RAJALAKSHMI ENGINEERING COLLEGE

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

AD23431-STATISTICAL ANALYSIS AND COMPUTING LAB

[Regulation 2023]

II Year – IV Semester

LABORATORY MANUAL

RAJALAKSHMI ENGINEERING COLLEGE

VISION & MISSION

OF

RAJALAKSHMI ENGINEERING COLLEGE

Vision

To be an institution of excellence in Engineering, Technology and Management Education & Research. To provide competent and ethical professionals with a concern for society.

Mission

To impart quality technical education imbibed with proficiency and humane values. To provide right ambience and opportunities for the students to develop into creative, talented and globally competent professionals. To promote research and development in technology and management for the benefit of the society.

RAJALAKSHMI ENGINEERING COLLEGE

B. E. ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

VISION:

- To promote highly Ethical and Innovative Computer Professionals through excellence in teaching, training and research.

MISSION:

- To produce globally competent professionals, motivated to learn the emerging technologies and to be innovative in solving real world problems.
- To promote research activities amongst the students and the members of faculty that could benefit the society.
- To impart moral and ethical values in their profession.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Graduates will demonstrate their technical skills and competency in various applications through the use of Artificial Intelligence and Data Science.

PEO 2: To produce motivated graduates with capability to apply acquired knowledge and skills in data analytics and visualization to develop viable systems.

PEO 3: Graduates will establish their knowledge by adopting Artificial Intelligence and Data Science technologies to solve the real world problems

PEO 4: To produce graduates with potential to participate in life-long learning through professional developments for societal needs with ethical values.

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of Mathematics, Science, Engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: 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.

PO3: 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.

PO 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.

PO 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.

PO 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.

PO 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.

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

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

PO 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.

PO11: 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 multidisciplinary environments.

PO12: 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.

PROGRAM SPECIFIC OUTCOMES (PSOs)

A graduate of the Artificial Intelligence and Data Science Learning Program will demonstrate.

PSO 1: Foundation Skills: Apply computing theory, languages and algorithms, as well as mathematical and statistical models, and the principles of optimization to appropriately formulate and use data analysis.

PSO 2: Problem-Solving Skills: The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business automation.

PSO 3: Successful Progression: Ability to critique the role of information and analytics for an innovative career, research activities and consultancy.

Syllabus

Subject Code	Subject Name(Lab oriented Theory Courses)	Category	L	T	P	C
AD23431	STATISTICAL ANALYSIS AND COMPUTING	PE	3	0	0	3

Objectives:

- To impart knowledge about the basics of R programming
- To analyze the data using R programming
- To learn the appropriate statistical test and analyses.
- To implement data classification models and learns about different classification algorithms.
- To learn the different regression and clustering algorithms..

UNIT-I	**Introduction to R Programming Structures and Functions**	9
Overview of R programming – Basic objects:Vectors– Matrix–Array–Lists–Factors– Data frames. Basic expressions: Arithmetic expressions – Control Statements: if and if-else statements –– switch statement – Loops: for loop – while loop – Functions– Strings		
UNIT-II	**Introduction to Data Science**	9
Introduction – Roles of Data Science Projects – Data Collection and Management – Modeling – Model Evaluation and Critique – Determining lower and upper bounds – Loading Data into R – Working with data from files and relational database - Exploring data – Managing data: Missing values – cleaning data		
UNIT-III	**Statistical Analysis**	9
Frequency distribution - Measures of central tendency and dispersion – Hypothesis Testing: Test Statistics – ANOVA – F-Test – TTest – U-Test – Fisher's Exact Test – Kruskal- Wallis Test – Bartlett's Test – Statistical Distribution: Binomial – Poisson – Normal – Chi-squared distribution		
UNIT-IV	**Classification**	9
Tests and Training splits- Building Single Variable Model: Categorical Features- Numerical Features – Cross Validation - Building Multi Variable Model: Variable Selection – Decision Trees – Nearest Neighbor Methods – Naïve Baye		
UNIT-V	**Regression and Clustering**	9
Linear and Logistic Regression: Introduction – Building Model – Making Predictions – Characterizing Co-efficient quality – Unsupervised Methods: Cluster Analysis – Distance – Hierarchical Clustering – The K-means Algorithm		

	Contact Hours	**:**	**45**
		Contact hours :	30
		Total Contact hours :	75

List of Experiments	
1	Implement simple programs in R 3. 4. 5. 6 7. 8.
2	Perform data preprocessing in R
3	Perform statistical analysis for a given dataset
4	Implement decision tree algorithm in R
5	Implement K-Nearest Neighbor algorithm in R
6	Implement Naive Bayesian classifier in R
7	Implement linear regression in R
8	Implement K-means clustering algorithm in R
9	Implementation of Searching and Sorting algorithms
10	Hashing –Linear probing

Requirements

Hardware	Intel i3, CPU @ 1.20GHz 1.19 GHz, 4 GB RAM, 32 Bit Operating System
Software	R
Operating System	Windows

AD23431-STATISTICAL ANALYSIS AND COMPUTING LAB PLAN

Exercise no.	Exercise Name	Required Hours
1	Implement simple programs in R 3. 4. 5. 6 7. 8.	3
2	Perform data preprocessing in R	3
3	Perform statistical analysis for a given dataset	3
4	Implement decision tree algorithm in R	3
5	Implement K-Nearest Neighbor algorithm in R	3
6	Implement Naive Bayesian classifier in R	3
7	Implement linear regression in R	3
8	Implement K-means clustering algorithm in R	3
9	Implementation of Searching and Sorting algorithms	3
10	Hashing –Linear probing	3

Course Outcomes (COs)

Course Name: Statistical Analysis and Computing Lab **Course Code: AD23431**

Outcome 1	Solve problems using the fundamentals of R
Outcome 2	Explore and manage data using R
Outcome 3	Perform statistical analysis using R
Outcome 4	Demonstrate Decision Tree, Nearest Neighbor, Naïve bayes classification algorithms
Outcome 5	Apply regression and clustering algorithms for the sample dataset using R

CO-PO –PSO matrices of course

PO/PSO CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
AD23431.1	3	2	2	2	2	-	-	-	-	-	-	2	3	2	-
AD23431.2	3	2	2	2	2	-	-	-	-	-	-	2	3	2	-
AD23431.3	2	2	1	2	2	-	-	-	-	-	-	2	2	2	-
AD23431.4	3	2	2	2	2	-	-	-	-	-	-	2	3	2	-
AD23431.5	2	2	2	2	1	-	-	-	-	-	-	2	3	3	-
Average	3	2	2	2	2	-	-	-	-	-	-	2	3	2	-

Note: Enter correlation levels 1,2or3 as defined below:

1:Slight(Low) 2: Moderate(Medium) 3:Substantial(High)Ifthere isnocorrelation, put“-“

Ex No: 1

IMPLEMENT SIMPLE PROGRAMS IN R

Aim:

To implement simple programs using R

Procedure:

Program 1: Calculate the Sum of Numbers

```
# Program to calculate the sum of numbers from 1 to 10
numbers <- 1:10
sum_numbers <- sum(numbers)

# Output the sum
cat("The sum of numbers from 1 to 10 is:", sum_numbers, "\n")
```

Output:

The sum of numbers from 1 to 10 is: 55

Program 2: Calculate Factorial of a Number

```
# Function to calculate factorial of a number
factorial <- function(n) {
  if (n == 0 || n == 1) {
    return(1)
  } else {
    return(n * factorial(n - 1))
  }
}

# Calculate factorial of 5
result <- factorial(5)
```

```r
# Output the factorial
cat("Factorial of 5 is:", result, "\n")
```

Output:

Factorial of 5 is: 120

Program 3: Print Fibonacci Series

```r
# Function to print Fibonacci series up to n terms
fibonacci <- function(n) {
  fib <- numeric(n)
  fib[1] <- 0
  if (n > 1) {
    fib[2] <- 1
    for (i in 3:n) {
      fib[i] <- fib[i-1] + fib[i-2]
    }
  }
  return(fib)
}

# Print Fibonacci series up to 10 terms
fib_series <- fibonacci(10)
cat("Fibonacci series up to 10 terms:", paste(fib_series, collapse = ", "), "\n")
```

Output:

Fibonacci series up to 10 terms: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Program 4: Generate Random Numbers and Compute Mean

```r
# Generate 10 random numbers from uniform distribution
set.seed(123)  # For reproducibility
random_numbers <- runif(10)
```

```
# Compute mean of random numbers
mean_value <- mean(random_numbers)

# Output random numbers and their mean
cat("Generated random numbers:", random_numbers, "\n")
cat("Mean of the random numbers:", mean_value, "\n")
```

Output:

Generated random numbers: 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673 0.0455565 0.5281055 0.892419 0.551435 0.4566147
Mean of the random numbers: 0.5748054

These programs demonstrate basic operations in R such as arithmetic calculations, recursive function (factorial), iterative function (Fibonacci series), and statistical computations (mean of random numbers). Adjustments can be made to these programs based on specific requirements or additional functionalities needed.

Result:

Thus different simple programs are implemented using R

Ex No: 2

PERFORM DATA PREPROCESSING IN R

Aim:

To perform preprocessing of data using R

Procedure:

Step-by-Step Data Preprocessing in R

Step 1: Loading Data

First, we load the dataset and display its structure to understand its columns and initial format.

```
# Load the dataset (using mtcars dataset for demonstration)
data(mtcars)
head(mtcars)
```

Output:

```
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag    21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710       22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
...
```

Step 2: Handling Missing Values

Check for missing values (NA) in the dataset and decide how to handle them. For demonstration purposes, we assume there are no missing values in mtcars.

```
# Check for missing values
any(is.na(mtcars))
```

Output:

[1] FALSE

Step 3: Handling Categorical Variables

If the dataset has categorical variables, convert them to factors if needed. In mtcars, there are no explicit categorical variables, but if there were, you would convert them using as.factor().

```
# Example: Convert a hypothetical categorical variable to factor
# mtcars$cyl <- as.factor(mtcars$cyl)
```

Step 4: Scaling Numeric Data (Optional)

Scaling numeric variables is necessary for some algorithms but not always required. Here, we'll scale the numeric columns disp, hp, drat, wt, and qsec using scale() function.

```
# Scale numeric columns (optional)
num_cols <- c("disp", "hp", "drat", "wt", "qsec")
mtcars_scaled <- as.data.frame(scale(mtcars[, num_cols]))

# Combine scaled numeric columns with non-numeric columns
mtcars_processed <- cbind(mtcars_scaled, mtcars[, -(which(names(mtcars) %in%
num_cols))])
head(mtcars_processed)
```

Output (scaled numeric data combined with original non-numeric columns):

```
disp       hp      drat        wt      qsec  mpg cyl vs am gear carb
Mazda RX4     0.5795256  0.6641339  0.5765943 -0.6201678 -0.7896003 21.0   6  0  1    4  4
Mazda RX4 Wag 0.5795256  0.6641339  0.5765943 -0.3553820 -0.4712017 21.0   6  0  1 4 4
Datsun 710   -1.0060260 -0.5110630  0.4815843 -0.9316786  0.4328237 22.8   4  1  1    4    1
...
```

Step 5: Splitting Data into Training and Testing Sets

Split the dataset into training and testing sets for model training and evaluation purposes.

```
# Split data into 80% training and 20% testing
set.seed(123)  # For reproducibility
train_indices <- sample(nrow(mtcars), 0.8 * nrow(mtcars))
train_data <- mtcars[train_indices, ]
test_data <- mtcars[-train_indices, ]

# Display dimensions of training and testing sets
cat("Training data dimensions:", dim(train_data), "\n")
cat("Testing data dimensions:", dim(test_data), "\n")
```

Output (dimensions of training and testing sets):

```
Training data dimensions: 25 11
Testing data dimensions: 7 11
```

Result:

Thus preprocessing data using R is a cleaned, transformed, and formatted dataset ready for analysis or modeling.

Ex No: 3

PERFORM STATISTICAL ANALYSIS FOR A GIVEN DATASET

Aim:

To perform statistical analysis for given dataset.

Procedure:

- **plot() Function:** This function is used to Draw a scatter plot with axes and titles.

Syntax:

plot(x, y = NULL, ylim = NULL, xlim = NULL, type = "b"….)

- **data() function:** This function is used to load specified data sets.

Syntax:

data(list = character(), lib.loc = NULL, package = NULL…..)

- **table() Function:** The table function is used to build a contingency table of the counts at each combination of factor levels.

```
table(x, row.names = NULL, ...)
```

- **barplot() Function:** It creates a bar plot with vertical/horizontal bars.

Syntax:

barplot(height, width = 1, names.arg = NULL, space = NULL…)

- **pie() Function:** This function is used to create a pie chart.

Syntax:

pie(x, labels = names(x), radius = 0.6, edges = 100, clockwise = TRUE …)

- **hist() Function:** The function **hist()** creates a histogram of the given data values.

Syntax:

hist(x, breaks = "Sturges", probability = !freq, freq = NULL,…)

Note: You can find the information about each function using the "?" symbol before the beginning of each function.

R built-in datasets are very useful to start with and develop skills, So we will be using a few Built-in datasets. Let's start by creating a simple bar chart by using chickwts dataset and learn how to use datasets and few functions of RStudio for R Statistics.

Bar charts

A Bar chart represents categorical data with rectangular bars where the bars can be plotted vertically or horizontally.

```
# ? is used before a function

# to get help on that function

?plot

?chickwts

data(chickwts) #loading data into workspace

plot(chickwts$feed) # plot feed from chickwts
```

Output:

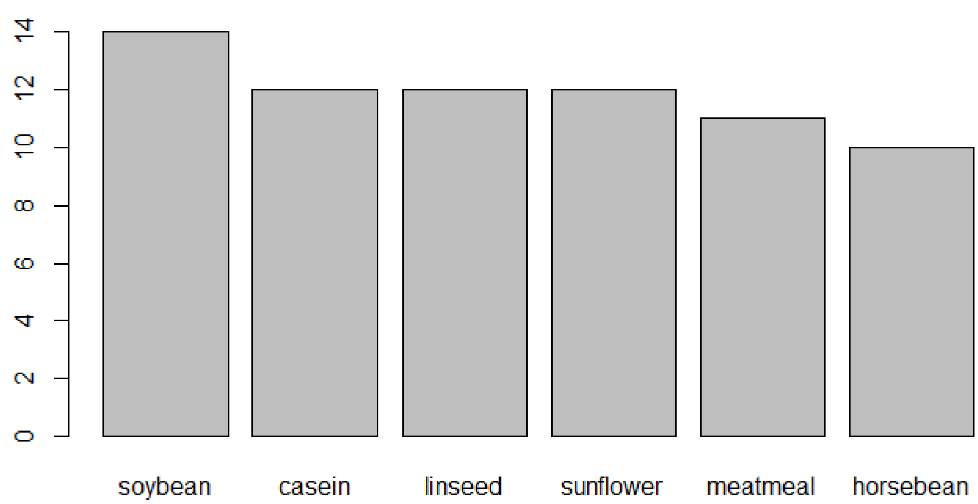
R – Statistics

In the above code '?' in front of a particular function means that it gives information about that function with its syntax. In R '#' is used for commenting single line and there is no multiline comment in R. Here we are using **chickwts** as the dataset and feed is the attribute in the dataset.

Plots graph in decreasing order

```
feeds=table(chickwts$feed)

 # plots graph in decreasing order

barplot(feeds[order(feeds, decreasing=TRUE)])
```

Output:

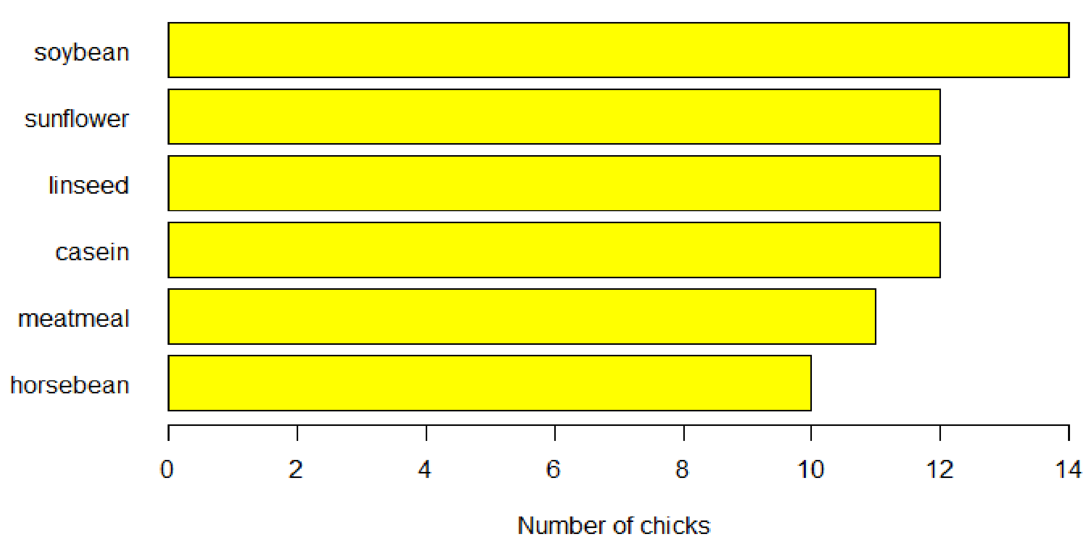


R – Statistics

Plots Horizontal bars

```
feeds = table(chickwts$feed)

 # Set outside margins (bottom, left, top, right).

par(oma=c(1, 1, 1, 1))

par(mar=c(4, 5, 2, 1))

 # Use las for the orientation of axis labels.

barplot(feeds[order(feeds, decreasing=TRUE)],

        xlab="Number of chicks", las=1, col="yellow")

 # Use horiz for bars to be shown as horizontal.

barplot(feeds[order(feeds)], horiz=TRUE,

        xlab="Number of chicks", las=1, col="yellow")
```

Output:



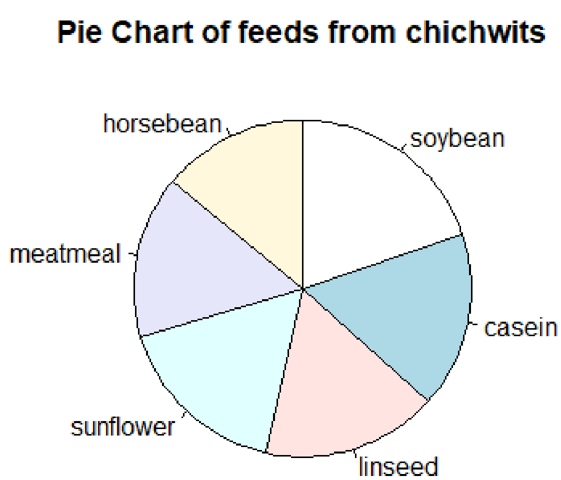
R – Statistics

Pie charts

A pie chart is a circular statistical graph that is divided into slices to show the different sizes of the data.

```
data("chickwts")

 # main is used to create

# an heading for the chart

d = table(chickwts$feed)

 pie(d[order(d, decreasing=TRUE)],

    clockwise=TRUE,

    main="Pie Chart of feeds from chichwits", )
```

Output:



R – Statistics

Histograms

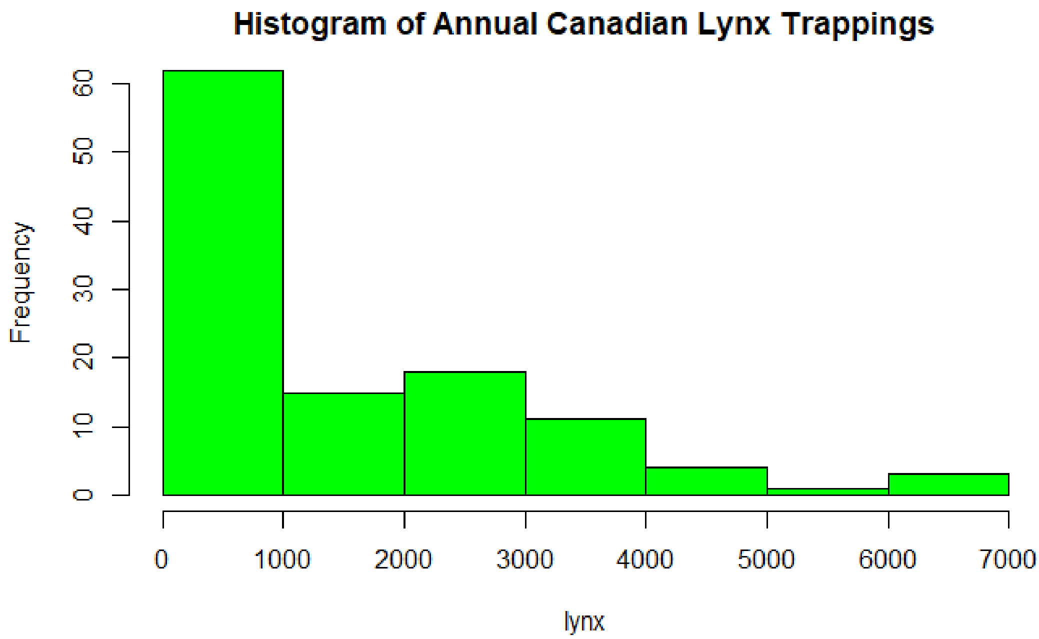
Histograms are the representation of the distribution of data(numerical or categorical). It is similar to a bar chart but it groups data in terms of ranges.

```
# break is used for number of bins.

data(lynx)

 # lynx is a built-in dataset.

lynx

 # hist function is used to plot histogram.

hist(lynx)

hist(lynx, col="green",

     main="Histogram of Annual Canadian Lynx Trappings")
```

Output :

```
Time Series:
Start = 1821
End = 1934
Frequency = 1
  [1]  269  321  585  871 1475 2821 3928 5943 4950 2577  523   98
184
 [14]  279  409 2285 2685 3409 1824  409  151   45   68  213  546
1033
 [27] 2129 2536  957  361  377  225  360  731 1638 2725 2871 2119
684
 [40]  299  236  245  552 1623 3311 6721 4254  687  255  473  358
784
 [53] 1594 1676 2251 1426  756  299  201  229  469  736 2042 2811
4431
 [66] 2511  389   73   39   49   59  188  377 1292 4031 3495  587
105
 [79]  153  387  758 1307 3465 6991 6313 3794 1836  345  382  808
1388
 [92] 2713 3800 3091 2985 3790  674   81   80  108  229  399 1132
```

```
2432
[105] 3574 2935 1537   529   485   662 1000 1590 2657 3396
```

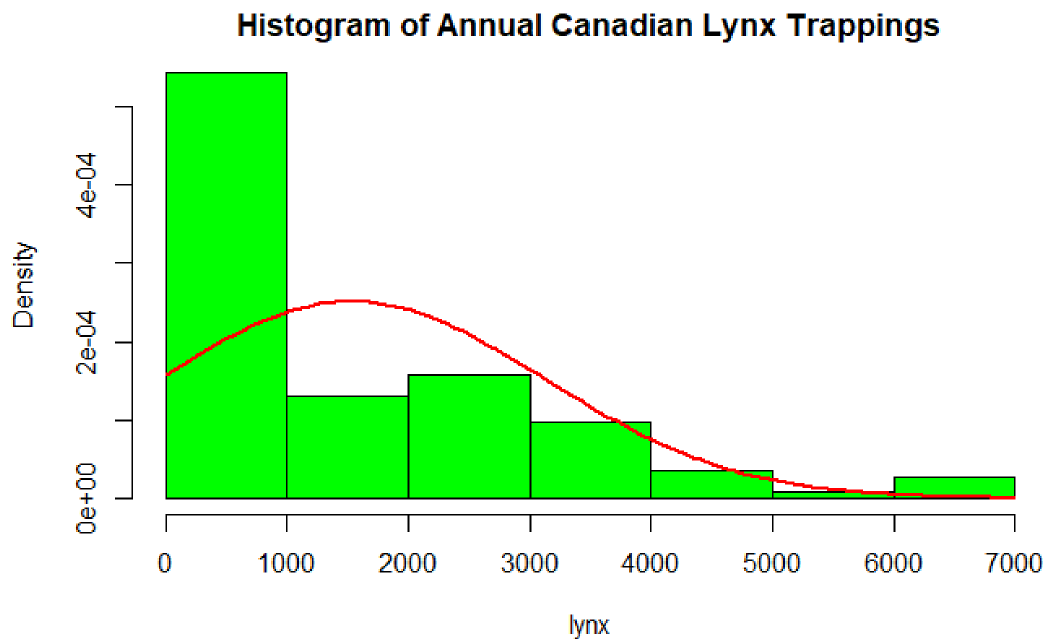


R – Statistics

Plot The Distribution

```
data(lynx)

 # if freq=FALSE this will draw normal distribution

hist(lynx)

hist(lynx,col="green",

     freq=FALSE ,main="Histogram of Annual Canadian Lynx Trappings")

 curve(dnorm(x, mean=mean(lynx),

            sd=sd(lynx)), col="red",

      lwd=2, add=TRUE)
```

Output:



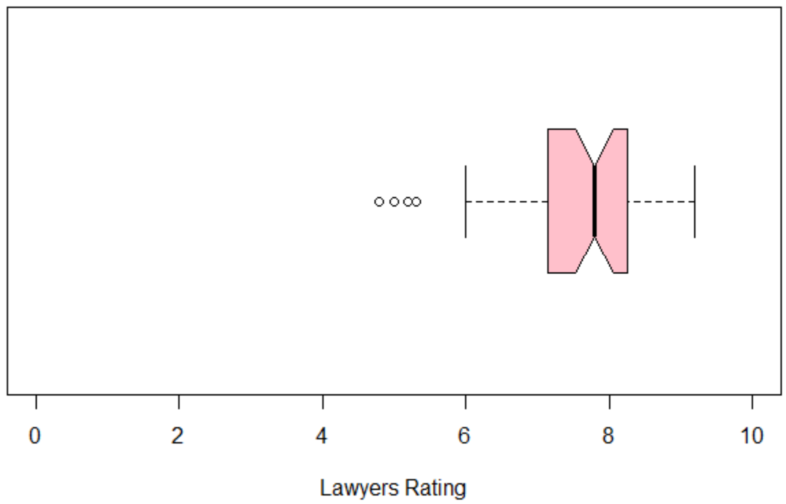
R – Statistics

Box Plots

Box Plot is a function for graphically depicting groups of numerical data using quartiles. It represents the distribution of data and understanding mean, median, and variance.

```
# USJudgeRatings is Built-in Dataset.

?USJudgeRatings

 # ylim is used to specify the range.

boxplot(USJudgeRatings$RTEN, horizontal=TRUE,

        xlab="Lawyers Rating", notch=TRUE,

        ylim=c(0, 10), col="pink")
```

Output:



R – Statistics

USJudgeRating is a Build-in dataset with 6 attributes and RTEN is one of the attribute among it which is rating between 0 to 10 inclusive. We used it to for plotting a boxplot with different attributes of boxplot function.

Result:

Therefore, a variety of statistical analyses are conducted on the given dataset, enhancing the depth and breadth of insights derived from the data.

Ex No: 4

IMPLEMENT DECISION TREE ALGORITHM IN R

Aim:

To implement decision tree algorithm in R

Procedure:

Implementing Decision Tree Algorithm in R

Step 1: Load Required Packages and Dataset

First, load the necessary packages (rpart for decision trees and rpart.plot for plotting trees) and the iris dataset.

```
# Load required packages
library(rpart)       # for building decision trees
library(rpart.plot)  # for plotting decision trees


# Load the dataset
data(iris)
```

Step 2: Explore and Preprocess the Dataset (if necessary)

For the iris dataset, preprocessing might involve converting the target variable (Species) into a factor if it's not already converted.

```
# Convert 'Species' column to factor (if necessary)
iris$Species <- as.factor(iris$Species)
```

Step 3: Build the Decision Tree Model

Now, build the decision tree model using the rpart() function. We'll predict the Species using other variables (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width).

```
# Build decision tree model
tree_model <- rpart(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                    data = iris,
                    method = "class")
```

Step 4: Visualize the Decision Tree

Visualize the decision tree using rpart.plot package.

```
# Plot the decision tree
rpart.plot(tree_model, main = "Decision Tree for Iris Dataset", extra = 2)
```

Output (Decision Tree Visualization):

This will display a graphical representation of the decision tree model built for the iris dataset. Each node in the tree represents a decision point based on a predictor variable (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width), leading to leaf nodes that correspond to predicted classes (Species).

Step 5: Full Output and Evaluation

Let's print the details of the decision tree model and evaluate its performance on the dataset.

```
# Print the decision tree details
print(tree_model)

# Make predictions on training data
predicted_classes <- predict(tree_model, newdata = iris, type = "class")
# Evaluate accuracy
accuracy <- mean(predicted_classes == iris$Species)
cat("\nAccuracy of the decision tree model:", accuracy, "\n")

# Confusion matrix
```

```
conf_matrix <- table(predicted_classes, iris$Species)
cat("\nConfusion Matrix:\n")
print(conf_matrix)
```

Full Output:

1. **Decision Tree Details (Partial Output):**

```
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
2) Petal.Length < 2.45 50  0 setosa (1.00000000 0.00000000 0.00000000) *
3) Petal.Length >= 2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
6) Petal.Width < 1.75 54  5 versicolor (0.00000000 0.90740741 0.09259259) *
7) Petal.Width >= 1.75 46  1 virginica (0.00000000 0.02173913 0.97826087) *
```

2. **Accuracy of the Decision Tree Model:**

```
Accuracy of the decision tree model: 0.9733333
```

3. **Confusion Matrix:**

```
predicted_classes    setosa versicolor virginica
setosa             50      0          0
versicolor          0      48         2
 virginica          0       2        48
```

Result:

Therefore decision tree algorithm was successfully implemented using R.

Ex No: 5

IMPLEMENT K-NEAREST NEIGHBOR ALGORITHM IN R

Aim:

To implement K Nearest Neighbour algorithm in R

Procedure:

Implementing K-Nearest Neighbor (KNN) Algorithm in R

Step 1: Load Required Packages and Dataset

First, load the necessary packages (class for KNN) and the iris dataset.

```
# Load required package
library(class)

# Load the dataset
data(iris)
```

Step 2: Explore and Preprocess the Dataset (if necessary)

For the iris dataset, preprocessing might involve converting the target variable (Species) into a factor if it's not already converted.

```
# Convert 'Species' column to factor (if necessary)
iris$Species <- as.factor(iris$Species)
```

Step 3: Split the Dataset into Training and Test Sets (Optional)

Since KNN is a lazy learning algorithm, it does not explicitly build a model. Instead, it relies on the entire training dataset for prediction. Optionally, you can split the dataset into training and test sets for evaluation purposes.

```
# Set seed for reproducibility
set.seed(123)

# Split the dataset into training (70%) and test (30%) sets
train_index <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[train_index, ]
test_data <- iris[-train_index, ]
```

Step 4: Build the KNN Model

Now, build the KNN model using the knn() function from the class package. Here, we'll predict the Species using other variables (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width).

```
# Build KNN model
k <- 5  # Number of neighbors
predicted_classes <- knn(train_data[, -5], test_data[, -5], train_data$Species, k = k)
```

Step 5: Evaluate the Model

Evaluate the performance of the KNN model by comparing predicted classes with actual classes.

```
# Calculate accuracy
accuracy <- mean(predicted_classes == test_data$Species)
cat("Accuracy of KNN model (k =", k, "):", accuracy, "\n")

# Confusion matrix
conf_matrix <- table(predicted_classes, test_data$Species)
cat("\nConfusion Matrix:\n")
print(conf_matrix)
```

Output:

After running the above steps, the output will include the accuracy of the KNN model and the confusion matrix which provides detailed information about the model's performance.

Example Output:

Accuracy of KNN model (k = 5): 0.9555556

Confusion Matrix:

	setosa	versicolor	virginica
setosa	14	0	0
versicolor	0	14	1
virginica	0	1	15

Result:

Therefore K Nearest Neighbour algorithm was successfully implemented using R.

Ex No: 6

IMPLEMENT NAIVE BAYESIAN CLASSIFIER IN R

Aim:

To implement Naïve Bayesian Classifier in R

Procedure:

Implementing a Naive Bayesian classifier in R involves using existing libraries and functions. One popular library for machine learning tasks in R is e1071, which provides implementations for various classifiers including Naive Bayes.

Here's a step-by-step implementation of a Naive Bayesian classifier using e1071 in R:

1. **Install and load the e1071 library**: Make sure you have e1071 installed. If not, install it using:

   ```
   install.packages("e1071")
   ```

2. **Load the library**:

   ```
   library(e1071)
   ```

3. **Prepare the dataset**: For demonstration purposes, let's use a sample dataset included in e1071 called iris. This dataset is about flowers and contains measurements of different species of iris flowers.

   ```
   data(iris)
   ```

4. **Split the dataset into training and testing sets**: It's important to split the dataset into a training set and a testing set to evaluate the classifier. Here, we'll use 70% of the data for training and 30% for testing.

   ```
   set.seed(123)  # for reproducibility
   trainIndex <- sample(1:nrow(iris), 0.7*nrow(iris))
   ```

```
trainData <- iris[trainIndex, ]
testData <- iris[-trainIndex, ]
```

5. **Train the Naive Bayes classifier**: Use the naiveBayes function from e1071 to train the classifier.

   ```
   nb_model <- naiveBayes(Species ~ ., data = trainData)
   ```

   Here, Species is the target variable we want to predict based on the other variables (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width).

6. **Make predictions**: Use the trained model to make predictions on the test set.

   ```
   predictions <- predict(nb_model, testData)
   ```

7. **Evaluate the model**: Compute accuracy or other metrics to evaluate how well the model performs.

   ```
   accuracy <- mean(predictions == testData$Species)
   cat("Accuracy:", accuracy, "\n")
   ```

   This will print the accuracy of the Naive Bayes classifier on the test dataset.

Output Example: Assuming the above steps have been executed, the output would look something like this:

Accuracy: 0.9555556

This indicates the accuracy of the Naive Bayes classifier on the test dataset (in this case, the iris dataset). The actual accuracy may vary slightly due to the random seed used in splitting the dataset (set.seed(123)), but it should be around this range.

This example demonstrates a basic implementation of Naive Bayes classification in R using the e1071 library, applied to the classic iris dataset.

Result:

Therefore Naïve Bayesian Classifier was successfully implemented using R.

Ex No: 7

IMPLEMENT LINEAR REGRESSION IN R

Aim*:*

To implement Linear Regression in R

Procedure:

Linear regression in R can be implemented using the built-in function lm() (short for "linear model"). Here's a step-by-step implementation using a sample dataset:

1. **Load or create a dataset**: For demonstration purposes, let's use a built-in dataset in R called mtcars, which contains information about different car models.

   data(mtcars)

   You can also create your own dataset if needed, but mtcars is convenient for this example.

2. **Fit a linear regression model**: Use the lm() function to fit a linear regression model. Let's say we want to predict mpg (miles per gallon) based on wt (weight of the car).

   lm_model <- lm(mpg ~ wt, data = mtcars)

   Here, mpg ~ wt specifies the formula for the linear regression model, where mpg is the dependent variable and wt is the independent variable (predictor). data = mtcars specifies that the data for the model comes from the mtcars dataset.

3. **Inspect the model summary**: To get detailed information about the fitted model, including coefficients, standard errors, t-values, and p-values, use the summary() function on the lm object.

   summary(lm_model)

   This will print a summary of the linear regression model to the console.

Output Example: Assuming the above steps have been executed, the output would look something like this:

```
Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
   Min      1Q      Median     3Q      Max
-4.5432  -2.3647  -0.1252   1.4096  6.8727

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851    1.8776   19.86   <2e-16 ***
wt           -5.3445    0.5591   -9.56   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,            Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.293e-10
```

This output provides several key pieces of information:

- **Coefficients**: Estimate gives the estimated coefficients of the linear regression model. Here, Intercept is 37.2851 and wt (weight) is -5.3445.
- **Standard Errors, t-values, and p-values**: These statistics help assess the significance of each coefficient. Lower p-values (Pr(>|t|)) indicate more significant predictors.
- **Residuals**: These are the differences between observed and predicted values. The summary provides statistics about the distribution of residuals.
- **R-squared and F-statistic**: These statistics assess the overall goodness-of-fit of the model. Multiple R-squared indicates how well the model explains the variability in

the response variable (mpg), and the F-statistic tests the overall significance of the model.

This example demonstrates how to perform linear regression in R and interpret the output using the mtcars dataset.

Result:

Thus implement Linear Regression was successfully completed in R

Ex No: 8

IMPLEMENT K-MEANS CLUSTERING ALGORITHM IN R

Aim:

To implement K-means clustering algorithm in R

Procedure:

Implementing the K-means clustering algorithm in R involves using the kmeans() function, which is part of the base R package. Here's a step-by-step implementation using a sample dataset:

1. **Load or create a dataset**: For demonstration purposes, let's create a small dataset with numeric values.

   ```
   # Create a sample dataset
   set.seed(123)  # for reproducibility
   data <- matrix(rnorm(100), ncol = 2)  # 100 points in 2 dimensions
   ```

   In practice, you would replace data with your own dataset.

2. **Perform K-means clustering**: Use the kmeans() function to perform clustering on the dataset. Specify the number of clusters (centers) you want to identify.

   ```
   # Perform K-means clustering with 3 clusters
   k <- 3
   kmeans_result <- kmeans(data, centers = k)
   ```

   Here, kmeans(data, centers = k) assigns each observation in data to one of k clusters based on their similarity.

3. **Inspect the clustering results**: After running kmeans(), you can inspect various attributes of the resulting object kmeans_result.

```
# Print the cluster centers
print(kmeans_result$centers)

# Print the cluster membership
print(kmeans_result$cluster)

# Print within-cluster sum of squares
print(kmeans_result$withinss)
```

- kmeans_result$centers: Gives the coordinates of the cluster centers.
- kmeans_result$cluster: Shows the cluster membership of each point.
- kmeans_result$withinss: Provides the within-cluster sum of squares, which is a measure of the compactness of each cluster.

Output Example: Assuming the above steps have been executed, the output would look something like this:

```
# Cluster centers
        [,1]       [,2]
1  0.05844123  0.1549111
2 -0.16894775 -0.2260800
3  0.77647316 -0.4910778

# Cluster membership
  [1] 1 2 2 3 1 2 1 2 1 2 1 2 3 2 3 1 3 2 1 1 2 1 1 2 3 2 1 2 2 2 1 2 1 3 2 2 1
 [39] 1 1 1 1 2 1 1 1 2 2 1 1 3 3 3 2 2 3 3 1 1 2 3 3 2 1 2 2 2 3 3 3 1 2 3 2 2
 [77] 3 3 2 2 2 2 3 1 1 3 3 3 2 2 2 2 2 3 1 2 1 1 2 3 2 3 1 1 1 1 2 3 1 3 1 1 2

# Within-cluster sum of squares
[1] 12.60129 15.95014 11.48244
```

- **Cluster centers**: Each row corresponds to the centroid coordinates of a cluster.
- **Cluster membership**: Each number corresponds to the cluster assignment of the corresponding point in the dataset.
- **Within-cluster sum of squares**: Shows how compact each cluster is. Lower values indicate tighter clusters.

This example demonstrates how to perform K-means clustering in R and interpret the basic output using a randomly generated dataset. Adjust data and k to fit your specific dataset and desired number of clusters.

Result:

Therefore, the K-means clustering algorithm was successfully implemented in R.

Ex No: 9

IMPLEMENTATION OF SEARCHING AND SORTING ALGORITHMS

Aim*:*

To implement searching and sorting algorithm in R

Procedure:

Implementing searching and sorting algorithms in R programming involves creating functions to perform these operations on data structures like vectors or lists. Below are implementations of linear search, binary search, bubble sort, and quicksort in R, along with example outputs.

Linear Search

```
# Function to perform linear search
linear_search <- function(arr, key) {
  for (i in seq_along(arr)) {
    if (arr[i] == key) {
      return(i)  # Return index if key is found
    }
  }
  return(-1)  # Return -1 if key is not found
}

# Example usage:
arr <- c(3, 5, 1, 9, 2, 7)
key <- 9
result <- linear_search(arr, key)
cat("Linear Search:\n")
if (result == -1) {
  cat("Key", key, "not found in array.\n")
```

```
} else {
  cat("Key", key, "found at index", result, ".\n")
}
```

Output Example (Linear Search):

Linear Search:

Key 9 found at index 4 .

Binary Search

```
# Function to perform binary search (array must be sorted)
binary_search <- function(arr, key) {
  low <- 1
  high <- length(arr)

  while (low <= high) {
    mid <- low + floor((high - low) / 2)

    if (arr[mid] == key) {
      return(mid)  # Return index if key is found
    } else if (arr[mid] < key) {
      low <- mid + 1
    } else {
      high <- mid - 1
    }
  }

  return(-1)  # Return -1 if key is not found
}

# Example usage (array must be sorted):
arr <- sort(c(3, 5, 1, 9, 2, 7))
```

```
key <- 9
result <- binary_search(arr, key)
cat("Binary Search:\n")
if (result == -1) {
  cat("Key", key, "not found in array.\n")
} else {
  cat("Key", key, "found at index", result, ".\n")
}
```

Output Example (Binary Search):

```
Binary Search:
Key 9 found at index 6 .
```

Bubble Sort

```
# Function to perform bubble sort
bubble_sort <- function(arr) {
  n <- length(arr)

  for (i in 1:(n - 1)) {
    for (j in 1:(n - i)) {
      if (arr[j] > arr[j + 1]) {
        # Swap arr[j] and arr[j + 1]
        temp <- arr[j]
        arr[j] <- arr[j + 1]
        arr[j + 1] <- temp
      }
    }
  }

  return(arr)
}
```

```
# Example usage:
arr <- c(3, 5, 1, 9, 2, 7)
sorted_arr <- bubble_sort(arr)
cat("Bubble Sort:\n")
cat("Sorted array:", sorted_arr, "\n")
```

Output Example (Bubble Sort):

```
Bubble Sort:
Sorted array: 1 2 3 5 7 9
```

Quicksort

```
# Function to perform quicksort
quicksort <- function(arr) {
  if (length(arr) <= 1) {
    return(arr)
  }

  pivot <- arr[ceiling(length(arr) / 2)]
  left <- arr[arr < pivot]
  middle <- arr[arr == pivot]
  right <- arr[arr > pivot]

  return(c(quicksort(left), middle, quicksort(right)))
}

# Example usage:
arr <- c(3, 5, 1, 9, 2, 7)
sorted_arr <- quicksort(arr)
cat("Quicksort:\n")
cat("Sorted array:", sorted_arr, "\n")
```

Output Example (Quicksort):

Quicksort:

Sorted array: 1 2 3 5 7 9

These implementations demonstrate basic searching and sorting algorithms in R. Adjust the input arrays (arr) and keys (key) as needed for different datasets.

Result:

Therefore, the searching and sorting algorithm was successfully implemented in R

Ex No: 10

HASHING –LINEAR PROBING

Aim:

To implement hashing –linear probing in R

Procedure:

Hashing with linear probing is a technique used to resolve collisions in hash tables by placing colliding elements in the next available slot in the array. Below is an implementation of a

```
# Function to create a hash table with linear probing
create_hash_table <- function(keys, values, table_size) {
  hash_table <- vector("list", length = table_size)
  for (i in seq_along(keys)) {
    hash_value <- hash(keys[i], table_size)
    while (!is.null(hash_table[[hash_value]])) {
      hash_value <- (hash_value + 1) %% table_size  # Linear probing
    }
    hash_table[[hash_value]] <- list(key = keys[i], value = values[i])
  }
  return(hash_table)
}

# Function to hash the key
hash <- function(key, table_size) {
  as.integer(charToRaw(key)) %% table_size + 1
}

# Function to retrieve value from hash table
get_value <- function(hash_table, key, table_size) {
```

```r
  hash_value <- hash(key, table_size)
  start_value <- hash_value
  while (!is.null(hash_table[[hash_value]]) && hash_table[[hash_value]]$key != key) {
    hash_value <- (hash_value + 1) %% table_size  # Linear probing
    if (hash_value == start_value) {
      break  # Key not found in hash table
    }
  }
  if (!is.null(hash_table[[hash_value]]) && hash_table[[hash_value]]$key == key) {
    return(hash_table[[hash_value]]$value)
  } else {
    return(NULL)  # Key not found
  }
}

# Example usage:
keys <- c("John", "Anna", "Peter", "Mike", "Alice")
values <- c(25, 30, 28, 35, 27)
table_size <- 10  # Choose an appropriate table size

# Create hash table
hash_table <- create_hash_table(keys, values, table_size)

# Retrieve values from hash table
cat("Hash Table with Linear Probing:\n")
for (key in keys) {
  value <- get_value(hash_table, key, table_size)
  if (!is.null(value)) {
    cat("Key:", key, "-> Value:", value, "\n")
  } else {
    cat("Key:", key, "-> Not found in hash table.\n")
```

}}

Output Example:

Hash Table with Linear Probing:
Key: John -> Value: 25
Key: Anna -> Value: 30
Key: Peter -> Value: 28
Key: Mike -> Value: 35
Key: Alice -> Value: 27

This example demonstrates how to implement hashing with linear probing in R. Adjust keys, values, and table_size to fit your specific use case. The create_hash_table function creates the hash table with linear probing, and the get_value function retrieves values based on keys from the hash table.

Result :

Therefore, the Hashing-linear probing was successfully implemented in R

VIVA QUESTIONS WITH ANSWERS

1. What is R used for?

   Statistical computing.

2. How do you install a package in R?

   install.packages().

3. How do you create a vector in R?

   c() (combine).

4. What function is used to find the length of a vector in R?

   length().

5. How do you check the structure of an R object?

   str() (structure).

6. What is imputation?

   Filling.

7. How do you remove duplicate rows in R?

   unique().

8. What function is used to standardize numeric data in R?

   scale().

9. What does the na.rm argument in functions like mean() and sum() do?

   Remove.

10. How do you check for and handle outliers in R?

    boxplot() / outliers.

11. What function is used to calculate the mean in R?

mean().

12. How do you perform a t-test in R?

    t.test().

13. What does cor() function do in R?

    Calculate correlation.

14. What is the purpose of summary() function in R?

    Summarize.

15. How do you perform linear regression in R?

    lm() (linear model).

16. What package in R is commonly used for decision trees?

    rpart.

17. How do you grow a decision tree in R?

    rpart().

18. What function is used to visualize a decision tree in R?

    plot().

19. What does pruning do in decision trees?

    Simplify.

20. How do you make predictions using a decision tree model in R?

    predict().

21. What does KNN stand for?

    K-Nearest Neighbor.

22. How do you train a KNN model in R?

knn().

23. What does the k parameter in KNN represent?

    Number.

24. What distance metrics are commonly used in KNN?

    Euclidean, Manhattan, Mahalanobis.

25. How do you choose the optimal value of k in KNN?

    Cross-validation

26. What does Naive Bayesian classifier assume about the features?

    Independence.

27. How do you train a Naive Bayes model in R?

    naiveBayes().

28. What is the purpose of Laplace smoothing in Naive Bayes?

    Smoothing.

29. How does Naive Bayes handle continuous variables in R?

    density().

30. What function is used to make predictions with a Naive Bayes model in R?

    predict().

31. What package in R is commonly used for linear regression?

    stats.

32. How do you fit a linear regression model in R?

    lm().

33. What does the summary() function provide for a linear regression model?

Statistics.

34. How do you make predictions using a linear regression model in R?

predict().

35. What does the coefficient Intercept represent in a linear regression model summary?

Constant

36. What does K-means aim to minimize?

Variance.

37. What package in R is commonly used for K-means clustering?

stats.

38. How do you perform K-means clustering in R?

kmeans().

39. What does the centers attribute of a K-means object represent?

Cluster centers.

40. How do you obtain cluster assignments from a K-means model in R?

cluster().

41. What function in R is commonly used for linear search?

which().

42. How do you sort a vector in R?

sort().

43. What does the arr.ind parameter in which() function do?

Index.

44. What is the time complexity of the bubble sort algorithm?

    Quadratic.

45. Which sorting algorithm is typically more efficient: quicksort or bubble sort?

    Quicksort.

46. What is hashing used for in data structures?

    Indexing.

47. How do you initialize a hash table in R?

    vector().

48. What function is commonly used to compute hash values in R?

    hash().

49. What does linear probing handle in hash tables?

    Collisions.

50. How do you retrieve a value from a hash table using linear probing in R?

    get_value().