Welcome to Math's Tutorial

Session-3

Data handling

Packages required

- Packages should be installed and then load packages using library function.

- install(rJava)
- install(XLSConnectjars)
- install(xls)
- Data.table

Types of files

- Text files.
- Excel files.
- CSV files.

How to Import file in RStudio

Steps:

1. Create file (text file, excel file, csv file,…)
2. Importing into Rstudio:
   - Click on Import Dataset in Environment workspace
   - Choose your saved file
   - View it in top level workspace
3. Command to run the file:

   source(“file_path”)

Work with Table File

- A data table can resides in a text file. The cells inside the table are separated by blank characters.

Example:

```
100   a1   b1
200   a2   b2
```

- Save with **.txt extension**. Then load the data into the workspace with the function read.table.

Command:

```
mydata = read.table("file_path")      # read text file
mydata                                # print data frame
```

WORK WITH .CSV FILE

- The sample data can be in **comma separated values** (CSV) format. Each cell inside such data file is separated by comma.
- The first row of the data file should contain the column names instead of the actual data
- Example:

  Col1,Col2,Col3
  100,a1,b1
  200,a2,b2

- Save with **.csv extension**

Command:

mydata = read.csv("Filename") # read csv file

Example

- **Mydata.csv**

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

- **read.csv()** function to read a CSV file available in your current working directory:

```
data <- read.csv(“mydata.csv")
print(data)
```

Analyzing the CSV File

- By default the read.csv() function gives the output as a data frame.
- Also we can check the number of columns and rows.

```
data <- read.csv("mydata.csv")
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

CONTD..

- Once we read data in a data frame, we can apply all the functions applicable to data frames.
- **Get the maximum salary:**

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

CONTD..

- **Get the details of the person with max salary**

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

```
# 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)
```

CONTD..

- **Get the people who joined on or after 2014.**

```
# Create a data frame.
 retval <- subset(data,
as.Date(start_date) > as.Date("2014-01-01"))

print(retval)
```

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.

```
# Write filtered data into a new file.
write.csv(retval,"output.csv")
newdata <- read.csv("output.csv")
print(newdata)
```

- Here the column X comes from the data set newdata.
- This can be dropped using additional parameters while writing the file.

CONTD..

```
# Write filtered data into a new file.

write.csv(retval,"output.csv", row.names=FALSE)
newdata <- read.csv("output.csv")

print(newdata)
```

To extract specific columns from csv

- # will extract 1 and 3 columns.

  modifiedDataFile1 =data [,c(1,3)];

#write extracted data to file

- write.csv(modifiedDataFile1, file = "Myinfo.csv")

EXAMPLE

```
vec_rev=c(100,20,500)
vec_mar=vec_rev*0.02
vec_city=c("HUBLI","DHARWAD",“BELGAUM")

salesdf=data.frame(vec_rev,vec_mar,vec_city)

write.csv(salesdf,"mydataframe.csv",
row.names=FALSE)
salesdf_2=read.csv("mydataframe.csv")
salesdf_2
```

R – Data Reshaping

- Data Reshaping in R is about changing the way data is organized into rows and columns.
- Most of the time data processing in R is done by taking the input data as a data frame.
- It is easy to extract data from the rows and columns of a data frame but there are situations when we need the data frame in a format that is different from format in which we received it.
- R has many functions to split, merge and change the rows to columns and vice-versa in a data frame.

CONTD..

- We can join multiple vectors to create a data frame using the **cbind()** function.

- Also we can merge two data frames using **rbind()** function.

Rbind and cbind functions

```
first_row <- c(1,2,3)
second_row <- c(10,20,30)
third_row <- c(100,200,300)
fourth_row <- c(1000,1000,1000)

tmp <- rbind(first_row, second_row, third_row, fourth_row)

row_scores <- rowSums(tmp)
scores <- cbind(tmp, row_scores)

rownames(scores) <- c("row1", "row2", "row3", "row4")

colnames(scores) <- c("c1", "c2", "c3", "total")
scores
```

EXAMPLE

```
#rbind
rd1=data.frame(cI=c(1:6),product=c(rep("toster",3),rep("radio"
    ,3)))
rd2=data.frame(cI=c(1:4),product=c(rep("TV",3),rep("Mobile",
    1)))
rd3=rbind(rd1,rd2)
rd3


#cbind
cd1=data.frame(cI=c(1:6),product=c(rep("toster",3),rep("radio"
    ,3)))
cd2=data.frame(cI=c(rep("IND")))
cd3=cbind(cd1,cd2)
cd3
```

Data tables

- package called data.table that extends and enhances the functionality of data.frames.

- data.tables have an index like databases. This allows faster value accessing, group by operations and joins.

EXAMPLE

```
theDF<-data.frame( A=1:10,
                   B=letters[1:10],
                   C=LETTERS[11:20],
                   D=rep(c("one","two","three"),length.out=10))
theDF
class(theDF$B)
write.csv(theDF,"datatable2.csv",row.names=FALSE)


theDT<-data.table( A=1:10,
                    B=letters[1:10],
                    C=LETTERS[11:20],
                    D=rep(c("one","two","three"),length.out=10))
theDT
class(theDT$B)
write.csv(theDT,"datatable1.csv",row.names=FALSE)
```

R – Strings

- Any value written within a pair of single quote or double quotes in R is treated as a string.
- Internally R stores every string within double quotes, even when you create them with single quote.
- Examples of Valid Strings

```
a <- 'Start and end with single quote'
b <- "Start and end with double quotes"
c <- "single quote ' in between double quotes"
d <- 'Double quotes " in between single quote'
```

String Manipulation

- Concatenating Strings - paste() function

Syntax:

paste(..., sep = " ", collapse = NULL)

- ... represents any number of arguments to be combined.
- sep represents any separator between the arguments. It is optional.
- collapse is used to eliminate the space in between two strings. But not the space within two words of one string.

EXAMPLE

```
a <- "Hello"
b <- 'How'
c <- "are you? "

print(paste(a,b,c))

print(paste(a,b,c, sep = "-"))

print(paste(a,b,c, sep = "", collapse = ""))
```

String manipulation

- paste("good","bad")
- paste(c("good","bad"),c("morning","evening"))
- paste(c("good","bad"),c("morning","evening"),sep="/")
- paste("good",c("morning","evening"))

Counting number of characters in a string - nchar() function

- This function counts the number of characters including spaces in a string.
- Syntax :

  nchar(x)
- x is the vector input.
- Examle:

```
result <- nchar("Count the number of characters")
print(result)
```

Changing the case – toupper() & tolower()

- functions These functions change the case of characters of a string.
- Syntax :

  toupper(x)

  tolower(x)
- Example

```
# Changing to Upper case.
result <- toupper("Changing To Upper")
print(result)
# Changing to lower case.
result <- tolower("CHANGING TO LOWER")
print(result)
```

Extracting parts of a string - substring() function

- Syntax :

  substring(x,first,last)

- x is the character vector input.
- first is the position of the first character to be extracted.
- last is the position of the last character to be extracted.
- Example:

```
# Extract characters from 5th to 7th position.
result <- substring("Extract", 5, 7)
print(result)
```

THANK YOU!!!