GURU TEGH BAHADUR 4TH CENTENARY ENGINEERING COLLEGE

G-8 AREA, RAJOURI GARDEN, NEW DELHI-110064

DATA SCIENCE PRACTICAL FILE

Course Code: CIE-405P

Semester: 7th

Submitted to :

Geeta Yadav

Submitted by :

SAIMA

03423802722

CSE 1

GURU TEGH BAHADUR 4TH CENTENARY ENGINEERING COLLEGE

G-8 AREA, RAJOURI GARDEN, NEW DELHI-110064

DATA SCIENCE PRACTICAL FILE

Course Code: CIE-405P

Semester: 7th

Submitted to :	Submitted by :
Geeta Yadav	Kanishka Bhatt
	01623802722
	CSE - 01

S.NO.	AIM OF EXPERIMENT	DATE OF EXPERIMENT	SIGNATURE
1.	Installation of R and Basic Syntax	29-08-2025	
2.	Describing Data, Viewing, and Manipulating Data Using R	29-08-2025	
3.	Plotting Probability Distribution Curve in R	12-09-2025	
4.	Implementing Linear Regression in R	19-09-2025	
5.	Logistic Regression	19-09-2025	
6.	Principal Component Analysis (PCA) on dataset using R.	26-09-2025	
7.	Chi-Square Test in R	26-09-2025	
8.	Edit and Execute a Program Involving Functions in R	17-10-2025	
9.	Edit and Execute a Program Involving Flow Chart	17-10-2025	
10.	Data Aggregation and Group Operations in R	24-10-2025	

Experiment 1: Installation of R and Basic Syntax.

AIM: To install R and learn the basic syntax of R programming.

Theory

R is a statistical programming language widely used for data analysis and visualization. The installation of R and RStudio provides an environment to write, edit, and execute R programs.

Basic syntax includes variables, arithmetic operations, vectors, sequences, and printing results.

Program (R code):

```
# Basic syntax demonstration in R

# Assigning values

x <- 25

y <- 15

# Arithmetic operations

sum_result<- x + y

diff_result<- x - y

prod_result<- x * y

div_result<- x / y

# Creating a vector

numbers<- c(2, 4, 6, 8, 10)

# Generating a sequence

sequence<- seq(1, 20, by = 2)

# Mean and Sum of vector

mean_val<- mean(numbers)

sum_val<- sum(numbers
# Displaying results
print(paste("Sum of x and y is:", sum_result))
print(paste("Difference of x and y is:", diff_result))
print(paste("Product of x and y is:", prod_result))
print(paste("Division of x and y is:", div_result))
print("Vector values:")
print(numbers)
print("Sequence from 1 to 20 with step 2:")
print(sequence)
print(paste("Mean of vector:", mean_val))
print(paste("Sum of vector:", sum_val))
```

Expected Output:

```
[1] "Sum of x and y is: 40"
[1] "Difference of x and y is: 10"
[1] "Product of x and y is: 375"
[1] "Division of x and y is: 1.6667"
[1] "Vector values:"
[1]  2  4  6  8 10
[1] "Sequence from 1 to 20 with step 2:"
[1]  1  3  5  7  9 11 13 15 17 19
[1] "Mean of vector: 6"
[1] "Sum of vector: 30"
```

Conclusion:

The installation of R was successfully verified. Basic syntax like variables, arithmetic operations, vectors, and sequences were executed correctly.

Experiment 2: Describing Data, Viewing, and Manipulating Data Using R

AIM: To learn how to describe, view, and manipulate datasets in R.

Theory

R provides various data structures such as vectors, lists, and data frames. Data can be viewed using head(), tail(), and described using functions like summary() and str(). Manipulations include column selection, row filtering, and transformations.

Program (R code):

```
# Creating a sample data frame

students<- data.frame(

RollNo = 1:5,

  Name = c("Aman", "Riya", "Kunal", "Simran", "Arjun"),

  Marks = c(85, 92, 76, 89, 95),

  Age = c(20, 21, 20, 22, 21)

)

# Viewing data

print("Complete Data Frame:")

print(students)

print("First 3 rows:")

print(head(students, 3))

print("Last 2 rows:")

print(tail(students, 2)

# Descriptive statistics

print("Summary of dataset:")
print(summary(students))
```

# Manipulation - selecting a column

```
print("Marks Column:")
print(students$Marks)
```

# Filtering data

```
print("Students with Marks > 85:")
print(subset(students, Marks > 85))
```

Output (Sample):

```
[1] "Complete Data Frame:"
RollNo   Name Marks Age
1      1   Aman85  20
2      2   Riya92  21
3      3  Kunal    76  20
4      4 Simran89  22
5      5  Arjun    95  21
[1] "First 3 rows:"
RollNo  Name Marks Age
1      1  Aman    85  20
2      2  Riya    92  21
3      3 Kunal76  20
[1] "Last 2 rows:"
RollNo   Name Marks Age
4      4 Simran89  22
5      5  Arjun    95  21
```

```
[1] "Summary of dataset:"

RollNo      Name              Marks          Age

Min.:1   Length:5          Min.   :76.0   Min.   :20.0

 1st Qu.:2   Class :character   1st Qu.:85.0   1st Qu.:20.0

Median :3   Mode  :character   Median :89.0   Median :21.0

 Mean   :3                     Mean   :87.4   Mean   :20.8

 3rd Qu.:4                     3rd Qu.:92.0   3rd Qu.:21.0

 Max.   :5                    Max.   :95.0   Max.   :22.0

[1] "Marks Column:"

[1] 85 92 76 89 95

[1] "Students with Marks > 85:"

RollNo   Name Marks Age

2      2   Riya92  21

4      4 Simran89  22

5      5  Arjun    95  21
```

Conclusion:

The dataset was successfully created, viewed, and manipulated in R using data frames, column selection, and filtering.

Experiment 3: Plotting Probability Distribution Curve in R

AIM: To plot and visualize probability distribution curves (Normal distribution) using R software.

Theory

A probability distribution describes how values of a random variable are spread across possible outcomes.

Normal Distribution (Gaussian Distribution): Symmetrical, bell-shaped curve.

Defined by two parameters:

Mean (μ): Determines the center/shift of the curve.

Standard Deviation (σ): Determines the spread/width of the curve.

Properties of Normal Distribution:

The total area under the curve = 1.

R Functions for Probability Distribution:

dnorm(x, mean, sd) → Probability density at value x.

pnorm(x) → Cumulative probability up to x.

rnorm(n, mean, sd) → Generates n random values.

hist() and plot() → Visualize data.

ggplot2 package → Advanced plotting with better visuals.

Program (Rcode):

```
# Installpackageifnotalreadyinstalled

# install.packages("ggplot2")
```

```
library(ggplot2)

# Createasequenceofvalues
x<- seq(-5, 5, by = 0.1)

# Createdataframeformultipledistributions
df<- data.frame(
x = rep(x, 3),
density = c(dnorm(x, mean = 0, sd = 1),
dnorm(x, mean = 0, sd = 2),
dnorm(x, mean = 2, sd = 1)),
dist = factor(rep(c("mean=0, sd=1", "mean=0, sd=2", "mean=2, sd=1"),
each = length(x)))
)

# Plotdistributioncurves
ggplot(df, aes(x = x, y = density, color = dist)) +
geom_line(size = 1.2) +
labs(title = "NormalDistributionCurvesusingggplot2",
x = "Xvalues", y = "ProbabilityDensity") +
theme_minimal()

# --- HistogramExamplewithggplot2 ---
# Generate1000randomnormalvalues
random_data<- data.frame(values = rnorm(1000, mean = 0, sd = 1)
# Plothistogramwithdensityoverlay
ggplot(random_data, aes(x = values)) +
geom_histogram(aes(y = ..density..), bins = 30, fill = "skyblue", color = "black") +
geom_density(color = "red", size = 1.2) +
labs(title = "HistogramwithNormalCurve (ggplot2)",
x = "RandomValues", y = "Density") +
theme_minimal()
```

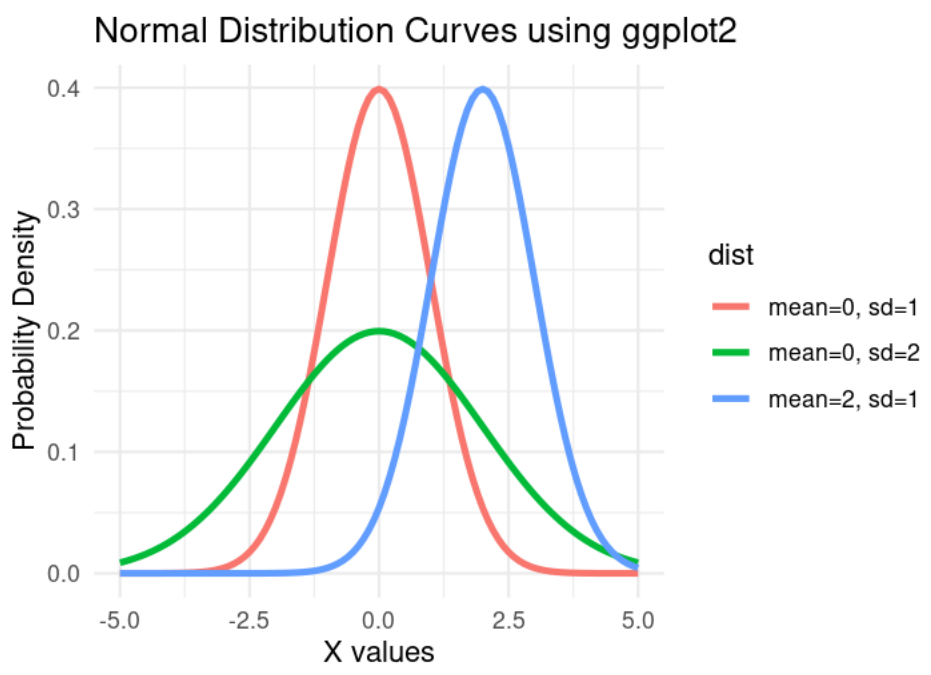
ExpectedOutput:

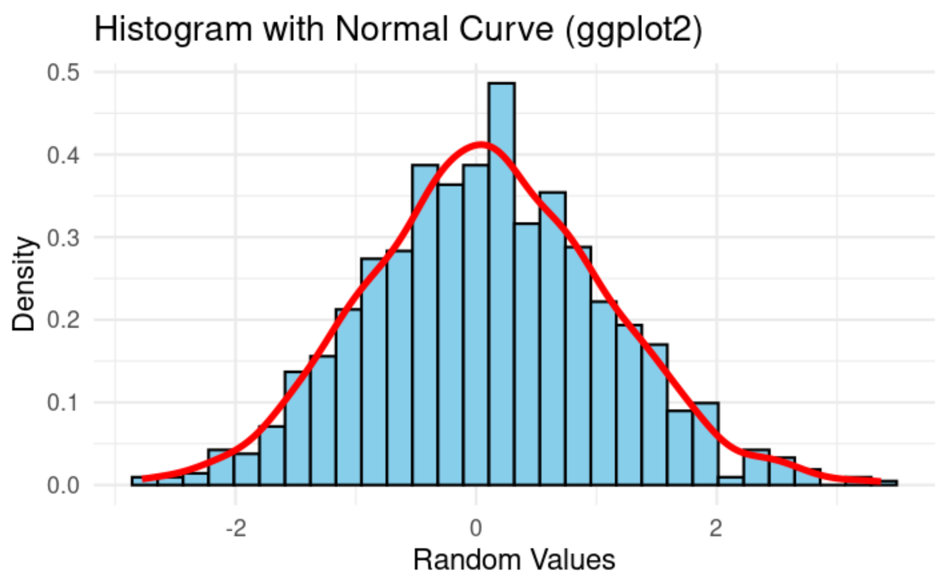
LineGraph → 3 curves (blue, red, green) showing different means & standard deviations.

Histogram → Bars in light blue with a smooth red density curve overlaid.

Conclusion :

Using ggplot2, the probability distribution curves and histograms look more professional and clear compared to base R. This demonstrates both traditional and modern plotting methods in R.


Normal Distribution Curves using ggplot2
Probability Density
0.4
0.3
0.2
0.1
0.0
-5.0
-2.5
0.0
2.5
5.0
X values
dist
mean=0, sd=1
mean=0, sd=2
mean=2, sd=1


Histogram with Normal Curve (ggplot2)
Density
0.5
0.4
0.3
0.2
0.1
0.0
-2
0
2
Random Values

Experiment 4: Implementing Linear Regression in R

AIM:

Toimplementsimplelinearregressionin Randunderstandhowtomodeltherelationshipbetweentwovar iables.

Theory

RegressionAnalysisisastatisticalmethodformodelingrelationshipsbetweenvariables.

InR:

lm(Y ~ X, data) → fits a linear model.

summary(model) → displays coefficients and accuracy.

plot() and abline() → visualize regression line.

Program (R code):

```
# Simple Linear Regression in R


# Create sample dataset

study_hours<- c(2, 3, 4, 5, 6, 7, 8)

marks<- c(50, 55, 60, 65, 70, 75, 80)


data<- data.frame(study_hours, marks)


# Fit linear regression model

model<- lm(marks ~ study_hours, data = data)


# Display summary

print(summary(model))
```

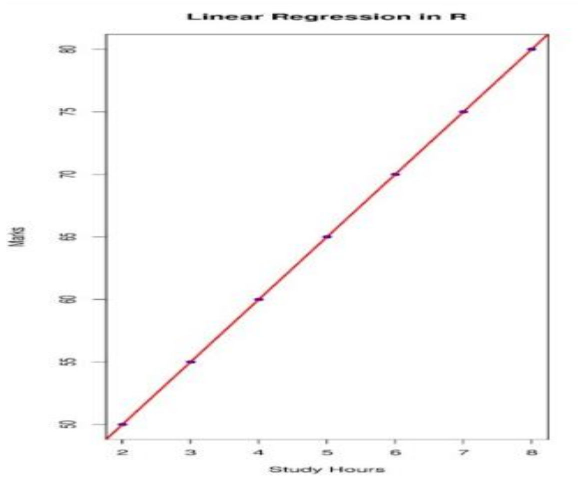
```
# Plot scatter plot with regression line
plot(data$study_hours, data$marks,
col = "blue", pch = 16,
main = "Linear Regression in R",
xlab = "Study Hours",
ylab = "Marks")
abline(model, col = "red", lwd = 2)

# Predict marks for new data
new_hours<- data.frame(study_hours = c(9, 10))
predicted_marks<- predict(model, new_hours)
print("Predicted Marks for 9 and 10 hours of study:")
print(predicted_marks)
```

Expected Output:

Summary → Regression coefficients (Intercept ≈ 45, Slope ≈ 5).

Plot → Blue points (data), red regression line.

Predictions → Marks ≈ 85 (for 9 hrs), 90 (for 10 hrs).



Conclusion:

Linear regression successfully modeled the relationship between study hours and marks. The model predicts that every additional study hour increases marks by about 5.

Experiment 5: Logistic Regression

Aim: ToimplementlogisticregressioninRandanalyzetherelationshipbetweenstudyhoursandexamresult (pass/fail).

Theory

LogisticRegressionisastatisticalmethodusedforclassificationproblemswherethedependentvariableiscategorical (e.g., 0 = Fail, 1 = Pass).

Itmodelstheprobabilityofaneventoccurringusingthesigmoid (logistic) function:

Unlike linear regression, the output of logistic regression is always between 0 and 1, making it suitable for probability prediction.

It is widely used in fields like medicine, machine learning, and social sciences for binary classification problems.

Program (R Code)

```
# Logistic Regression Example


# Dataset: study hours vs pass (0=Fail, 1=Pass)

study_hours<- c(1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 9, 10)

pass<-      c(0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1)


data<- data.frame(study_hours, pass)


# Fit logistic regression model

model<- glm(pass ~ study_hours, data = data, family = binomial)


# Show summary

print(summary(model))
```

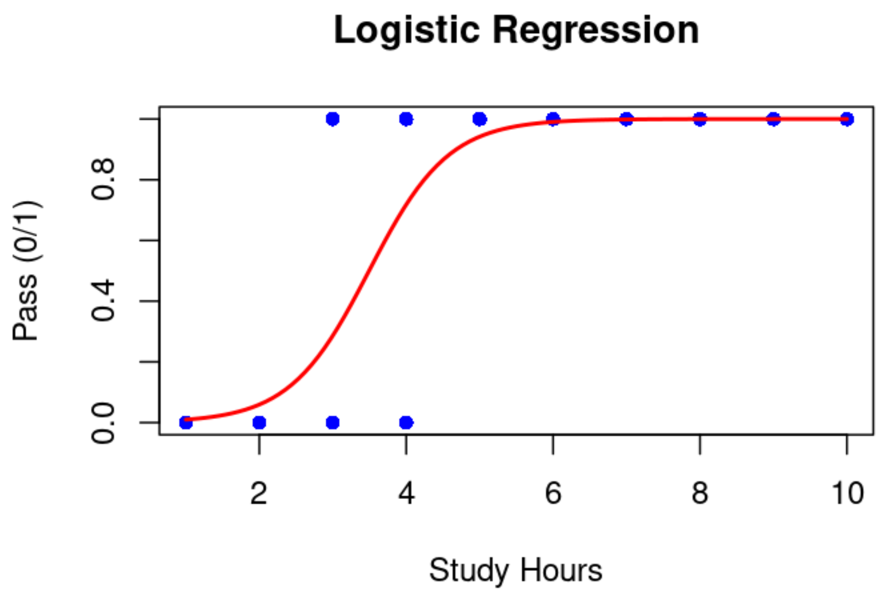
```
# Predict probabilities

data$predicted_prob<- predict(model, type = "response")

print(data)


# Plot logistic regression curve

plot(data$study_hours, data$pass,

col = "blue", pch = 16,

xlab = "Study Hours", ylab = "Pass (0/1)",

main = "Logistic Regression")

curve(predict(model, data.frame(study_hours = x), type = "response"),

add = TRUE, col = "red", lwd = 2)
```

Expected Output

Summary of Model – Coefficients for intercept and slope showing how study hours affect passing probability.

Predicted Probabilities – Each student’s probability of passing.

Graph – Blue points (actual data) and a red S-shaped curve showing logistic regression fit.



Conclusion

Logistic regression successfully models the relationship between study hours and exam results. As the number of study hours increases, the probability of passing the exam also increases. The S-shaped logistic curve clearly shows the transition from low pass probability (for fewer hours) to high pass probability (for more hours).

Experiment 6: Principal Component Analysis (PCA) on dataset using R.

Aim: ToperformPrincipalComponentAnalysis (PCA) onadatasetinRandvisualizetheresults.

Theory

PrincipalComponentAnalysis (PCA) isadimensionalityreductiontechnique.

Ittransformscorrelatedvariablesintoasetofuncorrelatedvariablescalledprincipalcomponents.

Properties: Thefirstprincipalcomponentcapturesthemaximumvarianceinthedata.

Eachsubsequentcomponentcapturestheremainingvariancewhilebeingorthogonaltothepreviousones.

Applications:

Reducing dataset dimensions.

Removing redundancy.

Data visualization and preprocessing for machine learning.

Program (RCode)

```
# PCAinR


# Load dataset (built-inirisdataset)

data(iris)


# Remove the categorical column (Species) forPCA

iris_data<- iris[, 1:4]


# PerformPCA (scale=TRUEstandardizesdata)

pca_result<- prcomp(iris_data, scale. = TRUE)
```

```
# PrintPCAsummary

summary(pca_result)


# Print loadings (rotationmatrix)

print(pca_result$rotation)


# BiplotofPCA

biplot(pca_result, col = c("red", "blue"), main = "PCABiplotofIrisData")


# Screeplot (varianceexplainedbyeachcomponent)

plot(pca_result, type = "l", main = "ScreePlotofPCA")
```

ExpectedOutput

Summary of PCA – Shows standard deviation, proportion of variance explained by each component, and cumulative variance.

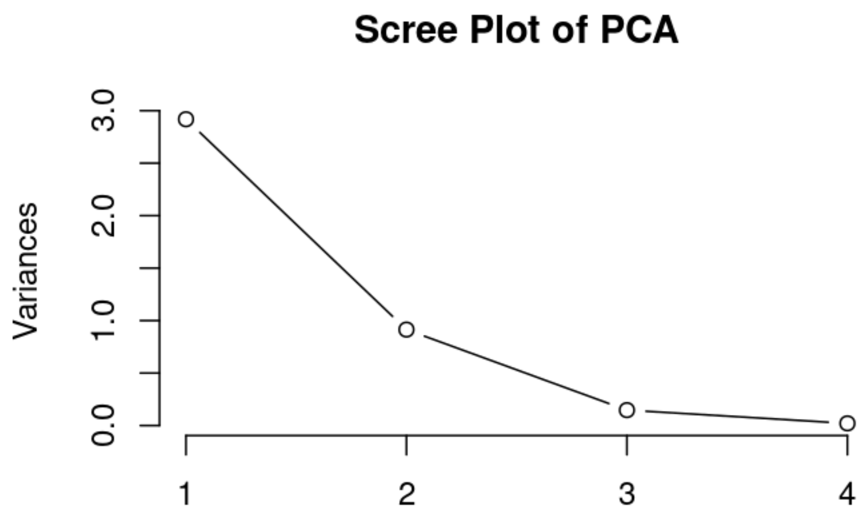
Loadings (rotation matrix) – Shows contribution of each original variable to principal components.

Biplot – Graph with data points and variable vectors showing direction of variance.

Scree Plot – Line graph showing how much variance is explained by each principal component.

Conclusion

PCA reduces the dimensionality of the dataset while retaining most of the information. In the iris dataset, the first two principal components capture most of the variance, making it possible to visualize a 4-dimensional dataset in just 2 dimensions without losing much information.


Scree Plot of PCA
Variances
0.0
1.0
2.0
3.0
1
2
3
4

Experiment 7: Chi-Square Test in R

Aim: ToperformtheChi-Squaretestondifferentdatasetsin Rtocheck the associationbetweencategoricalvariables.

Theory

TheChi-SquareTestisanon-parametricsstatisticaltestusedtocheckifthereisasignificantassociationbetweentwocategoricalvariables.

Types of Chi-Square tests:

Goodness of Fit Test – Checks if sample data matches expected distribution.

Test of Independence – Checks if two categorical variables are related.

Applications:

Market research.

Genetics.

Survey data analysis.

Program (R Code)

1. Chi-Square Goodness of Fit Test

```
# Observed frequencies (e.g., dice roll results)

observed<- c(18, 22, 20, 16, 24, 20)


# Expected frequencies (fair dice → equal probability)

expected<- rep(sum(observed) / 6, 6)
```

```
# Perform Chi-Square test
chisq_test1 <- chisq.test(x = observed, p = rep(1/6, 6))

print(chisq_test1)
```

2. Chi-Square Test of Independence

```
# Example dataset: Gender vs Preference
data<- matrix(c(30, 10,   # Male: Like, Dislike
                20, 20),  # Female: Like, Dislike
nrow = 2, byrow = TRUE)

colnames(data) <- c("Like", "Dislike")
rownames(data) <- c("Male", "Female")

print(data)

# Perform Chi-Square test
chisq_test2 <- chisq.test(data)

print(chisq_test2)
```

Expected Output

Goodness of Fit Test – p-value shows whether dice is fair.

Test of Independence – p-value indicates whether gender and preference are related.

Conclusion

The Chi-Square test helps determine if differences between observed and expected frequencies are due to chance or a real relationship. In practical applications, it is used for survey data analysis and testing associations between categorical variables.

Experiment 8: Edit and Execute a Program Involving Functions in R

Aim: Tocreate, edit, andexecuteaprograminRusingfunctionsformodularprogramming.

Theory

AfunctioninRisablockofcodethatperformsaspecifictask.

Functionsimprovemodularity, reusability, andreadabilityofcode.

StructureofafunctioninR:

```
function_name<- function(arguments) {
  # bodyofthefunction
return(result)
}
```

Rhastwotypesoffunctions:

Built-infunctions (e.g., mean(), sum())

User-definedfunctions (writtenbytheprogrammer)

Inthisexperiment, wedefineauser-definedfunctiontocalculatethefactorialofanumber.

Program (RCode)

```
# Definefactorialfunction
factorial_fun<- function(n) {
if (n == 0) {
return(1)
  } else {
result<- 1
for (iin1:n) {
result<- result*i
    }
return(result)
  }
}

# Testthefunction
num<- 5
cat("Factorialof", num, "is", factorial_fun(num), "\n")

num<- 0
cat("Factorialof", num, "is", factorial_fun(num), "\n")
```

ExpectedOutput

```
Factorialof5is120
Factorialof0is1
```

Conclusion

Thefactorialprogramwassuccessfullyimplementedin Rusingauser-definedfunction.Thisdemonstratestheuseofmodularprogramming, wherefunctionsmakecodeeasiertoedit, reuse, andexecutefordifferentinputs.

Experiment 9: Edit and Execute a Program Involving Flow Chart

Aim: To design a flow chart and implement its equivalent program in R for decision making.

Theory

A flow chart is a step-by-step diagram that shows the flow of control in a program.

Symbols used:

Oval → Start / End

Rectangle → Process / Task

Diamond → Decision (Yes/No)

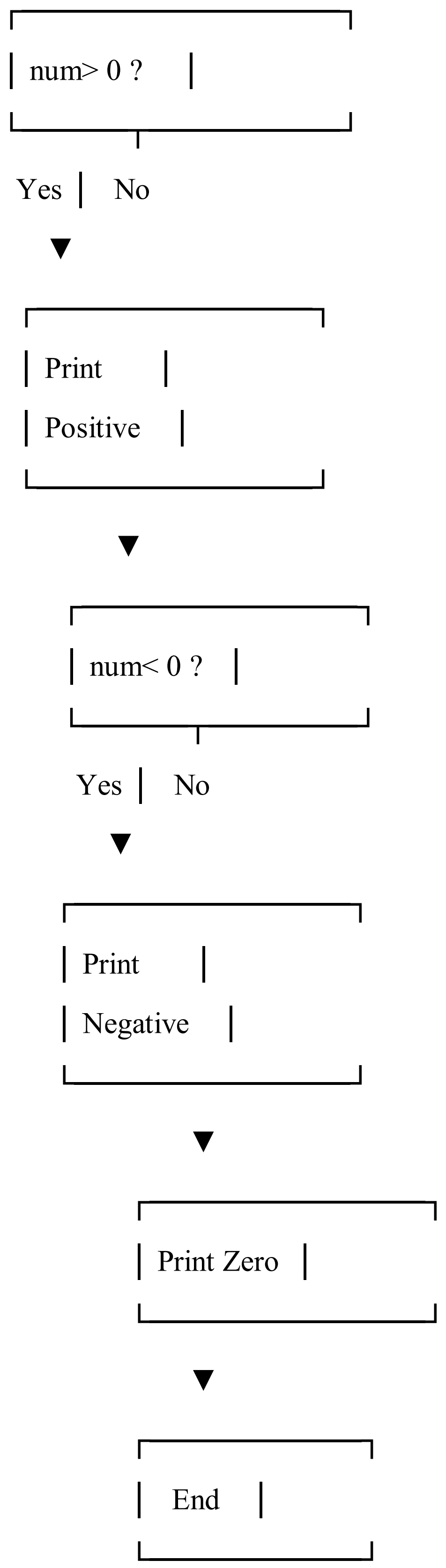
Arrow → Flow direction

Flow charts make program logic easier to understand before writing code.

In this practical, we check whether a number is positive, negative, or zero.

Flow Chart (Logic)

```
 ┌─────────────┐
│ Start │
 └───────┬─────┘
    │
    ▼
┌──────────────────┐
│ Input number │
└────────┬─────────┘
   │
   ▼
```


num> 0 ?
Yes
No
Print
Positive
num< 0 ?
Yes
No
Print
Negative
Print Zero
End

Program (R Code)

⚡ Corrected for Posit Cloud (no input issues):

```
# Assign number directly (change value for testing)
num<- -3   # try 0 or 5 also

# Decision making using if-else
if (num > 0) {
print("The number is Positive")
} else if (num < 0) {
print("The number is Negative")
} else {
print("The number is Zero")
}
```

Expected Output

Case 1:

```
num<- 5
[1] "The number is Positive"
```

Case 2:

```
num<- -3
[1] "The number is Negative"
```

Case 3:

```
num<- 0
[1] "The number is Zero"
```

Conclusion

The flow chart was successfully implemented in R.

By using if–else conditions, we checked whether a number is positive, negative, or zero.

This practical shows how algorithm design using flow charts can be directly translated into executable R programs.

Experiment 10: Data Aggregation and Group Operations in R

Aim: To perform data aggregation and group operations on datasets using R software.

Theory

Data aggregation is the process of summarizing data, such as calculating sums, means, or counts over groups of observations.

Group operations allow us to divide the dataset into groups based on one or more variables and then apply summary functions.

Common R functions/packages used:

aggregate() – base R function for group-wise operations

tapply() – applies a function to subsets of a vector

dplyr package (group_by(), summarise()) – modern, faster way for aggregation

Program (R Code)

```
install.packages("dplyr")

library(dplyr)

# Sample dataset: Employee salary data

data<- data.frame(

  Department = c("HR", "HR", "IT", "IT", "Finance", "Finance", "Finance"),

  Employee = c("A", "B", "C", "D", "E", "F", "G"),

  Salary = c(40000, 45000, 60000, 62000, 50000, 52000, 48000)

)


print("Original Dataset:")

print(data)


# 1. Using aggregate() - average salary by department
```

```
avg_salary<- aggregate(Salary ~ Department, data = data, FUN = mean)
print("Average Salary by Department:")
print(avg_salary)

# 2. Using tapply() - total salary by department
total_salary<- tapply(data$Salary, data$Department, sum)
print("Total Salary by Department:")
print(total_salary)

# 3. Using dplyr for group operations
library(dplyr)

summary_data<- data %>%
group_by(Department) %>%
summarise(
Avg_Salary = mean(Salary),
Max_Salary = max(Salary),
Min_Salary = min(Salary),
Total_Salary = sum(Salary)
  )

print("Department-wise Summary using dplyr:")
print(summary_data)
```

Expected Output

Original Dataset

```
  Department Employee Salary
1         HR        A  40000
2         HR        B  45000
3         IT        C  60000
4         IT        D  62000
5    Finance        E  50000
6    Finance        F  52000
7    Finance        G  48000
```

Average Salary by Department

```
  Department Salary
1    Finance  50000
2         HR  42500
3         IT  61000
```

Total Salary by Department

```
 Finance      HR     IT
 150000    85000  122000
```

Department-wise Summary using dplyr

```
# Atibble: 3 × 5
  Department Avg_SalaryMax_SalaryMin_SalaryTotal_Salary
<chr><dbl><dbl><dbl><dbl>
1 Finance       50000     52000     48000      150000
2 HR            42500     45000     40000       85000
3 IT            61000     62000     60000      122000
```

Conclusion

Data aggregation and group operations were successfully performed using aggregate(), tapply(), and dplyr.

This practical demonstrates how datasets can be summarized efficiently to extract meaningful insights.