# PRACTICAL NO: 1

**Aim: Implementation of Analytical queries (Rank and dense Rank).**

>>> CREATE TABLE anchal\_119 ( rollno NUMBER,name VARCHAR2(10),subject VARCHAR2(7), marks NUMBER);

>>>INSERT INTO anchal\_119 VALUES (119, 'Anchal', 'ADMS', 90);

INSERT INTO anchal\_119 VALUES (11, 'Anurag', 'DS', 85);

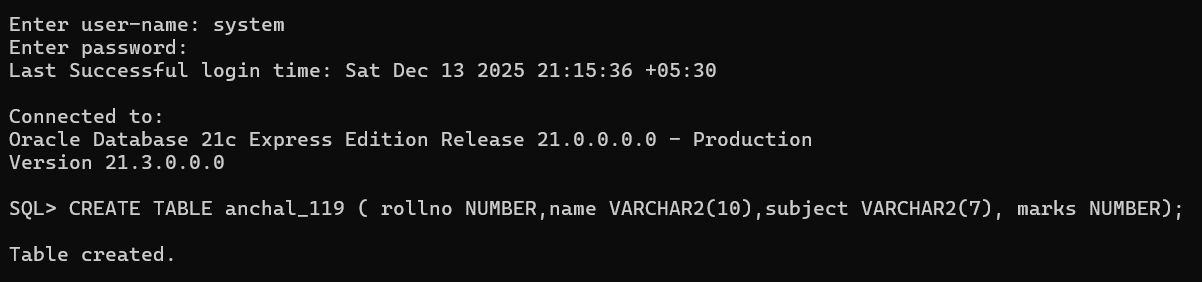
INSERT INTO anchal\_119 VALUES (12, 'Sonam', 'MFCS', 88);

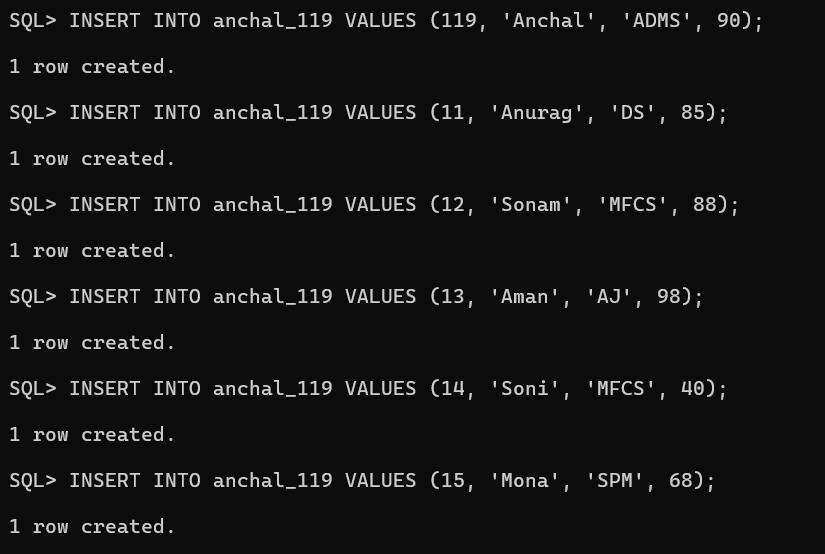
INSERT INTO anchal\_119 VALUES (13, 'Aman', 'AJ', 98);

INSERT INTO anchal\_119 VALUES (14, 'Soni', 'MFCS', 40);

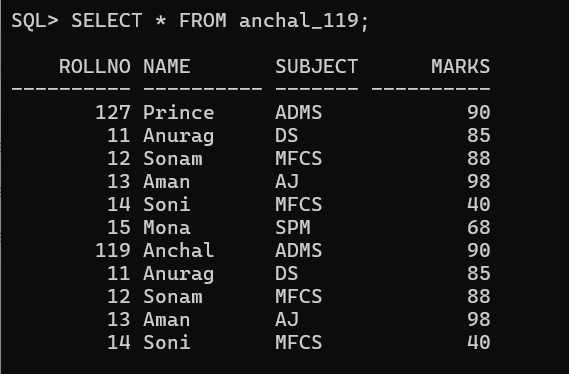
INSERT INTO anchal\_119 VALUES (15, 'Mona', 'SPM', 68);

**Output:**

****



>>> SELECT \* FROM anchal\_119;



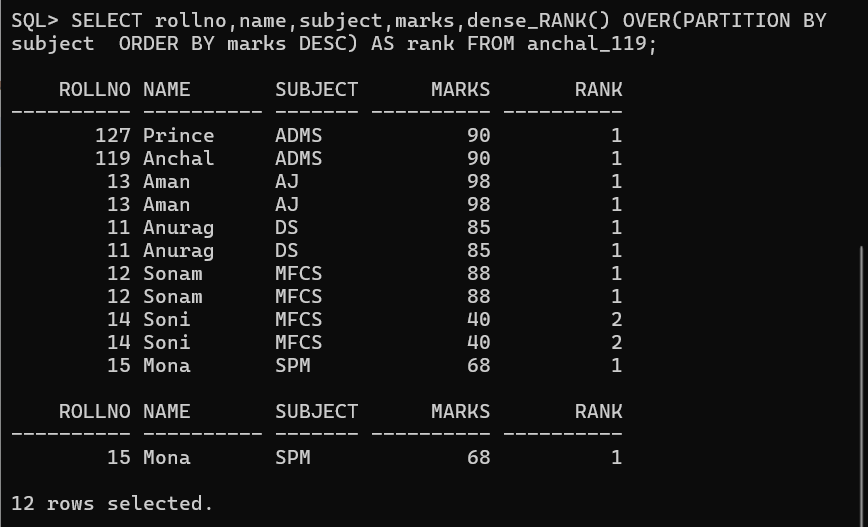
1. **Rank**

>>> SELECT rollno,name,marks,subject,RANK() OVER(PARTITION BY subject ORDER BY marks) AS rank FROM anchal\_119;



1. **Dense\_Rank**

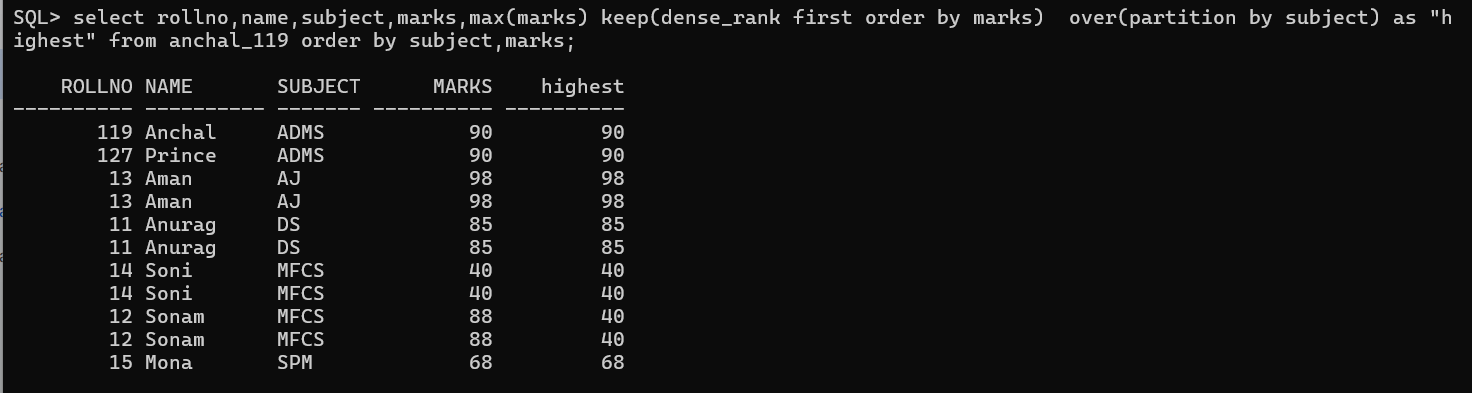
>>> SELECT rollno,name,subject,marks,dense\_RANK() OVER(PARTITION BY subject ORDER BY marks DESC) AS rank FROM anchal\_119;



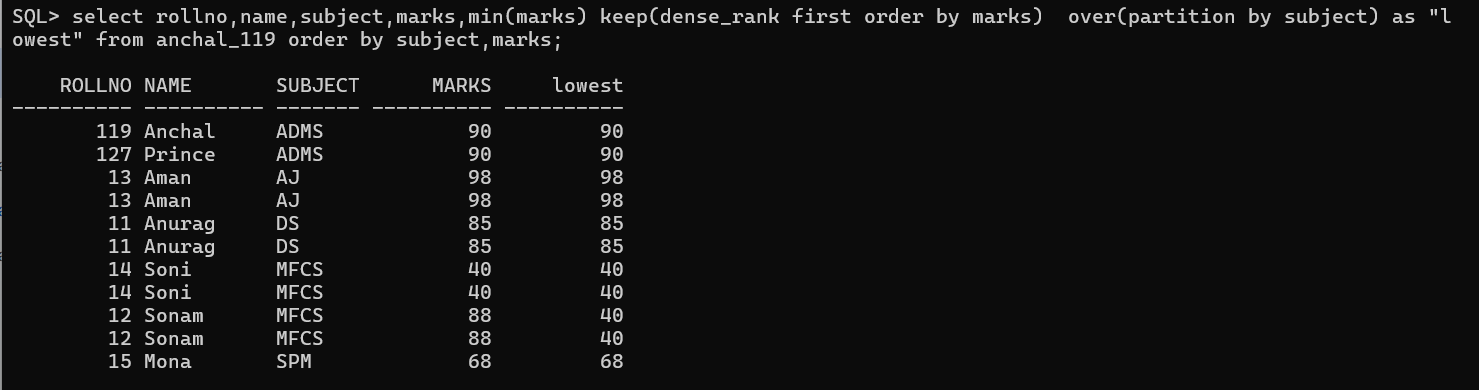
# PRACTICAL NO: 2

**Aim: Implementation of Analytical queries (first, last)**

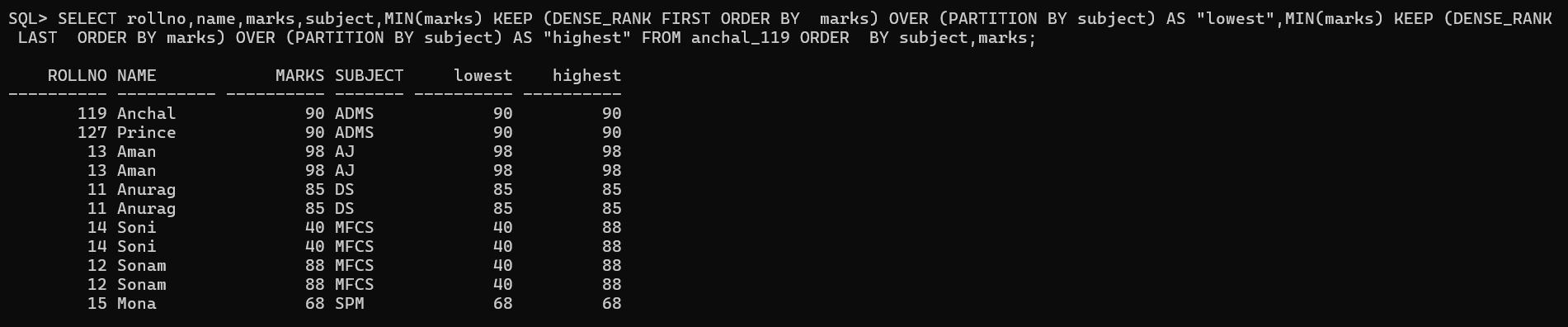
>>> select rollno,name,subject,marks,max(marks) keep(dense\_rank first order by marks) over(partition by subject) as "highest" from anchal\_119order by subject,marks;



>>> select rollno,name,subject,marks,min(marks) keep(dense\_rank first order by marks) over(partition by subject) as "lowest" from anchal\_119order by subject,marks;



>>> SELECT rollno,name,marks,subject,MIN(marks) KEEP (DENSE\_RANK FIRST ORDER BY marks) OVER (PARTITION BY subject) AS "lowest",MIN(marks) KEEP (DENSE\_RANK LAST ORDER BY marks) OVER (PARTITION BY subject) AS "highest" FROM anchal\_119ORDER BY subject,marks;



>>>CREATE TABLE anchal\_emp (empid INT, empname VARCHAR(10), dept VARCHAR(20), salary INT);

>>> INSERT INTO anchal\_emp VALUES (1, 'Ram', 'marketing', 10000); INSERT INTO anchal\_emp VALUES (2, 'anchal', 'marketing', 103000);

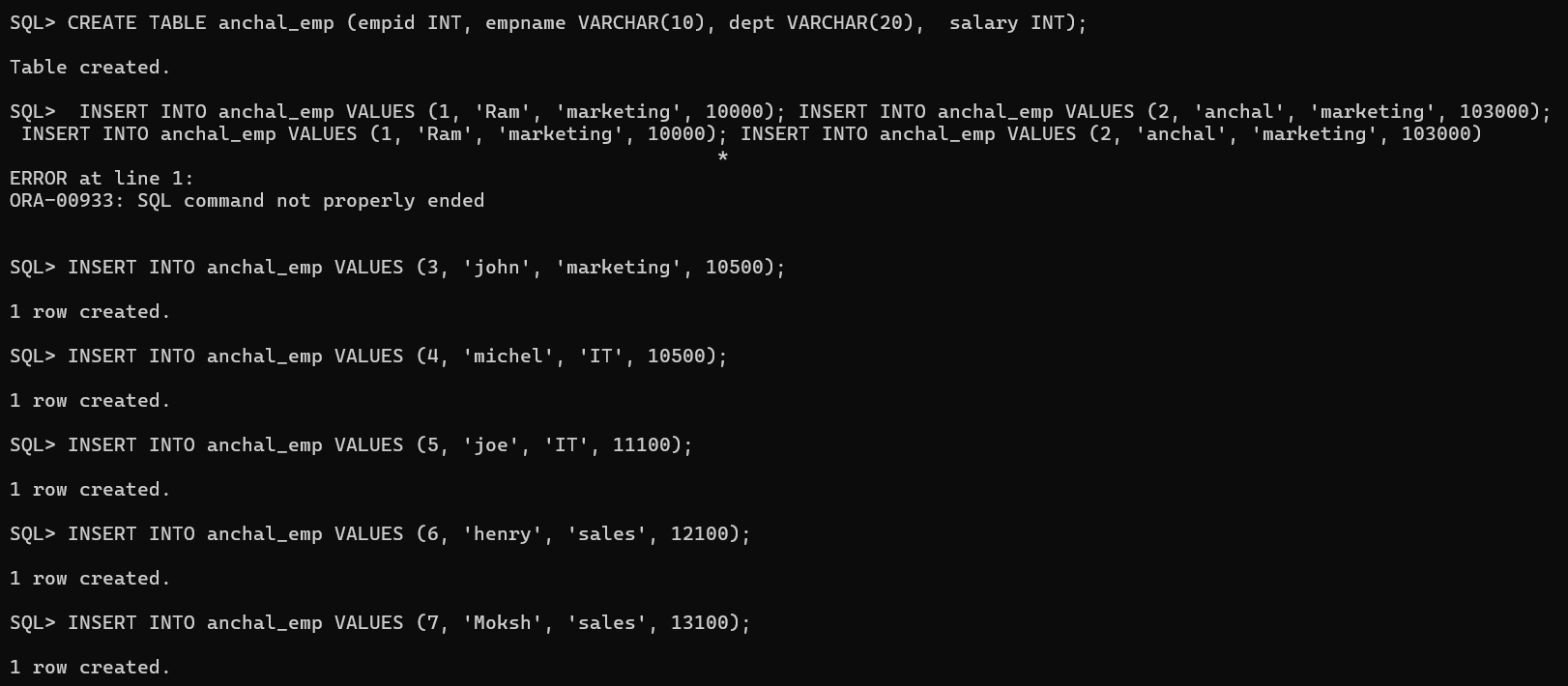
INSERT INTO anchal\_emp VALUES (3, 'john', 'marketing', 10500);

INSERT INTO anchal\_emp VALUES (4, 'michel', 'IT', 10500);

INSERT INTO anchal\_emp VALUES (5, 'joe', 'IT', 11100);

INSERT INTO anchal\_emp VALUES (6, 'henry', 'sales', 12100);

INSERT INTO anchal\_emp VALUES (7, 'Moksh', 'sales', 13100);

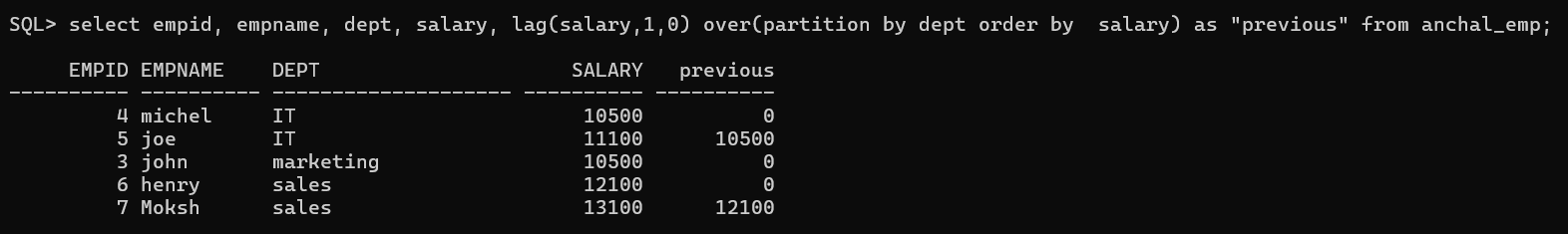


# PRACTICAL NO: 3

**Aim: Implementation of analytical queries (lead and lag)**

1. **LAG**

>>> select empid, empname, dept, salary, lag(salary,1,0) over(partition by dept order by salary) as "previous" from anchal\_emp;



1. **LEAD**

>>> select empid, empname, dept, salary, lead(salary,1,0) over(partition by dept order by salary) as "previous" from anchal\_emp;



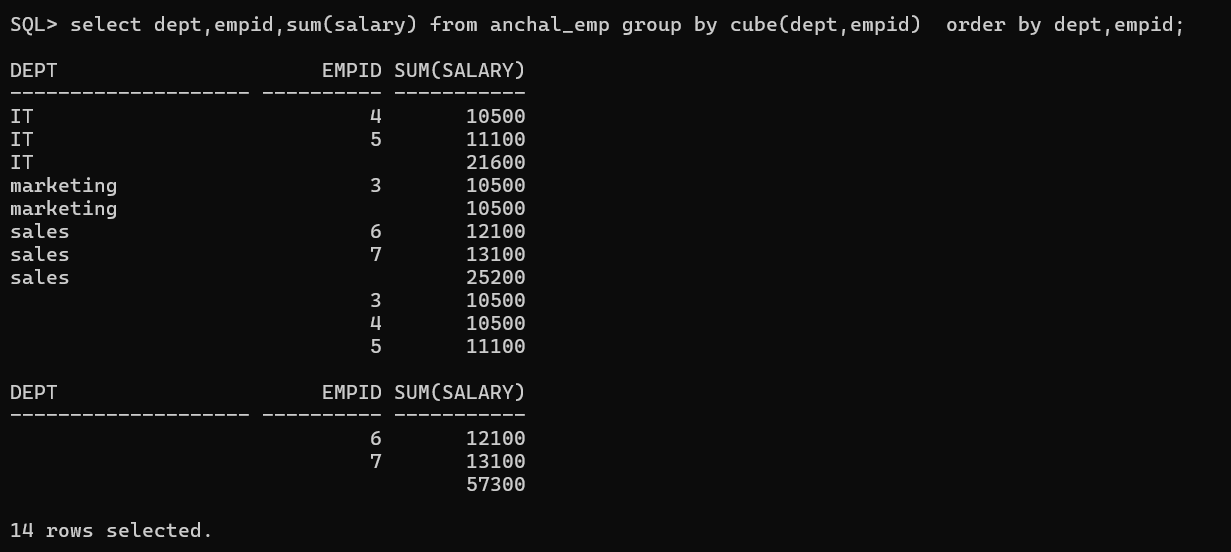
# PRACTICAL NO: 4

**Implementation of analytical queries (Rollup and Cube)**

**1.Cube:**

* CUBE generates **all possible combinations of subtotals** for the columns you specify.
* Rows where **dept is NULL but empid is NOT NULL** come **only from CUBE**, not ROLLUP.

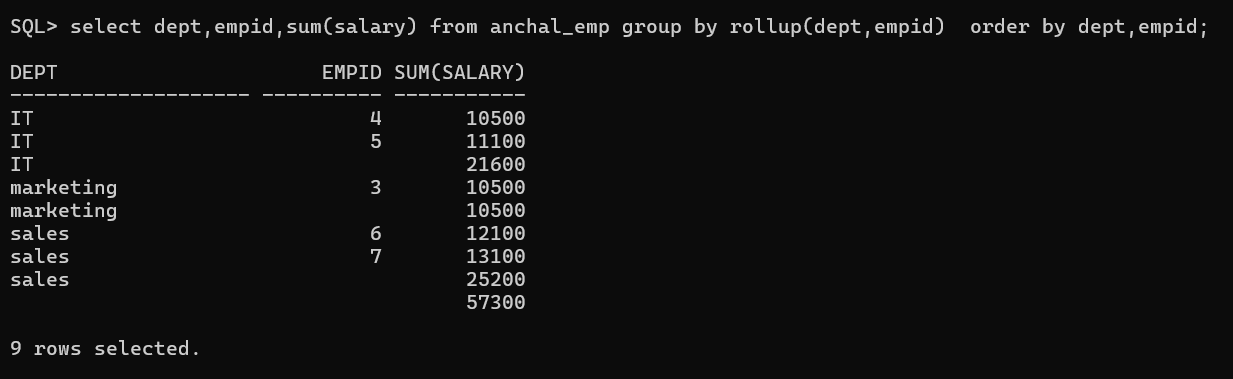
**S**QL> select dept,empid,sum(salary) from anchal\_emp group by cube(dept,empid) order by dept,empid;



**2. Roll up**

* ROLLUP is used here to generate **subtotals and a grand total** in the same result set without writing multiple queries using union.

SQL> select dept,empid,sum(salary) from anchal\_emp group by rollup(dept,empid) order by dept,empid;



# PRACTICAL NO: 5

**Aim: Implementation of Aggregate Window Function (Sum, Avg Count, Min, Max)**

SELECT subject, marks,

SUM(marks) OVER( PARTITION BY subject ORDER BY marks) AS "Total",

AVG(marks) OVER( PARTITION BY subject ORDER BY marks) AS "Average",

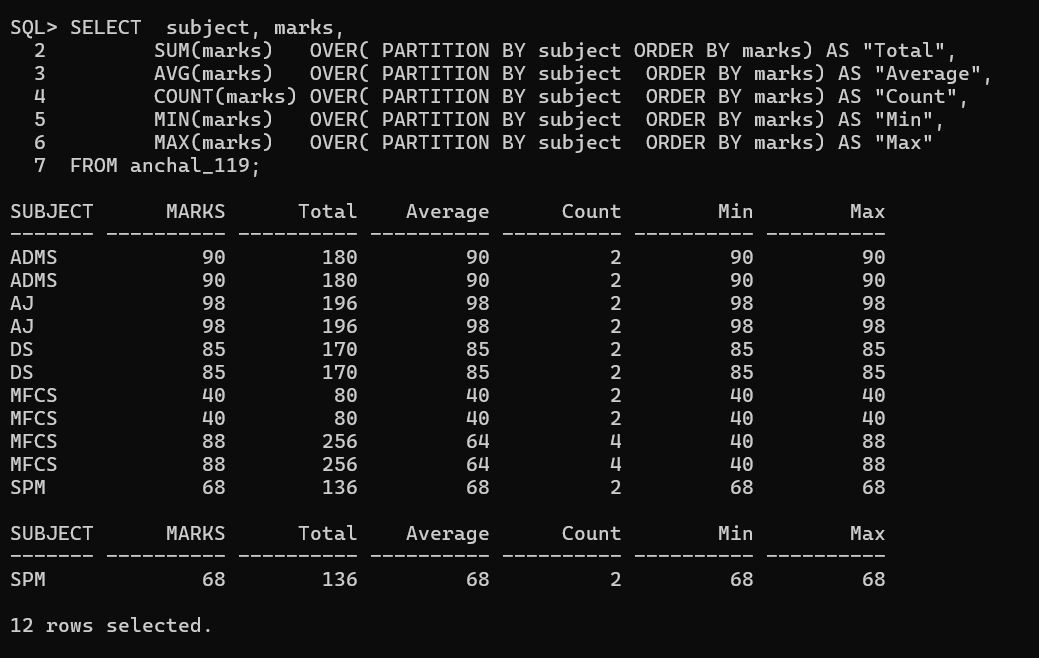
COUNT(marks) OVER( PARTITION BY subject ORDER BY marks) AS "Count",

MIN(marks) OVER( PARTITION BY subject ORDER BY marks) AS "Min",

MAX(marks) OVER( PARTITION BY subject ORDER BY marks) AS "Max"

FROM anchal\_119;

**Output:**



# PRACTICAL NO: 6

**Aim: Implementation of Window Function (ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)**

SELECT subject, marks,

SUM(marks) OVER (

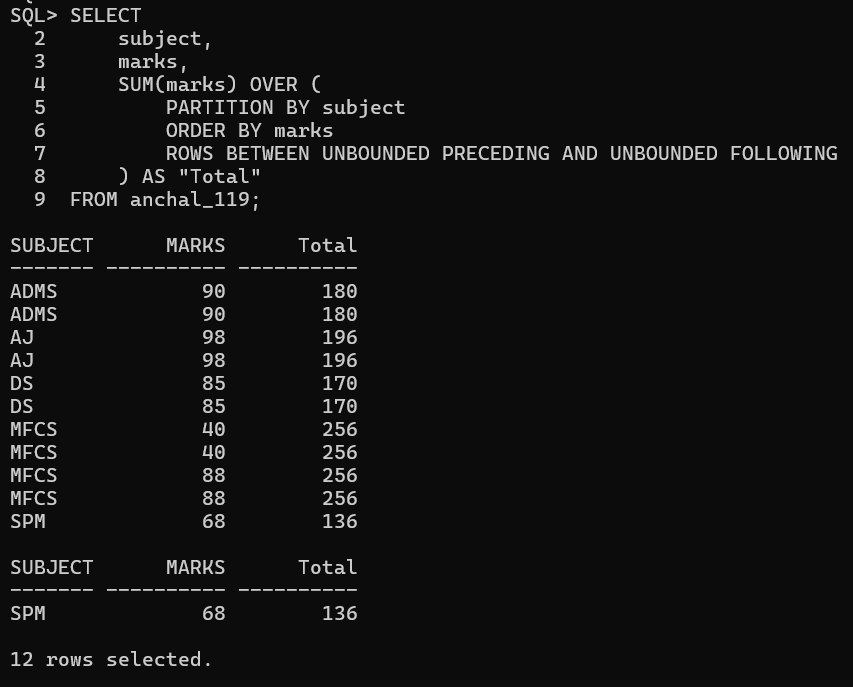
PARTITION BY subject

ORDER BY marks

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

) AS "Total"

FROM anchal\_119;



**UNBOUNDED PRECEDING** → starts from the **first row of the subject**

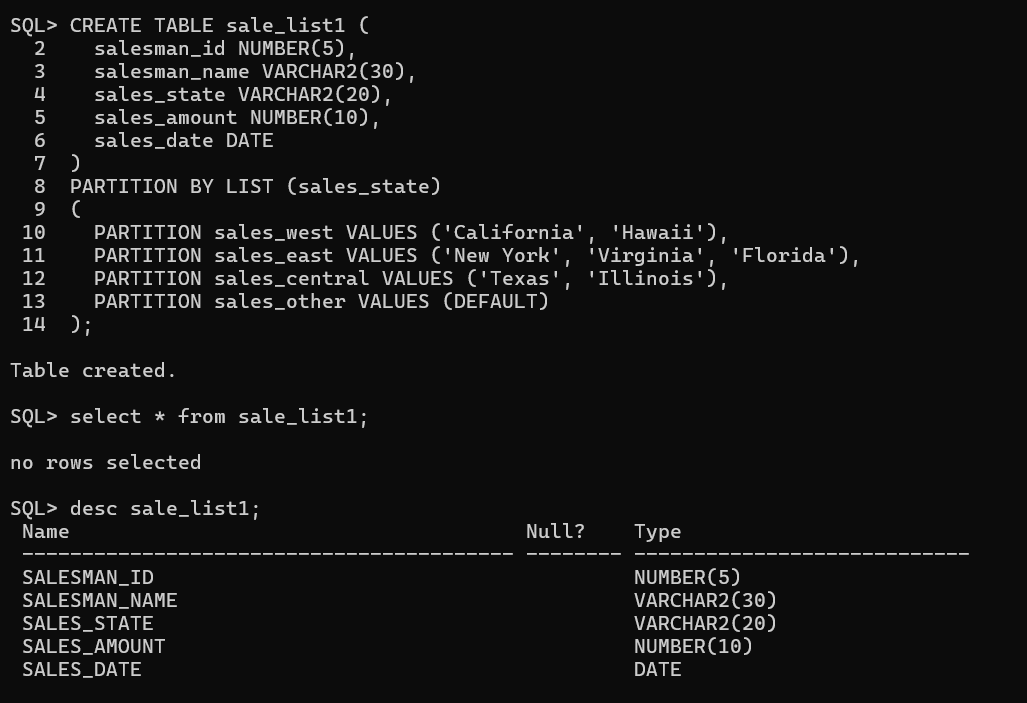
**UNBOUNDED FOLLOWING** → goes till the **last row of the subject**

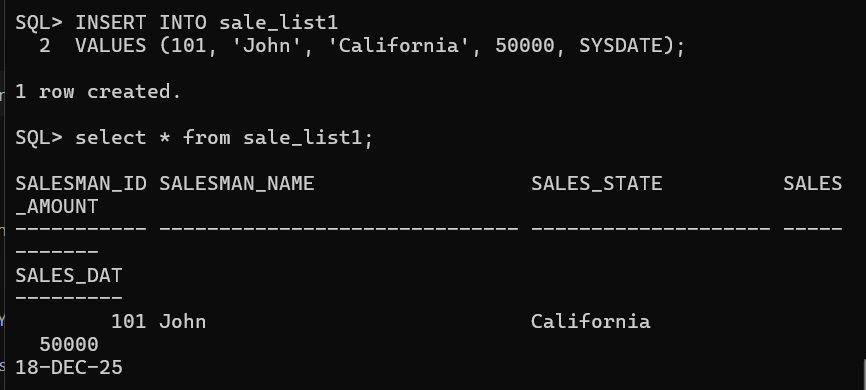
Use **ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING** when you want **partition-level aggregates repeated on every row**

# PRACTICAL NO: 7

**Aim: Implementation of List partition.**

CREATE TABLE sale\_list1 (  
 salesman\_id NUMBER(5),  
 salesman\_name VARCHAR2(30),  
 sales\_state VARCHAR2(20),  
 sales\_amount NUMBER(10),  
 sales\_date DATE  
)  
PARTITION BY LIST (sales\_state)  
(  
 PARTITION sales\_west VALUES ('California', 'Hawaii'),  
 PARTITION sales\_east VALUES ('New York', 'Virginia', 'Florida'),  
 PARTITION sales\_central VALUES ('Texas', 'Illinois'),  
 PARTITION sales\_other VALUES (DEFAULT)  
);





Explanation of Partitions

1. sales\_west: Stores records where sales\_state is California or Hawaii.  
2. sales\_east: Stores records where sales\_state is New York, Virginia, or Florida.  
3. sales\_central: Stores records where sales\_state is Texas or Illinois.  
4. sales\_other: Stores records with any other sales\_state value.

Advantages of LIST Partitioning

- Improves query performance.  
- Makes data management easier.  
- Useful when column values are fixed and known.  
- Reduces full table scan.

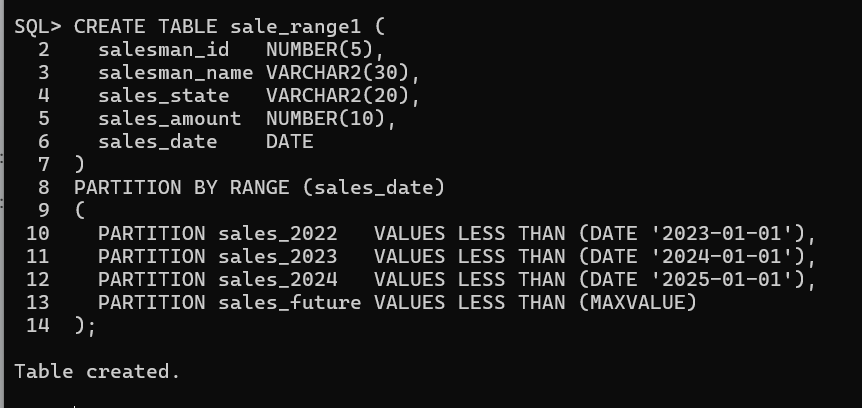
Conclusion

LIST partitioning is ideal when data falls into predefined categories. It helps in organizing data efficiently and enhances performance in large databases.

# PRACTICAL NO: 8

**Aim: Implementation of Range partition**

CREATE TABLE sale\_range1 (  
 salesman\_id NUMBER(5),  
 salesman\_name VARCHAR2(30),  
 sales\_state VARCHAR2(20),  
 sales\_amount NUMBER(10),  
 sales\_date DATE  
)  
PARTITION BY RANGE (sales\_date)  
(  
 PARTITION sales\_2022 VALUES LESS THAN (DATE '2023-01-01'),  
 PARTITION sales\_2023 VALUES LESS THAN (DATE '2024-01-01'),  
 PARTITION sales\_2024 VALUES LESS THAN (DATE '2025-01-01'),  
 PARTITION sales\_future VALUES LESS THAN (MAXVALUE)  
);



RANGE partitioning is ideal for time-based data. It helps in organizing large tables efficiently and improves query performance.Oracle is dividing the table **based on date ranges** of sales\_date.

Each row is placed into a partition depending on **which date range it falls into**.

# PRACTICAL NO: 9

**Aim: Implementation of HASH Partition in Oracle**

HASH partitioning is used when data **cannot be grouped into predefined values** and needs to be **evenly distributed** across partitions.  
Oracle uses a **hashing algorithm** on the partition key to decide in which partition a row should be stored.

CREATE TABLE anchal\_119\_hash(

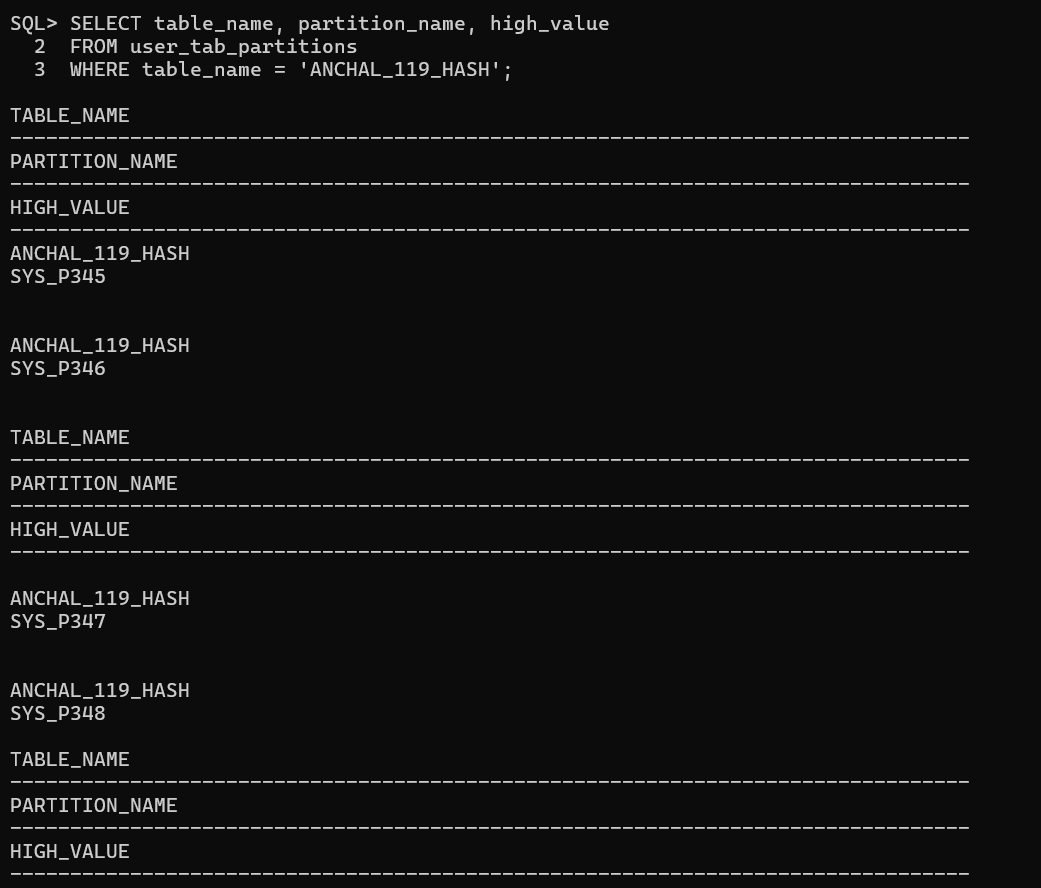
salesman\_id NUMBER(5), salesman\_name VARCHAR2(30), sales\_state VARCHAR2(20),

sales\_amount NUMBER(10), sales\_date DATE

) PARTITION BY HASH (sales\_state)

PARTITIONS 4;





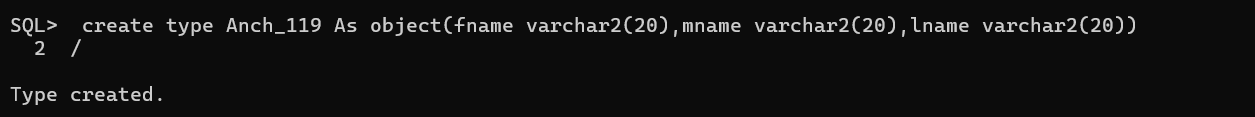
# PRACTICAL NO: 10

**Aim: Implementation of Abstract data Type**

SQL> create type Anch\_119 As object(fname varchar2(20),mname varchar2(20),lname varchar2(20))

2 /

Type created.



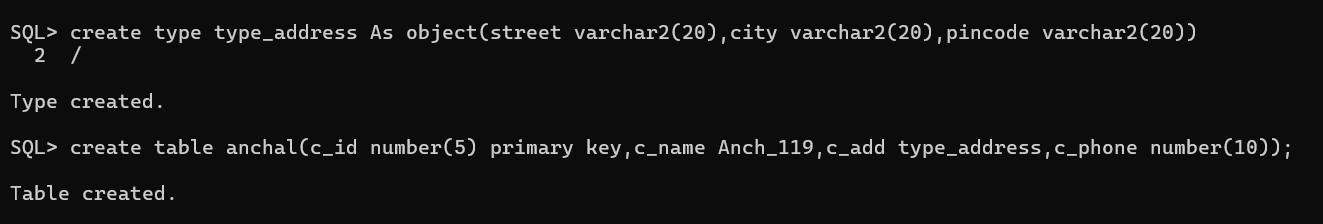
SQL> create type type\_address As object(street varchar2(20),city varchar2(20),pincode varchar2(20))

2 /

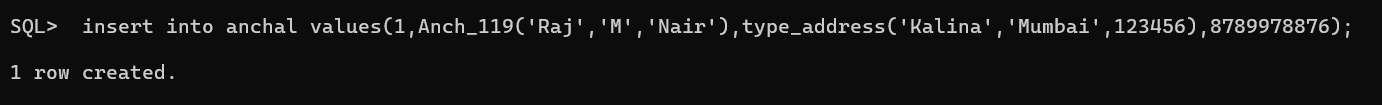
Type created.

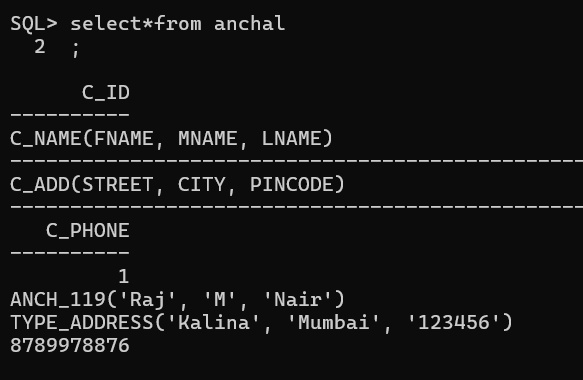
SQL> create table anchal(c\_id number(5) primary key,c\_name Anch\_119,c\_add type\_address,c\_phone number(10));

Table created.



SQL> insert into anchal values(1,Anch\_119('Raj','M','Nair'),type\_address('Kalina','Mumbai',123456),8789978876);





1. **Creating an ADT in SQL\*Plus**

**Step 1: Create the Object Type**

CREATE TYPE student\_type AS OBJECT (  
 student\_id NUMBER(5),  
 student\_name VARCHAR2(30),  
 student\_age NUMBER(3)  
);  
/

**Step 2: Create a Table of This Type**

CREATE TABLE student\_table OF student\_type;

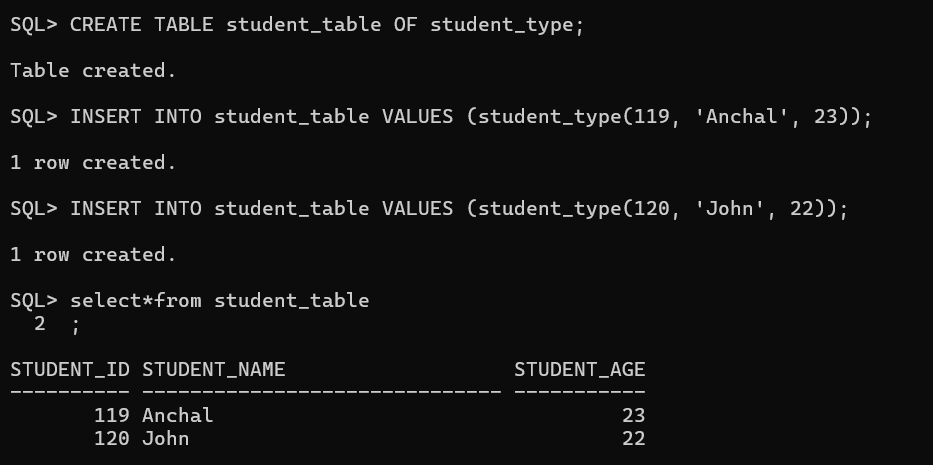
**Step 3: Insert Records**

INSERT INTO student\_table VALUES (student\_type(119, 'Anchal', 23));

INSERT INTO student\_table VALUES (student\_type(120, 'John', 22));

**Step 4: View Records**

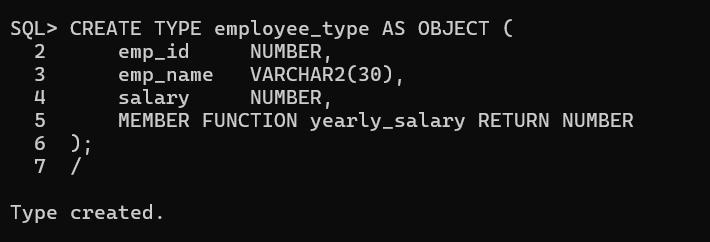
SELECT \* FROM student\_table;



1. **ADT with Methods (Function Example)**

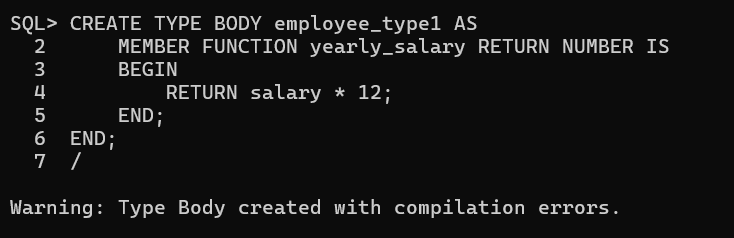
**Step 1: Create an ADT with a Member Function**

CREATE TYPE employee\_type AS OBJECT (  
 emp\_id NUMBER,  
 emp\_name VARCHAR2(30),  
 salary NUMBER,  
 MEMBER FUNCTION yearly\_salary RETURN NUMBER  
);  
/



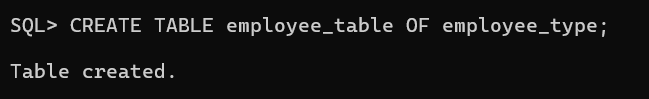
**Step 2: Create the Type Body**

CREATE TYPE BODY employee\_type1 AS  
 MEMBER FUNCTION yearly\_salary RETURN NUMBER IS  
 BEGIN  
 RETURN salary \* 12;  
 END;  
END;  
/



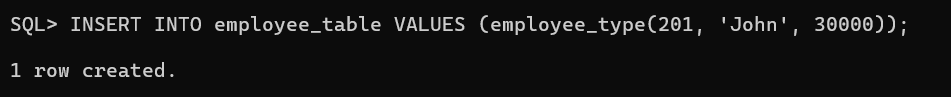
**Step 3: Create Table**

CREATE TABLE employee\_table OF employee\_type;



**Step 4: Insert Data**

INSERT INTO employee\_table VALUES (employee\_type(201, 'John', 30000));



**Step 5: Execute Function**

SELECT emp\_name, salary, yearly\_salary() FROM employee\_table;

**5. Conclusion**

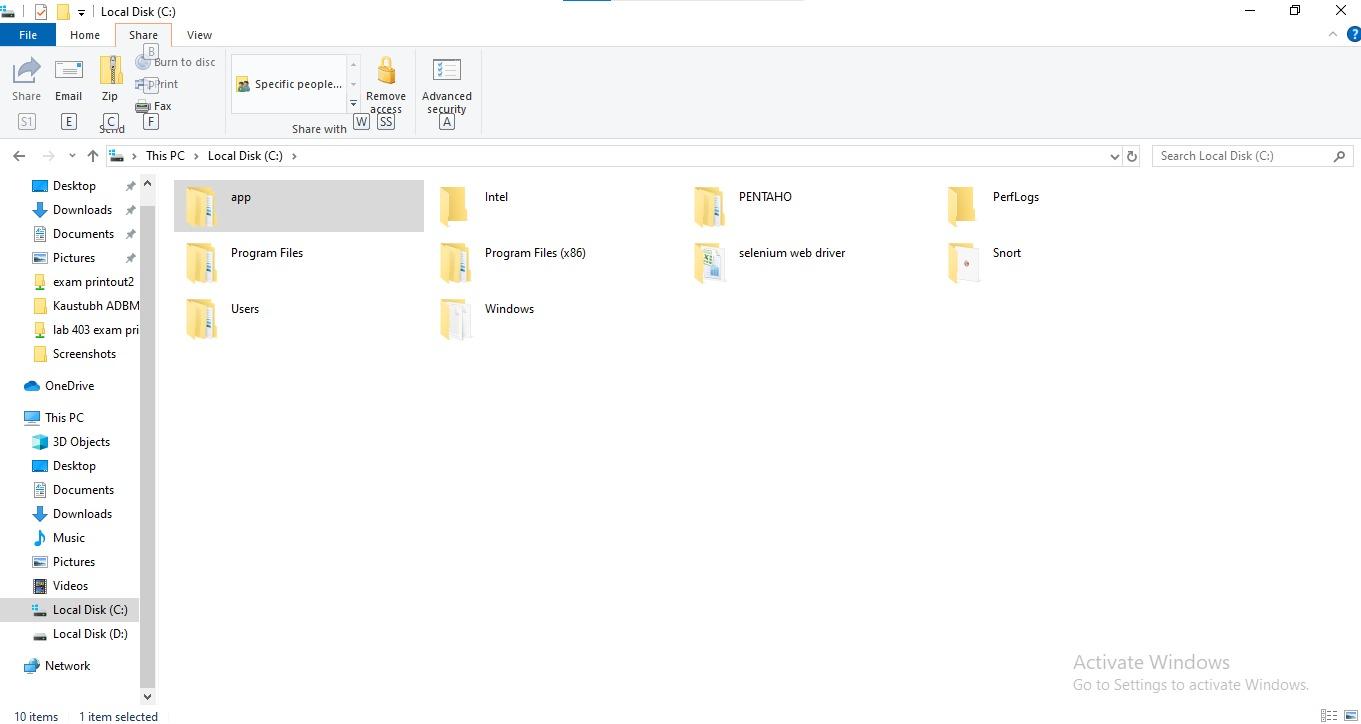
In SQL\*Plus, ADTs provide an object-oriented approach to storing and working with complex data types. They help improve modularity, reusability, and database design structure.

# PRACTICAL NO: 11

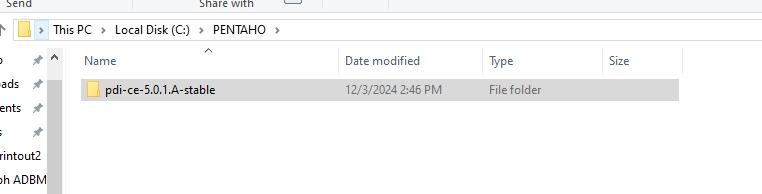
**Aim: Implementation of Input Functions (Excel Input, Table Input) using Pentaho.**

**AIM : ETL through Pentaho: ETL Transformation with Pentaho.**

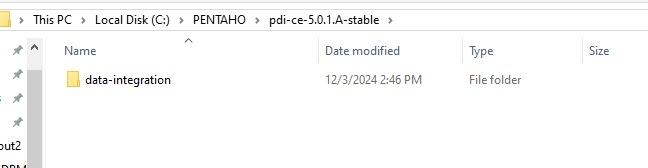
**Step 1:In C Drive -> go to the “PENTAHO” folder**



**Step 2: Click on the pdi folder**

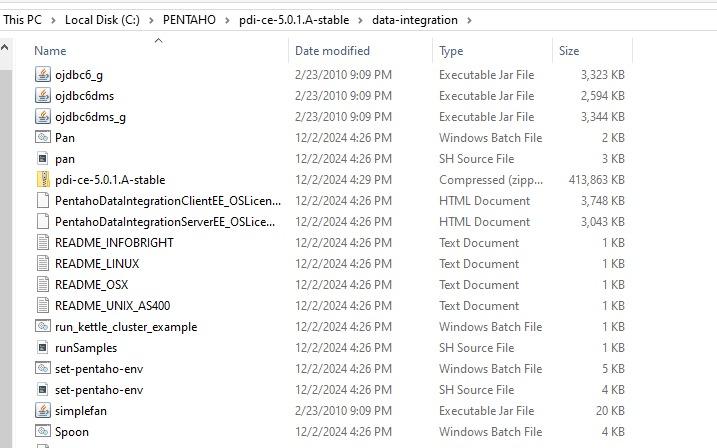


**Step 3: Inside the folder click on data-integration folder**



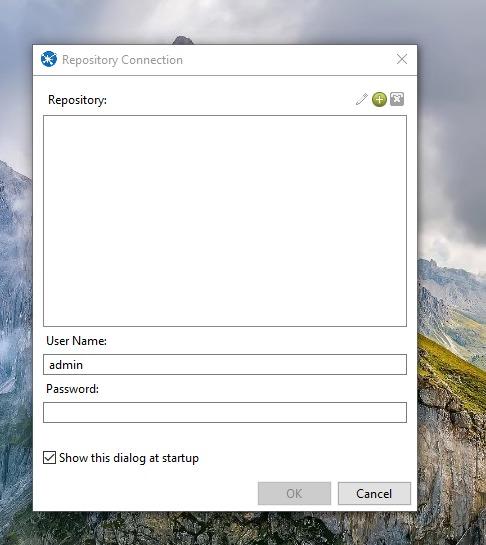
**Step 4:In data integration extract all jar files from the lib folder**

**Step 5: Set system variable for java updated version java 1.8**

**Step 6: After Extracting all the jar files click on “spoon”**



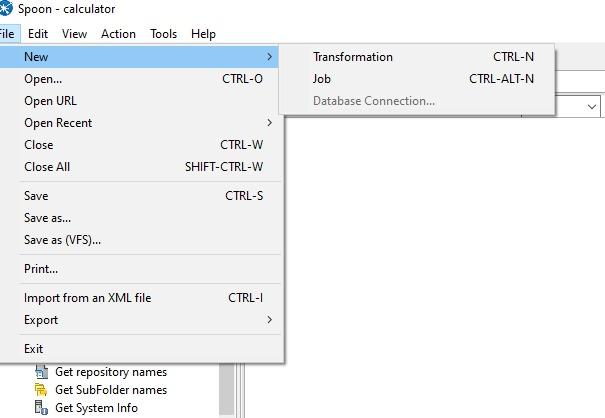
**Step 7: This window will appear click on the “cancel”**



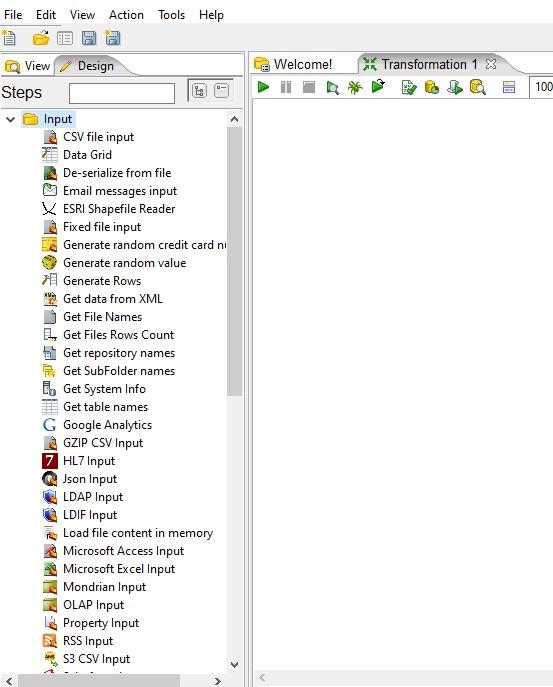
**Step 8: “PENTAHO” will be started**

**Step 9: Run the Pentaho Data Integration tools (Spoon).**

**Step 10: Pull down the File menu and select the New menu item followed by Transformation.**

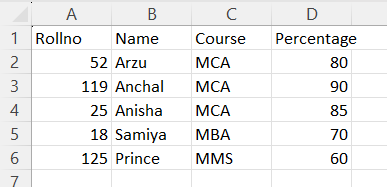


**Step 11: Drag “Microsoft Excel Input” form the “Input” folder and drag it on main window of Pentaho**

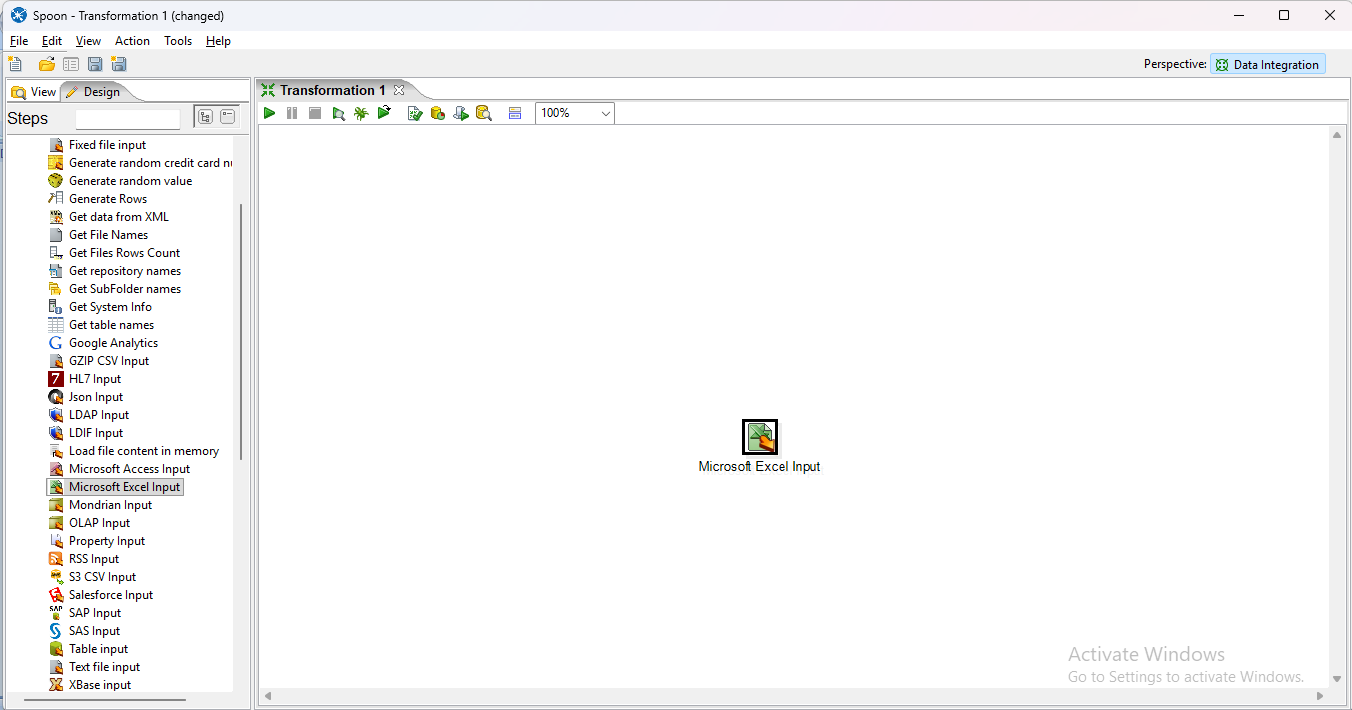


**a) Excel Input:**

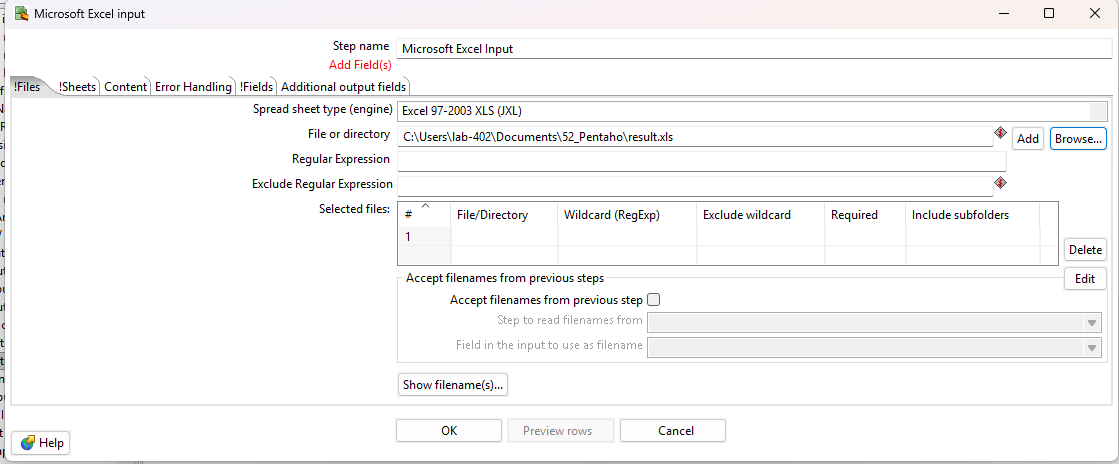
**(i) Create an Excel file with .xls extension.**

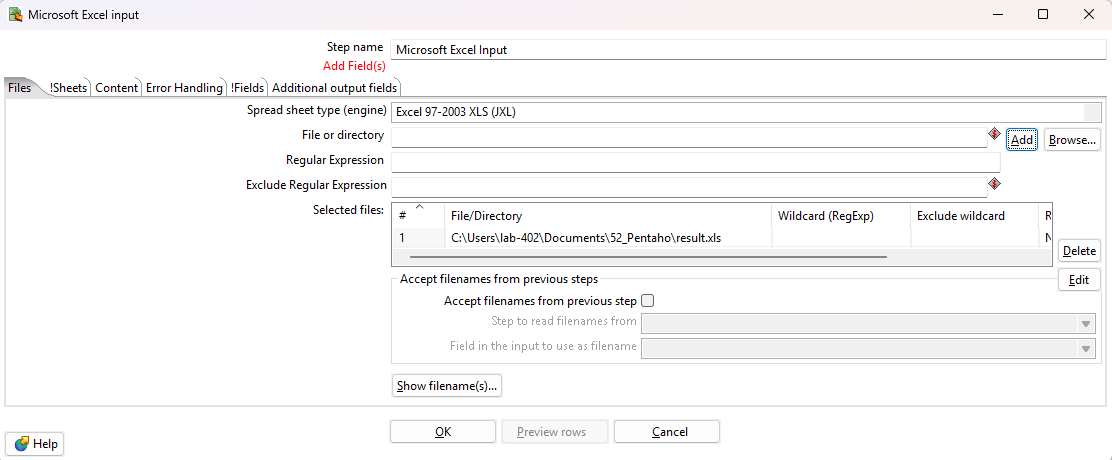


**(ii) Open up the Input folder and drag and drop Microsoft Excel input.**

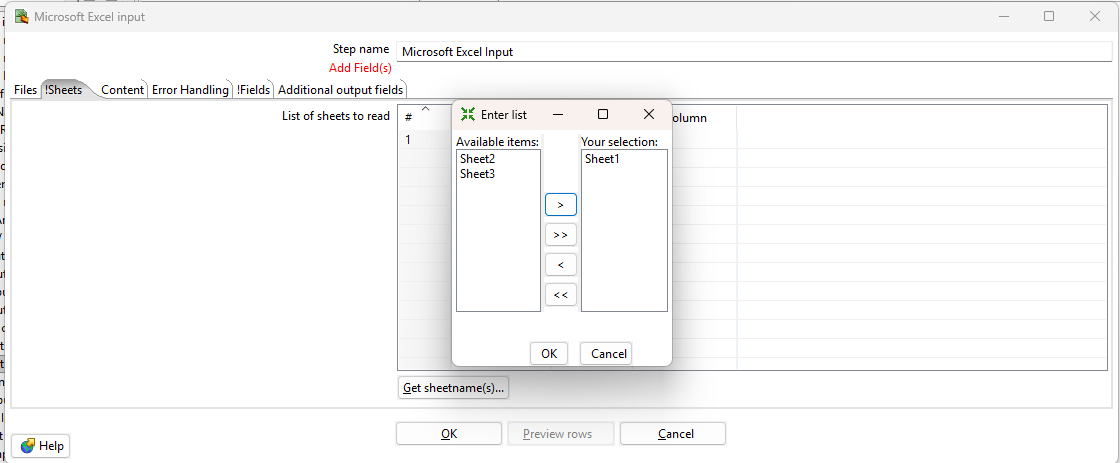


**(iii) Double-click on the Microsoft Excel Input step to view its properties. Click on the Browse button next to the Filename field and navigate to the folder with the Excel files. Select the Excel file as shown below and then click the Open button.**

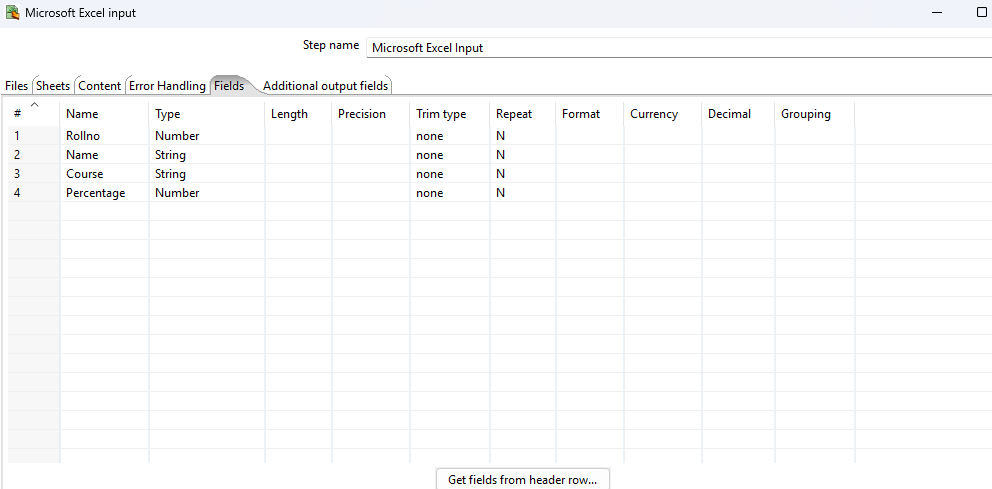


**(iv) Click on the ADD button to add an excel file .**

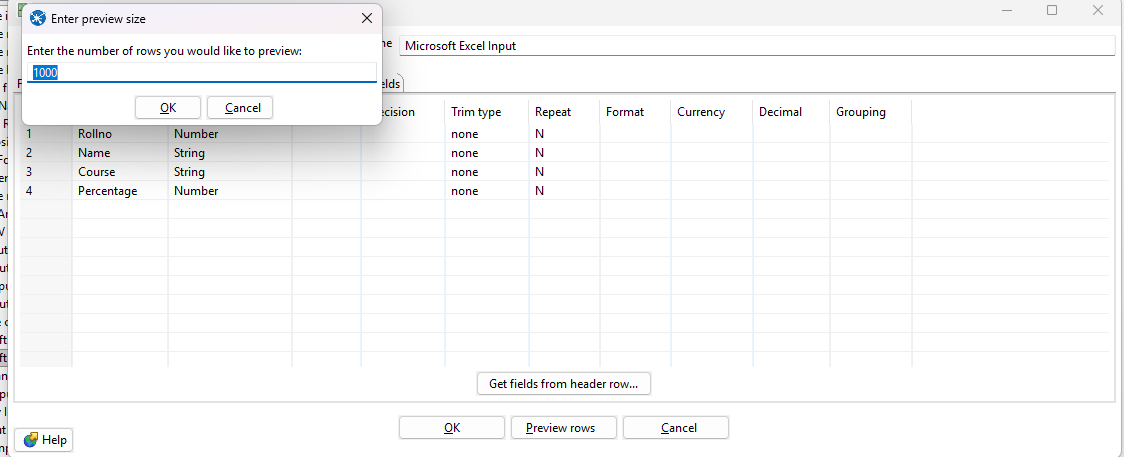
**(v) Navigate to the Sheet tab to select the sheet which you want to select. And select Sheetname(s) Button. The following form will appear then select sheet and select (>)button to add sheet.**

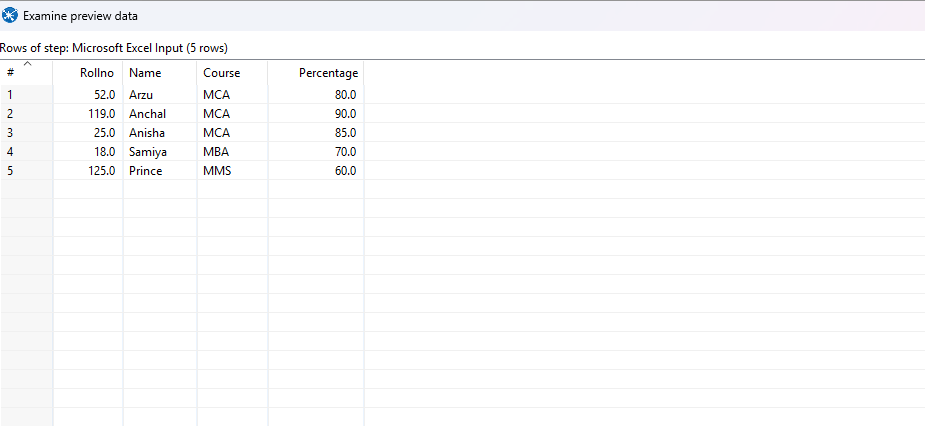


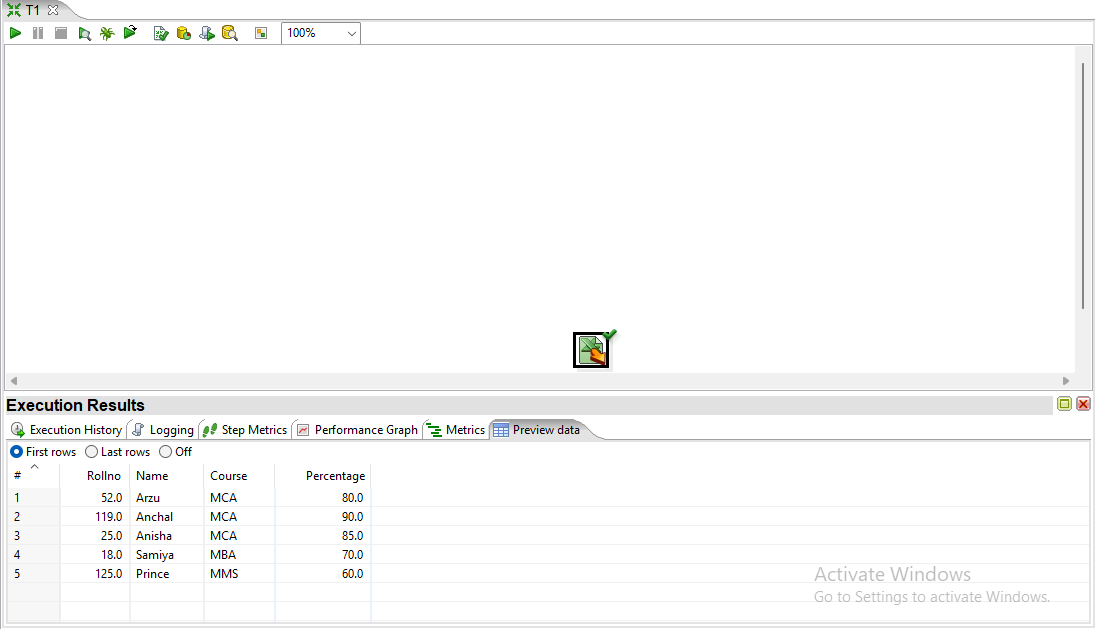
**(vi) Navigate to the Fields tab to assert the header name. Click on “Get Fields from header row” button to get column list .**



**(vii) Click on “Preview Row” button to select top 1000 row and click on OK.**

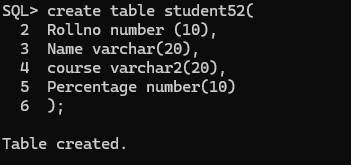


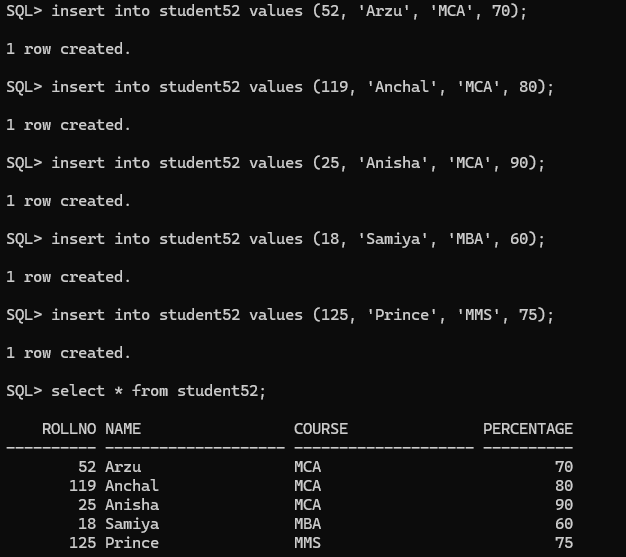


**(viii) Close the preview and run the Microsoft Excel Input.**

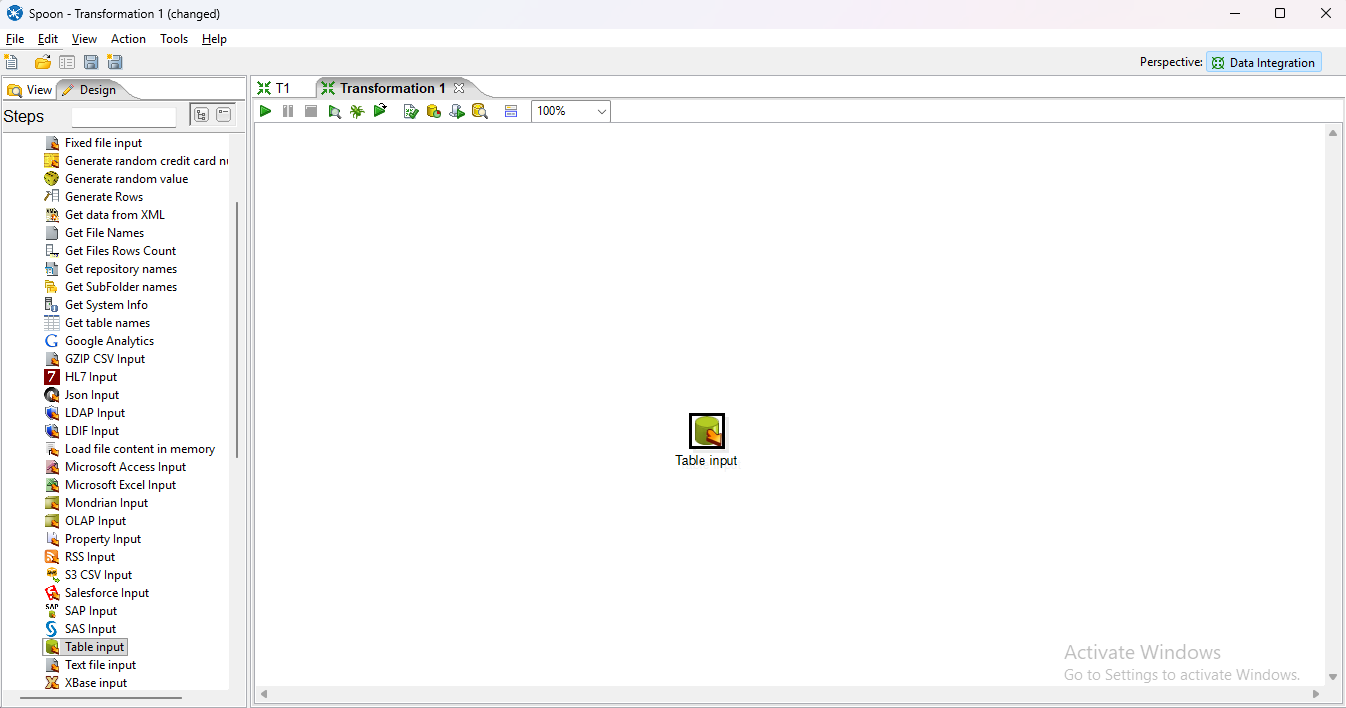
**b) Table Input:**

**(i) Create a SQL Table.**

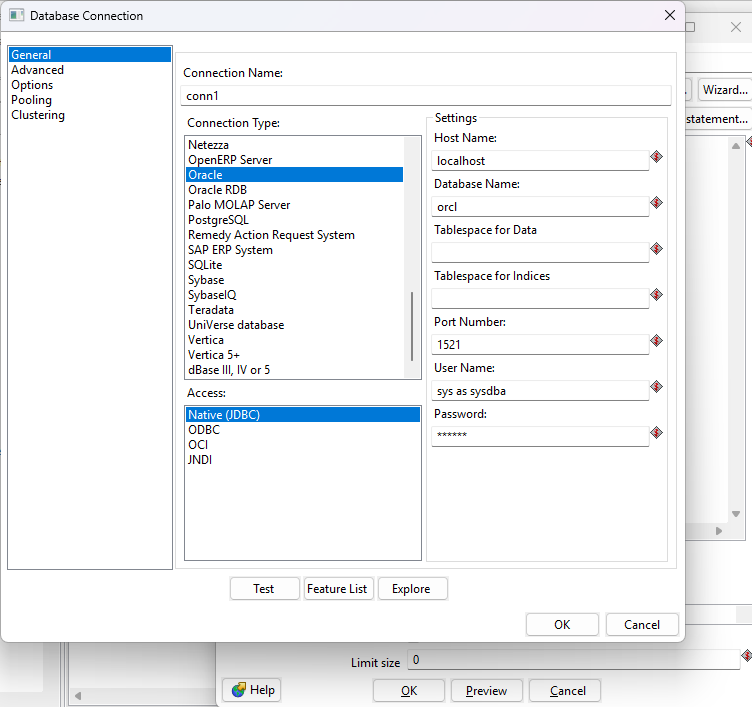
****

****

**(ii) Drag & drop Table input.**

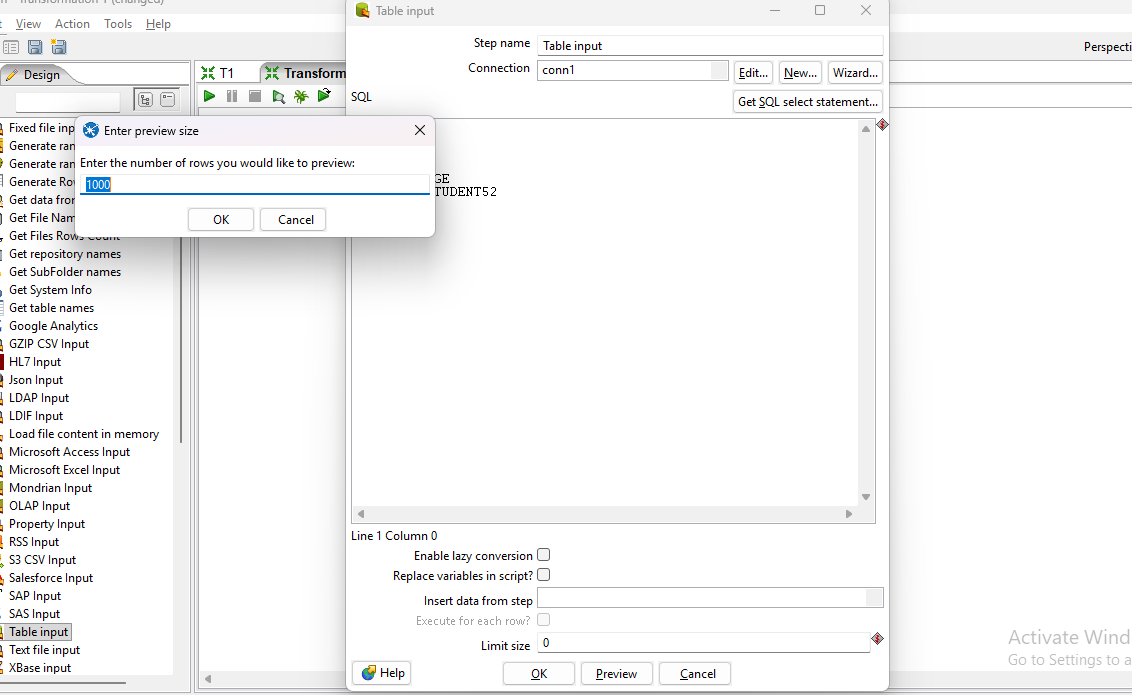
****

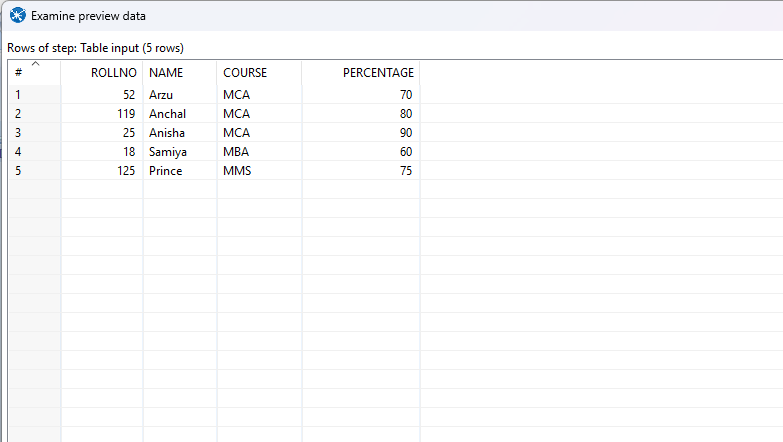
**(iii) Create a new connection & connect it with SQL.**

****

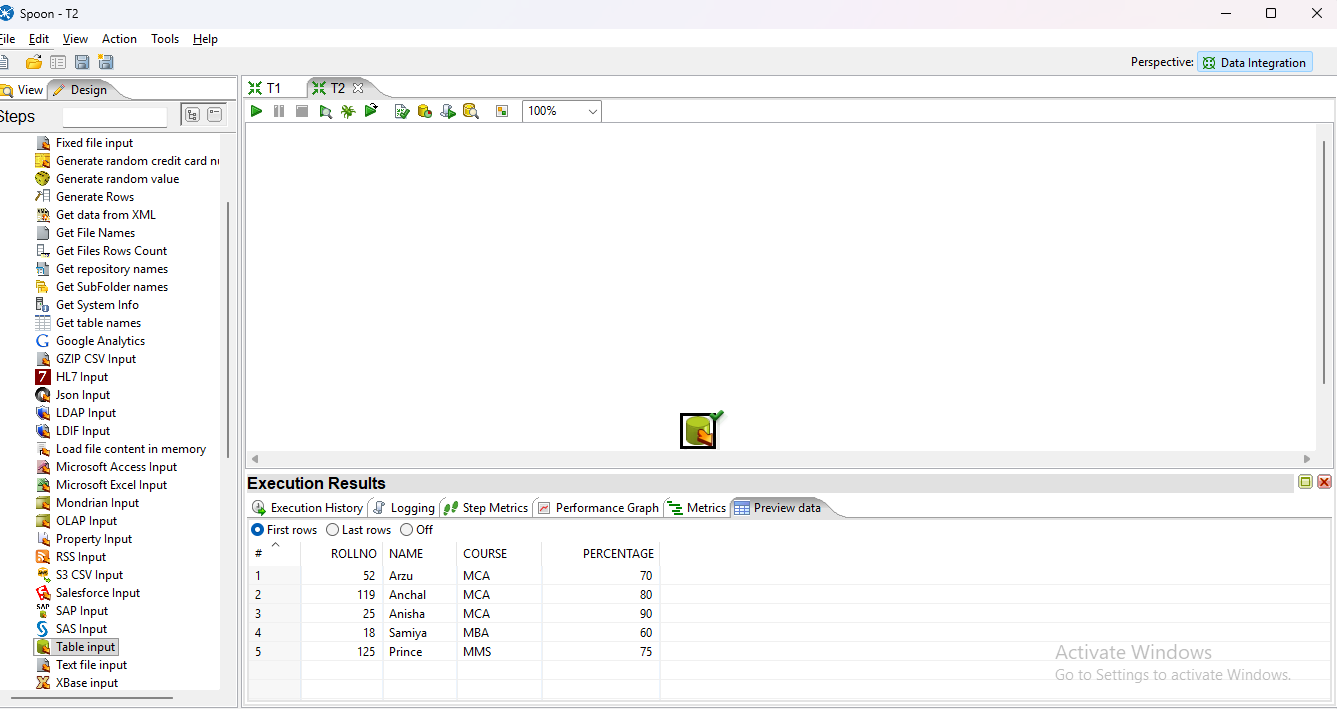
**(iv) Click on Get Select Statement & preview the table.**

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

****



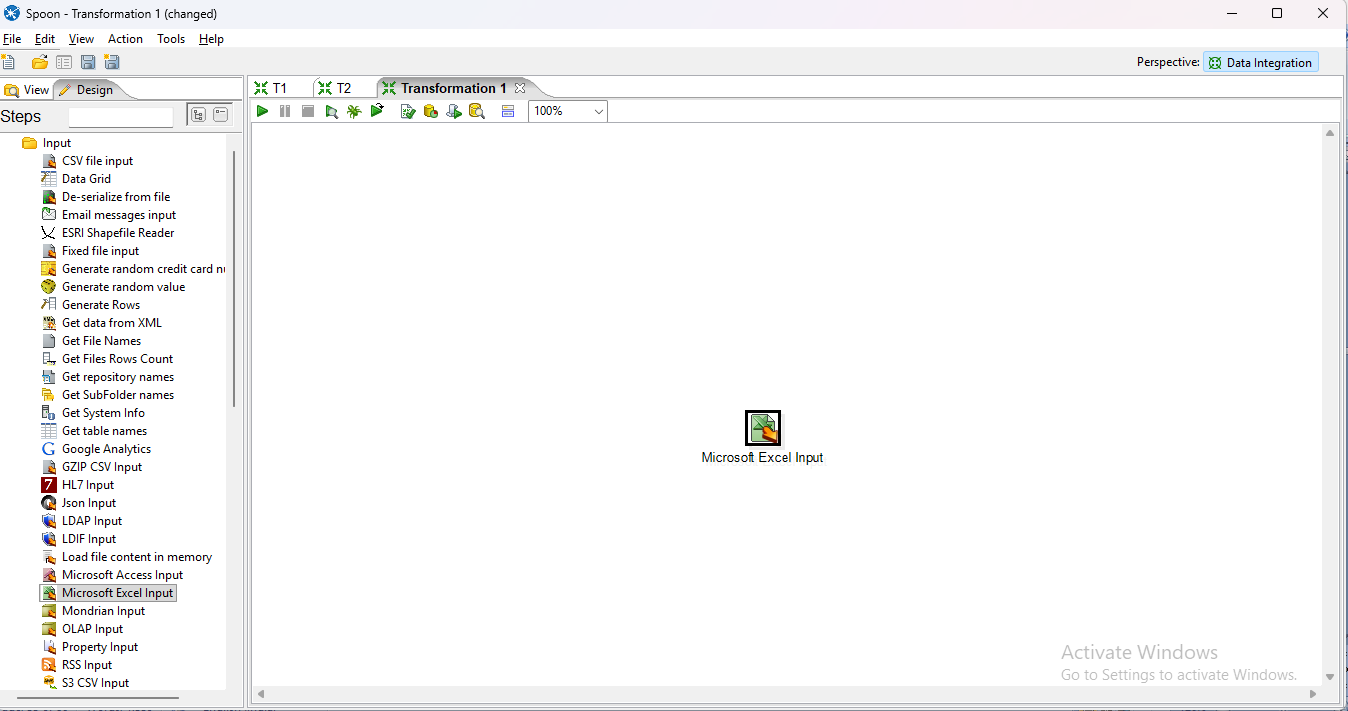
**(v) Run the Table input.**

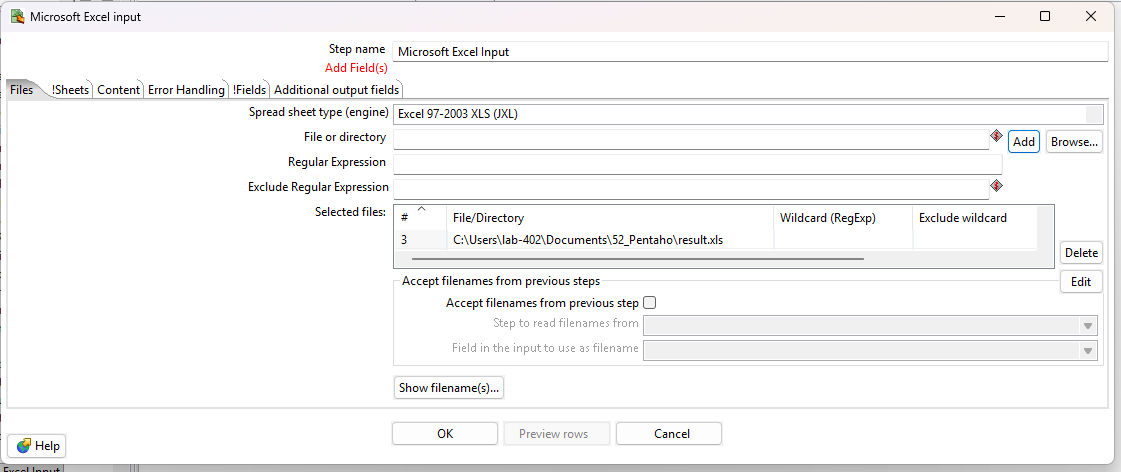
****

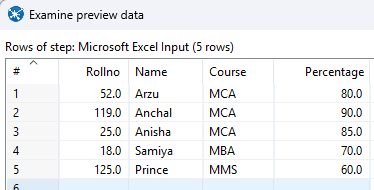
# PRACTICAL NO: 12

**Aim: Implementation of Output Functions – Excel input to Excel Output and Excel input to Table output, excel input to XML output.**

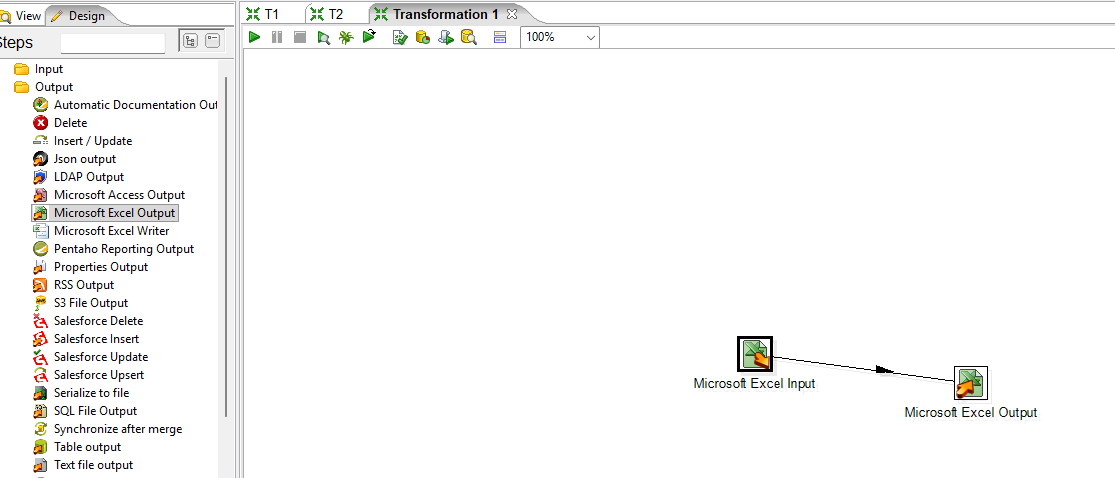
**a) Excel input to excel Output.**

**(i) Add Excel File in Microsoft Excel Input.**

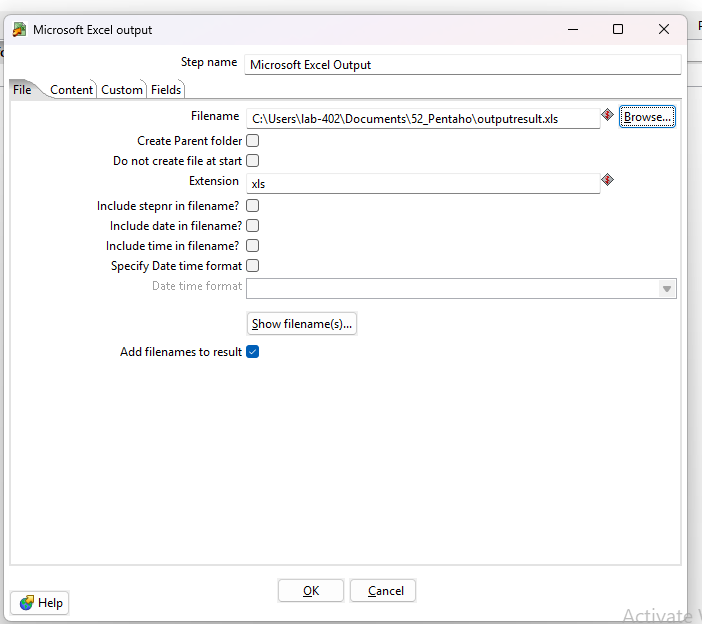


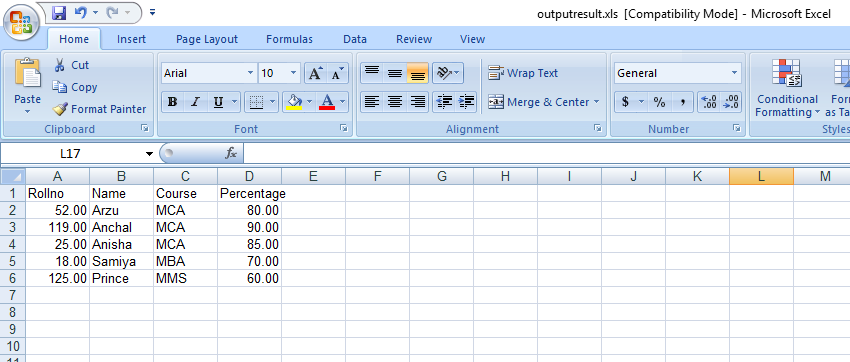
****

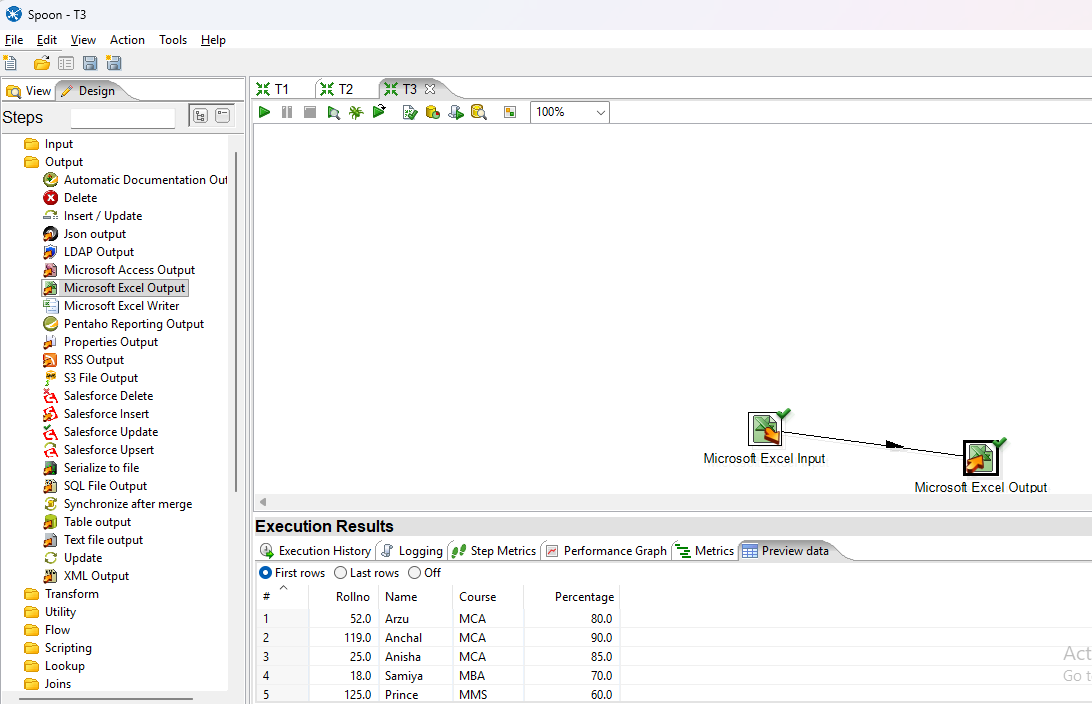
**(ii) Drag & drop Microsoft Excel Output and connect with Microsoft Excel Input.**



**(iii) Double click on Microsoft Excel Output and create & Add Empty Excel file.**

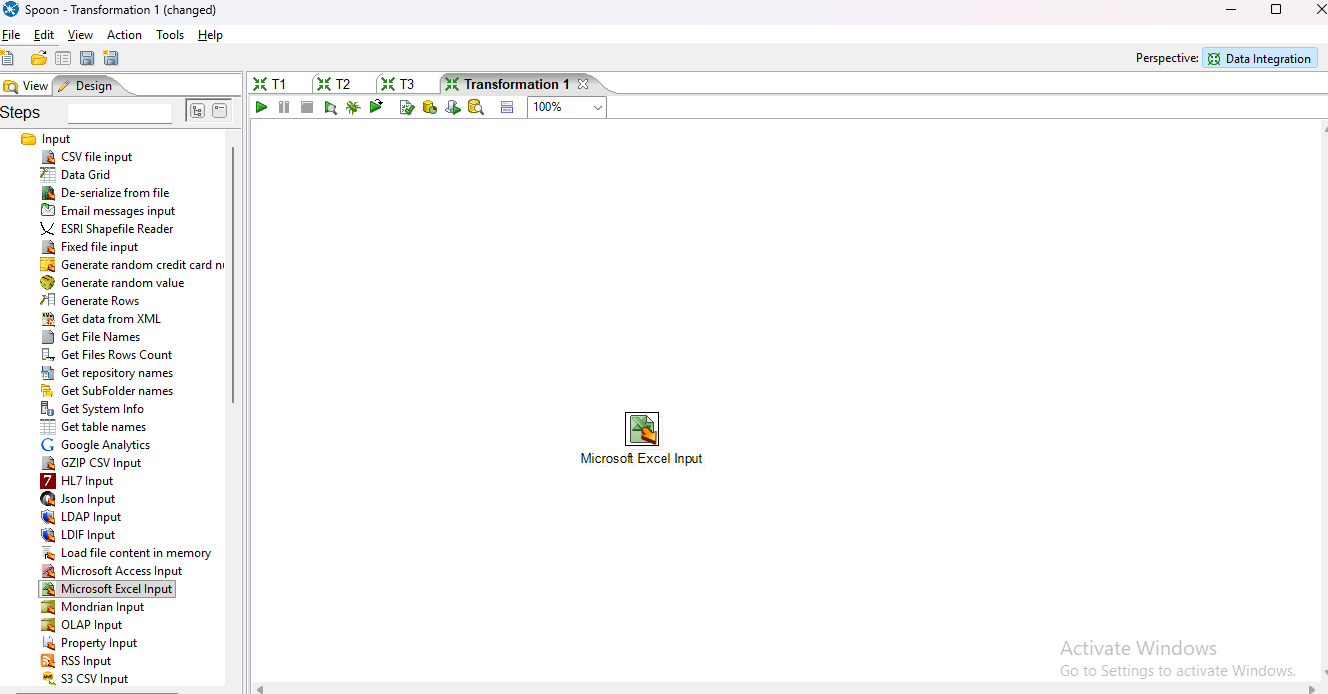


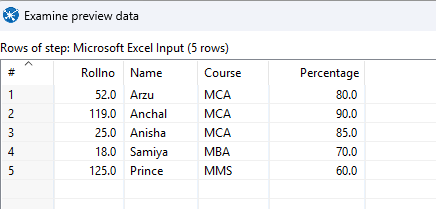


**(iv) Run the transformation file.**

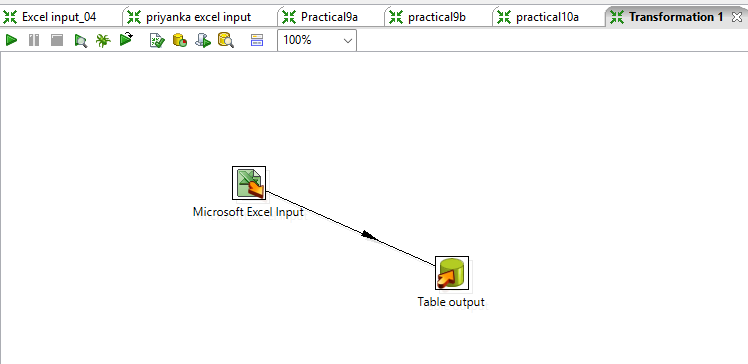
**b) Excel Input to Table Output.**

**(i) Add Excel File in Microsoft Excel Input.**

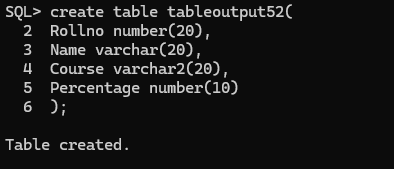




**(ii) Drag & drop Table Output and connect with Microsoft Excel Input.**

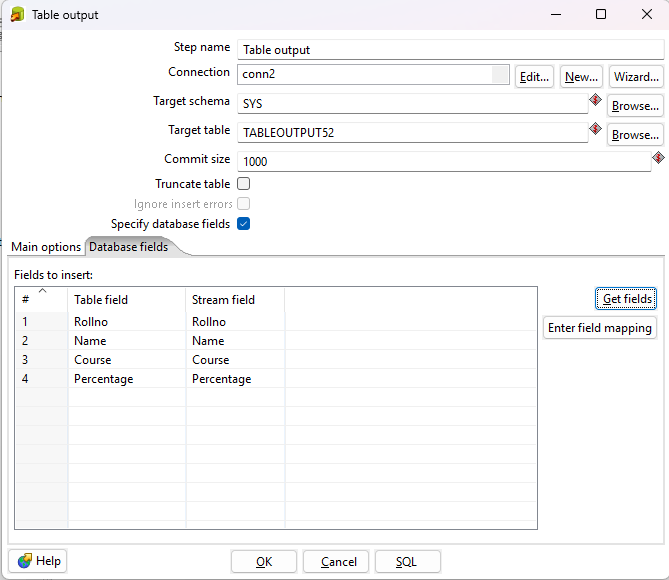


**(iii) Create an SQL Empty table.**

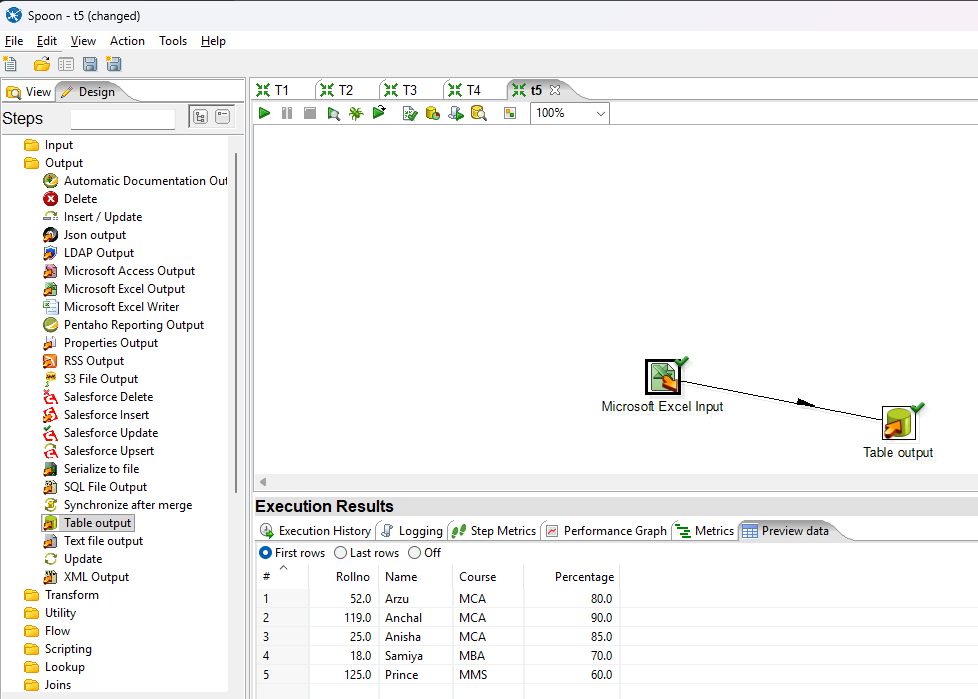


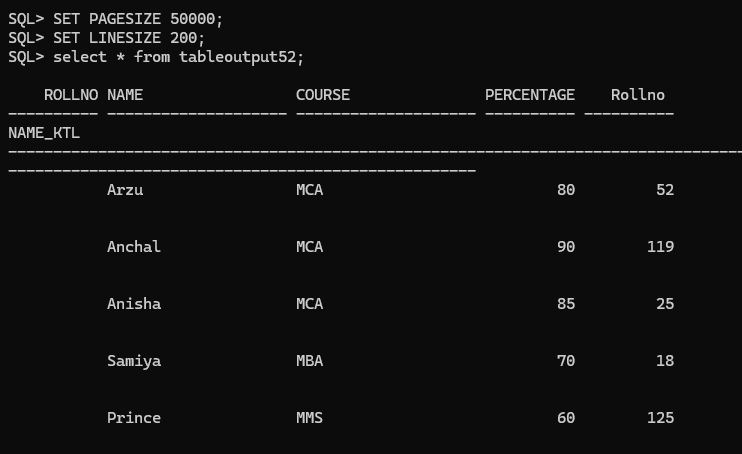
**(iv) Double click on Table Output , create a new connection & Connect with SQL.**

**(v) Set the target Schema, set the target table and click on get fields.**



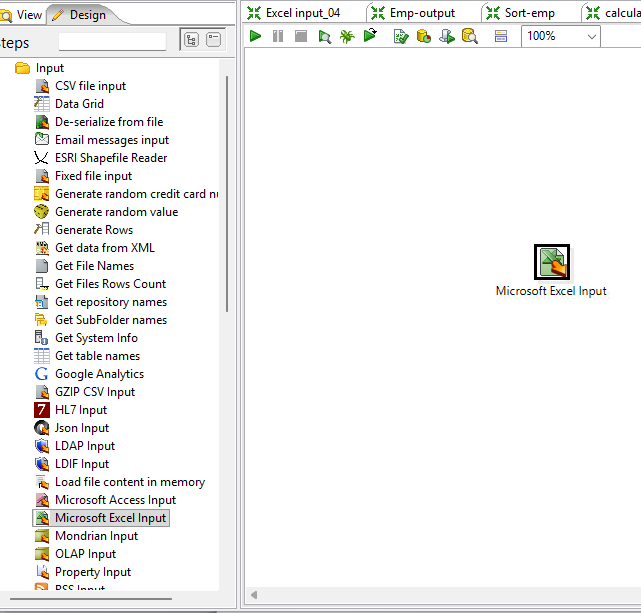
**(vi) Run the transformation file .**

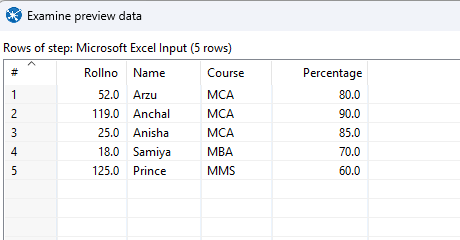




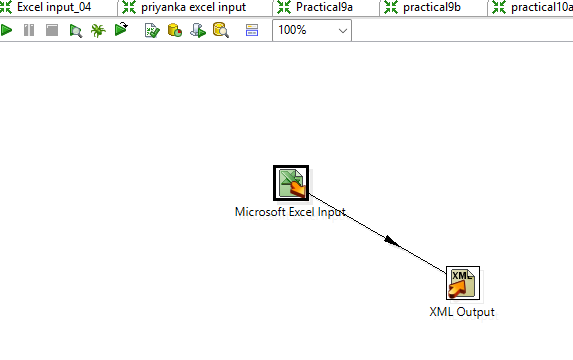
**c) Excel input to XML output.**

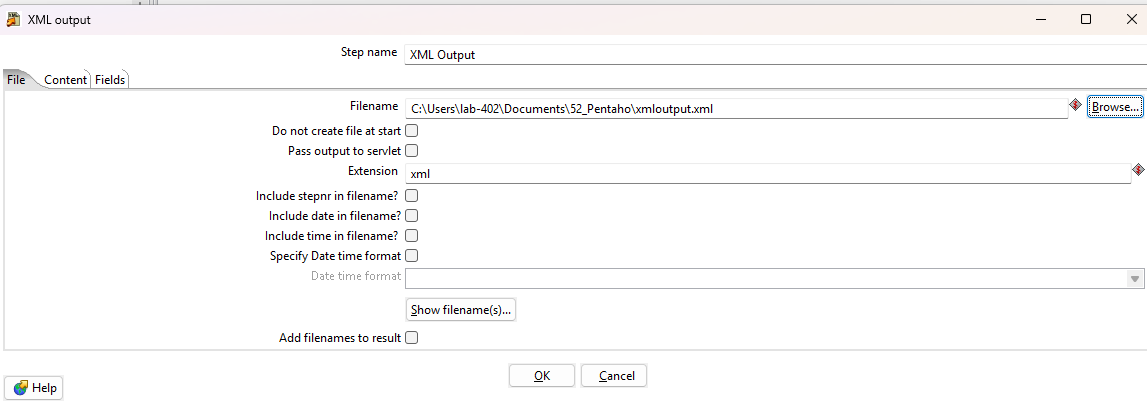
**(i) Add Excel File in Microsoft Excel Input.**

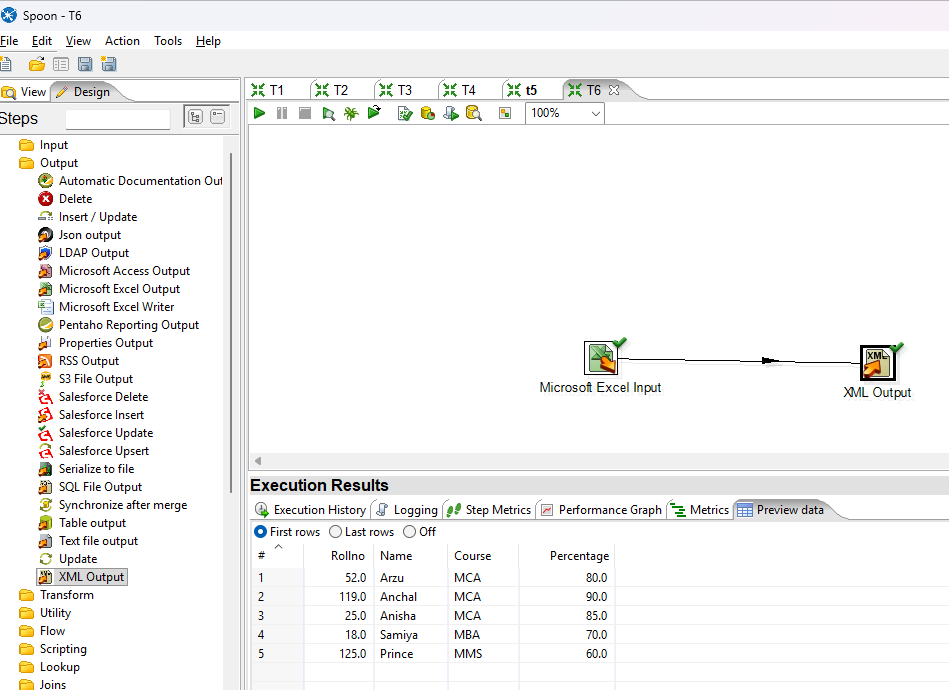




**(ii) Drag & Drop XML output and connect with Microsoft Excel Input.**



**(iii) Double click on XML Output and Create & Add empty Xml file.** 

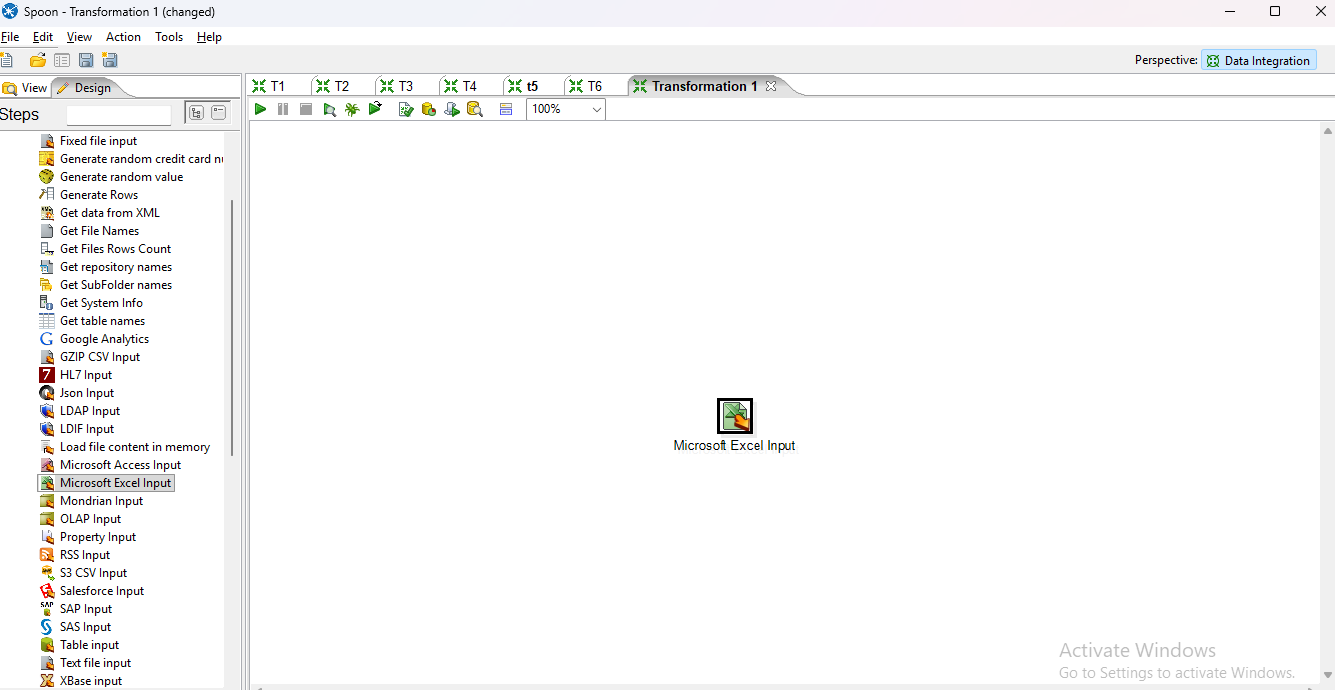
**(iv) Run the transformation file & Show xml output in the notepad.** 

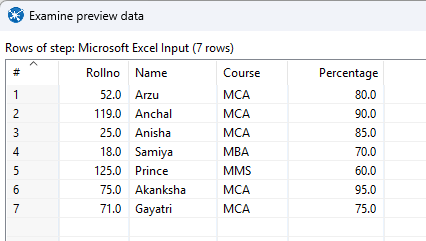
# 

# PRACTICAL NO: 13

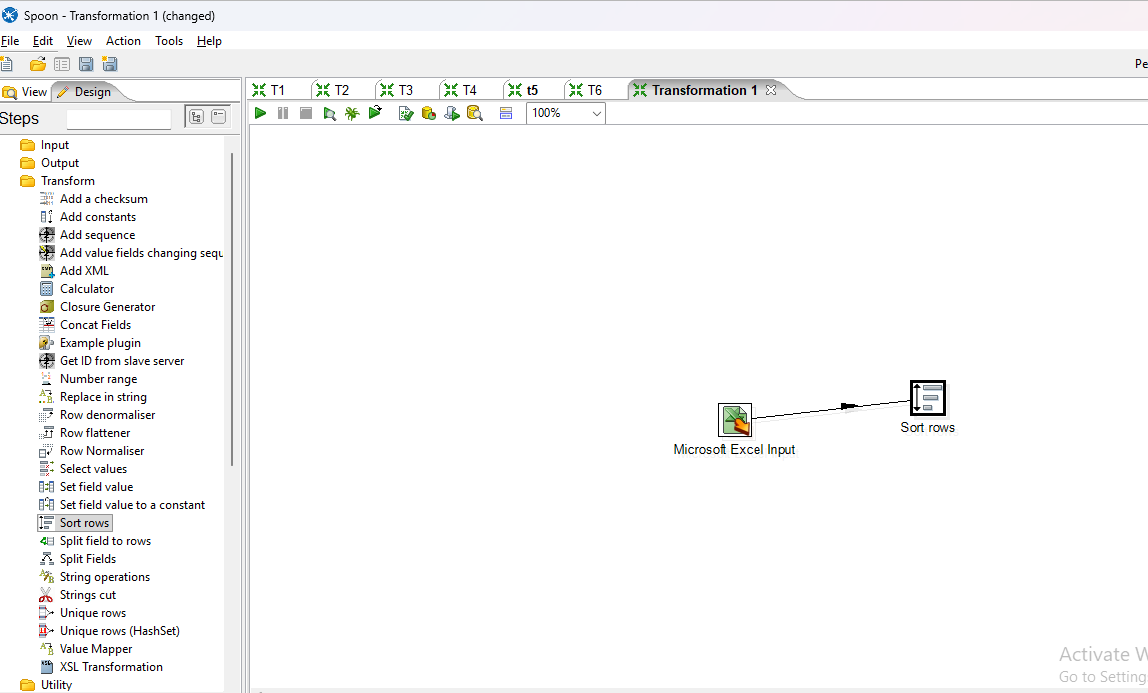
**Aim: Implementation of Transformations – Sort, Sequence, Null If, Calculator, String Operations, Upper &amp; Lower, Padding.**

**1. Sort**

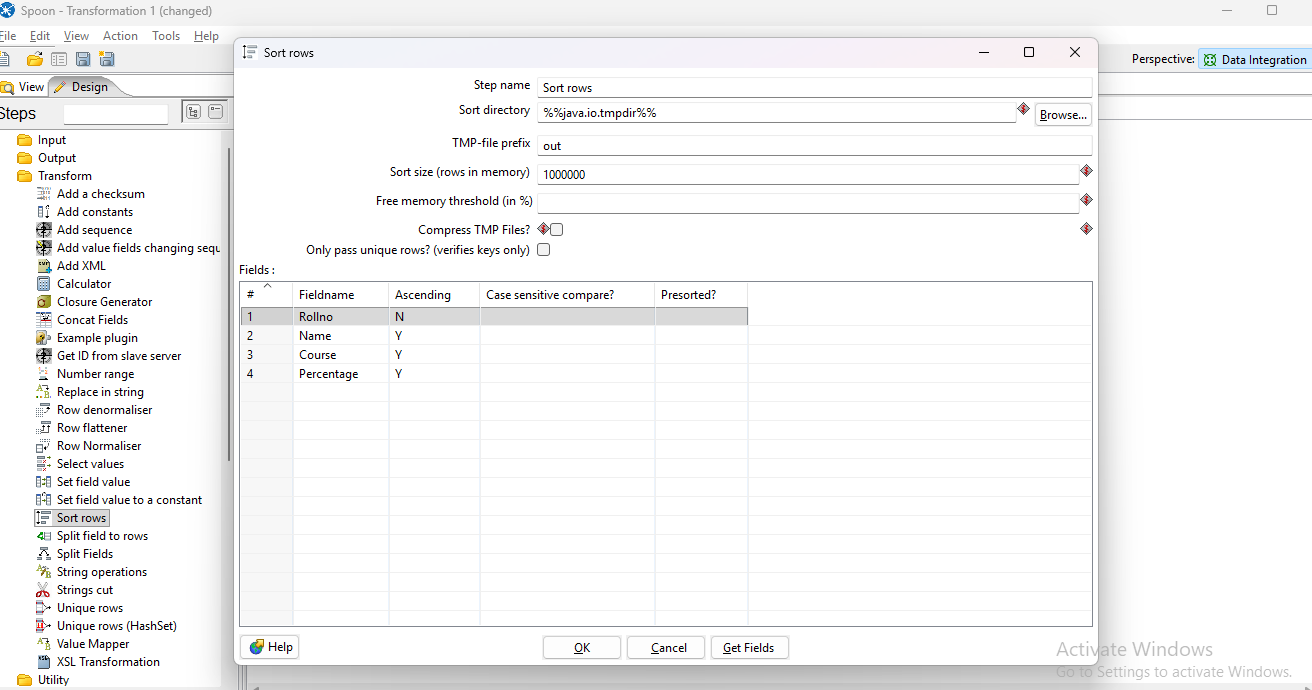
**(i) Add Excel File in Microsoft Excel Input.** 



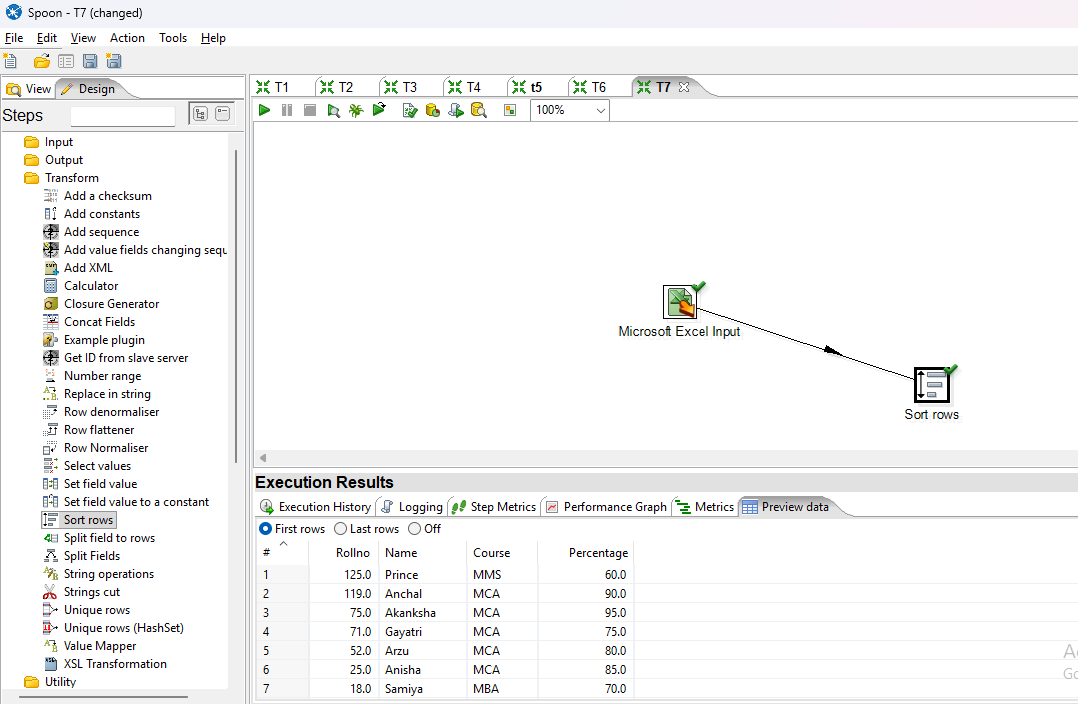
**(ii) Drag & Drop sort rows and connect with Microsoft excel input.**



**(iii) Click on Sort rows and go to get fields to set the order.**

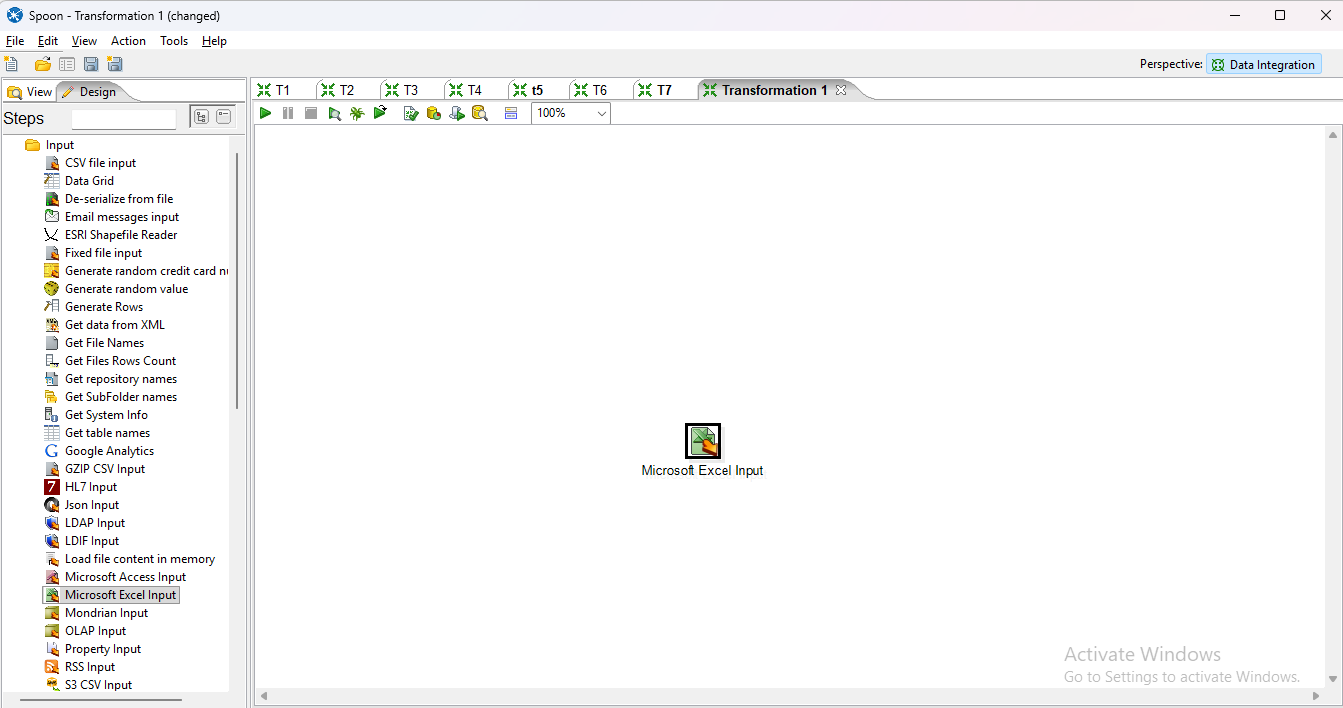


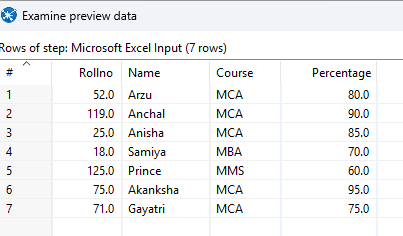
**(iv) Run the transformation**

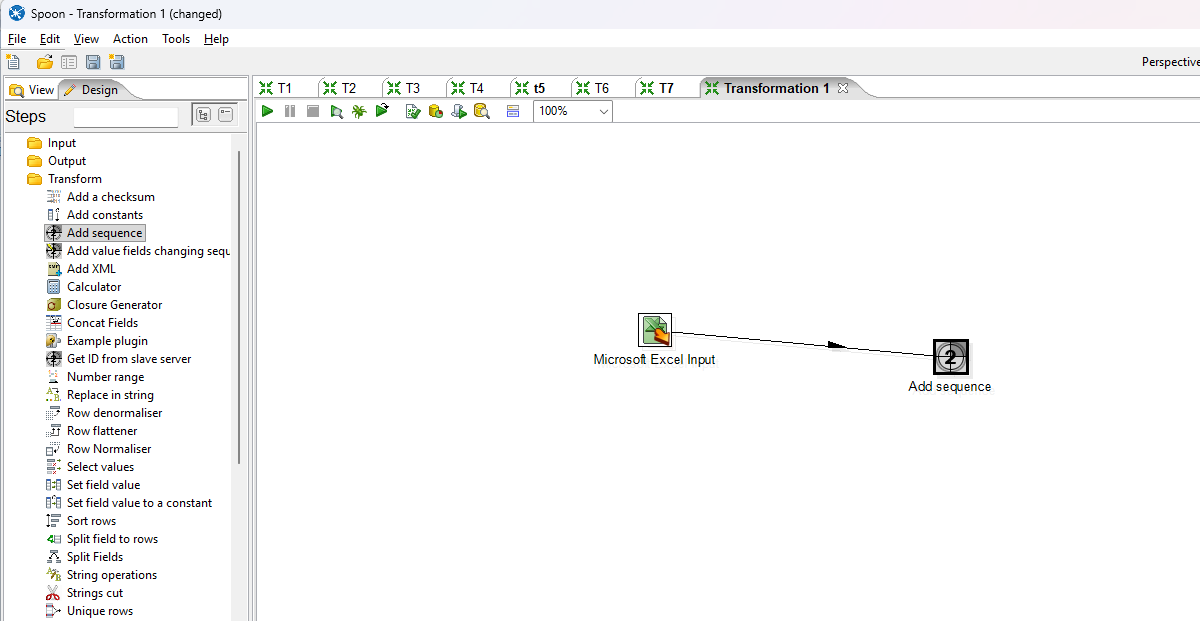


**2. Sequence**

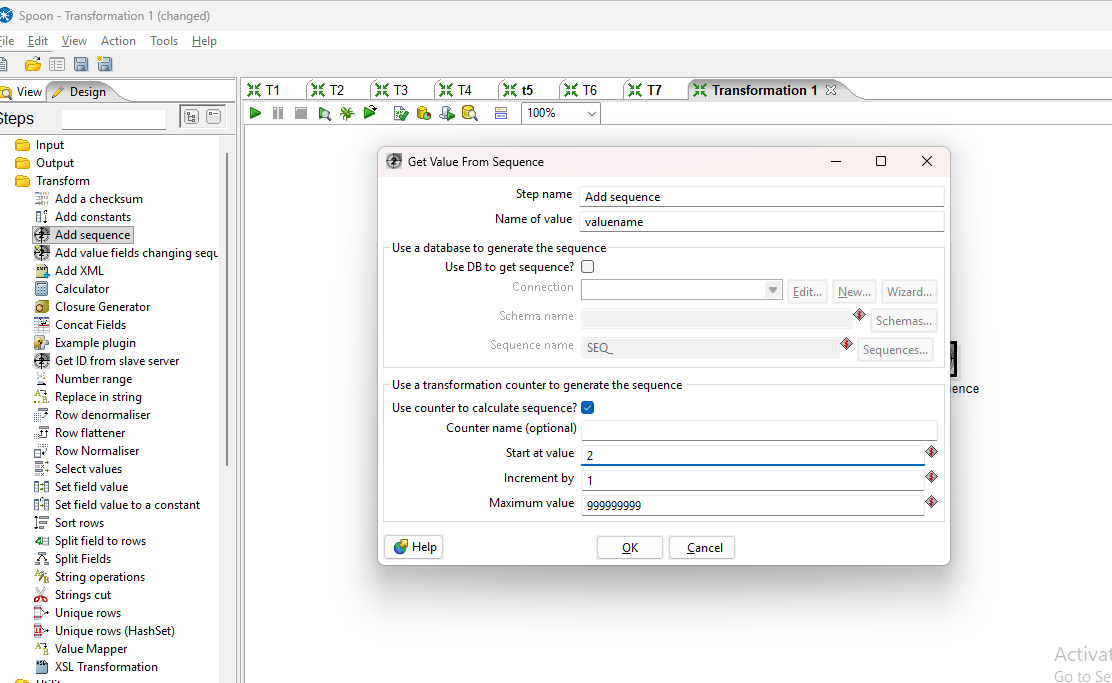
**(i) Add Excel File in Microsoft Excel Input.**



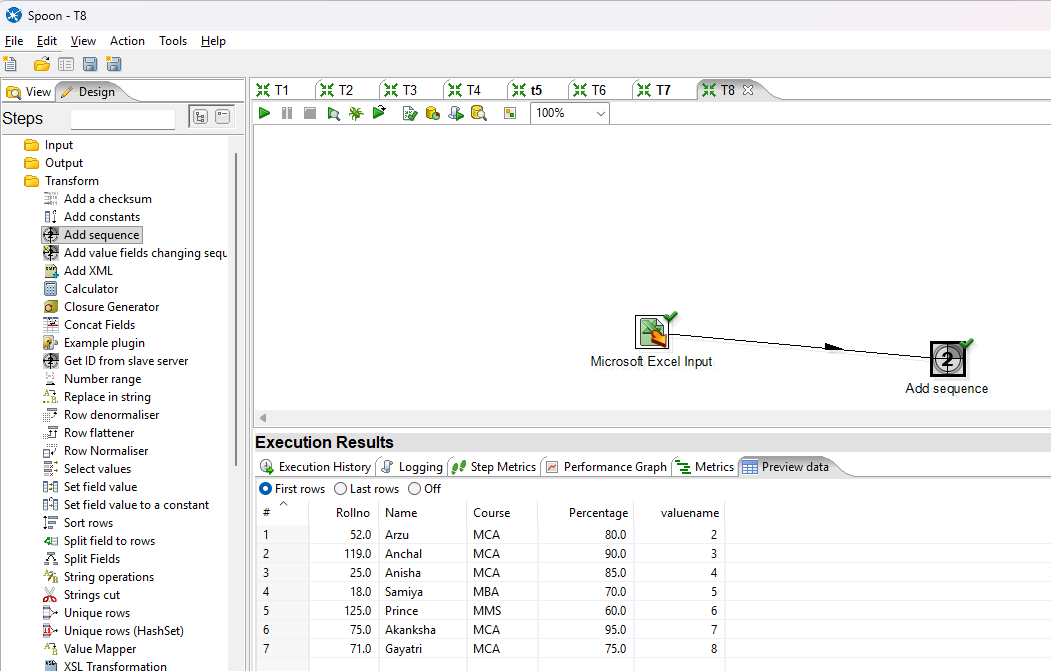


**(ii) Drag & Drop Add Sequence and connect with Microsoft excel input.**

**(iii) Double click on Add sequence and set the Start values.**

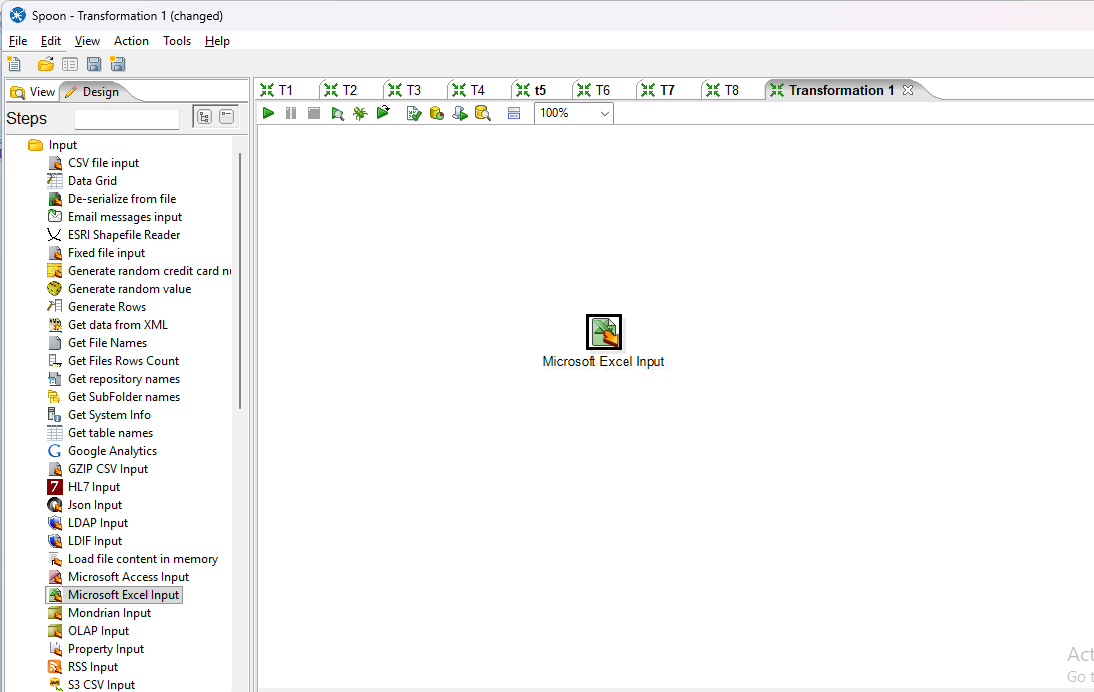


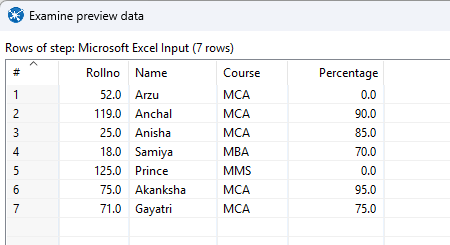
**(iv) Run the transformation.**

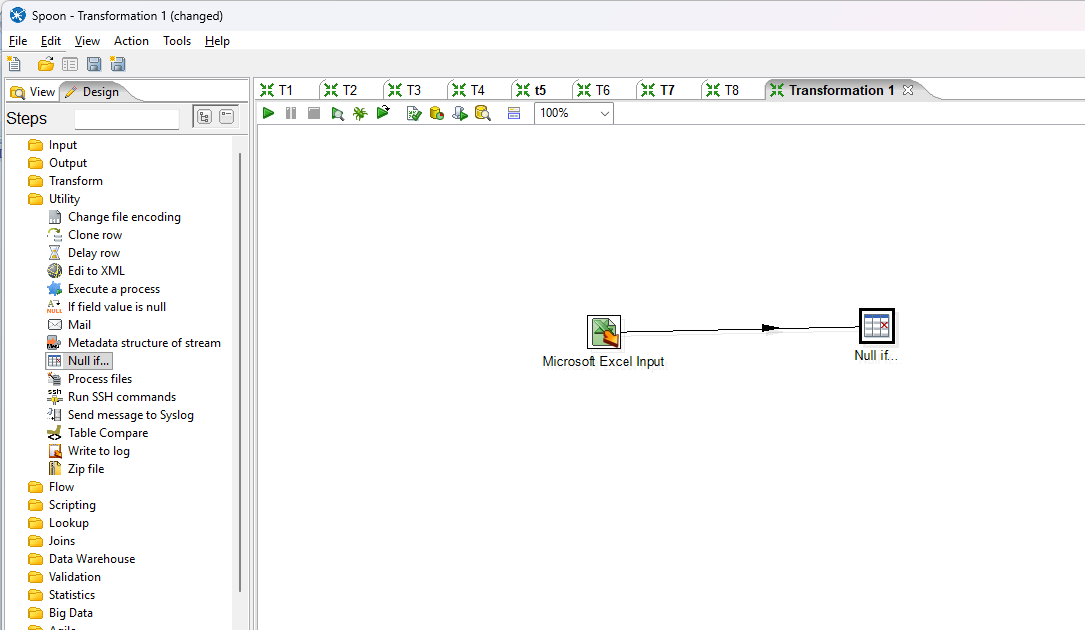


**3. Null if**

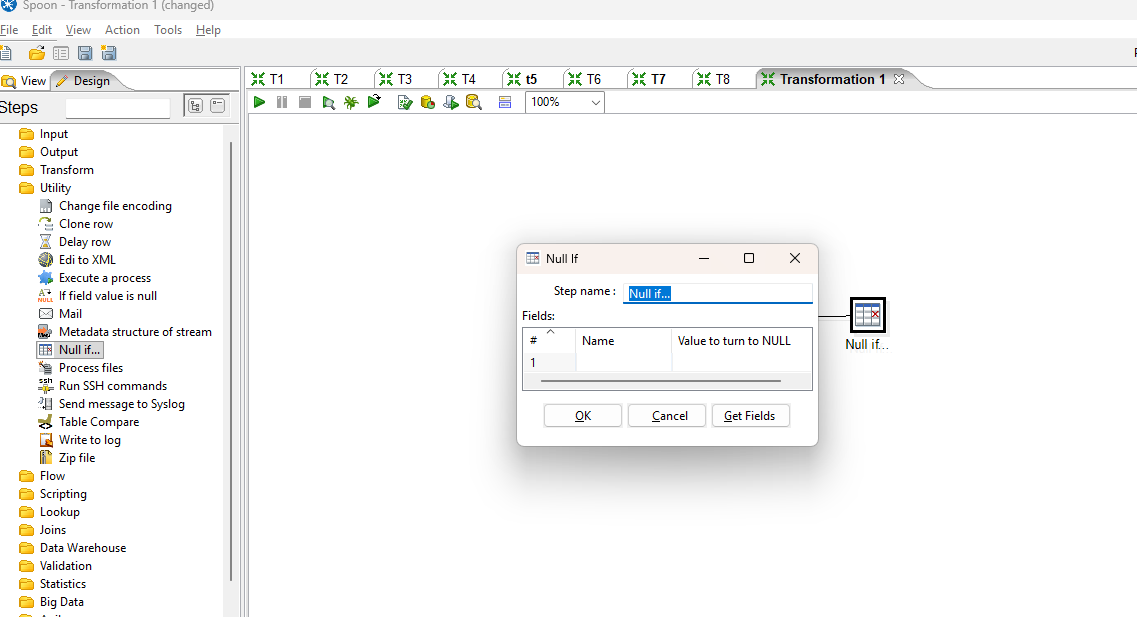
**(i) Add Excel File in Microsoft Excel Input.**

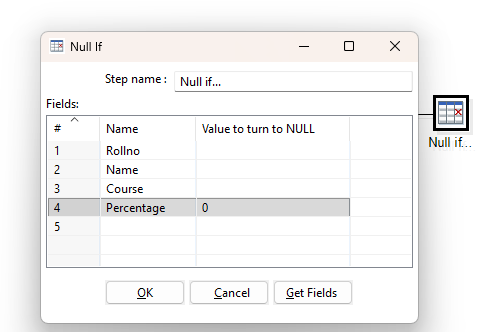




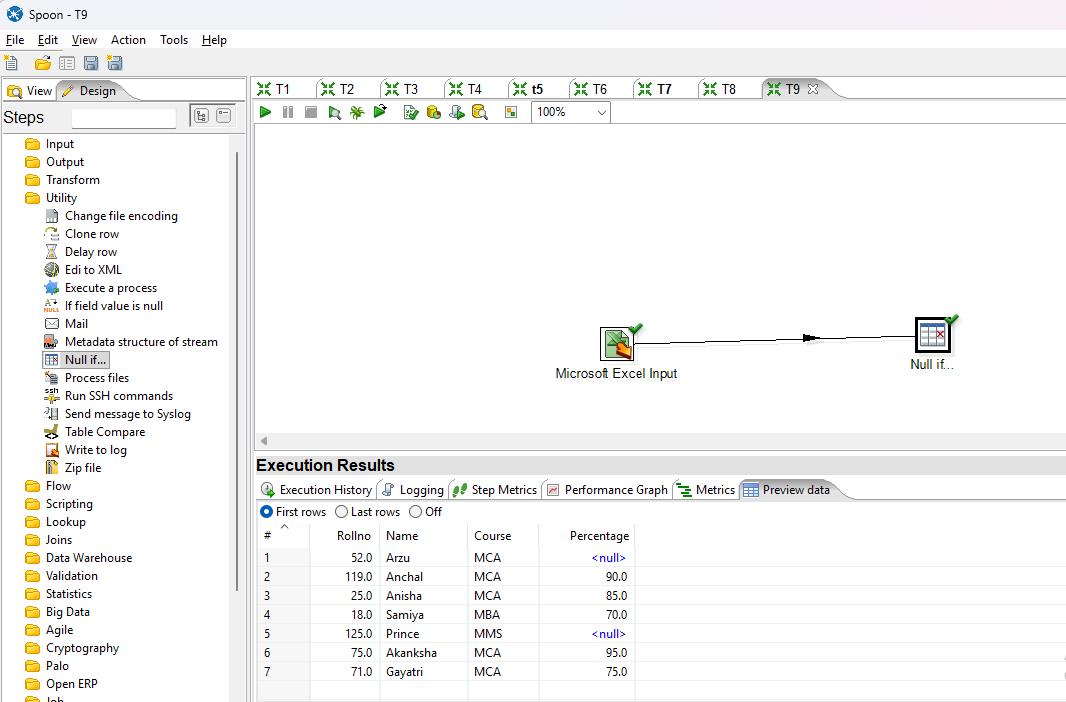
**(ii) Drag & Drop Null if and connect with Microsoft excel input.** 

**(iii) Double click on Null if and set the value to Null.**

****

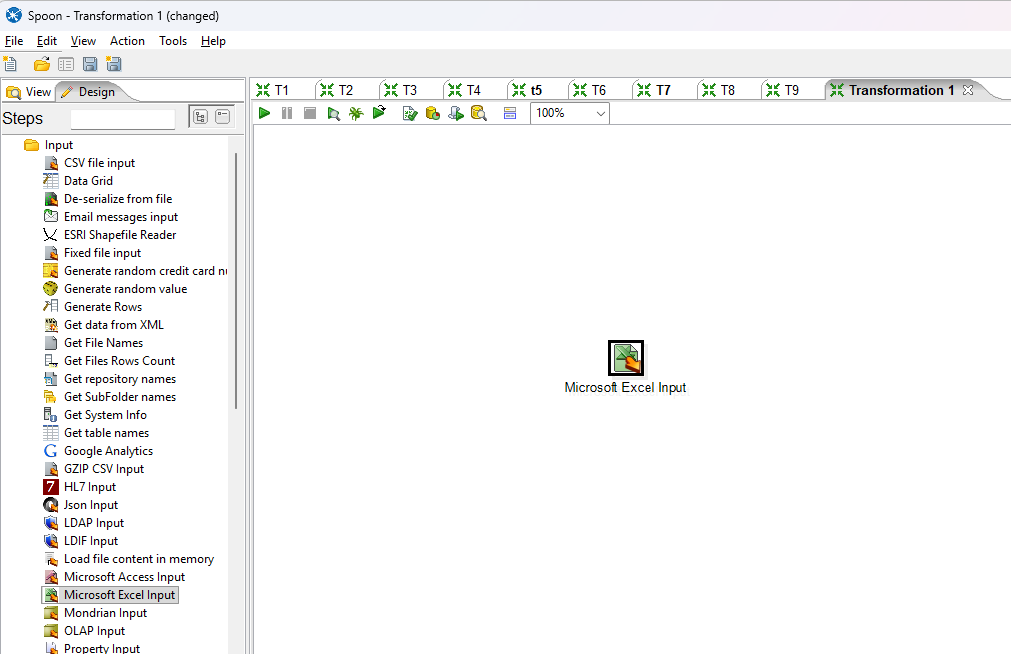
****

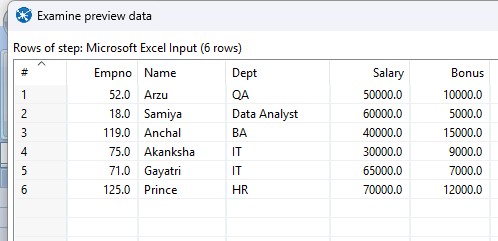
**(iv) Run the transformation.**



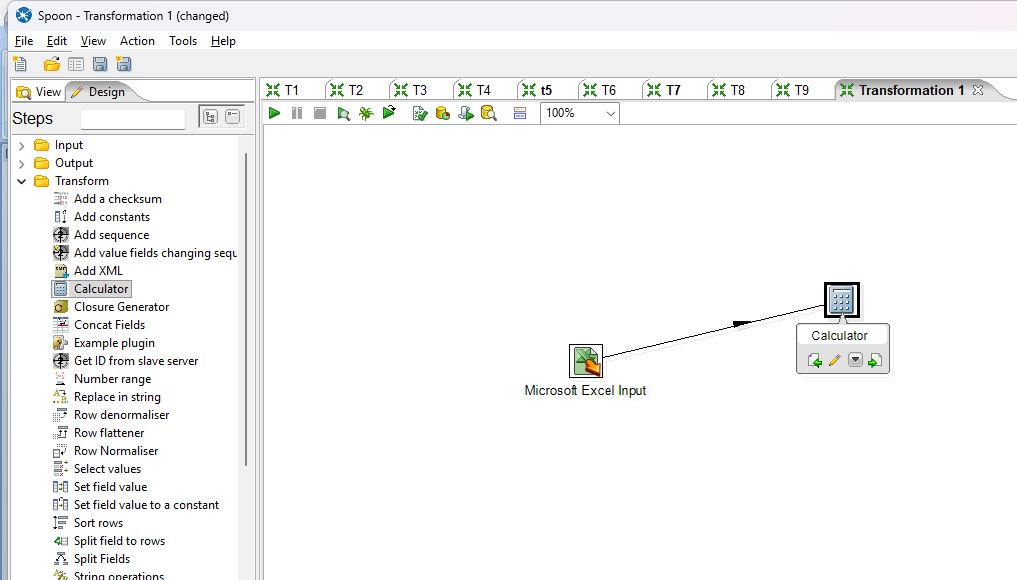
**4. Calculator**

**(i) Add Excel File in Microsoft Excel Input.**

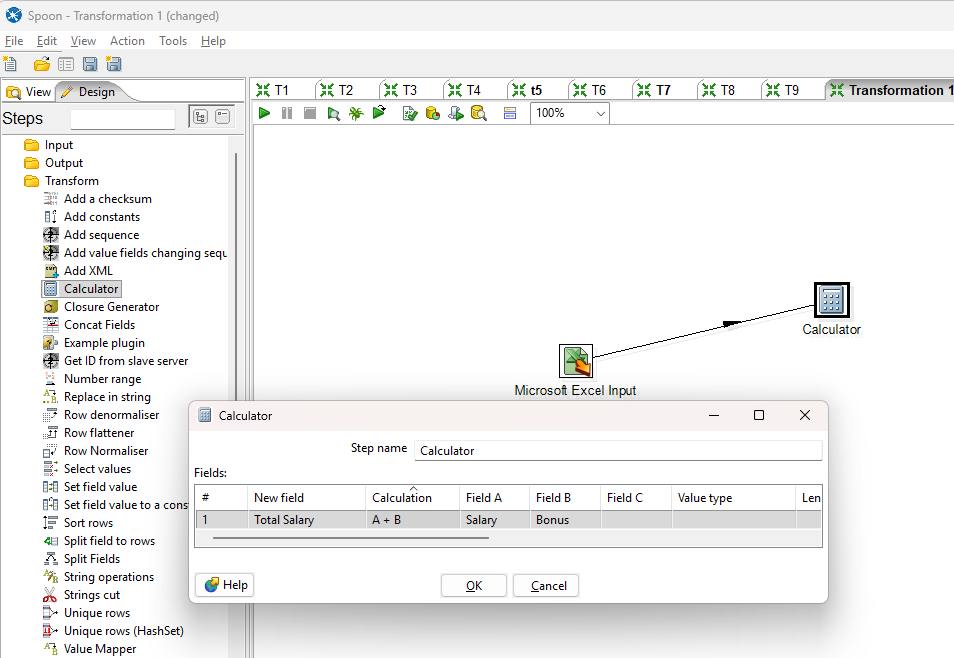


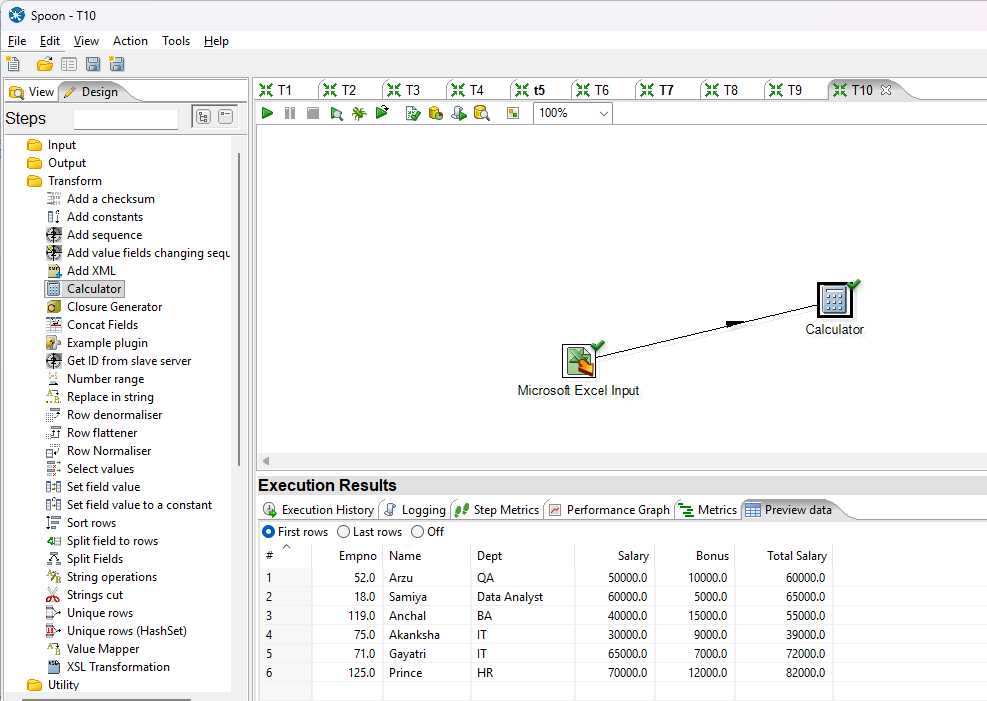


**(ii) Drag & Drop Calculator and connect with Microsoft Excel input.**



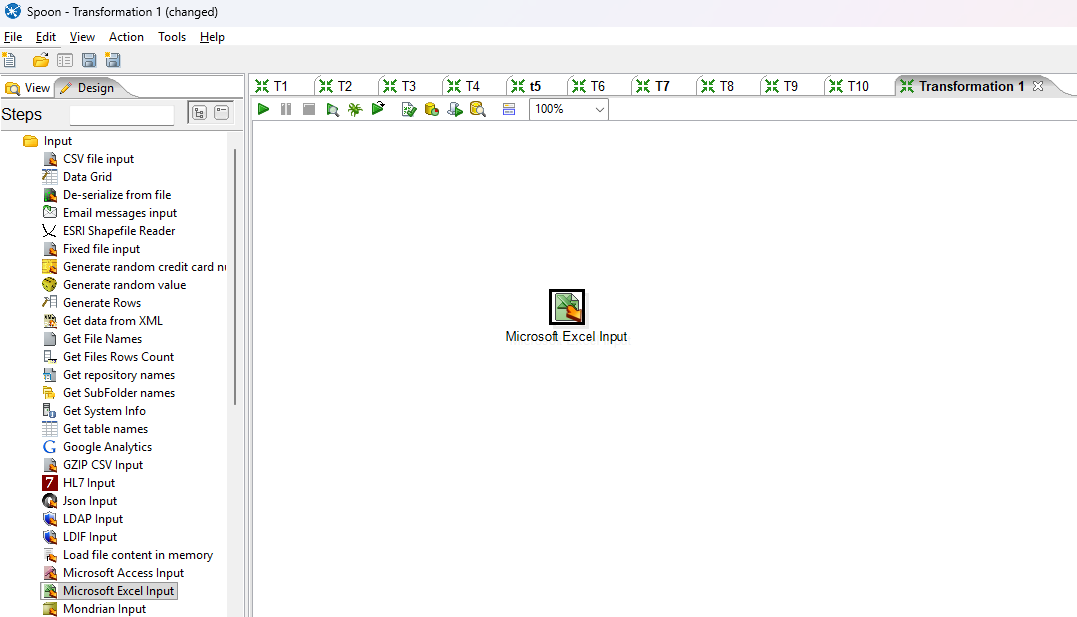
**(iii) Double click on calculator and set the value of fields as:**

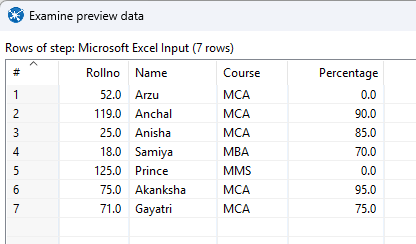


**(iv) Run the Transformation file.** 

**5. String Operations (Upper / Lower & Padding).**

**(i) Add an Excel file in Microsoft Excel Input.**

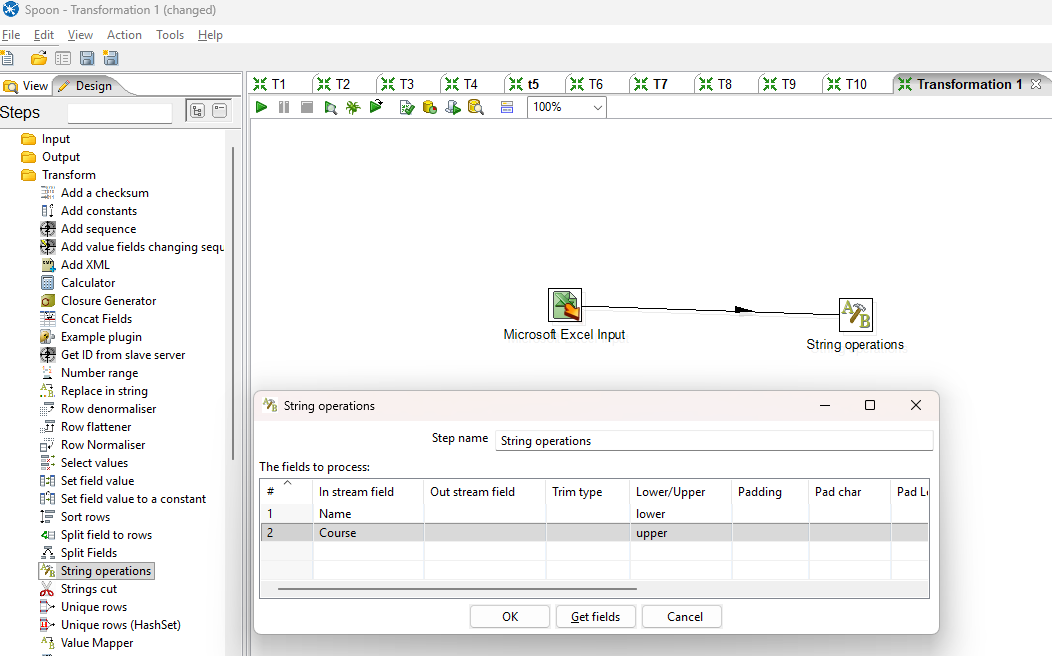




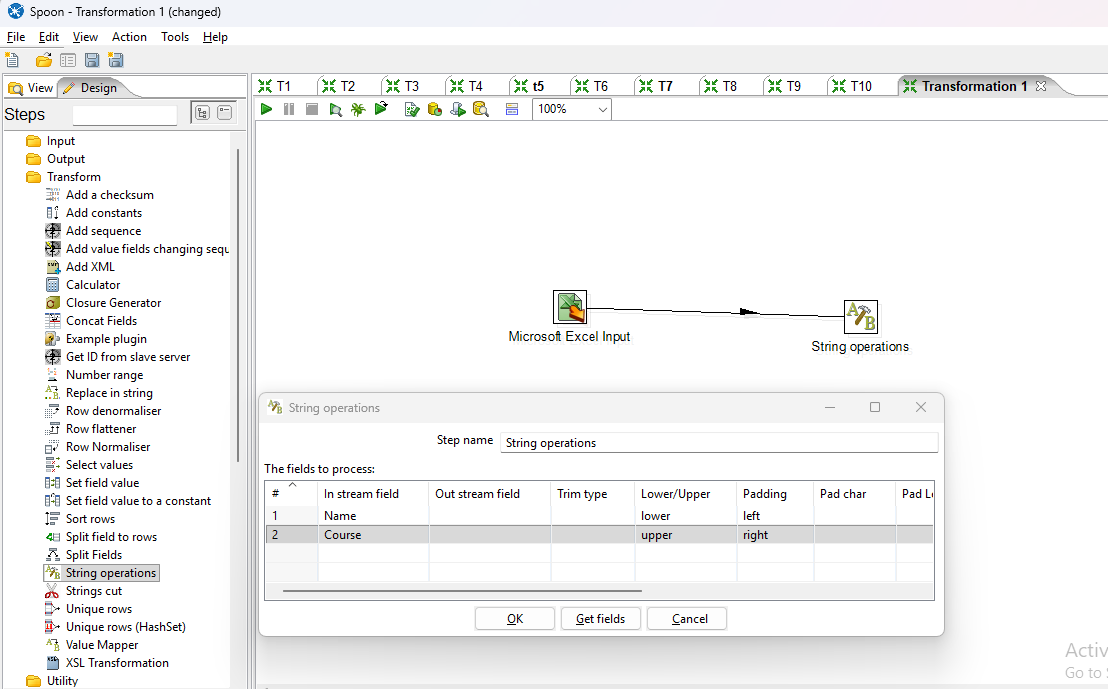
**(ii) Drag & drop string operations and connect with Microsoft Excel Input.**



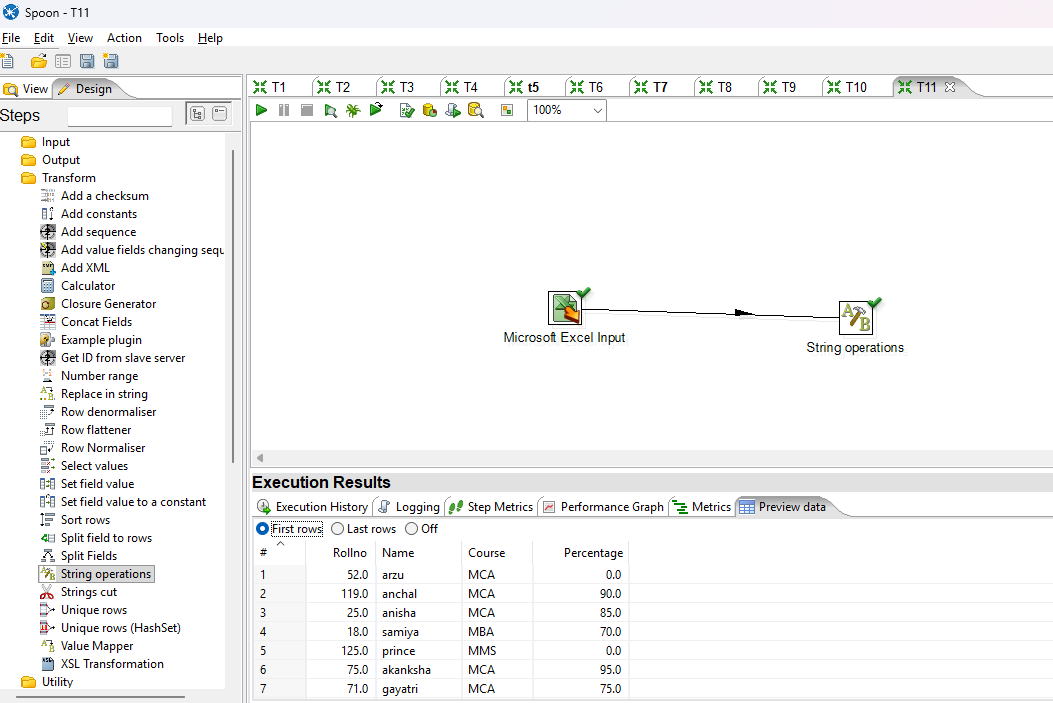
**(iii) Double click on string operations and set the Upper & Lower .**



**(iv) Double click on string operations and set the Padding.**



**(v) Run the Transformation file.**



# PRACTICAL NO: 14

**A] Aim: Implementation of Basic Command using R.**

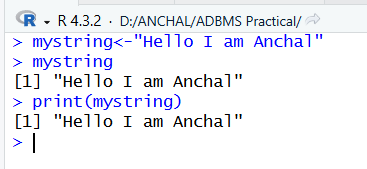
**1) Write a command to print a string in R.**

mystring<-"Hello I am Anchal"

mystring

print(mystring)

**Output**:



**2) Write a command to print working directory.**

getwd()



**Output:**

**3) Write a command to print all the files from directory.**

dir()



**4) Write a command to print variables listed.**

ls()

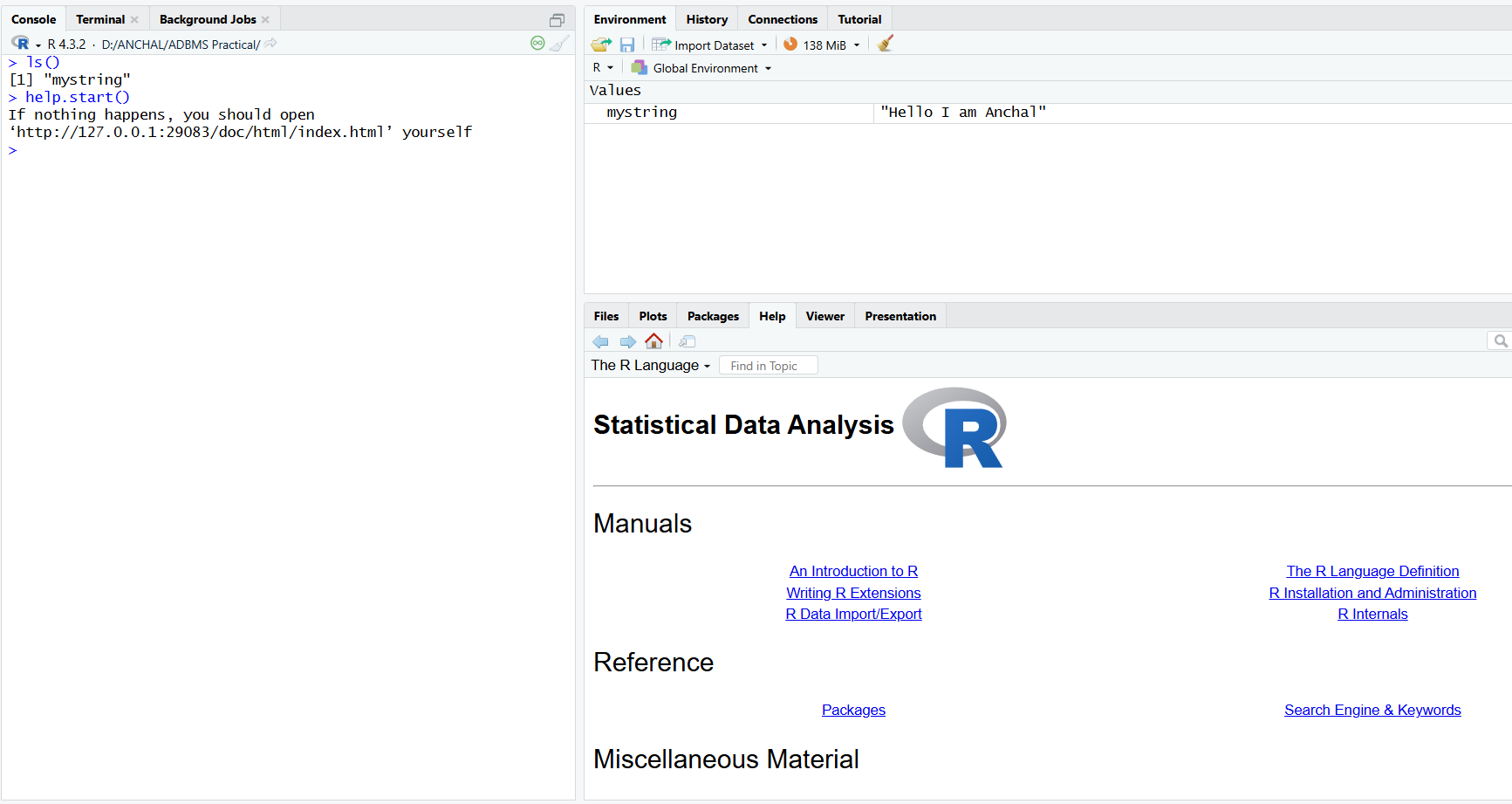
**Output**



**5) Write a command to start a html page**

help.start()

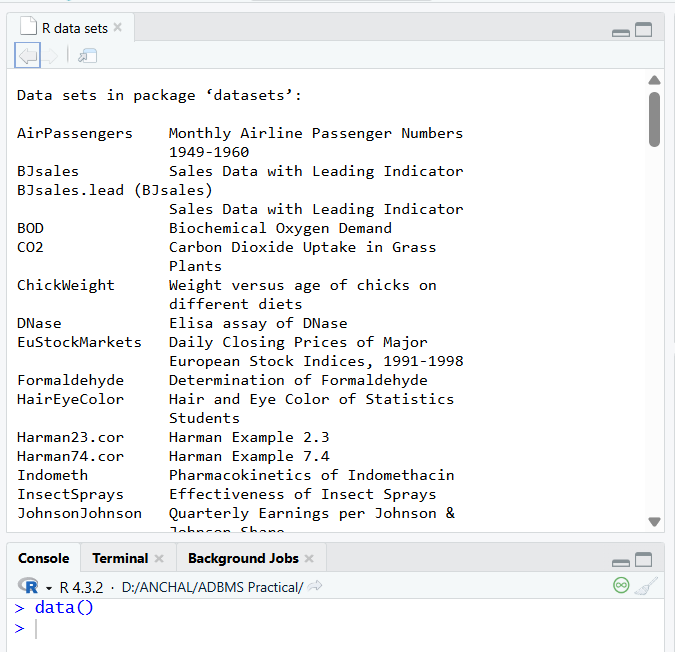
**Output:**



**6) Write a command to list all data sets available.**

data()

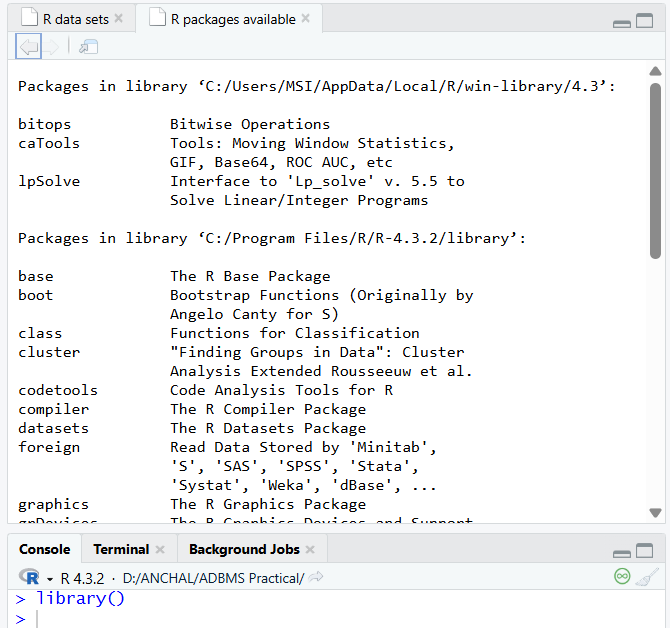
**Output**



**7) Write a command to list all packages available**

library()

**Output**

****

**8) Write a command to retrieve the data type of the variable.**

num<-107

class(num)

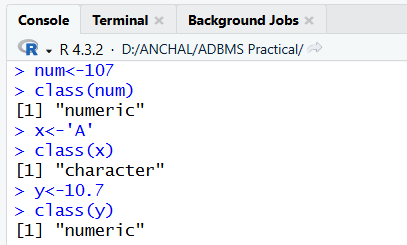
x<-'A'

class(x)

y<-10.7

class(y)

**Output:**

****

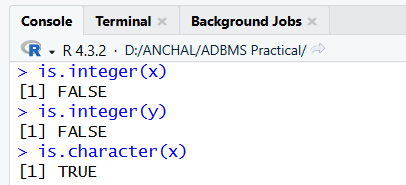
**9) Write a command to verify its data type.**

is.integer(x)

is.integer(y)

is.character(x)

**Output:**

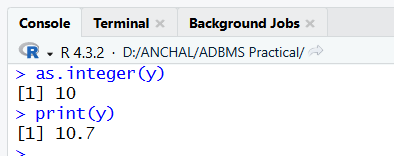
****

**10) Write a command to parse float value to int.**

as.integer(y)

print(y)

**Output:**

****

**11) Write an expression to create a vector using the c function and print it.**

### Creating vectors: It contains objects of same class

### creating vectors c function

x1<-c(61.3,107.5,91.9)

x2<-c(3,5,6,7,4,2)

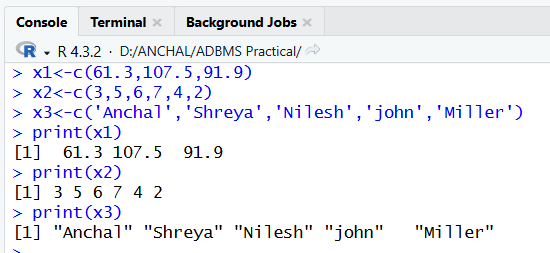
x3<-c('Prince','Anurag','Satish','Gayle','Miller')

print(x1)

print(x2)

print(x3)

**Output:**

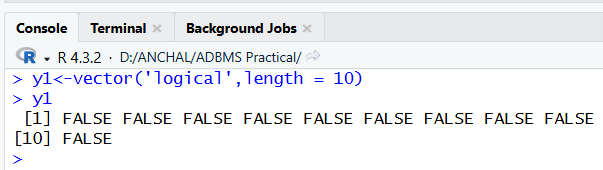
****

**12) Write an expression to create a vector using the vector function and print it.** #Using vector function

y1<-vector('logical',length = 10)

y1

**Output**

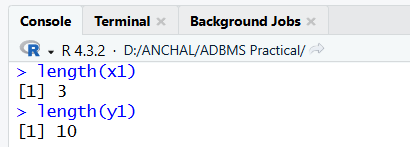
****

**13) Write an expression to print the length of the vector.**

length(x1)

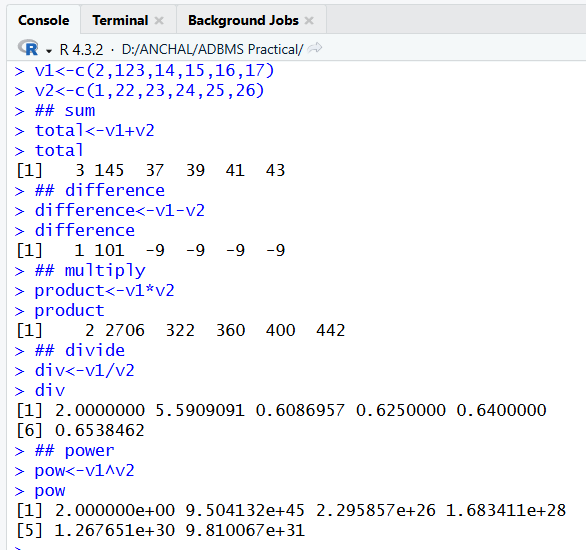
length(y1)

**Output:**

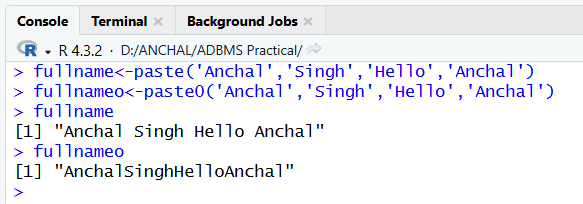
****

**14) Write an expression to create 2 vectors and perform arithmetic operations on it.**

### Create 2 numeric vectors v1 and v2 with 3 values each and perform the arithmetic operations and print it.

****

**15) Write an expression to concatenate strings.**

****

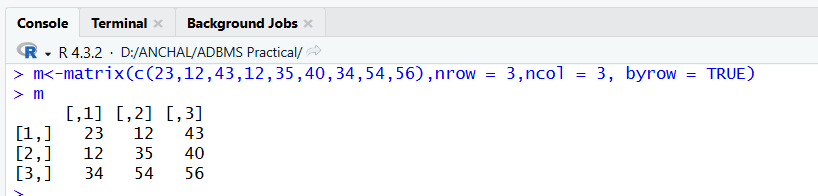
**16) Write an expression to create a matrix and print it.**

## Creating Matrix: Two dimensional array having elements of same class

## Using matix function

m<-matrix(c(23,12,43,12,35,40,34,54,56),nrow = 3,ncol = 3, byrow = TRUE) m

**Output:**

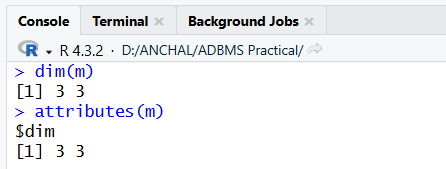
****

**17) Write an expression to find the dimensions of the matrix**

dim(m)

attributes(m)

**Output**

****

**18) Write an expression to bind vectors.**

s1<-c(12,23,34,45,56)

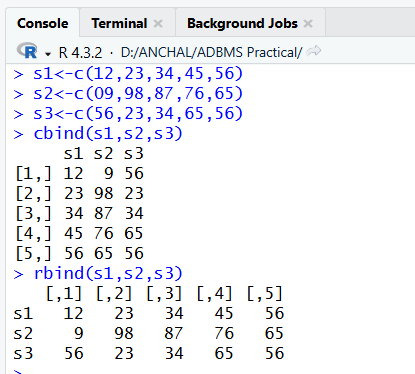
s2<-c(09,98,87,76,65)

s3<-c(56,23,34,65,56)

cbind(s1,s2,s3)

rbind(s1,s2,s3)

**Output**

****

**19) Write an expression to create 2 matrices and perform addition, subtraction and multiplications on it.**

mat1<-matrix(c(1,2,3,4,5,6),nrow = 3,ncol = 2, byrow = TRUE)

mat2<-matrix(c(7,8,9,10,11,12),nrow = 2,ncol = 3, byrow = TRUE)

## multiplication

matproduct <- mat1 %\*% mat2

matproduct

## addition

mat3<-matrix(c(1,2,3,4),nrow = 2,ncol = 2, byrow = TRUE)

mat4<-matrix(c(7,8,9,10),nrow = 2,ncol = 2, byrow = TRUE)

matsum <- mat3 + mat4

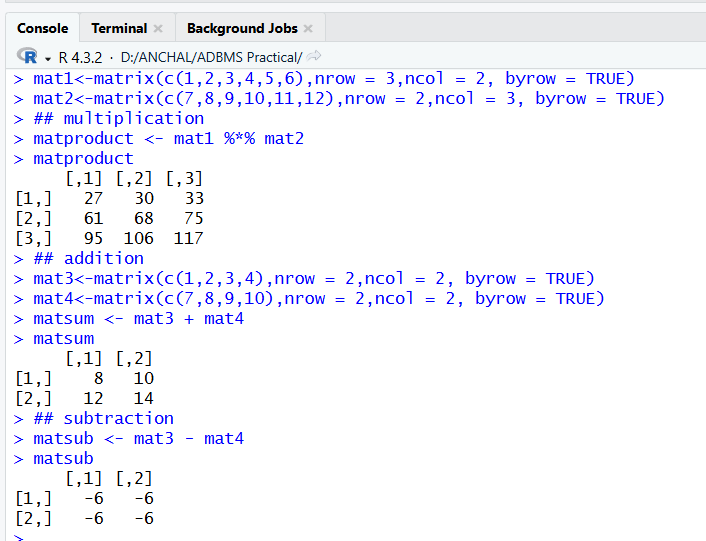
matsum

## subtraction

matsub <- mat3 - mat4

matsub

**Output**

****

**20) Write an expression to create a list in r.**

#LIST: Special type of vector containing elements of different classes.

#Elements of a list can be accessed by giving the name of the element index or namein[[]] #Creating LIST

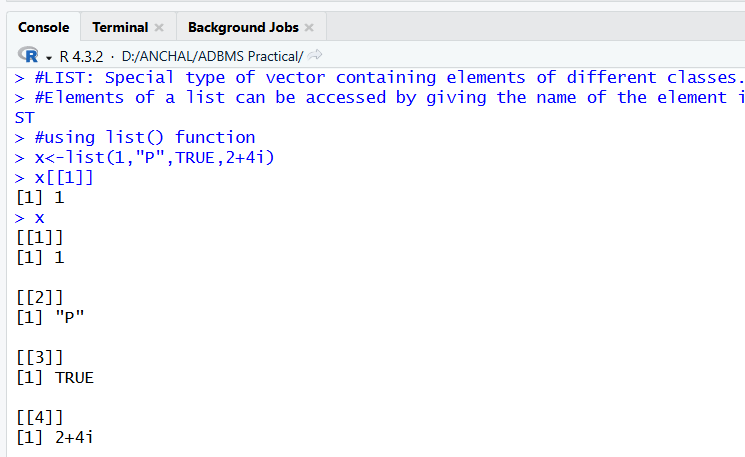
#using list() function

x<-list(1,"P",TRUE,2+4i)

x[[1]]

x

**Output**

****

**21) Write an expression to execute the factor in r.**

#FACTOR: Represents categorical data. Can be ordered or unordered.

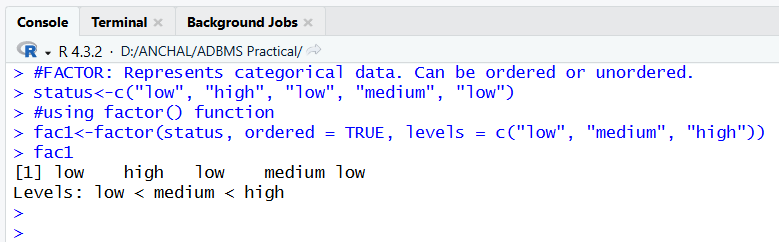
status<-c("low", "high", "low", "medium", "low")

#using factor() function

fac1<-factor(status, ordered = TRUE, levels = c("low", "medium", "high"))

fac1

**Output**

****

**B] Aim: Implementation of Data Frame using R.**

**1) Write an expression to execute the factor in r.**

# Factor example

status <- c("low", "high", "low", "medium", "low")

fac1 <- factor(status, ordered = TRUE, levels = c("low", "medium", "high")) fac1

# Example of data frame with a factor column

student\_id<- c(1, 2, 3, 4, 5)

student\_name<- c("Anchal", "Ayush", "Ram", "Divya", "Shyam")

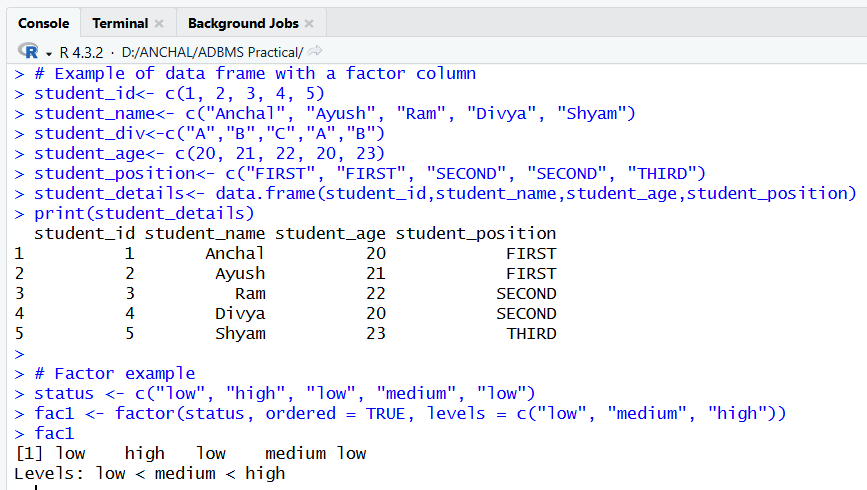
student\_div<-c("A","B","C","A","B")

student\_age<- c(20, 21, 22, 20, 23)

student\_position<- c("FIRST", "FIRST", "SECOND", "SECOND", "THIRD")

student\_details<- data.frame(student\_id,student\_name,student\_age,student\_position) print(student\_details)

**Output:**

****

**2) Write an expression for accessing a particular column in Dataframe.**

#accessing particular columns in a data frame

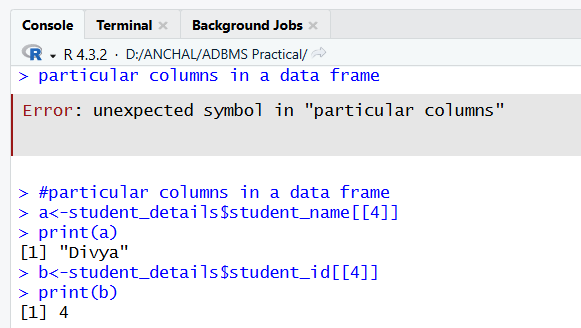
a<-student\_details$student\_name[[4]]

print(a)

b<-student\_details$student\_id[[4]]

print(b)

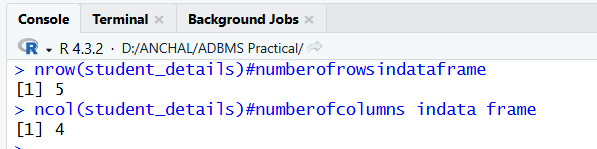
**Output:**

****

**3) Write an expression to get the number of rows and columns of the dataframe.** nrow(student\_details)#numberofrowsindataframe

ncol(student\_details)#number of columns in data frame

**Output:**

****

**4) Write an expression to access the name of the rows and columns.**

#accessing the names of columns of the table

names(student\_details)

colnames(student\_details)

rownames(student\_details)

**Output:**

****

**5) Write an expression to create a 2D table in r using TABLE command.** #TABLEcommand:used create 2 dimensional table in R

smoke<-matrix(c(63,81,42,31,32,33,34,52,35), nrow = 3, ncol = 3, byrow = TRUE) colnames(smoke)

<- c("High", "Middle", "Low")

rownames(smoke)

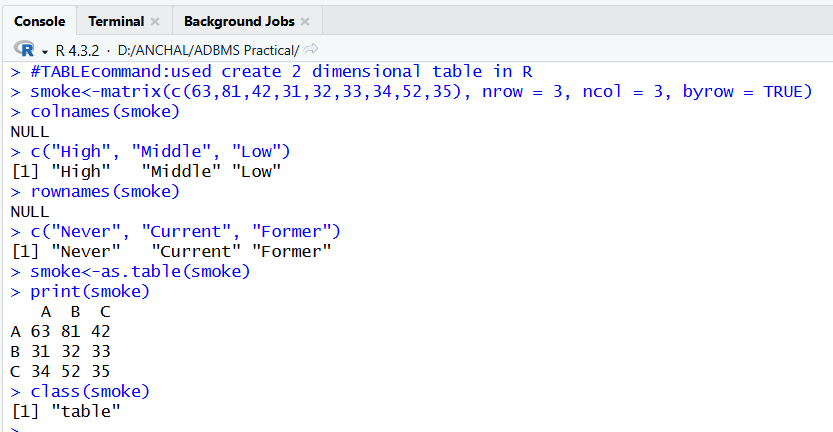
<- c("Never", "Current", "Former")

smoke<-as.table(smoke)

print(smoke)

class(smoke)

**Output:**

****

**C] Aim: Implementation of excel\_read\_and\_write using R.**

**1) Write a program for reading from Excel using readxl.**

#Reading and Writing a data from Excel using the readxl

install.packages("readxl")

library("readxl")

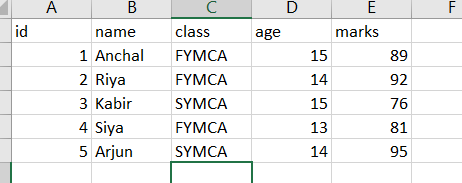
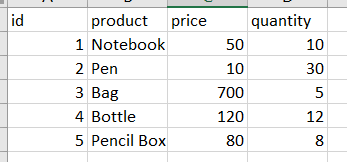
my\_data <- read\_excel("D:/ANCHAL/ADBMS Practical/tables.xlsx"

,sheet = 1) print(my\_data)

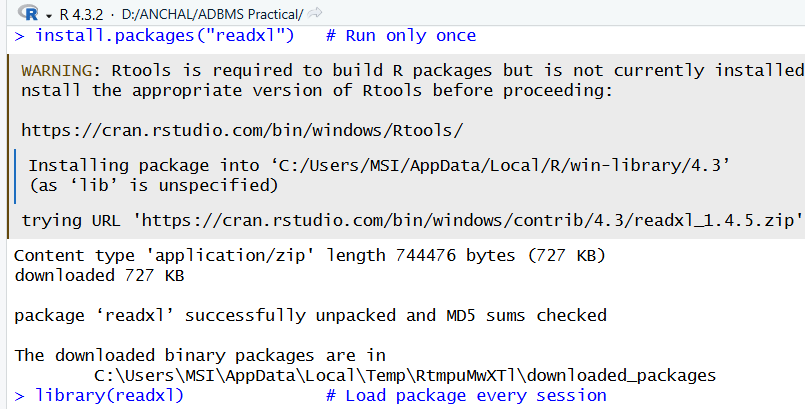
my\_data <- read\_excel("D:/ANCHAL/ADBMS Practical/tables.xlsx"

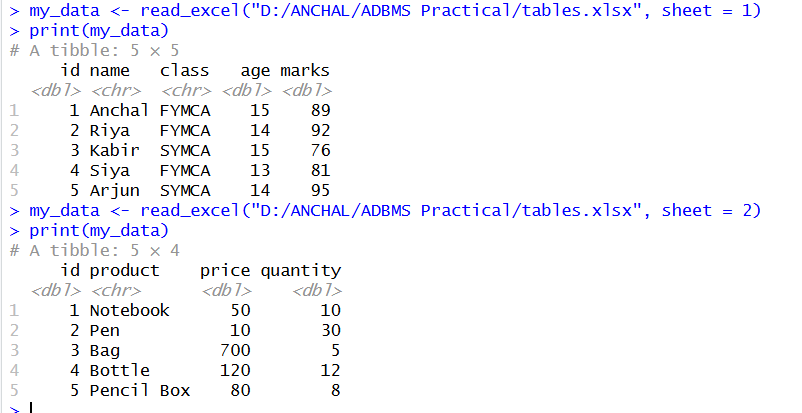
, sheet = 2) print(my\_data)

Input Sheet 1 Input Sheet 2

**Output:**

****



**2) Write a program to write data into an excel file.**

install.packages("writexl")

# Create vectors

student\_id <- c(3041601, 3041602, 3041603, 3041604, 3041605)

student\_name <- c("Anchal", "Miller", "Aman", "Smith", "Smith")

student\_class <- c("FYMCA", "SYMCA", "FYMCA", "SYMCA", "FYMCA")

student\_age <- c(20, 21, 20, 21, 22)

student\_position <- c("FIRST", "FIRST", "SECOND", "SECOND", "THIRD")

# Create a data frame

student\_details <- data.frame(

student\_id, student\_name, student\_class, student\_age, student\_position )

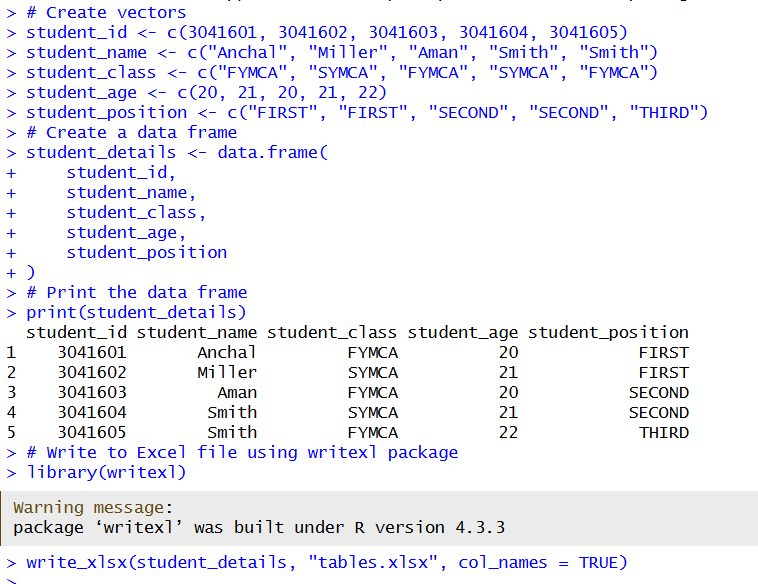
print(student\_details)

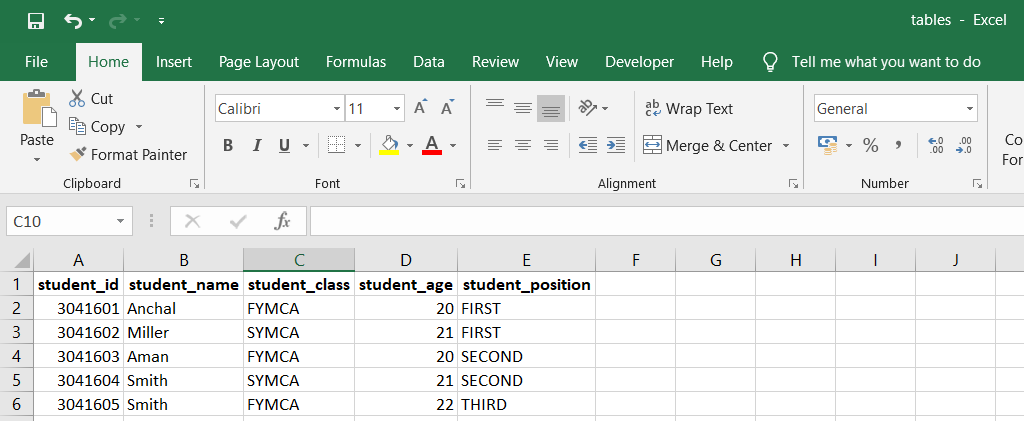
# Write to Excel file using writexl package

library(writexl)

write\_xlsx(student\_details, "tables.xlsx", col\_names = TRUE)

**Ouput:**

****



# PRACTICAL NO: 15

**Aim: Implementation of data preprocessing using R.**

1. **Imputation**

## The process of estimating or deriving missing value

install.packages("Hmisc")

library(Hmisc)

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

X <- impute(x, fun = mean)

X

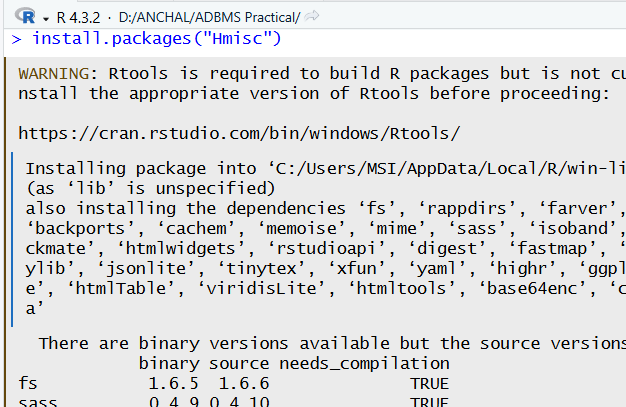
Y <- impute(x, fun = median)

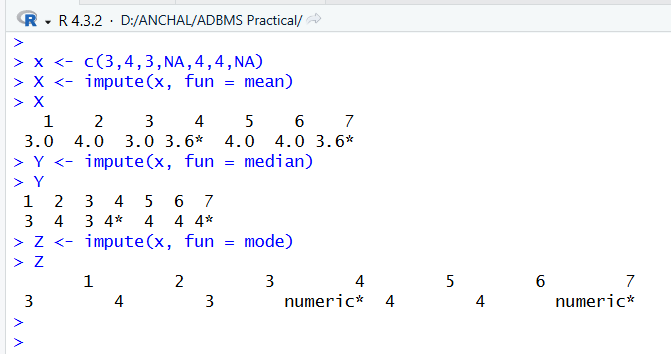
Y

Z <- impute(x, fun = mode)

Z

**Output**

****



1. **Categorical data**

## Factors are variables in R which take on a class

gender\_vector<-c("Male","Female","Female","Male")

class(gender\_vector)

factor\_gender\_vector <- factor(gender\_vector)

class(factor\_gender\_vector)

timing <- c("Morning", "Evening", "Afternoon", "Night", "Evening", "Afternon", "Midnight", "Midnight")

factor\_timing <- factor(timing, order=TRUE, levels =

c("Morning","Afternoon","Evening","Night","Midnight"))

factor\_timing

age <- c(21,32,34,65,32,34,22)

salary <- c(12000, 32000, 20000, 23000, 50000, 35000, 40000)

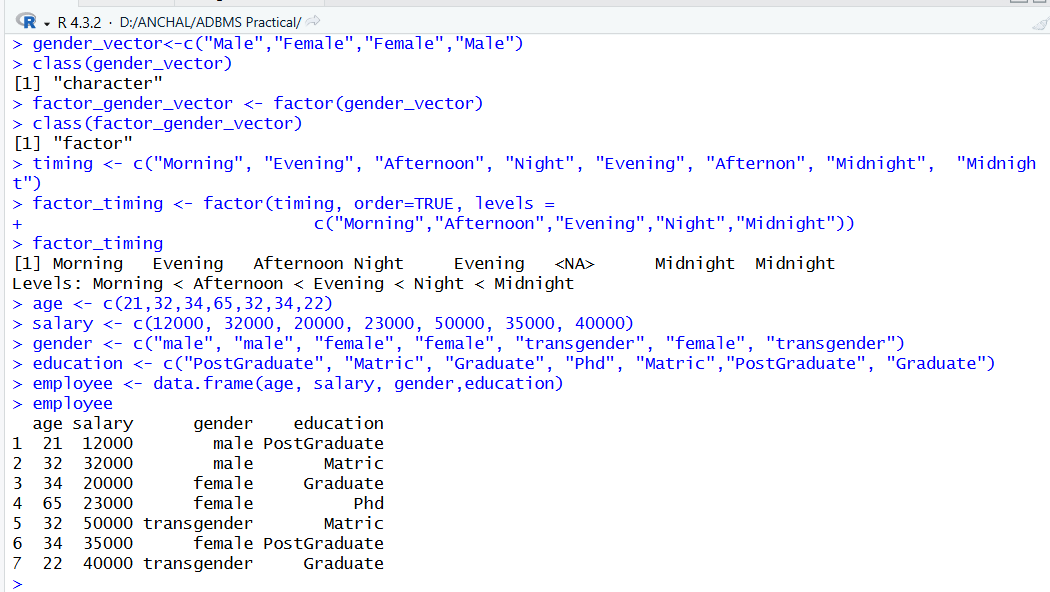
gender <- c("male", "male", "female", "female", "transgender", "female", "transgender") education <- c("PostGraduate", "Matric", "Graduate", "Phd", "Matric",

"PostGraduate", "Graduate")

employee <- data.frame(age, salary, gender,education)

employee

**Output**

****

1. **Creating a factor corresponding to age with table**

efact <- cut(employee$age, 3, labels = c("Young", "Medium", "Aged"))

table(efact)

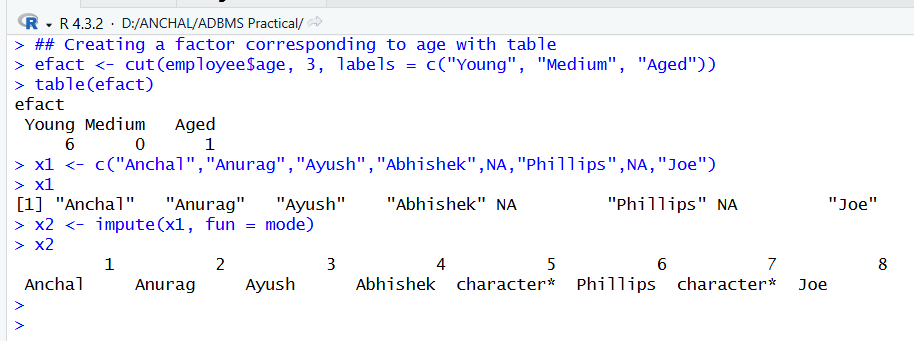
x1 <- c("Anchal","Anurag","Ayush","Abhishek",NA,"Phillips",NA,"Joe")

x1

x2 <- impute(x1, fun = mode)

x2

**Output**

****

1. **simple regression**

x<-c(151, 174, 138, 186, 128, 179, 163, 152, 131)

y<-c(63, 81, 56, 91, 47, 57, 76, 72, 62)

relation<-lm(y~x)

print(relation)

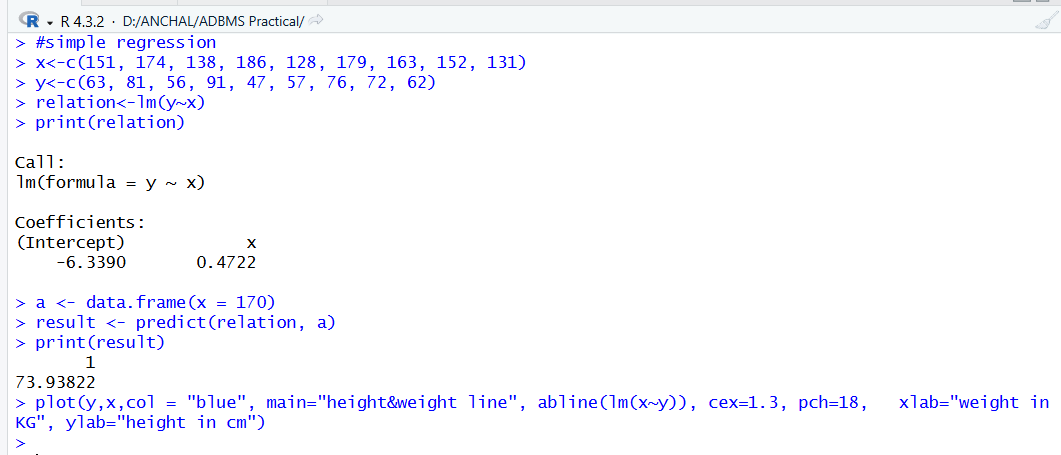
a <- data.frame(x = 170)

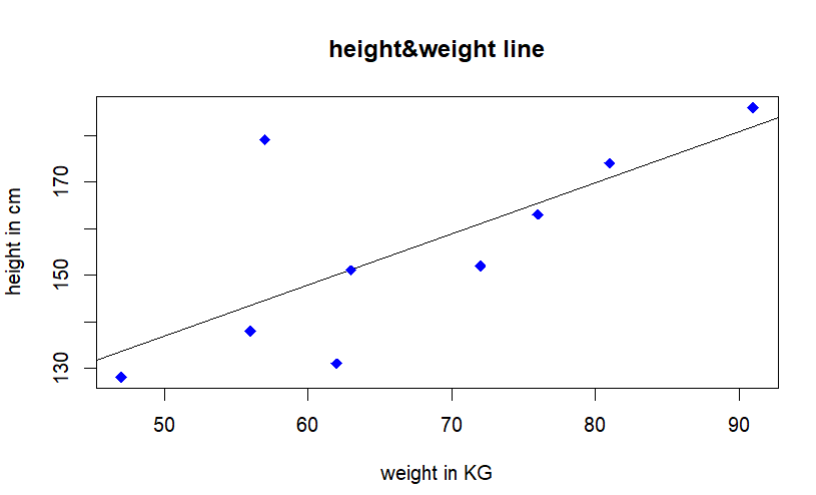
result <- predict(relation, a)

print(result)

plot(y,x,col = "blue", main="height&weight line", abline(lm(x~y)), cex=1.3, pch=18, xlab="weight in KG", ylab="height in cm")

**Output**

****



# PRACTICAL NO: 16

**Aim: Implementation of Linear regression**

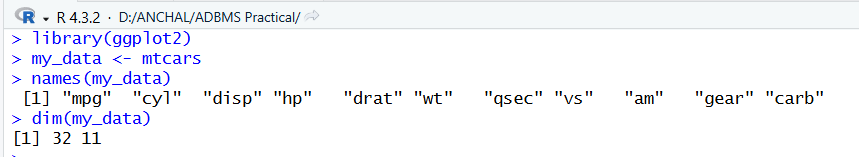
library(ggplot2)

my\_data <- mtcars

names(my\_data)

dim(my\_data)

**Output:**

****

1. **randomize**

my\_data <- my\_data[sample(nrow(my\_data), ), ]

head(my\_data)

TrainData<-my\_data[1:20, ]

TestData <- my\_data[21:32, ]

**Output**

****

1. **Linear Model**

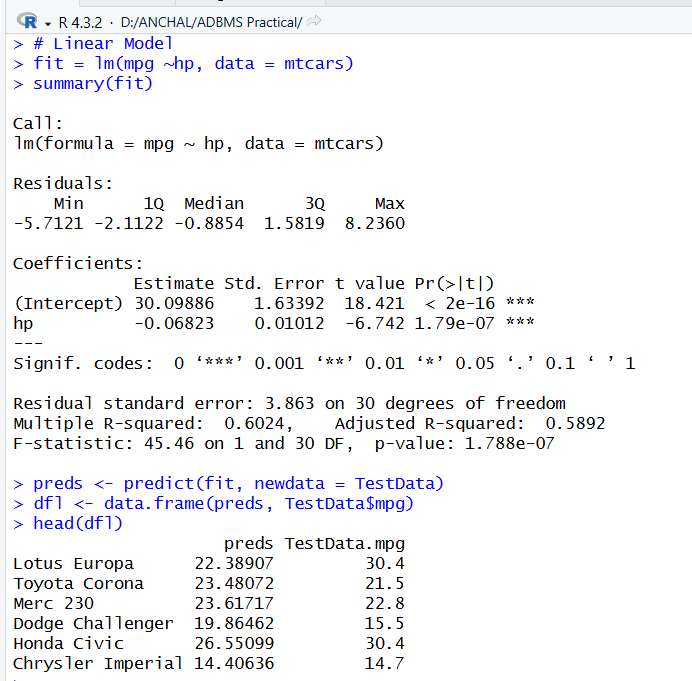
fit = lm(mpg ~hp, data = mtcars)

summary(fit)

preds <- predict(fit, newdata = TestData)

dfl <- data.frame(preds, TestData$mpg)

head(dfl)



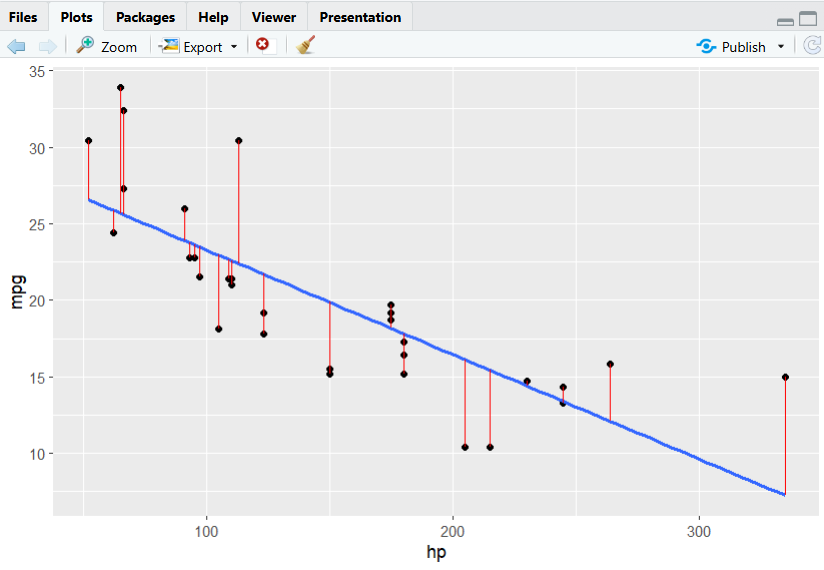
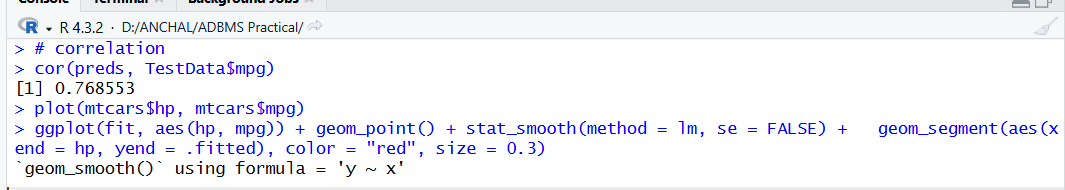
1. **correlation**

cor(preds, TestData$mpg)

plot(mtcars$hp, mtcars$mpg)

ggplot(fit, aes(hp, mpg)) + geom\_point() + stat\_smooth(method = lm, se = FALSE) + geom\_segment(aes(xend = hp, yend = .fitted), color = "red", size = 0.3)

**Output**

****

**PRACTICAL NO: 17**

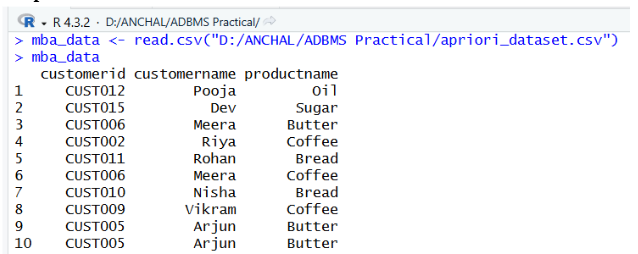
**Aim: Implementation of Apriori Algorithms using Market Basket.**

**Code:**

mba\_data <- read.csv("D:/ANCHAL/ADBMS Practical/apriori\_dataset.csv")

mba\_data

**Output:**

****

**Code:**

trans <- split (mba\_data$productname, mba\_data$customerid, "transactions")

head(trans)

**Output:**

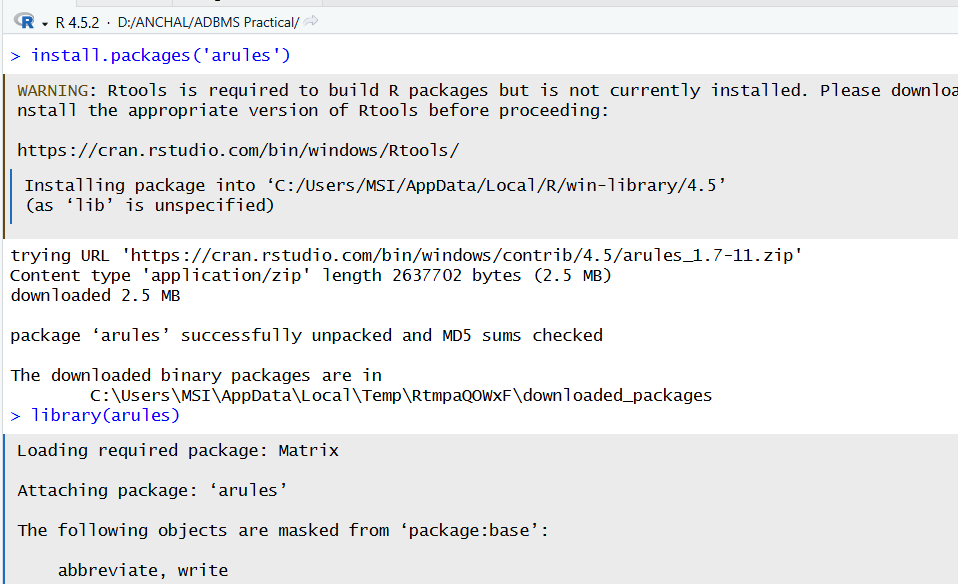
****

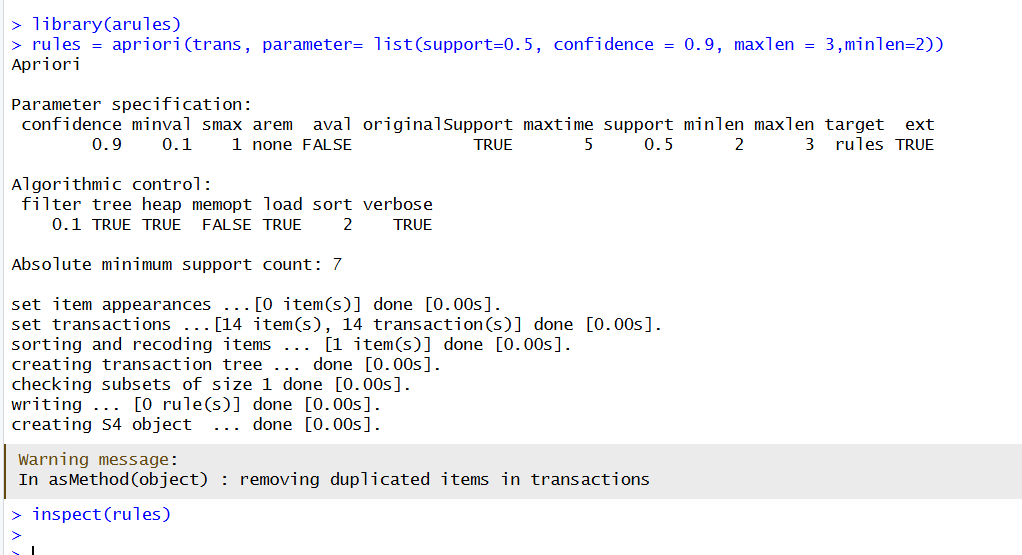
**Code:**

library(arules)

rules = apriori(trans, parameter= list(support=0.5, confidence = 0.9, maxlen = 3,minlen=2)) inspect(rules)

**Output**

****

****

# 

# PRACTICAL NO: 18

**Aim: Implementation of K-nearest neighbor Classification**

iris\_train<-iris\_norm[ran,]

 From a normalized dataset **iris\_norm**, this selects the rows whose indices are in ran.

 These rows make up the **training dataset** for k-NN.

iiris\_test<-iris\_norm[-ran,]

 A likely typo: should be **iris\_test**, not iiris\_test.

 This selects all rows **not** in ran, creating the **test dataset**.

iris\_target\_category<-iris[ran,5]

 Takes the **original (non-normalized)** iris dataset and extracts column 5 (**Species**) only for the training rows.

 These are your **training labels**.

iris\_test\_category<-iris[ran,5]

You are incorrectly taking the **training labels again**, not the test labels.

library(class)

Loads the **class** package, which contains the knn() function.

pr<-knn(iris\_train,iris\_test,cl=iris\_target\_category,k=13)

 Runs **k-nearest neighbors classification**.

 iris\_train → training features

 iris\_test → test features

 cl = iris\_target\_category → the true class labels for training data

 k = 13 → uses the 13 nearest neighbors to classify each test point

 Output pr contains predicted species for each test sample.

tab<-table(pr,iris\_test\_category)

 Creates a **confusion matrix** comparing:

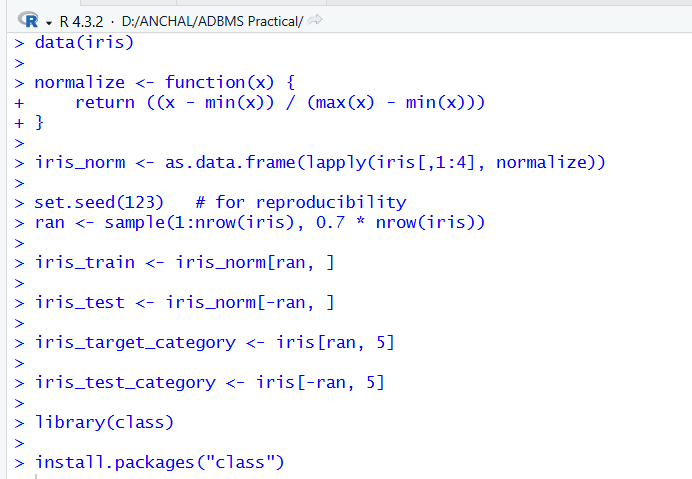
* pr = predicted species
* iris\_test\_category = true test species

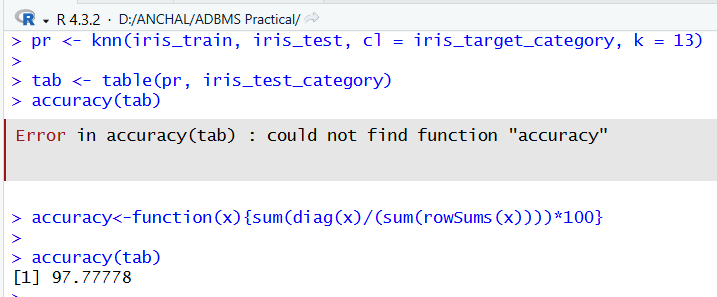
 This table shows how many predictions were correct or incorrect.

accuracy<-function(x){sum(diag(x)/(sum(rowSums(x))))\*100}

Accuracy=Correct predictionsTotal samples×100\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total samples}} \times 100Accuracy=Total samplesCorrect predictions​×100

accuracy(tab)





# PRACTICAL NO: 19

**Aim: Implementation of Naive Bayesian Classification**

**install.packages("caret", dependencies = TRUE)**

library(caret)

library(e1071)

set.seed(123)

index <- sample(nrow(iris), floor(nrow(iris) \* 0.7))

train <- iris[index, ]

test <- iris[-index, ]

xTrain <- train[, -5]

yTrain <- train$Species

xTest <- test[, -5]

yTest <- test$Species

model <- train(

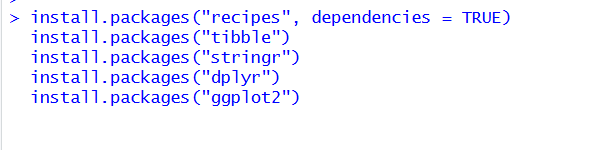
xTrain, yTrain,

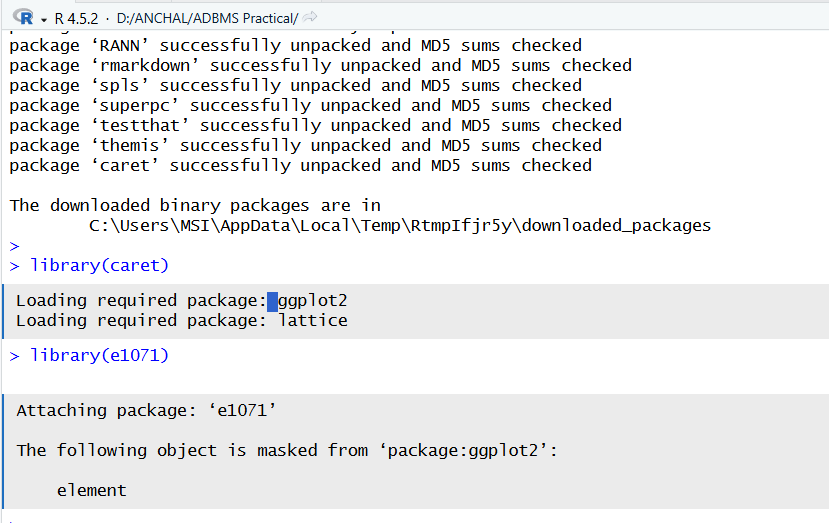
method = "nb",

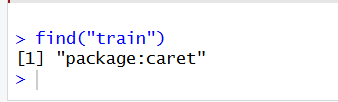
trControl = trainControl(method = "cv", number = 10)

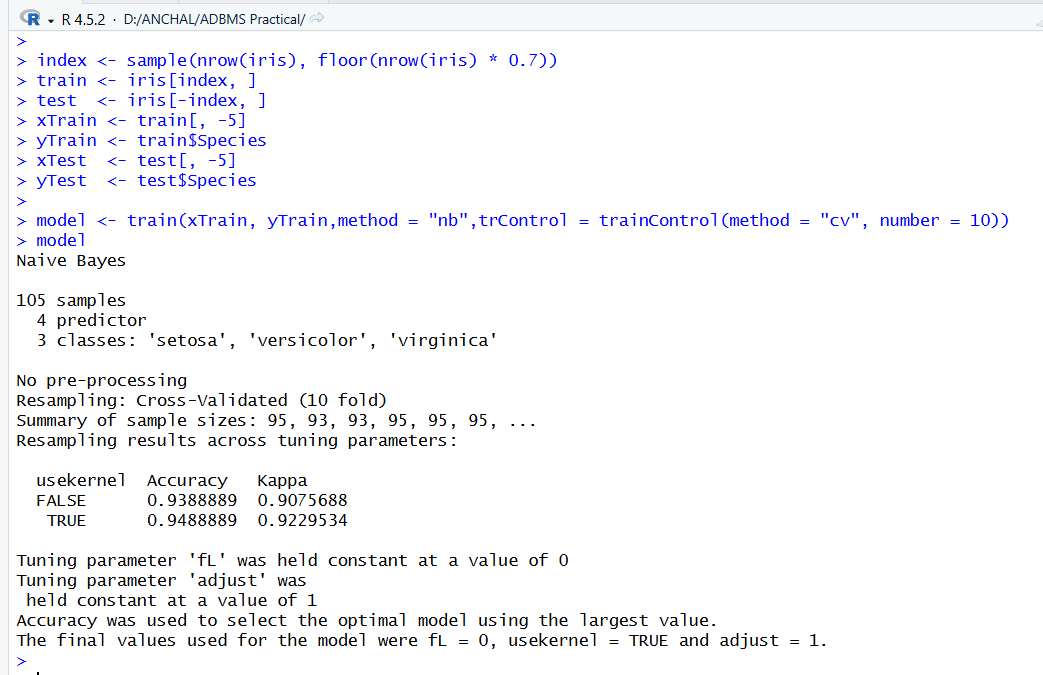
)

Model



****

****

****

# PRACTICAL NO: 20

**Aim: Implementation of K-means Classification Algorithm**

head(iris)

library(ggplot2)

ggplot(iris, aes(Petal.Length,Petal.Width,colour = Species))+geom\_point()

set.seed(20)

irisCluster <- kmeans(iris[,3:4],3, nstart = 20)

irisCluster

table(irisCluster$cluster,iris$Species)

irisCluster$cluster <- as.factor(irisCluster$cluster)

ggplot(iris, aes(Petal.Length,Petal.Width,colour = irisCluster$cluster)) +geom\_point()

