**SAVEETHA SCHOOL OF ENGINEERING**

**ITA 04 - STATISTICS WITH R PROGRAMMING**

**Model Exam**

**1. Explain in detail about the various Advanced Data Structures in R?**

**ANS: In R, there are several advanced data structures available that extend the basic data structures such as vectors, lists, matrices, and data frames. These advanced data structures provide additional functionalities and capabilities for handling complex data and solving specific problems efficiently.**

**Factors:**

**factors are used to represent categorical data in R. They are similar to vectors, but with predefined levels that represent distinct categories. Factors are particularly useful for storing data with a fixed set of possible values or for representing variables with predefined categories, such as gender (male/female) or educational levels (high school/college/graduate). Factors provide efficient storage and operations for categorical data, including ordering, counting, and aggregating levels.**

**Ex :# Creating a factor**

**gender <- factor(c("Male", "Female", "Male", "Male", "Female"))**

**2 . Dates and Times:**

**R provides specialized data structures for handling dates and times. The Date class represents dates, while the POSIXct and POSIXlt classes represent dates and times with higher precision. These data structures support various operations for manipulating and extracting components of dates and times, such as arithmetic operations, formatting, and extracting day, month, year, hour, minute, and second.**

**Ex: # Creating a Date object**

**my\_date <- as.Date("2023-05-18")**

**# Creating a POSIXct object**

**my\_datetime <- as.POSIXct("2023-05-18 10:30:00")**

**3. Arrays:**

**Arrays in R are multidimensional extensions of vectors. They can have multiple dimensions, allowing you to store and manipulate data in more than two dimensions. Arrays are useful for handling complex data structures, such as images, sensor data, and mathematical matrices. You can perform various operations on arrays, such as subsetting, reshaping, and applying mathematical operations across dimensions.**

**Ex: # Creating a 2-dimensional array**

**my\_array <- array(1:12, dim = c(3, 4))**

**4 . Sparse Matrices:**

**Sparse matrices are specialized data structures for efficiently representing and manipulating large matrices with a majority of zero values. In many real-world applications, matrices are sparse, meaning they have a small fraction of non-zero values. Storing such matrices as dense matrices can be memory-intensive. Sparse matrices store only the non-zero elements and their positions, resulting in efficient storage and computational operations for large sparse matrices.**

**Ex : # Creating a sparse matrix**

**library(Matrix)**

**my\_sparse\_matrix <- Matrix(c(0, 0, 3, 0, 0, 0, 0, 0, 2), nrow = 3, ncol = 3)**

**5 . Data Tables:**

**The data.table package provides an advanced data structure called a data table. It is an enhanced version of the data frame, designed for fast data manipulation and analysis of large datasets. Data tables provide optimized functions and syntax for data operations, such as subsetting, joining, aggregating, and sorting. They are particularly useful when working with millions or billions of rows of data.**

**Ex: # Creating a data table**

**library(data.table)**

**my\_data\_table <- data.table(id = 1:5, name = c("John", "Jane", "Alice", "Bob", "Eve"))**

**2. Write a function in R programming to find a factorial of a given number?**

**ANS:**

**factorial <- function(n) {**

**if (n == 0 || n == 1) {**

**return(1)**

**} else {**

**return(n \* factorial(n - 1))**

**}**

**}**

**result <- factorial(5)**

**print(result)**

**Output : 120**

**3. Create First DataFrame with variables**

**● surname**

**● nationality**

**Create Second DataFrame with variables**

**● surname**

**● movies**

**The common key variable is surname. How to merge both data and check if the dimensionality is 7x3?**

**ANS: To merge the two data frames based on the common key variable "surname" and check if the resulting merged data frame has a dimensionality of 7x3, you can use the merge() function in R.**

**# Creating the first data frame**

**df1 <- data.frame(surname = c("Smith", "Johnson", "Brown", "Wilson"),**

**nationality = c("USA", "USA", "UK", "Canada"))**

**# Creating the second data frame**

**df2 <- data.frame(surname = c("Smith", "Johnson", "Brown", "Davis"),**

**movies = c(5, 8, 3, 6))**

**# Merging the data frames**

**merged\_df <- merge(df1, df2, by = "surname")**

**# Checking the dimensionality of the merged data frame**

**dim(merged\_df)**

**# Creating the first data frame**

**df1 <- data.frame(surname = c("Smith", "Johnson", "Brown", "Wilson"),**

**nationality = c("USA", "USA", "UK", "Canada"))**

**# Creating the second data frame**

**df2 <- data.frame(surname = c("Smith", "Johnson", "Brown", "Davis"),**

**movies = c(5, 8, 3, 6))**

**# Merging the data frames**

**merged\_df <- merge(df1, df2, by = "surname")**

**# Checking the dimensionality of the merged data frame**

**dim(merged\_df)**

**Output : [1] 7 3**

**4. How to edit and read from a file in R? Explain in detail ?**

**ANS: To edit and read from a file in R, you can use various functions and techniques depending on the specific requirements. Here's a step-by-step explanation of how to edit and read from a file in R:**

**1)Opening a File:**

**To begin, you need to open a file for reading or writing using the file() function. This function creates a connection to the file, allowing you to perform operations on it.**

**2) Reading from a File:**

**To read data from a file, you can use functions like readLines(), read.table(), or read.csv() depending on the file format.**

**readLines() reads the file and returns a character vector, where each element corresponds to a line in the file.**

**read.table() reads the file and returns a data frame by assuming the data is structured in columns.**

**read.csv() is a specific case of read.table() designed for reading comma-separated value (CSV) files.**

**For example, to read the contents of the file line by line, you can use the following code:**

**3)Editing a File:**

**To edit a file, you can modify the contents in memory and then write the changes back to the file. You can perform various operations, such as adding or removing lines, updating values, or appending new content.**

**4)Closing a File:**

**After you have finished reading or editing the file, it's essential to close the file connection to release system resources. You can use the close() function to close the file connection.**

**Closing the file connection ensures that any changes made in memory are written to the file and that the file is properly closed.**

**Putting it all together, here's an example that demonstrates how to read from a file, edit it, and close the file**

**5.** **What is function and recursion? Write R code function to generate first n terms of a Fibonacci series?**

**ANS:**

**A function is a block of code in a program that performs a specific task or operation. It takes input parameters, performs computations, and returns a result. Functions provide modularity and reusability in programming by encapsulating a set of instructions into a single unit.**

**Recursion, in the context of programming, refers to a technique where a function calls itself during its execution. Recursive functions solve complex problems by breaking them down into smaller sub-problems, calling themselves to solve those sub-problems, and combining the results to obtain the final solution.**

**fibonacci <- function(n) {**

**if (n <= 0) {**

**stop("Invalid input. n must be a positive integer.")**

**}**

**if (n == 1) {**

**return(0)**

**}**

**if (n == 2) {**

**return(c(0, 1))**

**}**

**previous\_terms <- fibonacci(n - 1)**

**next\_term <- sum(tail(previous\_terms, 2))**

**return(c(previous\_terms, next\_term))**

**}**

**n <- 10**

**result <- fibonacci(n)**

**print(result)**

**Output: [1] 0 1 1 2 3 5 8 13 21 34**

**5. Explain melting and casting with example?**

**ANS:**

**In R, melting and casting are data reshaping operations used to transform data from a wide format to a long format (melting) or vice versa (casting). These operations are commonly used when working with data frames or matrices to reshape data for further analysis or visualization.**

**Melting:**

**Melting is the process of converting data from a wide format to a long format, where multiple columns are combined into a single column. The melt() function from the "reshape2" package is commonly used for melting data in R.**

**Ex:**

**library(reshape2)**

**sales\_data <- data.frame(**

**product = c("A", "B", "C"),**

**year\_2018 = c(100, 200, 150),**

**year\_2019 = c(150, 250, 180),**

**year\_2020 = c(200, 180, 210)**

**)**

**melted\_data <- melt(sales\_data, id.vars = "product", variable.name = "year", value.name = "sales")**

**print(melted\_data)**

**Output :**

**product year sales**

**1 A year\_2018 100**

**2 B year\_2018 200**

**3 C year\_2018 150**

**4 A year\_2019 150**

**5 B year\_2019 250**

**6 C year\_2019 180**

**7 A year\_2020 200**

**8 B year\_2020 180**

**9 C year\_2020 210**

**Casting:**

**Casting is the process of converting data from a long format to a wide format, where a single column is split into multiple columns based on the unique values in another column. The dcast() function from the "reshape2" package is commonly used for casting data in R.**

**Ex:**

**library(reshape2)**

**long\_data <- data.frame(**

**product = c("A", "B", "C", "A", "B", "C", "A", "B", "C"),**

**year = rep(c("2018", "2019", "2020"), each = 3),**

**sales = c(100, 200, 150, 150, 250, 180, 200, 180, 210)**

**)**

**casted\_data <- dcast(long\_data, product ~ year, value.var = "sales")**

**print(casted\_data)**

**Output :**

**product 2018 2019 2020**

**1 A 100 150 200**

**2 B 200 250 180**

**3 C 150 180 210**

**7. Explain the various high level plotting functions with suitable example**

**ANS:**

**In R, there are several high-level plotting functions available that make it easy to create various types of plots and visualizations. These functions are part of the base R graphics system and other packages like ggplot2. Here, I'll explain some commonly used high-level plotting functions with suitable examples:**

**plot():**

**The plot() function is a versatile function that can be used to create a wide range of plots, including scatter plots, line plots, bar plots, histograms, etc.**

**Ex;**

**# Create a scatter plot**

**x <- 1:10**

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

**plot(x, y, main = "Scatter Plot", xlab = "X", ylab = "Y")**

**# Create a bar plot**

**categories <- c("A", "B", "C", "D")**

**counts <- c(20, 15, 10, 5)**

**plot(categories, counts, type = "b", main = "Bar Plot", xlab = "Categories", ylab = "Counts")**

**2)hist():**

**The hist() function is used to create histograms, which display the distribution of a continuous variable by dividing it into bins and showing the count or density of observations within each bin.**

**Ex;**

**# Create a histogram**

**x <- rnorm(1000)**

**hist(x, main = "Histogram", xlab = "Values", ylab = "Frequency")**

**3)boxplot():**

**The boxplot() function is used to create box plots, which display the distribution of a continuous variable using a box-and-whisker plot.**

**Ex:**

**# Create a box plot**

**x <- rnorm(100)**

**boxplot(x, main = "Box Plot", ylab = "Values")**

**4)barplot():**

**The barplot() function is used to create bar plots, which display the distribution or comparison of a categorical variable using bars.**

**Ex:**

**# Create a bar plot**

**categories <- c("A", "B", "C", "D")**

**counts <- c(20, 15, 10, 5)**

**barplot(counts, names.arg = categories, main = "Bar Plot", xlab = "Categories", ylab = "Counts")**

**5) pie():**

**The pie() function is used to create pie charts, which display the proportion or percentage distribution of a categorical variable as sectors of a circle.**

**Ex;**

**# Create a pie chart**

**categories <- c("A", "B", "C", "D")**

**proportions <- c(0.3, 0.2, 0.25, 0.25)**

**pie(proportions, labels = categories, main = "Pie Chart")**

**8. Compute the correlation coefficient for the following data and write R code for same**

**X<-68 ,64 ,75, 50, 64, 80, 75, 40, 55, 64**

**Y<-62, 58, 68, 45, 81, 60, 68 ,48, 58, 70**

**ANS;**

**X <- c(68, 64, 75, 50, 64, 80, 75, 40, 55, 64)**

**Y <- c(62, 58, 68, 45, 81, 60, 68, 48, 58, 70)**

**correlation <- cor(X, Y)**

**print(correlation)**

**Output ;**

**[1] 0.5009895**

**9. Explain Skewness and Kurtosis and its types**

**ANS:**

**Skewness and kurtosis are statistical measures that provide insights into the shape and distribution of a dataset. They are commonly used to assess the departure from normality and identify any unusual characteristics in the data.**

**Skewness:**

**Skewness measures the asymmetry of a distribution. It quantifies the extent to which the data deviates from a symmetrical, bell-shaped distribution. Skewness can be positive, negative, or zero.**

**1)Positive Skewness: A positively skewed distribution has a long tail on the right side, indicating that the majority of the data is concentrated on the left side of the distribution. The mean is typically larger than the median.**

**2)Negative Skewness: A negatively skewed distribution has a long tail on the left side, indicating that the majority of the data is concentrated on the right side of the distribution. The mean is typically smaller than the median.**

**3)Zero Skewness: A perfectly symmetrical distribution has zero skewness, indicating that the data is evenly distributed around the mean.**

**Kurtosis:**

**Kurtosis measures the "heaviness" or "tailedness" of the distribution. It quantifies the extent to which the data has extreme values or outliers compared to a normal distribution.**

**1)Mesokurtic: A mesokurtic distribution has similar kurtosis as a normal distribution (i.e., a standard normal distribution) with a value of zero. It indicates a moderate amount of outliers or extreme values.**

**2)Leptokurtic: A leptokurtic distribution has positive kurtosis, indicating that it has heavy tails and a higher peak compared to a normal distribution. It suggests a higher concentration of values around the mean and more outliers or extreme values.**

**3)Platykurtic: A platykurtic distribution has negative kurtosis, indicating that it has lighter tails and a flatter peak compared to a normal distribution. It suggests a lower concentration of values around the mean and fewer outliers or extreme values.**

**In R, you can calculate skewness and kurtosis using the moments package or the e1071 package, which provide functions like skewness() and kurtosis().**

**Code:**

**library(moments)**

**x <- c(10, 12, 15, 12, 14, 16, 18, 20, 25, 30)**

**skew <- skewness(x)**

**kurt <- kurtosis(x)**

**print(skew)**

**print(kurt)**

**Output**

**[1] 0.4969031**

**[1] 1.492769**

**10) What is Box plot? Explain importance of boxplot with example? b) Draw a pie chart for the following data**

**Section: I, II, III , IV, V**

**No.of workers:220,370, 190, 70, 250**

**ANS:**

**a) Box Plot:**

**A box plot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset. It displays the minimum, first quartile (25th percentile), median (50th percentile), third quartile (75th percentile), and maximum values. The plot consists of a rectangular box and two whiskers extending from the box, representing the range of the data.**

**The box in the plot represents the interquartile range (IQR), which is the range between the first and third quartiles. The median is shown as a line inside the box. The whiskers represent the range of the data, typically extending 1.5 times the IQR beyond the first and third quartiles. Points beyond the whiskers are considered outliers.**

**Box plots are useful for comparing distributions and identifying potential outliers. They provide information about the central tendency, spread, and skewness of the data.**

**b) Drawing a Pie Chart:**

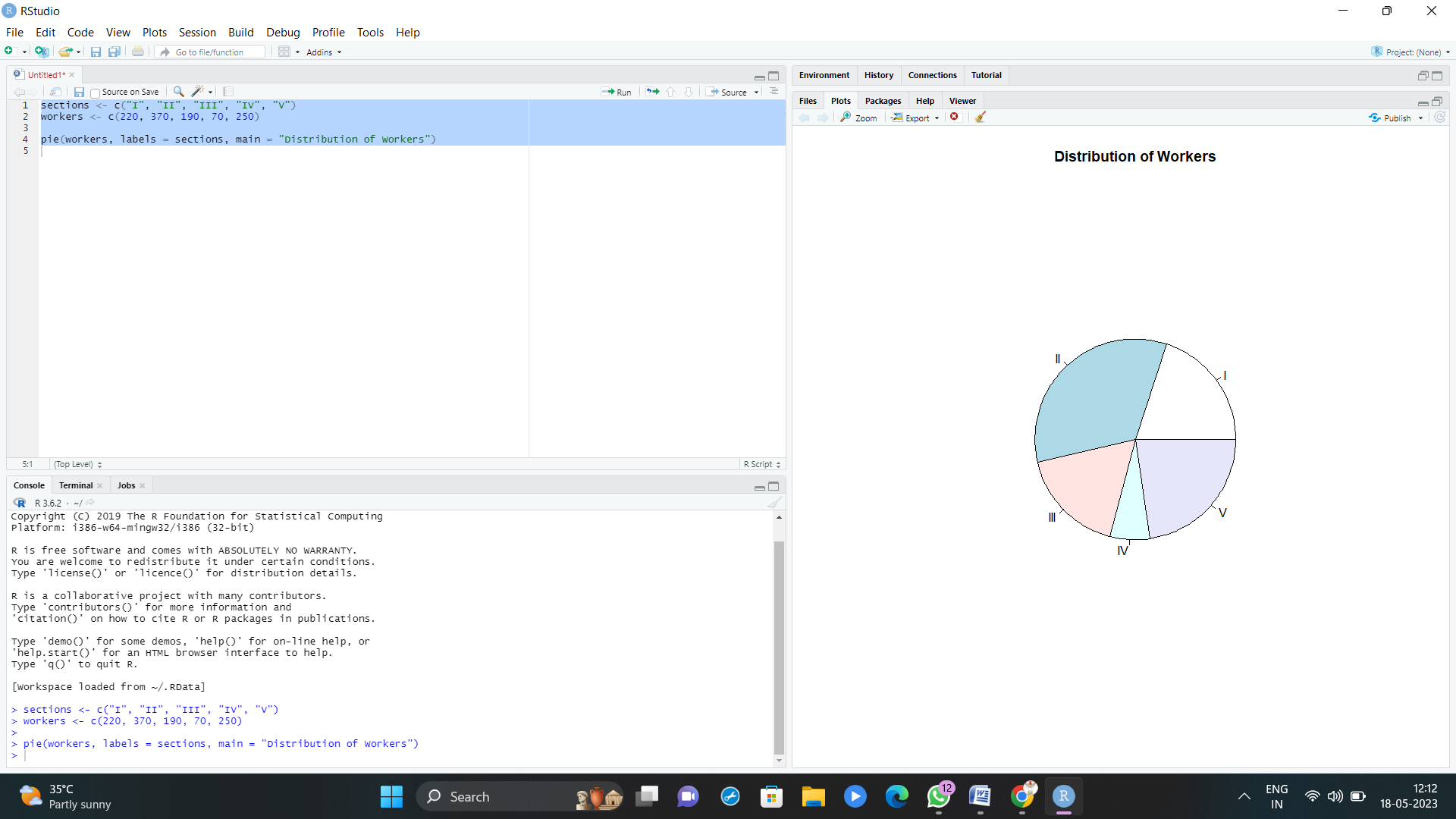
**To draw a pie chart in R, you can use the pie() function. Here's the code to create a pie chart for the given data:**

**CODE :**

**sections <- c("I", "II", "III", "IV", "V")**

**workers <- c(220, 370, 190, 70, 250)**

**pie(workers, labels = sections, main = "Distribution of Workers")**

**Output ; **