of course, everyone knows that behind the scenes there's a lot that went into this report and that's what I call the data science pipeline.

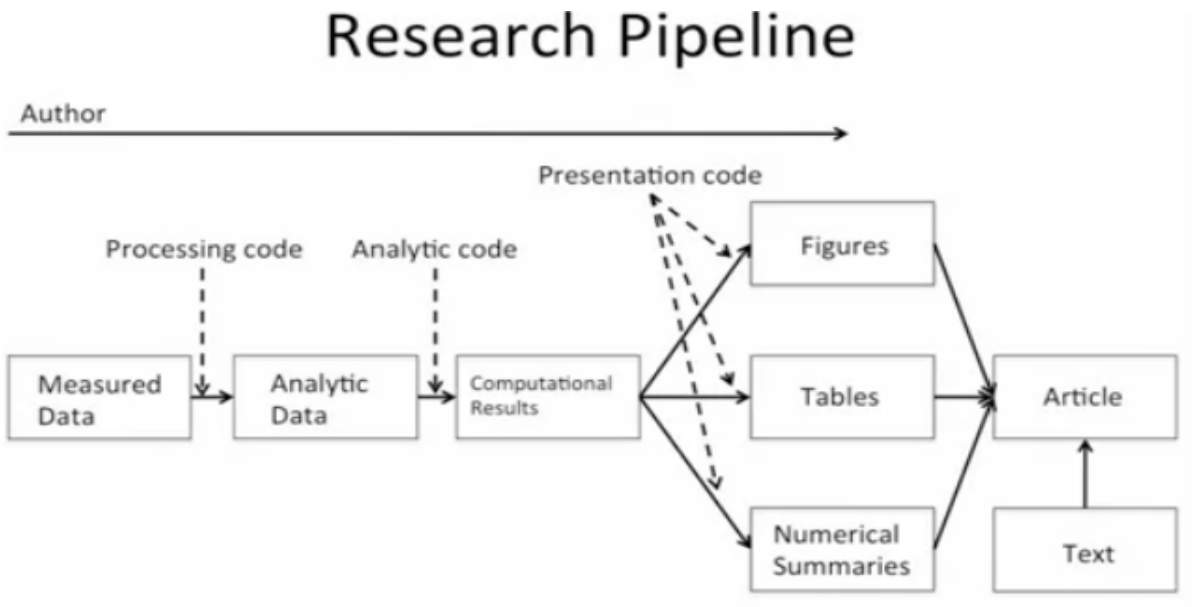


Fig:1 The Data Science Pipeline

In this pipeline, there are two "actors": the author of the report/article and the reader. On the left side, the author is going from left to right along this pipeline. The reader is going from right to left. If you're the reader you read the article and you want to know more about what happened: Where is the data? What was used here? The basic idea behind reproducibility is to focus on the elements in the blue blox: the analytic data and the computational results. With reproducibility the goal is to allow the author of a report and the reader of that report to "meet in the middle".

Elements of Reproducibility:

What do we need for reproducibility? There's a variety of always to talk about this, but one basic definition that we've come up with is that there are four things that are required to make results reproducible:

1. Analytic data:

The data that were used for the analysis that was presented should be available for others to access. This is different from the raw data because very often in a data analysis the raw data are not all used for the analysis, but rather some subset is used. It may be interesting to see the raw data but impractical to actually have it. Analytic data is key to examining the data analysis.

2. Analytic code:

The analytic code is the code that was applied to the analytic data to produce the key results. This may be preprocessing code, regression modeling code, or really any other code used to produce the results from the analytic data.

3. Documentation:

Documentation of that code and the data is very important.

4. Distribution:

Finally, there needs to be some standard means of distribution, so all this data in the code is easily accessible

Literate (Statistical) Programming

- An article is a stream of **text** and **code**
- Analysis code is divided into text and code "chunks"
- Each code chunk loads data and computes results
- Presentation code formats results (tables, figures, etc.)
- Article text explains what is going on
- Literate programs can be **weaved** to produce human-readable documents and **tangled** to produce machine-readable documents

Literate (Statistical) Programming

- Literate programming is a general concept that requires
  1. A documentation language (human readable)
  2. A programming language (machine readable)
- Sweave uses LaTeX and R as the documentation and programming languages
- Sweave was developed by Friedrich Leisch (member of the R Core) and is maintained by R core
- Main web site: `http://www.statistik.lmu.de/~leisch/Sweave`

Literate (Statistical) Programming

- knitr is an alternative (more recent) package
- Brings together many features added on to Sweave to address limitations
- knitr uses R as the programming language (although others are allowed) and variety of documentation languages
  - LaTeX, Markdown, HTML
- knitr was developed by Yihui Xie (while a graduate student in statistics at Iowa State)
- See http://yihui.name/knitr/

R Markdown

Introduction

R Markdown provides a unified authoring framework for data science, combining your code, its results, and your prose commentary. R Markdown documents are fully reproducible and support dozens of output formats, like PDFs, Word files, slideshows, and more.

R Markdown files are designed to be used in three ways:

1. For communicating to decision makers, who want to focus on the conclusions, not the code behind the analysis.
2. For collaborating with other data scientists (including future you!), who are interested in both your conclusions, and how you reached them (i.e. the code).
3. As an environment in which to *do* data science, as a modern day lab notebook where you can capture not only what you did, but also what you were thinking.

R Markdown integrates a number of R packages and external tools. This means that help is, by-and-large, not available through?. Instead, as you work through this chapter, and use R Markdown in the future, keep these resources close to hand:

- R Markdown Cheat Sheet: *Help > Cheatsheets > R Markdown Cheat Sheet*,
- R Markdown Reference Guide: *Help > Cheatsheets > R Markdown Reference Guide*.

Prerequisites

You need the rmarkdown package, but you don't need to explicitly install it or load it, as RStudio automatically does both when needed.

R Markdown basics:

This is an R Markdown file, a plain text file that has the extension .Rmd:

````
---
title: "Diamond sizes"
date: 2016-08-25
output: html_document
---
```{r setup, include = FALSE}
library(ggplot2)
library(dplyr)

smaller <- diamonds %>%
  filter(carat <= 2.5)
```

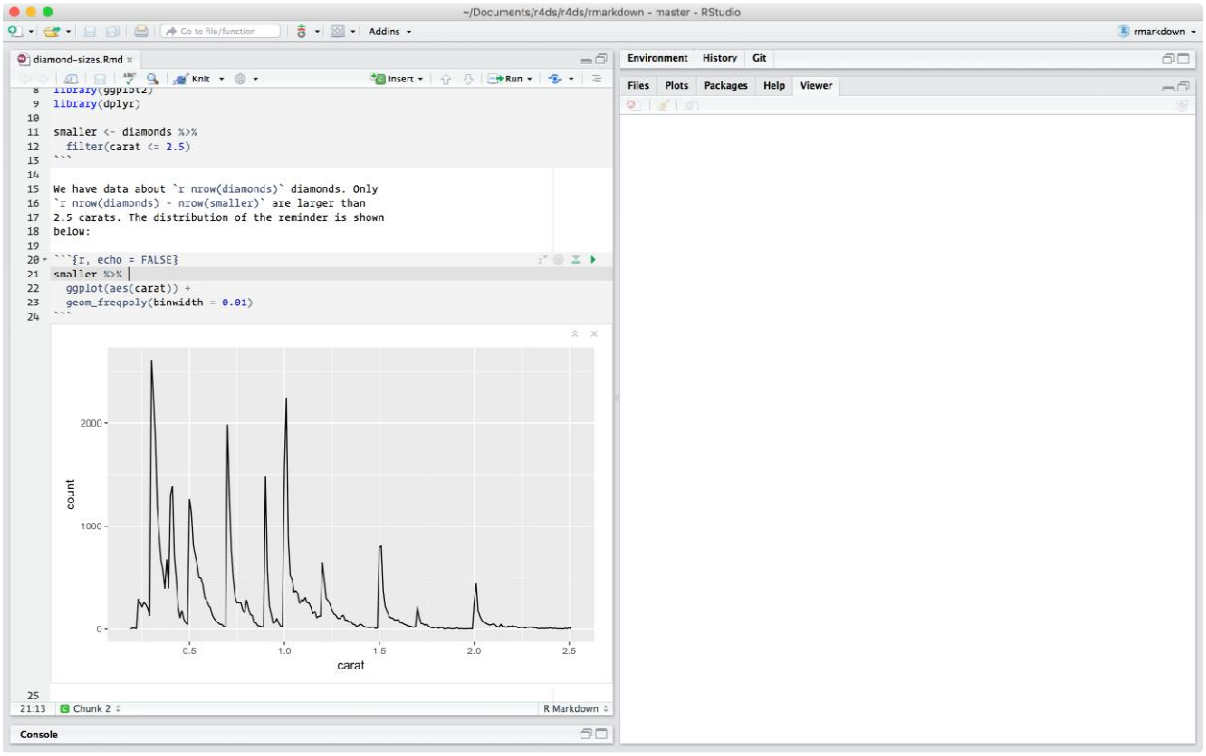
We have data about `r nrow(diamonds)` diamonds. Only
`r nrow(diamonds) - nrow(smaller)` are larger than
2.5 carats. The distribution of the remainder is shown
below:

```{r, echo = FALSE}
smaller %>%
  ggplot(aes(carat)) +
  geom_freqpoly(binwidth = 0.01)
```
````

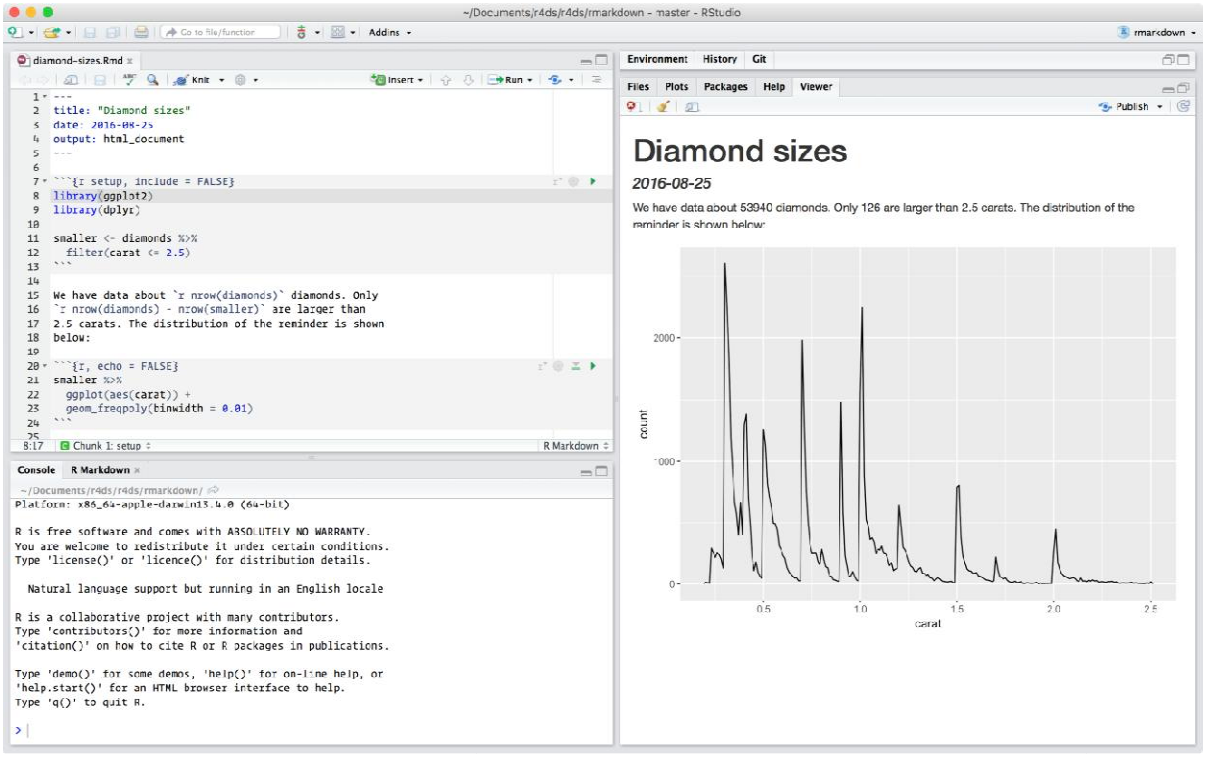
It contains three important types of content:

1. An (optional) YAML header surrounded by ---s.
2. Chunks of R code surrounded by ```.
3. Text mixed with simple text formatting like # heading and _italics_.

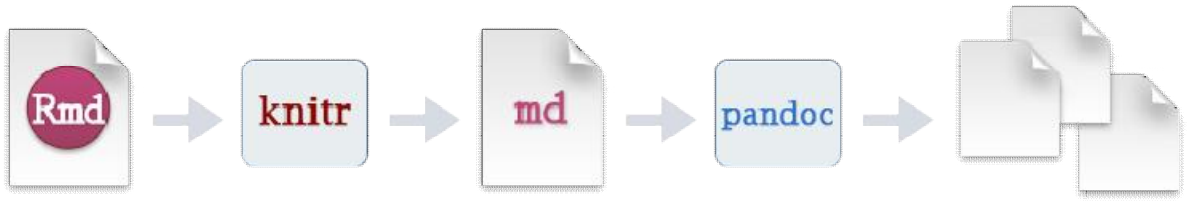
When you open an .Rmd, you get a notebook interface where code and output are interleaved. You can run each code chunk by clicking the Run icon (it looks like a play button at the top of the chunk), or by pressing Cmd/Ctrl + Shift + Enter. RStudio executes the code and displays the results inline with the code:



To produce a complete report containing all text, code, and results, click "Knit" or press Cmd/Ctrl + Shift + K. You can also do this programmatically with rmarkdown::render("1-example.Rmd"). This will display the report in the viewer pane, and create a self-contained HTML file that you can share with others.



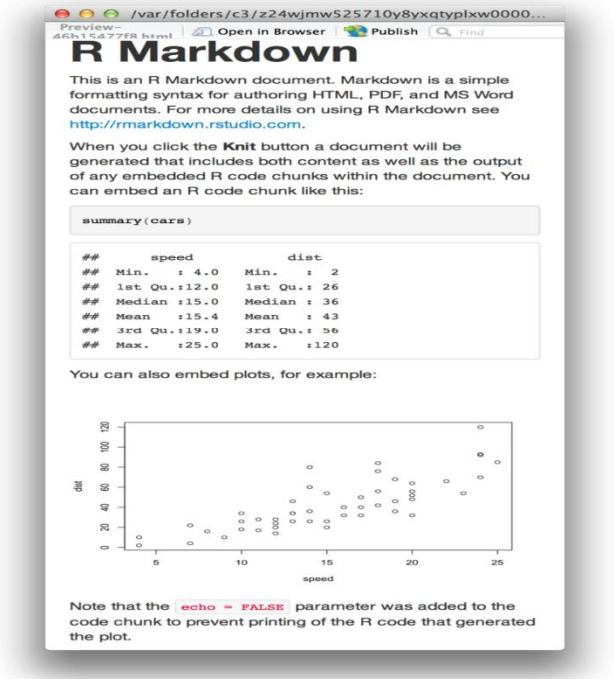
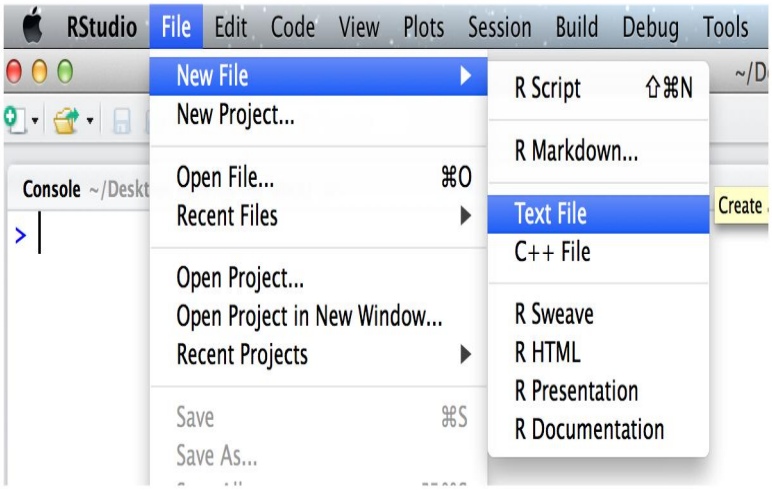
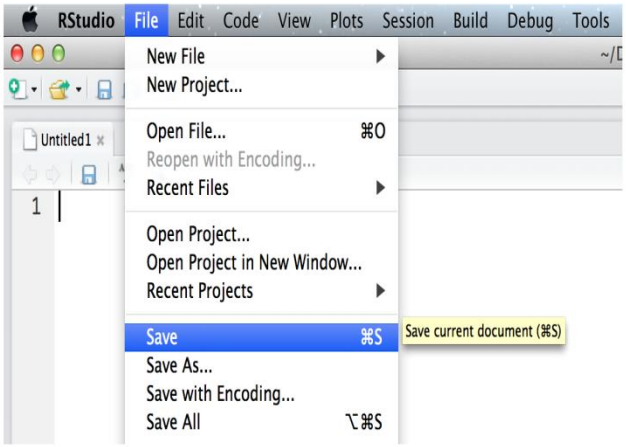
When you knit the document, R Markdown sends the .Rmd file to knitr, http://yihui.name/knitr/, which executes all of the code chunks and creates a new markdown (.md) document which includes the code and its output. The markdown file generated by knitr is then processed by pandoc, http://pandoc.org/, which is responsible for creating the finished file. The advantage of this two step workflow is that you can create a very wide range of output formats, as you'll learn about in R markdown formats.



To get started with your own .Rmd file, select *File > New File > R Markdown…* in the menu bar. RStudio will launch a wizard that you can use to pre-populate your file with useful content that reminds you how the key features of R Markdown work.

Compiling R mark down documents

R Markdown is a file format for making dynamic documents with R. An R Markdown document is written in markdown (an easy-to-write plain text format) and contains chunks of embedded R code, like the document below.

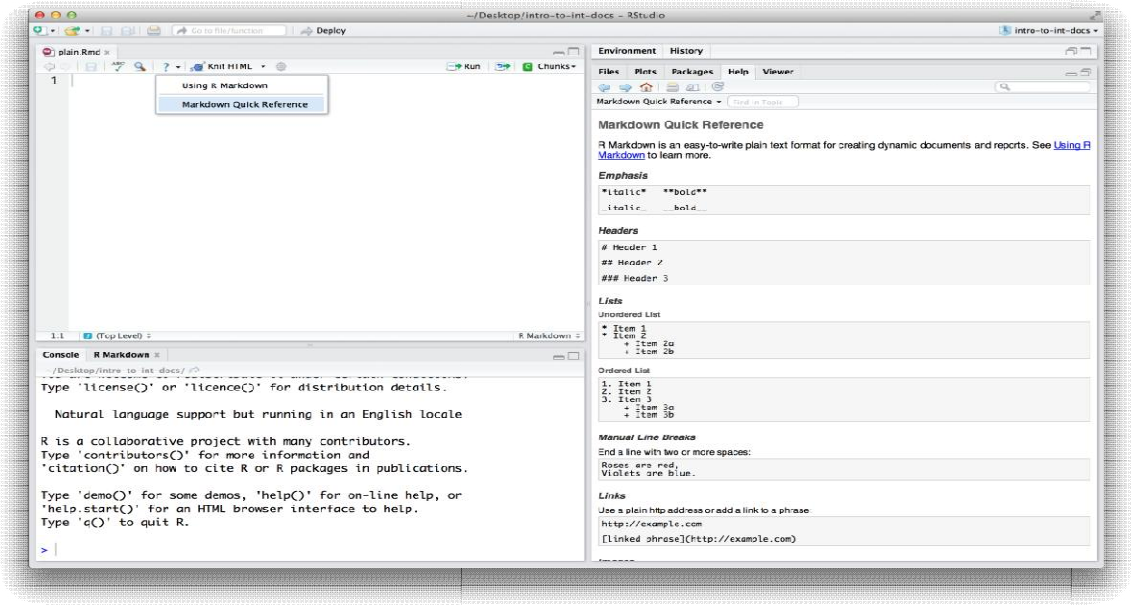
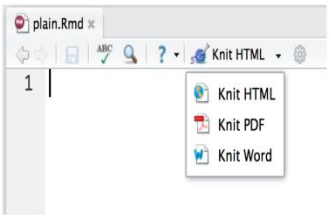
```
---
output: html_document
---
This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see .
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:
```{r}
summary(cars)
```
You can also embed plots, for example:
```{r, echo=FALSE}
plot(cars)
```
Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.
```

R Markdown files are designed to be used with the rmarkdown package. rmarkdown comes installed with the RStudio IDE, but you can acquire your own copy of rmarkdown from CRAN with the command

```
install.packages("rmarkdown")
```

R Markdown files are the source code for rich, reproducible documents. You can transform an R Markdown file in two ways.

1. **knit** - You can *knit* the file. The rmarkdown package will call the knitr package. knitr will run each chunk of R code in the document and append the results of the code to the document next to the code chunk. This workflow saves time and facilitates reproducible reports.

   Consider how authors typically include graphs (or tables, or numbers) in a report. The author makes the graph, saves it as a file, and then copy and pastes it into the final report. This process relies on manual labor. If the data changes, the author must repeat the entire process to update the graph.

   In the R Markdown paradigm, each report contains the code it needs to make its own graphs, tables, numbers, etc. The author can automatically update the report by re-knitting.

2. **convert** - You can *convert* the file. The rmarkdown package will use the pandoc program to transform the file into a new format. For example, you can convert your .Rmd file into an HTML, PDF, or Microsoft Word file. You can even turn the file into an HTML5 or PDF slideshow. rmarkdown will preserve the text, code results, and formatting contained in your original .Rmd file.

   Conversion lets you do your original work in markdown, which is very easy to use. You can include R code to knit, and you can share your document in a variety of formats.

In practice, authors almost always knit and convert their documents at the same time. In this article, I will use the term *render* to refer to the two step process of knitting and converting an R Markdown file.

You can manually render an R Markdown file with rmarkdown::render(). This is what the above document looks like when rendered as a HTML file.



In practice, you do not need to call rmarkdown::render(). You can use a button in the RStudio IDE to render your reprt. R Markdown is heavily integrated into the RStudio IDE .

Getting started

To create an R Markdown report, open a plain text file and save it with the extension *.Rmd*. You can open a plain text file in your scripts editor by clicking File > New File > Text File in the RStudio toolbar.



Be sure to save the file with the extension *.Rmd*. The RStudio IDE enables several helpful buttons when you save the file with the .Rmd extension. You can save your file by clicking File > Save in the RStudio toolbar.



R Markdown reports rely on three frameworks

1. markdown for formatted text
2. knitr for embedded R code
3. YAML for render parameters

Markdown for formatted text

.Rmd files are meant to contain text written in markdown. Markdown is a set of conventions for formatting plain text. You can use markdown to indicate

- bold and italic text
- lists
- headers (e.g., section titles)
- hyperlinks
- and much more

The conventions of markdown are very unobtrusive, which make Markdown files easy to read. The file below uses several of the most useful markdown conventions.

```
# Say Hello to markdown

Markdown is an **easy to use** format for writing reports. It resembles what you naturally write every time you compose an email. In fact, you may have already used markdown *without realizing it*. These websites all rely on markdown formatting

* [Github](www.github.com)

* [StackOverflow](www.stackoverflow.com)

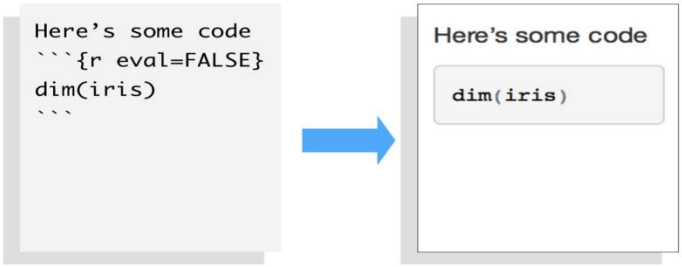
* [Reddit](www.reddit.com)
```

The file demonstrates how to use markdown to indicate:

1. **headers** - Place one or more hashtags at the start of a line that will be a header (or sub-header). For example, # Say Hello to markdown. A single hashtag creates a first level header. Two hashtags, ##, creates a second level header, and so on.
2. **italicized and bold text** - Surround italicized text with asterisks, like this *without realizing it*. Surround bold text with two asterisks, like this **easy to use**.
3. **lists** - Group lines into bullet points that begin with asterisks. Leave a blank line before the first bullet, like this

```
This is a list


* item 1
* item 2
* item 3
```

4. **hyperlinks -** Surround links with brackets, and then provide the link target in parentheses, like this [Github](www.github.com).

You can learn about more of markdown's conventions in the *Markdown Quick Reference* guide, which comes with the RStudio IDE.

To access the guide, open a *.md* or *.Rmd* file in RStudio. Then click the question mark that appears at the top of the scripts pane. Next, select "Markdown Quick Reference". RStudio will open the *Markdown Quick Reference* guide in the Help pane.



Rendering

To transform your markdown file into an HTML, PDF, or Word document, click the "Knit" icon that appears above your file in the scripts editor. A drop down menu will let you select the type of output that you want.



When you click the button, rmarkdown will duplicate your text in the new file format. rmarkdown will use the formatting instructions that you provided with markdown syntax.

Once the file is rendered, RStudio will show you a preview of the new output and save the output file in your working directory.

Here is how the markdown script above would look in each output format.

Note: RStudio does not build PDF and Word documents from scratch. You will need to have a distribution of Latex installed on your computer to make PDFs and Microsoft Word (or a similar program) installed to make Word files.

knitr for embedded R code

The knitr package extends the basic markdown syntax to include chunks of executable R code.

When you render the report, knitr will run the code and add the results to the output file. You can have the output display just the code, just the results, or both.

To embed a chunk of R code into your report, surround the code with two lines that each contain three backticks. After the first set of backticks, include {r}, which alerts knitr that you have included a chunk of R code. The result will look like this

````
Here's some code
```r
dim(iris)
```
```
## [1] 150   5
```
````

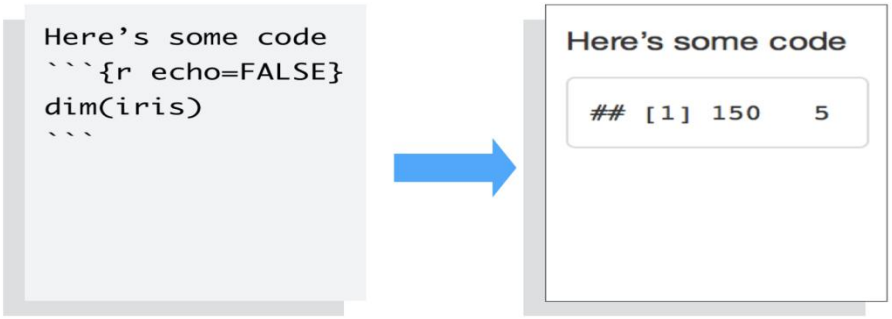
When you render your document, knitr will run the code and append the results to the code chunk. knitr will provide formatting and syntax highlighting to both the code and its results (where appropriate).

As a result, the markdown snippet above will look like this when rendered (to HTML).

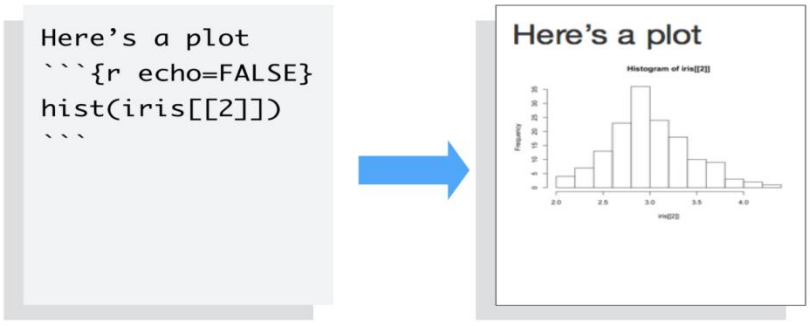
To omit the *results* from your final report (and not run the code) add the argument eval = FALSE inside the brackets and after r. This will place a copy of your code into the report.



To omit the *code* from the final report (while including the results) add the argument echo = FALSE. This will place a copy of the results into your report.



echo = FALSE is very handy for adding plots to a report, since you usually do not want to see the code that generates the plot.



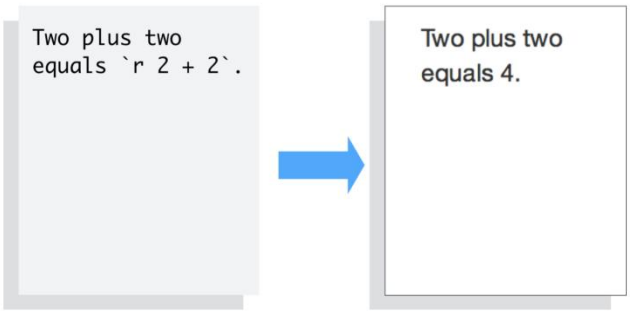
echo and eval are not the only arguments that you can use to customize code chunks. You can learn more about formatting the output of code chunks at the rmarkdown and knitr websites.

Inline code

To embed R code in a line of text, surround the code with a pair of backticks and the letter r, like this.

```
Two plus two equals `r 2 + 2`.
```

knitr will replace the inline code with its result in your final document (inline code is *always* replaced by its result). The result will appear as if it were part of the original text. For example, the snippet above will appear like this:



YAML for render parameters

You can use a YAML header to control how rmarkdown renders your .Rmd file. A YAML header is a section of key: value pairs surrounded by --- marks, like below

```
---
title: "Untitled"
author: "Garrett"
date: "July 10, 2014"
output: html_document
---
Some inline R code, 4.
```

The output: value determines what type of output to convert the file into when you call rmarkdown::render(). *Note: you do not need to specify output: if you render your file with the RStudio IDE knit button.*

Output : recognizes the following values:

- html_document, which will create HTML output (default)
- pdf_document, which will create PDF output
- word_document, which will create Word output

If you use the RStudio IDE knit button to render your file, the selection you make in the gui will override the output: setting.

Slideshows

You can also use the output: value to render your document as a slideshow.

- output: ioslides_presentation will create an ioslides (HTML5) slideshow
- output: beamer_presentation will create a beamer (PDF) slideshow

Note: The knit button in the RStudio IDE will update to show slideshow options when you include one of the above output values and save your .Rmd file.

What is Knitr?

knitr is an engine for dynamic report generation with **R**. It is a package in the statistical programming language **R** that enables integration of **R** code into LaTeX, LyX, HTML, Markdown, AsciiDoc, and reStructuredText documents.

The purpose of knitr is to allow reproducible research in R through the means of Literate Programming. It is licensed under the GNU General Public License. knitr was inspired by Sweave and written with a different design for better modularization, so it is easier to maintain and extend. Sweave can be regarded as a subset of knitr in the sense that all features of Sweave are also available in knitr. Some of knitr's extensions include the R Markdown format (used in reports published on RPubs), caching, TikZ graphics and support to other languages such as Python, Perl, C++, Shell scripts and CoffeeScript, and so on.

knitr is officially supported in the RStudio IDE for R, LyX, Emacs/ESS and the Architect IDE for data science.

Knitr commands:

Documents that contain R code must be saved with the extension .Rtex, otherwise the code won't work. Let's see an example:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[english]{babel}

\begin{document}
You can type R commands in your \LaTeX{} document and they will be properly run and the output printed in the document.

<<>>=
# Create a sequence of numbers
X = 2:10

# Display basic statistical measures
summary(X)

@
\end{document}
```

You can type R commands in your LaTeX document and they will be properly run and the output printed in the document.

```
# Create a sequence of numbers
X = 2:10

# Display basic statistical measures
summary(X)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2       4       6       6       8      10
```

As you see, the text in between the characters <<>>= and @ is R code, this code and its output is printed in a listing-like format.

This chunk of code can take some extra parameters to customize the dynamic output.

Chunks of code

A code block as the one presented in the previous section is usually called a *chunk*. You can set some extra options in knitr chunks. See the example below:

You can type R commands in your \LaTeX{} document and they will be properly run and the output printed in the document.

```
<<echo=FALSE, cache=TRUE>>=
# Create a sequence of numbers
X = 2:10

# Display basic statistical measures
summary(X)

@
```

You can type R commands in your LaTeX document and they will be properly run and the output printed in the document.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2       4       6       6       8      10
```

There are three additional options passed inside << and >>.

echo=FALSE
This hides the code and only prints the output generated by R.

cache=TRUE
If cache is set to true the chunk is not run, only the objects generated by it. This saves time if the data in that chunk haven't changed. *Note that the cache=TRUE option is not currently supported in ShareLaTeX, but it should work locally.*

Inline commands

It is possible to access objects generated in a chunk and print them in-line.

You can type R commands in your \LaTeX{} document and they will be properly run and the output printed in the document.

```
<<echo=FALSE, cache=TRUE>>=
# Create a sequence of numbers
X = 2:10

# Display basic statistical measures
summary(X)

@

So, the mean of the data is $\Sexpr{mean(X)}$
```

You can type R commands in your LaTeX document and they will be properly run and the output printed in the document.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2       4       6       6       8      10
```

So, the mean of the data is 6

The command \Sexpr{mean(X)} prints the output returned by the R code mean(X). Inside the braces any R command can be passed.

Plots

Plots can also be added to a **knitr** document. See the next example

```
<<plot1, fig.pos="t", fig.height=4, fig.width=4, fig.cap="First plot">>=

xdata = read.csv(file="data.txt", head=TRUE,sep=" ")

hist(xdata$data, main="ShareLaTeX histogram", xlab="Data")

@

The figure \ref{fig:plot1} is simple histogram.
```

This histogram uses data stored in "data.txt", saved in the current working directory. A few figure-related options are passed to the chunk.

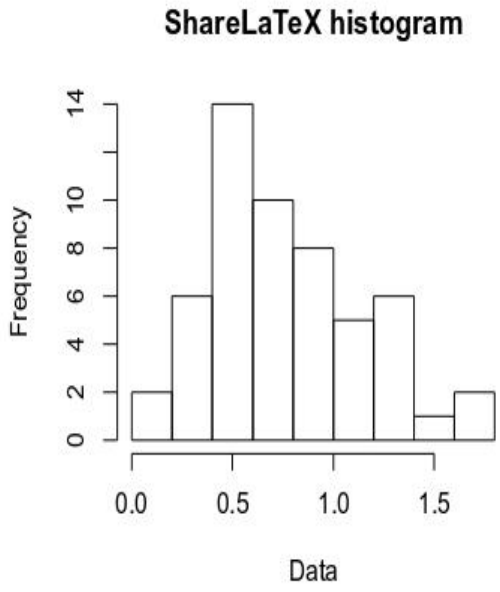


Figure 1: First plot

```
xdata = read.csv(file = "data.txt", head = TRUE, sep = " ")

hist(xdata$data, main = "ShareLaTeX histogram", xlab = "Data")
```

The figure 1 is simple histogram.

plot1

This is the label used to reference the plot. The prefix "fig:" is mandatory. You can see in the example that the figure is referenced with \ref{fig:plot1}.

fig.pos="t"

Positioning parameter. This is the same used in the figure environment.

fig.height=4, fig.width=4

Figure width and height

fig.cap="First plot"

Caption for the figure.

POSSIBLE QUESTIONS

2 MARKS

1. What is **Reproducible research**?
2. What is replication?
3. What is wrong with replication?
4. Why do we need Reproducible Research?
5. Define Research Pipeline.
6. List the elements of Reproducibility.
7. Define R Markdown.
8. What is Knitr?
9. How can you transform R Markdown files?

6 MARKS

1. Write about Reproducible Research in detail.
2. Explain in detail about R Markdown.
3. How to Compiling R mark down documents with example.
4. Explain Knitr commands with example.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

UNIT - V

Sno	Questions	opt1	opt2	opt3	opt4	Answer
1	_____ is an indication that a fatal problem has occurred and execution of the function stops	message	error	warning	stop	error
2	What will be the value of following expression ?	Warning in log(c(-1, 2)): NaNs produced	Error in log(c(-1, 2)): NaNs produced	Message	error	Error in log(c(-1, 2)): NaNs produced
3	_____ prints out the function call stack after an error occurs.	trace()	traceback()	back()	backerror()	traceback()
4	Point out the wrong statement :	The primary task of debugging any R code is correctly diagnosing what the problem is	R provides only two tools to help you with debugging your code	print statement can be used for debugging purpose	The traceback() function must be called immediately after an error occurs	R provides only two tools to help you with debugging your code
5	Which of the following is primary tool for debugging ?	debug()	trace()	browser()	traceback()	debug()
6	_____ allows you to insert debugging code into a function a specific places	debug()	trace()	browser()	traceback()	trace()
7	Point out the correct statement :	The traceback() function must be called immediately after an error occurs	The debugger calls the browser at the very low level of the function body	Every time you call the mod() function it will launch the interactive debugger	R provides only two tools to help you with debugging your code	The traceback() function must be called immediatel y after an error occurs

8	_____ allows you to modify the error behavior so that you can browse the function call stack	debug()	trace()	recover()	traceback()	recover()
9	_____ suspends the execution of a function wherever it is called and puts the function in debug mode	debug()	trace()	recover()	browser()	browser()
10	debug() flags a function for _____ mode in R mode.	debug	run	compile	recover	run
11	What would be the output of the following code ? > mean(x) Error in mean(x) : object 'x' not found > traceback()	1: mean(x)	Null	0	1	1: mean(x)
12	The recover() function will first print out the function call stack when an _____ occurs.	Error	Warning	Messages	stop	Error
13	_____ is a systematic way to examine how much time is spent in different parts of a program.	Profiling	Monitoring	Logging	Scheduling	Profiling
14	Point out the correct statement :	The Rprofiler() function starts the profiler in R	Using system.time() allows you to test certain functions or code blocks to see if they are taking excessive amounts of time	R must not be compiled with profiler support	Rprofiler() tabulates how much time is spent inside each function	Using system.time() allows you to test certain functions or code blocks to see if they are taking excessive amounts of time
15	R comes with a _____ to help you optimize your code and improve its performance.	debugger	monitor	browser	profiler	debugger
16	The _____ function computes the time (in seconds) needed to execute an expression.	system.timedeb()	system.time()	system.datetime()	system.time date()	system.time()

17	Point out the correct statement :	Rprofiler() tabulates how much time is spent inside each function	Rprof() keeps track of the function call stack at regularly sampled intervals	By default, the profiler samples the function call stack every 2 seconds	R must not be compiled with profiler support	Rprof() keeps track of the function call stack at regularly sampled intervals
18	system.time function returns an object of class _____ which contains two useful bits of information.	debug_time	proc_time	procedure_time	proced_time	proc_time
19	_____ time is time charged to the CPU(s) for the R expression.	elapsed	user	response	request	elapsed
20	The elapsed time may be _____ than the user time if your machine has multiple cores/processors	smaller	greater	equal to	not equal to	smaller
21	Parallel processing is done via _____ package can make the elapsed time smaller than the user time.	parallel	statistics	distributed	equal	parallel
22	You can time _____ expressions by wrapping them in curly braces within the call to system.time().	smaller	longer	error	warning	longer
23	The profiler can be turned off by passing _____ to Rprof().	0	1	2	NULL	NULL
24	Point out the correct statement :	Rprof() is used to turn off the profiler	At each line of the output, the profiler writes out the function call stack	The summaryprof() function tabulates the R profiler output	R must not be compiled with profiler support	At each line of the output, the profiler writes out the function call stack
25	How many methods exist for normalizing the data ?	one	two	three	profiler	two
26	_____ divides the time spend in each function by the total run time	“by.sum”	“by.total”	“by.self”	“by.mull”	“by.total”

27	Point out the correct statement :	“by.total” first subtracts out time spent in functions above the current function in the call stack	The summaryRprof() function calculates how much time is spend in which function	By default, the profiler samples the function call stack every 0.02 seconds	R must not be compiled with profiler support	By default, the profiler samples the function call stack every 0.02 seconds
28	Which of the following function actually fits the linear model ?	lm.time()	lm.date()	lm.fit()	lm.day()	lm.fit()
29	_____ time is time charged to the CPU(s) for the R expression.	elapsed	user	response	request	elapsed
30	The final bit of output that summaryRprof() provides is the _____ interval and the total runtime.	response	sampling	processing	request	sampling
31	Which of the following statement gives sampling interval ?	$sampling.interval	$sampling.time	$sampling.date	$sampling.day	$sampling.time
32	Which of the following code is not profiled ?	C	C++	Java	.Net	C
33	_____ generate random Normal variates with a given mean and standard deviation	dnorm	rnorm	pnorm	rpois	rnorm
34	Point out the correct statement :	R comes with a set of pseudo-random number generators	Random number generators cannot be used to model random inputs	Statistical procedure does not require random number generation	For each probability distribution there are typically three functions	R comes with a set of pseudo-random number generators
35	_____ evaluate the cumulative distribution function for a Normal distribution	dnorm	rnorm	pnorm	rpois	pnorm
36	_____ generate random Poisson variates with a given rate	dnorm	rnorm	pnorm	rpois	rpois

37	Point out the wrong statement :	For each probability distribution there are typically three functions	For each probability distributio n there are typically four functions	r function is sufficient for simulating random numbers	R comes with a set of pseudo-random number generators	For each probability distribution there are typically three functions
38	Which of the following evaluate the Normal probability density (with a given mean/SD) at a point ?	dnorm	rnorm	pnorm	rpois	dnorm
39	_____ is the most common probability distribution to work with.	Gaussian	Parametric	Paradox	paradix	Gaussian
40	What will be the output of the following code ? > pnorm(2)	0.9772499	1.9772499	0.6772499	0.8772499	0.9772499
41	_____ ensures reproducibility of the sequence of random numbers.	sets.seed()	set.seed()	set.seedval ue()	seedvalue()	set.seed()
42	Point out the correct statement :	It is not possible to generate random numbers from other probability distributions like the Poisson	When simulating any random numbers it is not essential to set the random number seed	You should always set the random number seed when conducting a simulation	The sample() function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions of numbers	You should always set the random number seed when conducting a simulation
43	5 Normal random numbers can be generated with rnorm() by setting seed value to :	1	2	3	4	1
44	_____ function is used to simulate binary random variables.	dnorm	rbinom	binom	rpois	rbinom

45	Point out the wrong statement :	Drawing samples from specific probability distributions can be done with “s” functions	The sample() function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions of numbers	The sampling() function draws randomly from a specified set of objects	You should always set the random number seed when conducting a simulation	The sample() function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions of numbers
46	What will be the output of the following code ? > set.seed(10) > x <- rbinom(100, 1, 0.5) > str(x)	int [1:100] 1 0 0 1 0 0 0 0 1 0 ...	int [1:100] 10 0 01 1 0 0 01 0 1 0 ...	int [1:100] 1 03 0 1 0 0 0 02 1 0 ...	int [1:100] 1 2 3 1 1 0 0 0 1 0 ...	int [1:100] 1 0 0 1 0 0 0 0 1 0 ...
47	_____ distribution is commonly used to model data that come in the form of counts.	Gaussian	Parametric	Poisson	Paradox	Poisson
48	What will be the output of the following code ? > rpois(10, 1)	[1] 7 0 1 1 2 1 1 4 1 2	[1] 0 8 1 1 2 1 1 4 1 2	[1] 0 0 1 1 2 1 1 4 1 2	[1] 0 9 1 1 2 1 1 5 1 2	[1] 0 0 1 1 2 1 1 4 1 2
49	Which of the following code represents count with mean of 2 ?	rpois(10, 2)	rpois(10, 20)	rpois(20, 2)	rpois(2, 20)	rpois(10, 2)
50	The _____ function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions of numbers.	sam()	seed()	sample()	samp()	sample()
51	_____ is an important (and big) topic for both statistics and for a variety of other areas where there is a need to introduce randomness.	Simulation	samplie	distribution	normal	Simulation
52	Setting the _____ number generator seed via set.seed() is critical for reproducibility	arbitrary	sample	random	sequence	random

53	The _____ function tabulates the R profiler output and calculates how much time is spend in which function.	prof()	summaryR prof()	Rprof()	Rpro()	summaryRp rof()
54	Interactive debugging tools _____,_____,_____,_____ and _____ can be used to find problematic code in functions	trace, debug, browser, backtrace, and recover	traceback, debug, browser, trace, and recover	traceback, debug, browser, trace, and request	traceback, debug, browser, request, and recover	traceback, debug, browser, trace, and recover
55	The _____ function will first print out the function call stack when an error occurrs.	debug()	trace()	recover()	traceback()	recover()
56	In simulating linear model can also simulate from_____ where the errors are no longer from a Normal distribution but come from some other distribution.	generalized model	generalized linear model	linear model	ungeneraliz ed linear model	generalized linear model
57	Simulating _____ numbers is useful but sometimes we want to simulate values that come from a specific model.	arbitrary	sample	random	sequence	random
58	The function call stack is the _____ of functions that was called before the error occurred.	arbitrary	sample	random	sequence	sequence
59	In which case the _____function tried to evaluate the formula y ~ x and realized the object y did not exist.	debug()	trace()	eval()	traceback()	eval()
60	_____ time charged to the CPU(s) for this expression	sample.time	user time	elapsed time	system.time	user time
61	Which of the following problem is solved by reproducibility ?	Scalability	Data Availability	Improved Data Analysis	interoperabi lity	Data Availability
62	Point out the wrong statement with respect to reproducibility:	Focuses on the validity of the data analysis	The ultimate standard for strengtheni ng scientific evidence	Important when replication is impossible	Important when the data was duplicate is impossible	The ultimate standard for strengtheni ng scientific evidence

63	Which of the following can be used for data analysis model ?	CRAN	CPAN	CTAN	All the above Mentioned	All the above Mentioned
64	_____ determines correctness of data analysis.	Reproducibility	Researchability	reanalysis	reoperability	Reproducibility
65	Which of the following gives reviewers an important tool without dramatically increasing the burden ?	Quality research	Replication research	Reproducible research	software research	Reproducible research

Register Number_____

[16CSU503B]

KARPAGAM ACADEMY FOR HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2016 onwards)

FIRST INTERNAL EXAMINATION, JULY 2018

Fifth Semester

COMPUTER SCIENCE

INTRODUCTION TO DATA SCIENCE

Date & Session: .07.2018 **Class : III B.Sc CS A & B**

Maximum : 50 Marks **Duration: 2 Hours**

PART-A (20 X 1 = 20 Marks)

(Answer ALL the Questions)

1. _____ programming language is a dialect of S.
   a) B b) C c) R d) K
2. In 1991, R was created by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of _____
   a) John Hopkins b) California c) Harvard d) Auckland
3. Which of the following describes R language?
   a) Free b) Paid c) Available for free trial only d) Trail
4. The primary R system is available from the _____
   a) CRAN b) CRWO c) GNU d) RAN
5. Files containing R scripts end with extension:
   a) .S b) .R c) .Rp d) .RR
6. Who developed S?
   a) Dennis Ritchie b) Bjarne Stroustrup c) James Gosling d) John Chambers
7. Which of the following statement would print “0” “1” “2” “3” “4” “5” “6” for the following code?
   a) as.character(x) b) as.logical(x) c) as.numeric(x) d) as.integer(x)
8. OPD stands for_____
   a) Organizing, Packaging and Delivering Data
   b) Optimizing, Packaging and Delivering Data
   c) Organizing, Producing and Delivering Data
   d) Online Product Delivery data
9. _____ includes data set, variables, vectors, functions etc.
   a) R Script b) R Environment c) R Console d) R Output
10. VCS are classified into _____
    a) 3 b) 2 c) 4 d) 5
11. _____is a system that records changes to a file or set of files over time.
    a) Time control b) Version control c) Section Control d) Source Control

12. _____area shows the output of code you run.
    a) R Script b) R Environment c) R Console d) R Output
13. To run R codes, simply select the line(s) of code and press _____
    a) Ctrl + S b) Ctrl + Shift c) Ctrl + Alt d) Ctrl + Enter
14. R is an example of ____
    a) FLASS b) FLOSS c) FLESS d) FLSSS
15. The <- Symbol is the _____ operator.
    a) Comparison Operator b) Assignment Operator
    c) Logical Operator d) Boolean Operator
16. _____ function is used to find the data type of the variable
    a) datatype( ) b) class( ) c) type( ) d) cls( )
17. In Matrices every element is of the ____ class.
    a) same b) different c) literal d) unique
18. R does not support _____ comments or comment blocks.
    a) single line b) * c) multi line d) //
19. What would be the result of following code ? > x<- 2 class(x)
    a) "integer" b) "numeric" c) "logical" d) "real"
20. R is_____ tool.
    a) platform-dependent b) platform-independent
    c) Multi platform d) only in LINUX

PART-B (3 X 2 = 6 Marks)
(Answer ALL the Questions)

21. What is Data science?
22. Compare Data Science vs. Data Analysis?
23. Mention the uses of R?

PART-C (3 X 8 = 24 Marks)
(Answer ALL the Questions)

24. a) Explain in detail about the tools that will be used in building data analysis software.

[OR]

b) Write a brief description about data science.

25. a) Demonstrate the R-Studio IDE Environment with neat diagram.

[OR]

b) Explain R data types in detail with proper example.

26. a) Write a detailed note on reading and writing data to and from R.

[OR]

b) Illustrate the Features in R Programming.

Register Number_____

[16CSU503B]

KARPAGAM ACADEMY FOR HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2016 onwards)

FIRST INTERNAL EXAMINATION, JULY 2018

Fifth Semester

COMPUTER SCIENCE

INTRODUCTION TO DATA SCIENCE

ANSWER KEY

PART-A (20 X 1 = 20 Marks)

(Answer ALL the Questions)

1. _____ programming language is a dialect of S.

a) B b) C **c) R** d) K

2. In 1991, R was created by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of _____

a) John Hopkins b) California c) Harvard **d) Auckland**

3. Which of the following describes R language?

a) Free b) Paid c) Available for free trial only d) Trail

4. The primary R system is available from the _____

a) CRAN b) CRWO c) GNU d) RAN

5. Files containing R scripts end with extension:

a) .S **b) .R** c) .Rp d) .RR

6. Who developed S?

a) Dennis Ritchie b) Bjarne Stroustrup c) James Gosling **d) John Chambers**

7. Which of the following statement would print "0" "1" "2" "3" "4" "5" "6" for the following code?

a) as.character(x) b) as.logical(x) **c) as.numeric(x)** d) as.integer(x)

8. OPD stands for_____

a) Organizing, Packaging and Delivering Data

b) Optimizing, Packaging and Delivering Data

c) Organizing, Producing and Delivering Data

d) Online Product Delivery data

9. _____ includes data set, variables, vectors, functions etc.

a) R Script b) R Environment c) R Console d) R Output

10. VCS are classified into _____

a) 3 b) 2 c) 4 d) 5

11. _____is a system that records changes to a file or set of files over time.

a) Time control **b) Version control** c) Section Control d) Source Control

12. _____area shows the output of code you run.

a) R Script b) R Environment **c) R Console** d) R Output

13. To run R codes, simply select the line(s) of code and press _____

a) Ctrl + S b) Ctrl + Shift c) Ctrl + Alt d) Ctrl + Enter

14. R is an example of ____

a) FLASS **b) FLOSS** c) FLESS d) FLSSS

15. The <- Symbol is the _____ operator.

a) Comparison Operator **b) Assignment Operator**

c) Logical Operator d) Boolean Operator

16. _____ function is used to find the data type of the variable

a) datatype( ) **b) class( )** c) type( ) d) cls( )

17. In Matrices every element is of the ____ class.

a) same b) different c) literal d) unique

18. R does not support _____ comments or comment blocks.

a) single line b) * **c) multi line** d) //

19. What would be the result of following code? > x<- 2 class(x)

a) "integer" **b) "numeric"** c) "logical" d) "real"

20. R is_____ tool.

a) platform-dependent **b) platform-independent**

c) Multi platform d) only in LINUX

PART-B (3 X 2 = 6 Marks)
(Answer ALL the Questions)

21. What is Data science?

Definition 1: Data science is a technique to change the raw data into information. It is the study of where the valuable data comes from, what it represents and how it can be turned into a valuable resource in the creation of business and IT strategies.

Definition 2: Data Science, it is also known as data driven science, which makes use of scientific methods, processes and systems to extract knowledge or insights from data in various forms, i.e either structured or unstructured.

22. Compare Data Science vs. Data Analysis?

Data Science	Data Analysis
Providing strategic actionable insights into the world	Providing operational observations into issues
Mathematical, technical and strategic knowledge are mandatory	Data analysis and visualization skills required
Deal with big data	Not necessarily deal with big data

23. Mention the uses of R?

- Weather Service uses R to predict severe flooding.
- Social networking companies are using R to monitor their user experience.
- Newspapers companies are using R to create infographics and interactive data journalism applications.

PART-C (3 X 8 = 24 Marks)
(Answer ALL the Questions)

24. a) Explain in detail about the tools that will be used in building data analysis software.

1. Version control and its Types - with appropriate diagrams

A) Local Version Control Systems

B) Centralized Version Control Systems

C) Distributed Version Control Systems

2. Markdown

Markdown is a lightweight markup language with plain text formatting syntax. It is designed so that it can be converted to HTMLand many other formats using a tool by the same name.Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor.

3. git

Git is considered to be a newer and faster emerging star when it comes to version control systems. First developed by the creator of Linux kernel, Linus Torvalds, Git has begun to take the community for web development and system administration by storm, offering a largely different form of control.

4. GitHub

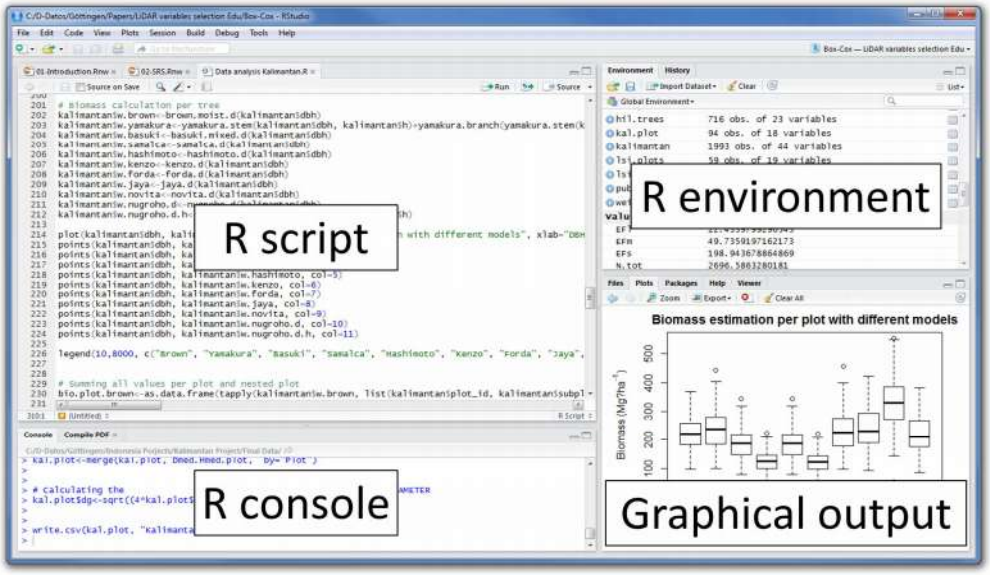
GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

5. R

R provides a wide variety of statistical (linear and nonlinear modeling, 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.

6. RStudio.

RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. RStudio was founded by JJ Allaire, creator of the programming language ColdFusion. Hadley Wickham is the Chief Scientist at RStudio.



1. **R Console:** This area shows the output of code you run. Also, you can directly write codes in console. Code entered directly in R console cannot be traced later. This is where R script comes to use.
2. **R Script:** As the name suggest, here you get space to write codes. To run those codes, simply select the line(s) of code and press Ctrl + Enter. Alternatively, you can click on little 'Run' button location at top right corner of R Script.
3. **R environment:** This space displays the set of external elements added. This includes data set, variables, vectors, functions etc. To check if data has been loaded properly in R, always look at this area.
4. **Graphical Output:** This space display the graphs created during exploratory data analysis. Not just graphs, you could select packages, seek help with embedded R's official documentation.

[OR]

b) Write a brief description about data science.

1) What is Data science? – Definition
2) Who is a Data Scientist?

Data Scientist = Programmer + Computer Scientist + Mathematician + Story teller + Domain Expert

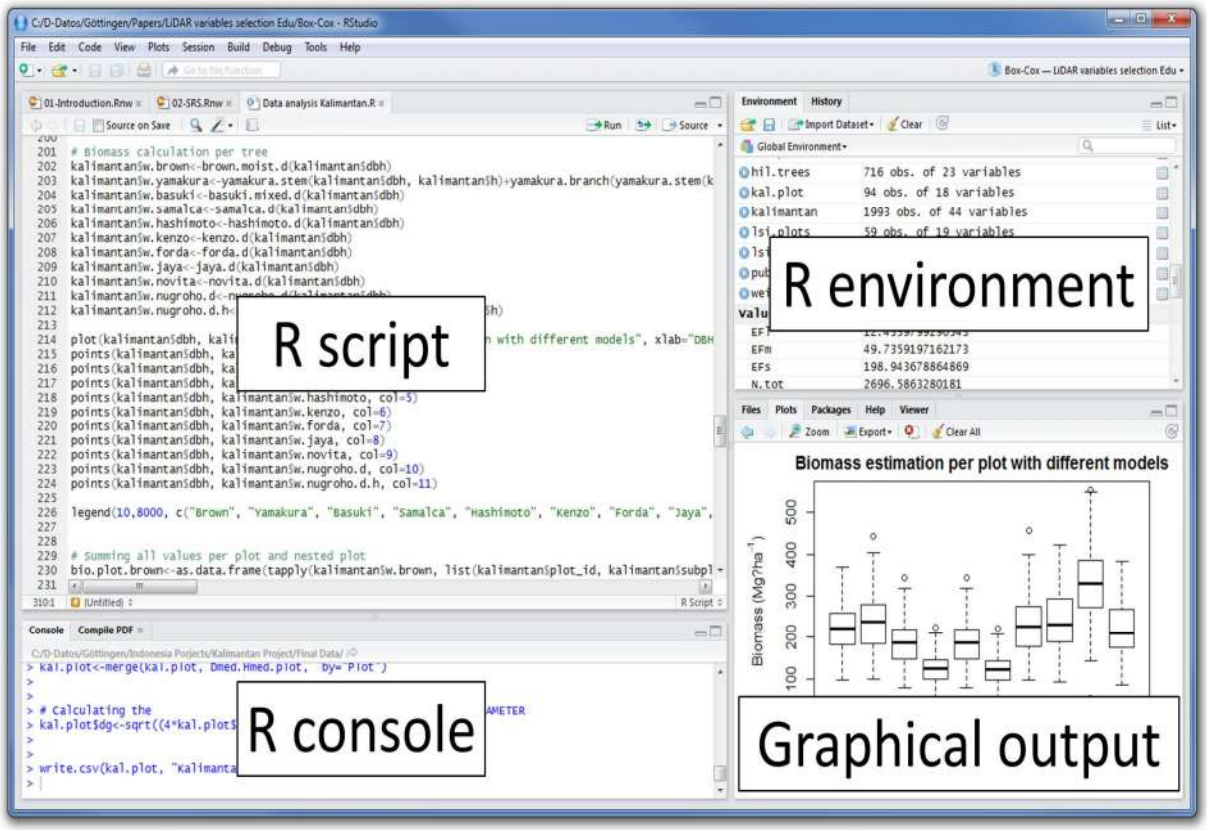
3) OPD Data Science Process

4) Data Science Components

i). Datasets

ii). R Studio

25. a) Demonstrate the R-Studio IDE Environment with neat diagram.



1. **R Console:** This area shows the output of code you run. Also, you can directly write codes in console. Code entered directly in R console cannot be traced later. This is where R script comes to use.
2. **R Script:** As the name suggest, here you get space to write codes. To run those codes, simply select the line(s) of code and press Ctrl + Enter. Alternatively, you can click on little 'Run' button location at top right corner of R Script.
3. **R environment:** This space displays the set of external elements added. This includes data set, variables, vectors, functions etc. To check if data has been loaded properly in R, always look at this area.
4. **Graphical Output:** This space display the graphs created during exploratory data analysis. Not just graphs, you could select packages, seek help with embedded R's official documentation.

[OR]

b) Explain R data types in detail with proper example.

Detailed explanation of the following with appropriate examples.

- Numeric
- Integer
- Complex
- Logical
- Character
- R has a wide variety of data types including **scalars, vectors (numerical, character, logical), matrices, data frames, and lists.**

26. a) Write a detailed note on reading and writing data to and from R.

Functions for Reading Data into R:
There are a few very useful functions for reading data into R.

1. **read.table()** and **read.csv()** are two popular functions used for reading tabular data into R.
2. **readLines()** is used for reading lines from a text file.
3. **source()** is a very useful function for reading in R code files from a another R program.
4. **dget()** function is also used for reading in R code files.
5. **load()** function is used for reading in saved workspaces
6. **unserialize()** function is used for reading single R objects in binary format.

Functions for Writing Data to Files:
There are similar functions for writing data to files

1. **write.table()** is used for writing tabular data to text files (i.e. CSV).
2. **writeLines()** function is useful for writing character data line-by-line to a file or connection.
3. **dump()** is a function for dumping a textual representation of multiple R objects.
4. **dput()** function is used for outputting a textual representation of an R object.
5. **save()** is useful for saving an arbitrary number of R objects in binary format to a file.

6. **serialize()** is used for converting an R object into a binary format for outputting to a connection (or file).

[OR]

b) Illustrate the Features in R Programming.

- R is simple and effective programming language which has been well-developed, as well as R is data analysis software.
- R is a well - designed, easy and effective language that has the concepts of conditionals, looping, user-defined recursive procedures and various I/O facilities.
- R has a large, consistent and incorporated set of tools used for data analysis.
- R contains suite of operators for different types of calculations on arrays, lists and vectors.
- R provides highly extensible graphical techniques.
- R graphical techniques for data analysis output either directly display to the computer, or can be print on paper.
- R has an effective data handling and storage facility.
- R is an online vibrant community.
- R is free, open source, powerful and highly extensible.