SHARDA SCHOOL OF BASIC SCIENCES AND RESEARCH

Department of Mathematics

LAB REPORT FILE

Course Title: Data Cleaning lab

Course Code: BDA 253

Program-Bachelor of Science (Hons.)-Data Science

Semester: 4th

Session 2022-2023

Student Name: - Manish kumar

System ID: - 2021509877

Roll No. - 2107158014

Submitted to

Dr. Parul Saini

(Assistant Professor)

Practical - 1

❖ Aim:

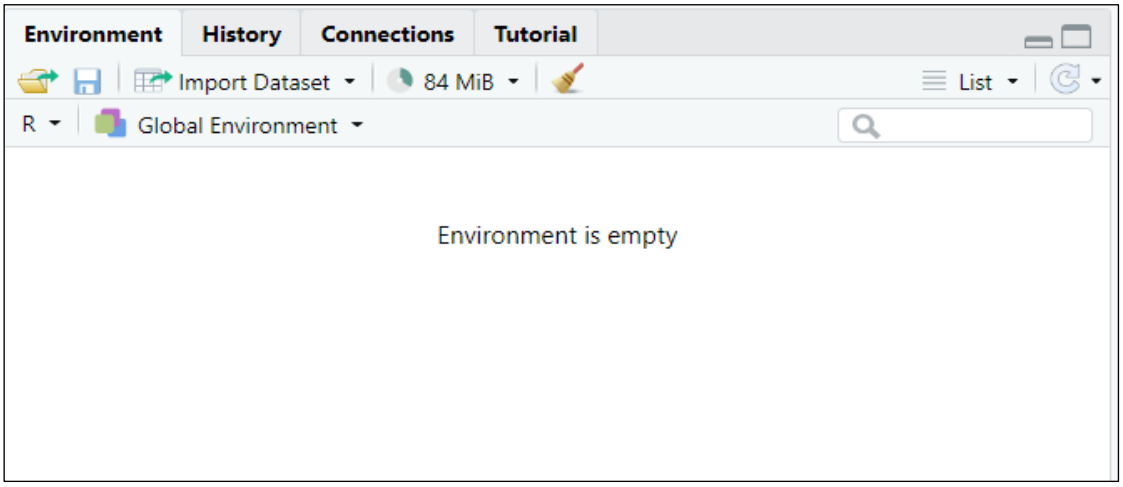
How to import data from different sources in R.

❖ Theory:

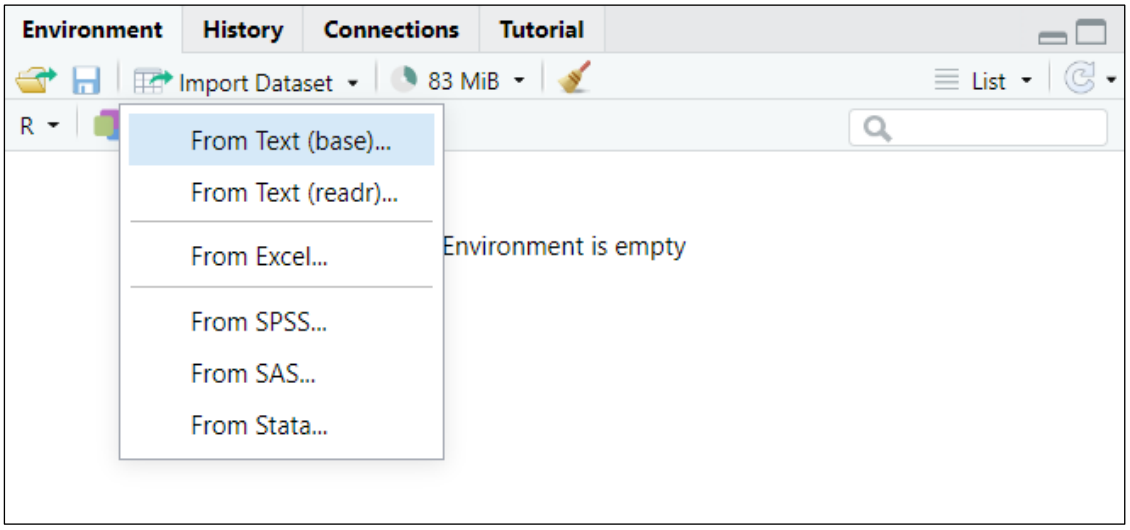
R is capable of reading data from most formats, including files created in other statistical packages. Whether the data was prepared using Excel (in CSV, XLSX, or TXT format), SAS, Stata, SPSS, or others, R can read and load the data into memory.

❖ Steps of importing a csv file:

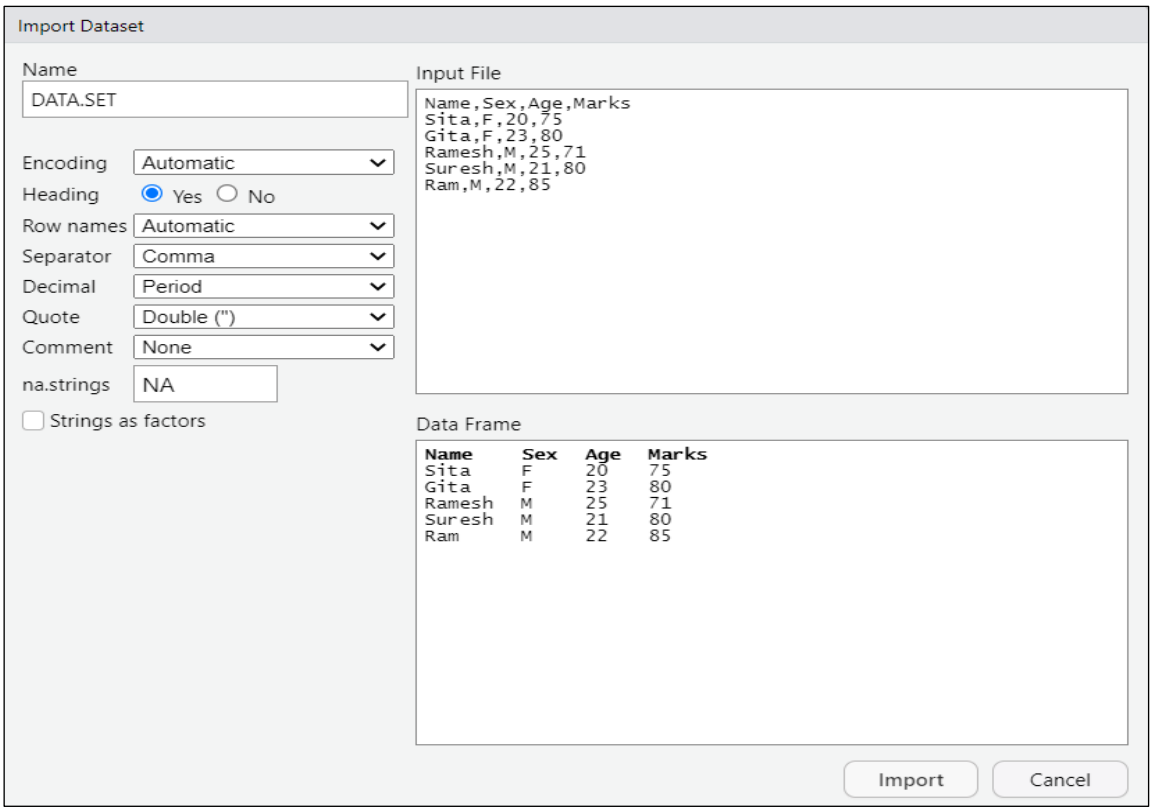
1. Click on the **'import'** button in the right window of R studio.



2. A pop-down menu will open. Choose the first option (From text(base))

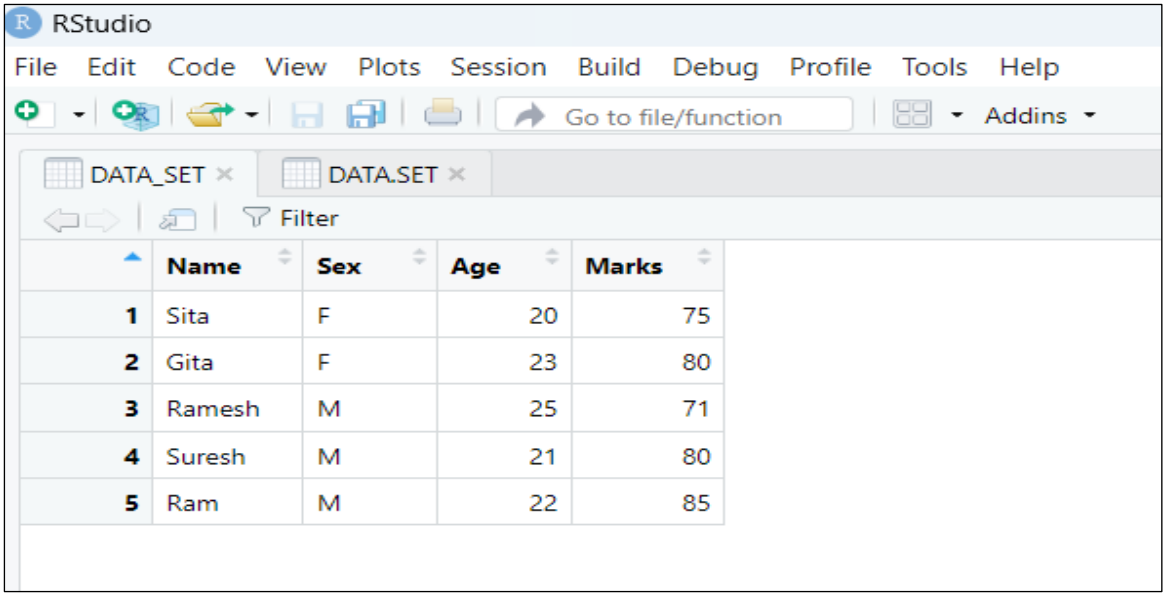


3. This will open another window enabling you to browse your computer to locate the file you want to import. Double click on the file, and another window will open as show below.



4. Click on the **Import** button at the bottom of this window and the CSV file is successfully imported in the R studio as follows.

❖ Output:

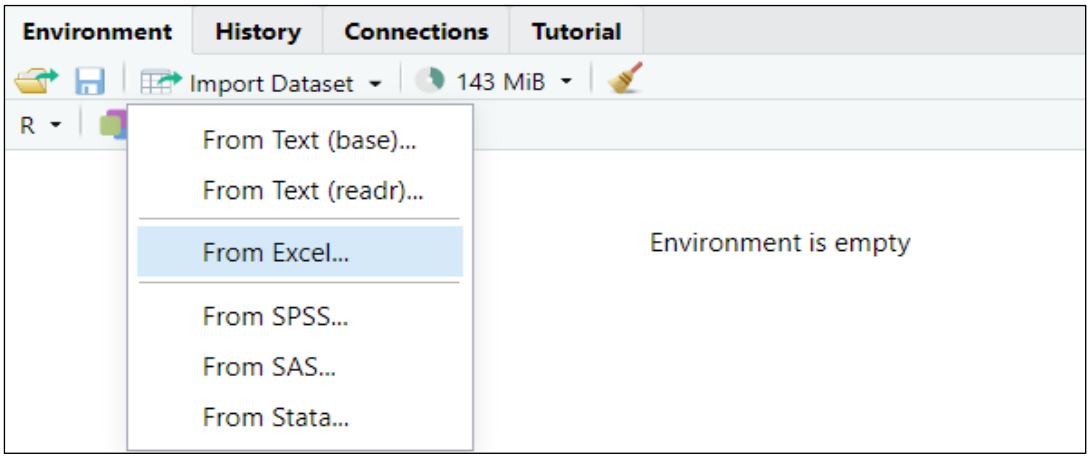


	Name	Sex	Age	Marks
1	Sita	F	20	75
2	Gita	F	23	80
3	Ramesh	M	25	71
4	Suresh	M	21	80
5	Ram	M	22	85

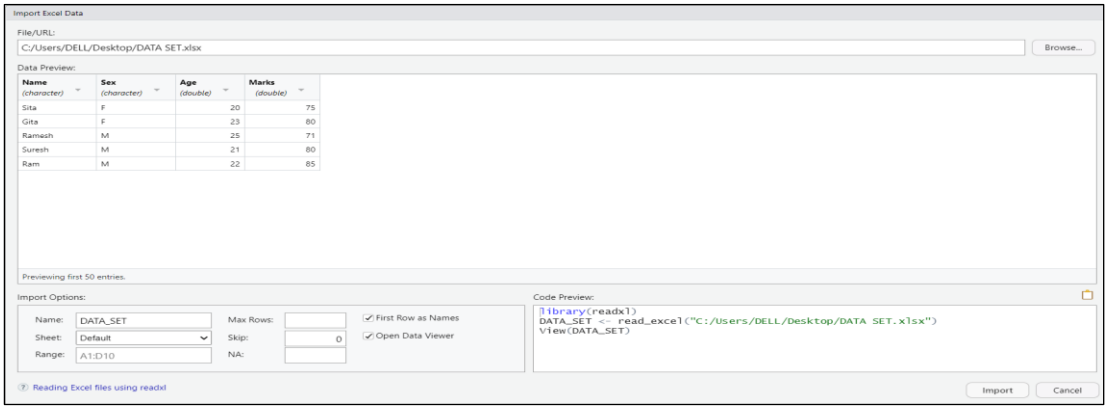
❖ Steps of importing a Excel file:

I. Click on the **'import'** button in the right window of R studio

II. A pop-down menu will open. Choose the first option From Excel.

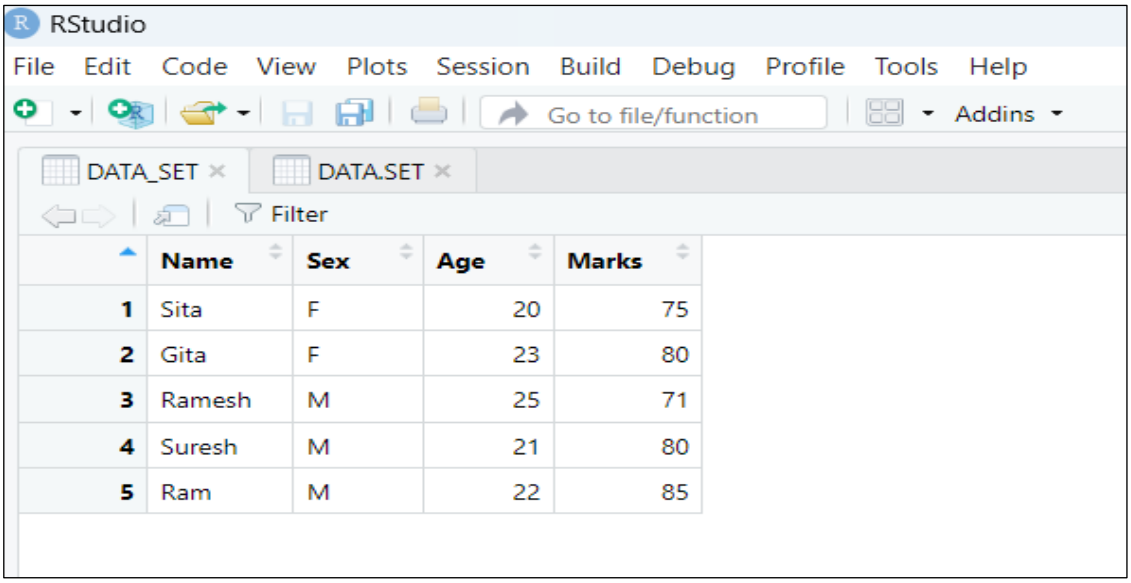


III. This will open another window enabling you to browse your computer to locate the file you want to import. Double click on the file, and another window will open as show below.



IV. Click on the **Import** button at the bottom of this window and the EXCEL file is successfully imported in the R studio as follows.

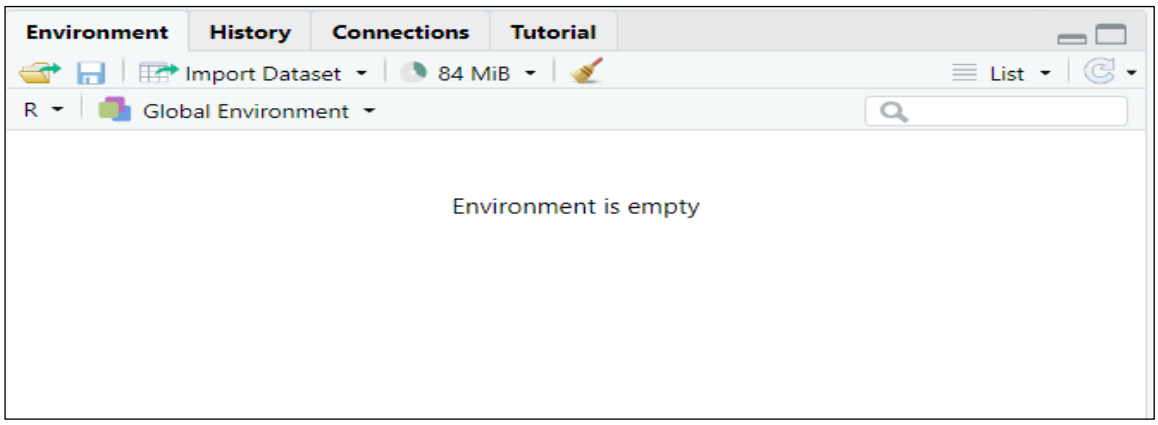
❖ OUTPUT:



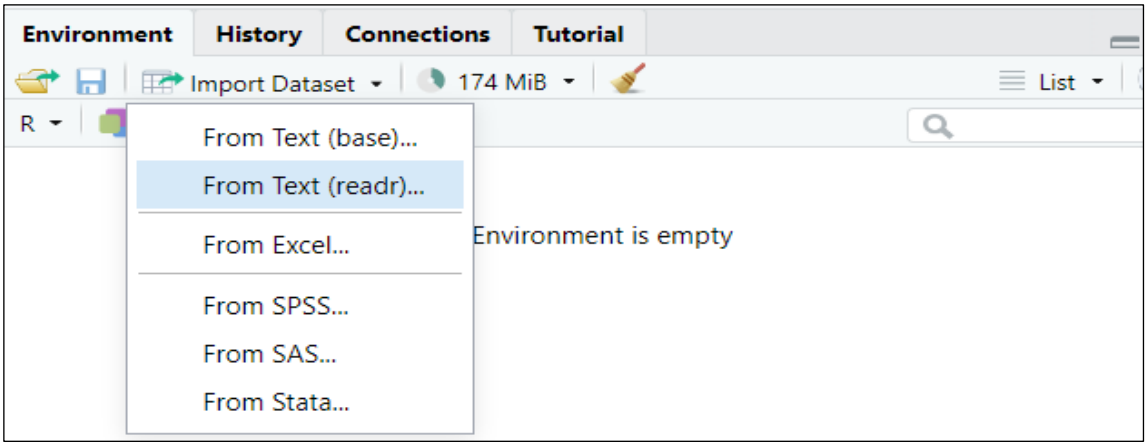
	Name	Sex	Age	Marks
1	Sita	F	20	75
2	Gita	F	23	80
3	Ramesh	M	25	71
4	Suresh	M	21	80
5	Ram	M	22	85

❖ Steps of importing a TEXT file:

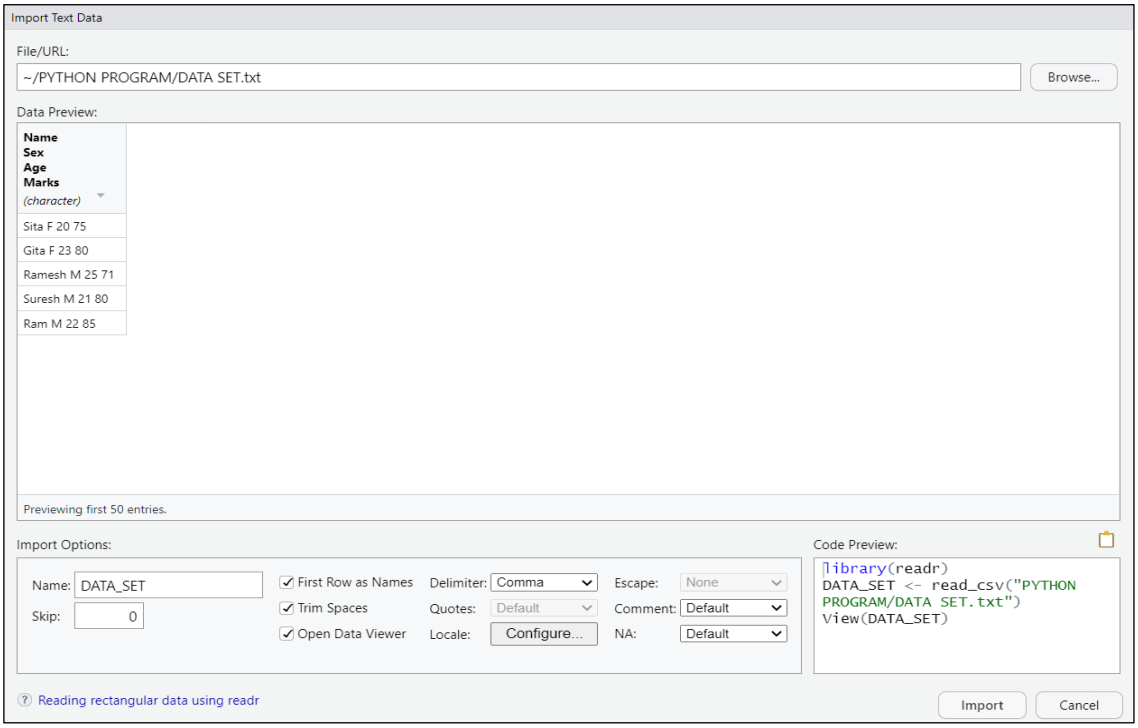
1) Click on the **'import'** button in the right window of R studio.



2) A pop-down menu will open. Choose the first option (From text(readr)).

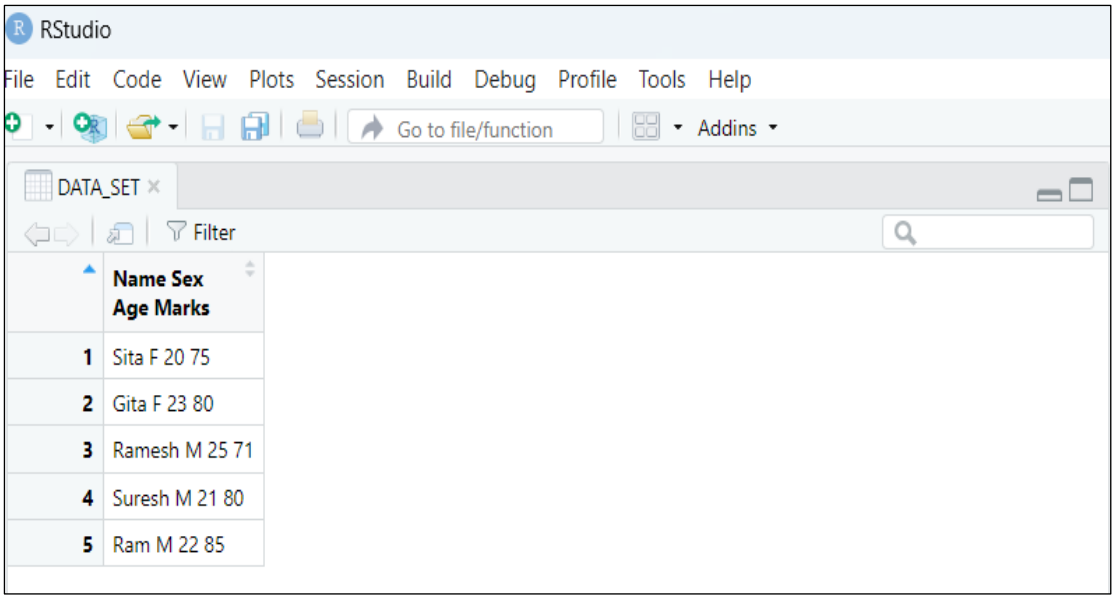


3) This will open another window enabling you to browse your computer to locate the file you want to import. Double click on the file, and another window will open as show below.



4) Click on the **Import** button at the bottom of this window and the CSV file is successfully imported in the R studio as follows.

❖ Output:



Practical-2

❖ Aim:

How to enter matrix through different command in R with their use.

❖ Theory:

A matrix is a 2-D data set with columns and rows. A column is a vertical representation of data, while a row is a horizontal representation of data.

Syntax

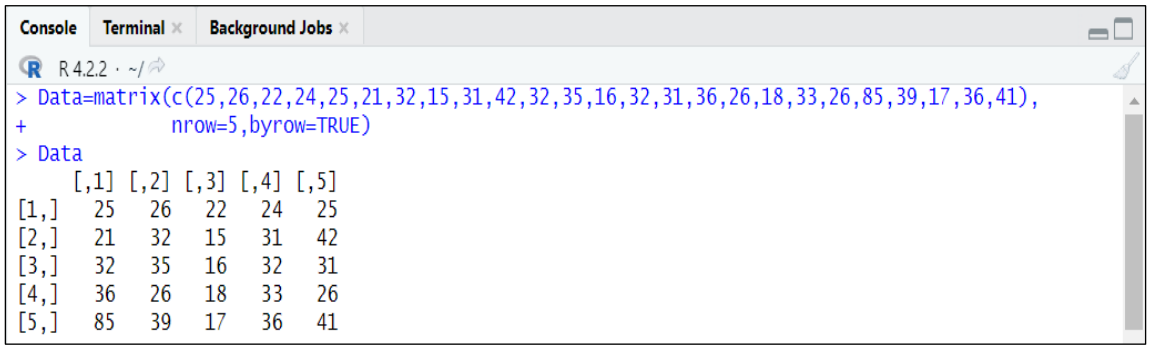
The basic syntax for creating a matrix in R is −

Matrix (data, nrow, ncol, byrow, dimnames)

❖ Creating a Matrix in R:

- To create a matrix in R you need to use the function called **matrix ()**.

❖ Output:



❖ Concatenation of a Row wise:

- The concatenation of a row to a matrix is done using **rbind ()**.

❖ Output:

❖ Concatenation of a column wise:

- ➢ The concatenation of a column to a matrix is done using **cbind()**

❖ Output:

```
Console  Terminal  Background Jobs
R 4.2.2 · ~/
> c1=c(25,21,32,34,85)
> c2=c(26,32,35,26,39)
> c3=c(22,15,16,18,17)
> c4=c(24,31,32,33,36)
> c5=c(25,42,31,26,41)
> cbind= cbind(c1,c2,c3,c4,c5)
> cbind
     c1 c2 c3 c4 c5
[1,] 25 26 22 24 25
[2,] 21 32 15 31 42
[3,] 32 35 16 32 31
[4,] 34 26 18 33 26
[5,] 85 39 17 36 41
>
```

❖ Create a inverse of Matrix in R:

- ➢ Using the solve () function:

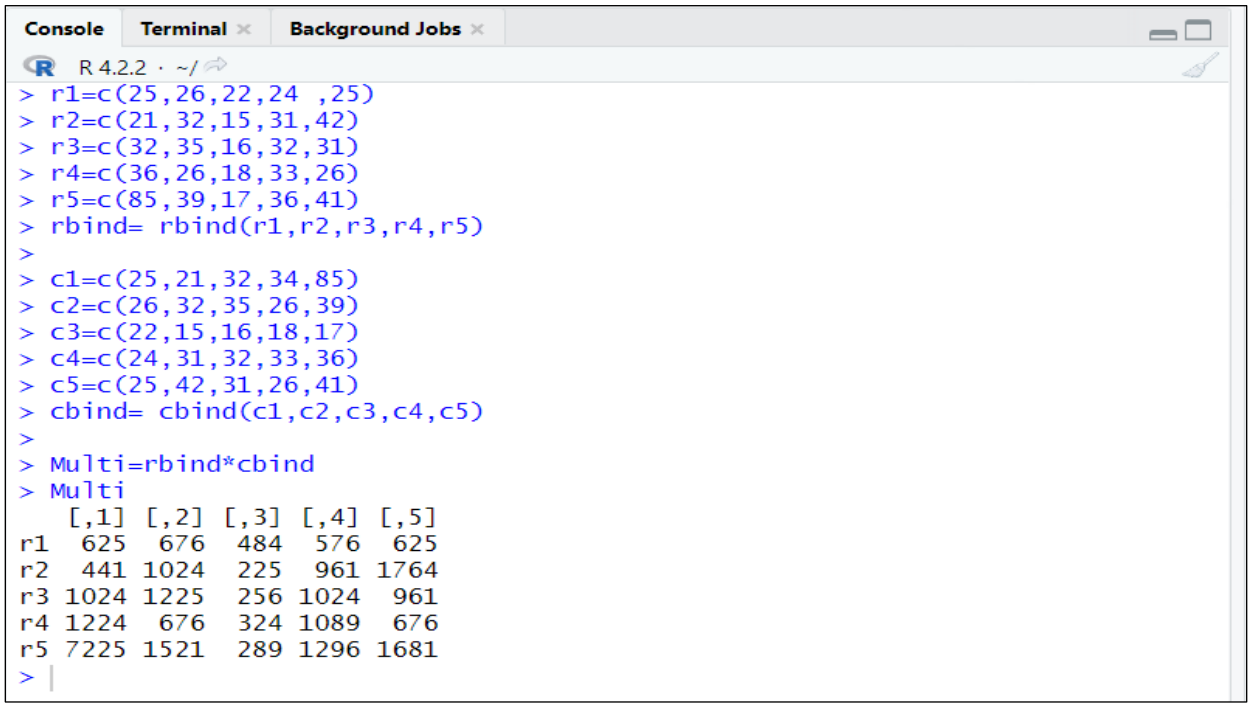
❖ Output:

```
Console  Terminal  Background Jobs
R 4.2.2 · ~/
> Data=matrix(c(25,26,22,24,25,21,32,15,31,42,32,35,16,32,31,36,26,18,33,26,85,39,17,36,41),
+             nrow=5,byrow=TRUE)
> Data
     [,1] [,2] [,3] [,4] [,5]
[1,]   25   26   22   24   25
[2,]   21   32   15   31   42
[3,]   32   35   16   32   31
[4,]   36   26   18   33   26
[5,]   85   39   17   36   41
>
> solve(Data)
             [,1]         [,2]          [,3]          [,4]         [,5]
[1,]  0.001238593 -0.009125311 -0.0138934602  1.006898e-03  0.018458931
[2,]  0.010981035 -0.044888596  0.1331179799 -9.072346e-02 -0.003830543
[3,]  0.098940180 -0.012156832 -0.0573775958  1.255916e-05 -0.004500942
[4,] -0.069420096 -0.001835369 -0.0003265381  1.113980e-01 -0.026186277
[5,]  0.006917065  0.068269417 -0.0737434779 -1.360763e-02  0.014624490
>
```

❖ Create a Multiplication of Matrix in R:

The multiplication operator * is used for multiplying a matrix by scalar or element-wise multiplication of two matrices.

❖ Output:



Practical-3

❖ Aim:

Create a dataframe and rearrange the columns.

1) A Column is Moved to the First Position
2) Move a Column to the Last Position
3) Rearrange Several Columns
4) Alphabetically reorder the columns
5) Reverse Column Order

❖ Theory:

Each vector will represent a DataFrame column, and the length of any vector will correspond to the number of rows in the new DataFrame. It's also possible to pass only one vector to the data. frame() function, in which case a DataFrame with a single column will be created.

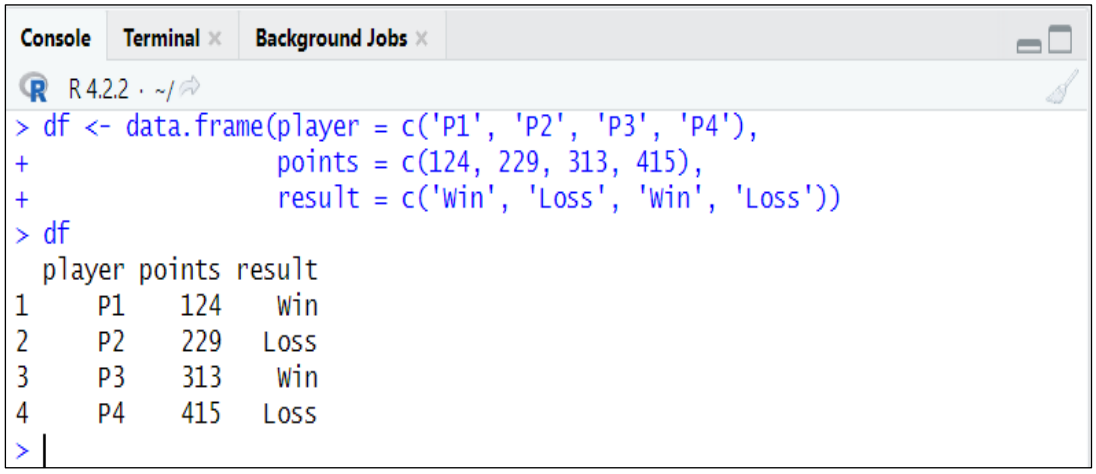
❖ Rearranging Columns in R:

Tips for Rearranging Columns in R, you might frequently want to reorder the columns in a data frame.

The **select()** function from the **dplyr package,** fortunately, makes this simple to accomplish.

library(dplyr)

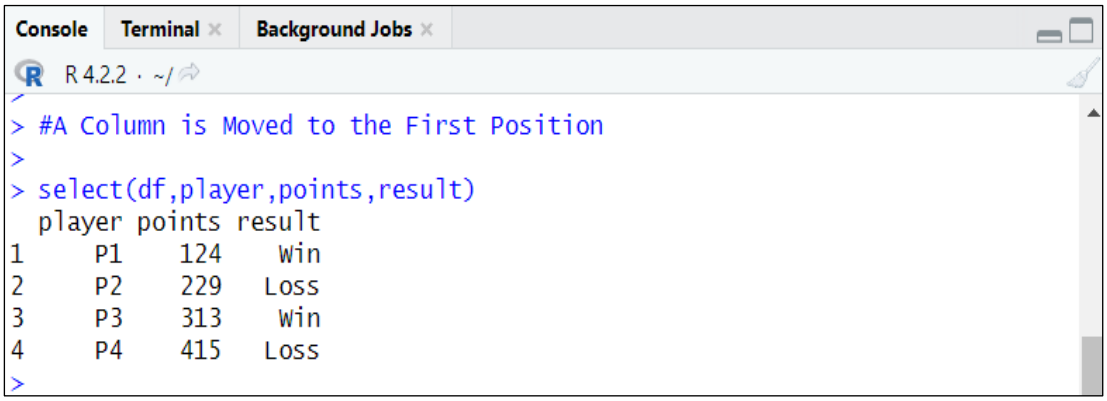
Create a data frame:



1. A Column is Moved to the First Position:

How to shift a particular column in a data frame to the front position is demonstrated by the code below.

❖ **Output:**

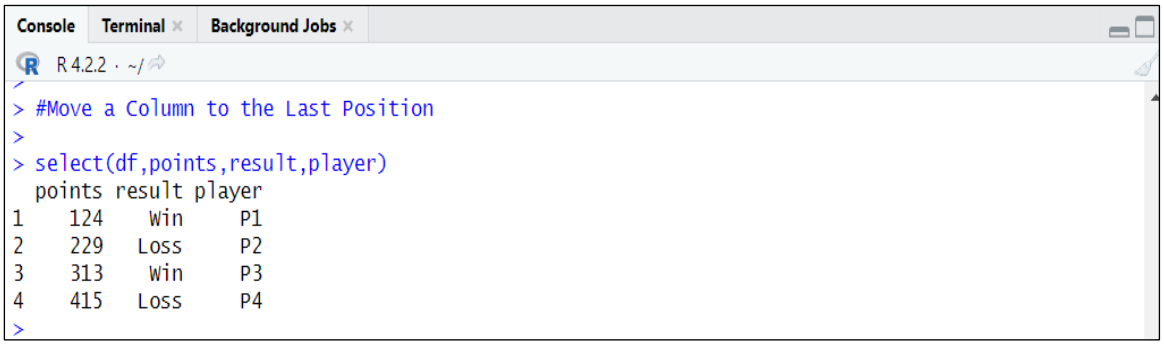


:

2. Move a Column to the Last Position:

How to relocate a particular column in a data frame to the last location is demonstrated by the code below:

❖ **OUTPUT:**



3. Rearrange Several Columns:

The code that follows demonstrates how to sequentially reorder multiple columns.

❖ **OUTPUT:**

4. Alphabetically reorder the columns:

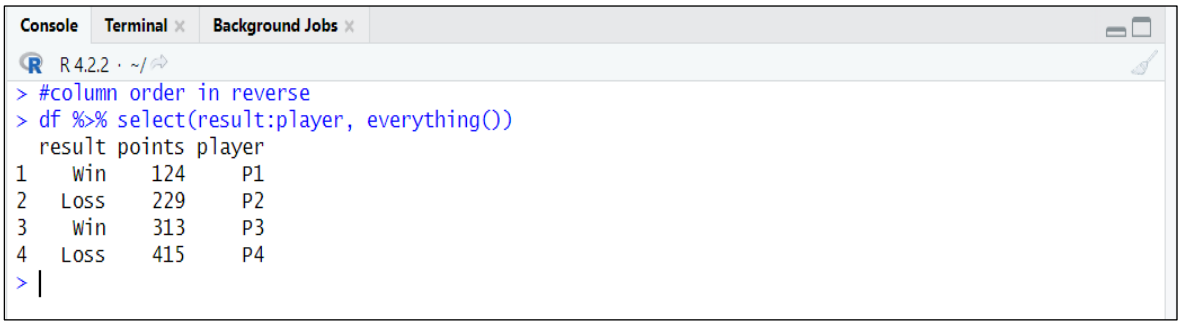
The code below demonstrates how to alphabetically arrange the columns. alphabetize the columns

❖ Output:

5. Reverse Column Order:

Reversing the column order in a data frame is demonstrated by the code below.

❖ Output:



PRACTICAL-4

❖ AIM:

Create a dataframe and remove the rows.

1) Remove any row with NA's.
2) Delete any rows that contain NA's in specific columns.
3) Rows that are duplicated should be removed.
4) Rows are removed based on their index position.
5) Rows are removed based on their condition.

❖ Steps:

❖ Remove any row with NA's

Code:

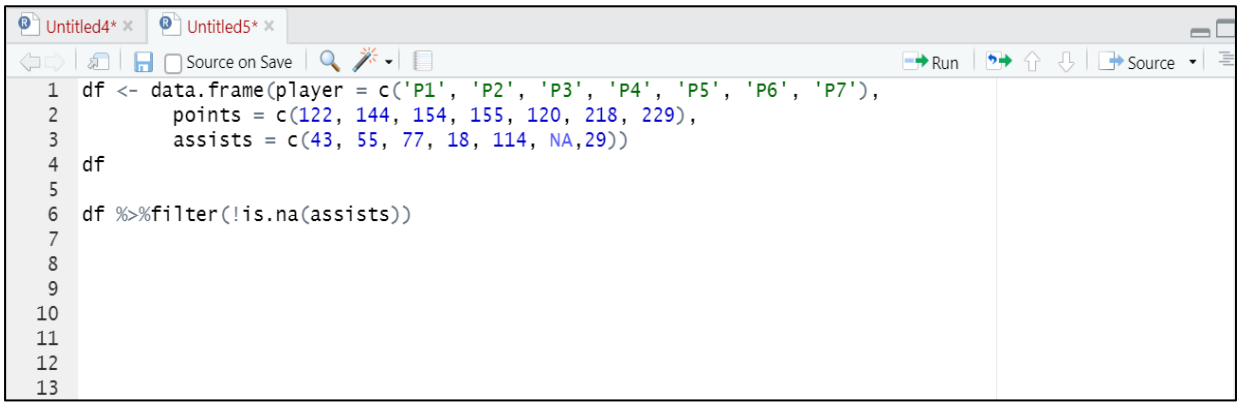
```
df <- data.frame(player = c('P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7'),
                 points = c(122, 144, 154, 155, 120, 218, 229),
                 assists = c(43, 55, 77, 18, 114, NA,29))
df

df %>%  na.omit()
```

Output:

```
> source("~/.active-rstudio-document", echo=TRUE)

> df <- data.frame(player = c('P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7'),
+                  points = c(122, 144, 154, 155, 120, 218, 229),
+                  assists =  .... [TRUNCATED]

> df
  player points assists
1     P1    122      43
2     P2    144      55
3     P3    154      77
4     P4    155      18
5     P5    120     114
6     P6    218      NA
7     P7    229      29

> df %>%  na.omit()
  player points assists
1     P1    122      43
2     P2    144      55
3     P3    154      77
4     P4    155      18
5     P5    120     114
7     P7    229      29
```

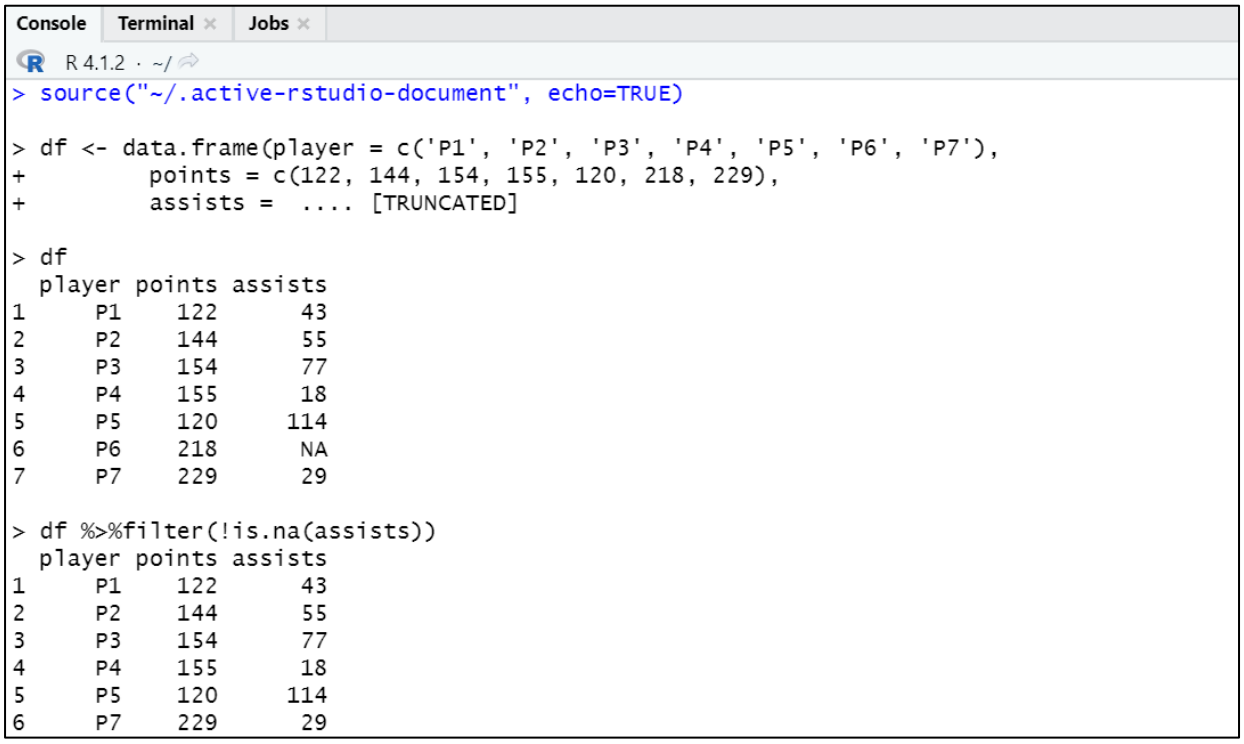
❖ Delete any rows that contain NA's in specific columns

Code:



```
Untitled4*  Untitled5*
Source on Save   Run   Source
1  df <- data.frame(player = c('P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7'),
2         points = c(122, 144, 154, 155, 120, 218, 229),
3         assists = c(43, 55, 77, 18, 114, NA,29))
4  df
5
6  df %>%filter(!is.na(assists))
7
8
9
10
11
12
13
```

Output:



```
Console  Terminal  Jobs
R 4.1.2 · ~/
> source("~/.active-rstudio-document", echo=TRUE)

> df <- data.frame(player = c('P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7'),
+         points = c(122, 144, 154, 155, 120, 218, 229),
+         assists =  .... [TRUNCATED]

> df
  player points assists
1     P1    122      43
2     P2    144      55
3     P3    154      77
4     P4    155      18
5     P5    120     114
6     P6    218      NA
7     P7    229      29

> df %>%filter(!is.na(assists))
  player points assists
1     P1    122      43
2     P2    144      55
3     P3    154      77
4     P4    155      18
5     P5    120     114
6     P7    229      29
```

❖ **Rows that are duplicated should be removed**

Code:

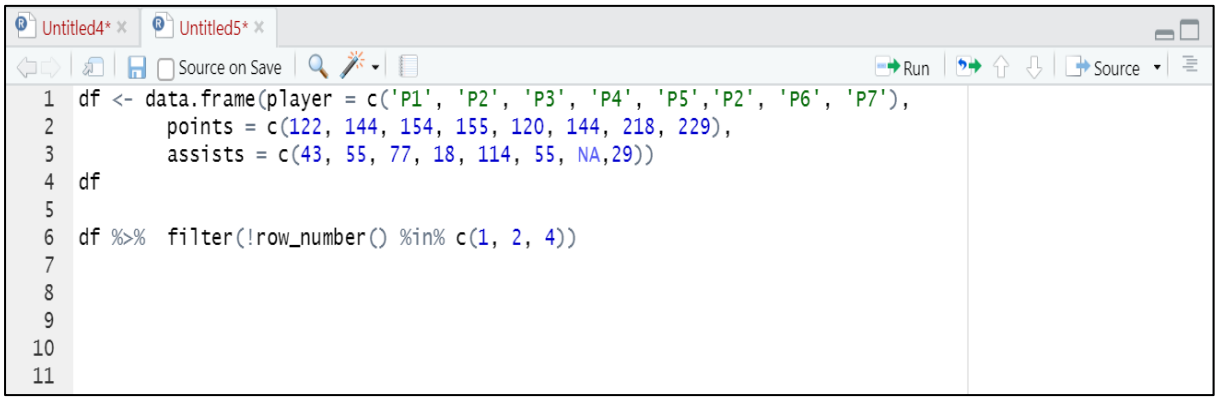
```
df <- data.frame(player = c('P1', 'P2', 'P3', 'P4', 'P5','P2', 'P6', 'P7'),
        points = c(122, 144, 154, 155, 120, 144, 218, 229),
        assists = c(43, 55, 77, 18, 114, 55, NA,29))
df

df %>% distinct()
```

Output:

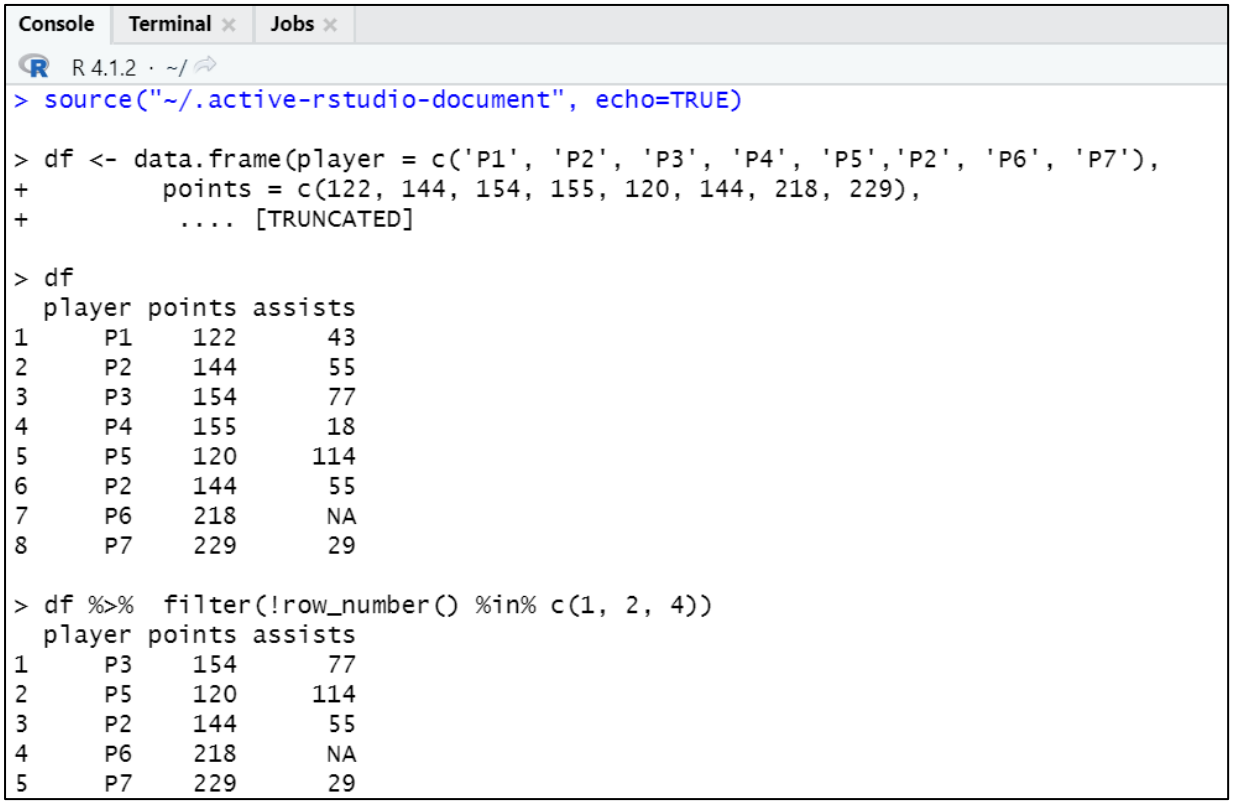
```
> source("~/.active-rstudio-document", echo=TRUE)

> df <- data.frame(player = c('P1', 'P2', 'P3', 'P4', 'P5','P2', 'P6', 'P7'),
+         points = c(122, 144, 154, 155, 120, 144, 218, 229),
+          .... [TRUNCATED]

> df
  player points assists
1     P1    122      43
2     P2    144      55
3     P3    154      77
4     P4    155      18
5     P5    120     114
6     P2    144      55
7     P6    218      NA
8     P7    229      29

> df %>% distinct()
  player points assists
1     P1    122      43
2     P2    144      55
3     P3    154      77
4     P4    155      18
5     P5    120     114
6     P6    218      NA
7     P7    229      29
```

❖ **Rows are removed based on their index position**

Code:



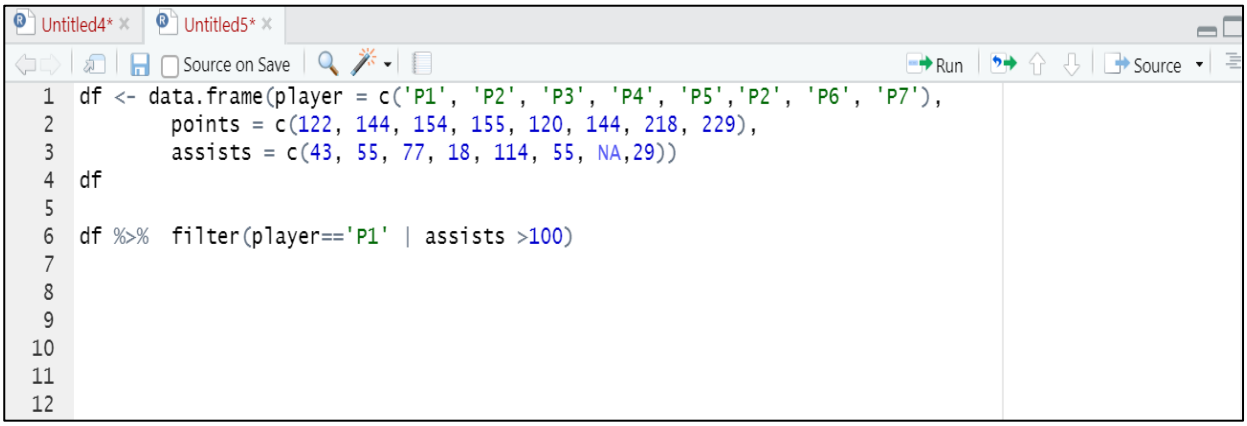
Output:



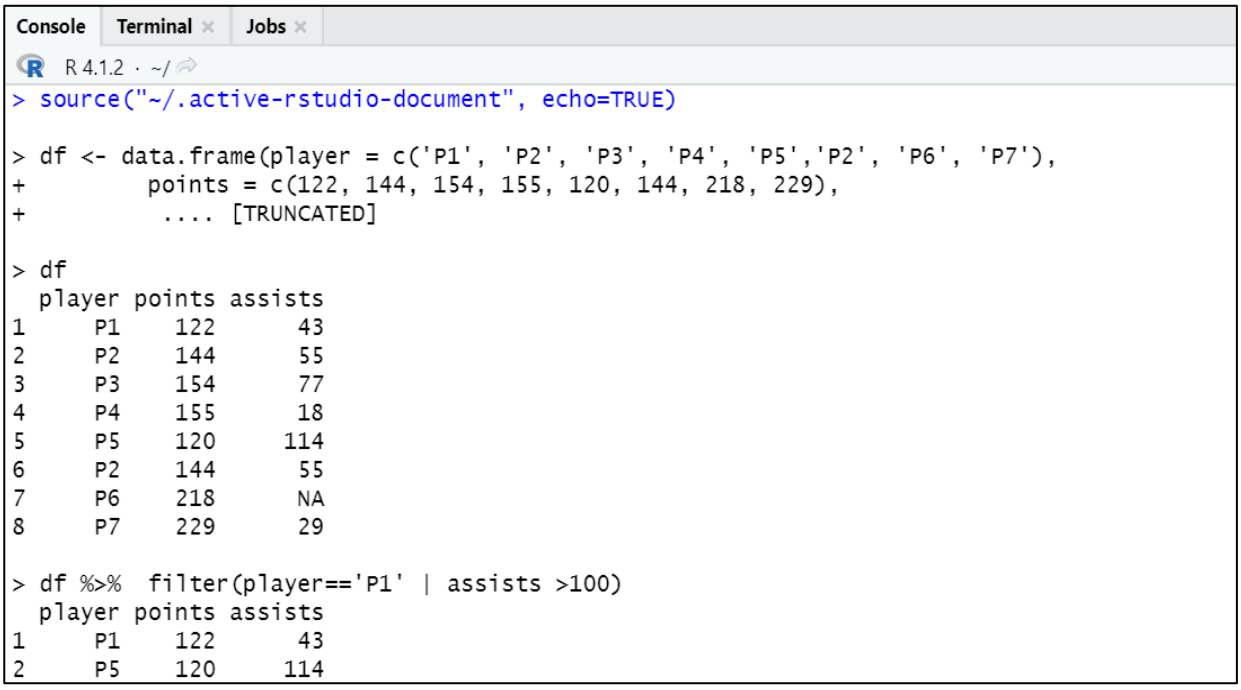
❖ Rows are removed based on their condition

The code below demonstrates how to eliminate rows based on certain criteria.

Code:



Output:



Practical-5

❖ Aim:

Create a random sample whit the help of distribution of 100 observations where minimum value is 2 and maximum value is 5 than categorize the data into class interval and calculate the frequency.

❖ Theory:

The continuous uniform distribution is also referred to as the probability distribution of any random number selection from the continuous interval defined between intervals a and b. A uniform distribution holds the same probability for the entire interval. Thus, its plot is a rectangle, and therefore it is often referred to as Rectangular distribution. Here we will discuss various functions and cases in which these functions should be used to get a required probability.

For uniform distribution, we first need a randomly created sequence ranging between two numbers. The **runif()** function in R programming language is used to generate a sequence of random following the uniform distribution.

❖ Steps:

❖ Code:

X = runif(100,min = 2,max=5)

X

❖ Output:

```
Console   Terminal   Background Jobs
R 4.2.2 · ~/
> x=runif(100,min = 2,max=5)
> x
  [1] 4.229366 3.648670 2.470825 4.697780 3.665430 3.979671 2.590638 3.769937 3.622669 3.010770
 [11] 2.103855 2.447682 2.569629 3.234095 2.832481 4.180026 2.061652 3.052620 2.741426 4.036524
 [21] 3.469464 4.125786 3.882981 2.045550 2.876360 4.603257 3.228292 4.443260 2.511440 4.717443
 [31] 4.611144 3.791045 3.152134 4.336543 4.975306 4.417326 3.650589 4.446128 4.128002 2.410453
 [41] 4.076416 3.490275 4.985402 3.736638 2.434505 4.609148 3.390100 4.707212 4.642082 3.176952
 [51] 3.659680 4.638265 4.465851 4.640360 4.433190 4.847820 2.429830 2.034587 4.171535 2.183667
 [61] 3.876159 3.538624 3.436538 2.476033 2.698864 2.938936 2.544906 2.735065 4.468965 4.909216
 [71] 3.999326 2.516241 3.862779 3.142486 3.295650 2.469510 3.788727 2.475304 3.145866 3.399584
 [81] 3.131395 3.974765 2.710018 3.315854 2.216899 2.489713 4.378430 3.350518 2.883106 2.355461
 [91] 4.274276 4.929096 2.774339 2.301527 3.252839 2.029658 4.922699 4.805254 3.370783 3.711839
```

❖ Code:

Z = cut(x,breaks=c(1,2,3,4,5))

Z

❖ Output:

```
Console   Terminal   Background Jobs
R 4.2.2 · ~/
> z=cut(x,breaks=c(1,2,3,4,5))
> z
  [1] (4,5] (3,4] (2,3] (4,5] (3,4] (3,4] (2,3] (3,4] (3,4] (3,4] (2,3] (2,3] (2,3] (3,4] (2,3]
 [16] (4,5] (2,3] (3,4] (2,3] (4,5] (3,4] (4,5] (3,4] (2,3] (2,3] (4,5] (3,4] (4,5] (2,3] (4,5]
 [31] (4,5] (3,4] (3,4] (4,5] (4,5] (4,5] (3,4] (4,5] (4,5] (2,3] (4,5] (3,4] (4,5] (3,4] (2,3]
 [46] (4,5] (3,4] (4,5] (4,5] (3,4] (3,4] (4,5] (4,5] (4,5] (4,5] (4,5] (2,3] (2,3] (4,5] (2,3]
 [61] (3,4] (3,4] (3,4] (2,3] (2,3] (2,3] (2,3] (2,3] (4,5] (4,5] (3,4] (2,3] (3,4] (3,4] (3,4]
 [76] (2,3] (3,4] (2,3] (3,4] (3,4] (3,4] (3,4] (2,3] (3,4] (2,3] (2,3] (4,5] (3,4] (2,3] (2,3]
 [91] (4,5] (4,5] (2,3] (2,3] (3,4] (2,3] (4,5] (4,5] (3,4] (3,4]
Levels: (1,2] (2,3] (3,4] (4,5]
```

❖ **Code:**

calculate the frequency of sample

```
table(z)
```

❖ **Output:**

❖ **Result:**

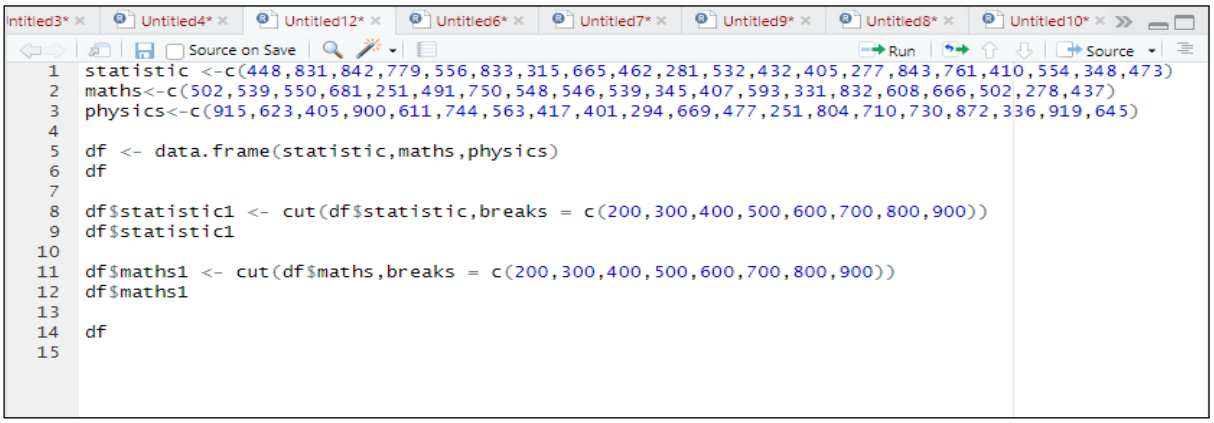
we generate a random sample of 100 observations from a uniform distribution using the runif() function with a minimum value of 2 and a maximum value of 5. Next, we create class intervals using the seq() function with a step size of 0.5 to create intervals ranging from 2 to 5. We then categorize the random sample into these class intervals using the **cut()** function, which returns a factor variable representing the category of each observation. Finally, we use the **table()** function to calculate the frequency of each class interval and print the result using **cat().**

Practical-6

❖ AIM:

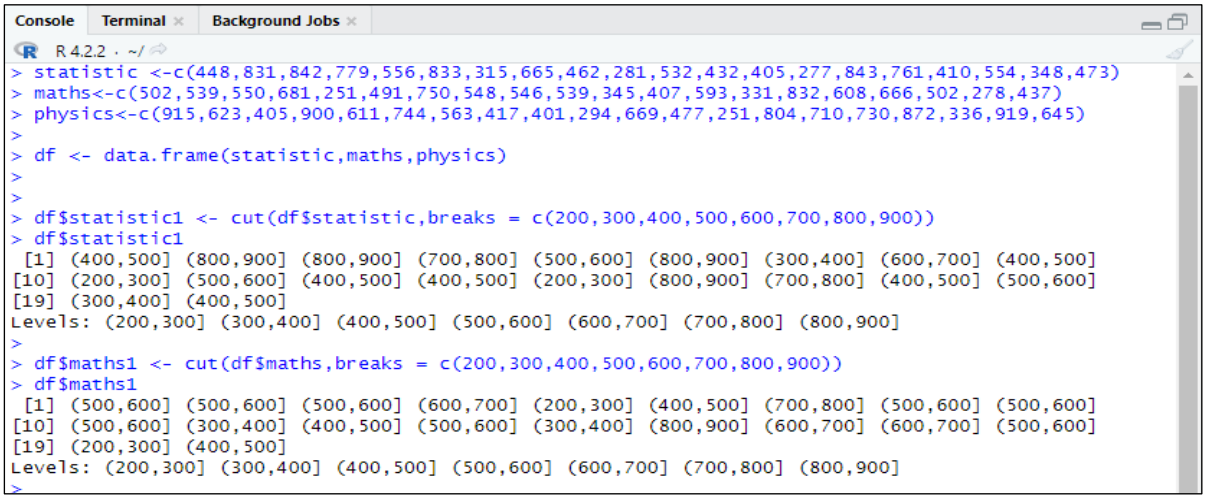
The following data is the number of pages is 20 books on Statistic Mathematics and Physics construct a data frame and categorize the continuous variable statistic & Mathematics in the different class interval

❖ Code:



```
statistic <-c(448,831,842,779,556,833,315,665,462,281,532,432,405,277,843,761,410,554,348,473)
maths<-c(502,539,550,681,251,491,750,548,546,539,345,407,593,331,832,608,666,502,278,437)
physics<-c(915,623,405,900,611,744,563,417,401,294,669,477,251,804,710,730,872,336,919,645)

df <- data.frame(statistic,maths,physics)
df

df$statistic1 <- cut(df$statistic,breaks = c(200,300,400,500,600,700,800,900))
df$statistic1

df$maths1 <- cut(df$maths,breaks = c(200,300,400,500,600,700,800,900))
df$maths1

df
```

❖ Output:



```
> statistic <-c(448,831,842,779,556,833,315,665,462,281,532,432,405,277,843,761,410,554,348,473)
> maths<-c(502,539,550,681,251,491,750,548,546,539,345,407,593,331,832,608,666,502,278,437)
> physics<-c(915,623,405,900,611,744,563,417,401,294,669,477,251,804,710,730,872,336,919,645)
>
> df <- data.frame(statistic,maths,physics)
>
>
> df$statistic1 <- cut(df$statistic,breaks = c(200,300,400,500,600,700,800,900))
> df$statistic1
 [1] (400,500] (800,900] (800,900] (700,800] (500,600] (800,900] (300,400] (600,700] (400,500]
[10] (200,300] (500,600] (400,500] (400,500] (200,300] (800,900] (700,800] (400,500] (500,600]
[19] (300,400] (400,500]
Levels: (200,300] (300,400] (400,500] (500,600] (600,700] (700,800] (800,900]
>
> df$maths1 <- cut(df$maths,breaks = c(200,300,400,500,600,700,800,900))
> df$maths1
 [1] (500,600] (500,600] (500,600] (600,700] (200,300] (400,500] (700,800] (500,600] (500,600]
[10] (500,600] (300,400] (400,500] (500,600] (300,400] (800,900] (600,700] (600,700] (500,600]
[19] (200,300] (400,500]
Levels: (200,300] (300,400] (400,500] (500,600] (600,700] (700,800] (800,900]
>
```

```
> df
   statistic maths physics statistic1    maths1
1        448   502     915  (400,500] (500,600]
2        831   539     623  (800,900] (500,600]
3        842   550     405  (800,900] (500,600]
4        779   681     900  (700,800] (600,700]
5        556   251     611  (500,600] (200,300]
6        833   491     744  (800,900] (400,500]
7        315   750     563  (300,400] (700,800]
8        665   548     417  (600,700] (500,600]
9        462   546     401  (400,500] (500,600]
10       281   539     294  (200,300] (500,600]
11       532   345     669  (500,600] (300,400]
12       432   407     477  (400,500] (400,500]
13       405   593     251  (400,500] (500,600]
14       277   331     804  (200,300] (300,400]
15       843   832     710  (800,900] (800,900]
16       761   608     730  (700,800] (600,700]
17       410   666     872  (400,500] (600,700]
18       554   502     336  (500,600] (500,600]
19       348   278     919  (300,400] (200,300]
20       473   437     645  (400,500] (400,500]
>
```

❖ **Result:**

Construct a data frame with variables Statistic Mathematics and Physics using **data.frame ()** Function and then categorize the continuous variables statistic and mathematics in different class interval using cut () function.

Practical - 7

❖ Aim:

To work with dates and times in R

❖ Theory:

R provides several options for dealing with date and date/time data. Three date/time classes commonly used in R are Date, POSIXct and POSIXlt.

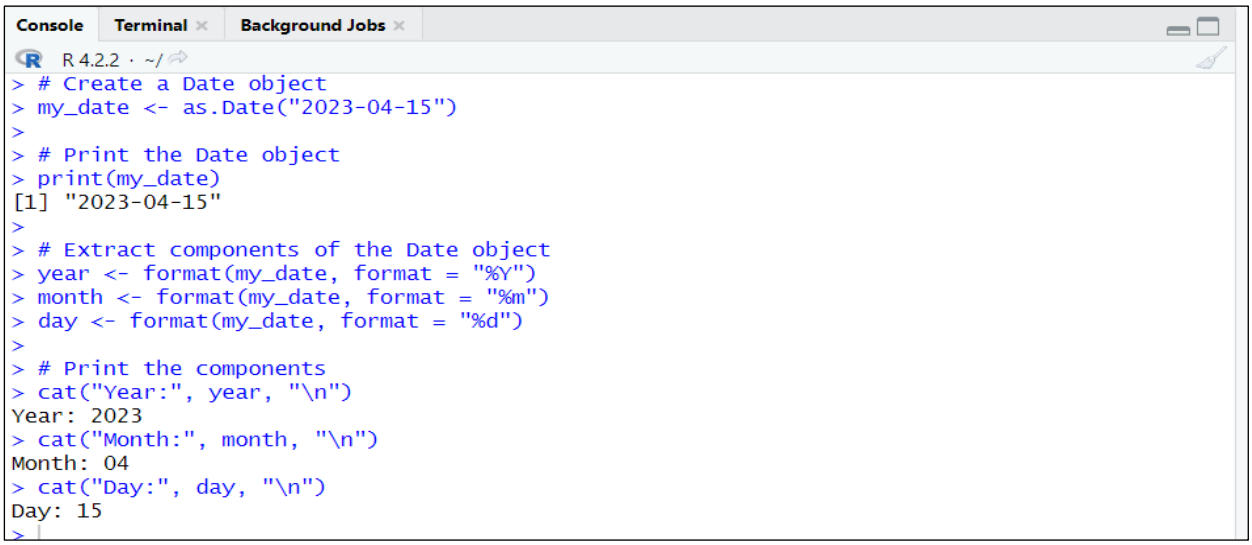
❖ formats and their descriptions used in date and time in r:

In R, you can use format codes to specify the desired format for representing dates and times when using functions like **format(), strftime(), and as.Date()**. Here are some commonly used format codes and their descriptions:

- %Y: Year with century as a decimal number (e.g., 2023).
- %y: Year without century as a decimal number (00-99).
- %m: Month as a decimal number (01-12).
- %d: Day of the month as a decimal number (01-31).
- %H: Hour (00-23) as a decimal number.
- %M: Minute (00-59) as a decimal number.
- %S: Second (00-59) as a decimal number.
- %b: Abbreviated month name.
- %B: Full month name.
- %a: Abbreviated weekday name.
- %A: Full weekday name.
- %p: Either AM or PM.
- %z: Time zone offset from UTC as +HHMM or -HHMM.
- %Z: Time zone name**.**

❖ Date class:

The Date class represents dates without any time information. Dates in R are typically represented in the format "YYYY-MM-DD". Here's an example:



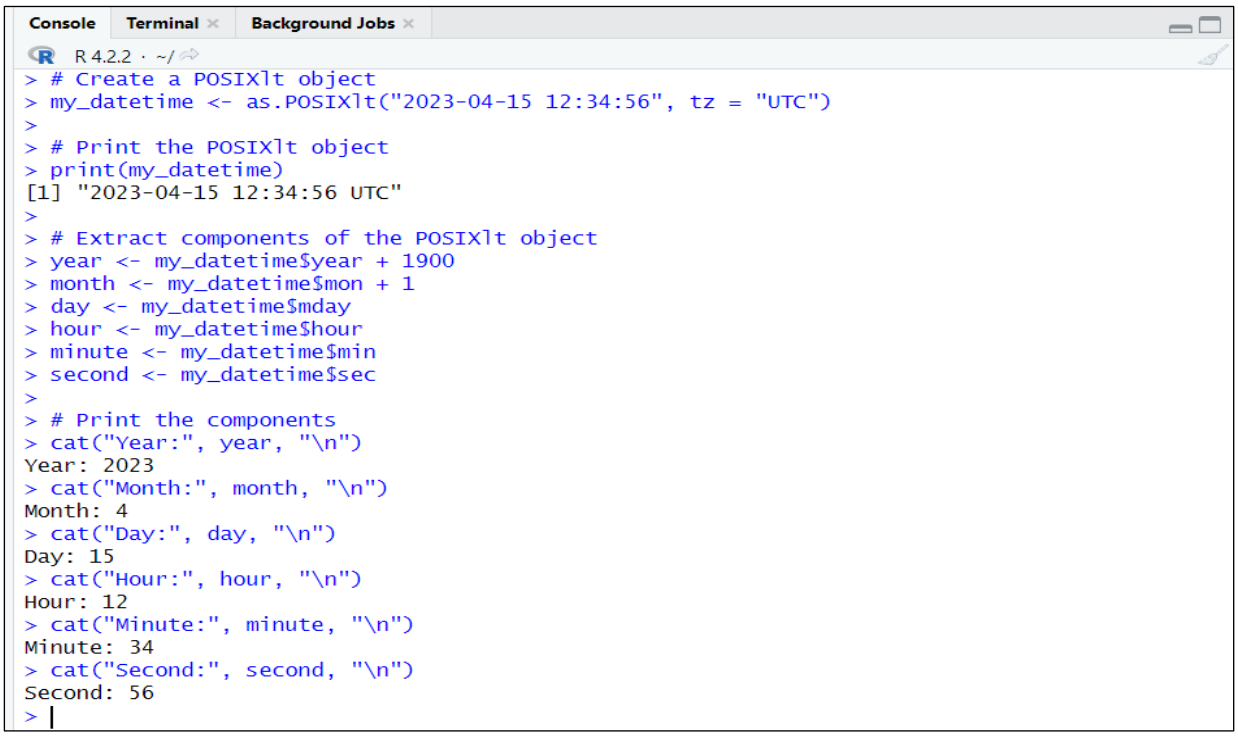
❖ POSIXct class:

The POSIXct class represents date and time information as a continuous numeric value representing the number of seconds since the UNIX epoch (January 1, 1970, 00:00:00 UTC). Here's an example:

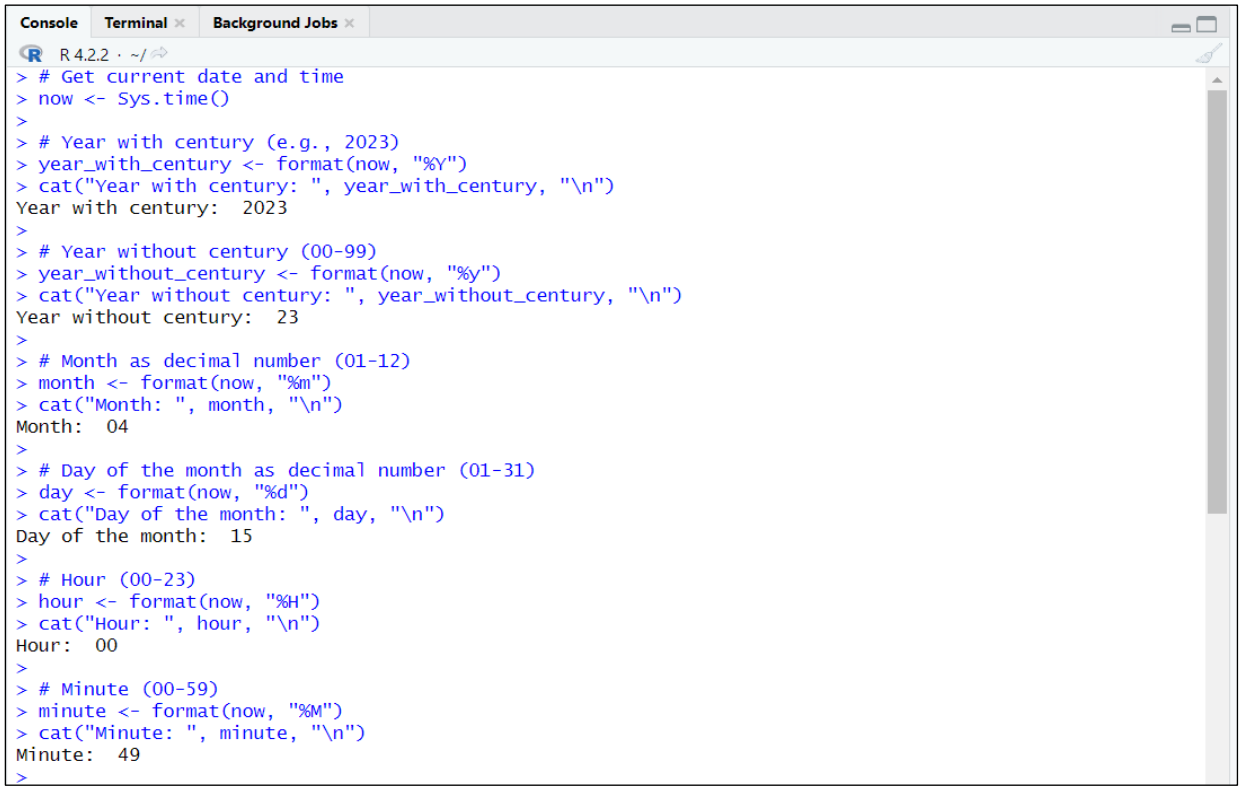
```
> # Create a POSIXct object
>
> my_datetime <- as.POSIXct("2023-04-15 12:34:56", tz = "UTC")
>
> # Print the POSIXct object
> print(my_datetime)
[1] "2023-04-15 12:34:56 UTC"
>
> # Extract components of the POSIXct object
>
> year <- format(my_datetime, format = "%Y")
> month <- format(my_datetime, format = "%m")
> day <- format(my_datetime, format = "%d")
> hour <- format(my_datetime, format = "%H")
> minute <- format(my_datetime, format = "%M")
> second <- format(my_datetime, format = "%S")
>
> # Print the components
>
> cat("Year:", year, "\n")
Year: 2023
> cat("Month:", month, "\n")
Month: 04
> cat("Day:", day, "\n")
Day: 15
> cat("Hour:", hour, "\n")
Hour: 12
> cat("Minute:", minute, "\n")
Minute: 34
> cat("Second:", second, "\n")
Second: 56
```

❖ POSIXlt class:

The POSIXlt class represents date and time information as a list of components (e.g., year, month, day, hour, minute, second) which allows for more fine-grained manipulation. Here's an example:



❖ Here's an example of how you can use these format codes in R to get the current date and time in various formats:



```
Console   Terminal ×   Background Jobs ×
R 4.2.2 · ~/
> # Get current date and time
> now <- Sys.time()
>
> # Year with century (e.g., 2023)
> year_with_century <- format(now, "%Y")
> cat("Year with century: ", year_with_century, "\n")
Year with century:  2023
>
> # Year without century (00-99)
> year_without_century <- format(now, "%y")
> cat("Year without century: ", year_without_century, "\n")
Year without century:  23
>
> # Month as decimal number (01-12)
> month <- format(now, "%m")
> cat("Month: ", month, "\n")
Month:  04
>
> # Day of the month as decimal number (01-31)
> day <- format(now, "%d")
> cat("Day of the month: ", day, "\n")
Day of the month:  15
>
> # Hour (00-23)
> hour <- format(now, "%H")
> cat("Hour: ", hour, "\n")
Hour:  00
>
> # Minute (00-59)
> minute <- format(now, "%M")
> cat("Minute: ", minute, "\n")
Minute:  49
>
```

```
> # Second (00-59)
> second <- format(now, "%S")
> cat("Second: ", second, "\n")
Second:  53
>
> # Abbreviated month name
> abbreviated_month <- format(now, "%b")
> cat("Abbreviated month name: ", abbreviated_month, "\n")
Abbreviated month name:  Apr
>
> # Full month name
> full_month <- format(now, "%B")
> cat("Full month name: ", full_month, "\n")
Full month name:  April
>
> # Abbreviated weekday name
> abbreviated_weekday <- format(now, "%a")
> cat("Abbreviated weekday name: ", abbreviated_weekday, "\n")
Abbreviated weekday name:  Sat
>
> # Full weekday name
> full_weekday <- format(now, "%A")
> cat("Full weekday name: ", full_weekday, "\n")
Full weekday name:  Saturday
>
```

❖ Result:

The actual result of dates and times in R will depend on the format and class of the object being used, as well as any formatting or conversion operations applied to the object.

Practical-8

❖ Aim:

To select subset of data in R.

❖ Theory:

In R, a subset of data refers to a portion of a larger data set that is selected based on specific conditions or criteria. Subsetting data is a common operation in data analysis and allows you to extract and work with a smaller, more focused portion of the data for further analysis or visualization.

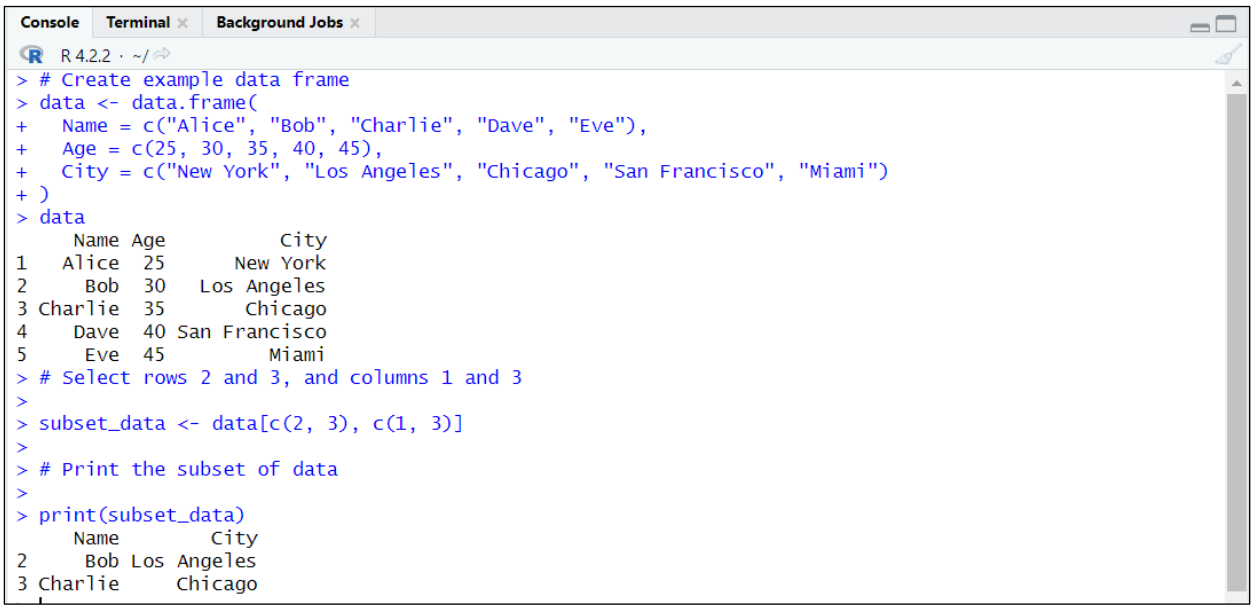
There are several ways to subset data in R, including using indexing, logical conditions, or functions from various packages. Here are some common methods for creating subsets of data in R:

1. **Indexing:**
   You can use square brackets [] to subset data based on row and column indices. For example, data[row_indices, col_indices] can be used to select specific rows and columns from a data frame or a matrix.

   Code:

```
# Create example data frame
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "Dave", "Eve"),
  Age = c(25, 30, 35, 40, 45),
  City = c("New York", "Los Angeles", "Chicago", "San Francisco", "Miami")
)
```

   Output:



```
Console  Terminal  Background Jobs
R 4.2.2 · ~/
> # Create example data frame
> data <- data.frame(
+   Name = c("Alice", "Bob", "Charlie", "Dave", "Eve"),
+   Age = c(25, 30, 35, 40, 45),
+   City = c("New York", "Los Angeles", "Chicago", "San Francisco", "Miami")
+ )
> data
     Name Age          City
1   Alice  25      New York
2     Bob  30   Los Angeles
3 Charlie  35       Chicago
4    Dave  40 San Francisco
5     Eve  45         Miami
> # Select rows 2 and 3, and columns 1 and 3
>
> subset_data <- data[c(2, 3), c(1, 3)]
>
> # Print the subset of data
>
> print(subset_data)
     Name        City
2     Bob Los Angeles
3 Charlie     Chicago
```

2. **Logical conditions:**
   You can use logical conditions to subset data based on values in one or more columns. For example, data[data$Age > 30,] selects rows where the value in the Age column is greater than 30.

   Output:

```
Console  Terminal  Background Jobs
R 4.2.2 · ~/
> data <- data.frame(
+   Name = c("Alice", "Bob", "Charlie", "Dave", "Eve"),
+   Age = c(25, 30, 35, 40, 45),
+   City = c("New York", "Los Angeles", "Chicago", "San Francisco", "Miami")
+ )
>
> # Select rows where Age is greater than 30
> subset_data <- data[data$Age > 30, ]
>
> # Print the subset of data
> print(subset_data)
     Name Age          City
3 Charlie  35       Chicago
4    Dave  40 San Francisco
5     Eve  45         Miami
>
```

3. Functions:

R provides various functions for subsetting data, such as subset(), filter() from the dplyr package, and slice() from the dplyr or data.table package. These functions allow you to specify conditions or criteria to filter and extract specific rows or columns from a data frame.

OUTPUT:

```
Console  Terminal  Background Jobs
R 4.2.2 · ~/
> data <- data.frame(
+   Name = c("Alice", "Bob", "Charlie", "Dave", "Eve"),
+   Age = c(25, 30, 35, 40, 45),
+   City = c("New York", "Los Angeles", "Chicago", "San Francisco", "Miami")
+ )
> # Load the dplyr package
> library(dplyr)
>
> # Select rows where Age is greater than 30 using filter()
> subset_data <- filter(data, Age > 30)
>
> # Print the subset of data
> print(subset_data)
     Name Age          City
1 Charlie  35       Chicago
2    Dave  40 San Francisco
3     Eve  45         Miami
>
```

4. Regular expressions:

You can use regular expressions with functions like grep() or grepl() to subset data based on patterns or text matching in character vectors.

OUTPUT:

```
Console  Terminal  Background Jobs
R 4.2.2 · ~/
> data <- data.frame(
+   Name = c("Alice", "Bob", "Charlie", "Dave", "Eve"),
+   Age = c(25, 30, 35, 40, 45),
+   City = c("New York", "Los Angeles", "Chicago", "San Francisco", "Miami")
+ )
> # Create example character vector
> text <- c("apple", "banana", "cherry", "date", "elderberry")
>
> # Select elements that contain "a"
> subset_text <- text[grep("a", text)]
>
> # Print the subset of text
> print(subset_text)
[1] "apple"  "banana" "date"
>
```

Practical-9

❖ Aim:

To select a random sample from a dataset in R.

❖ Theory:

A random sample from a dataset in R refers to a subset of data that is randomly selected without any specific pattern or order. This random selection is useful for various purposes, such as statistical inference, data exploration, and model validation.

In R, you can use the sample() function to generate a random sample from a dataset. The sample() function randomly selects elements from a vector or a data frame. Here's an example:

Code:

```
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "Dave", "Eve"),
  Age = c(25, 30, 35, 40, 45),
  City = c("New York", "Los Angeles", "Chicago", "San Francisco", "Miami")
)
```

Output:

❖ Result:

we use the resulting row indices to select the corresponding rows from the data frame and store them in random_sample, which is then printed using cat() and print() functions.

Practical-10

- **Aim:**

- **Theory:**