R Graphics

**Charts & Graphs**

▷ **Plot**

* The plot() function is used to draw points (markers) in a diagram.
* The function takes parameters for specifying points in the diagram.
  + Parameter 1 speciﬁes points on the x-axis.
  + Parameter 2 speciﬁes points on the y-axis.
* At its simplest, you can use the plot() function to plot two numbers against each other:
* Example - Draw one point in the diagram, at position (1) and position (3): plot(1, 3)
* To draw more points, use vectors:
* Example - Draw two points in the diagram, one at position (1, 3) and one in position (8,10):

plot(c(1, 8), c(3, 10))

▷ Multiple Points

* You can plot as many points as you like, just make sure you have the same number of points in both axis:
* Example

plot(c(1, 2, 3, 4, 5), c(3, 7, 8, 9, 12))

* + For better organization, when you have many values, it is better to use variables:
  + Example

x <- c(1, 2, 3, 4, 5)

y <- c(3, 7, 8, 9, 12)

plot(x, y)

## ▷ Sequences of Points

▷ If you want to draw dots in a sequence, on both the x-axis and the y-axis, use the : operator:

▷ Example plot(1:10)

▷ **Line Graphs**

* + A line chart is a graph that connects a series of points by drawing line segments between them.
  + These points are ordered in one of their coordinate (usually the x-coordinate) value.
  + Line charts are usually used in identifying the trends in data.
  + The plot() function in R is used to create the line graph.

## ▷ Syntax - The basic syntax to create a line chart in R is −

plot(v,type,col,xlab,ylab,main,lwd,lty)

## ▷ Following is the description of the parameters used −

* + v is a vector containing the numeric values.
  + type takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
  + xlab is the label for x axis.
  + ylab is the label for y axis.
  + main is the Title of the chart.
  + col is used to give colors to both the points and lines.

# ▷ Line Graphs

* + A line graph has a line that connects all the points in a diagram.
  + To create a line, use the plot() function and add the type parameter with a value of "l":
  + Example

plot(1:10, type="l")

# ▷ Line Color

* + The line color is black by default. To change the color, use the col parameter:
  + Example

plot(1:10, type="l", col="blue")

# ▷ Line Width

* + To change the width of the line, use the lwd parameter (1 is default, while 0.5 means 50% smaller, and 2 means 100% larger):
  + Example

plot(1:10, type="l", lwd=2)

# ▷ Line Styles

* + The line is solid by default. Use the lty parameter with a value from 0 to 6 to specify the line format.
  + For example, lty=3 will display a dotted line instead of a solid line:
  + Example

plot(1:10, type="l", lwd=5, lty=3)

## ▷ Available parameter values for lty:

* + 0 removes the line
  + 1 displays a solid line
  + 2 displays a dashed line
  + 3 displays a dotted line
  + 4 displays a "dot dashed" line
  + 5 displays a "long dashed" line
  + 6 displays a "two dashed" line

## ▷ Multiple Lines

* + More than one line can be drawn on the same chart by using the lines()function.
  + After the ﬁrst line is plotted, the lines() function can use an additional vector as input to draw the second line in the chart,
  + To display more than one line in a graph, use the plot() function together with the lines() function:
  + Example

line1 <- c(1,2,3,4,5,10)

line2 <- c(2,5,7,8,9,10)

plot(line1, type = "l", col = "blue") lines(line2, type="l", col = "red")

# ▷ R - Scatterplots

* Scatterplots show many points plotted in the Cartesian plane.
* Each point represents the values of two variables.
* One variable is chosen in the horizontal axis and another in the vertical axis.
* The simple scatterplot is created using the plot() function.
* Syntax - The basic syntax for creating scatterplot in R is −

plot(x, y, main, xlab, ylab, xlim, ylim, axes)

Following is the description of the parameters used −

* x is the data set whose values are the horizontal coordinates.
* y is the data set whose values are the vertical coordinates.
* main is the tile of the graph.
* xlab is the label in the horizontal axis.
* ylab is the label in the vertical axis.
* xlim is the limits of the values of x used for plotting.
* ylim is the limits of the values of y used for plotting.
* axes indicates whether both axes should be drawn on the plot.

▷ Example

x <- c(5,7,8,7,2,2,9,4,11,12,9,6)

y <- c(99,86,87,88,111,103,87,94,78,77,85,86)

plot(x, y, main="Observation of Cars", xlab="Car age", ylab="Car speed")

## ▷ The observation in the example above should show the result of 12 cars passing by.

▷ The x-axis shows how old the car is.

▷ The y-axis shows the speed of the car when it passes.

# ▷ Compare Plots

* + To compare the plot with another plot, use the points() function:
  + Example - Draw two plots on the same ﬁgure: # day one, the age and speed of 12 cars:

x1 <- c(5,7,8,7,2,2,9,4,11,12,9,6)

y1 <- c(99,86,87,88,111,103,87,94,78,77,85,86)

# day two, the age and speed of 15 cars:

x2 <- c(2,2,8,1,15,8,12,9,7,3,11,4,7,14,12)

y2 <- c(100,105,84,105,90,99,90,95,94,100,79,112,91,80,85)

plot(x1, y1, main="Observation of Cars", xlab="Car age", ylab="Car speed", col="red", cex=2) points(x2, y2, col="blue", cex=2)

# ▷ Pie Charts

* + A pie chart is a circular graphical view of data. In R the pie chart is created using the pie() function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.
  + Syntax - The basic syntax for creating a pie-chart using the R is − pie(x, labels, radius, main, col, clockwise)
  + Following is the description of the parameters used −
    - x is a vector containing the numeric values used in the pie chart.
    - labels is used to give description to the slices.
    - radius indicates the radius of the circle of the pie chart.(value between −1 and +1).
    - main indicates the title of the chart.
    - col indicates the color palette.
    - clockwise is a logical value indicating if the slices are drawn clockwise or anti clockwise.

▷ Example

# Create a vector of pies x <- c(10,20,30,40)

# Display the pie chart pie(x)

▷ As you can see the pie chart draws one pie for each value in the vector (in this case 10, 20, 30, 40).

▷ By default, the plotting of the ﬁrst pie starts from the x-axis and move counterclockwise.

▷ Note: The size of each pie is determined by comparing the value with all the other values, by using this formula:

▷ The value divided by the sum of all values: x/sum(x)

# ▷ Labels and Header

* + Use the label parameter to add a label to the pie chart, and use the main parameter to add a header:
  + Example

# Create a vector of pies x <- c(10,20,30,40)

# Create a vector of labels

mylabel <- c("Apples", "Bananas", "Cherries", "Dates") # Display the pie chart with labels

pie(x, label = mylabel, main = "Fruits")

▷ Colors

* + You can add a color to each pie with the col parameter:
  + Example

# Create a vector of colors

colors <- c("blue", "yellow", "green", "black") # Display the pie chart with colors

pie(x, label = mylabel, main = "Fruits", col = colors)

▷ Legend

* + To add a list of explanation for each pie, use the legend() function:
  + The legend can be positioned as either: bottomright, bottom, bottomleft, left, topleft, top, topright, right, center
  + Example

# Create a vector of pies x <- c(10,20,30,40)

# Create a vector of labels

mylabel <- c("Apples", "Bananas", "Cherries", "Dates") # Create a vector of colors

colors <- c("blue", "yellow", "green", "black") # Display the pie chart with colors

pie(x, label = mylabel, main = "Pie Chart", col = colors) # Display the explanation box

legend("bottomright", mylabel, ﬁll = colors)

R Data Interfaces -

**Importing Data**

## ▷ In R, we can read data from ﬁles stored outside the R environment.

▷ We can also write data into ﬁles which will be stored and accessed by the operating system.

## ▷ R can read and write into various ﬁle formats like csv, excel, xml etc.

▷ The ﬁle should be present in current working directory so that R can read it.

## ▷ Of course we can also set our own directory and read ﬁles from there.

▷ The csv ﬁle is a text ﬁle in which the values in the columns are separated by a comma.

## ▷ You can create this ﬁle using windows notepad by copying and pasting this data.

▷ Save the ﬁle as input.csv using the save As All ﬁles(\*.\*) option in notepad.

id,name,salary,start\_date,dept 1,Rick,623.3,2012-01-01,IT

2,Dan,515.2,2013-09-23,Operations 3,Michelle,611,2014-11-15,IT

4,Ryan,729,2014-05-11,HR

5,Gary,843.25,2015-03-27,Finance

6,Nina,578,2013-05-21,IT

7,Simon,632.8,2013-07-30,Operations 8,Guru,722.5,2014-06-17,Finance

# ▷ Reading a CSV File

* + Following is a simple example of read.csv() function to read a CSV ﬁle available in your current working directory −

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

# ▷ Get Information

* + Use the dim() function to ﬁnd the dimensions of the data set, and the names() function to view the names of the variables:

dim(data) names(data) rownames(data)

▷ By default, the **read.csv() function gives the output as a data frame.**

## ▷ Access the data frame by using the $ sign, and the name of the variable.

data$name

## ▷ Once we read data in a data frame, we can apply all the functions applicable to data frames

▷ Example - **Get the maximum salary** # Get the max salary from data frame. sal <- max(data$salary)

print(sal)

## ▷ Sort data

* + To sort the values, use the sort() function: sort(data$salary)

## ▷ Analyzing the Data

* + Now that we have some information about the data set, we can start to analyze it with some statistical numbers.
  + For example, we can use the summary() function to get a statistical summary of the data:
  + Example

Data\_Cars <- mtcars summary(Data\_Cars)

## ▷ The summary() function returns six statistical numbers for each variable:

* + Min
  + First quantile (percentile)
  + Median
  + Mean
  + Third quantile (percentile)
  + Max

## ▷ Some Examples of accessing data

▷ Get the details of the person with max salary

## ▷ We can fetch rows meeting speciﬁc ﬁlter criteria similar to a SQL where clause.

# Create a data frame.

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

# Get the max salary from data frame. sal <- max(data$salary)

# Get the person detail having max salary. retval <- subset(data, salary == max(salary)) print(retval)

## ▷ Get all the people working in IT department

# Create a data frame.

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

details <- subset( data, dept == "IT") print(details)

## ▷ Get the persons in IT department whose salary is greater than 600

# Create a data frame.

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

info <- subset(data, salary > 600 & dept == "IT") print(info)

## ▷ Get the people who joined on or after 2014

# Create a data frame.

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

info <- subset(data, as.Date(start\_date) > as.Date("2014-01-01")) print(info)

# ▷ Writing into a CSV File

* + R can create csv ﬁle form existing data frame.
  + The write.csv() function is used to create the csv ﬁle. This ﬁle gets created in the working directory.

# Create a data frame.

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

retval <- subset(data, as.Date(start\_date) > as.Date("2014-01-01")) # Write ﬁltered data into a new ﬁle.

write.csv(retval,"output.csv")

newdata <- read.csv("output.csv") print(newdata)

▷ Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format.

▷ R can read directly from these ﬁles using some excel speciﬁc packages.

▷ Few such packages are - XLConnect, xlsx, gdata etc.

▷ We will be using xlsx package. R can also write into excel ﬁle using this package.

▷ Install xlsx Package

* + You can use the following command in the R console to install the "xlsx" package.
  + It may ask to install some additional packages on which this package is dependent.
  + Follow the same command with required package name to install the additional packages. install.packages("xlsx")

## ▷ Verify and Load the "xlsx" Package

* + Use the following command to verify and load the "xlsx" package.
    - # Verify the package is installed.
    - any(grepl("xlsx",installed.packages()))
  + Load the library into R workspace.
    - library("xlsx")
    - print(library("xlsx"))

# ▷ Input as xlsx File

* + You should save it in the current working directory of the R workspace.

# ▷ Reading the Excel File

* + The input.xlsx is read by using the read.xlsx() function as shown below. The result is stored as a data frame in the R environment.

# Read the ﬁrst worksheet in the ﬁle input.xlsx. data <- read.xlsx("input.xlsx", sheetIndex = 1) print(data)

# ▷ Writing data into Excel File

* + In R, we can also write the data into our .xlsx ﬁle.
  + R provides a write.xlsx() function to write data into the excel ﬁle.
  + There is the following syntax of write.xlsx() function: write.xlsx(data\_frame,ﬁle\_name,col.names,row.names,sheetnames,append)
    - The data\_frame is our data, which we want to insert into our excel ﬁle.
    - The ﬁle\_names is the name of that ﬁle in which we want to insert our data.
    - The col.names and row.names are the logical values that are specifying whether the column names/row names of the data frame are to be written to the ﬁle.
    - The append is a logical value, which indicates our data should be appended or not into an existing ﬁle.

▷ Example

# Create a data frame.

data <- read.xlsx("input.xlsx", sheetIndex = 1)

empdata <- subset(data, as.Date(start\_date) > as.Date("2014-01-01")) # Write ﬁltered data into a new ﬁle.

write.xlsx(empdata,"emp.xlsx",col.names=TRUE, row.names=TRUE,sheetName="Sheet2",append = TRUE)

newdata <- read.xlsx("emp.xlsx",sheetIndex = 1) print(newdata)