Experiment No - 1

Aim: Getting introduced to data analytics libraries in Python and R.

Theory:

R

R is a programming language for statistical computing and graphics. Created by statisticians Ross Ihaka and Robert Gentleman, R is used for data mining, bioinformatics, and data analysis.
Some common data I/O functions are:

- To read a rectangular dataset with readr, you combine two pieces: a function that parses the lines of the file into individual fields and a column specification. readr supports the following file formats with these read_*() functions:
  read_csv(): comma-separated values (CSV)
  read_tsv(): tab-separated values (TSV)
  read_csv2(): semicolon-separated values with , as the decimal mark
  read_delim(): delimited files (CSV and TSV are important special cases)
  read_table(): whitespace-separated files
- write() is used to write Data to a File. Usage:
  ```
  write(x, file = "data",
      ncolumns = if(is.character(x)) 1 else 5,
      append = FALSE, sep = " ")
  ```
- The generic function hist computes a histogram of the given data values. If plot = TRUE, the resulting object of class "histogram" is plotted by plot.histogram, before it is returned.
  Usage: hist(x, ...)

The different one dimensional/vector data types/classes in R are:

- numeric: any real number(s)
- character: strings or individual characters, quoted
- integer: any integer(s)/whole numbers
- factor: categorical/qualitative variables
- logical: variables composed of TRUE or FALSE
- Date/POSIXct: represents calendar dates and times

The complex R classes are:

- lists - The most generic data class is the list, which can be created using the list() function. A list can hold vectors, strings, matrices, models, lists of other lists, or any other object you can create in R. You reference elements of a list by using $, [], or [[]].
- data.frame/tibble - This is an R dataset where the number of rows corresponds to the total number of observations and each column corresponds to a variable.
- matrix - R also has another 2 dimensional class called a matrix. A matrix is a two dimensional array, composed of rows and columns (just like the data.frame), but unlike the data frame the entire matrix is composed of one R class, e.g. all numeric, all characters, all logical, etc.

Python

Python has various libraries for data analytics:

NumPy:

NumPy is a powerful Python library for numerical computing. It provides support for multidimensional arrays, along with a collection of functions to operate on these arrays efficiently. NumPy is widely used in scientific computing, data analysis, machine learning, and more. Some basic concepts of Numpy are:

- Arrays: The core of NumPy is the ndarray (n-dimensional array) object, which represents a multidimensional array of elements of the same type. These arrays can be of any dimensionality, and they support efficient mathematical operations.
- Data Types: NumPy provides a variety of data types, such as int, float, bool, etc., with different bit sizes.
- Vectorization: NumPy's universal functions (ufuncs) allow for element-wise operations on arrays, which can often be faster and more concise than using traditional loops.
- Broadcasting: Broadcasting is a powerful mechanism that allows NumPy to work with arrays of different shapes during arithmetic operations.
- Array Manipulation: NumPy provides functions to manipulate arrays, including reshaping, slicing, concatenation, splitting, and more.

Pandas:

Pandas is a Python library built on top of NumPy, providing high-level data structures and tools for data manipulation and analysis. It is widely used for tasks such as data cleaning, transformation, exploration, and visualization. Some basic concepts of Pandas are:

- DataFrame: The core of pandas is the DataFrame object, which is a two-dimensional labeled data structure with columns of potentially different types. It's similar to a spreadsheet or SQL table.
- Series: Pandas also provides the Series object, which is a one-dimensional labeled array capable of holding any data type.
- Indexing and Selection: Pandas provides intuitive methods for indexing and selecting data within DataFrames and Series.
- Data Alignment: Pandas automatically aligns data based on the label (index), which makes it easy to work with incomplete or differently indexed data.
- Data Cleaning and Transformation: Pandas offers powerful tools for handling missing data, reshaping, merging, grouping, and aggregating datasets.
- Input/Output: Pandas supports reading and writing data from various file formats, including CSV, Excel, SQL databases, and more.

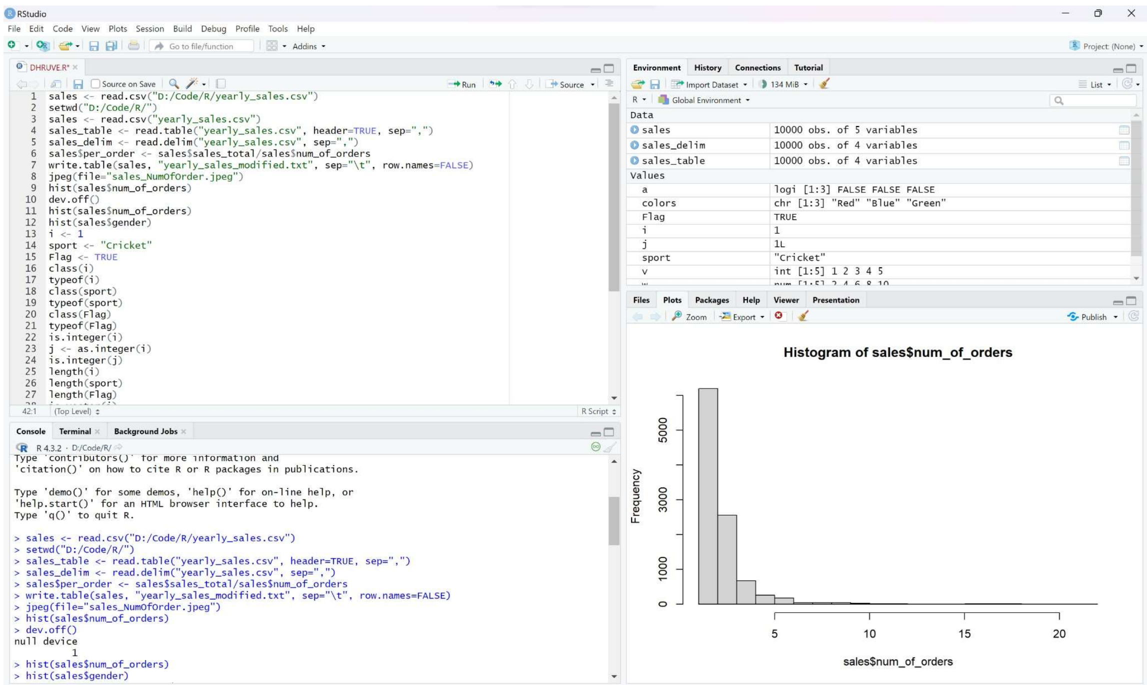
SciPy:

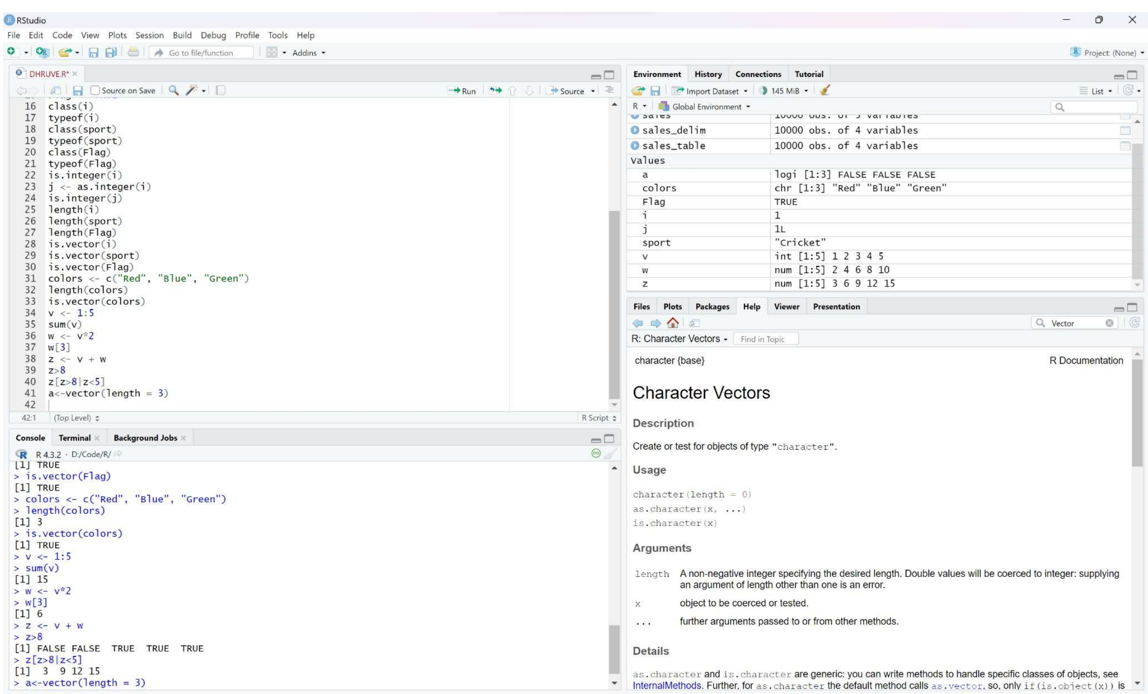
SciPy is an open-source Python library that is used for scientific and technical computing. It builds upon the capabilities of NumPy and provides additional modules for optimization, interpolation, integration, linear algebra, signal processing, and much more. SciPy is widely used in scientific research, engineering, and data analysis. Some basic concepts of SciPy are:

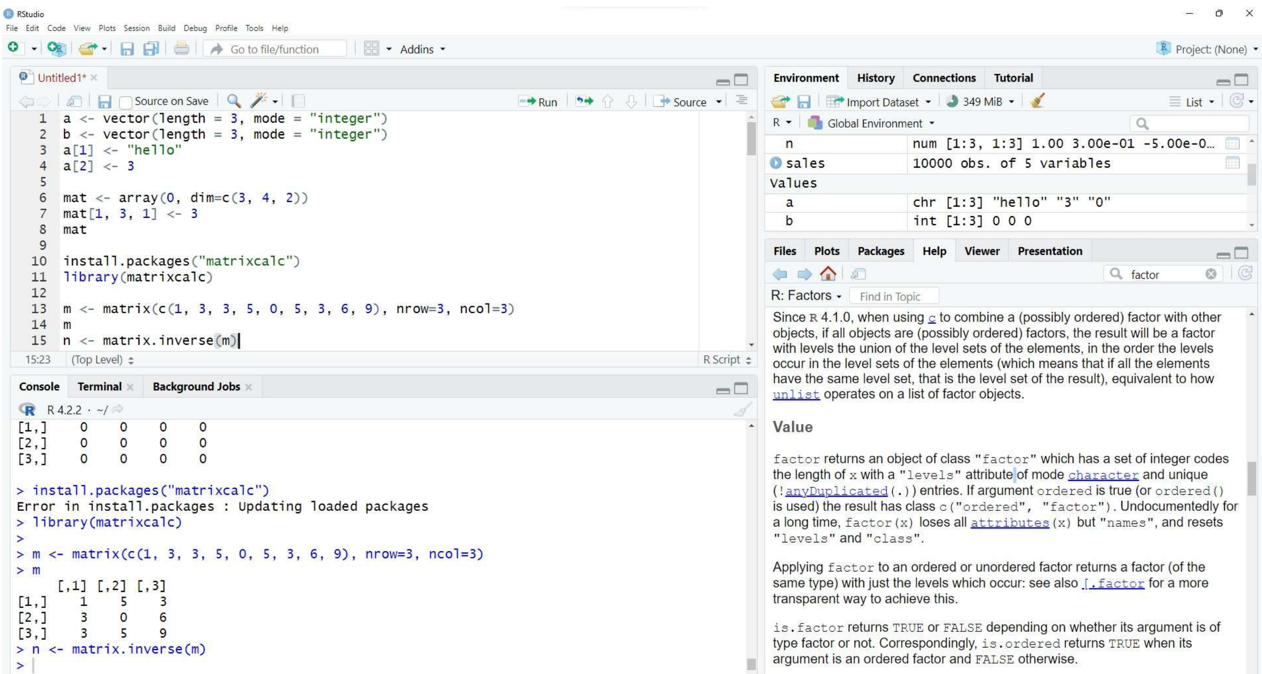
- Integration: SciPy provides functions for numerical integration, including single and multiple integrals, with support for both definite and indefinite integrals.
- Optimization: SciPy offers a wide range of optimization algorithms for finding the minimum (or maximum) of functions. These include methods for unconstrained and constrained optimization.
- Interpolation: SciPy provides functions for interpolating data points using various methods, such as linear, polynomial, spline, and more.
- Linear Algebra: SciPy includes functions for performing various linear algebra operations, such as solving linear equations, eigenvalue problems, singular value decomposition (SVD), and matrix factorization.

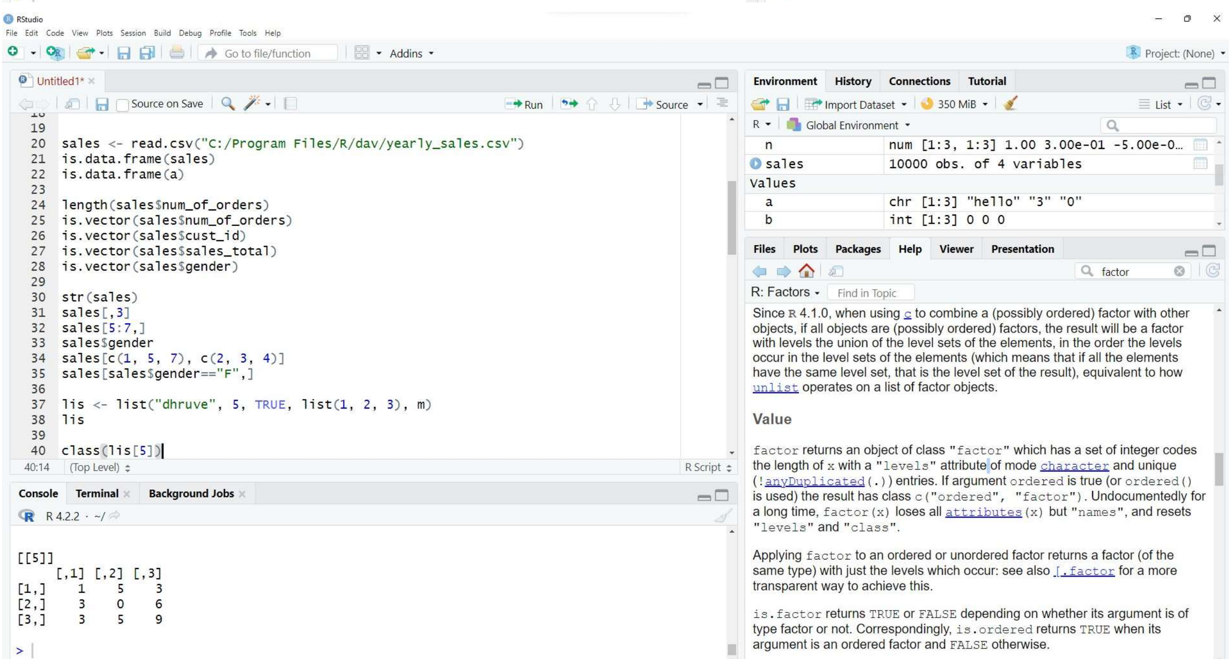
- Signal Processing: SciPy offers tools for processing and analyzing signals, including filtering, convolution, Fourier analysis, wavelet transforms, and more.

Code:






RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1*
Environment History Connections Tutorial
Global Environment
n num [1:3, 1:3] 1.00 3.00e-01 -5.00e-0…
sales 10000 obs. of 5 variables
Values
a chr [1:3] "hello" "3" "0"
b int [1:3] 0 0 0
a <- vector(length = 3, mode = "integer")
b <- vector(length = 3, mode = "integer")
a[1] <- "hello"
a[2] <- 3
mat <- array(0, dim=c(3, 4, 2))
mat[1, 3, 1] <- 3
mat
install.packages("matrixcalc")
library(matrixcalc)
m <- matrix(c(1, 3, 3, 5, 0, 5, 3, 6, 9), nrow=3, ncol=3)
m
n <- matrix.inverse(m)
15:23 (Top Level) R Script
Console Terminal Background Jobs
R 4.2.2 · ~/
[1,] 0 0 0 0
[2,] 0 0 0 0
[3,] 0 0 0 0
> install.packages("matrixcalc")
Error in install.packages : Updating loaded packages
> library(matrixcalc)
>
> m <- matrix(c(1, 3, 3, 5, 0, 5, 3, 6, 9), nrow=3, ncol=3)
> m
[,1] [,2] [,3]
[1,] 1 5 3
[2,] 3 0 6
[3,] 3 5 9
> n <- matrix.inverse(m)
>
Files Plots Packages Help Viewer Presentation
R: Factors
Value

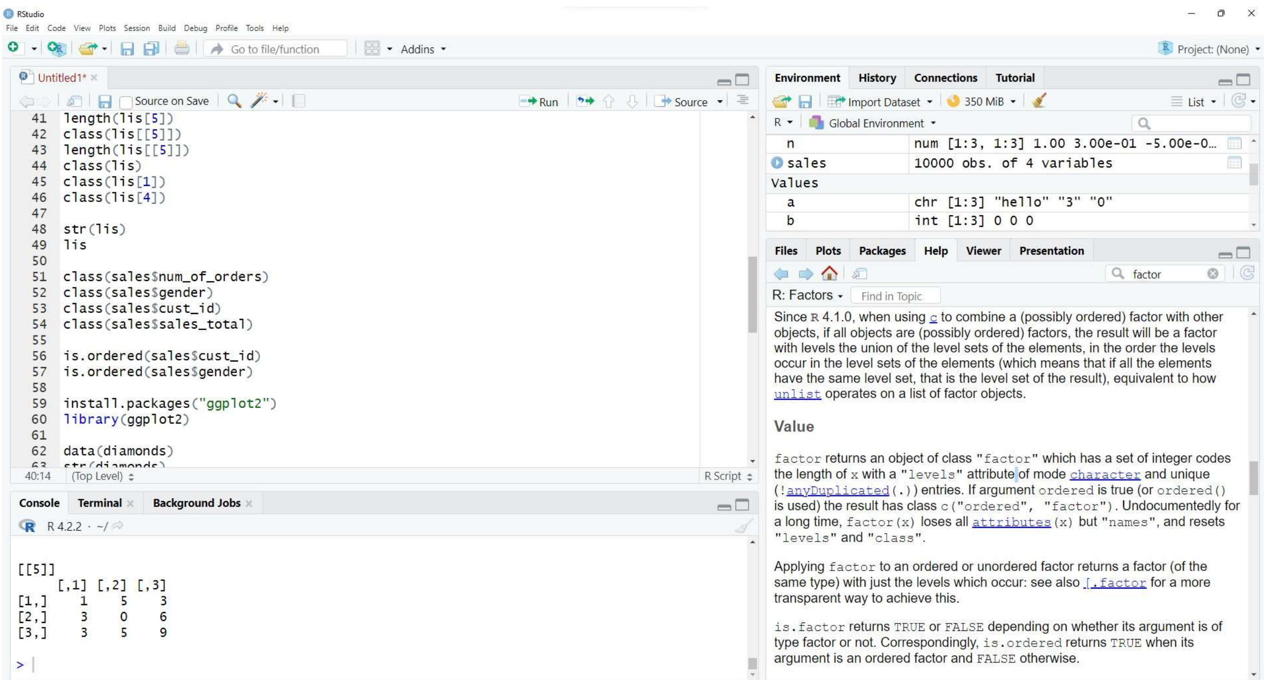

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1*
Environment History Connections Tutorial
Global Environment
n num [1:3, 1:3] 1.00 3.00e-01 -5.00e-0…
sales 10000 obs. of 4 variables
Values
a chr [1:3] "hello" "3" "0"
b int [1:3] 0 0 0
sales <- read.csv("C:/Program Files/R/dav/yearly_sales.csv")
is.data.frame(sales)
is.data.frame(a)
length(sales$num_of_orders)
is.vector(sales$num_of_orders)
is.vector(sales$cust_id)
is.vector(sales$sales_total)
is.vector(sales$gender)
str(sales)
sales[,3]
sales[5:7,]
sales$gender
sales[c(1, 5, 7), c(2, 3, 4)]
sales[sales$gender=="F",]
lis <- list("dhruve", 5, TRUE, list(1, 2, 3), m)
lis
class(lis[5])
40:14 (Top Level) R Script
Console Terminal Background Jobs
R 4.2.2 · ~/
[[5]]
[,1] [,2] [,3]
[1,] 1 5 3
[2,] 3 0 6
[3,] 3 5 9
>
Files Plots Packages Help Viewer Presentation
R: Factors
Value

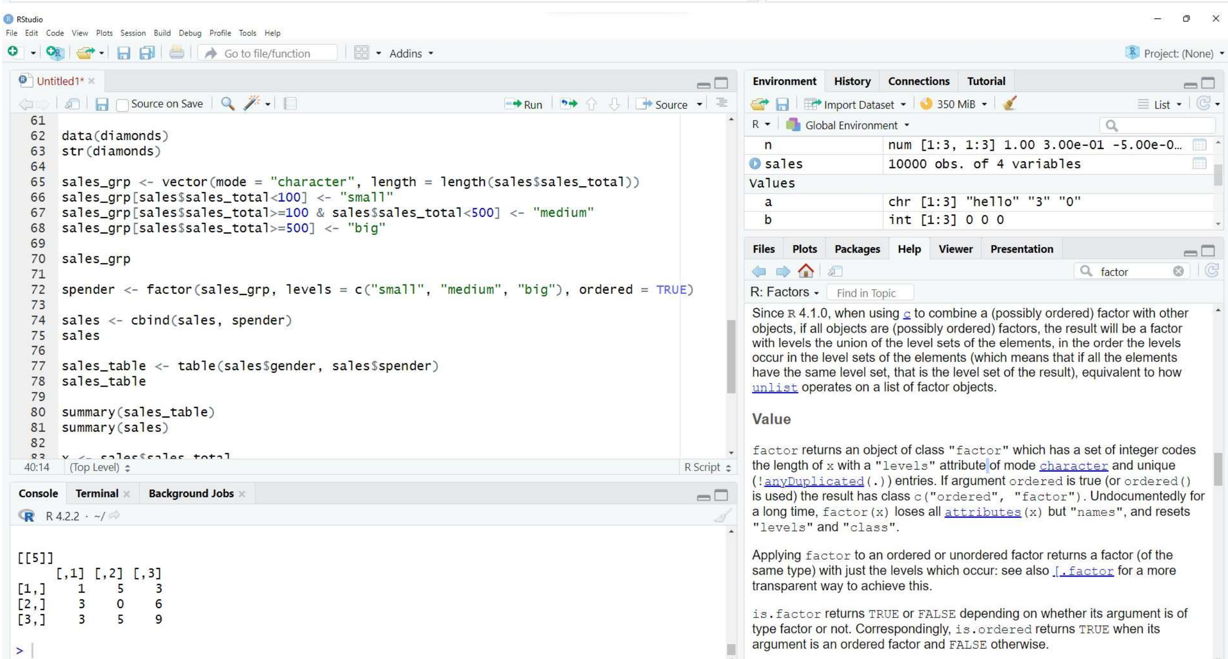
Mahavir Education Trust's

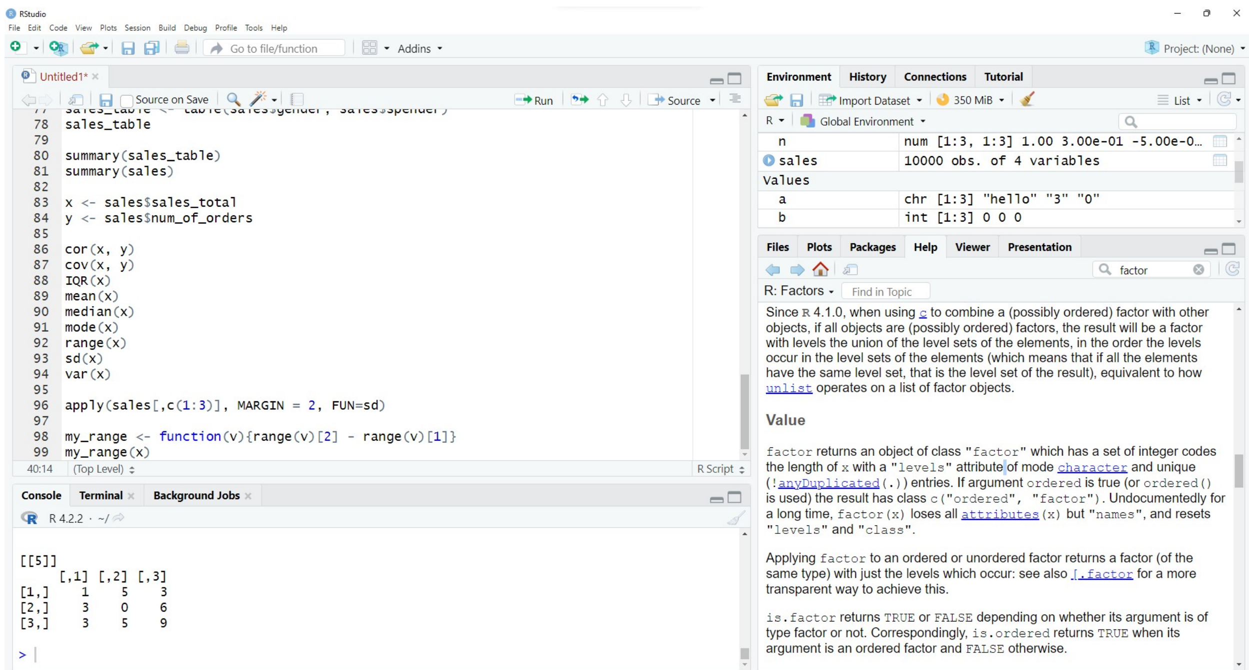
SHAH & ANCHOR KUTCHHI ENGINEERING COLLEGE

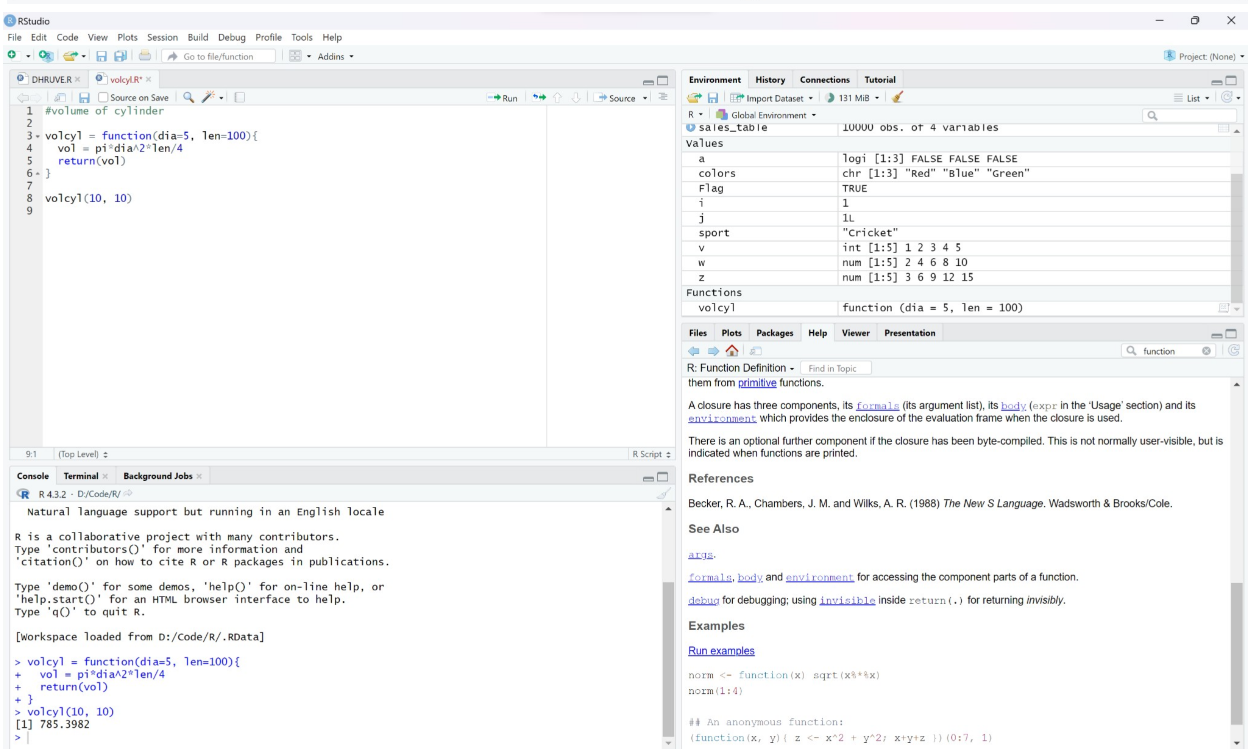
Chembur, Mumbai - 400 088

UG Program in Artificial Intelligence and Data Science






RStudio
Untitled1*
sales_table
summary(sales_table)
summary(sales)
x <- sales$sales_total
y <- sales$num_of_orders
cor(x, y)
cov(x, y)
IQR(x)
mean(x)
median(x)
mode(x)
range(x)
sd(x)
var(x)
apply(sales[,c(1:3)], MARGIN = 2, FUN=sd)
my_range <- function(v){range(v)[2] - range(v)[1]}
my_range(x)
Console
R 4.2.2 · ~/
[[5]]
[,1] [,2] [,3]
[1,] 1 5 3
[2,] 3 0 6
[3,] 3 5 9
Environment
n num [1:3, 1:3] 1.00 3.00e-01 -5.00e-0...
sales 10000 obs. of 4 variables
Values
a chr [1:3] "hello" "3" "0"
b int [1:3] 0 0 0
R: Factors
Value
factor returns an object of class "factor" which has a set of integer codes the length of x with a "levels" attribute of mode character and unique (!anyDuplicated(.)) entries.

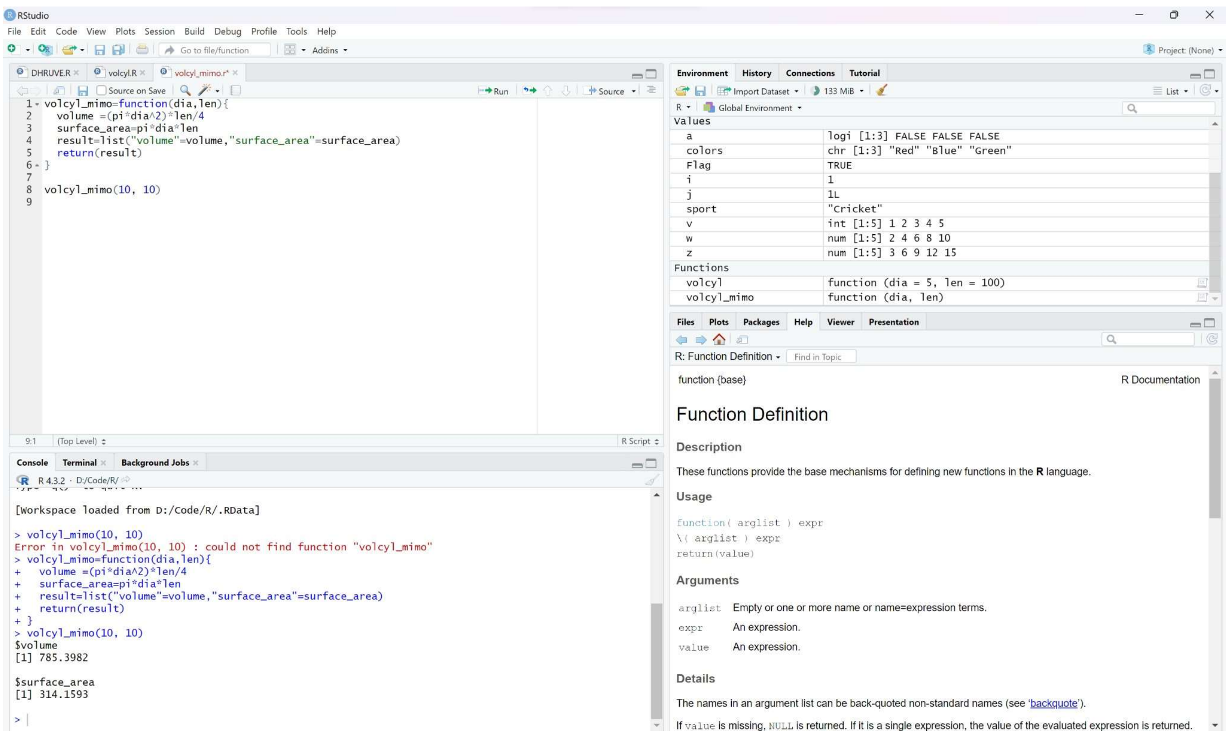

RStudio
DHRUVE.R
volcyl.R*
#volume of cylinder
volcyl = function(dia=5, len=100){
vol = pi*dia^2*len/4
return(vol)
}
volcyl(10, 10)
Console
R 4.3.2 · D:/Code/R/
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[Workspace loaded from D:/Code/R/.RData]
> volcyl = function(dia=5, len=100){
+ vol = pi*dia^2*len/4
+ return(vol)
+ }
> volcyl(10, 10)
[1] 785.3982
Environment
Values
a logi [1:3] FALSE FALSE FALSE
colors chr [1:3] "Red" "Blue" "Green"
Flag TRUE
i 1
j 1L
sport "Cricket"
v int [1:5] 1 2 3 4 5
w num [1:5] 2 4 6 8 10
z num [1:5] 3 6 9 12 15
Functions
volcyl function (dia = 5, len = 100)
R: Function Definition
References
See Also
Examples
Run examples

Mahavir Education Trust's

SHAH & ANCHOR KUTCHHI ENGINEERING COLLEGE

Chembur, Mumbai - 400 088

UG Program in Artificial Intelligence and Data Science



In [1]:
```
pip install scipy
```

```
Requirement already satisfied: scipy in c:\users\tina maru\anaconda3\lib\site-package
s (1.9.1)
Requirement already satisfied: numpy<1.25.0,>=1.18.5 in c:\users\tina maru\anaconda3
\lib\site-packages (from scipy) (1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
from scipy import constants
```

In [3]:
```
from scipy import constants

print(constants.liter)
```

```
0.001
```

In [4]:
```
print(constants.pi)
```

```
3.141592653589793
```

In [5]:
```
print(dir(constants))
```

```
['Avogadro', 'Boltzmann', 'Btu', 'Btu_IT', 'Btu_th', 'ConstantWarning', 'G', 'Julian_
year', 'N_A', 'Planck', 'R', 'Rydberg', 'Stefan_Boltzmann', 'Wien', '__all__', '__bui
ltins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package_
_', '__path__', '__spec__', '_codata', '_constants', '_obsolete_constants', 'acre',
'alpha', 'angstrom', 'arcmin', 'arcminute', 'arcsec', 'arcsecond', 'astronomical_uni
t', 'atm', 'atmosphere', 'atomic_mass', 'atto', 'au', 'bar', 'barrel', 'bbl', 'blob',
'c', 'calorie', 'calorie_IT', 'calorie_th', 'carat', 'centi', 'codata', 'constants',
'convert_temperature', 'day', 'deci', 'degree', 'degree_Fahrenheit', 'deka', 'dyn',
'dyne', 'e', 'eV', 'electron_mass', 'electron_volt', 'elementary_charge', 'epsilon_
0', 'erg', 'exa', 'exbi', 'femto', 'fermi', 'find', 'fine_structure', 'fluid_ounce',
'fluid_ounce_US', 'fluid_ounce_imp', 'foot', 'g', 'gallon', 'gallon_US', 'gallon_im
p', 'gas_constant', 'gibi', 'giga', 'golden', 'golden_ratio', 'grain', 'gram', 'gravi
tational_constant', 'h', 'hbar', 'hectare', 'hecto', 'horsepower', 'hour', 'hp', 'inc
h', 'k', 'kgf', 'kibi', 'kilo', 'kilogram_force', 'kmh', 'knot', 'lambda2nu', 'lb',
'lbf', 'light_year', 'liter', 'litre', 'long_ton', 'm_e', 'm_n', 'm_p', 'm_u', 'mac
h', 'mebi', 'mega', 'metric_ton', 'micro', 'micron', 'mil', 'mile', 'milli', 'minut
e', 'mmHg', 'mph', 'mu_0', 'nano', 'nautical_mile', 'neutron_mass', 'nu2lambda', 'oun
ce', 'oz', 'parsec', 'pebi', 'peta', 'physical_constants', 'pi', 'pico', 'point', 'po
und', 'pound_force', 'precision', 'proton_mass', 'psi', 'pt', 'short_ton', 'sigma',
'slinch', 'slug', 'speed_of_light', 'speed_of_sound', 'stone', 'survey_foot', 'survey
_mile', 'tebi', 'tera', 'test', 'ton_TNT', 'torr', 'troy_ounce', 'troy_pound', 'u',
'unit', 'value', 'week', 'yard', 'year', 'yobi', 'yocto', 'yotta', 'zebi', 'zepto',
'zero_Celsius', 'zetta']
```

In [6]:
```
print(constants.yotta)    #1e+24
print(constants.zetta)    #1e+21
print(constants.exa)      #1e+18
print(constants.peta)     #1000000000000000.0
print(constants.tera)     #1000000000000.0
print(constants.giga)     #1000000000.0
print(constants.mega)     #1000000.0
print(constants.kilo)     #1000.0
print(constants.hecto)    #100.0
print(constants.deka)     #10.0
print(constants.deci)     #0.1
print(constants.centi)    #0.01
print(constants.milli)    #0.001
```

```
print(constants.micro)    #1e-06
print(constants.nano)     #1e-09
print(constants.pico)     #1e-12
print(constants.femto)    #1e-15
print(constants.atto)     #1e-18
print(constants.zepto)    #1e-21
```

```
1e+24
1e+21
1e+18
1000000000000000.0
1000000000000.0
1000000000.0
1000000.0
1000.0
100.0
10.0
0.1
0.01
0.001
1e-06
1e-09
1e-12
1e-15
1e-18
1e-21
```

In [7]:
```
print(constants.kibi)    #1024
print(constants.mebi)    #1048576
print(constants.gibi)    #1073741824
print(constants.tebi)    #1099511627776
print(constants.pebi)    #1125899906842624
print(constants.exbi)    #1152921504606846976
print(constants.zebi)    #1180591620717411303424
print(constants.yobi)    #1208925819614629174706176
```

```
1024
1048576
1073741824
1099511627776
1125899906842624
1152921504606846976
1180591620717411303424
1208925819614629174706176
```

In [8]:
```
print(constants.gram)        #0.001
print(constants.metric_ton)  #1000.0
print(constants.grain)       #6.479891e-05
print(constants.lb)          #0.45359236999999997
print(constants.pound)       #0.45359236999999997
print(constants.oz)          #0.028349523124999998
print(constants.ounce)       #0.028349523124999998
print(constants.stone)       #6.3502931799999995
print(constants.long_ton)    #1016.0469088
print(constants.short_ton)   #907.1847399999999
print(constants.troy_ounce)  #0.031103476799999998
print(constants.troy_pound)  #0.37324172159999996
print(constants.carat)       #0.0002
print(constants.atomic_mass) #1.66053904e-27
```

```
print(constants.m_u)        #1.66053904e-27
print(constants.u)          #1.66053904e-27
```

```
0.001
1000.0
6.479891e-05
0.45359236999999997
0.45359236999999997
0.028349523124999998
0.028349523124999998
6.3502931799999995
1016.0469088
907.1847399999999
0.031103476799999998
0.37324172159999996
0.0002
1.6605390666e-27
1.6605390666e-27
1.6605390666e-27
```

In [9]:

```
print(constants.degree)     #0.017453292519943295
print(constants.arcmin)     #0.0002908882086657216
print(constants.arcminute) #0.0002908882086657216
print(constants.arcsec)     #4.84813681109536e-06
print(constants.arcsecond) #4.84813681109536e-06
```

```
0.017453292519943295
0.0002908882086657216
0.0002908882086657216
4.84813681109536e-06
4.84813681109536e-06
```

In [10]:

```
print(constants.minute)      #60.0
print(constants.hour)        #3600.0
print(constants.day)         #86400.0
print(constants.week)        #604800.0
print(constants.year)        #31536000.0
print(constants.Julian_year) #31557600.0
```

```
60.0
3600.0
86400.0
604800.0
31536000.0
31557600.0
```

In [11]:

```
print(constants.inch)              #0.0254
print(constants.foot)              #0.30479999999999996
print(constants.yard)              #0.9143999999999999
print(constants.mile)              #1609.3439999999998
print(constants.mil)               #2.5399999999999997e-05
print(constants.pt)                #0.00035277777777777776
print(constants.point)             #0.00035277777777777776
print(constants.survey_foot)       #0.3048006096012192
print(constants.survey_mile)       #1609.3472186944373
print(constants.nautical_mile)     #1852.0
print(constants.fermi)             #1e-15
print(constants.angstrom)          #1e-10
print(constants.micron)            #1e-06
print(constants.au)                #149597870691.0
print(constants.astronomical_unit) #149597870691.0
```

```
print(constants.light_year)       #9460730472580800.0
print(constants.parsec)           #3.0856775813057292e+16
```

```
0.0254
0.30479999999999996
0.9143999999999999
1609.3439999999998
2.5399999999999997e-05
0.00035277777777777776
0.00035277777777777776
0.3048006096012192
1609.3472186944373
1852.0
1e-15
1e-10
1e-06
149597870700.0
149597870700.0
9460730472580800.0
3.085677581491367e+16
```

In [12]:
```
print(constants.atm)          #101325.0
print(constants.atmosphere)   #101325.0
print(constants.bar)          #100000.0
print(constants.torr)         #133.32236842105263
print(constants.mmHg)         #133.32236842105263
print(constants.psi)          #6894.757293168361
```

```
101325.0
101325.0
100000.0
133.32236842105263
133.32236842105263
6894.757293168361
```

In [13]:
```
print(constants.hectare) #10000.0
print(constants.acre)    #4046.8564223999992
```

```
10000.0
4046.8564223999992
```

In [14]:
```
print(constants.liter)             #0.001
print(constants.litre)             #0.001
print(constants.gallon)            #0.0037854117839999997
print(constants.gallon_US)         #0.0037854117839999997
print(constants.gallon_imp)        #0.00454609
print(constants.fluid_ounce)       #2.9573529562499998e-05
print(constants.fluid_ounce_US)    #2.9573529562499998e-05
print(constants.fluid_ounce_imp)   #2.84130625e-05
print(constants.barrel)            #0.15898729492799998
print(constants.bbl)               #0.15898729492799998
```

```
0.001
0.001
0.0037854117839999997
0.0037854117839999997
0.00454609
2.9573529562499998e-05
2.9573529562499998e-05
2.84130625e-05
0.15898729492799998
0.15898729492799998
```

In [15]:
```
print(constants.kmh)            #0.2777777777777778
print(constants.mph)            #0.44703999999999994
print(constants.mach)           #340.5
print(constants.speed_of_sound) #340.5
print(constants.knot)           #0.5144444444444445
```

```
0.2777777777777778
0.44703999999999994
340.5
340.5
0.5144444444444445
```

In [16]:
```
print(constants.zero_Celsius)      #273.15
print(constants.degree_Fahrenheit) #0.5555555555555556
```

```
273.15
0.5555555555555556
```

In [17]:
```
print(constants.eV)            #1.6021766208e-19
print(constants.electron_volt) #1.6021766208e-19
print(constants.calorie)       #4.184
print(constants.calorie_th)    #4.184
print(constants.calorie_IT)    #4.1868
print(constants.erg)           #1e-07
print(constants.Btu)           #1055.05585262
print(constants.Btu_IT)        #1055.05585262
print(constants.Btu_th)        #1054.3502644888888
print(constants.ton_TNT)       #4184000000.0
```

```
1.602176634e-19
1.602176634e-19
4.184
4.184
4.1868
1e-07
1055.05585262
1055.05585262
1054.3502644888888
4184000000.0
```

In [18]:
```
print(constants.hp)         #745.6998715822701
print(constants.horsepower) #745.6998715822701
```

```
745.6998715822701
745.6998715822701
```

In [19]:
```
print(constants.dyn)            #1e-05
print(constants.dyne)           #1e-05
print(constants.lbf)            #4.4482216152605
print(constants.pound_force)    #4.4482216152605
```

```
print(constants.kgf)              #9.80665
print(constants.kilogram_force)   #9.80665
```

```
1e-05
1e-05
4.4482216152605
4.4482216152605
9.80665
9.80665
```

In [20]:
```
from scipy.optimize import root
from math import cos

def eqn(x):
  return x + cos(x)

myroot = root(eqn, 0)

print(myroot.x)
```

```
[-0.73908513]
```

In [21]:
```
print(myroot)
```

```
    fjac: array([[-1.]])
     fun: array([0.])
 message: 'The solution converged.'
    nfev: 9
     qtf: array([-2.66786593e-13])
       r: array([-1.67361202])
  status: 1
 success: True
       x: array([-0.73908513])
```

In [22]:
```
from scipy.optimize import minimize

def eqn(x):
  return x**2 + x + 2

mymin = minimize(eqn, 0, method='BFGS')

print(mymin)
```

```
      fun: 1.75
 hess_inv: array([[0.50000001]])
      jac: array([0.])
  message: 'Optimization terminated successfully.'
     nfev: 8
      nit: 2
     njev: 4
   status: 0
  success: True
        x: array([-0.50000001])
```

In [23]:
```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])

print(csr_matrix(arr))
```

```
  (0, 5)	1
  (0, 6)	1
  (0, 8)	2
```

In [24]:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

print(csr_matrix(arr).data)
```

```
[1 1 2]
```

In [25]:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

print(csr_matrix(arr).count_nonzero())
```

```
3
```

In [26]:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

mat = csr_matrix(arr)
mat.eliminate_zeros()

print(mat)
```

```
  (1, 2)	1
  (2, 0)	1
  (2, 2)	2
```

In [29]:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

mat = csr_matrix(arr)
mat.sum_duplicates()

print(mat)
```

```
  (1, 2)	1
  (2, 0)	1
  (2, 2)	2
```

In [30]:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

newarr = csr_matrix(arr).tocsc()

print(newarr)
```

```
  (2, 0)	1
  (1, 2)	1
  (2, 2)	2
```

In [31]:

```
import numpy as np
from scipy.sparse.csgraph import connected_components
from scipy.sparse import csr_matrix

arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])

newarr = csr_matrix(arr)

print(connected_components(newarr))
```

```
(1, array([0, 0, 0]))
```

In [ ]:

Conclusion:

Hence, we successfully introduced data analytics in R and Python.