



# Online Workshop on Life science meets Programming

13 - 15 September 2022



TRAINING MANUAL

Lecture notes

*Organized by*

**Bioinformatics Centre**

ICAR-Indian Institute of Spices Research

Kozhikode, Kerala

**Online workshop on “Life science meets Programming”  
September 13-15, 2022**

**Training Manual: Lecture Notes**

**Compiled by**

**Sona Charles**

**Mukesh Sankar S**

**Jayarajan K**

**Fayad M A**

**Organized by**

**Bioinformatics Centre,**

**ICAR-Indian Institute of Spices Research,**

**Kozhikode, Kerala, India**

**2022**

**Published by**

Dr. CK Thankamani  
Director, ICAR- Indian Institute of Spices Research

**Citation:**

Charles.S *et al.* (2022) Life science meets programming –Training Manual: Lecture notes. ICAR- Indian Institute of Spices Research, Kozhikode, Kerala, India (pp.)

**Manuscript No.:** IISR 2022/01

**Workshop Convenor**

Ms. Sona Charles,  
Scientist (Agricultural Bioinformatics),  
Bioinformatics Centre,  
ICAR- Indian Institute of Spices Research,  
Kozhikode, Kerala-673012.

**Course associate**

Mr. Mukesh Sankar. S,  
Scientist (Crop Improvement & Biotechnology),  
ICAR- Indian Institute of Spices Research,  
Kozhikode, Kerala-673012.

Mr. Jayarajan. K,  
Chief Technical Officer,  
ICAR- Indian Institute of Spices Research,  
Kozhikode, Kerala-673012.

**Published by:**

ICAR-Indian Spices Research Institute, Kozhikode, Kerala.  
<http://www.spices.res.in/>

**Disclaimer:** The contents of the manual are lecture materials provided by the resource persons and collected from other resources available in public domain. The contents are non-peer reviewed. Anything contained herein does not account to the views of Indian Council of Agricultural Research, ICAR- Indian Institute of Spices Research.

## ONLINE WORKSHOP ON “Life science meets Programming”

### PROGRAM SCHEDULE

Day 1: 13-09-2022		
10:00 am	Welcome Address	Dr. Mukesh Sankar S, Scientist, ICAR-Indian Institute of Spices Research
10:10 am	Introductory Remarks and Release of Training Manual	Dr. CK Thankamani, Director, ICAR-Indian Institute of Spices Research
10:20 am	Felicitations	Dr. KV Saji, Head, Crop Improvement and Biotechnology Division, ICAR-Indian Institute of Spices Research
10:25 am	Introduction to the course and vote of thanks	Ms. Sona Charles, Scientist (Bioinformatics), ICAR-Indian Institute of Spices Research
<i>Pre-workshop evaluation and photo session</i>		
11:00 am	Inaugural Lecture	<b>“Coding for decoding secrets of life”</b> Dr. Santhosh J Eapen, Former Director, ICAR- Indian Institute of Spices Research
12:15 pm	Setting up the computer	Dr. Mukesh Sankar S Mr. Jayarajan Mr. Fayad M
02:00 pm	Utilities in Bioinformatics	Ms. Sona Charles Scientist (Bioinformatics), ICAR- Indian Institute of Spices Research
03:30 pm	Introduction to R	Dr. Mukesh Sankar S, Scientist (Plant Breeding), ICAR-Indian Institute of Spices Research
Day 2: 14-09-2022		
10:00 am	Data Visualization using R	Ms. Sona Charles
02:00pm	Introduction to Linux	Dr. Merlin Lopez, Scientist, Community Agrobiodiversity Centre, MS Swaminathan Research Foundation, Kerala
02:30pm	Linux- Hands on exercise	Dr. Merlin Lopez Mr. Fayad M, Research scholar, ICAR-IISR, Kerala.
Day 3: 15-09-2022		
10:00 am	Introduction to Python	Mr. Subeesh A, Scientist (Computer Applications), ICAR- Central Institute of Agricultural Engineering
02:00 pm	Introduction to Galaxy	Dr. Prashanth N Suravajhala, Principal Scientist, School of Biotechnology, Amrita Vishwa Vidyapeetham
<i>Post-workshop evaluation</i>		
Concluding Session		
04:00 pm	Feedback by participants	
04:15 pm	Concluding Remarks	Dr. Prasath D, HRD Nodal Officer, ICAR-Indian Institute of Spices Research
04:20 pm	Vote of Thanks	Ms. Sona Charles

## LIST OF RESOURCE PERSONS INVOLVED IN ONLINE TRAINING

Sl. No.	Name	Designation	Affiliation	email
<b>External Resource Persons</b>				
1	Dr. Santhosh J Eapen	Former Director	ICAR-Indian Institute of Spices Research, Kerala	<a href="mailto:santhosh.eapen@icar.gov.in">santhosh.eapen@icar.gov.in</a>
2	Dr. Merlin Lopez	Scientist	Community Agrobiodiversity Centre, MS Swaminathan Research Foundation, Wayanad, Kerala	merlinettizha@gmail.com
3	Mr. Subeesh A	Scientist	Computer Applications, ICAR- Central Institute of Agricultural Engineering, Bhopal, Madhya Pradesh, India	subeesh.a@icar.gov.in
4	Dr. Prashanth N Suravajhala	Principal Scientist	School of Biotechnology, Amrita Vishwa Vidyapeetham, Kollam, Kerala	prash@am.amrita.edu
<b>Internal Resource Persons</b>				
1	Ms. Sona Charles	Scientist	ICAR-Indian Institute of Spices Research, Kerala	<a href="mailto:sona.charles@icar.gov.in">sona.charles@icar.gov.in</a>
2	Mr. S Mukesh Sankar	Scientist	ICAR-Indian Institute of Spices Research, Kerala	<a href="mailto:mukesh.genetics@gmail.com">mukesh.genetics@gmail.com</a>
3	Mr. Jayarajan K	Chief Technical Officer	ICAR-Indian Institute of Spices Research, Kerala	Jayarajan.K@icar.gov.in
4	Mr. Fayad M.A	Research Scholar	ICAR-Indian Institute of Spices Research, Kerala	

## Contents

<b>S.No.</b>	<b>Title</b>	<b>Page No.</b>
1	Coding for decoding secret of life	6
2	Utilities in Bioinformatics	9
3	Introduction to R	26
4	Data Visualization using R	41
5	Introduction to Linux	50
6	Introduction to Python	54
7	Introduction to Galaxy	64

**Topic 1:**

**Coding for decoding secret of life**

Dr. Santhosh J Eapen  
Former Director, ICAR-Indian Institute of Spices Research  
Email: santhosh.Eapen@icar.gov.in

The field of Bioinformatics emerged in 1980s as a sub-discipline of biology and computer science concerned with the acquisition, storage, analysis, and dissemination of biological data, most often DNA and amino acid. Advancements in sequencing technologies mainly Next Generation Sequencing has resulted in huge escalation in the number of genomes from few to many thousands during the past 25 years (Figure 1). Such cumulative Genome Sequencing and the corresponding explosion of DNA sequencing data has necessitated computer databases that feature rapid assimilation, usable formats and algorithm software programs for efficient management of biological data. A fundamental activity in Bioinformatics is sequence analysis of DNA and proteins with the help of various programs and databases. Bioinformatics uses computer programs for a variety of applications, including determining gene and protein functions, establishing evolutionary relationships, and predicting the three-dimensional shapes of proteins.

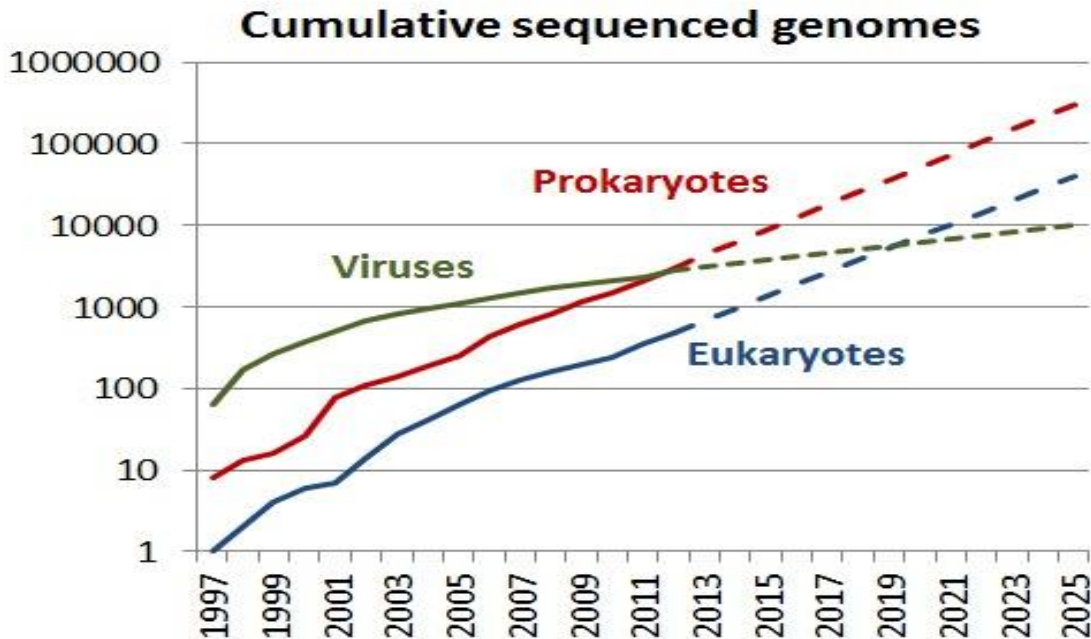


Figure 1

Because of the diverse nature of emerging data, no single comprehensive database exists for accessing all this information. However, a growing number of databases that contain helpful information for clinicians and researchers are available. Information provided by most of these databases is free of charge to academics, although some sites require subscription and industrial users pay a license fee for particular sites. Examples range from sites providing comprehensive descriptions of clinical disorders, listing disease susceptibility genetic mutations and polymorphisms, to those enabling a search for disease genes given a DNA sequence.

Recent technological advances allow for high throughput profiling of biological systems in a cost-efficient manner. The low cost of data generation is leading us to the “big data” era. The availability of big data provides unprecedented opportunities but also raises new challenges for data mining and analysis.

### **Applications of Bioinformatics**

- Bioinformatics plays a vital role in the areas of structural genomics, functional genomics, and nutritional genomics.
- It covers emerging scientific research and the exploration of proteomes from the overall level of intracellular protein composition (protein profiles), protein structure, protein-protein interaction, and unique activity patterns (e.g. post-translational modifications).
- Bioinformatics is used for transcriptome analysis where mRNA expression levels can be determined.
- Bioinformatics is used to identify and structurally modify a natural product, to design a compound with the desired properties and to assess its therapeutic effects, theoretically.
- Chemoinformatics analysis includes analyses such as similarity searching, clustering, QSAR modeling, virtual screening, etc.
- Bioinformatics is playing an increasingly important role in almost all aspects of drug discovery and drug development.
- Bioinformatics tools are very effective in prediction, analysis and interpretation of clinical and preclinical findings.



Anyone, from clinicians to molecular biologists, with access to the internet and relevant websites can now freely discover the composition of biological molecules such as nucleic acids and proteins by using basic bioinformatic tools. This does not imply that handling and analysis of raw genomic data can easily be carried out by all. Bioinformatics is an evolving discipline, and expert bioinformaticians now use complex software programs for retrieving, sorting out, analyzing, predicting, and storing DNA and protein sequence data.

### **Skill-sets required in Bioinformatics**

In bioinformatics and data analysis among the various skill sets required, knowledge in statistics and programming are the most important ones. Bioinformatics programming skills are becoming a necessity across many facets of biology and medicine, owed in part to the continuing explosion of biological data aggregation and the complexity and scale of questions now being addressed through modern bioinformatics. Although many are now receiving formal training in bioinformatics through various university degree and certificate programs, this training is often focused strongly on bioinformatics methodology, leaving many important and practical aspects of bioinformatics to self-education and experience.

## Topic 2:

## Utilities in Bioinformatics

Ms. Sona Charles  
Scientist (Bioinformatics),  
ICAR-Indian Institute of Spices Research, Kozhikode, Kerala.  
Email: [sona.charles@icar.gov.in](mailto:sona.charles@icar.gov.in)

### Common File Formats

#### 1. FASTA

File extensions : file.fa, file.fasta, file.fsa

```
>XR_002086427.1 Candida albicansSC5314 uncharacterized ncRNA (SCR1), ncRNA  
TGGCTGTGATGGCTTTTACGCGGAAGCGCGCTGTTTCGCGTACCTGCTGTTTGTTGAAAA  
TTTAAGAGCAAAGTGTCCGGCTCGATCCCTGCGAATTGAATTCTGAACGCTAGAGTAA  
TCAGTGTCTTTCAAGTTCTGGTAATGTTTAGCATAACCACTGGAGGGAAGCAATTCAG  
CACAGTAATGCTAATCGTGGTGGAGGCGAATCCGGATGGCACCTTGTTTGTTGATAAA  
TAGTGCGGTATCTAGTGTTGCAACTCTATTTTT
```

Fasta format is a simple way of representing nucleotide or amino acid sequences of nucleic acids and proteins. This is a very basic format with two minimum lines. First line referred as comment line starts with '>' and gives basic information about sequence. There is no set format for comment line. Any other line that starts with ';' will be ignored. Lines with ';' are not a common feature of fasta files. After comment line, sequence of nucleic acid or protein is included in standard one letter code. Any tabulators, spaces, asterisks etc in sequence will be ignored.

#### 2. FASTQ

File extensions : file.fastq, file.sanfastq, file.fq

```
@K00188:208:HFLNGBBXX:3:1101:1428:1508 2:N:0:CTTGTA  
ATAATAGGATCCCTTTTCCTGGAGCTGCCTTITAGGTAATGTAGTATCTNATNGACTGN  
CNCCANANGGCTAAAGT  
+  
AAAFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ#FJ#JJJF#F#FJJ#F#JJFJ  
JJJJ
```

Fastq format was developed by Sanger institute in order to group together sequence and its quality scores (Q: phred quality score). In fastq files each entry is associated with 4 lines.

Line 1 begins with a '@' character and is a sequence identifier and an optional description.

K00185	the unique instrument name
208	the run id
HF1.NGBBXX	the flowcell id
3	flowcell lane
1101	tile number within the flowcell lane
1428	'x'-coordinate of the cluster within the tile
1508	'y'-coordinate of the cluster within the tile
2	the member of a pair, 1 or 2 (paired-end or mate-pair reads only)
N	Y if the read is filtered, N otherwise
0	0 when none of the control bits are on, otherwise it is an even number
CTTGTA	index sequence or Barcode

Line 2 Sequence in standard one letter code.

DNA Sequence

Line 3 begins with a '+' character and is optionally followed by the same sequence identifier (and any additional description) again.

No comment added

Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

A quality score (PHRED scale) for each base pair. It indicates how confident we can be that the base was sequenced and identified correctly.

$$Q = -10\log_{10}(p)$$

where p is the probability that the corresponding base call is incorrect.

Phred score	quality	Probability that the base is called wrong	Accuracy of the base call
10		1 in 10	90%
20		1 in 100	99%
30		1 in 1000	99.90%
40		1 in 10000	99.99%
50		1 in 100000	100.00%

Fastq-sanger holds PHRED score from 0-93 whereas fastq-Illumina provides PHRED scores from 0-62. Rather than giving numeric values of PHRED score they are provided in ASCII character codes from 33 to 126.

Val	Char	Val	Char	Val	Char	Val	Char	Val	Char
33	!	53	5	73	I	93	]	113	q
34	"	54	6	74	J	94	^	114	R
35	#	55	7	75	K	95	_	115	S
36	\$	56	8	76	L	96	`	116	T
37	%	57	9	77	M	97	a	117	U
38	&	58	:	78	N	98	b	118	V
39	'	59	;	79	O	99	c	119	W
40	(	60	<	80	P	100	d	120	X
41	)	61	=	81	Q	101	e	121	Y
42	*	62	>	82	R	102	f	122	Z
43	+	63	?	83	S	103	g	123	{
44	,	64	@	84	T	104	h	124	
45	-	65	A	85	U	105	i	125	}
46	.	66	B	86	V	106	j	126	~
47	/	67	C	87	W	107	k		
48	0	68	D	88	X	108	l		
49	1	69	E	89	Y	109	m		
50	2	70	F	90	Z	110	n		
51	3	71	G	91	[	111	o		
52	4	72	H	92	\	112	p		

Letter	Quality	Estimated probability	error
(	40	20%	
7	55	0.6%	
F	70	0.02%	
U	85	0.0006%	
d	100	0.00002%	

### 3. SAM

File extensions :file.sam

The SAM Format is a text format for storing sequence data in a series of tab delimited ASCII columns. Most often it is generated as a human readable version of its sister BAM format, which stores the same data in a compressed, indexed, binary form.

SAM format files are generated following mapping of the reads to reference sequence. It is TAB-delimited text format with header and a body. Header lines start with '@' while alignment lines do not. Header hold generic information on SAM file along with version

information, if the file is sorted, information on reference sequence, etc. The alignment records constitute the body of the file. Each alignment line/record has 11 mandatory fields describing essential alignment information.

Template: The DNA fragment that was measured

Reads: Depending on the methodology a template may produce one or more reads. These reads may cover the entire template or just a subsection of it. Reads originating from the same template typically cover different parts of the template, and, may represent the template itself or the reverse complement of it.

Segments: Each read may produce one or more alignments that in turn will have aligned regions called segments. From these segments it may be possible to infer the size of the original template.

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	[!-?A-~]{1,254}	Query template NAME
2	FLAG	Int	[0,2 <sup>16</sup> -1]	bitwise FLAG
3	RNAME	String	\*[!-( )+<>-~][!-~]*	Reference sequence NAME
4	POS	Int	[0,2 <sup>31</sup> -1]	1-based leftmost mapping POSition
5	MAPQ	Int	[0,2 <sup>8</sup> -1]	MAPping Quality
6	CIGAR	String	\* ([0-9]+[MIDNSHPX=])+	CIGAR string
7	RNEXT	String	\* = [!-( )+<>-~][!-~]*	Ref. name of the mate/next read
8	PNEXT	Int	[0,2 <sup>31</sup> -1]	Position of the mate/next read
9	TLEN	Int	[-2 <sup>31</sup> +1,2 <sup>31</sup> -1]	observed Template LENgth
10	SEQ	String	\*[A-Za-z=.]+	segment SEQuence
11	QUAL	String	[!-~]+	ASCII of Phred-scaled base QUALity+33

Col. 1 QNAME:

Query NAME. Reads/segments having identical QNAME are regarded to come from the same template. A QNAME "\*" indicates the information is unavailable. In a SAM file, a read may occupy multiple alignment lines, when its alignment is chimeric .

Col. 2 FLAG:

Combination of bitwise flags.

BIT		Description
1	0x1	template having multiple segments in sequencing
2	0x2	each segment properly aligned according to the aligner
4	0x4	segment unmapped
8	0x8	next segment in the template unmapped

16	0x10	SEQ being reverse complemented
32	0x20	SEQ of the next segment in the template being reverse complemented the first segment in the template
64	0x40	the first segment in the template
128	0x80	the last segment in the template
256	0x100	secondary alignment
512	0x200	not passing filters, such as platform/vendor quality controls
1024	0x400	PCR or optical duplicate
2048	0x800	supplementary alignment

**Col. 3 RNAME:**

Name of reference sequence. It generally refers to chromosome number.

**Col. 4 POS:**

Leftmost mapping position of the first matching base in read. It has 1-based indexing. If pos is set as 0, it represents a unmapped read. For a read pair READ1/1 and READ1/2 and single Read2

Reference:	
GTACGACTGACTAGACGATAC**GTAACGATCAGTCTCGATAGCCGATAGGCTAGCTAGCAGCTAGCAG	
12345678901234567890123456789012345678901234567890123456789012345678	
Read1/1	ACGACTGA
-Read1/2	CGATAGC
Read2ccccGATACTAGTAA*GAT..GTCT	
Pos values	3 38

**Col. 5 MAPQ:**

It indicates MAPping Quality. MAPQ=  $-10\log_{10}(\text{Probability of mapping position being wrong})$ . MAPQ=255 indicates mapping quality is unavailable.

**Col. 6 CIGAR:**

A string that describes alignment.

OP	BAM	Description
M	0	alignment match (can be a sequence match or mismatch)
I	1	insertion to the reference
D	2	deletion from the reference
N	3	skipped region from the reference
S	4	soft clipping (clipped sequences present in SEQ)
H	5	hard clipping (clipped sequences NOT present in SEQ)
P	6	padding (silent deletion from padded reference)
=	7	sequence match
X	8	sequence mismatch

Difference between H and S is that if the mismatch sequence is reported as part of read sequence in alignment file it is a soft clipping. Often mismatch region matches somewhere else in reference sequence and in that case the mismatch region is removed from reported read sequence in alignment and is referred as Hard clipping.

Col. 7 RNEXT , Col. 8 PNEXT:

RNEXT and PNEXT is to know the reference and position of a paired end read's partner for visualisation tools. RNEXT is the name of the chromosome or contig to which the next template in a pair aligns. RNEXT of value '=' means align to same reference and '\*' represent no information available (single end sequencing). PNEXT where the other read of the pair aligns (Information unavailable =0, Otherwise POS value of pair).

Read1/1 and Read1/2 are pair and Read 3 is unpaired. So the RNEX and PNEXT values will be

READ	RNEXT	PNEXT	TLEN
Read1/1	=	38	42
Read1/2	=	3	-42
Read3	*	0	0

**Col. 9 TLEN** : Observed Template LENgth

It represents the length of reference that is covered by pair end reads. The distance between leftmost mapped base to rightmost mapped base in paired reads. For unpaired reads it is 0.

**Col. 10 SEQ:** Sequence of the read or Segment.

**Col. 11 QUAL:** PHRED score values of read. If '\*' no values are stored.

```
Coord      12345678901234 5678901234567890123456789012345
ref        AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCCAGTCAGCGCCAT

+r001/1    TTAGATAAAGGATA*CTG
+r002      aaaAGATAA*GGATA
+r003      gcctaAGCTAA
+r004      ATAGCT.....TCAGC
-r003      ttagctTAGGC
-r001/2    CAGCGGCAT

The corresponding SAM format is:1
Version
@HD VN:1.5 SO:coordinate
@SQ SN:ref LN:45
r001 99 ref 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
r002 0 ref 9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA *
r003 0 ref 9 30 5S6M * 0 0 GCCTAAGCTAA * SA:Z:ref,29,-,6H5M,17,0;
r004 0 ref 16 30 6M14N5M * 0 0 ATAGCTTCAGC *
r003 2064 ref 29 17 6H5M * 0 0 TAGGC * SA:Z:ref,9,+,5S6M,30,1;
r001 147 ref 37 30 9M = 7 -39 CAGCGGCAT * NM:i:1
QNAME FLAG RNAME POS MAPQ CIGAR RNEXT PNEXT TLEN SEQ QUAL
```

#### 4. BAM

File extensions :file.bam

A BAM (Binary Alignment/Map) file is the compressed binary version of the Sequence Alignment/Map (SAM), a compact and indexable representation of nucleotide sequence alignments. The data between SAM and BAM is exactly same. Being Binary BAM files are small in size and ideal to store alignment files. Require samtools to view the file.

#### 5. VCF

File extensions : file.vcf

```
##fileformat=VCFv4.2
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66
beb2da,species="Homo sapiens",taxonomy=x>
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With
Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build
```



```

129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA00001 NA00002
NA00003
20 14370 rs6054257 G A 29 PASS NS=3;DP=14;AF=0.5;DB;H2 GT:GQ:DP:HQ
0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,.
20 17330 . T A 3 q10 NS=3;DP=11;AF=0.017 GT:GQ:DP:HQ 0|0:49:3:58,50
0|1:3:5:65,3 0/0:41:3
20 1110696 rs6040355 A G,T 67 PASS NS=2;DP=10;AF=0.333,0.667;AA=T;DB
GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2 2/2:35:4
20 1230237 .T . 47 PASS NS=3;DP=13;AA=T GT:GQ:DP:HQ 0|0:54:7:56,60
0|0:48:4:51,51 0/0:61:2
20 1234567 microsat1 GTC G,GTCT 50 PASS NS=3;DP=9;AA=G GT:GQ:DP 0/1:35:4
0/2:17:2 1/1:40:3

```

VCF is a text file format with a header (information VCF version, sample etc) and data lines constitute the body of file.

#### HEADER:

This contains meta-information and is included after ‘##’ string. It is recommended to include INFO, FILTER and FORMAT entries for a better explanation of the data field.

Metadata format:

```

##INFO=<ID=ID,Number=number,Type=type,Description="description",Source="source",Version="version">
##FILTER=<ID=ID,Description="description">##FORMAT=<ID=ID,Number=number,Type=type,Description="description">

```

Other information like alternate allele, assembly field, Contig field, sample field, pedigree field can also be included.

#### DATA FIELD:

Data lines have 8 mandatory columns.

#CHROM, POS, ID, REF, ALT, QUAL, FILTER, INFO.

#### 6. GFF

File extensions : file.gff2, file.gff3, file.gff

#### GFF2:

```

browser position chr22:10000000-10025000
browser hide all
track name=regulatory description="TeleGene(tm) Regulatory Regions"

```

```
visibility=2
```

```
chr22TeleGene enhancer 10000000 10001000 500 + . touch1
```

```
chr22TeleGene promoter 10010000 10010100 900 + . touch1
```

```
chr22TeleGene promoter 10020000 10025000 800 - . touch2
```

### **GFF3:**

It has first 8 fields like GFF2 but differs in field 9 in assigning attributes. 2 are highlighted here.

(a) GFF3 has better nesting feature. Links features to parent tag

```
##gff-version 3
```

```
ctg123 .mRNA      1300 9000 . + . ID=mrna0001;Name=sonichedgehog
```

```
ctg123 .exon     1300 1500 . + . ID=exon00001;Parent=mrna0001
```

```
ctg123 .exon     1050 1500 . + . ID=exon00002;Parent=mrna0001
```

```
ctg123 .exon     3000 3902 . + . ID=exon00003;Parent=mrna0001
```

```
ctg123 .exon     5000 5500 . + . ID=exon00004;Parent=mrna0001
```

```
ctg123 .exon     7000 9000 . + . ID=exon00005;Parent=mrna0001
```

(b) The most general way of representing a protein-coding gene is the so-called “three-level gene.” The top level is a feature of type “gene” which bundles up the gene’s transcripts and regulatory elements. Beneath this level are one or more transcripts of type “mRNA”. This level can also accommodate promoters and other cis-regulatory elements. At the third level are the components of the mRNA transcripts, most commonly CDS coding segments and UTRs. This example shows how to represent a gene named “EDEN” which has three alternatively-spliced mRNA transcripts:

```
ctg123 example gene      1050 9000 . + . ID=EDEN;Name=EDEN;Note=protein kinase
```

```
ctg123  example  mRNA                                1050  9000  .  +  .  
ID=EDEN.1;Parent=EDEN;Name=EDEN.1;Index=1
```

```
ctg123 example five_prime_UTR 1050 1200 . + . Parent=EDEN.1
```

```
ctg123 example CDS      1201 1500 . + 0 Parent=EDEN.1
```

```
ctg123 example CDS      3000 3902 . + 0 Parent=EDEN.1
```

```

ctg123 example CDS      5000 5500 . + 0 Parent=EDEN.1
ctg123 example CDS      7000 7608 . + 0 Parent=EDEN.1
ctg123 example three_prime_UTR 7609 9000 . + . Parent=EDEN.1

ctg123 example mRNA      1050 9000 . + .
ID=EDEN.2;Parent=EDEN;Name=EDEN.2;Index=1
ctg123 example five_prime_UTR 1050 1200 . + . Parent=EDEN.2
ctg123 example CDS      1201 1500 . + 0 Parent=EDEN.2
ctg123 example CDS      5000 5500 . + 0 Parent=EDEN.2
ctg123 example CDS      7000 7608 . + 0 Parent=EDEN.2
ctg123 example three_prime_UTR 7609 9000 . + . Parent=EDEN.2

ctg123 example mRNA      1300 9000 . + .
ID=EDEN.3;Parent=EDEN;Name=EDEN.3;Index=1
ctg123 example five_prime_UTR 1300 1500 . + . Parent=EDEN.3
ctg123 example five_prime_UTR 3000 3300 . + . Parent=EDEN.3
ctg123 example CDS      3301 3902 . + 0 Parent=EDEN.3
ctg123 example CDS      5000 5500 . + 1 Parent=EDEN.3
ctg123 example CDS      7000 7600 . + 1 Parent=EDEN.3
ctg123 example three_prime_UTR 7601 9000 . + . Parent=EDEN.3

```

GFF (General Feature Format or Gene Finding Format). GFF can be used for any kind of feature (Transcripts, exon, intron, promoter, 3' UTR, repetitive elements etc) associated with the sequence, whereas GTF is primarily for genes/transcripts. GFF3 is the latest version and an improvement over GFF2 format. However, many databases are still not equipped to handle GFF3 version. The differences will be explained later in text.

The GFF format has 9 mandatory columns and they are TAB separated. The 9 columns are as follows.

Col. 1 Reference Sequence:

This is the ID of reference sequence used to establish coordinate system for annotation. Usually chromosome name or number.

Col. 2 Source:

This explains how the feature annotation is derived. The source is a free text qualifier intended to describe the algorithm or operating procedure that generated this feature. Typically this is the name of a piece of software, such as “Genescan” or a database name, such as “Genbank.” In effect, the source is used to extend the feature ontology by adding a qualifier to the type creating a new composite type that is a subclass of the type in the type column. It is not necessary to specify a source. If there is no source, put a “.” (a period) in this field.

Col. 3 Feature:

The feature type name, like “gene” or “exon”. In a well-structured GFF file, all the children(exons, introns etc) features always follow their parents(Transcript) feature line. This way they are part of a single block

Col. 4 Start:

Genomic Start of the feature.

Col. 5 End:

Genomic Start of the feature

Col. 6 Score:

Numeric value that generally indicates the confidence of the source on the annotated feature. A value of “.” (a dot) is used to define a null value. The semantics of the score are ill-defined. In GFF3format,It is strongly recommended that E-values be used for sequence similarity features, and that P-values be used for ab initio gene prediction features. If there is no score, put a “.” (a period) in this field.

Col. 7 Strand:

Field that indicates the sense strand of the feature. “+” :Watson strand and “-”: crick strand. “?” can be used for features whose strandedness is relevant, but unknown.

Col. 8 Frame (GFF2 and GTF) or Phase (GFF3):

For features of type “CDS”, the phase indicates where the feature begins with reference to the reading frame. The phase is one of the integers 0, 1, or 2, indicating the number of bases that should be removed from the beginning of this feature to reach the first base of the next codon. In other words, a phase of “0” indicates that the next codon begins at the first base of the region described by the current line, a phase of “1” indicates that the next codon begins at the second base of this region, and a phase of “2” indicates that the codon begins at the third base of this region. This is NOT to be confused with the frame, which is simply start modulo 3. If there is no phase, put a “.” (a period) in this field.

For forward strand features, phase is counted from the start field. For reverse strand features, phase is counted from the end field.

The phase is required for all CDS features.

Explained let say ### and \*\*\* represent consecutive exons.

CTG C is first base (0), T is second base (1), G is third base(2)

Col. 9 Attribute or Group field:

All lines with the same group are linked together into a single item. The group field is a challenge. It is used in several distinct ways:

- to group together a single sequence feature that spans a discontinuous range, such as a gapped alignment.
- to name a feature, allowing it to be retrieved by name.
- to add one or more notes to the annotation.
- to add an alternative name

Problems with GFF2:

One of GFF2's problems is that it is only able to represent one level of nesting of features. This is mainly a problem when dealing with genes that have multiple alternatively-spliced transcripts. GFF2 is unable to deal with the three-level hierarchy of gene → transcript → exon. Most people get around this by declaring a series of transcripts and giving them similar names to indicate that they come from the same gene.

The second limitation is that while GFF2 allows you to create two-level hierarchies, such as transcript → exon, it doesn't have any concept of the direction of the hierarchy. So it doesn't know whether the exon is a subfeature of the transcript, or vice-versa. This means you have to use "aggregators" to sort out the relationships. This is a major pain in the neck. For this reason, GFF2 format has been deprecated in favor of GFF3 format databases.

## 7. GTF

File extensions : file.gtf

```
AB000381Twinscan exon      150  200  .  +  .  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscan exon      300  401  .  +  .  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscan CDS       380  401  .  +  0  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscan exon      501  650  .  +  .  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscan CDS       501  650  .  +  2  gene_id "AB000381.000";
```

```

transcript_id "AB000381.000.1";

AB000381Twinscan exon      700  800  .  +  .  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscan CDS      700  707  .  +  2  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscan exon      900 1000  .  +  .  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscanstart_codon 380  382  .  +  0  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

AB000381Twinscanstop_codon  708  710  .  +  0  gene_id "AB000381.000";
transcript_id "AB000381.000.1";

```

GTF has the same format as GFF files. It has the same 9 fields that describe the gene/transcript related features. The group/attribute field has been expanded into a list of attributes. Each attribute consists of a type/value pair. Attributes must end in a semi-colon, and be separated from any following attribute by exactly one space. The attribute list must begin with the two mandatory attributes:

**gene\_id value:** A globally unique identifier for the genomic source of the sequence.

**transcript\_id value:** A globally unique identifier for the predicted transcript.

### **Gene Ontology: tool for the unification of biology**

The Gene Ontology resource (GO; <http://geneontology.org>) is the most comprehensive and widely used knowledgebase concerning the functions of genes. In GO, all functional knowledge is structured and represented in a form amenable to computational analysis, which is essential to support modern biological research. The GO knowledgebase is structured using a formal ontology, by defining classes of gene functions (GO terms) that have specified relations to each other. GO terms are often given logical definitions, or equivalence axioms, that define the term relative to other terms in the GO or other ontologies, so that their relationships can be computationally inferred using logical reasoning. The GO structure has been meticulously constructed over the course of 20 years by a small team of ontology developers; it is constantly evolving in response to new scientific discoveries and continuously refined to represent the most current state of biological knowledge. The members of the ontology development team are expert biologists and knowledge representation specialists who read the scientific literature and engage biocurators and biological domain experts to collaboratively develop this representation of biological information.

Biological process refers to a biological objective to which the gene or gene product contributes. A process is accomplished via one or more ordered assemblies of molecular functions. Processes often involve a chemical or physical transformation, in the sense

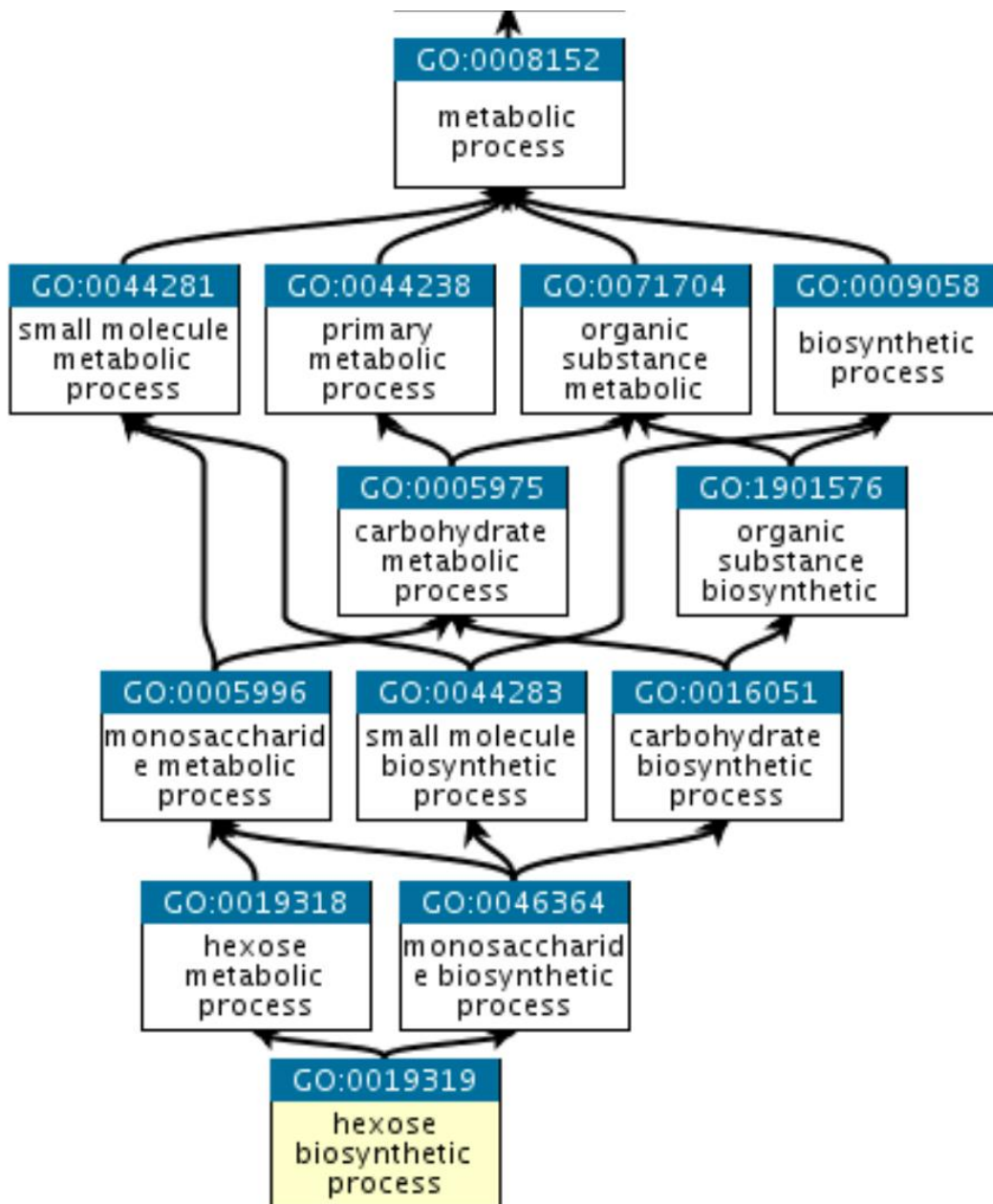
that something goes into a process and something different comes out of it. Examples of broad (high level) biological process terms are 'cell growth and maintenance' or 'signal transduction'. Examples of more specific (lower level) process terms are 'translation', 'pyrimidine metabolism' or 'cAMP biosynthesis'.

Molecular function is defined as the biochemical activity (including specific binding to ligands or structures) of a gene product. This definition also applies to the capability that a gene product (or gene product complex) carries as a potential. It describes only what is done without specifying where or when the event actually occurs. Examples of broad functional terms are 'enzyme', 'transporter' or 'ligand'. Examples of narrower functional terms are 'adenylate cyclase' or 'Toll receptor ligand'.

Cellular component refers to the place in the cell where a gene product is active. These terms reflect our understanding of eukaryotic cell structure. As is true for the other ontologies, not all terms are applicable to all organisms; the set of terms is meant to be inclusive. Cellular component includes such terms as 'ribo-some' or 'proteasome', specifying where multiple gene products would be found. It also includes terms such as 'nuclear membrane' or 'Golgi apparatus'.

### The GO Graph

The structure of GO can be described in terms of a graph, where each GO term is a node, and the relationships between the terms are edges between the nodes. GO is loosely hierarchical, with 'child' terms being more specialized than their 'parent' terms, but unlike a strict hierarchy, a term may have more than one parent term (note that the parent/child model does not hold true for all types of relations, see the relations documentation). For example, the biological process term hexose biosynthetic process has two parents, hexose metabolic process and monosaccharide biosynthetic process. This reflects the fact that biosynthetic process is a subtype of metabolic process and a hexose is a subtype of monosaccharide.



## Gene Set Enrichment Analysis

GSEA is an algorithm that performs differential expression analysis at the level of gene sets (Subramanian et al., 2005). The input to GSEA consists of a collection of gene sets and microarray expression data with replicates for two conditions to be compared. GSEA employs a permutation-based test which uses Kolmogorov–Smirnov running sum statistic to determine which of the gene sets from the collection are differentially expressed between the two conditions. GSEA differs from differential gene expression analysis in the sense that it might identify genes which are part of a differentially expressed set but which might not be identified as significantly differentially expressed alone.

The GSEA approach is best-suited to a comparison-based analysis between two classes of data, multiple scenarios of data points or a single dataset across a series of



timepoints. This approach requires a biological metric (e.g., Fold-change or expression) for each biomolecule of interest in order to rank them. A maximum enrichment score (MES) for all genes in a particular category is generated as an output of the GSEA analysis. Once MES scores are compiled, a p-valuebased assessment is done by comparing the ranked MES to the randomly generated MES distributions to assess for enrichment for particular terms beyond what would be expected due to chance.

Main steps involved in GSEA are:

- a. Calculation of enrichment score (ES): ES represents degree to which gene set is overrepresented at the extremes (top or bottom) of the entire ranked list. This score corresponds to weighted Kolmogorov–Smirnov like statistics.
- b. Estimation of significance level of ES: Statistical significance is estimated by empirical phenotypic based permutation test procedures in order to generate a null distribution for the ES.
- c. Adjustment for multiple hypothesis testing: When a large number of gene sets are being analyzed at one time, ES for each gene set is normalized, and false discovery rate is calculated (Subramanian et al., 2005).

## **Integrative Genomics Viewer (IGV)**

The Integrative Genomics Viewer (IGV) is a high-performance, easy-to-use, interactive tool for the visual exploration of genomic data. It supports flexible integration of all the common types of genomic data and metadata, investigator-generated or publicly available, loaded from local or cloud sources. IGV is available in multiple forms, including:

the original IGV - a Java desktop application,

IGV-Web - a web application,

igv.js - a JavaScript component that can be embedded in web pages (for developers)

### *Parts of IGV Browser*

The tool bar provides access to commonly used functions.

The red box on the chromosome ideogram indicates which portion of the chromosome is displayed. When zoomed out to display the full chromosome, the red box disappears from the ideogram.

The ruler reflects the visible portion of the chromosome. The tick marks indicate chromosome locations. The span lists the number of bases currently displayed.

IGV displays data in horizontal rows called tracks. Typically, each track represents one sample or experiment. This example shows segmented copy number data.

IGV also displays features, such as genes, in tracks. By default, IGV displays data in one panel and features in another, as shown here. Drag-and-drop a track name to move a track from one panel to another.

Track names are listed in the far left panel. Legibility of the names depends on the height of the tracks; i.e., the smaller the track the less legible the name.

An optional attribute panel displays sample/track attributes represented as colored blocks, where each unique value is assigned a unique color.

## **UCSC Genome Browser**

The UCSC Genome Browser provides a rapid and reliable display of any requested portion of genomes at any scale, together with dozens of aligned annotation tracks (known genes, predicted genes, ESTs, mRNAs, CpG islands, assembly gaps and coverage, chromosomal bands, mouse homologies, and more). Half of the annotation tracks are computed at UCSC from publicly available sequence data. The remaining tracks are provided by collaborators worldwide. Users can also add their own custom tracks to the browser for educational or research purposes.

The Genome Browser stacks annotation tracks beneath genome coordinate positions, allowing rapid visual correlation of different types of information. The user can look at a whole chromosome to get a feel for gene density, open a specific cytogenetic band to see a positionally mapped disease gene candidate, or zoom in to a particular gene to view its spliced ESTs and possible alternative splicing. The Genome Browser itself does not draw conclusions; rather, it collates all relevant information in one location, leaving the exploration and interpretation to the user.

The Genome Browser supports text and sequence based searches that provide quick, precise access to any region of specific interest. Secondary links from individual entries within annotation tracks lead to sequence details and supplementary off-site databases. To control information overload, tracks need not be displayed in full. Tracks can be hidden, collapsed into a condensed or single-line display, or filtered according to the user's criteria. Zooming and scrolling controls help to narrow or broaden the displayed chromosomal range to focus on the exact region of interest. Clicking on an individual item within a track opens a details page containing a summary of properties and links to off-site repositories such as PubMed, GenBank, Entrez, and OMIM. The page provides item-specific information on position, cytoband, strand, data source, and encoded protein, mRNA, genomic sequence and alignment, as appropriate to the nature of the track.

## Topic 3:

## Introduction to R

Mr. Mukesh Sankar. S  
Scientist (Crop Improvement & Biotechnology),  
ICAR-Indian Institute of Spices Research, Kozhikode, Kerala.  
Email: [mukesh.genetics@gmail.com](mailto:mukesh.genetics@gmail.com)

### General Overview

[R](#) is a comprehensive statistical environment and programming language for professional data analysis and graphical display. The R software is free and runs on all common operating systems such as Windows, MacOS and Linux. The key feature of the environment is that it is open source, rapidly evolving, interactive data analytic platform with large global support system. 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.

### History of R

R can be regarded as an open source implementation of the S language which was developed at Bell Laboratories by Rick Becker, John Chambers and Allan Wilks. R was initially written by Robert Gentleman and Ross Ihaka—also known as “R & R” of the Statistics Department of the University of Auckland. Since mid-1997 there has been a core group, the R Core Team, with write access to the R source, who contributed by donating code, bug fixes and documentation. R is currently the result of a collaborative effort with contributions from all over the world. (Written by statisticians (and some of us) for statisticians (and the rest of us) and available under GNU Copy-left in source code form. R is a group project run by a core group of developers (with new releases semiannually). The current version of R is 4.2.1 has been released on 23.06.2022.

### Packages

R can be extended (easily) via packages (again freely). There are about eight packages supplied with the R distribution (called “standard” and “recommended” packages) and many more are available through the Comprehensive R Archive Network (CRAN) family of Internet sites covering a very wide range of modern statistics. The CRAN main site at WU (Wirtschaftsuniversität Wien) in Austria can be found at the URL, <https://CRAN.R-project.org/> and is mirrored daily to many sites around the world. See <https://CRAN.R-project.org/mirrors.html> for a complete list of mirrors. The CRAN mirror India server is hosted by National Institute of Science Education and Research (NISER), Bhubaneswar.

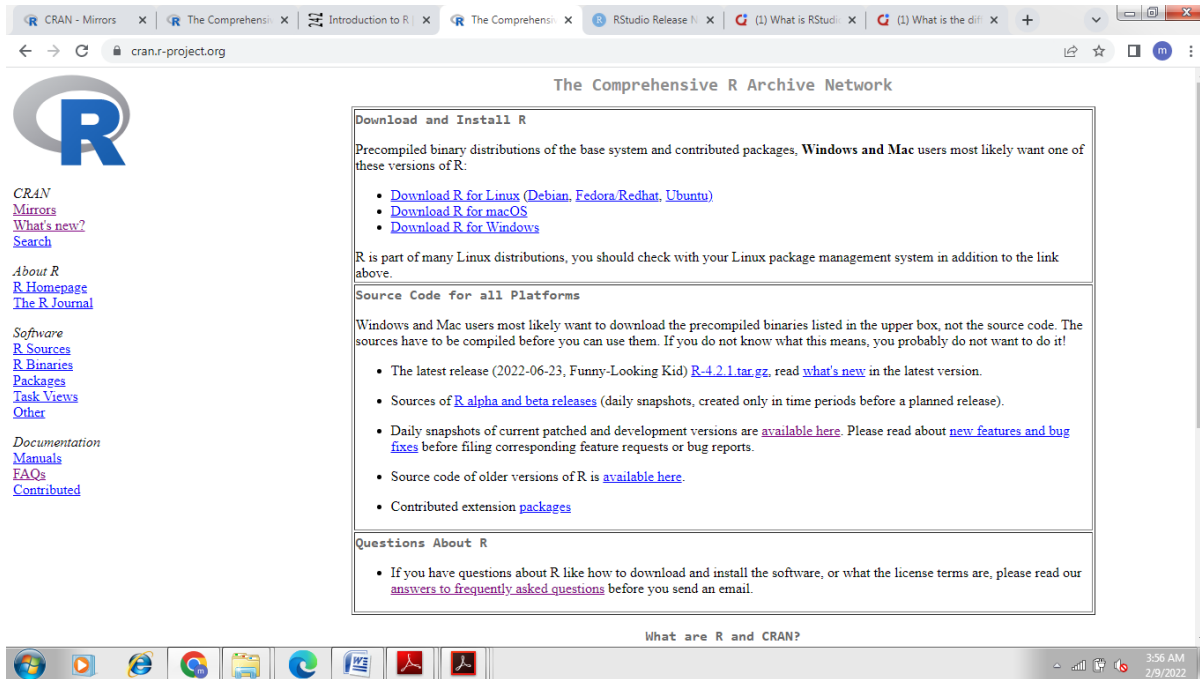
The associated Bioconductor project, (can be accessed via URL, <http://bioconductor.org/>) provides many additional R packages for statistical data analysis in different life science areas, such as tools for microarray, next generation sequence and genome analysis.

### R Package Repositories

- CRAN (>14,000 packages) general data analysis
- Bioconductor (>2,000 packages) bioscience data analysis
- Omegahat (>90 packages) programming interfaces
- RStudio packages

### Downloading and Installation of the Software

Precompiled binary distributions of the base system and contributed packages, Windows and Mac users most likely want one of these versions of R: Linux , MacOS X, Windows.



Download and Installation of R for *Windows* is as follows:

- Visit <http://cran.r-project.org/>
- Browse Windows
- Click on “base” link - Binaries for base distribution (managed by Duncan Murdoch)
- Click “README on the Windows binary distribution” for Installation and other instructions
- Click “Download R-4.2.1 for Windows (79 megabytes, 64 bit)” for downloading R-4.2.1 software
- Once download is complete, run “R--4.2.1-win32.exe”.
- Follow the instructions to install R software.


*For Linux:*

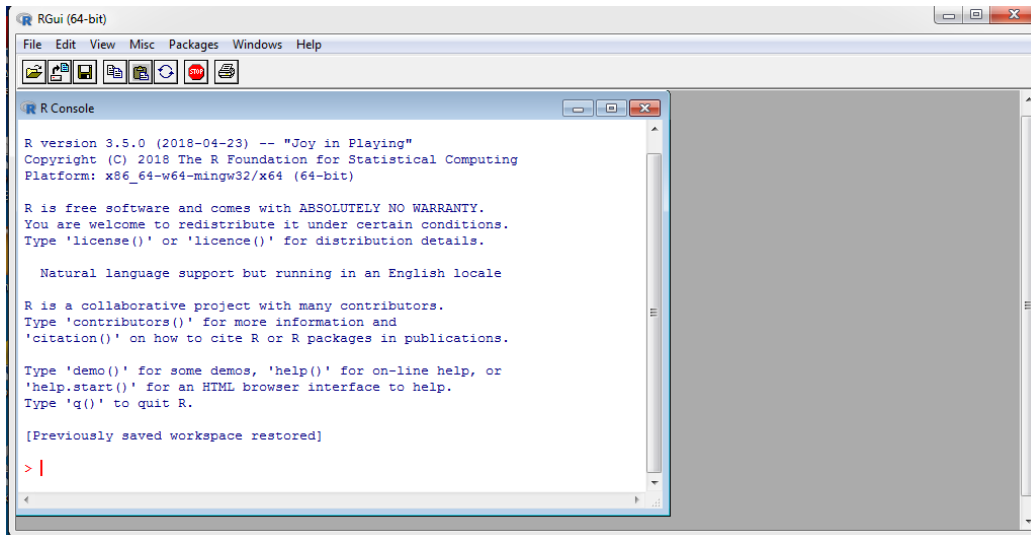
R can be installed on Ubuntu, using the following Bash script:

```
sudo apt-get install r-base
```

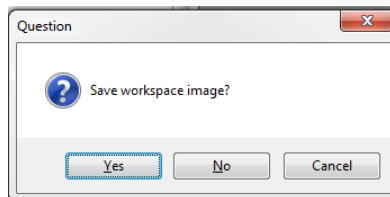
### Invoking R

If properly installed, usually R has a shortcut icon on the desktop screen and/or you can find it under Start

→All Programs→R menu. Click “R” shortcut icon.  A “RGui” based “R Console” will appear.



To quit R, type `q()` at the R prompt (`>`) and press Enter key. A dialog box will ask whether to save the objects you have created during the session so that they will become available next time when R will be invoked.



## RStudio

RStudio is an IDE (integrated development environment), that is used to develop R programs more easily and efficiently. It is also available as open source or commercial editions which forms front end editor for R programming. So it means, RStudio in itself is not very useful without R. Now RStudio can also work well with Python.

### Installation of RStudio

RStudio requires R 3.0.1+ that means R software should be pre-installed before using RStudio.

RStudio 2022.07.1+554 requires a 64-bit operating system, and works exclusively with the 64 bit version of R. If you are on a 32 bit system or need the 32 bit version of R, you can use an older version of RStudio (<https://support.rstudio.com/hc/en-us/articles/206569407-Older-Versions-of-RStudio>).

RStudio free desktop version can be downloaded from the following link:

<https://www.rstudio.com/products/rstudio/download/#download>

### Parts of R Studio

The first time RStudio is opened, three windows are seen. A fourth window is hidden by default, but can be opened by clicking the File drop-down menu, then New File, and then R Script.

#### *The Script editor pane*

The Source Editor can help you open, edit and execute these programs. It is the pane on the top left of your screen.

#### *The R Console Pane*

The R Console is where you can type code that executes immediately. This is also known as the command line. It is at the bottom left of your screen. It is the only part of RStudio that is actually R itself.

### *The R Environment pane*

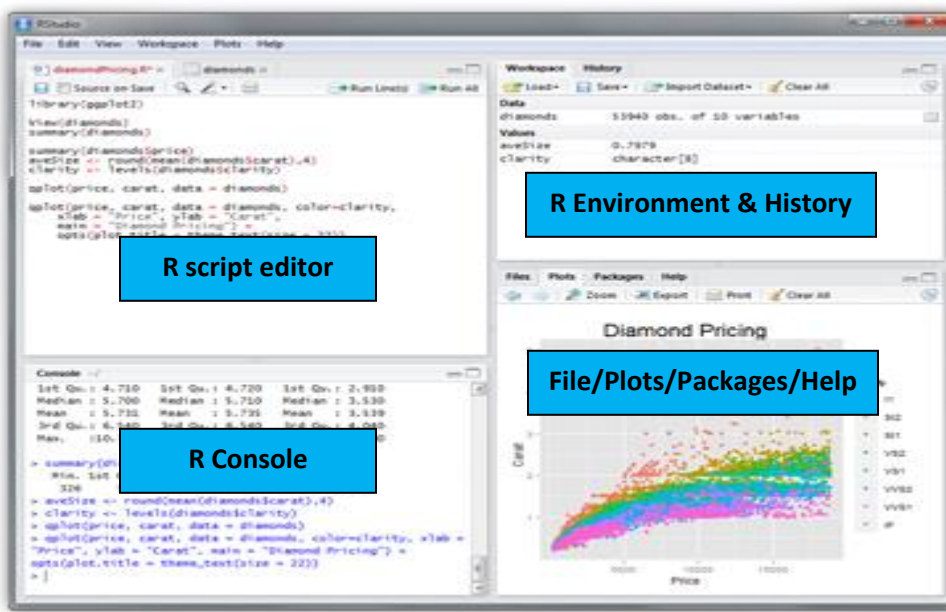
The Environment pane is visible from the top right window as it shows you what objects (i.e., dataframes, arrays, values and functions) you have in your environment (workspace). You can see the values for objects with a single value and for those that are longer, R will tell you their class.

When you have data in your environment that have two dimensions (rows and columns) you may click on them and they will appear in the script editor pane like a spreadsheet. It is at the top right of your screen.

### *Files/Plots/Packages/Help pane*

The last pane appear at bottom right is a basic file browser has a number of different tabs.

- The Files tab has a navigable file manager, just like the file system on your operating system.
- The Plot tab is where graphics you create will appear.
- The Packages tab shows you the packages that are installed and those that can be installed.
- The Help tab allows you to search the R documentation for help and is where the help appears when you ask for it from the Console. It is at the bottom right of your screen.



**View of RStudio IDE**

## **Installation of R Packages**

Install **CRAN Packages** from R console like this:

```
install.packages(c("pkg1", "pkg2"))  
install.packages("pkg.zip", repos=NULL)
```

Install **Bioconductor packages** as follows:

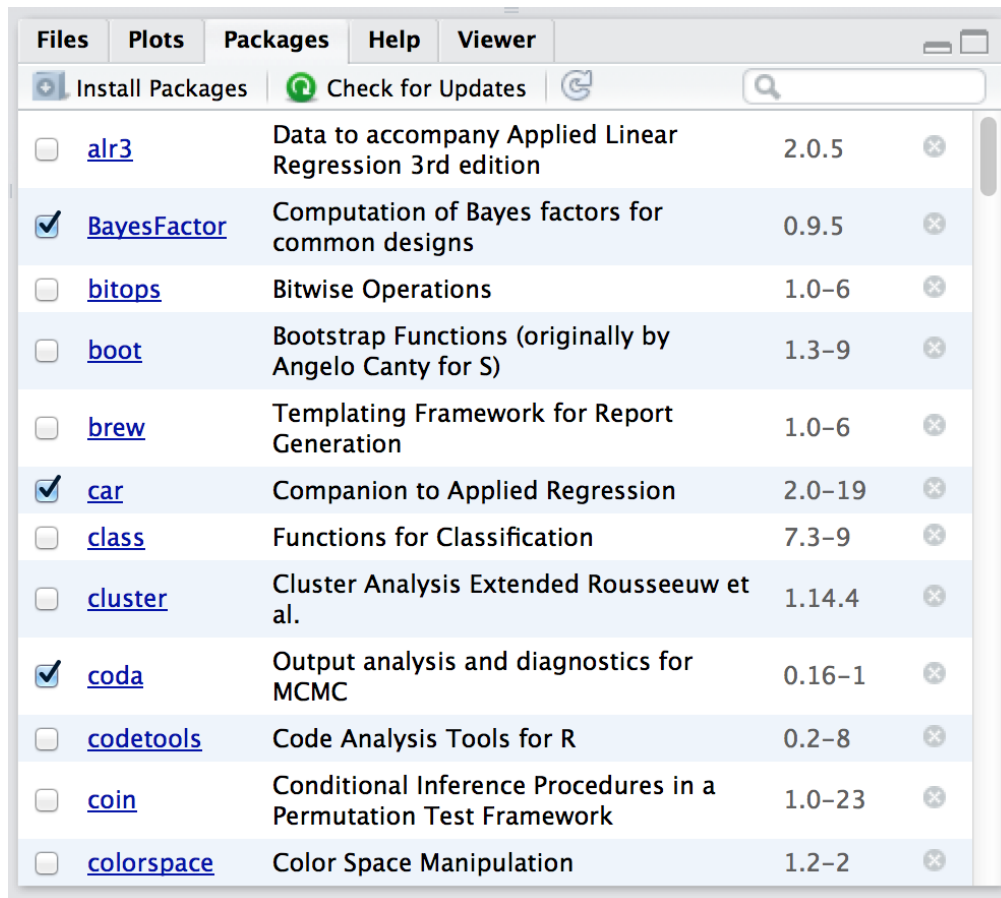
```
if (!requireNamespace("BiocManager", quietly = TRUE))  
install.packages("BiocManager") # Installs BiocManager if not available yet  
BiocManager::version() # Reports Bioconductor version  
BiocManager::install(c("pkg1", "pkg2")) # Installs packages specified under "pkg1"
```

## Loading and Unloading of Packages

R packages are a collection of R functions, compiled code and sample data. They are stored under a directory called "library" in the R environment. All the packages are need not required at all time. Specific package which are required for a specific analysis can be either called by library command

```
library ()  
library("pkg")
```

or click to check the corresponding package(s) on R packages pane visible on bottom right.



Similarly for unloading the packages from current session, one can uncheck specific packages from package pane or following the command

```
detach("pkg", unload=TRUE)
```

## Navigating directories

The entities that R creates and manipulates are known as objects. These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components. During an R session, objects are created and stored by name (we discuss this process in the next section).

The R command ls() List objects in current R session

```
ls()
```

The collection of objects currently stored is called the workspace.

Return content of current working directory

```
dir()
```

Return path of current working directory

```
getwd()
```

Change current working directory

```
setwd("/home/user")
```

To remove objects the function rm is available:

```
rm(object_name)
```

All objects created during an R session can be stored permanently in a file for use in future R sessions. At the end of each R session you are given the opportunity to save all the currently available objects. If you indicate that you want to do this, the objects are written to a file called .RData5 in the current directory, and the command lines used in the session are saved to a file called .Rhistory.

When R is started at later time from the same directory it reloads the workspace from this file. At the same time the associated commands history is reloaded.

It is recommended that you should use separate working directories for analyses conducted with R. It is quite common for objects with names x and y to be created during an analysis. Names like this are often meaningful in the context of a single analysis, but it can be quite hard to decide what they might be when the several analyses have been conducted in the same directory.

## Vignettes

Vignettes are documents where the authors show some functionalities of their package in a more detailed way.

```
browseVignettes(package = 'name_of_package' )
```

## How to get help?

Complete help files in HTML and PDF forms are available. There is an excellent introduction to R package can be found in Help → Manuals (in pdf) "An introduction to R". Several other documentations are also given at the end of this lecture note. To get help on a particular commands or functions etc., type help (command name). For example to get help on function 'mean',

```
help(mean)
```

This will open the help file with the page containing the description of the function mean.

Another way to get help is to use "?" followed by function name. For example,

```
?mean
```

will open the same window again.

In this lecture note, all R commands and corresponding outputs are given as shaded text to differentiate the normal texts. It should be noted that R is case-sensitive, i.e. typing Help(mean), would get an error message,

```
Help(mean)  
Error: could not find function "Help"
```



## R commands

Technically R is an expression language with a very simple syntax. Some general set of rules followed in writing R commands are as follows:-

- i. R commands are case sensitive, so X and x are different symbols and would refer to different variables.
- ii. Elementary commands consist of either expressions or assignments.
- iii. If an expression is given as a command, it is evaluated, printed and the value is lost.
- iv. An assignment also evaluates an expression and passes the value to a variable but the result is not automatically printed.
- v. Commands are separated either by a semi-colon (;), or by a newline.
- vi. Elementary commands can be grouped together into one compound expression by braces '{' and '}'.
- vii. Comments can be put almost anywhere, starting with a hashmark (#). Anything written after # marks to the end of the line is considered as a comment.
- viii. Window can be cleared of lines by pressing Ctrl + L keys.

### Executing commands from or diverting output to a file

If commands are stored in an external file, say 'D:/commands.R' they may be executed at any time in an R session with the command

```
source("d:/commands.R")
```

for **Windows** Source is also available on the **File** menu.

The function sink(),

```
sink("d:/record.txt")
```

will divert all subsequent output from the console to an external file, 'record.txt' in D drive. The command

```
sink()
```

restores it to the console once again.

### Keyboard shortcuts in R

Shortcut	Function
<b>Ctrl-Z/Shift-Z</b>	Undo/Redo
<b>Ctrl-Enter</b>	Execute the current line or code selection in the Source pane
<b>Ctrl-Alt-R</b>	Execute all the R code in the currently open file in the Source pane
<b>Ctrl-Left/Right</b>	Navigate code quickly, word by word
<b>Home/End</b>	Navigate to the beginning/end of the current line
<b>Alt-Shift-Up/Down</b>	Duplicate the current line up or down
<b>Ctrl-D</b>	Delete the current line

### Datatypes in R

**Numeric:** Numbers that have a decimal value or are a fraction in nature have a data type as numeric.

```
x <- c(1, 2, 3)
x
```

```
[1] 1 2 3
is.numeric(x)
[1] TRUE
```

**Integer:** Numbers that do not contain decimal values have a data type as an integer.

**Character:** As the name suggests, it can be a letter or a combination of letters enclosed by quotes is considered as a character data type by R. It can be alphabets or numbers.

```
x <- c("1", "2", "3")
x
[1] "1" "2" "3"
is.character(x)
[1] TRUE
```

**Logical:** A variable that can have a value of True and False like a boolean is called a logical variable.

```
x <- 1:10 < 5
x
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

**Factor:** They are a data type that is used to refer to a qualitative relationship like colors, good & bad, treatment vs normal, etc. ie. vectors with grouping information is known as factor.

```
x <- factor(c("dog", "cat", "mouse", "dog", "dog", "cat"))
x
[1] dog cat mouse dog dog cat
## Levels: cat dog mouse
```

The datatypes for given dataset can be accessed through command

```
str(dataset_name)
```

## Data Structures in R

R operates on named data structures.

### Vector

A vector is a sequence of data elements of the same basic type. Members in a vector are officially called components/elements. The simplest such structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers. To set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

The function c() assigns the five numbers to the vector x. The assignment operator (<-) 'points' to the object receiving the value of the expression. Once can use the '=' operator as an alternative.

```
x = c(10.4, 5.6, 3.1, 6.4, 21.7)
```

A single number is taken as a vector of length one.

Assignments can also be made in the other direction, using the obvious change in the assignment operator. So the same assignment could be made using

```
c(10.4, 5.6, 3.1, 6.4, 21.7) -> x
```

If an expression is used as a complete command, the value is printed. So now if we were to use the command

```
> 1/x  
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
```

the reciprocals of the five values would be printed at the terminal.

### ***R can be used like a standard calculators using***

#### ***A. The elementary arithmetic operators***

- + addition
- subtraction
- \* multiplication
- / division
- ^ exponentiation

#### ***B. Arithmetic functions***

log, exp, sin, cos, tan, sqrt,

#### ***C. Other basic functions***

- max(x) – maximum element of vector x,
- min(x)- minimum element of vector x,
- range (x) – range of the values of vector x ,
- length(x) - the number of elements in x,
- sum(x) - the total of the elements in x,
- prod(x) – product of the elements in x
- mean(x) – average of the elements of x
- var(x) – sample variance of the elements of (x)
- sort(x) – returns a vector with elements sorted in increasing order.

#### ***D. Logical operators***

- < - less than
- <= less than or equal to
- >greater than
- >= greater than or equal to
- == equal to
- != not equal to.

### **Matrix**

A matrix is a two-dimensional rectangular data set and thus it can be created using vector input to the matrix function. A matrix can have two dimensional structures with data of same type.

### **Array**

In an array, data is stored in the form of matrices, row, and as well as in columns. We can use the matrix level, row index, and column index to access the matrix elements.

## Difference between arrays and matrices

Arrays	Matrices
Arrays can contain greater than or equal to 1 dimensions.	Matrices contains 2 dimensions in a table like structure.
Array is a homogeneous data structure.	Matrix is also a homogeneous data structure.
It is a singular vector arranged into the specified dimensions.	It comprises of multiple equal length vectors stacked together in a table.
<b>array()</b> function can be used to create matrix by specifying the third dimension to be 1.	<b>matrix()</b> function however can be used to create at most 2-dimensional array.
Arrays are superset of matrices.	Matrices are a subset, special case of array where dimensions is two.
Limited set of collection-based operations.	Wide range of collection operations possible.
Mostly, intended for storage of data.	Mostly, matrices are intended for data transformation.

## Lists

Lists are the objects which contain elements of different types – like strings, numbers, vectors and another list inside them. A list can also contain a matrix or a function as its elements. In other words, a list is a generic vector containing other objects.

## Data Frames

It generally refers to tabular data: a data structure representing the cases (rows), each of which consists of numbers of observation or measurement (columns). A data frame is used for storing data tables. It is a list of vectors of equal length.

*Characteristics of a Data Frame:*

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain the same number of data items.
- Datasets imported in R are stored as data frames by default.

## Datasets for R

R comes with several built-in data sets, which are generally used as demo data for playing with R functions.

<code>data(package = "datasets")</code>	All datasets in Datasets package
<code>library(help = "datasets")</code>	Details about the datasets

## List of datasets available in Standard R packages

AirPassengers	Monthly Airline Passenger Numbers 1949-1960
BJsales	Sales Data with Leading Indicator
BOD	Biochemical Oxygen Demand
CO2	Carbon Dioxide Uptake in Grass Plants
ChickWeight	Weight versus age of chicks on different diets
DNase	Elisa assay of DNase
EuStockMarkets	Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde	Determination of Formaldehyde
HairEyeColor	Hair and Eye Color of Statistics Students
Harman23.cor	Harman Example 2.3
Harman74.cor	Harman Example 7.4
Indometh	Pharmacokinetics of Indomethacin
InsectSprays	Effectiveness of Insect Sprays
JohnsonJohnson	Quarterly Earnings per Johnson & Johnson Share
LakeHuron	Level of Lake Huron 1875-1972
LifeCycleSavings	Intercountry Life-Cycle Savings Data
Loblolly	Growth of Loblolly pine trees
Nile	Flow of the River Nile
Orange	Growth of Orange Trees
OrchardSprays	Potency of Orchard Sprays
PlantGrowth	Results from an Experiment on Plant Growth
Puromycin	Reaction Velocity of an Enzymatic Reaction
Theoph	Pharmacokinetics of Theophylline
Titanic	Survival of passengers on the Titanic
ToothGrowth	The Effect of Vitamin C on Tooth Growth in Guinea Pigs
UCBAdmissions	Student Admissions at UC Berkeley
UKDriverDeaths	Road Casualties in Great Britain 1969-84
UKLungDeaths	Monthly Deaths from Lung Diseases in the UK
UKgas	UK Quarterly Gas Consumption
USAccDeaths	Accidental Deaths in the US 1973-1978
USArrests	Violent Crime Rates by US State
USJudgeRatings	Lawyers' Ratings of State Judges in the US Superior Court
USPersonalExpenditure	Personal Expenditure Data
VADeaths	Death Rates in Virginia (1940)
WWWusage	Internet Usage per Minute

WorldPhones	The World's Telephones
ability.cov	Ability and Intelligence Tests
airmiles	Passenger Miles on Commercial US Airlines, 1937-1960
airquality	New York Air Quality Measurements
anscombe	Anscombe's Quartet of 'Identical' Simple Linear Regressions
attenu	The Joyner-Boore Attenuation Data
attitude	The Chatterjee-Price Attitude Data
austres	Quarterly Time Series of the Number of Australian Residents
beavers	Body Temperature Series of Two Beavers
cars	Speed and Stopping Distances of Cars
chickwts	Chicken Weights by Feed Type
co2	Mauna Loa Atmospheric CO2 Concentration
crimtab	Student's 3000 Criminals Data
datasets-package	The R Datasets Package
discoveries	Yearly Numbers of Important Discoveries
esoph	Smoking, Alcohol and (O)esophageal Cancer
euro	Conversion Rates of Euro Currencies
eurodist	Distances Between European Cities and Between US Cities
faithful	Old Faithful Geyser Data
freeny	Freeny's Revenue Data
infert	Infertility after Spontaneous and Induced Abortion
iris	Edgar Anderson's Iris Data
islands	Areas of the World's Major Landmasses
lh	Luteinizing Hormone in Blood Samples
longley	Longley's Economic Regression Data
lynx	Annual Canadian Lynx trappings 1821-1934
morley	Michelson Speed of Light Data
mtcars	Motor Trend Car Road Tests
nhtemp	Average Yearly Temperatures in New Haven
nottem	Average Monthly Temperatures at Nottingham, 1920-1939 Classical N, P, K Factorial Experiment
occupationalStatus	Occupational Status of Fathers and their Sons
precip	Annual Precipitation in US Cities
presidents	Quarterly Approval Ratings of US Presidents
pressure	Vapor Pressure of Mercury as a Function of Temperature
quakes	Locations of Earthquakes off Fiji
randu	Random Numbers from Congruential Generator RANDU

rivers	Lengths of Major North American Rivers
rock	Measurements on Petroleum Rock Samples
sleep	Student's Sleep Data
stackloss	Brownlee's Stack Loss Plant Data
state	US State Facts and Figures
sunspot.month	Monthly Sunspot Data, from 1749 to "Present"
sunspot.year	Yearly Sunspot Data, 1700-1988
sunspots	Monthly Sunspot Numbers, 1749-1983
swiss	Swiss Fertility and Socioeconomic Indicators (1888) Data
treering	Yearly Treering Data, -6000-1979
trees	Diameter, Height and Volume for Black Cherry Trees
uspop	Populations Recorded by the US Census
volcano	Topographic Information on Auckland's MaungaWhau Volcano
warpbreaks	The Number of Breaks in Yarn during Weaving
women	Average Heights and Weights for American Women

## Reading and Writing External Data

### 1. Import of tabular data

#### A. Import of a tab-delimited tabular file

```
myDF <- read.delim("myData.xls", sep="\t")
```

read.csv()	Reads a CSV file into the memory.
read.delim()	Used to read in delimited text files
read.table()	Loads data from a file into a tabular data set (table) in memory.

#### B. Import of Google Sheets.

The following example imports a sample Google Sheet from [here](#). Detailed instructions for interacting from R with Google Sheets with the required googlesheets4 package are [here](#).

```
library(googlesheets4)
gs4_deauth() # Easiest method for reading public access sheets
mysheet <- read_sheet("1U-32UcwZP1k3saKeaH1mbvEAOfZRdNHNkWK2GI1rpPM", skip=4)
myDF <- as.data.frame(mysheet)
myDF
```

#### C. Import from Excel sheets

Import from Excel sheets works well with readxl. For details see the readxl package manual [here](#).

```
library("readxl")
mysheet <- read_excel(targets_path, sheet="Sheet1")
```

Note: working with tab- or comma-delimited files is more flexible and highly preferred for automated analysis workflows.

## 2. Export of tabular data

```
write.table(myDF, file="myfile.xls", sep="\t", quote=FALSE, col.names=NA)
```

## 3. Line-wise import

```
myDF <- readLines("myData.txt")
```

## 4. Line-wise export

```
writeLines(month.name, "myData.txt")
```

## 5. Export R object

```
mylist <- list(C1=iris[,1], C2=iris[,2]) # Example to export  
saveRDS(mylist, "mylist.rds")
```

## 6. Import R object

```
mylist <- readRDS("mylist.rds")
```

## 7. Copy and paste into R

On Windows/Linux systems

```
read.delim("clipboard")
```

On Mac OS X systems

```
read.delim(pipe("pbpaste"))
```

## 8. Copy and paste from R

On Windows/Linux systems

```
write.table(iris, "clipboard", sep="\t", col.names=NA, quote=FALSE)
```

On Mac OS X systems

```
zz <- pipe('pbcopy', 'w')  
write.table(iris, zz, sep="\t", col.names=NA, quote=FALSE)  
close(zz)
```



## Data Wrangling using Dplyr package

Data analysis can be divided into three parts:

- Extraction: First, we need to collect the data from many sources and combine them.
- Transform: This step involves the data manipulation. Once we have consolidated all the sources of data, we can begin to clean the data.
- Visualize: The last move is to visualize our data to check irregularity.

One of the most significant challenges faced by data scientists is the data manipulation. Data is never available in the desired format. Data scientists need to spend at least half of their time, cleaning and manipulating the data. That is one of the most critical assignments in the job. If the data manipulation process is not complete, precise and rigorous, the model will not perform correctly.

### Reference

- R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. <http://www.R-project.org>.
- Zuur, A.F., Ieno, E.N. and Meesters, E.H. (2009). A Beginner's Guide to R (p. 150). New York: Springer.
- [http://manuals.bioinformatics.ucr.edu/home/R\\_BioCondManual#TOC-Introduction](http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual#TOC-Introduction)
- <https://girke.bioinformatics.ucr.edu/GEN242/tutorials/rbasics/rbasics/>

## **Topic 4:**

## **Data Visualization using R**

Ms. Sona Charles  
Scientist (Bioinformatics),  
ICAR-Indian Institute of Spices Research, Kozhikode, Kerala.  
Email: [sona.charles@icar.gov.in](mailto:sona.charles@icar.gov.in)

### **Why data visualization is important?**

The volume of data used in research and technological development is massive and continues to grow. It becomes harder and harder for a user to grab a key message from this universe of data. That's where data visualization comes in: summarizing and presenting large data in simple and easy-to-understand visualizations to give readers insightful information.

There are many advanced visualizations (e.g., networks, 3D-models and map overlays) used for specialized purposes such as 3D medical imaging, agricultural yield monitoring etc. But regardless of the complexity of a visualization, its purpose is to help readers see a pattern or trend in the data being analyzed, rather than having them read tedious descriptions. A good visualization summarizes information and organizes in a way that enables the reader to focus on the points that are relevant to the key message being conveyed.

First you must import your data into R. This typically means that you take data stored in a file, database, or web application programming interface (API), and load it into a data frame in R. If you can't get your data into R, you can't do data science on it!

Once you've imported your data, it is a good idea to tidy it. Tidying your data means storing it in a consistent form that matches the semantics of the dataset with the way it is stored. In brief, when your data is tidy, each column is a variable, and each row is an observation. Tidy data is important because the consistent structure lets you focus your struggle on questions about the data, not fighting to get the data into the right form for different functions.

Once you have tidy data, a common first step is to transform it. Transformation includes narrowing in on observations of interest (like all people in one city, or all data from the

last year), creating new variables that are functions of existing variables (like computing speed from distance and time), and calculating a set of summary statistics (like counts or means). Together, tidying and transforming are called wrangling, because getting your data in a form that's natural to work with often feels like a fight!

Once you have tidy data with the variables you need, there are two main engines of knowledge generation: visualisation and modelling. These have complementary strengths and weaknesses so any real analysis will iterate between them many times.

Visualisation is a fundamentally human activity. A good visualisation will show you things that you did not expect, or raise new questions about the data. A good visualisation might also hint that you're asking the wrong question, or you need to collect different data. Visualisations can surprise you, but don't scale particularly well because they require a human to interpret them.

Models are complementary tools to visualisation. Once you have made your questions sufficiently precise, you can use a model to answer them. Models are a fundamentally mathematical or computational tool, so they generally scale well. Even when they don't, it's usually cheaper to buy more computers than it is to buy more brains! But every model makes assumptions, and by its very nature a model cannot question its own assumptions. That means a model cannot fundamentally surprise you.

The last step of data science is communication, an absolutely critical part of any data analysis project. It doesn't matter how well your models and visualisation have led you to understand the data unless you can also communicate your results to others.

Surrounding all these tools is programming. Programming is a cross-cutting tool that you use in every part of the project. You don't need to be an expert programmer to be a data scientist, but learning more about programming pays off because becoming a better programmer allows you to automate common tasks, and solve new problems with greater ease.

Exploratory data visualization is perhaps the greatest strength of R. One can quickly go from idea to data to plot with a unique balance of flexibility and ease. Many other approaches are available for creating plots in R. In fact, the plotting capabilities that come with a basic installation of R are already quite powerful. There are also other

packages for creating graphics such as grid and lattice. We chose to use ggplot2 because it breaks plots into components in a way that permits beginners to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and comparatively easy to remember.

One reason ggplot2 is generally more intuitive for beginners is that it uses a grammar of graphics, the gg in ggplot2. This is analogous to the way learning grammar can help a beginner construct hundreds of different sentences by learning just a handful of verbs, nouns and adjectives without having to memorize each specific sentence. Similarly, by learning a handful of ggplot2 building blocks and its grammar, you will be able to create hundreds of different plots.

Another reason ggplot2 is easy for beginners is that its default behavior is carefully chosen to satisfy the great majority of cases and is visually pleasing. As a result, it is possible to create informative and elegant graphs with relatively simple and readable code.

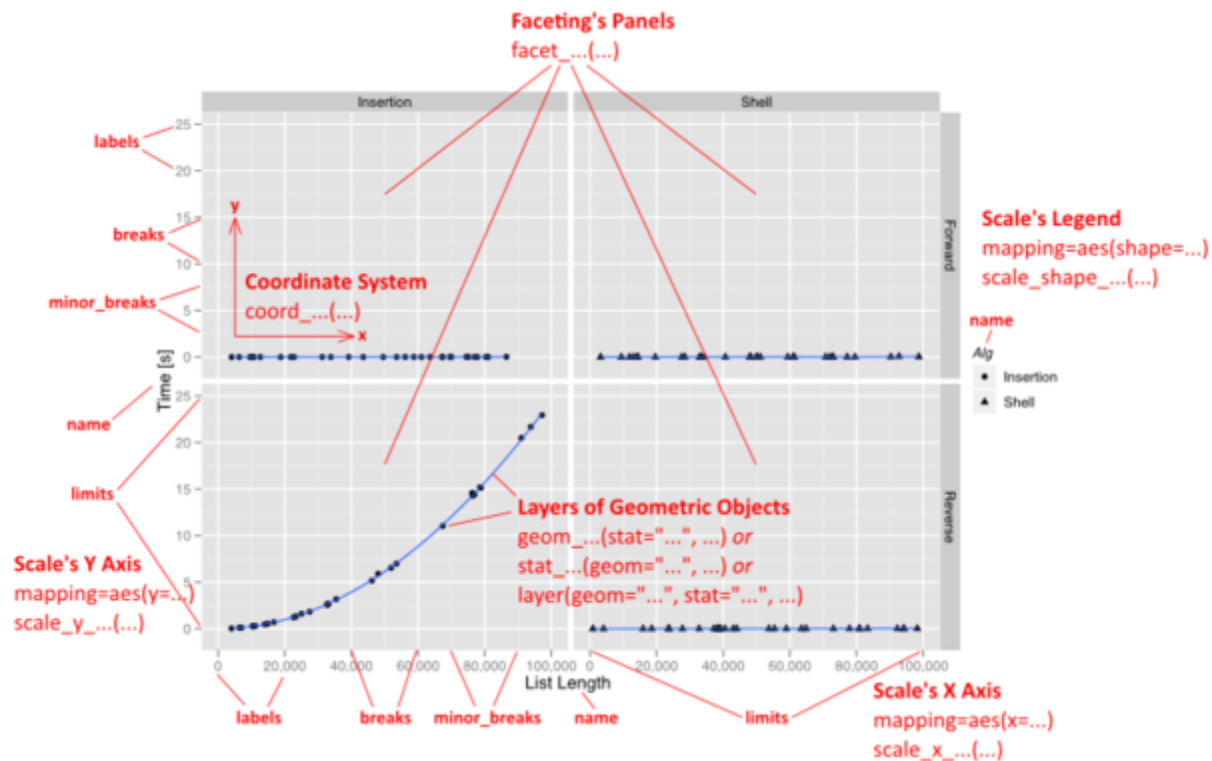
One limitation is that ggplot2 is designed to work exclusively with data tables in tidy format (where rows are observations and columns are variables). However, a substantial percentage of datasets that beginners work with are in, or can be converted into, this format. An advantage of this approach is that, assuming that our data is tidy, ggplot2 simplifies plotting code and the learning of grammar for a variety of plots.

### **What is the grammar of graphics?**

Wilkinson created the grammar of graphics to describe the fundamental features that underlie all statistical graphics. The grammar of graphics is an answer to the question of what is a statistical graphic? ggplot2 builds on Wilkinson's grammar by focussing on the primacy of layers and adapting it for use in R. In brief, the grammar tells us that a graphic maps the data to the aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also include statistical transformations of the data and information about the plot's coordinate system. Facetting can be used to plot for different subsets of the data. The combination of these independent components are what make up a graphic.

## Anatomy of a ggplot

All plots are composed of the data, the information you want to visualise, and a mapping, the description of how the data's variables are mapped to aesthetic attributes. There are five mapping components



A **layer** is a collection of geometric elements and statistical transformations. Geometric elements, geoms for short, represent what you actually see in the plot: points, lines, polygons, etc. Statistical transformations, stats for short, summarise the data: for example, binning and counting observations to create a histogram, or fitting a linear model.

**Scales** map values in the data space to values in the aesthetic space. This includes the use of colour, shape or size. Scales also draw the legend and axes, which make it possible to read the original data values from the plot (an inverse mapping).

A **coord**, or coordinate system, describes how data coordinates are mapped to the plane of the graphic. It also provides axes and gridlines to help read the graph. We normally use the Cartesian coordinate system, but a number of others are available, including polar coordinates and map projections.

A **facet** specifies how to break up and display subsets of data as small multiples. This is also known as conditioning or latticing/trellising.

A **theme** controls the finer points of display, like the font size and background colour. While the defaults in `ggplot2` have been chosen with care, you may need to consult other references to create an attractive plot.

## SCATTERPLOTS

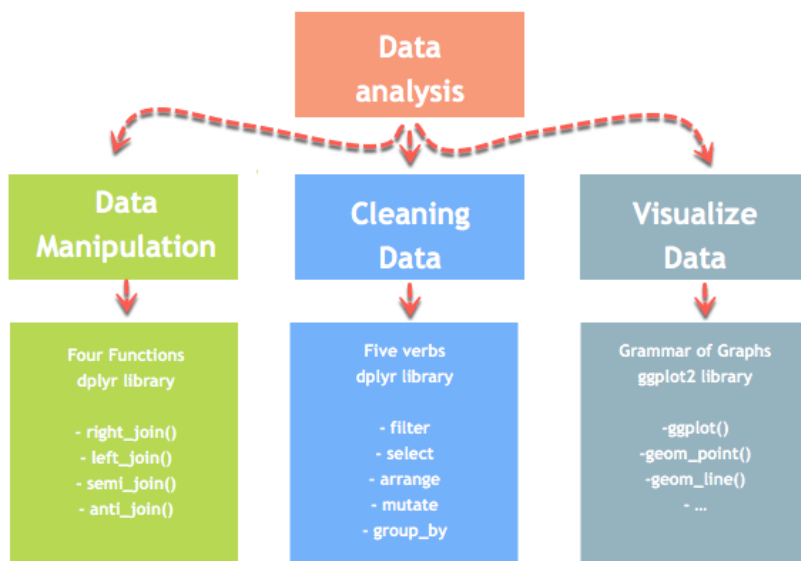
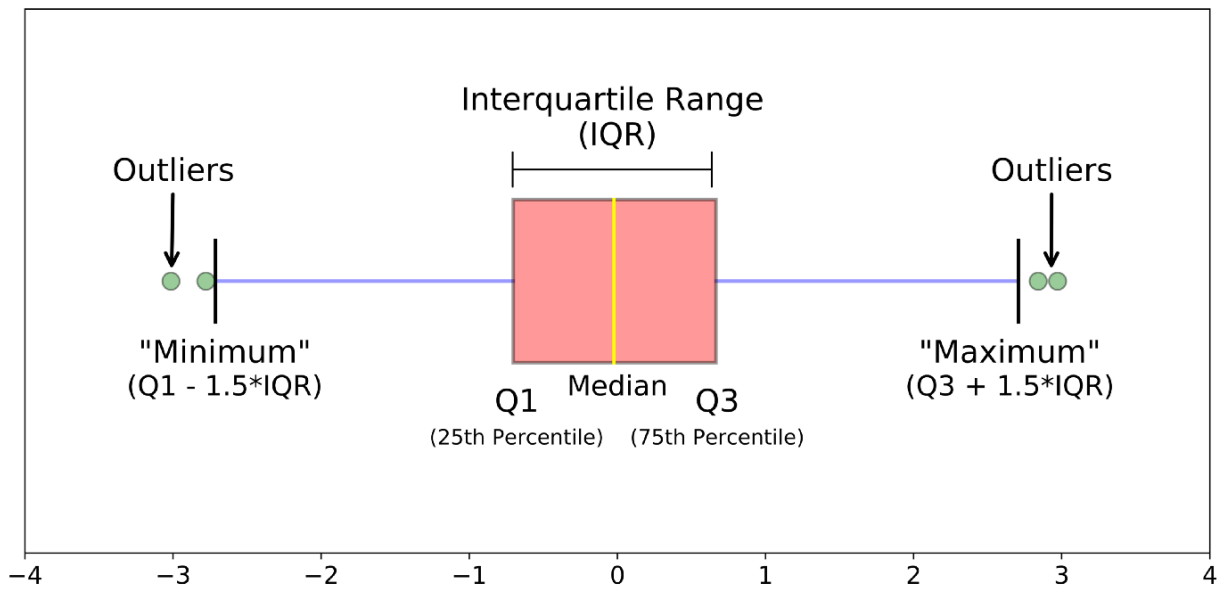
A scatter plot is a chart type that is normally used to observe and visually display the relationship between variables. The values of the variables are represented by dots. Scatter plots are also known as scattergrams, scatter graphs, or scatter charts.

Boxplots are a standardized way of displaying the distribution of data based on a five number summary median (Q2/50th Percentile): the middle value of the dataset.

- first quartile (Q1/25th Percentile): the middle number between the smallest number (not the “minimum”) and the median of the dataset.
- third quartile (Q3/75th Percentile): the middle value between the median and the highest value (not the “maximum”) of the dataset.
- interquartile range (IQR): 25th to the 75th percentile.
- whiskers (shown in blue)
- outliers (shown as green circles)

“maximum”:  $Q3 + 1.5 * IQR$

“minimum”:  $Q1 - 1.5 * IQR$



The package dplyr provide easy tools for the most common data manipulation tasks. It is built to work directly with data frames.

We're going to learn some of the most common dplyr functions: `select()`, `filter()`, `mutate()`, `arrange()`, and `summarize()`.

## **DOT PLOT**

Dot Plot is a graph for displaying the distribution of quantitative variable where each dot represents a value.

## **VIOLIN PLOT**

A violin plot depicts distributions of numeric data for one or more groups using density curves. The width of each curve corresponds with the approximate frequency of data points in each region.

## **HISTOGRAM**

A histogram contains rectangular area to display the statistical information which is proportional to the frequency of a variable and its width in successive numerical intervals.

## **DENSITY PLOT**

A density plot is a representation of the distribution of a numeric variable. It is a smoothed version of the histogram and is used in the same concept.

## **THEMES IN GGLOT**

Themes are a powerful way to customize the non-data components of your plots: i.e. titles, labels, fonts, background, gridlines, and legends. Themes can be used to give plots a consistent customized look.

## **PIE CHART**

A pie chart is a circle divided into sectors that each represent a proportion of the whole.

## **RIDGELINE PLOT**

A Ridgeline plot (sometimes called Joyplot) shows the distribution of a numeric value for several groups. Distribution can be represented using histograms or density plots, all aligned to the same horizontal scale and presented with a slight overlap.



## **VOLCANO PLOT**

Volcano plots are commonly used to display the results of RNA-seq or other omics experiments. A volcano plot is a type of scatterplot that shows statistical significance (P value) versus magnitude of change (fold change). It enables quick visual identification of genes with large fold changes that are also statistically significant. These may be the most biologically significant genes. In a volcano plot, the most upregulated genes are towards the right, the most downregulated genes are towards the left, and the most statistically significant genes are towards the top.

To generate a volcano plot of RNA-seq results, we need a file of differentially expressed results, a sample of which is provided in this workshop.

## **CIRCOS PLOT/ IDIOGRAM**

Circular layout is very useful to represent complicated information. First, it elegantly represents information with long axes or a large amount of categories; second, it intuitively shows data with multiple tracks focusing on the same object; third, it easily demonstrates relations between elements. It provides an efficient way to arrange information on the circle and it is beautiful.

Circular visualization is popular in Genomics and related omics fields. It is efficient in revealing associations in high dimensional genomic data. In genomic plots, categories are usually chromosomes and data on x axes are genomic positions, but it can also be any kind of general genomic categories.

To make it easy for Genomics analysis, circlize package particularly provides functions which focus on genomic plots. Genomic data is usually stored as a table where the first three columns define the genomic regions and following columns are values associated with the corresponding regions. Each genomic region is composed by three elements: genomic category (in most case, it is the chromosome), start position on the genomic category and the end position. Such data structure is known as BED format and is broadly used in genomic research.

## **HEATMAP**

Heatmap is a powerful tool for the visual display of microarray data or data from next-generation sequencing studies such as microbiome analysis. Heatmap is a graphical representation of data that uses a system of color-coding in representing different values contained in a matrix. As early as the 19th century, heatmaps were used in statistical analysis and progressed in 2008 as a useful tool for almost every field such as engineering, medicine, and even in research. Heatmap is also user-friendly, more importantly to those who are not accustomed to reading large quantities of data since it is more visually accessible than traditional data formats. Heatmap is considered a useful tool because it can provide a comprehensive overview as its data visualization tools are easy to understand and are often self-explanatory. It is a lot different from a table or chart which both need to be interpreted or studied to be understood.

## **DENDROGRAM**

A dendrogram (or tree diagram) is a network structure. It is constituted of a root node that gives birth to several nodes connected by edges or branches. The last nodes of the hierarchy are called leaves. Many options are available to build one with R.

## **Topic 5:**

## **Introduction to Linux**

Dr. Merlin Lopez  
Scientist (Bioinformatics),  
Community Agrobiodiversity Centre,  
MS Swaminathan Research Foundation, Wayanad, Kerala  
Email: merlinettizha@gmail.com

Linux is an operating system's kernel. It is the software on a computer that enables applications and the computer operator to access the devices on the computer to perform desired functions. People started calling GNU OS, Linux because of its kernel name. At its core, the Linux operating system is derived from the Unix OS. It comes in different “distributions” to serve different purpose.

### **History**

#### UNIX

In order to understand the popularity of Linux, we need to travel back in time, about 30 years ago..

Imagine computers as big as houses, even stadiums. While the sizes of those computers posed substantial problems, there was one thing that made this even worse: every computer had a different operating system. Software was always customized to serve a specific purpose, and software for one given system didn't run on another system. Being able to work with one system didn't automatically mean that you could work with another. It was difficult, both for the users and the system administrators. Computers were extremely expensive then, and sacrifices had to be made even after the original purchase just to get the users to understand how they worked. The total cost per unit of computing power was enormous.

Technologically the world was not quite that advanced, so they had to live with the size for another decade. In 1969, a team of developers in the Bell Labs laboratories started working on a solution for the software problem, to address these compatibility issues. They developed a new operating system, which was

1. Simple and elegant.
2. Written in the C programming language instead of in assembly code.
3. Able to recycle code.

The Bell Labs developers named their project "UNIX."

The code recycling features were very important. Until then, all commercially available computer systems were written in a code specifically developed for one system. UNIX on the other hand needed only a small piece of that special code, which is now commonly named the kernel. This kernel is the only piece of code that needs to be adapted for every specific system

and forms the base of the UNIX system. The operating system and all other functions were built around this kernel and written in a higher programming language, C.

### **Linus and Linux**

By the beginning of the 90s home PCs were finally powerful enough to run a full blown UNIX. Linus Torvalds, a young man studying computer science at the university of Helsinki, thought it would be a good idea to have some sort of freely available academic version of UNIX, and promptly started to code which led to the development of Linux.

Linux is a full UNIX clone, fit for use on workstations as well as on middle-range and high-end servers. Today, a lot of the important players on the hard- and software market each have their team of Linux developers; at your local dealer's you can even buy pre-installed Linux systems with official support - eventhough there is still a lot of hard- and software that is not supported, too.

### **Current application of Linux systems**

Today Linux has joined the desktop market. Linux developers concentrated on networking and services in the beginning, and office applications have been the last barrier to be taken down. We don't like to admit that Microsoft is ruling this market, so plenty of alternatives have been started over the last couple of years to make Linux an acceptable choice as a workstation, providing an easy user interface and MS compatible office applications like word processors, spreadsheets, presentations and the like. On the server side, Linux is well-known as a stable and reliable platform, providing database and trading services for companies like Amazon, the well-known online bookshop, US Post Office, the German army and many others. Especially Internet providers and Internet service providers have grown fond of Linux as firewall, proxy- and web server, and you will find a Linux box within reach of every UNIX system administrator who appreciates a comfortable management station. Clusters of Linux machines are used in the creation of movies such as "Titanic", "Shrek" and others. In post offices, they are the nerve centers that route mail and in large search engine, clusters are used to perform internet searches. These are only a few of the thousands of heavy-duty jobs that Linux is performing day-to-day across the world. It is also worth to note that modern Linux not only runs on workstations, mid- and high-end servers, but also on "gadgets" like PDA's, mobiles, a shipload of embedded applications and even on experimental wristwatches. This makes Linux the only operating system in the world covering such a wide range of hardware.

### **Linux distributions**

Companies such as RedHat, SuSE and Mandriva have sprung up, providing packaged Linux distributions suitable for mass consumption. They integrated a great deal of graphical user interfaces (GUIs), developed by the community, in order to ease management of programs and services. As a Linux user today you have all the means of getting to know your system inside out, but it is no longer necessary to have that knowledge in order to make the system comply to your requests.

While development in the service area continues, great things are being done for desktop users, generally considered as the group least likely to know how a system works. Developers of

desktop applications are making incredible efforts to make the most beautiful desktops you've ever seen, or to make your Linux machine look just like your former MS Windows or an Apple workstation. The latest developments also include 3D acceleration support and support for USB devices, single-click updates of system and packages, and so on. Linux has these, and tries to present all available services in a logical form that ordinary people can understand. Below is a short list containing some great examples; these sites have a lot of screenshots that will give you a glimpse of what Linux on the desktop can be like:

- <http://www.gnome.org>
- <http://kde.org/screenshots/>
- <http://www.openoffice.org>
- <http://www.mozilla.org>

## **Advantages of Linux**

Linux is free:

As in free beer, they say. If you want to spend absolutely nothing, you don't even have to pay the price of a CD. Linux can be downloaded in its entirety from the Internet completely for free. No registration fees, no costs per user, free updates, and freely available source code in case you want to change the behavior of your system. Most of all, Linux is free as in free speech: The license commonly used is the GNU Public License (GPL). The license says that anybody who may want to do so, has the right to change Linux and eventually to redistribute a changed version, on the one condition that the code is still available after redistribution. In practice, you are free to grab a kernel image, for instance to add support for teletransportation machines or time travel and sell your new code, as long as your customers can still have a copy of that code.

Linux is portable to any hardware platform:

A vendor who wants to sell a new type of computer and who doesn't know what kind of OS his new machine will run (say the CPU in your car or washing machine), can take a Linux kernel and make it work on his hardware, because documentation related to this activity is freely available

Linux was made to keep on running:

As with UNIX, a Linux system expects to run without rebooting all the time. That is why a lot of tasks are being executed at night or scheduled automatically for other calm moments, resulting in higher availability during busier periods and a more balanced use of the hardware. This property allows for Linux to be applicable also in environments where people don't have the time or the possibility to control their systems night and day.

Linux is secure and versatile:

The security model used in Linux is based on the UNIX idea of security, which is known to be robust and of proven quality. But Linux is not only fit for use as a fort against enemy attacks from the Internet: it will adapt equally to other situations, utilizing the same high standards for security. Your development machine or control station will be as secure as your firewall.

Linux is scalable:

From a Palmtop with 2 MB of memory to a petabyte storage cluster with hundreds of nodes: add or remove the appropriate packages and Linux fits all. You don't need a supercomputer anymore, because you can use Linux to do big things using the building blocks provided with the system. If you want to do little things, such as making an operating system for an embedded processor or just recycling your old 486, Linux will do that as well.

The Linux OS and most Linux applications have very short debug-times:

Because Linux has been developed and tested by thousands of people, both errors and people to fix them are usually found rather quickly. It sometimes happens that there are only a couple of hours between discovery and fixing of a bug.

## Reference

Machtelt Garrels, Introduction to Linux, A Hands on Guide

(file:///C:/Users/ADMIN/Dropbox/My%20PC%20(LAPTOP9EF6MEMP)/Desktop/IISR%20Workshop/intro-linux.pdf)

**Topic 6:****Introduction to Python**

Mr. Subeesh A  
Scientist (Computer Applications),  
ICAR- Central Institute of Agricultural Engineering, Bhopal, Madhya pradesh  
Email: subeesh.a@icar.gov.in

**Overview**

Python is a widely used high-level object oriented programming language created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It is also called general-purpose programming language as it is used in almost every domain we can think of such as:

- Web Development
- Software Development
- Game Development
- Artificial Intelligence and Machine learning
- Data Analytics, etc.

The main reasons for the wide adoption of python are very simple to understand, scalable because of which the speed of development is so fast. Python has simpler syntax similar to the English language and also the syntax allows developers to write programs with fewer lines of code than some other programming language. Since it is open-source there are many libraries available that make developers' jobs easy ultimately results in high productivity. This means that prototyping can be very quick. IEE spectrum has ranked python as #1 popular language of 2021.

The most recent major version of Python is Python 3, which we shall be using in this training manual.

Language Ranking: IEEE Spectrum						
Rank	Language	Type				Score
1	Python	🌐	🖥️	⚙️		100.0
2	Java	🌐	📱	🖥️		95.4
3	C		📱	🖥️	⚙️	94.7
4	C++		📱	🖥️	⚙️	92.4
5	JavaScript	🌐				88.1
6	C#	🌐	📱	🖥️	⚙️	82.4
7	R			🖥️		81.7
8	Go	🌐		🖥️		77.7
9	HTML	🌐				75.4
10	Swift		📱	🖥️		70.4

Figure 1 : IEE spectrum ranking of languages 2021 (<https://spectrum.ieee.org/top-programming-languages/>)



## **Notes:**

- Python runs on an interpreter system, meaning that code can be executed as soon as it is written.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.
- This training manual uses google Colab to execute python commands. All the codes are written in Python 3.7 version. Python programs can be written in a text editor as well. It is also possible to write Python in an Integrated Development Environment, such as Spyder, Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

## **Beginning with Python Programming**

### **1. Python print statements**

The print() function in Python is used to print a specified message on the screen. The print command in Python prints strings or objects which are converted to a string while printing on a screen.

```
>>>print ("Hello python")
```

### **2. Python Indentations**

Indentation refers to the spaces at the beginning of a code line. The indentation in Python is very important and it indicates a block of code.

Eg:

```
if 6 > 2:  
print("Six is greater than two!")
```

### **3. Python Comments**

Comments can be used to explain a python code and it makes the code more readable.

Comments start with a #, and Python will ignore them during the execution.

E.g:

```
#This is a comment  
print("Hello, World!")
```

#### **4. Python Variables**

Variables are containers for storing data values. Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

Eg:

```
x = 6  
y = "Sam"  
print(x)  
print(y)
```

When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

E.g:

```
data = "Welcome"  
print(data)
```

Assigning multiple values to multiple variables can be performed using the below code.

```
a, b, c = 5, 4.5, "Testdata"
```

```
print (a)
```

```
print (b)
```

```
print (c)
```

#### **5. Identifiers**

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, \$, and % within identifiers. Python is a case sensitive programming language.

Examples of valid identifiers: test, a65, \_num, n\_9data, etc.

Examples of invalid identifiers: 1a, n%4, n 9, etc.

## **6. Keywords**

Keywords are the reserved words in Python and we cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

E.g : if, break, import, else, for, is, etc.

## **7. Data types**

Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function, which returns the type of the variable passed.

```
a=10
```

```
b="Hi Python"
```

```
c = 10.5
```

```
print(type(a)) # Outputs <type 'int'>
```

```
print(type(b)) # Outputs <type 'str'>
```

```
print(type(c)) # Outputs <type 'float'>
```

Some of the standard datatypes used in python are given below.

### **7.1. Python Numbers**

Integers, floating point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python.

```
a = 7    # Integer type
a = 2.2  # Float type
a = 1+3j # Complex type
```

## 7.2. Python List

List is an ordered sequence of elements. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. A python list is declared with elements separated by commas are enclosed within brackets [ ].

**Eg:**

```
a = [1, 4.3,'data']
```

Slicing operator [ ] to extract an item or a range of items from a list. The index starts from 0 in Python.

## 7.3. Python Tuple

Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified and it is faster than lists.

It is defined within parentheses () where items are separated by commas.

**E.g:**

```
test = (5,'data', 1+5j)
```

```
print("test[1] = ", test[1])    #outputs 5
```

```
t[1] = 56                       #Generates error
```

## 7.4. Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """".

**Eg:**

```
s = "This is a string"
```

```
s = """A multiline
```

string"

### **7.5. Python Set**

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

Eg:

```
a = {5,2,3,1,4}
```

### **7.6. Python Dictionary**

Dictionaries are used to store data values in key:value pairs. It is a collection of changeable items and do not allow duplicates. Dictionaries are written with curly brackets, and have keys and values:

Eg:

```
Sample_dict= {  
    "name": "James",  
    "Rollno": "123",  
    "year": 2001  
}
```

## **Python Flow Control**

### **7.7. if...else Statement**

The if...else statement in python is used for decision making. The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.. If the condition provided in the if statement is false, then the else statement will be executed.

Eg:

if test expression:

    Body of if

else:

    Body of else

## 7.8. For loop

The for loop in Python is used to iterate over a sequence (list, dictionary, tuple, string) or other iterable objects.

For loop has the following syntax in python.

```
for i in sequence:
```

```
    loop body
```

Eg:

```
names = ["John", "Sam", "James"]
for x in names:
    print(x)
```

## 7.9. While loop

With the while loop we can execute a set of statements as long as a condition is true. We need to define an indexing variable and change it in each iteration, otherwise the loop may continue forever.

```
while test_expression:
```

```
    Body of while
```

Eg:

```
i = 1
while i < 5:
    print(i)                # Prints the numbers 1 to 4
    i += 1
```

## 8. Python Functions

A function is a block of code which only runs when it is called. Functions help in breaking the complex program into smaller chunks. Functions make the code more readable, less repetitive, reusable and highly manageable. In Python a function is defined using the def keyword. To call a function, use the function name followed by parenthesis. Information can be passed into

functions as arguments and values can be returned. Arguments are specified after the function name, inside the parentheses, separated with a comma.

### **Eg 1: Function without arguments**

```
def my_function():  
    print("Hello, this is a function")  
  
my_function()
```

### **Eg. 2 :Function with arguments**

```
def square( num ):  
    return num**2  
  
object_ = square(3) # Returns square of the argument passed
```

## **Python for Data Analysis**

### **1. Numpy**

NumPy is an array processing package in Python that provides a high-performance multidimensional array object and tools for working with it. It is the fundamental package for scientific computing with Python.

### **2. Pandas**

Pandas is referred as Python Data Analysis Library. It is another open source Python library for availing high-performance data structures and analysis tools. It is developed over the Numpy package. It contains DataFrame as its main data structure. With DataFrame you can store and manage data from tables by performing manipulation over rows and columns. Pandas can handle multiple data format such as excel, csv, SQL, HDFS, etc.

### **3. Matplotlib**

Matplotlib is a python library used to create graphs and plots by using python scripts. It has a module named pyplot which can ease the plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc.

#### **4. Scipy**

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc

#### **5. Scikit-learn**

Scikit-learn is one of the most popular python libraries for implementing machine learning algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms.

#### **6. Keras**

Keras is one of the most powerful Python libraries which allow high-level neural networks APIs for integration. Keras was created for reducing challenges faced in complex researches allowing them to compute faster. Due to its modular nature, one can use varieties of modules from neural layers, optimizers, activation functions etc., for developing a new model.

#### **7. TensorFlow**

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team. It is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications and is widely used in the field of deep learning research and application.

#### **8. Pytorch**

Pytorch is a Python-based scientific computing package that uses the power of graphics processing unit. It specializes in tensor computations, automatic differentiation, and GPU acceleration. For those reasons, PyTorch is one of the most popular deep learning libraries, competing with both Keras and TensorFlow. The framework is built to speed up the process between research prototyping and deployment.



**Topic 7:****Introduction to Galaxy**

Dr. Prashanth N Suravajhala  
Principal scientist, School of Biotechnology,  
Amrita Vishwa Vidyapeetham, Amritapuri, Kollam, Kerala  
Email: prash@am.amrita.edu

Biomedical studies have become data-intensive, with ever evolving technological and computational demands. Increasing reliance on complex computational methods prevents many biomedical researchers, from accessing and making effective use of these datasets and methods. This also presents significant barriers to reproducibility, dissemination and generalized reuse. Since 2005, the Galaxy project (<https://galaxyproject.org>) has provided free and open solutions to address these considerable barriers in biomedical research. Galaxy is an open source, community-driven, and web-based platform for accessible, reproducible, and transparent computational research and training. Galaxy supports accessibility by enabling complex computational analysis to be performed from a web browser without requiring programming experience or training in high performance computing. Reproducibility is ensured, as Galaxy automatically captures execution information (e.g. tool name, version, inputs, outputs and parameters) so that a user doesn't have to manually track provenance; hence, any user can repeat and understand a complete computational analysis, from tool parameters to the dependency tree. Galaxy users are able to share and publish their exact analysis histories, results, workflows and visualizations directly over the web, enabling transparency of computational research efforts and artifacts.

The Galaxy software ecosystem consists of multiple components: (a) an integrated repository of tools for a wide-range of biomedical studies including sequence and variant analysis, metagenomics, proteomics, and transcriptomics (b) a web application that enables exploratory data analysis using the integrated tools via a web interface; (c) a multitude of specialized installations of the web application (d) a training network that provides tutorials and organizes workshops on using Galaxy for different studies and (e) An inclusive and diverse community of developers, educators, and researchers encompassing a wide range of skill sets, scientific domains, and research practices that provide development and support.

In the past year, the Galaxy project has seen major growth as a platform, a resource, and a community. The Galaxy framework has been widely deployed by others, with 125 other known public instances (<https://galaxyproject.org/use>). The developer community has thrived, with >7500 tools contributed to the Galaxy ToolShed as of January 2020.