|  |  |  |
| --- | --- | --- |
|  |  |  |

**Big Data Technologies**

**Lab Manual**

**AY 2020-21**

****

**Department of Computer Science & Engineering**



|  |  |
| --- | --- |
| ***Department:****Computer Science and Engineering* | ***Course Type:*** *Programme Core* |
| ***Course Title****: Big Data Technologies Laboratory* | ***Course Code:****18CSL66* |
| ***L-T-P:****0-0-2* | ***Credits:****1* |
| ***Total Contact Hours:****36 Hours* | ***Duration of SEE:****3Hours* |
| ***SEE Marks****: 50* | ***CIE Marks:****50* |

**COURSE DESCRIPTION**

In this course you will learn how to program in R and how to use R for effective data analysis and will learn scala and Spark to drive better business decisions and solve real-world problems.

**PREREQUISITES**

* Basics of Java (preferred), [Python](https://cognitiveclass.ai/courses/python-for-data-science/) or another object-oriented language.

**COURSE OBJECTIVES**

**At the end of the course students will be able to**

1. Get a solid understanding of the fundamentals of the language, the tooling, and the development process.
2. Tackle data analysis problems involving Big Data, Scala and Spark.
3. Design and write efficient programs using R to perform routine and specialized data manipulation/management and analysis tasks.
4. Document, share, and collaborate on code development using a suite of Open Source standards and tools.
5. Develop a good application of more advanced features.

**LAB EXERCISES**

**Part A**

**Implement the following exercises using R**

1. **a.** Create three different variables, one that is numeric type and other two are vector of characters. Use these to create data frame of student.(USN, Name, Marks)

**b.** Add a new numeric data column to the existing data frame (Age). Provide summary of the data

**c.**  Display the list of students whose Age is less than 20 and Marks greater than 25.

1. Write a program to create the csv file for storing Employee data, containing the fields

(EmpID, EmpName, DOJ,Dept, Desig.)

1. Read the suitable number of employee details from the user.
2. Create a dataframe of Employee
3. Store the dataframe in the csv file
4. Read the data from csv and Display the contents
5. Append a new row into the csv file
6. Exploring Dataset

**a.** List the data set available in your system using suitable command

**b.** Select “mtcars” data set, find and display the number of rows and columns in that data set

**c.** Find are there more automatic (0) *or* manual (1) transmission-type cars in the dataset?

*Hint: 9th column indicates the transmission type*

**d.** Get a scatter plot of ‘hp’ vs ‘weight’.

**e.** Change ‘am’, ‘cyl’ and ‘vs’ to *integer* and store the new dataset as ‘newmtc’.

**f.** Extract the cases where cylinder is less than 5

1. Consider “Airquality” dataset
2. Display the dimension ofthe dataset
3. Display the class of each fields in the data set
4. Test the missing values
5. Recode the missing values, as mean ofthe column values
6. Exclude the missing values

**Implement the following exercises using Scala**

1. Write a program that reads words from a file. Use a mutable map to count how often each word appears.
2. Write a function minmax (values: Array[Int]) that returns a pair containing thesmallest and largest values in the array.
3. Write the menu driven program to implement quick sort algorithm using imperative style and functional style.
4. Write the program to illustrate the use of pattern matching in scala, for the following

Matching on case classes. Define two case classes as below:  
**abstractclassNotification**  
**caseclassEmail**(sender: **String**, title: **String**, body: **String**) **extendsNotification**  
**caseclassSMS**(caller: **String**, message: **String**) **extendsNotification**Define a function showNotification which takes as a parameter the abstract type Notification and matches on the type of Notification (i.e. it figures out whether it’s an Email or SMS).   
In the case it’s an Email(email, title, \_) return the string: s"You got an email from $email with title: $title“  
In the case it’s an SMS return the String: s"You got an SMS from $number! Message: $message“

**Part B**

**Implement the following exercises using Spark**

1. WordCount: Here the goal is to count how many times each word appears in a file and write out a list ofwords whose count is strictly greater than 4.  
     
   Use the file log.txt accompanying this assignment to count the words.Save the wordcounts in text form in the "wordcountsDir" using the saveAsTextFile RDD method.Examine the contents ofthe above directory, andthe contents ofthe files ofthe directory.
2. Tweet Mining: A dataset with the 8198 reduced tweets, reduced-tweets.json will be provided.The data contains reduced tweets as in the sample below:  
     
   {"id":"572692378957430785",  
   "user":"Srkian\_nishu :)",  
   "text":"@always\_nidhi @YouTube no idnt understand bti loved of this mve is rocking",  
   "place":"Orissa",  
   "country":"India"}  
     
   A function to parse the tweets into an RDD will be provided. The task is to print the top 10 tweeters.

**Self Demonstartion of the below programs**

1. IPLTossWinStats: You will be provided with a dataset from the Indian Premier League containing the following files:  
   Ball\_by\_Ball.csv, Match.csv, Player.csv, Player\_Match.csv, Season.csv, Team.csv.  
   We want to find the percentage of game wins by teams which win the toss.Solets say N games have been played. Let us say there are M games where the team which has won the toss has also won the game. So we are looking for the percentage (M \* 100 / N).Perform the task using SQL code only.
2. Streaming Rainfall Averages: Consider the scenario that there are three weather stations in Bangalore which report the rainfall at the respective locations once every 15 minutes. You have to write a Spark Streaming application which will gather the rainfall data from the three stations and print the average rainfall, also once every 15 minutes.  
   You will be provided with a scala program, generate Events, which can simulate generation of the rainfall data from the three stations in JSON format as shown below to a folder:{"Creation\_Time": 1.53633593969400013E18,"Station": "Bengaluru-1","Rainfall": 100.0} Write a Spark streaming application which reads the files written to the above folder and updates the average rainfall value every 15 minutes and prints the averages to the console.

**ASSESSMET METHODS:**

|  |  |
| --- | --- |
| **Parameters** | **Marks** |
| Experiment Write up + Execution + Viva | 15 |
| Lab Record Writing | 10 |
| Lab Internals Test | 15 |
| **Total** | **50** |
| Final Exam will be conducted for 100 marks (SEE) | |

**COURSE OUTCOMES**

At the end of the course student will be able to

|  |  |  |
| --- | --- | --- |
| **COs** | **Description** | **Bloom’s Level** |
| **CO 1** | Apply the concept of R programming for cleansing, imputation, and computation of simple statistical measures on the data. | **L3** |
| **CO 2** | Understand the basics of Scala for data analysis. | **L2** |
| **CO 3** | Design a Spark code for basic data manipulation and aggregate analysis. | **L4** |
| **CO 4** | Implement real time application such as word count, data mining into a set of distributed computations and implement the same in Spark using Scala. | **L3** |
| **CO 5** | Implement analytics on higher level data objects like Tables using Spark SQL and analytics on Streaming datasets using Spark Streaming. | **L3** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Mapping of Course outcomes (COs) to Program outcomes (POs\*)& PSO \*\*** | | | | | | | | | | | | | | | |
| **Course Outcomes mapping to Program Outcomes** | | | | | | | | | | | | | **PSOs** | | |
| **POs**  **COs** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **PSO1** | **PSO2** | **PSO3** |
| **CO1** | 3 | 2 |  |  |  |  |  |  |  |  |  |  | 2 |  |  |
| **CO2** | 3 | 3 | 1 |  |  | 2 | 2 |  |  |  |  |  |  | 2 |  |
| **CO3** | 3 | 3 | 2 |  | 2 |  |  |  |  |  |  |  |  | 2 |  |
| **CO4** | 3 | 3 | 2 | 3 | 2 |  |  |  |  |  |  |  |  | 2 |  |
| **CO5** | 3 | 3 | 3 |  | 3 | 2 |  |  |  |  |  |  |  |  | 2 |

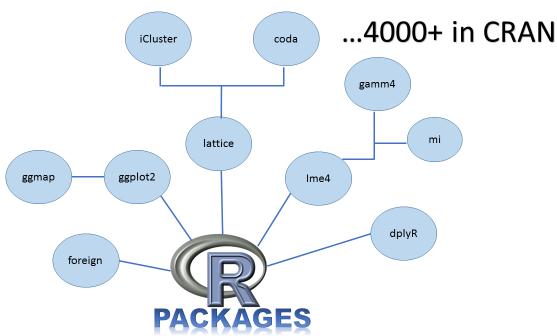
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Course Outcome** | **POs** | **PSOs** | **Class  Sessions** | **Lab  Sessions/Assignments/Course Project/Tutorials (Hrs)** |
| CO1 | Apply the concept of R programming for cleansing, imputation, and computation of simple statistical measures on the data. | PO1, PO3,  ,PO5 | PSO1 | 12 | 10  8 |
| CO2 | Understand the basics of Scala for data analysis. | PO1, PO3,  PO5 | PSO1 |
| CO3 | Design a Spark code for basic data manipulation and aggregate analysis. | PO1, PO3,  PO5 | PSO1  PSO2 |
| CO4 | Implement real time application such as word count, data mining into a set of distributed computations and implement the same in Spark using Scala. | PO1,PO3,  PO5 | PSO1 |
| CO5 | Implement analytics on higher level data objects like Tables using Spark SQL and analytics on Streaming datasets using Spark Streaming. | PO1,PO3,PO8,PO9,PO10 | PSO1, PSO2, PSO3 |
| Total Hours of instruction | | | | 12 | 12+18=30  Hours |

**INTRODUCTION TO R**

* The R language is a terminology of S language which was designed in the 1980s by John Chambers at Bell labs and has been in widespread use in the statistical community since.
* It was grown up by Robert Gentleman and Ross Ihaka of the University of Auckland. R has been with us since 1993.
* After the four versions of ‘S’ language we came up with **R Programming tool** and **R Programming language**.
* It is concluded to adopt the syntax of the S language which has developed at Bell Laboratories

**The Development of R**

* 1991: Created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand
* 1993: First announcement of R to the public.
* 1995: R was made as free software.
* 1997: The R Core Group is formed (containing some people associated with S-PLUS). The core group controls the source code for R.
* 2000: R version 1.0.0 is released.
* 2013: R version 3.1.2 has been released on 2014-10-31.
* Latest version 3.4.1 released on 2017-6-30



**Why R essential?**

R provides you number of Datasets that can be used in Analytics and these datasets are built in and are available in packages

* Time Series Data
* Numeric Data
* Categorical Data
* Character Data
* Small Data
* Large Data

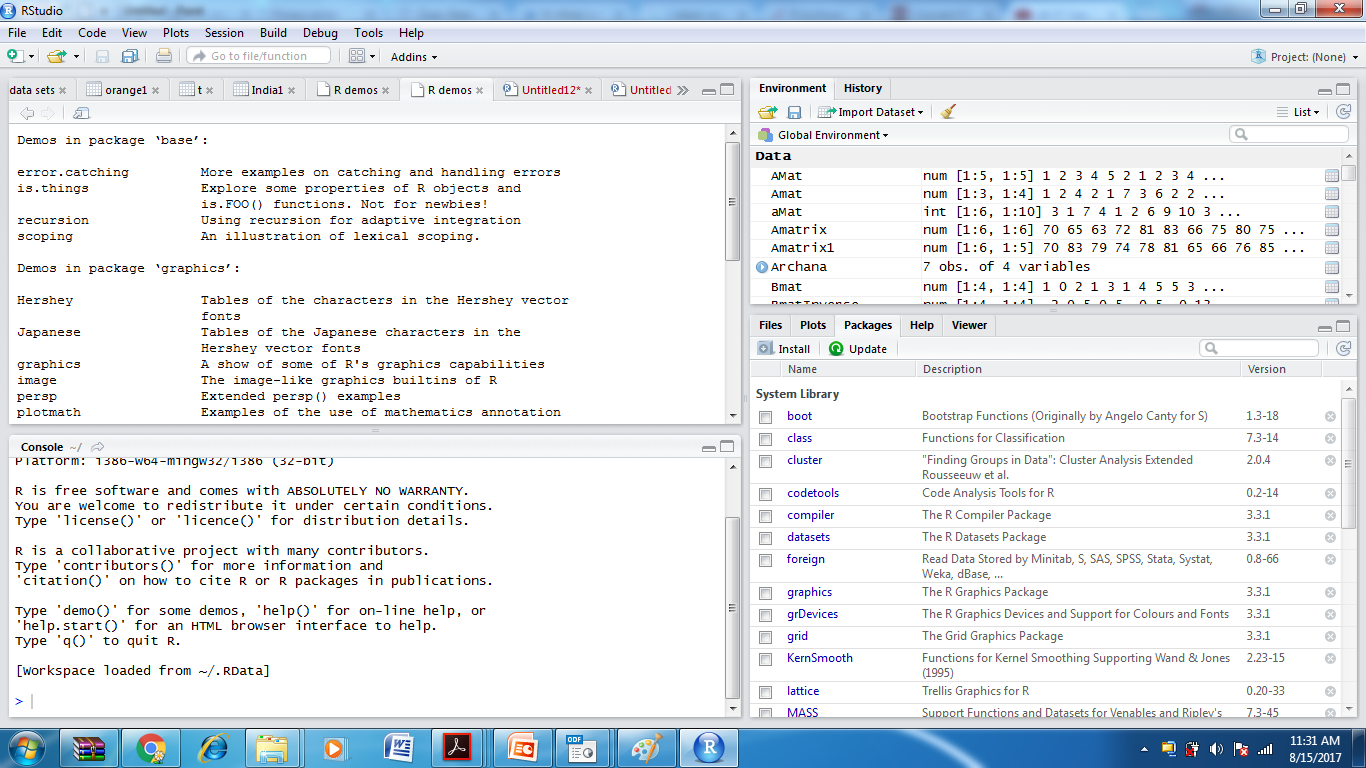
**What is R?**

* R is rather a programming language
* Limited user-friendly interfaces for data analysis
* Is object oriented and almost non declarative
* Similar to programming languages like Fortran, C, Java, Python
* The source code for the R software environment is written primarily in C, Fortran, and R

**Features of R**

* Provides Statistical and Graphical techniques
  + linear and nonlinear modelling
  + classical statistical tests
  + time-series analysis,
  + classification, clustering
  + Others
  + Available through additional packages
* easily extensible through functions and extensions
* C, C++, and Fortran code can be linked and called at run time
* Object-oriented, growing user base, scripting features
* Free and open-source

**R Studio –IDE**

****

**Working with R**

* Download & Install R: http://www.r-project.org/
* Download & Install R Studio:
* Materials
  + https://www.r-project.org/

**R Basics**

Basic Data types and how to create them

Operators and how to use them

Variable naming and how to create, manipulate and destroy

* R is functional language
* Each command given on the command prompt is either

Expression

1. + 5

sqrt(30)

Assignment

X <- 4 + 5

1. -> Y

Functions

User Defined

Built in ( Take the help)

Variables

Similar conventions used as in other programming languages can be declared as and when required

* 1. **Data Types**
     1. R calls data types /categorizes data types as ***classes***
     2. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable.
     3. Frequently used data types in R are
        1. Vectors
        2. Lists
        3. Matrices
        4. Arrays
        5. Factors
        6. Data Frames
           1. **Vectors**
  + There are six data types of these ***atomic*** vectors, also termed as six classes of vectors.
  + The other R-Objects are built upon the atomic vectors.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Example** |  |
| Logical | TRUE, FALSE |  |
| Numeric | 12.3,5.0,0.999 | Numbers , including decimals |
| Integer | 12L | Integers |
| Complex | 5+9i | Complex() |
| Character | ‘a’, ’Hello’ | Data within the quotes |
| Raw | ‘Hello’ | Raw() – used to create  Raw numbers are represented as a two digit sequence of hex numbers. Valid hex digits include 0 − 9 as well as a, b, c, d, e, & f. |

* **Constants**

|  |  |
| --- | --- |
| **Name** | **Description** |
| pi | Mathematical functions ¶ |
| NULL | Absence of type |
| Nan | Not a Number |
| Infinity | ∞ as well as -∞ |
| NA | Used to represent missing data |

* + - * 1. **Lists**
  + can contain many different types of elements inside it like vectors, functions and even another list inside it.

>A= list(c(2,5,4), 21.4, 5+8i)

> print(A)

[[1]]

[1] 2 5 4

 [[2]]

[1] 21.4

[[3]]

[1] 5+8i

* (Vector all data are of the same data type, Lists store the data of different data types)
  + - * 1. **Matrices**
  + Matrices are 2-dimensional vectors
  + Created using the default constructor matrix() function
  + passing it a number for nrow and ncol
  + M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)

> M [,1] [,2] [,3]

[1,] "a" "a" "b"

[2,] "c" "b" "a“

* + Default will be by column

**d. Arrays**

* 1. Any numbers of dimensions
  2. takes a *dim* attribute which creates the required number of dimension
  3. To create array of 2X3, with elements 1,2,3, dimension 3 times

> A<- array(c(1,2,3), dim=c(2,3,3))

> A

, **, 1**

**[,1] [,2] [,3]**

**[1,] 1 3 2**

**[2,] 2 1 3**

**, , 2**

**[,1] [,2] [,3]**

**[1,] 1 3 2**

**[2,] 2 1 3**

**, , 3**

**[,1] [,2] [,3]**

**[1,] 1 3 2**

**[2,] 2 1 3**

**e.Factors**

* 1. Used to categorize the data and store as levels
  2. Useful in statistical modelling

> A<- c(2,3,5,34,3,2,56,34)

> Afact= factor(A)

> Afact

[1] 2 3 5 34 3 2 56 34

Levels: 2 3 5 34 56

1. **Data Frames**
   1. Same as Spreadsheet
   2. Data frames are used to store tabular data
   3. They are represented as a special type of list where every element of the list has to have the same length
   4. Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
   5. Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class

* Example of creating Data frame

> id= c(1:4)

> Name= c("Asha","Geetha", "Rita", "Raj")

table1= data.frame(id,Name)

> table1 id Name

1 1 Asha

2 2 Geetha

3 3 Rita

4 4 Raj

Marks<<- c(25,28,30,22)

> (table1=data.frame(id,Name,Marks))

id Name Marks

1 1 Asha 25

2 2 Geetha 28

3 3 Rita 30

4 4 Raj 22

**II. Operators**

* Arithmetic Operator (Numeric Operator)
  + + Addition
  + - Subtraction
  + \* Multiplication
  + / Division
  + ^ Exponentioation
* Assignment Operator
  + ->
  + <-
  + <<-
  + =
* Relational and Logical Operator
  + >
  + <
  + ==
  + !=
  + &
  + |
  + !
* Special Operator
  + %% Mod operation
  + %/% Integer Division
  + %\*% Matrix Multiplication
  + : To build sequence
  + %in% Value Matching

**Functions/Commands**

**Flow Control**

1. If
2. If else
3. Continue
4. Break, Next
5. Switch
6. Repeat
7. For
8. while

**R data interfaces**

1. **CSV file**
   * 1. Reading

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

print(data)

2. Writing

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

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

write.csv(retval, "output.csv“)

1. **Excel File**

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.

Reading

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

print(data)

**3. Binary Files**

Sometimes, the data generated by other programs are required to be processed by R as a binary file. Also R is required to create binary files which can be shared with other programs.

R has two functions **WriteBin()** and **readBin()** to create and read binary files.

Syntax

writeBin(object, con)

readBin(con, what, n )

**4.XML file**

You can read a xml file in R using the "XML" package. This package can be installed using following command.

install.packages("XML")

# Also load the other required package. library("methods")

# Give the input file name to the function.

result <- **xmlParse**(file = "input.xml")

# Print the result. print(result)

standard ASCII text

**5. JSON file**

JSON file stores data as text in human-readable format. Json stands for JavaScript Object Notation. R can read JSON files using the rjson package.

# Load the package required to read JSON files. library("rjson")

# Give the input file name to the function.

result <- fromJSON(file = "input.json")

# Print the result.

print(result)

**6. Web Data**

The following packages are required for processing the URL’s and links to the files.

install.packages("RCurl")

install.packages("XML")

install.packages("stringr")

install.packages("plyr")

getHTMLLinks() to gather the URLs of the files

download.file() to save the files to the local system

**7. Databases**

RMySQL" which provides native connectivity between with MySql database

install.packages("RMySQL")

Connecting R to MySql

# Create a connection Object to MySQL database.

# We will connect to the sampel database named "sakila" that comes with MySql installation. mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'sakila', host = 'localhost')

# List the tables available in this database. dbListTables(mysqlconnection)

dbSendQuery()- used to query the database

The query gets executed in MySql and the result set is returned using the R fetch()

**Data Visualization**

6 different kind of graphs can be plotted

1. Pie Chart
2. Bar Chart
3. Box plot
4. Histogram
5. Line graph
6. Scatter Plot

**References**

1. <http://blog.easylearning.guru/r-programming-the-present/>
2. Iigsaw Academy, Analytics for Professionals, -Introduction to R
3. An overview of R: Text Analytics, Ashraf Uddin, PhD Scholar, Dept. of Computer Science South Asian University, New Delhi <https://sites.google.com/site/ashrafuddininfo/>
4. <https://www.youtube.com/watch?v=eDrhZb2onWY>
5. <https://www.tutorialspoint.com/r/r_data_types.htm>
6. https://www.tutorialspoint.com/r/r\_csv\_files.htm

**Installation of Ubuntu 18.04 on VMware Workstation Player Guide**

# **Contents**

1. Introduction
2. Requirements
3. Installation Instructions
4. Further Steps
5. References

Please go to **View** and enable **Show document outline** for easier navigation.

# **Introduction**

This guide will give you step by step instructions on installing Ubuntu 18.04 on VMware Workstation Player on your laptop.

There are other distros of Linux you can also use, like Mint OS and Fedora. They will have slightly different instructions. Also, there are other solutions to run a virtual machine, like Oracle’s VirtualBox, but this guide will use VMware Workstation Player.

If you run Windows 7, you can also consider dual booting as an option, as it is slightly easier to setup in Windows 7 compared to in Windows 10.

You can view the References, where I have linked the videos followed to compile the below instructions.

* + - 1. **Requirements**

You will need a laptop with the following minimum requirements,

**CPU**: 2 GHz dual core processor or better (as specified in the Ubuntu website)

Please note that your CPU supports virtualization and it should be enabled in the BIOS (Most modern CPUs support virtualization and it is enabled out of the box).

**RAM**: It is recommended to have a minimum of 8 GB, as the Virtual Machine runs on top of your already existing OS.

**Hard Drive**: Between 30GB to 60GB of free space.

# **Installation Instructions**

## Downloading Required Software

1. First download VMware Workstation Player setup from [here](https://www.vmware.com/in/products/workstation-player/workstation-player-evaluation.html).

(<https://www.vmware.com/in/products/workstation-player/workstation-player-evaluation.html> if not visible)

1. Then download the Ubuntu image from [here](https://www.ubuntu.com/download/desktop). It should automatically pick the 64-bit version and Ubuntu 18.04.1.

(<https://www.ubuntu.com/download/desktop> if not visible)  
The first download is around 90MB, while the second is around 1.8GB, so you might want to copy that from someone who has it already downloaded.

## VMWare Installation and Creating a Virtual Machine

1. Run the VMware Workstation Player installation setup. You can choose to install it in any location.
2. Run the VMware Workstation Player application. Choose “Use VMware Workstation Player 14 for free for non-commercial use” and click **Continue**, and then **Finish**.
3. Click “Create a New Virtual Machine”. In the window that opens, click “I will install the operating system later” and click **Next**.
4. Choose “Linux” and then “Ubuntu 64-bit” from the dropdown menu, and click **Next**.
5. Give your VM a name. Note that this the is the name in VMware Workstation Player and not the actual root username.  
   You can also choose where you want to store the VM. Then click **Next**.
6. Provide the Maximum disk size. It is recommended you give between 40GB to 60GB, depending on how much free space you have, as you will have to install many other things other than the OS.  
   Choose “Store virtual disk as single file”, and click **Next**.
7. Click **Customize Hardware** in the next window.
   1. Provide a minimum of 2GB of **Memory**. If you have more than 8GB of RAM on your laptop you can provide 4GB+ of **Memory** for better performance.
   2. Change the “Number of processor cores” to 2 in **Processors**.
   3. Go to **New CD/DVD** and choose “Use ISO image file”. Then click **Browse**. Locate the folder you downloaded the Ubuntu image and select the ISO.
   4. In **Display**, disable the “Accelerate 3D graphics” option.
   5. Click **Close**, and then **Finish**.

## Installing Ubuntu on the VM

1. Click the **Play virtual machine** button to run the installation.  
   If you get a popup asking you if you want to install VMware Tools, click **Remind Me Later.**
2. A new window should appear. Click **Install Ubuntu**.
   1. Choose the Keyboard Layout. You can leave it at the default. Click **Continue**.
   2. Choose “Normal Installation” and “Install third-party software…”. Click **Continue**.
   3. Choose “Erase disk and install Ubuntu”. Click **Install Now**. Click **Continue** if you get a warning.
   4. Choose Kolkata as the Location, if it isn’t selected by default. Click **Continue**.
   5. Add all the required details. Please choose carefully as these are the names your system will use (You can give a simple name for the computer’s name such as “ubuntu-18”). Click **Continue** after completing.
3. It could take a few minutes for the installation. After that is completed, click **Restart Now**.
4. Login to check if the installation was successful.

# **Further Steps**

These steps will help get your VM ready for development use.

1. Open a new Terminal by right-clicking and selecting “Open Terminal” or pressing **Ctrl+Alt+T**.
2. Type “sudo apt update” and click Enter. Then type “sudo apt upgrade” and click Enter. Type “Y” at the next prompt.
3. Now to install the VMware Tools,
   1. Click **Player** (top left) -> **Manage** -> **Install VMware Tools**. Click on **Install** (if you get a dialog box).
   2. Open Files and you should see a DVD called “VMware Tools”. Open it and copy the “VMwareTools-10…” folder to the Desktop or your preferred location.
   3. Open a Terminal in that location and type “tar -zvxf VMwareTools” and press Tab. It should autocomplete. Press Enter.
   4. Change to the new folder that was unpacked by typing “cd vmware-” and pressing Tab. You can type ls to view the files/folders present.
   5. Type “ sudo ./vmware-install.pl” and press Enter.
   6. For all prompts you can just press Enter, as it will take the default value.
   7. After that installation is done, restart the VM.
   8. Now you will be able to enjoy features such as full screen view and clipboard and file sharing between systems.
   9. You can eject the disk and delete the folders.
4. After the restart, open a Terminal and type “sudo apt-get install build-essential git”. Press Enter. This will get you all the latest essential compilers like gcc, g++ among some other helpful things.
5. You can install an advanced text editor like VS Code if you want. The instructions to install VS Code can be found [here](https://code.visualstudio.com/docs/setup/linux).  
   (<https://code.visualstudio.com/docs/setup/linux> if not visible)
6. And that’s all! You have Ubuntu 18.04 running on a VM in VMware Workstation Player on your laptop.

# **References**

* Installation of VM: <https://www.youtube.com/watch?v=oyNjjzg-UXo>
* Installation of VMware Tools: <https://www.youtube.com/watch?v=bxNsXPReIRE>

Summary of Installation (On Linux)

Url: <http://sparkubuntu.blogspot.com/2019/07/installation-of-jdk-8-rr-studiosbt.html>

## **[Installation of java 8](http://sparkubuntu.blogspot.com/2018/08/installation-r-in-ubuntu-1804.html)**

#### **sudo su**

#### **sudo apt install openjdk-8-jdk**

## **Steps to install R & R Studio**

#### **sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9**

#### **sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran35/'**

#### **sudo apt update**

#### **sudo apt install r-base**

#### **Test Install**

#### **Start R’s interactive shell as root.**

#### **sudo -i R**

### **Install Rstudio**

#### **a. download latest Rstudio version**

#### [**https://download1.rstudio.org/desktop/bionic/amd64/rstudio-1.2.1335-amd64.deb**](https://download1.rstudio.org/desktop/bionic/amd64/rstudio-1.2.1335-amd64.deb)

#### **cd Downloads**

#### **sudo dpkg -i**[**rstudio-1.2.1335-amd64.deb**](https://download1.rstudio.org/desktop/bionic/amd64/rstudio-1.2.1335-amd64.deb)

## 

## **COMMANDS TO INSTALL SBT**

#### **echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list**

#### **sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823**

#### **sudo apt-get update**

#### **sudo apt-get install sbt**

#### **if anything gives error while sbt installation then try belove command**

#### **sudo apt --fix-broken install**

#### **# the following command will take several minutes. it will finally start a sbt shell.**

#### **sbt**

#### **# exit the shell by typing exit at the prompt, i.e.**

#### **exit**

#### **Installation of scala**

#### **sudo apt-get install scala**

### **SPARK installation steps in ubuntu 18.04**

#### **Download Spark:**

#### [**http://spark.apache.org/downloads.html**](http://spark.apache.org/downloads.html)

#### **cd Downloads**

#### **tar xzvf spark-2.4.3-bin-hadoop2.7.tgz**

#### **mv spark-2.4.3-bin-hadoop2.7 spark**

#### **sudo su**

#### **sudo mv spark /usr/lib/**

#### **Add Path:**

#### **sudo nano ~/.bashrc**

#### **export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64**

#### **export SPARK\_HOME=/usr/lib/spark/**

#### **export PATH=$PATH:$SPARK\_HOME/bin**

#### **Open a new terminal and verify installation by running the spark shell:**

#### **machineName:~ spark-shell**

**Suggestions to solve the Problem**

1. Perform the required operations using methods
   * 1. As.integer() to declare integer number
     2. Vector() to declare vector
     3. Readline() to read intput from the user
     4. Dataframe()
     5. Loop - for loop
     6. Cbind()
2. Steps to be followed
   1. Use vector() method to declare the employee details
   2. Use data.frame() method to convert multiple vector to a dataframe
   3. Use method write.csv() to create csv file
   4. Verify the csv & csv2 file types
   5. Use read.csv() to read from csv file
   6. Use write.table() to append the record in csv
3. Perform the required operations using methods
   1. Data()
   2. Head()
   3. Nrow()
   4. Ncol()
   5. Data.frame()
   6. Scatter.smooth()
   7. As.integer()
4. Perform the required operations using methods
   * + - 1. Read data from airquality data set
         2. Use method dim() to know the dimension of data set

Use sapply() to check the class of each field

* + - * 1. To check missing values method is.na()
        2. Check the missing values in each column , replace it with mean of the data. Use mean()
        3. Use na.omit()

1. **Problem statement:**

Write a program that reads words from a file. Use a mutable map to count how often each word appears

-----------------

The goal is to count how many times each word appears in a text file which is passed as an argument to the application.

The word counts should be printed to the console.

Import required packages

Use mutable map to store count of each word in the key value pair. Key holding the word, and value holding the count of words

1.Read single at a time from the file use

line <- Source.fromFile(filename).getLines

2.Split the line into words based on space

word <- line.split(" ")

1. Store the word and its respective count as the (key, value) pair in the Map collection

To comile the program use:

Sbt compile

To run the program

Sbt run

To provide command line argument:

sbt "run t.txt"

1. **Problem statement:**

Write a function minmax(values: Array[Int]) that returns a pair containing thesmallest and largest values in the array.

Take an input from the user via the console of the number of elements he/she will provide for which minimum and maximum values have to be found. Assume that all the elements are of type Int.

Let us say the user wants to provide 10 integers for which we have to find min and max.

* + - 1. Read in the 10 integers from the console.
      2. Define the function, which takes the array of intgers as parameter
      3. Function finds the minimum & maximum number
      4. Return the result as a tuple
      5. Print out the minimum and maximum.

Use StdIn and ArrayBuffer packages.

1. **Problem statement:**

Write the program using imperative style and functional style for the quick sort algorithm.

Write a function to implement quicksort in scala using functional style programming. Use the filter operator with appropriate filters to find the elements less than the pivot and the elements more than the pivot. The sort should be recursive.

The elements to be sorted can be provided as a list in the main function itself and the quicksort functions should be called to sort the list elements. Print both the original list and the sorted list onto the console.

1. **Problem statement:**

Write the program to illustrate the use of pattern matching in scala, for the following

Matching on case classes. Define two case classes as below:  
**abstractclassNotification**  
**caseclassEmail**(sender: **String**, title: **String**, body: **String**) **extendsNotification**  
**caseclassSMS**(caller: **String**, message: **String**) **extendsNotification**Define a function showNotification which takes as a parameter the abstract type Notification and matches on the type of Notification (i.e. it figures out whether it’s an Email or SMS).   
In the case it’s an Email(email, title, \_) return the string: s"You got an email from $email with title: $title“  
In the case it’s an SMS return the String: s"You got an SMS from $number! Message: $message“

Description

Let us say we have the following requirement.

We need to build and app which will notify a user when she/he receives and email or an sms.

An email consists of the senders email address, subject and a body.

An SMS consists of the senders mobile number and a body.

Further processing will be done with the email / sms, which could include storing a history of emails/sms'es.

But we will limit ourselves to the notification only.

Assume that the information contained in an email/sms is stored in instances of an email/sms class.

Define a function, "showNotification", which will be passed the above instance. The function has to return a notification as a string as follows:

In the case it's an Email(email, title, \_) return the string: s"You got an email from $email with title: $title

In the case it's an SMS return the String: s"You got an SMS from $number! Message: $message.

Create an instance of an email and print what showNotification returns. Do the same for an SMS instance.

Note: write a scala script which can be run in a scala shell.

i.e. no need to write a program which needs to be compiled and subsequently run.

To implement the code for the above requirement, the following information will help.

We need to define an email class and an sms class.

if we define them as case classes, then we can use the instances of the classes in pattern matching to return the notifications desired.

Case classes are defined as below:

abstract class Notification

case class Email(sender: String, title: String, body: String) extends Notification

case class SMS(caller: String, message: String) extends Notification

We define the Email and SMS class with base class "Notification" so that we can pass an

object of either type to our function showNotification.

Define a function showNotification which takes as a parameter the abstract type Notification

and matches on the type of Notification (i.e. it figures out whether it's an Email or SMS)

as below.

def showNotification(notification: Notification): String = {

notification match {

case Email(emailAddr, subject, \_) =>

return desired notification as string.

case SMS(number, message) =>

return desired notification as string.

}

}

Now create an instance of each type and call the showNotification class and print the notification.

To create an instance of the case class, the new keyword is not required.

**Program9 -Problem Statement**

WordCount: Here the goal is to count how many times each word appears in a file and write out a list ofwords whose count is strictly greater than 4.  
Use the file log.txt accompanying this assignment to count the words.Save the wordcounts in text form in the "wordcountsDir" using the saveAsTextFile RDD method.Examine the contents of the above directory, and the contents of the files of the directory.

**9.1 Code Outline**

packagecom.nmit.spark.wordcount

importorg.apache.spark.SparkContext

importorg.apache.spark.SparkConf

importorg.apache.spark.rdd.RDD

objectwordcount {

def main(args: Array[String]) {

valpathToFile = "/home/subhrajit/sparkProjects/data/log.txt"

// create spark configuration and spark context: the Spark context is the entry point in Spark.

// It represents the connexion to Spark and it is the place where you can configure

// the common properties

// like the app name, the master url, memories allocation...

valconf = new SparkConf()

.setAppName("Wordcount")

.setMaster("local[\*]")

valsc = new SparkContext(conf)

// load data and create an RDD where each element will be a word by using the split operation

// with flatMap to separate the words in a line using the space separator.

// You can experiment with "map" instead of "flatMap"to understand why flatMap is required.

valwordsRdd = sc.textFile(pathToFile)

.flatMap(\_.split(" "))

/\*

\* Now count how many times each word appears!

\*/

// TODO

// Step 1: the mapper step

// We want to attribute the number 1 to each word: so we create couples (word, 1).

// TODO

// Step 2: reducer step

// Now you have a tuple (key, 1) where the key is a word,

// you want to count the occurrences of (key, 1).

// One way to do this is by using the reduce operation.

// TODO

// Step 3: Filter Step

// Now keep those words which appear strictly more than 4 times!

// You can do this using the filter operation.

// save the word counts in a textfile "wordcountsDir".

highFreqWords.saveAsTextFile("wordcountsDir")

}

}

**9.2 Supporting WordCount.sbtdependency file**

name := "WordCount"

version := "1.0"

scalaVersion := "2.11.12"

libraryDependencies += "org.apache.spark" %% "spark-core" % "2.3.0"

**9.3 Sample of InputFile log.txt**

main.cpp: In function ‘HashTable\* initializeTable(int)’:

main.cpp:175:40: warning: converting to non-pointer type ‘int’ from NULL [-Wconversion-null]

htable->table[i].element = NULL;

^~~~

main.cpp:176:40: warning: converting to non-pointer type ‘int’ from NULL [-Wconversion-null]

htable->table[i].element = NULL;

^~~~

main.cpp:177:40: warning: converting to non-pointer type ‘int’ from NULL [-Wconversion-null]

htable->table[i].element = NULL;

**Program10-Problem Statement**

**Tweet Mining**: A dataset with the 8198 reduced tweets, reduced-tweets.json will be provided.The data contains

reducedtweets as in the sample below:  
  
{"id":"572692378957430785",  
"user":"Srkian\_nishu :)",  
"text":"@always\_nidhi @YouTube no idnt understand bti loved of this mve is rocking",  
"place":"Orissa",  
"country":"India"}  
  
A function to parse the tweets into an RDD will be provided. The task is to print the top 10 tweeters.

**10.1 Code Outline**

//A function to parse the tweets in the text file into an RDD of tweets will be provided.

packagecom.nmit.spark.tweetmining

importorg.apache.spark.{SparkContext, SparkConf}

importorg.apache.spark.rdd.\_

objecttweetmining {

def main(args: Array[String]) {

if (args.length != 1) {

println()

println("Dude, I need exactly one argument.")

println("But you have given me " + args.length +".")

println("The argument should be path to json file containing a bunch of tweets. esired.")

System.exit(1)

}

pathToFile = args(0)

// Create the spark configuration and spark context

valconf = new SparkConf()

.setAppName("User mining")

.setMaster("local[\*]")

valsc = new SparkContext(conf)

// Needs an argument which is the path to the tweets file in json format.

// e.g. "/home/subhrajit/sparkProjects/data/reduced-tweets.json"

varpathToFile = ""

// The code below creates an RDD of tweets. Please look at the case class Tweet towards the end

// of this file.

val tweets =

sc.textFile(pathToFile).mapPartitions(TweetUtils.parseFromJson(\_))

// Create an RDD of (user, Tweet).

// TODO: Look at the tweet class. An object of type tweet will have fields "id", "user", "userName"

// etc. Collect all his tweets by the user. For this, you can use the field "user" of an object

// in the tweet RDD.

// Hint: the Spark API provides a groupBy method

// The code below should return RDD's with tuples (user, List of user tweets).

// TODO: For each user, find the number of tweets he/she has made.

// Hint: we need tuples of (user, number of tweets by user)

// TODO: Sort the users by the number of their tweets.

// Find the Top 10 twitterers and print them to the console.

sortedUsersByNumTweets.take(10).foreach(println)

}

}

importcom.google.gson.\_

objectTweetUtils {

case class Tweet (

id : String,

user : String,

userName : String,

text : String,

place : String,

country : String,

lang : String

)

defparseFromJson(lines:Iterator[String]):Iterator[Tweet] = {

valgson = new Gson

lines.map(line =>gson.fromJson(line, classOf[Tweet]))

}

}

**10.2 Supporting userMining.sbt Dependency file**

name := "userMining"

version := "1.0"

scalaVersion := "2.11.12"

libraryDependencies += "org.apache.spark" %% "spark-core" % "2.3.0"

**10.3Sample of reduced-tweets.json file**

{"id":"572692378957430785","user":"Srkian\_nishu :)","text":"@always\_nidhi @YouTube no i dnt understand bt i loved the music nd their dance awesome all the song of this mve is rocking","place":"Orissa","country":"India"}

{"id":"572575240615796737","user":"TagineDiningGlobal","text":"@OnlyDancers Bellydancing this Friday with live music call 646-373-6265 http://t.co/BxZLdiIVM0","place":"Manhattan","country":"United States"}

{"id":"572575243883036672","user":"Daniel Beer","text":"1/ \"Without the ability to ask the right questions, big data only leads to big misdirection...\"","place":"Claremont","country":"United States"}

**10.4 Summary Note**

Whenever you have heavyweight initialization that should be done once for many RDD elements rather than once per RDD element, use mapPartitions() instead of map(). mapPartitions() provides for the initialization to be done once per worker task/thread/partition instead of once per RDD data element for example.

Example Scenario: If we have 100K elements in a particular RDD partition then we will fire off the function being used by the mapping transformation 100K times when we use map.

Conversely, if we use mapPartitions then we will only call the particular function one time, but we will pass in all 100K records and get back all responses in one function call.There will be performance gain since map works on a particular function so many times, especially if the function is doing something expensive each time that it wouldn't need to do if we passed in all the elements at once(in case of mappartitions).

1. **Problem Statement**

IPLTossWinStats: You will be provided with a dataset from the Indian Premier League containing the following files:Ball\_by\_Ball.csv, Match.csv, Player.csv, Player\_Match.csv, Season.csv, Team.csv.  
We want to find the percentage of game wins by teams which win the toss.So let’s say N games have been played.Let us say there are M games where the team which has won the toss hasalso won the game.So we are looking for the percentage (M \* 100 / N).Perform the task using SQL code only.

* 1. **Code Outline**
* Please check that you have the following csv files, preferably in a single folder name "indian-premier-league-csv-dataset":Ball\_by\_Ball.csv Match.csv Player.csv Player\_Match.csv Season.csv Team.csv
* Find the schema in each file. To do so, you can either inspect the files or use the spark sql program "IPLTopTeams".
* We want to find the percentage of game wins by teams which win the toss. So let’s say N games have been played. Let us say there are M games where the team which has won the toss has also won the game. So we are looking for the percentage (M \* 100 / N). Perform the task using SQL code only.

packagecom.nmit.spark.ipltosswinstats

importorg.apache.spark.sql.SparkSession

objectipltosswinstats {

def main(args: Array[String]) {

valpathToDB = "/home/subhrajit/sparkProjects/data/indian-premier-league-csv-dataset"

valsparkSession = SparkSession.builder().appName("My SQL Session").getOrCreate()

importsparkSession.implicits.\_

// The Match.csv file has the toss won/match won data for every game.

// Read the file into a dataframe.

valmatchDF = sparkSession.read.format("csv").

option("sep", ",").

option("inferSchema", "true").

option("header", "true").

load(pathToDB + "/Match.csv")

// Since we have to use SQL queries, the dataframe has to be registered as a table.

// We can create a temporary table view.

matchDF.createOrReplaceTempView("matchStats")

// find the total number of entries in the table. this is equal to number of matches

played or N.

val N = sparkSession.sql("SELECT COUNT(\*) FROM matchstats")

.first()(0)

.asInstanceOf[Long]

// N.show()

// Find the subset of entries where the toss winner is also the match winner. This

will be a dataframe.

valtossNMatchwinnersDF = sparkSession.sql("SELECT \* FROM matchstats WHERE Toss\_Winner\_Id = Match\_Winner\_Id")

// register the dataframe as a temporary table so that you can use SQL queries on

it.

tossNMatchwinnersDF.createOrReplaceTempView("tossNMatchwinners")

// find the count of entries in this Table. This gives us M.

val M = sparkSession.sql("SELECT COUNT(\*) FROM tossNMatchwinners")

.first()(0)

.asInstanceOf[Long]

// M.show()

// print M \* 100 / N.

println("Percentage of times Toss Winners have won the match = " + (M\*100.0)/N +

"%")

}

}

* 1. **Supporting IPLTossWinStats.sbt file**

name := "IPLTopTeams"

version := "1.0"

scalaVersion := "2.11.12"

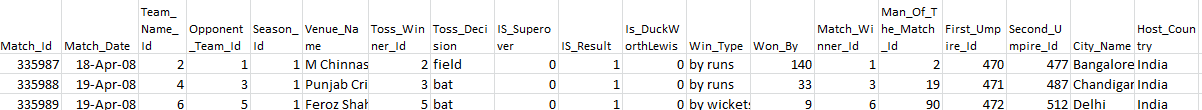
libraryDependencies ++= Seq(

"org.apache.spark" %% "spark-core" % "2.3.0",

"org.apache.spark" % "spark-sql\_2.11" % "2.1.0"

)

* 1. **Sample of Match.csv File**



1. **Problem Statement**

**Streaming Rainfall Averages**:Consider the scenario that there are three weather stations in Bangalore which report the rainfall at the respective locationsonce every 15 minutes. You have to write a Spark Streaming application which will gather the rainfall data from the threestations and print the average rainfall, also once every15 minutes.

You will be provided with a scala program, generateEvents, which can simulate generation of the rainfall data from the three stations in JSON formatas shown below to a folder:{"Creation\_Time": 1.53633593969400013E18,"Station": "Bengaluru-1","Rainfall": 100.0}  
  
Write a Spark streaming application which reads the files written to the above folder and updates the average rainfall value every 15 minutes and prints the averages to the console.

* 1. **Code Outline**

packagecom.nmit.spark.sparkwindowedstreaming

importorg.apache.spark.sql.functions.\_

importorg.apache.spark.sql.SparkSession

importorg.apache.spark.sql.functions.{window, col}

importorg.apache.spark.sql.types.StructType

objectsparkWindowedStreaming {

def main(args: Array[String]) {

val spark = SparkSession.builder.appName("sparkWindowedStreaming").getOrCreate()

importspark.implicits.\_

spark.conf.set("spark.sql.shuffle.partitions", 5)

valuserSchema = new StructType().

add("Creation\_Time", "double").

add("Station", "string").

add("Rainfall", "float")

val streaming = spark.

readStream.

schema(userSchema).

json("/home/subhrajit/sparkProjects/data/event-data/threeWindows")

valwithEventTime = streaming.selectExpr(

"\*",

"cast(cast(Creation\_Time as double)/1000000000 as timestamp) as event\_time")

// group events by 15 minute time windows and print the averages to the console.

events\_per\_window.awaitTermination()

}

}

**Supporting sparkWindowedStreaming.sbtDependency file**

name := "sparkWindowedStreaming"

version := "1.0"

scalaVersion := "2.11.12"

// scalaVersion := "2.11.8"

libraryDependencies ++= Seq(

"org.apache.spark" %% "spark-core" % "2.3.0",

"org.apache.spark" %% "spark-streaming" % "2.3.0",

"org.apache.spark" %% "spark-sql" % "2.3.0"

)

// "org.apache.spark" % "spark-sql\_2.11" % "2.1.0"

* 1. **Scala program to generate Events**

packagecom.nmit.spark.generateEvents

importjava.io.PrintWriter

importjava.text.SimpleDateFormat

objectgenerateEvents {

defgenerateNextEvent(currentDateTime: Double, stationID: String, rainFall:Float) = {

"{" +

"\"Creation\_Time\"" +

": " +

currentDateTime +

"," +

"\"Station\"" +

": " +

" \"" +

stationID +

"\"" +

"," +

"\"Rainfall\"" +

": " +

rainFall +

"}"

}

defgenCurrentDateTime(currentUnixTime: Double) = {

valnanoToMilli = 1000000.0

valdf = new SimpleDateFormat("yyyy-MM-ddHH:mm:ss")

df.format(currentUnixTime / nanoToMilli)

}

defwriteFile(currentTimeInMin: Double, stationID: String, rainFall: Float) {

// System.currentTimeMillis()

// res8: Long = 1536335039694

valnanoSec = 1000000000.0

valunixStartTime = 1536335039694.0 \* 1000000.0

valcurrentUnixTime = unixStartTime + currentTimeInMin\*60\*nanoSec

valcurrentDateTime = genCurrentDateTime(currentUnixTime)

valformattedCDT = currentDateTime.replaceAll("[ :]", "-")

valfileName = "/home/subhrajit/sparkProjects/data/event-data/threeWindows/" +

stationID +

"\_" +

formattedCDT + ".json"

println(fileName)

val out = new PrintWriter(fileName)

out.println(generateNextEvent(currentUnixTime, stationID, rainFall))

out.close()

}

deforigScenario() {

val r = scala.util.Random

val stationID1 = "Bengaluru-1"

val stationID2 = "Bengaluru-2"

val stationID3 = "Bengaluru-3"

// Generate rainfall data for the first 15 mins for three stations.

// Heavy rainfall recorded.

varcurrentTimeInMin = 15

// varcurrRainFall = r.nextFloat \* 100

varcurrRainFall = 100

writeFile(currentTimeInMin, stationID1, currRainFall)

// varcurrRainFall = r.nextFloat \* 100

currRainFall = 80

writeFile(currentTimeInMin, stationID2, currRainFall)

// varcurrRainFall = r.nextFloat \* 100

currRainFall = 120

writeFile(currentTimeInMin, stationID3, currRainFall)

Thread.sleep(10000)

// Generate rainfall data for the second 15 Mins for three stations.

// Light rainfall recorded.

// One of the events arrives out of order.

currentTimeInMin = 30

// currRainFall = r.nextFloat \* 10

currRainFall = 20

writeFile(currentTimeInMin, stationID1, currRainFall)

// out of order event.

// varcurrRainFalloo = r.nextFloat \* 10

varcurrRainFalloo = 10

varcurrentTimeInMinoo = currentTimeInMin

varstationIDoo = stationID2

// writeFile(currentTimeInMin, stationID2, currRainFall)

// currRainFall = r.nextFloat \* 10

currRainFall = 30

writeFile(currentTimeInMin, stationID3, currRainFall)

Thread.sleep(10000)

// Generate rainfall data for the third 15 mins for three stations.

// Again heavy rainfall.

// The OO event arrives now.

currentTimeInMin = 45

// currRainFall = r.nextFloat \* 100

currRainFall = 100

writeFile(currentTimeInMin, stationID1, currRainFall)

// currRainFall = r.nextFloat \* 100

currRainFall = 80

writeFile(currentTimeInMin, stationID2, currRainFall)

// currRainFall = r.nextFloat \* 100

currRainFall = 120

writeFile(currentTimeInMin, stationID3, currRainFall)

// out of order event.

writeFile(currentTimeInMinoo, stationIDoo, currRainFalloo)

}

defthreeWindows() {

val r = scala.util.Random

val stationID1 = "Bengaluru-1"

val stationID2 = "Bengaluru-2"

val stationID3 = "Bengaluru-3"

// Generate rainfall data for the first 15 mins for three stations.

// Heavy rainfall recorded.

varcurrentTimeInMin = 15

varcurrRainFall = 100

writeFile(currentTimeInMin, stationID1, currRainFall)

currRainFall = 80

writeFile(currentTimeInMin, stationID2, currRainFall)

currRainFall = 120

writeFile(currentTimeInMin, stationID3, currRainFall)

Thread.sleep(10000)

// Generate rainfall data for the second 15 Mins for three stations.

// Light rainfall recorded.

currentTimeInMin = 30

currRainFall = 20

writeFile(currentTimeInMin, stationID1, currRainFall)

currRainFall = 10

writeFile(currentTimeInMin, stationID2, currRainFall)

currRainFall = 30

writeFile(currentTimeInMin, stationID3, currRainFall)

Thread.sleep(10000)

// Generate rainfall data for the third 15 mins for three stations.

// Again heavy rainfall.

currentTimeInMin = 45

currRainFall = 100

writeFile(currentTimeInMin, stationID1, currRainFall)

currRainFall = 80

writeFile(currentTimeInMin, stationID2, currRainFall)

currRainFall = 120

writeFile(currentTimeInMin, stationID3, currRainFall)

}

def main(args: Array[String]) {

threeWindows()

}

}