Batch: E2 Roll No.: 16010123325

Experiment No. 1

Title: Exploring R for Data Science

Course Outcome:

CO1, CO3

Books/ Journals/ Websites referred:

1. The Comprehensive R Archive Network
2. Posit

Resources used: Only this document, and the teaching in the classroom

---

In this lab, we will introduce some simple R commands. R is a programming language for statistical computing and data visualization. It has been adopted in the fields of data mining, bioinformatics and data analysis

The best way to learn a new language is to try out the commands. R can be downloaded

from http://cran.r-project.org/ . We recommend that you run R within an integrated development environment (IDE) such as RStudio, which can be freely downloaded from http://rstudio.com

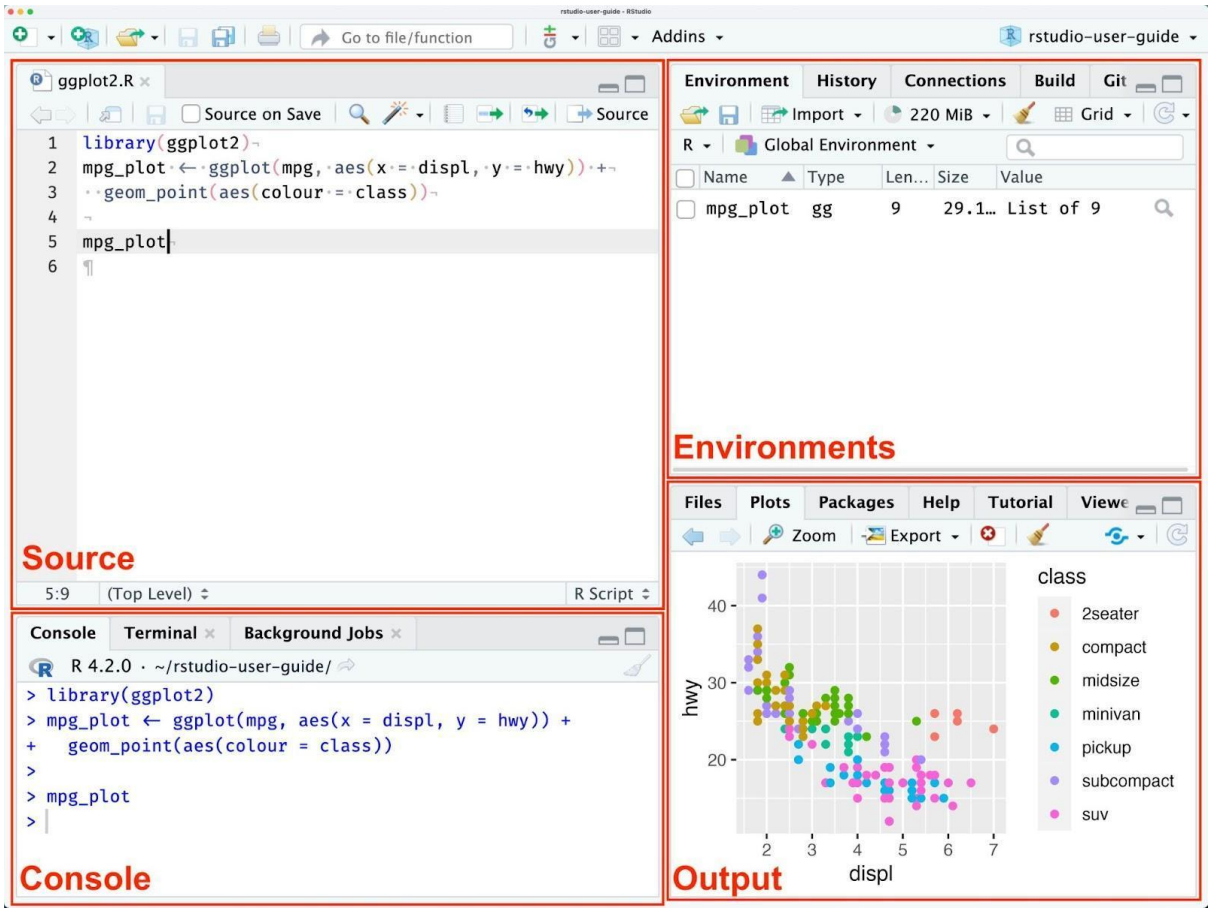
Pane Layout

The RStudio user interface has 4 primary panes:

- **Source pane**
- **Console pane**
- **Environment pane, containing the Environment, History, Connections, Build, VCS , and Tutorial tabs**

- **Output pane, containing the Files, Plots, Packages, Help, Viewer, and Presentation tabs**

Each pane can be minimized or maximized within the column by clicking the minimize/maximize buttons.



> print("Hello, World!")

[1] "Hello, World!"

Basic Arithmetic Operators

- **Arithmetic operations: +, -, *, /, ^, %%, %/%**
- **Assigning values: <- and =**

```
> x <- 10
> y <- 5
> quotient <- x %/% y
> remainder <- x %% y
> quotient
[1] 2
> remainder
[1] 0
```

Basic Data Types

```
> # numeric
> x <- 10.5
> class(x)
[1] "numeric"
>
> # integer
> x <- 1000L
> class(x)
[1] "integer"
>
> # complex
> x <- 9i + 3
> class(x)
[1] "complex"
>
> # character/string
> x <- "R is exciting"
> class(x)
[1] "character"
>
> # logical/boolean
> x <- TRUE
> class(x)
[1] "logical"
```

```
> num <- as.numeric("123")
> char <- as.character(123)
> num
[1] 123
> typeof(num)
[1] "double"
> char
[1] "123"
> typeof(char)
[1] "character"
```

Vectors and Basic Vector Operations

R uses functions to perform operations.

To run a function called *funcname*, we type *funcname(input1, input2)*, where the inputs (or arguments) *input1* and *input2* tell R how to run the function.

A function can have any number of inputs. For example, to create a vector of numbers, we use the function c() (for concatenate).

Any numbers inside the parentheses are joined together.

The following command instructs R to join together the numbers 1, 3, 2, and 5, and to save them as a vector named x.

```
> x <- c(1, 3, 2, 5)
> x
[1] 1 3 2 5
```

When we type x, it gives us back the vector.

Other ways to create vectors are using seq() and rep()

```
> seq_vec <- seq(1, 10, by = 2)
> seq_vec
[1] 1 3 5 7 9
> rep_vec <- rep(5, times = 3)
> rep_vec
[1] 5 5 5
```

We can tell R to add two sets of numbers together. It will then add the first number from x to the first number from y, and so on. However, x and y should be the same length. We can check their length using the length() function.

```
> y = c(1, 4, 3)
```

```
> length(x)
[1] 3
> length(y)
[1] 3
> x + y
[1]  2 10  5
```

The ls() function allows us to look at a list of all of the objects, such as data and functions, that we have saved so far. The rm() function can be used to delete any that we don't want.

```
> ls()
[1] "x" "y"
> rm(x, y)
```

```
> ls()
character(0)
```

It's also possible to remove all objects at once:

```
> rm(list = ls())
```

The matrix() function can be used to create a matrix of numbers. Before we use the matrix() function, we can learn more about it:

```
> ?matrix
```

The help file reveals that the matrix() function takes a number of inputs, but for now we focus on the first three: the data (the entries in the matrix), the number of rows, and the number of columns. First, we create a simple matrix.

```
> x <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Note that we could just as well omit typing data=, nrow=, and ncol= in the matrix() command above: that is, we could just type

```
> x <- matrix(c(1, 2, 3, 4), 2, 2)
```

and this would have the same effect. However, it can sometimes be useful to specify the names of the arguments passed in, since otherwise R will assume that the function arguments are passed into the function in the same order that is given in the function's help file. As this example illustrates, by default R creates matrices by successively filling in columns. Alternatively, the byrow = TRUE option can be used to populate the matrix in order of the rows.

```
> matrix(c(1, 2, 3, 4), 2, 2, byrow = TRUE)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Notice that in the above command we did not assign the matrix to a value such as x. In this case the matrix is printed to the screen but is not saved for future calculations.

```
> mat <- matrix(1:9, nrow = 3, byrow = TRUE)
>
> mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> mat[1,2]
[1] 2
```

The sqrt() function returns the square root of each element of a vector or matrix. The command x^2 raises each element of x to the power 2; any powers are possible, including fractional or negative powers.

```
> sqrt(x)
     [,1] [,2]
[1,] 1.00 1.73
[2,] 1.41 2.00
> x^2
     [,1] [,2]
[1,]    1    9
[2,]    4   16
```

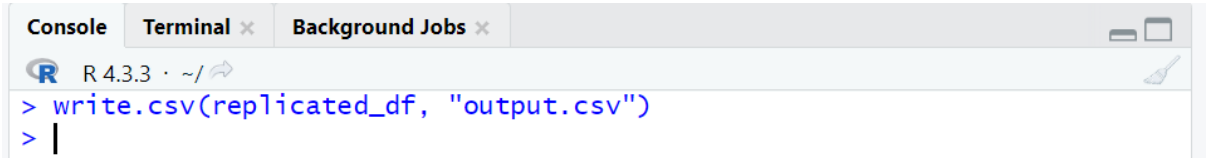
Creating a list

```
> my_list <- list(name = "John", age = 25)
> my_list
$name
[1] "John"

$age
[1] 25

> my_list$name
[1] "John"
```

// do from here: Creating dataframes

```
> # Create a sample data frame
> df <- data.frame(
+     Name = c("Alice", "Bob", "Charlie", "David"),
+     Age = c(25, 30, 35, 40),
+     City = c("New York", "London", "Paris", "Tokyo")
+ )
> df
     Name Age     City
1   Alice  25 New York
2     Bob  30   London
3 Charlie  35    Paris
4   David  40    Tokyo
> # Replicate each row twice
> replicated_df <- cbind(df, rep(row.names(df), each = 2))
> replicated_df
     Name Age     City rep(row.names(df), each = 2)
1   Alice  25 New York                            1
2     Bob  30   London                            1
3 Charlie  35    Paris                            2
4   David  40    Tokyo                            2
5   Alice  25 New York                            3
6     Bob  30   London                            3
7 Charlie  35    Paris                            4
8   David  40    Tokyo                            4
```

Writing dataframe to csv file



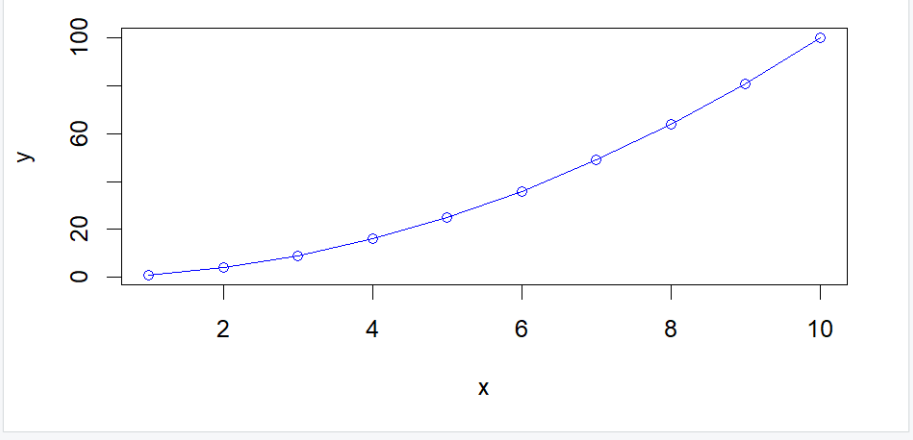
Reading dataframe from a csv file

```
> write.csv(replicated_df, "output.csv")
> new_df <- read.csv("output.csv")
> new_df
  X    Name Age     City rep.row.names.df...each...2.
1 1   Alice  25 New York                             1
2 2     Bob  30   London                             1
3 3 Charlie  35    Paris                             2
4 4   David  40    Tokyo                             2
5 5   Alice  25 New York                             3
6 6     Bob  30   London                             3
7 7 Charlie  35    Paris                             4
8 8   David  40    Tokyo                             4
```

Basic Visualization

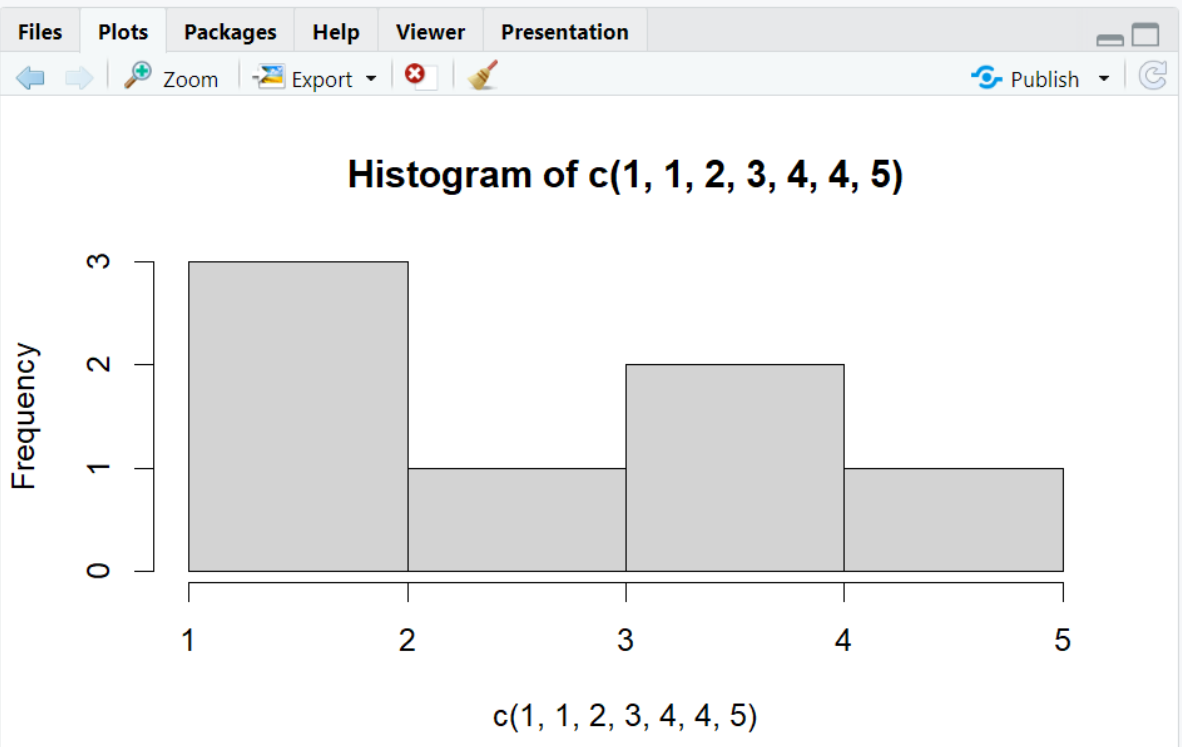
1. **Line plot**

```
> x <- 1:10
> y <- x^2
> plot(x, y, type = "o", col = "blue")
>
```



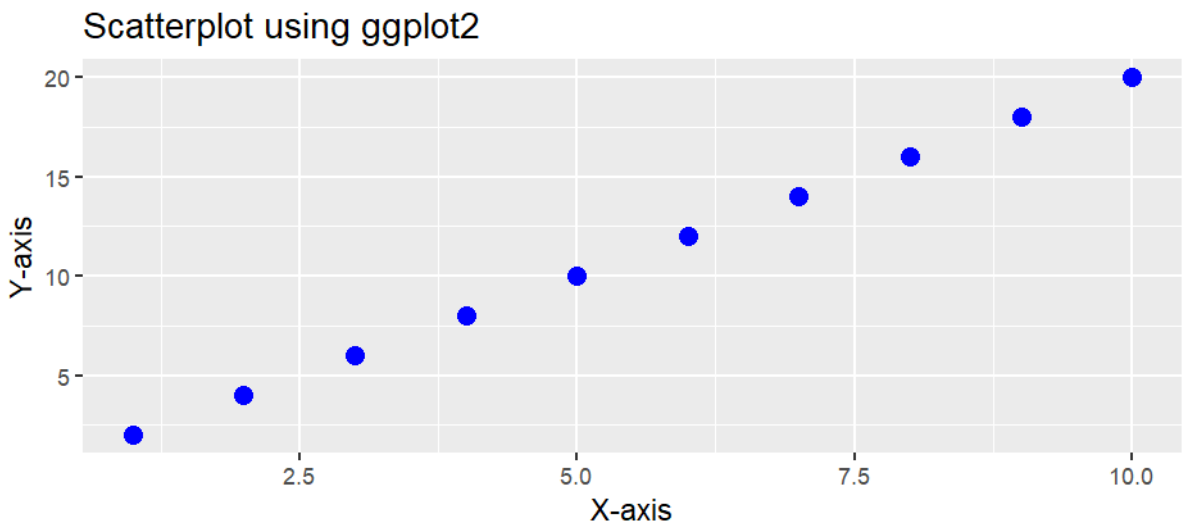
2. **Histogram**

```
> hist(x, col = "green", main = "Histogram")
> hist(c(1,1,2,3,4,4,5))
>
```



3. **Scatterplot, using ggplot2**

```
Console   Terminal ×   Background Jobs ×
R 4.3.3 · ~/
> library(ggplot2)
Learn more about the underlying theory at
https://ggplot2-book.org/
> df <- data.frame(
+     x = 1:10,
+     y = c(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
+ )
> ggplot(df, aes(x = x, y = y)) +
+     geom_point(color = "blue", size = 3) +
+     ggtitle("Scatterplot using ggplot2") +
+     xlab("X-axis") +
+     ylab("Y-axis")
```



Students have to perform the above tasks and add their code and screenshots of the output here:

1) Vectors

```
Console   Terminal ×   Background Jobs ×
R 4.2.2 · ~/
> x <- c(1,2,3,4)
> y <- C(1,2,3,4) #to try if C also works for concantenation
Error in C(1, 2, 3, 4) : object not interpretable as a factor
> x
[1] 1 2 3 4
> seq_vec <- seq(1,10, by =2)
> seq_vec
[1] 1 3 5 7 9
> rep_vec <- rep(1, times =4)
> rep_vec
[1] 1 1 1 1
> y <- c(2,5,6,7)
> length(x)
[1] 4
> length(y)
[1] 4
> x+y
[1]  3  7  9 11
> ls()
[1] "a"       "r"       "rep_vec" "seq_vec" "x"       "y"       "z"
> rm(a)
> rm(z)
> r = readline();
sdkkwmekwemkwmekw
> r = as.integer(readline())
1234
> ls()
[1] "r"       "rep_vec" "seq_vec" "x"       "y"
> rm(list = ls())
>
```

Output:

values

rep_vec	num [1:4] 1 1 1 1
seq_vec	num [1:5] 1 3 5 7 9
x	num [1:4] 1 2 3 4
y	num [1:4] 2 5 6 7

```
> x+y
[1]  3  7  9 11
> ls()
[1] "rep_vec" "seq_vec" "x"       "y"
>
```

2) Matrix

```
Console | Terminal | Background Jobs
R 4.2.2 · ~/
> rm(list = ls())
> x <- matrix(data = c(1,2,3,4), nrow = 2, ncol=2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> x <- matrix(data = c(1,2,3,4), nrow = 2, ncol=2, byrow = TRUE)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> sqrt(x)
         [,1]     [,2]
[1,] 1.000000 1.414214
[2,] 1.732051 2.000000
> x^2
     [,1] [,2]
[1,]    1    4
[2,]    9   16
> x = x^2
> x
     [,1] [,2]
[1,]    1    4
[2,]    9   16
>
```

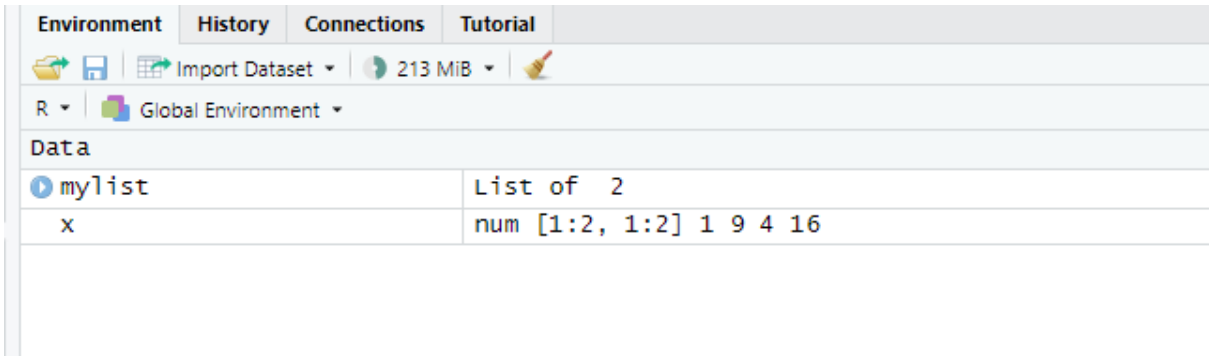
Output:

```
Environment | History | Connections | Tutorial
Import Dataset | 213 MiB
R | Global Environment
Data
  x                    num [1:2, 1:2] 1 9 4 16
```

3) Lists

```
Console   Terminal ×   Background Jobs ×
R 4.2.2 · ~/
> mylist <- list(name = "Shreya", age = 25)
> name
Error: object 'name' not found
> mylist
$name
[1] "Shreya"

$age
[1] 25
```

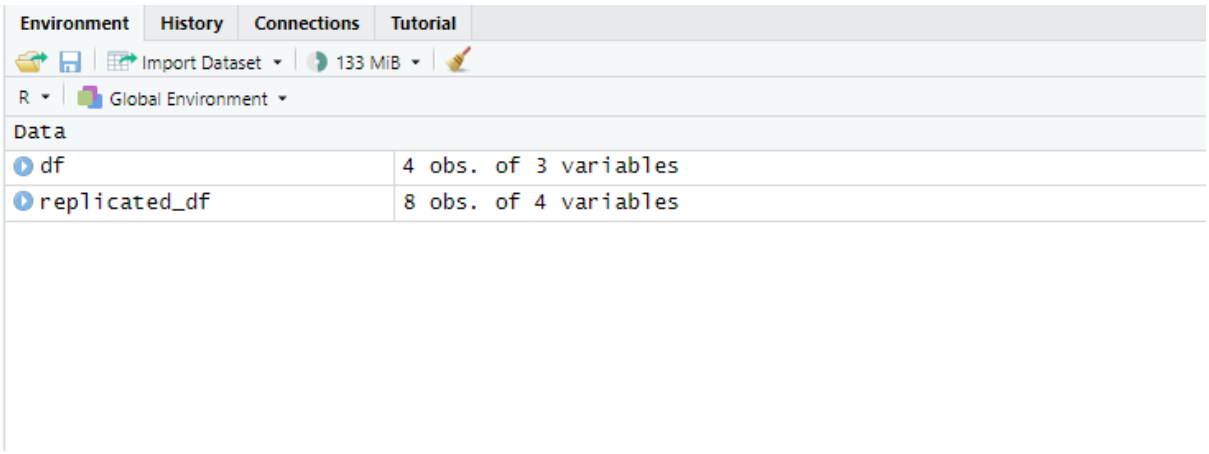
Output:



4) Creating Dataframe

```
> df<- data.frame(
+     Name = c("Alice", "Bob", "Charlie", "Meow")
+     Age = c(25,30, 35, 40)
Error: unexpected symbol in:
"    Name = c("Alice", "Bob", "Charlie", "Meow")
    Age"
> df<- data.frame(
+     Name = c("Alice", "Bob", "Charlie", "Meow"),
+     Age = c(25,30, 35, 40),
+     City = c("NYC", "London" , "Bay Area", "Paris")
+ )
> print(df)
     Name Age     City
1   Alice  25      NYC
2     Bob  30   London
3 Charlie  35 Bay Area
4    Meow  40    Paris
> replicated_df <- cbind(df, rep(row.names(df), each=2))
> print(replicated_df)
     Name Age     City rep(row.names(df), each = 2)
1   Alice  25      NYC                             1
2     Bob  30   London                             1
3 Charlie  35 Bay Area                             2
4    Meow  40    Paris                             2
5   Alice  25      NYC                             3
6     Bob  30   London                             3
7 Charlie  35 Bay Area                             4
8    Meow  40    Paris                             4
>
```

Output:



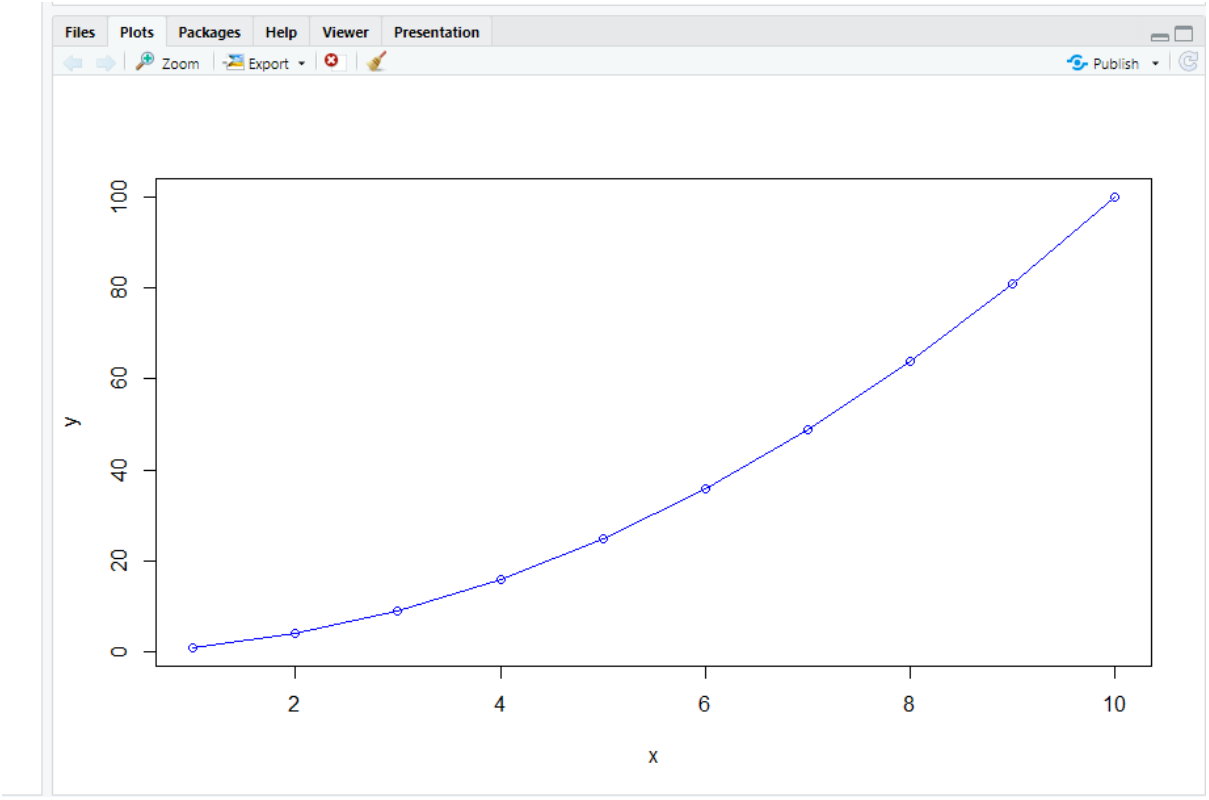
5) Converting Dataframe into csv

```
> write.csv(replicated_df, "output.csv")
> new_df <- read.csv("output.csv")
> print(new_df)
  X    Name Age     City rep.row.names.df...each...2.
1 1   Alice  25      NYC                             1
2 2     Bob  30   London                             1
3 3 Charlie  35 Bay Area                             2
4 4    Meow  40    Paris                             2
5 5   Alice  25      NYC                             3
6 6     Bob  30   London                             3
7 7 Charlie  35 Bay Area                             4
8 8    Meow  40    Paris                             4
>
```

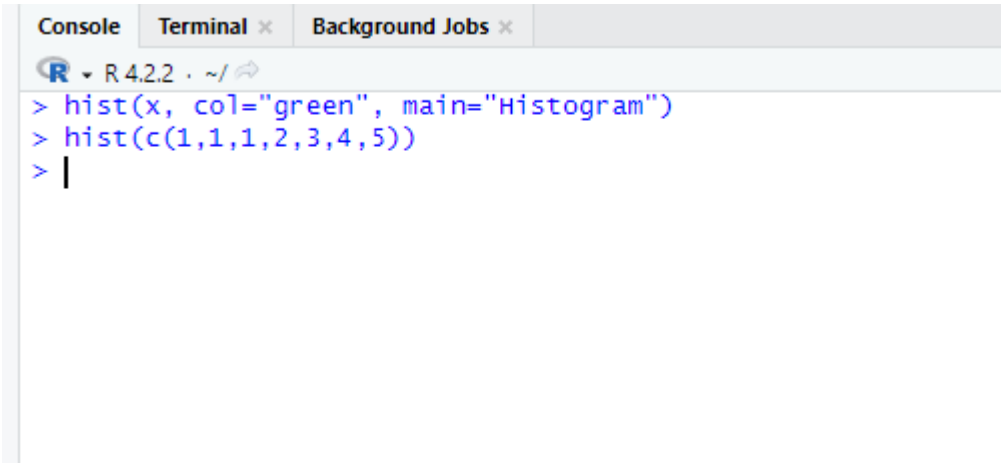
6) Line plot

```
Console   Terminal   Background Jobs
R 4.2.2 · ~/
> x<- 1:10
> y<- x^2
> plot(x,y, type="o", col="blue")
>
```

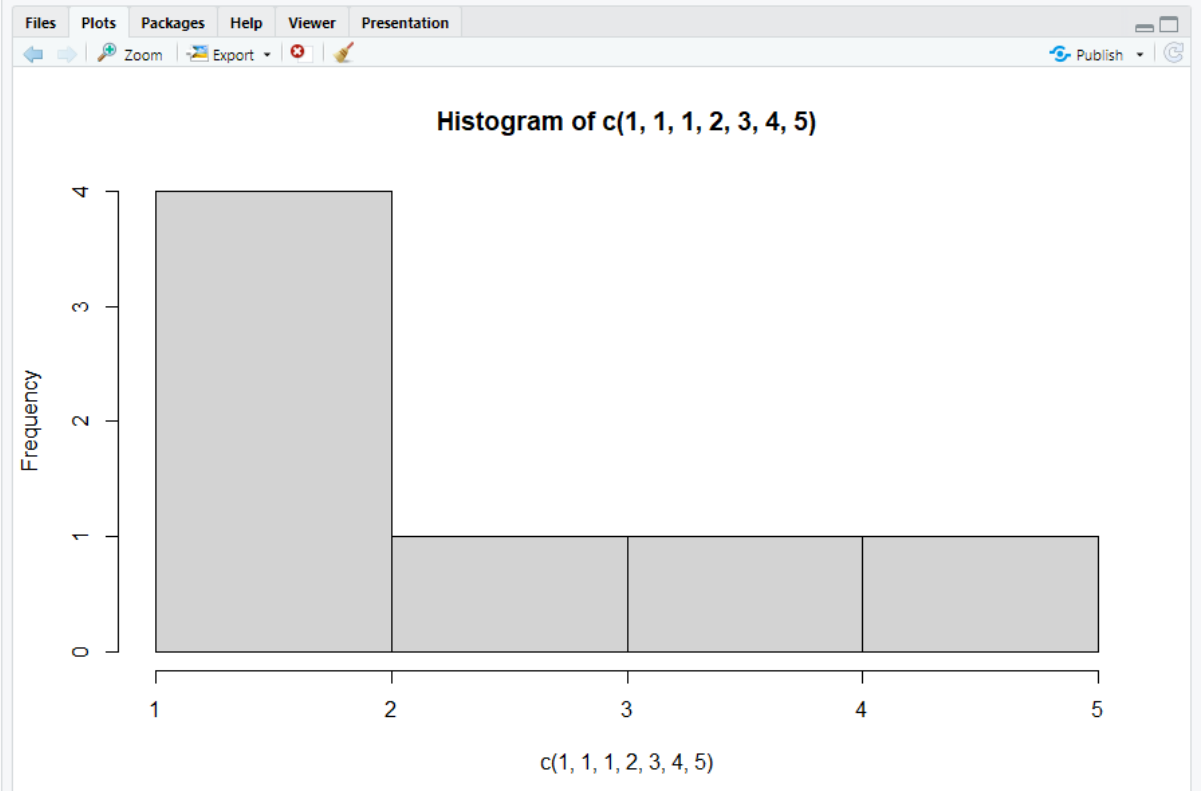
Output:



7) Histogram



Output:

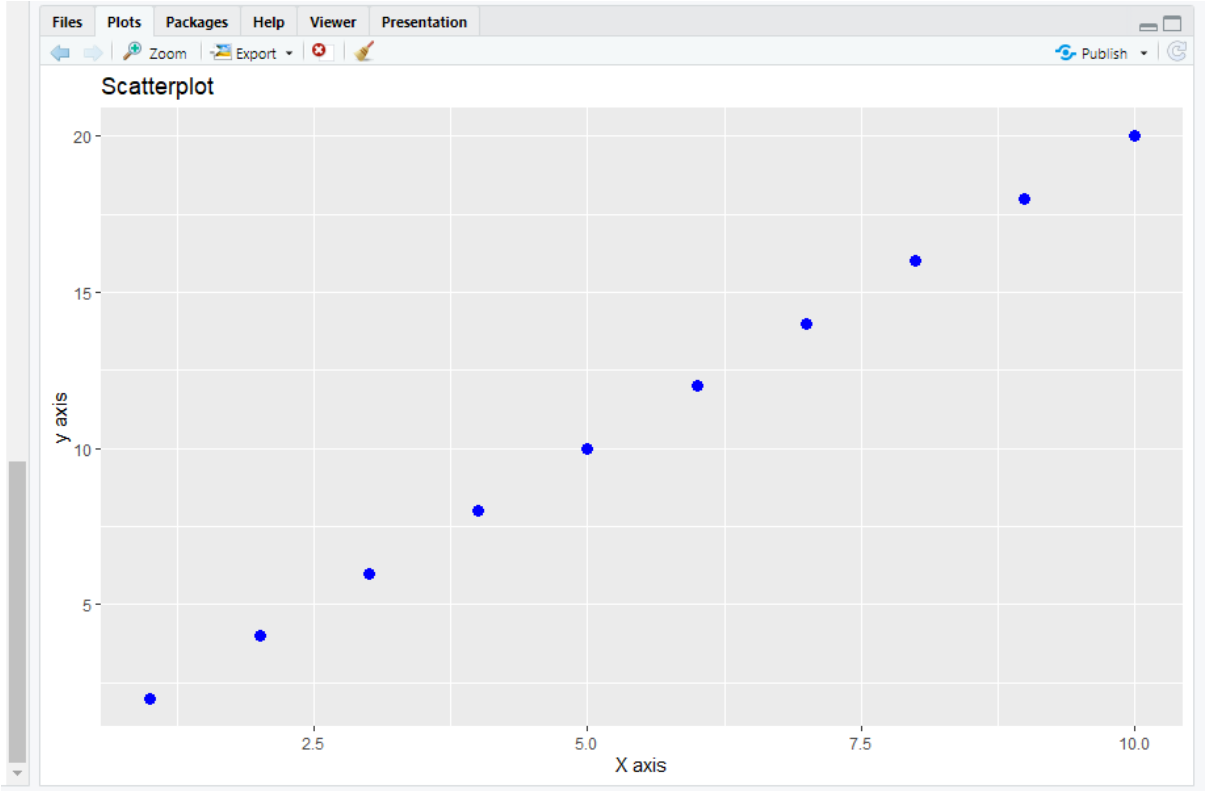


8) Scatter Plot

```
> install.packages("ggplot2")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate
version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/KJSCE/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
also installing the dependencies 'colorspace', 'utf8', 'farver', 'labeling', 'munsell', 'R6', 'RColorBrewer', 'viridisLit
e', 'fansi', 'magrittr', 'pillar', 'pkgconfig', 'cli', 'glue', 'gtable', 'isoband', 'lifecycle', 'rlang', 'scales', 'tibbl
e', 'vctrs', 'withr'


  There are binary versions available but the source versions are later:
            binary source needs_compilation
colorspace  2.1-0  2.1-1              TRUE
farver      2.1.1  2.1.2              TRUE
pillar      1.9.0 1.10.1             FALSE
cli         3.6.2  3.6.3              TRUE
glue        1.7.0  1.8.0              TRUE
gtable      0.3.5  0.3.6             FALSE
rlang       1.1.3  1.1.4              TRUE
withr       3.0.0  3.0.2             FALSE
```

```
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (withr)

The downloaded source packages are in
        'C:\Users\KJSCE\AppData\Local\Temp\Rtmpa28z6D\downloaded_packages'
> library(ggplot2)
Warning message:
package 'ggplot2' was built under R version 4.2.3
> ggplot(df, aes(x=x, y=y))+
+     geom_point(color="blue", size=3)+
+     ggtitle("Scatterplot")+
+     xlab("X axis")+
+     ylab("y axis")
>
```

Output:



Conclusion: In this experiment, we learnt some basic R commands for statistical computing and data visualization.

Post Lab Questions:

1. **Explain the difference between vectors, lists, and data frames in R. Provide an example of when each would be used.**

Ans:

In R, vectors, lists, and data frames serve different purposes in managing data. A vector is a one-dimensional array that holds elements of the same type, such as numbers, characters, or logical values. It is ideal for simple collections of homogeneous data, like `numbers <- c(1, 2, 3, 4, 5)`. Lists, however, are more flexible as they can contain elements of different types, such as numbers, strings, or even other lists. This makes lists useful when you need to group different types of data together, like `person <- list(name = "Alice", age = 25, scores = c(85, 90, 95))`. Lastly, data frames are two-dimensional structures similar to tables, where each column is a vector, but columns can hold different data types. Data frames are particularly useful when working with structured datasets, such as in statistical analysis or data manipulation, like `df <- data.frame(Name = c("Alice", "Bob"), Age = c(25, 30), Score = c(90, 85))`. Each of these structures is suited to specific scenarios: vectors for simple, homogeneous data; lists for mixed data types; and data frames for structured, tabular data.

2. **What are the differences between the plot() and ggplot() functions in R? When would you use each?**

Ans: The `plot()` and `ggplot()` functions in R both serve the purpose of creating visualizations, but they are fundamentally different in their capabilities and complexity. The `plot()` function is part of base R and is used for creating basic plots with minimal setup. It is quick and easy to use for straightforward visualizations, but its customization options are somewhat limited. For instance, you can use `plot(x, y)` to quickly create a scatter plot, but finer control over aesthetics and design requires additional steps. On the other hand, `ggplot()` is part of the `ggplot2` package and is based on the "Grammar of Graphics," which allows for highly customizable and complex plots. It enables users to build visualizations in layers, combining different elements like data points, lines, and statistical summaries. An example of using `ggplot()` is `ggplot(data = mtcars, aes(x = mpg, y = hp)) + geom_point() + ggtitle("ggplot2 Scatter Plot")`, which provides far more flexibility and control over the plot's appearance and functionality. While `plot()` is often used for quick and simple visualizations, `ggplot()` is preferred for more detailed and professional-level graphics, especially when dealing with large or complex datasets.

3. **What parameters can you customize in the hist() function to change the appearance of the histogram?**

Ans: The `hist()` function in R is used to create histograms, and it offers several parameters to customize the appearance of the plot. One of the most important parameters is `breaks`, which controls the number or width of the bins in the histogram, allowing you to adjust the granularity of the data representation. For instance, using `breaks = 20` specifies that the histogram should have 20 bins. You can also change the color of the bars using the `col` parameter, like `col = "blue"`, to make the histogram visually appealing. The `border` parameter allows you to modify the color of the bar borders, which can be helpful for visual distinction, such as `border = "red"`. Additionally, the `main` parameter lets you add a title to the histogram, like `main = "Histogram of Data"`, while the `xlab` and `ylab` parameters allow customization of the axis labels. For example, `xlab = "Values"` and `ylab = "Frequency"`. If you need to adjust the axis limits, the `xlim` and `ylim` parameters come in handy. Overall, these customizable options in the `hist()` function provide flexibility to control the appearance and presentation of the histogram according to the specific needs of the analysis.

4. **Write an R command to create a vector of numbers from 1 to 100, but only include multiples of 5.**

Ans:

In R, you can create a vector of numbers from 1 to 100 that includes only the multiples of 5 using the `seq()` function. The `seq()` function generates sequences of numbers, and by specifying a starting point, an endpoint, and an increment, you can select multiples of 5. The command would be: `multiples_of_5 <- seq(5, 100, by = 5)`. This command generates a sequence starting from 5, ending at 100, and incrementing by 5. The resulting vector will include all the multiples of 5 between 1 and 100, such as `[1] 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100`. This approach is concise and efficient for selecting multiples of any number within a specified range.