ARTIFICIAL INTELLIGENCE & DATA SCIENCE LAB (R20A1281)

LABORATORY MANUAL

B.TECH

(III YEAR–I SEM)

(2023-2024)

PREPARED BY
T.SHILPA
P.V.NARESH

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution–UGC,Govt.ofIndia)
Recognized under2(f)and12(B) of UGCACT1956
Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA Tier 1 & NAAC – 'A' Grade - ISO 9001:2015Certified) Maisammaguda,Dhulapally(PostVia.Hakimpet),Secunderabad–500100,TelanganaState,India

DEPARTMENT OF INFORMATION TECHNOLOGY

Vision & Mission

Vision

* To achieve high quality in technical education that provides the skills and attitude to adapt to the global needs of the Information Technology sector, through academic and research excellence.

Mission

* To equip the students with the cognizance for problem solving and to improve the teaching learning pedagogy by using innovative techniques.

* To strengthen the knowledge base of the faculty and students with motivation towards possession of effective academic skills and relevant research experience.

* To promote the necessary moral and ethical values among the engineers, for the betterment of the society.

Quality Policy

* Strives to inculcate the students with the world class Technical Knowledge, Entrepreneurial Competence and Social Ethics by providing continual improvement and innovation in the curriculum; based upon well-defined measurements and best practices.

* Develop faculty competencies, creativity, empowerment and accountability through faculty development programs and show strong management involvement and commitment.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS:

- To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS:

- To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS:

- To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS:

- To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Information Technology, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:-** Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3. **Applications of Computing Domain & Research:** Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design / development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance : Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
Maisammaguda,DhulapallyPost,ViaHakimpet,Secunderabad–500100

DEPARTMENT OF INFORMATION TECHNOLOGY

GENERAL LABORATORY INSTRUCTIONS

DEPARTMENT OF INFORMATION TECHNOLOGY
GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time),those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3. Student should enter into the laboratory with:
   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
   b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
   c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer systemallotted to you by the faculty.

5. Execute your task in the laboratory, and record the results / output in the lab observation notebook, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high end branded systems, which should beutilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severepunishment.

9. Students must take the permission of the faculty in case of any urgency to go out; if anybodyfound loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is keptproperly.

HEAD OF THE DEPARTMENT **PRINCIPAL**

INDEX

ARTIFICIAL INTELLIGENCE : PROLOG PROGRAMMING

1) Write a Prolog program for the usage of all arithmetic Operators.

Program:

```
Calculate: - X is 100 + 200, write('100 + 200 is '),write(X),nl,
      Y is 400 - 150, write('400 - 150 is '),write(Y),nl,
    Z is 10 * 300, write('10 * 300 is'),write(Z),nl,
    A is 100 / 30,write('100 / 30 is '),write(A),nl,
    B is 100 // 30,write('100 // 30'),write(B),nl,
    C is 100 ** 2,write('100 ** 2'),write(C),nl,
    D is 100 mod 30,write('100 mod 30 is '),write(D),nl.
```

Output:

```
Compiling C:/GNU-Prolog/bin/art.pl for byte code...
C:/GNU-Prolog/bin/art.pl compiled, 6 lines read - 2356 bytes written, 6 ms
| ?- calculate.
100 + 200 is 300
400 - 150 is 250
10 * 300 is3000
100 / 30 is 3.3333333333333335
100 // 303
100 ** 210000.0
100 mod 30 is 10
```

Signature of the Faculty

2. Write a Prolog program for solving the Towers of Hanoi problem**.**

Program:

```
move(1,X,Y,_) :-
write('Move top disk from'),
        write(X),
        write(' to '),
        write(Y),
        nl.
move(N,X,Y,Z) :-
N>1,
M is N-1,
move(M,X,Z,Y),
move(1,X,Y,_),
move(M,Z,Y,X).
```

output:

```
compiling C:/GNU-Prolog/bin/tower.pl for byte code...
C:/GNU-Prolog/bin/tower.pl compiled, 11 lines read - 1375 bytes written, 5 ms
| ?- listing.

% file: C:/GNU-Prolog/bin/tower.pl

move(1, A, B, _) :-
        write('Move top disk from'),
        write(A),
        write(' to '),
        write(B),
        nl.
move(A, B, C, D) :-
        A > 1,
        E is A - 1,
        move(E, B, D, C),
        move(1, B, C, _),
        move(E, D, C, B).
yes
| ?- move(3,source,target,interm).
Move top disk from source to target
Move top disk from source to interm
Move top disk from target to interm
Move top disk from source to target
Move top disk from interm to source
Move top disk from interm to target
Move top disk from source to target
```

Signature of the Faculty

3. Write a Prolog program to solve Monkey and banana problem.

Program:

```
move(state(middle,onbox,middle,hasnot),
grasp,state(middle,onbox,middle,has)).
move(state(P,onfloor,P,H),
climb,
state(P,onbox,P,H)).
move(state(P1,onfloor,P1,H),
push(P1,P2),
state(P2,onfloor,P2,H)).
move(state(P1,onfloor,B,H),
walk(P1,P2),
state(P2,onfloor,B,H)).
canget(state(_,_,_,has)).
canget(State1) :-
move(State1,_,State2),
canget(State2).
```

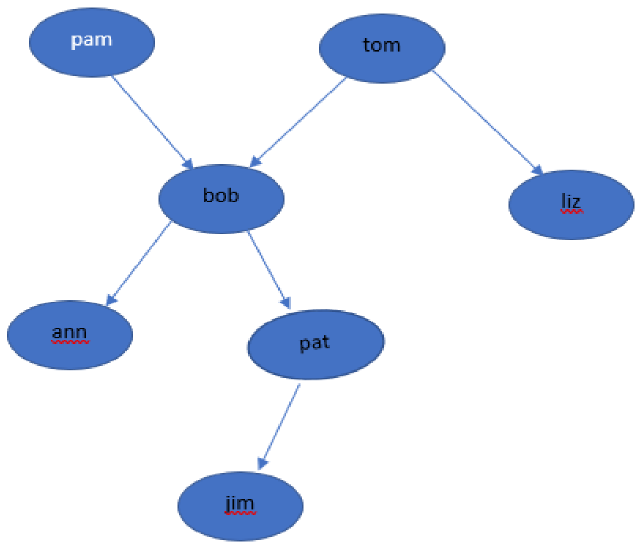
Output:

```
compiling C:/GNU-Prolog/bin/monkey.pl for byte code...
C:/GNU-Prolog/bin/monkey.pl compiled, 16 lines read - 2137 bytes written, 7 ms
| ?- [monkey].
compiling C:/GNU-Prolog/bin/monkey.pl for byte code...
C:/GNU-Prolog/bin/monkey.pl compiled, 16 lines read - 2137 bytes written, 5 ms

yes
| ?- canget(state(atdoor,onfloor,atwindow,hasnot)).

true ?

yes
| ?- trace.
The debugger will first creep -- showing everything (trace)

yes
{trace}
| ?- canget(state(atdoor,onfloor,atwindow,hasnot)).
1    1  Call: canget(state(atdoor,onfloor,atwindow,hasnot)) ?
2    2  Call: move(state(atdoor,onfloor,atwindow,hasnot),_71,_111) ?
2    2 Exit:
move(state(atdoor,onfloor,atwindow,hasnot),walk(atdoor,_99),state(_99,onfloor,atwindow,hasnot))  ?
3    2  Call: canget(state(_99,onfloor,atwindow,hasnot)) ?
4    3  Call: move(state(_99,onfloor,atwindow,hasnot),_129,_169) ?
4    3  Exit: move(state(atwindow,onfloor,atwindow,hasnot),climb,state(atwindow,onbox,atwindow,hasnot))
```

```
?
5    3  Call: canget(state(atwindow,onbox,atwindow,hasnot)) ?
6    4  Call: move(state(atwindow,onbox,atwindow,hasnot),_184,_224) ?
6    4  Fail: move(state(atwindow,onbox,atwindow,hasnot),_184,_212) ?
5    3  Fail: canget(state(atwindow,onbox,atwindow,hasnot)) ?

4    3 Redo:
move(state(atwindow,onfloor,atwindow,hasnot),climb,state(atwindow,onbox,atwindow,hasnot))
?
4    3 Exit:
move(state(atwindow,onfloor,atwindow,hasnot),push(atwindow,_157),state(_157,onfloor,_157,hasnot
))  ?
5    3  Call: canget(state(_157,onfloor,_157,hasnot)) ?
6    4  Call: move(state(_157,onfloor,_157,hasnot),_187,_227) ?
6    4  Exit: move(state(_157,onfloor,_157,hasnot),climb,state(_157,onbox,_157,hasnot)) ?
7    4  Call: canget(state(_157,onbox,_157,hasnot)) ?
8    5  Call: move(state(_157,onbox,_157,hasnot),_242,_282) ?
8    5  Exit: move(state(middle,onbox,middle,hasnot),grasp,state(middle,onbox,middle,has)) ?
9    5  Call: canget(state(middle,onbox,middle,has)) ?
9    5  Exit: canget(state(middle,onbox,middle,has)) ?
7    4  Exit: canget(state(middle,onbox,middle,hasnot)) ?
5    3  Exit: canget(state(middle,onfloor,middle,hasnot)) ?
3    2  Exit: canget(state(atwindow,onfloor,atwindow,hasnot)) ?
1. 1  Exit: canget(state(atdoor,onfloor,atwindow,hasnot)) ?
```

Signature of the Faculty

4. **Write a Prolog program for depicting and inferring from the given Family relationship diagram.**



Program:

```
% This program depicts family relationships...
/*
domains
name=symbol
predicates
parent(name,name)
female(name)
male(name)
mother(name,name)
father(name,name)
haschild(name)
sister(name,name)
brother(name,name)
clauses
*/
female(pam).
female(liz).
female(pat).
female(ann).
male(jim).
male(bob).
male(tom).
male(peter).
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).

parent(pat,jim).
parent(bob,peter).
parent(peter,jim).
mother(X,Y):-parent(X,Y),female(X).
father(X,Y):-parent(X,Y),male(X).
haschild(X):-parent(X,_).
sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),X\==Y.
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.
```

Output:

```
compiling C:/GNU-Prolog/bin/rel.pl for byte code...
C:/GNU-Prolog/bin/rel.pl compiled, 24 lines read - 3066 bytes written, 10 ms
| ?- parent(X,jim).
X = pat ? ;
X = peter

(16 ms) yes
| ?- mother(X,Y).

X = pam
Y = bob ?

yes
| ?- mother(X,Y).

X = pam
Y = bob ? ;

X = pat
Y = jim ?

yes
| ?- haschild(X).
X = pam ? ;
X = tom ? ;
X = tom ? ;
X = bob ?

yes
| ?- haschild(X).
```

```
X = pam ? ;

X = tom ? ;

X = tom ? ;

X = bob ? ;

X = bob ? ;

X = pat ? ;

X = bob ? ;

X = peter

(47 ms) yes
| ?- brother(X,Y).

X = bob
Y = liz ? ;

X = peter
Y = ann ? ;
X = peter
Y = pat ? ;

(15 ms) no
```

Signature of the Faculty

5. Write a Prolog program for implementing the solution for 8-Puzzle problem.

Program:

```
goal([1,2,3,4,0,5,6,7,8]).

% move empty spot left in the top row
move([P1,0,P3, P4,P5,P6, P7,P8,P9],
[0,P1,P3, P4,P5,P6, P7,P8,P9]).
move([P1,P2,0, P4,P5,P6, P7,P8,P9],
[P1,0,P2, P4,P5,P6, P7,P8,P9]).

% move empty spot left in the middle row
move([P1,P2,P3, P4,0,P6, P7,P8,P9],
[P1,P2,P3, 0,P4,P6, P7,P8,P9]).
move([P1,P2,P3, P4,P5,0, P7,P8,P9],
[P1,P2,P3, P4,0,P5, P7,P8,P9]).

% move empty spot left in the bottom row
move([P1,P2,P3, P4,P5,P6, P7,0,P9],
[P1,P2,P3, P4,P5,P6, 0,P7,P9]).
move([P1,P2,P3, P4,P5,P6, P7,P8,0],
[P1,P2,P3, P4,P5,P6, P7,0,P8]).

% move empty spot right in the top row
move([0,P2,P3, P4,P5,P6, P7,P8,P9],
[P2,0,P3, P4,P5,P6, P7,P8,P9]).
move([P1,0,P3, P4,P5,P6, P7,P8,P9],
[P1,P3,0, P4,P5,P6, P7,P8,P9]).

% move empty spot right in the middle row
move([P1,P2,P3, 0,P5,P6, P7,P8,P9],
[P1,P2,P3, P5,0,P6, P7,P8,P9]).
move([P1,P2,P3, P4,0,P6, P7,P8,P9],
[P1,P2,P3, P4,P6,0, P7,P8,P9]).

% move empty spot RIGHT in the bottom row
move([P1,P2,P3, P4,P5,P6, 0,P8,P9],
[P1,P2,P3, P4,P5,P6, P8,0,P9]).
move([P1,P2,P3, P4,P5,P6, P7,0,P9],
[P1,P2,P3, P4,P5,P6, P7,P9,0]).

% move empty spot UP FROM  the middle row
move([P1,P2,P3, 0,P5,P6, P7,P8,P9],
[0,P2,P3, P1,P5,P6, P7,P8,P9]).
move([P1,P2,P3, P4,0,P6, P7,P8,P9],
[P1,0,P3, P4,P2,P6, P7,P8,P9]).
```

```
move([P1,P2,P3, P4,P5,0, P7,P8,P9],
[P1,P2,0, P4,P5,P3, P7,P8,P9]).
% move empty spot UP FROM the bottom row
move([P1,P2,P3, P4,P5,P6, P7,0,P9],
[P1,P2,P3, P4,0,P6, P7,P5,P9]).
move([P1,P2,P3, P4,P5,P6, P7,P8,0],
[P1,P2,P3, P4,P5,0, P7,P8,P6]).
move([P1,P2,P3, P4,P5,P6, 0,P8,P9],
[P1,P2,P3, 0,P5,P6, P4,P8,P6]).

% move empty spot DOWN FROM in the top row
move([0,P2,P3,  P4,P5,P6, P7,P8,P9],
[P4,P2,P3, 0,P5,P6,  P7,P8,P9]).
move([P1,0,P3,  P4,P5,P6, P7,P8,P9],
[P1,P5,P3, P4,0,P6,  P7,P8,P9]).
move([P1,P2,0,  P4,P5,P6, P7,P8,P9],
[P1,P2,P6, P4,P5,0,  P7,P8,P9]).

% move empty spot DOWN FROM the middle row
move([P1,P2,P3, 0,P5,P6,  P7,P8,P9],
[P1,P2,P3, P7,P5,P6, 0,P8,P9]).
move([P1,P2,P3, P4,0,P6, P7,P8,P9],
[P1,P2,P3, P4,P8,P6, P7,0,P9]).
move([P1,P2,P3, P4,P5,0, P7,P8,P9],
[P1,P2,P3, P4,P5,P9, P7,P8,0]).

dfsSimplest(S, [S]) :- goal(S).
dfsSimplest(S, [S|Rest]) :-
move(S, S2),
 dfsSimplest(S2, Rest).

dfs(S, Path, Path) :- goal(S).

dfs(S,Checked,Path) :-
% Let us try for a move
  move(S,S2),
  % ensure the resulting state is a new state
  \+member(S2,Checked),
  % and that this state leads to the goal
 dfs(S2, [S2|Checked], Path).
```

Output:

```
compiling C:/GNU-Prolog/bin/8puzzle.pl for byte code...
C:/GNU-Prolog/bin/8puzzle.pl:54-55: warning: singleton variables [P9] for move/2
C:/GNU-Prolog/bin/8puzzle.pl compiled, 85 lines read - 23872 bytes written, 12 ms
| ?- dfsSimplest([1,2,3, 0,4,5, 6,7,8], Path).
Path = [[1,2,3,0,4,5,6,7,8],[1,2,3,4,0,5,6,7,8]] ?
yes
| ?- dfs([1,2,3,0,4,5,6,7,8], [], Path).
Path = [[1,2,3,4,0,5,6,7,8]] ?
yes
| ?- dfs([1,2,3, 0,4,5, 6,7,8], [], Path),length(Path,N).
N = 1
Path = [[1,2,3,4,0,5,6,7,8]] ?
```

Signature of the Faculty

6. Construct Prolog program to implement Depth first and Breadth first Search.

Program: DFS

```
s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
s(c,g).
s(d,h).
s(e,i).
s(e,j).
s(f,k).
goal(f).
goal(j).

member(X,[ X | _ ]).
member(X,[_|Tail]):-member(X,Tail).

solve(Node,Solution):-
depthfirst([],Node,Solution).

depthfirst(Path,Node,[Node|Path]):-
goal(Node).
depthfirst(Path,Node,Sol):-
s(Node,Node1),
not(member(Node1,Path)),
depthfirst([Node|Path],Node1,Sol).
```

OUTPUT:

```
solve(a,Sol).
Sol=[j,e,b,a];
```

```
Sol=[f,c,a];
```

Program: BFS

```
s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
s(c,g).
s(d,h).
s(e,i).
s(e,j).
s(f,k).
goal(f).
goal(j).
solve(Start,Solution).
breadthfirst([[Start]],Solution).
breadthfirst([[Node|Path]|_],[Node|Path]):- goal(Node).
breadthfirst([Path|Paths],Solution):-
extend(Path,NewPaths),write(NewPaths),nl,conc(Paths,NewPaths,Paths1),breadthfirst(Paths1,Solution).
extend([Node|Path],NewPaths):-
bagof([NewNode,Node|Path],(s(Node,NewNode),not(member(NewNode,[Node|Path]))),NewPaths),!.
extend(_,[]).

conc([],L,L).
conc([X|L1],L2,[X|L3]):-write('conc'),write(X),write(''),write(L1),write(''),write(L2),conc(L1,L2,L3).
```

Output:

```
solve(a,Solution).
Solution=[j,e,b,a];
Solution=[f,c,a];
```

Signature of the Faculty

DATA SCIENCE : R PROGRAMMING

1.a.Write a R program to create a list containing strings, numbers, vectors and logical values.

Source code:

```
list_data = list("Python", "PHP", c(5, 7, 9, 11), TRUE, 125.17, 75.83)
print("Data of the list:")
print(list_data)
```

Output:

```
[1] "Data of the list:"
[[1]]
[1] "Python"

[[2]]
[1] "PHP"

[[3]]
[1] 5  7 9 11

[[4]]
[1] TRUE

[[5]]
[1] 125.17

[[6]]
[1] 75.83
```

Signature of the Faculty

1.b. Write a R program to merge two given lists into one list.

Source code:

```
n1 = list(1,2,3)
c1 = list("Red", "Green", "Black")
print("Original lists:")
print(n1)
print(c1)
print("Merge the said lists:")
mlist =  c(n1, c1)
print("New merged list:")
print(mlist)
```

Output:

```
[1] "Original lists:"
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[1]]
[1] "Red"
[[2]]
[1] "Green"

[[3]]
[1] "Black"
[1] "Merge the said lists:"
[1] "New merged list:"
[[1]]
[1] 1
[[2]]
[1] 2
[[3]]
[1] 3
[[4]]
[1] "Red"
[[5]]
[1] "Green"
[[6]]
[1] "Black"
```

Signature of the Faculty

1.c Write a R program to create a list containing a vector, a matrix and a list and give names to the elements in the list. Access the first and second element of the list.

Source code:

```
list_data <- list(c("Red","Green","Black"), matrix(c(1,3,5,7,9,11), nrow = 2), list("Python", "PHP", "Java"))
print("List:")
print(list_data)
names(list_data) = c("Color", "Odd numbers", "Language(s)")
print("List with column names:")
print(list_data)
print('1st element:')
print(list_data[1])
print('2nd element:')
print(list_data[2])
```

Output:

```
[1] "List:"
[[1]]
[1] "Red""Green""Black"

[[2]]
[,1] [,2] [,3]
[1,]    1    5    9
[2,]    3    7  11

[[3]]
[[3]][[1]]
[1] "Python"

[[3]][[2]]
[1] "PHP"

[[3]][[3]]
[1] "Java"


[1] "List with column names:"
$Color
[1] "Red""Green""Black"

$`Odd numbers`
[,1] [,2] [,3]
[1,]    1    5    9
[2,]    3    7  11
```

```
$`Language(s)`
$`Language(s)`[[1]]
[1] "Python"

$`Language(s)`[[2]]
[1] "PHP"

$`Language(s)`[[3]]
[1] "Java"


[1] "1st element:"
$Color
[1] "Red""Green""Black"

[1] "2nd element:"
$`Odd numbers`
[,1] [,2] [,3]
[1,]    1    5    9
[2,]    3    7  11
```

Signature of the Faculty

2.a. Write a R program to find factors of a number

```
print_factors <- function(x) {
print(paste("The factors of",x,"are:"))
for(i in 1:x) {
if((x %% i) == 0) {
print(i)
}
}
}
```

Output:

```
> print_factors(120)
[1] "The factors of 120 are:"
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 8
[1] 10
[1] 12
[1] 15
[1] 20
[1] 24
[1] 30
[1] 40
[1] 60
[1] 120
```

Signature of the Faculty

2.b. Write a R program to create an ordered factor from data consisting of the names of months.

```
mons_v = c("March","April","January","November","January",
"September","October","September","November","August","February",
"January","November","November","February","May","August","February",
"July","December","August","August","September","November","September",
"February","April")
print("Original vector:")
print(mons_v)
f = factor(mons_v)
print("Ordered factors of the said vector:")
print(f)
print(table(f))
```

Output:

```
1] "Original vector:"
[1] "March"     "April"     "January"   "November"  "January"   "September"
[7] "October"   "September" "November"  "August"    "February"  "January"
[13] "November"  "November"  "February"  "May"       "August"    "February"
[19] "July"      "December"  "August"    "August"    "September" "November"
[25] "September" "February"  "April"
[1] "Ordered factors of the said vector:"
[1] March     April     January   November  January   September October
[8] September November  August    February  January   November  November
[15] February  May       August    February  July      December  August
[22] August    September November  September February  April
11 Levels: April August December February January July March May ... September
f
April    August  December  February   January     July     March       May
2        4        1        4        3        1        1        1
November  October September
5        1        4
```

Signature of the Faculty

2.c.Write a R program to Read the data from same and different directory.

Read data from current Directory
Determine Your Working Directory: The getwd() function will print your curent working directory to the console:
```
getwd()
## [1] "C:/Users/R"
```
Determine Directory Contents:You can print the contents of your working directory with functions dir()&list.files().
These don’t require additional arguments.
```
dir()
list.files()
## [1] "2018-08-17 Reading in Data.rmd""2018-08-17.R"
## [3] "2018-08-17_Reading_in_Data.rmd"
```

Source code:
```
list.files(path=".", pattern=NULL, all.files=FALSE, full.names=FALSE)
```

output:
```
“a.rtf”
“hello.pdf”
“github”
“Hello.cpp”
```

Read data from different Directory
Changing Working Directories: You can change your working directory with function setwd().
Source code:
```
setwd("C:/R-tutorial")
dir()
list.files()
```
output:
```
“R-tutorial.txt”
dir←”C:/users/Data”
```
Source code:
```
BB←file.path(dir,”bodybuilders.txt”)
BB← read.table(file=bodybuilder,header=TRUE)
```
output:
```
>BB
group          weight    height     proteinintake
bodybuilder      96.6      185.70      167.9
bodybuilder      90.2       179.60      102.4
```

Signature of the Faculty

3.a) Write a R program to read a CSV file

```
data <- read.csv("input.csv")
print(data)
```

Output:

```
 id,   name,    salary,   start_date,     dept
 1      1    Rick     623.30    2012-01-01      IT
 2      2    Dan      515.20    2013-09-23      Operations
 3      3    Michelle 611.00    2014-11-15      IT
 4      4    Ryan     729.00    2014-05-11      HR
 5     NA    Gary     843.25    2015-03-27      Finance
 6      6    Nina     578.00    2013-05-21      IT
 7      7    Simon    632.80    2013-07-30      Operations
 8      8    Guru     722.50    2014-06-17      Finance
```

Signature of the Faculty

3.b Write R program to read Text file

```
    #  Read a text file using read.delim()
myData = read.delim(“1.txt”, header = FALSE)
print(myData)
```

Output:

```
Ok!! Welcome to Data Science
```

Signature of the Faculty

3.c Write a R Program to read and load data from larger datasets.

Source code:

```
library(data.table)
chicrimes←fread("chicrimes.csv")
nrow(chicrimes)
dim(chicrimes)
chicrimes[1:3]
names(chicrimes)
chicrimes(,.N,by=District]
```

output:

```
6450939
6450939 22


   ID                                                 casenumber                         date
8758086                                                HV432922                        08/15/2012
8758087                                                HV432927                        08/15/2012
```

4. Install the necessary R packages and apply data manipulation packages- dplyr, data.table,reshape2, tidyr,Lubridate.

dplyr Package
dplyr package provides various important functions that can be used for Data Manipulation. These are:

filter() Function: For choosing cases and using their values as a base for doing

```
so.# Create a data frame with missing data.
d < - data.frame(name=c("Abhi", "Bhavesh",  "Chaman", "Dimri"),
age=c(7, 5, 9, 16),
ht=c(46, NA, NA,
69),
school=c("yes", "yes", "no", "no"))
d
#   Finding   rows   with   NA
value      d      %      >      %
filter(is.na(ht))
#  Finding  rows  with  no  NA
valued % > % filter(! is.na(ht))
```

Output:

```
  # A tibble: 4 x 4
  name   age    ht school

1 Abhi        7    46 yes
2 Bhavesh     5    NA yes
3 Chaman      9    NA no
4 Dimri      16    69 no

  # A tibble: 2 x 4
  name   age    ht school

1 Bhavesh     5    NA yes
2 Chaman      9    NA no

  # A tibble: 2 x 4
  name age    ht school

1 Abhi      7    46 yes
2 Dimri    16    69 no
```

arrange(): For reordering of the cases.

```
# Create a data frame with missing data
d <- data.frame( name = c("Abhi", "Bhavesh", "Chaman", "Dimri"),
                                    age = c(7, 5, 9, 16),
                                    ht = c(46, NA, NA, 69),
                                    school = c("yes", "yes", "no", "no") )

# Arranging name according to the age
d.name<- arrange(d, age)
print(d.name)
```

Output:

```
# A tibble: 4 x 4
name      age    ht school
1 Bhavesh     5    NA yes
2 Abhi        7    46 yes
3 Chaman      9    NA no
4 Dimri      16    69 no
```

select() and rename(): For choosing variables and using their names as a base for doing so.

```
# Create a data frame with missing data
d < - data.frame(name=c("Abhi", "Bhavesh","Chaman", "Dimri"),
age=c(7, 5, 9, 16),
ht=c(46, NA, NA, 69),
school=c("yes", "yes", "no", "no"))
# startswith() function to print only ht data
select(d, starts_with("ht"))
# -startswith() function to print
# everything except ht data
select(d, -starts_with("ht"))
# Printing column 1 to 2
select(d, 1: 2)
# Printing data of column
# heading containing 'a'
select(d, contains("a"))
# Printing data of column
# heading which matches 'na'
select(d, matches("na"))
```

Output:

```
 # A tibble: 4 x 1
 ht
 1    46
 2    NA
 3    NA
 4    69
 A tibble: 4 x 3
name      age school
1 Abhi        7 yes
2 Bhavesh     5 yes
3 Chaman      9 no
4 Dimri      16 no
# A tibble: 4 x 2
name      age
1 Abhi        7
2 Bhavesh     5
3 Chaman      9
4 Dimri      16
# A tibble: 4 x 2
name      age
1 Abhi        7
2 Bhavesh     5
3 Chaman      9
4 Dimri      16
```

```
# A tibble: 4 x 1
name
1 Abhi
2 Bhavesh
3 Chaman
4 Dimri
```

summarise(): Condensing various values to one value.

```
# Create a data frame with missing data
d <- data.frame( name = c("Abhi", "Bhavesh","Chaman", "Dimri"),
                                    age = c(7, 5, 9, 16),
                                    ht = c(46, NA, NA, 69),
                                    school = c("yes", "yes", "no", "no") )
# Calculating mean of age
summarise(d, mean = mean(age))
# Calculating min of age
summarise(d, med = min(age))

# Calculating max of age
summarise(d, med = max(age))
# Calculating median of age
summarise(d, med = median(age))
```

Output:

```
# A tibble: 1 x 1
```

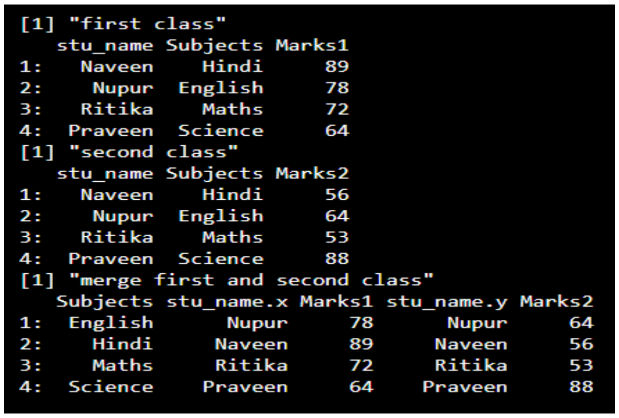
```
mean
1     9.25
# A tibble: 1 x 1
med
1     5



# A tibble: 1 x 1
2
3     med
4     116
5     # A tibble: 1
      x 1med
6     1 8
```

data.table

The purpose of data.table is to create tabular data.

Syntax:

install.packages('data.table')
To merge two data.table objects
Load data.table package
library("data.table")
print("first class")
# Create first data.table
class1 <- data.table(stu_name = c('Naveen','Nupur','Ritika','Praveen'), Subjects = c('Hindi','English','Maths','Science'), Marks1 = c(89,78,72,64))
# Print first data.table
print(class1)
print("second class")
# Create second data.table
class2 <- data.table(stu_name = c('Naveen','Nupur','Ritika','Praveen'), Subjects = c('Hindi','English','Maths','Science'), Marks2 = c(56,64,53,88))
# Print second data.table
print(class2)
print("merge first and second class")
# Merge data.tables
merge_class <- merge.data.table(class1, class2, by.x = "Subjects", by.y = "Subjects")
# Print merged data.table
print(merge_class)

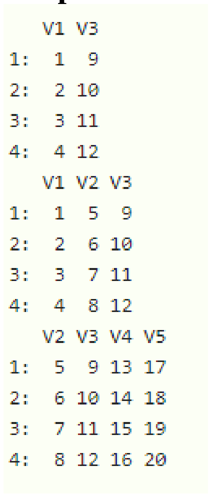


```
[1] "first class"
   stu_name Subjects Marks1
1:   Naveen    Hindi     89
2:    Nupur  English     78
3:   Ritika    Maths     72
4:  Praveen  Science     64
[1] "second class"
   stu_name Subjects Marks2
1:   Naveen    Hindi     56
2:    Nupur  English     64
3:   Ritika    Maths     53
4:  Praveen  Science     88
[1] "merge first and second class"
   Subjects stu_name.x Marks1 stu_name.y Marks2
1:  English      Nupur     78      Nupur     64
2:    Hindi     Naveen     89     Naveen     56
3:    Maths     Ritika     72     Ritika     53
4:  Science    Praveen     64    Praveen     88
```

select subset of columns from the data table

```
# load data.table package
library("data.table")
# create data table with matrix with 20 elements
# 4 rows and 5 columns
data= data.table(matrix(1:20, nrow=4,ncol = 5))
# display the subset that include v1 and v3 columns
print(data[ , c("V1", "V3"), with = FALSE])
# display the subset that include v1 , v2 and v3 columns
print(data[ , c("V1","V2", "V3"), with = FALSE])
# display the subset that include v2,v3,v4 and v5 columns
print(data[ , c("V2", "V3","V4","V5"), with = FALSE])
```

Output:



```
   V1 V3
1:  1  9
2:  2 10
3:  3 11
4:  4 12
   V1 V2 V3
1:  1  5  9
2:  2  6 10
3:  3  7 11
4:  4  8 12
   V2 V3 V4 V5
1:  5  9 13 17
2:  6 10 14 18
3:  7 11 15 19
4:  8 12 16 20
```

reshape2

The reshape2 package facilitates extracting values such as means from many columns quickly and easily.

```
library(reshape2)
mydata <- data.frame(id=c(1,1,2,2),time=c(1,2,1,2),x1=c(5,3,6,2),x2=c(6,5,1,4))
```

means of each of the “x” columns (x1, x2) over just the id. Without reshape, it would look like this:

```
aggregate(cbind(x1, x2) ~ id, mydata, mean)
## id x1 x2
## 1  1  4 5.5
## 2  2  4 2.5
```

This works, but we must specifiy the name of every “x” column. Suppose there are hundreds, and we want to calculate all of them automatically? This is where reshape2 helps. First, we “melt” the data:

```
melt_mydata <- melt(mydata, id=c("id", "time"))
melt_mydata
##  id time variable value
## 1  1    1       x1     5
## 2  1    2       x1     3
## 3  2    1       x1     6
## 4  2    2       x1     2
## 5  1    1       x2     6
## 6  1    2       x2     5
## 7  2    1       x2     1
## 8  2    2       x2     4
```

Now that the data is melted, you can easily get the means for each melted column, regardless of how many there are.

```
means_by_id <- dcast(melt_mydata, id ~ variable, mean)
means_by_id
## id x1 x2
## 1  1  4 5.5
## 2  2  4 2.5
```

The old reshape package had a cast method. Note that reshape2 has an acast method, which returns a vector/matrix/array, and dcast, which returns a data.matrix.

Note that not need to specify the names of the melted columns. If you wanted to restrict them, means for all of the “x” columns except x2.

```
means_by_id <- dcast(melt_mydata[melt_mydata$variable!=c("x2"),], id ~ variable, mean)
means_by_id
## id x1
## 1  1  4
## 2  2  4
```

But what if we wanted means by the time column, instead of id? No problem.

```
means_by_time <- dcast(melt_mydata, time ~ variable, mean)
means_by_time
##  time  x1  x2
## 1    1 5.5 3.5
## 2    2 2.5 4.5
```

Now just to demonstrate a bit more of the flexibility, let's expand the data set.

```
mydata <- data.frame(id=c(1,1,2,2),time=c(1,2,1,2),x1=c(3,4,7,8),x2=c(4,1,0,3),x3=c(7,1,9,11),x4=c(3,2,7,9))
mydata
##  id time x1 x2 x3 x4
## 1  1    1  3  4  7  3
## 2  1    2  4  1  1  2
## 3  2    1  7  0  9  7
## 4  2    2  8  3 11  9
```

Now we are up to four "x" columns. Let's try the same approach as before.

```
melt_mydata <- melt(mydata, id=c("id", "time"))
means_by_id <- dcast(melt_mydata, id ~ variable, mean)
means_by_id
## id x1 x2 x3  x4
## 1  1 3.5 2.5  4 2.5
## 2  2 7.5 1.5 10 8.0
```

tidyr

The sole purpose of the **tidyr** package is to simplify the process of creating tidy data. Tidy data describes a standard way of storing data that is used wherever possible throughout the **tidyverse**

```
install.packages("tidyverse")
```

alternatively, to install just **tidyr** package type this:

```
install.packages("tidyr")
```

Define a dataset **tidy_dataframe** that contains data about the frequency of people in a particular group.

```
# load the tidyr package
library(tidyr)
n = 10
# creating a data frame
tidy_dataframe = data.frame(
                              S.No = c(1:n),
                              Group.1 = c(23, 345, 76, 212, 88,
                                                  199, 72, 35, 90, 265),
                              Group.2 = c(117, 89, 66, 334, 90,
                                            101, 178, 233, 45, 200),
                              Group.3 = c(29, 101, 239, 289, 176,
                                                  320, 89, 109, 199, 56))

# print the elements of the data frame
tidy_dataframe
```

Output:

```
S.No Group.1 Group.2 Group.3
1    1      23     117      29
2    2     345      89     101
3    3      76      66     239
4    4     212     334     289
5    5      88      90     176
6    6     199     101     320
7    7      72     178      89
8    8      35     233     109
9    9      90      45     199
10   10    265     200      56
```

gather() function: It takes multiple columns and gathers them into key-value pairs. Basically it makes “wide” data longer. The **gather()** function will take multiple columns and collapse them into key-value pairs, duplicating all other columns as needed.

```
# using gather() function on tidy_dataframelong <- tidy_dataframe %>%
                                gather(Group, Frequency,
                                            Group.1:Group.3)
# print the data frame in a long format
long
```

Output:

```
S.No Group  Frequency
1     1 Group.1       23
2     2 Group.1      345
3     3 Group.1       76
4     4 Group.1      212
5     5 Group.1       88
6     6 Group.1      199
7     7 Group.1       72
8     8 Group.1       35
9     9 Group.1       90
10   10 Group.1      265
11    1 Group.2      117
12    2 Group.2       89
13    3 Group.2       66
14    4 Group.2      334
15    5 Group.2       90
16    6 Group.2      101
17    7 Group.2      178
18    8 Group.2      233
19    9 Group.2       45
20   10 Group.2      200
21    1 Group.3       29
22    2 Group.3      101
23    3 Group.3      239
24    4 Group.3      289
25    5 Group.3      176
26    6 Group.3      320
```

```
27    7 Group.3        89
28    8 Group.3       109
29    9 Group.3       199
30   10 Group.3        56
```

separate() function: It converts longer data to a wider format. The **separate()** function turns a single character column into multiple columns.

```
# import tidyr package
library(tidyr)
long <- tidy_dataframe %>%
                              gather(Group, Frequency,
                                          Group.1:Group.3)
# use separate() function to make data wider
separate_data <- long %>%
                              separate(Group, c("Allotment",
                                                                              "Number"))
# print the wider format
separate_data
```

Output:

```
S.No Allotment Number Frequency
1    1    Group     1       23
2    2    Group     1      345
3    3    Group     1       76
4    4    Group     1      212
5    5    Group     1       88
6    6    Group     1      199
7    7    Group     1       72
8    8    Group     1       35
9    9    Group     1       90
10   10    Group     1      265
11    1    Group     2      117
12    2    Group     2       89
13    3    Group     2       66
14    4    Group     2      334
15    5    Group     2       90
16    6    Group     2      101
17    7    Group     2      178
18    8    Group     2      233
19    9    Group     2       45
20   10    Group     2      200
21    1    Group     3       29
22    2    Group     3      101
23    3    Group     3      239
24    4    Group     3      289
25    5    Group     3      176
26    6    Group     3      320
27    7    Group     3       89
28    8    Group     3      109
29    9    Group     3      199
30   10    Group     3       56
```

Lubridate

Lubridate is an R package that makes it easier to work with dates and times.

```
install.packages("lubridate")
1.      ymd(20101215)
 #> [1] "2010-12-15"
mdy("4/1/17")
#> [1] "2017-04-01"
```

Simple functions to get and set components of a date-time, such as year(), month(), mday(), hour(), minute() and second():

```
bday <- dmy("14/10/1979")
month(bday)
#> [1] 10
wday(bday, label = TRUE)
#> [1] Sun
#> Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
year(bday) <- 2016
wday(bday, label = TRUE)
#> [1] Fri
#> Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

Helper functions for handling time zones: with_tz(), force_tz()

```
time <- ymd_hms("2010-12-13 15:30:30")
time
#> [1] "2010-12-13 15:30:30 UTC"
# Changes printing
with_tz(time, "America/Chicago")
#> [1] "2010-12-13 09:30:30 CST"
force_tz(time, "America/Chicago")
#> [1] "2010-12-13 15:30:30 CST
```

Signature of the Faculty

5 Write R Programs to implement decision tree and K-Nearest Neighbor algorithms.

Use the below command in R console to install the package. Install the dependent packages if any.

```
install.packages("party")
```

The package "party" has the function **ctree()** which is used to create and analyze decison tree.

Syntax

The basic syntax for creating a decision tree in R is −

```
ctree(formula, data)
```

Following is the description of the parameters used −

- **formula** is a formula describing the predictor and response variables.
- **data** is the name of the data set used.

data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker or not.

Here is the sample data.

```
# Load the party package. It will automatically load other
# dependent packages.
library(party)
# Print some records from data set readingSkills.
print(head(readingSkills))
```

it produces the following result and chart −

```
nativeSpeaker   age  shoeSize     score
1          yes     5  24.83189  32.29385
2          yes     6  25.95238  36.63105
3           no    11  30.42170  49.60593
4          yes     7  28.66450  40.28456
5          yes    11  31.88207  55.46085
6          yes    10  30.07843  52.83124
Loading required package: methods
Loading required package: grid
..............................
..............................
```

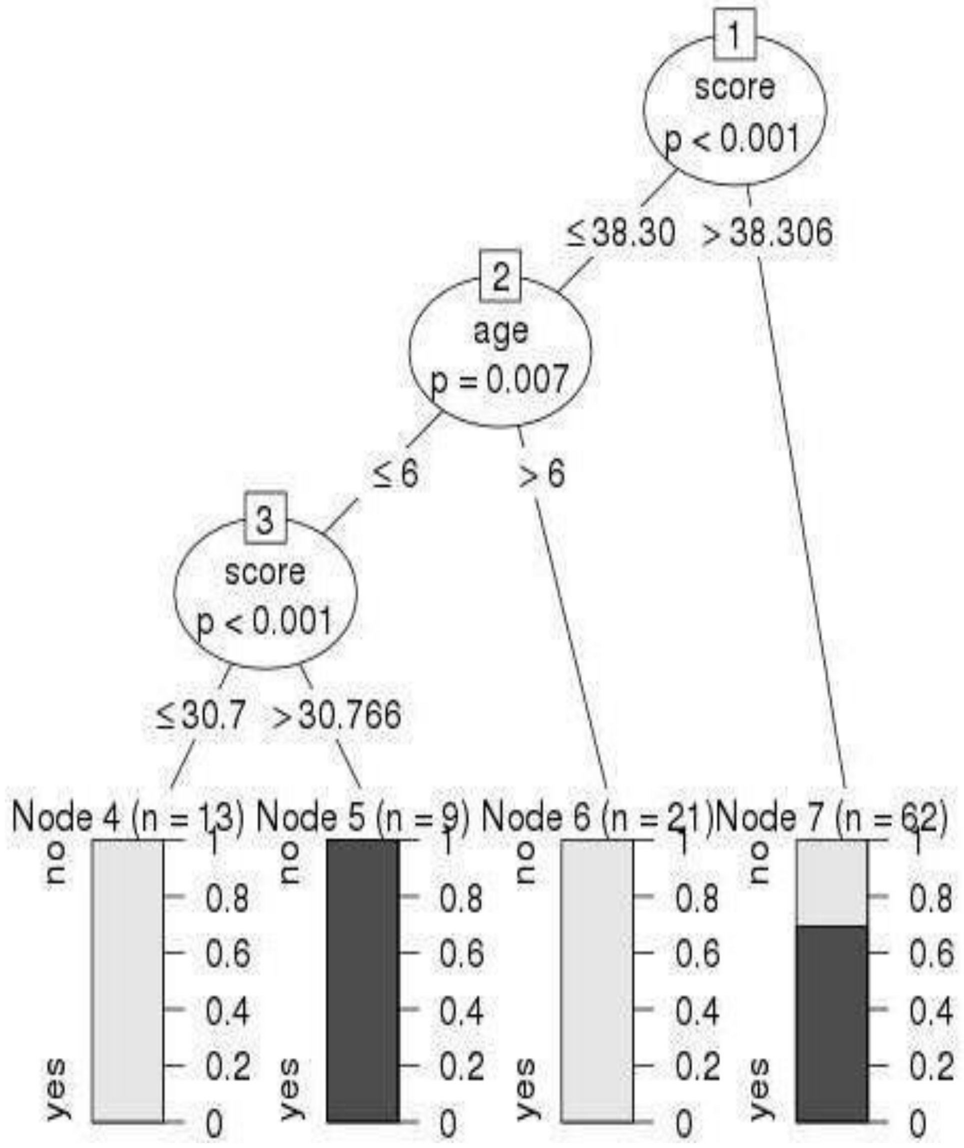
- **Example**

Use the **ctree()** function to create the decision tree and see its graph.

```
# Load the party package. It will automatically load other
# dependent packages.
library(party)

# Create the input data frame.
input.dat <- readingSkills[c(1:105),]
# Give the chart file a name.
png(file = "decision_tree.png")
# Create the tree.
output.tree <- ctree(
nativeSpeaker ~ age + shoeSize + score,
data = input.dat)
# Plot the tree.
plot(output.tree)
# Save the file.
dev.off()
```

it produces the following result −

```
null device
1
Loading required package: methods
Loading required package: grid
Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4
Loading required package: strucchange
Loading required package: zoo
Attaching package: ‘zoo’
The following objects are masked from ‘package:base’:
as.Date, as.Date.numeric
Loading required package: sandwich
```


1
score
p < 0.001
≤ 38.30
> 38.306
2
age
p = 0.007
≤ 6
> 6
3
score
p < 0.001
≤ 30.7
> 30.766
Node 4 (n = 13)
Node 5 (n = 9)
Node 6 (n = 21)
Node 7 (n = 62)
no
yes
1
0.8
0.6
0.4
0.2
0

Signature of the Faculty

K-Nearest Neighbor algorithm

K-Nearest Neighbor or K-NN is a Supervised Non-linear classification algorithm. K-NN is a Non-parametric algorithm i.e it doesn’t make any assumption about underlying data or its distribution. It is one of the simplest and widely used algorithm which depends on it’s k value(Neighbors) and finds it’s applications in many industries like finance industry, healthcare industry etc.

In the KNN algorithm, K specifies the number of neighbors and its algorithm is as follows:

1. Choose the number K of neighbor.
2. Take the K Nearest Neighbor of unknown data point according to distance.
3. Among the K-neighbors, Count the number of data points in each category.
4. Assign the new data point to a category, where you counted the most neighbors.

The Dataset

Iris dataset consists of 50 samples from each of 3 species of Iris(Iris setosa, Iris virginica, Iris versicolor) and a multivariate dataset introduced by British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems. Four features were measured from each sample i.e length and width of the sepals and petals and based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

```
# Loading
data
data(iris)
# Structure
str(iris)
```

Performing K Nearest Neighbor on Dataset

Using the K-Nearest Neighbor algorithm on the dataset which includes 11 persons and 6 variables or attributes.

```
# Installing Packages
install.packages("e1071")
install.packages("caTools")
install.packages("class")
# Loading package
library(e1071)
library(caTools)
library(class)
# Loading data
data(iris)
head(iris)
# Splitting data into train
# and test data
split <- sample.split(iris, SplitRatio = 0.7)
train_cl <- subset(iris, split == "TRUE")
test_cl <- subset(iris, split == "FALSE")
# Feature Scaling
train_scale <- scale(train_cl[, 1:4])
test_scale <- scale(test_cl[, 1:4])
# Fitting KNN Model
# to training dataset

classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 1)
classifier_knn
```

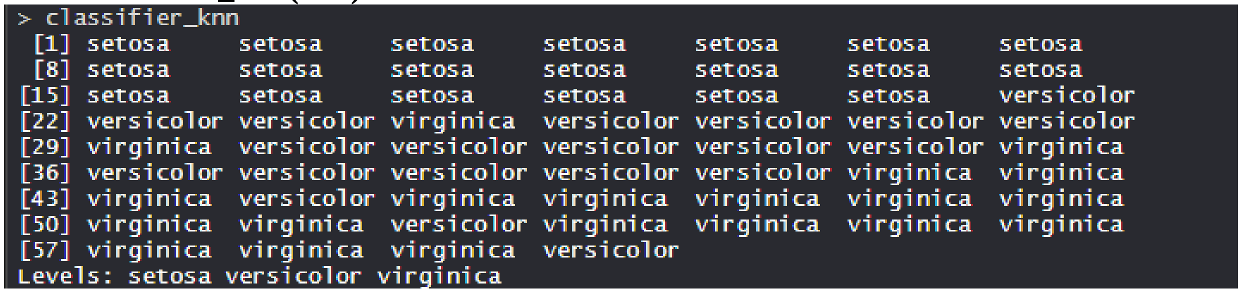
# Confusion Matrix

```
cm <- table(test_cl$Species, classifier_knn)
cm
# Model Evaluation - Choosing K
# Calculate out of Sample error
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))
# K = 3
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 3)
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))
# K = 5
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 5)
```

```
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))
# K = 7
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 7)
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))
# K = 15
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 15)
misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))

# K = 19
classifier_knn <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$Species,
                      k = 19)

misClassError <- mean(classifier_knn != test_cl$Species)
print(paste('Accuracy =', 1-misClassError))
```

Output:

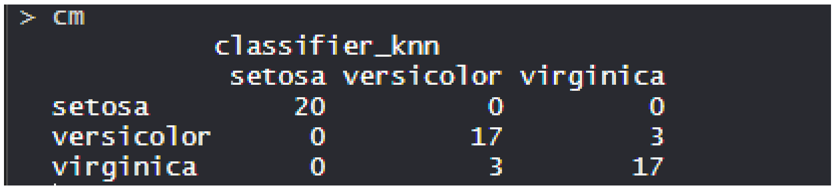
1. **Model classifier_knn(k=1):**



```
> classifier_knn
 [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa
 [8] setosa     setosa     setosa     setosa     setosa     setosa     setosa
[15] setosa     setosa     setosa     setosa     setosa     setosa     versicolor
[22] versicolor versicolor virginica  versicolor versicolor versicolor versicolor
[29] virginica  versicolor versicolor versicolor versicolor versicolor virginica
[36] versicolor versicolor versicolor versicolor versicolor virginica  virginica
[43] virginica  versicolor virginica  virginica  virginica  virginica  virginica
[50] virginica  virginica  versicolor virginica  virginica  virginica  virginica
[57] virginica  virginica  virginica  versicolor
Levels: setosa versicolor virginica
```

The KNN model is fitted with a train, test, and k value. Also, the Classifier Species feature is fitted in the model.

confusion Matrix:

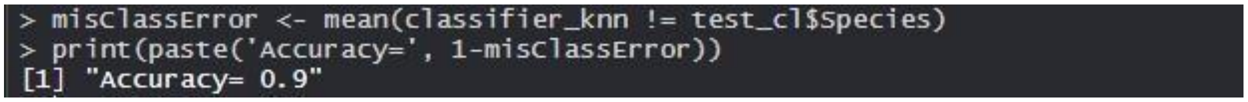


```
> cm
            classifier_knn
             setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         17         3
  virginica       0          3        17
```

So, 20 Setosa are correctly classified as Setosa. Out of 20 Versicolor, 17 Versicolor are correctly classified as Versicolor and 3 are classified as virginica. Out of 20 virginica, 17 virginica are correctly classified as virginica and 3 are classified as Versicolor.

Model Evaluation:

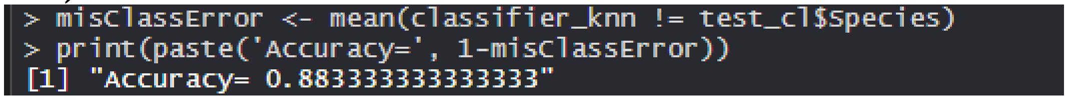
(k=1)



```
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy=', 1-misClassError))
[1] "Accuracy= 0.9"
```

The model achieved 90% accuracy with k is 1.

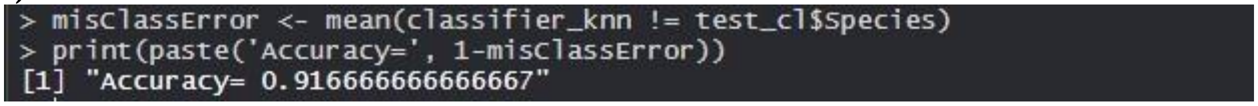
K=3)



```
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy=', 1-misClassError))
[1] "Accuracy= 0.883333333333333"
```

The model achieved 88.33% accuracy with k is 3 which is lower than when k was 1.

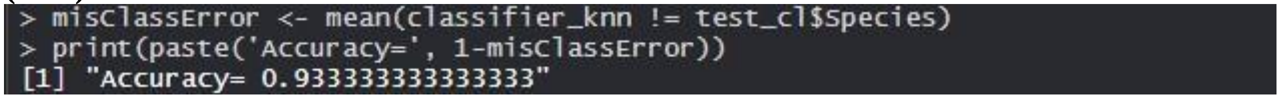
(K=5)



```
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy=', 1-misClassError))
[1] "Accuracy= 0.916666666666667"
```

The model achieved 91.66% accuracy with k is 5 which is more than when k was 1 and 3.

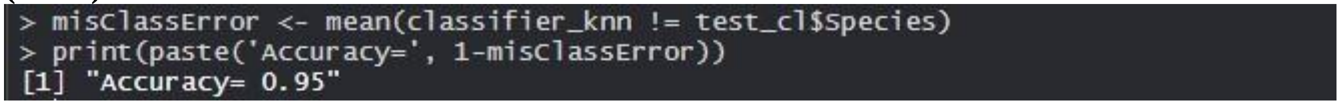
(K=7)



```
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy=', 1-misClassError))
[1] "Accuracy= 0.933333333333333"
```

The model achieved 93.33% accuracy with k is 7 which is more than when k was 1, 3, and 5.

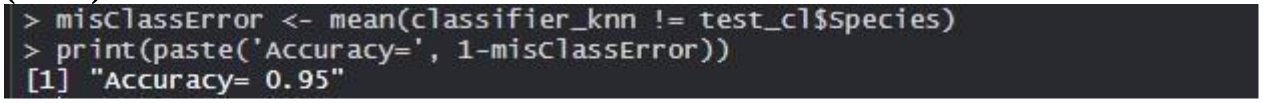
(K=15)



```
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy=', 1-misClassError))
[1] "Accuracy= 0.95"
```

The model achieved 95% accuracy with k is 15 which is more than when k was 1, 3, 5, and 7.

(K=19)



```
> misClassError <- mean(classifier_knn != test_cl$Species)
> print(paste('Accuracy=', 1-misClassError))
[1] "Accuracy= 0.95"
```

The model achieved 95% accuracy with k is 19 which is more than when k was 1, 3, 5, and 7.
Its same accuracy when k was 15 which means now increasing k values doesn't affect the accuracy.
So, K Nearest Neighbor is widely used in the industry.

Signature of the Faculty

6. Build a linear regression model and logistic regression model, check the model on a test data andpredict the numerical quantities.

Linear Regression: It is a commonly used type of predictive analysis. It is a statistical approach for modeling the relationship between a dependent variable and a given set of independent variables.

Input Data

Below is the sample data representing the observations −# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is
−lm(formula,data)

Following is the description of the parameters used −

1. **formula** is a symbol presenting the relation between x and y.
2. **data** is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

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

it produces the following result

```
−Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)        x
-38.4551      0.6746
```

Get the Summary of the Relationship

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

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

it produces the following result

```
−Call:
lm(formula = y ~
x)Residuals:
```

```
Min     1Q     Median      3Q     Max
-6.3002  -1.6629  0.0412    1.8944  3.9775
Coefficients:
Estimate Std. Error t value Pr(>|t|)

(Intercept) -38.45509 8.04901  -4.778  0.00139 **
x           0.67461    0.05191 12.997 1.16e-06 ***
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,  Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict()

FunctionSyntax

The basic syntax for predict() in linear regression is
−predict(object, newdata)
Following is the description of the parameters used −

1. **object** is the formula which is already created using the lm() function.
2. **newdata** is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm()
function.relation <-
lm(y~x)
# Find weight of a person with height
170.a <- data.frame(x = 170)
result <-
predict(relation,a)
print(result)
```

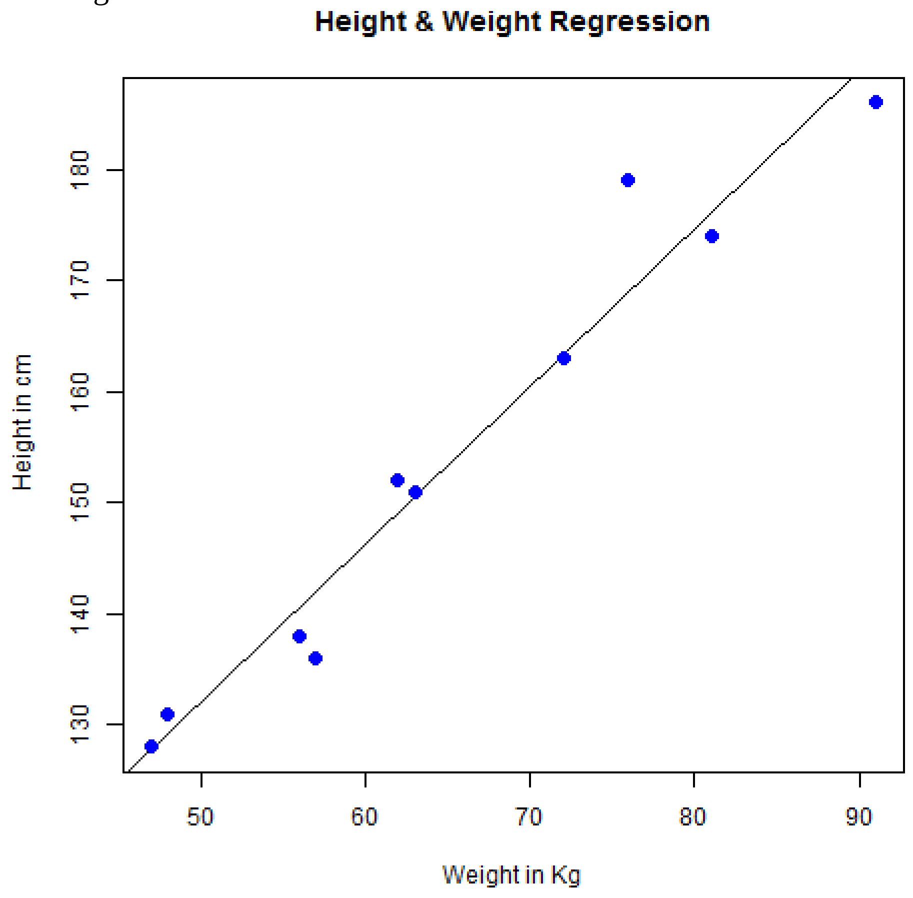
it produces the following result

```
−1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)
# Give the chart file a name.
png(file =
"linearregression.png")# Plot
the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression",
```

abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")# Save the file.
dev.off()
it produces the following result −



The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is −y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))

Following is the description of the parameters used −

1. **y** is the response variable.
2. **x** is the predictor variable.
3. **a** and **b** are the coefficients which are numeric constants.

The function used to create the regression model is the **glm()** function.

Syntax

The basic syntax for **glm()** function in logistic regression is − glm(formula,data,family)

Following is the description of the parameters used −

1. **formula** is the symbol presenting the relationship between the variables.
2. **data** is the data set giving the values of these variables.

3. **family** is R object to specify the details of the model. It's value is binomial for logistic regression.

Example

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wtand cyl.

```
# Select some columns form mtcars.
input <- mtcars[,c("am","cyl","hp","wt")]

print(head(input))
```

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

```
am   cyl  hp    wt
Mazda RX4          1   6    110 2.620
Mazda RX4 Wag      1   6    110  2.875
Datsun 710         1   4     93 2.320
Hornet 4 Drive     0   6    110 3.215
Hornet Sportabout  0   8    175 3.440
Valiant            0   6    105  3.460
```

Create Regression Model

We use the **glm()** function to create the regression model and get its summary for analysis.

```
Input <- mtcars[,c("am","cyl","hp","wt")]

am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)

print(summary(am.data))
```

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

```
Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)
 Deviance Residuals:
Min        1Q      Median        3Q       Max
-2.17272     -0.14907  -0.01464     0.14116  1.27641
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) 19.70288    8.11637  2.428  0.0152 *
cyl          0.48760    1.07162   0.455  0.6491
hp           0.03259    0.01886   1.728 0.0840 .
wt          -9.14947    4.15332  -2.203  0.0276 *
  Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

  (Dispersion parameter for binomial family taken to be 1)

   Null deviance: 43.2297 on 31 degrees of freedom
   Residual deviance: 9.8415  on 28  degrees of freedom
   AIC: 17.841

   Number of Fisher Scoring iterations: 8
```

Signature of the Faculty