Plot No.2, Sector 17-A, Yamuna Expressway,
Greater Noida, Gautam Buddha Nagar, U.P., India

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY RECORD

NAME : ....

SECTION / SEM : ....

ADMISSION NO. : ....

ROLL NO. : ....

SUBJECT CODE : ....

SUBJECT NAME : ....

GALGOTIAS UNIVERSITY

Plot No.2, Sector 17-A, Yamuna Expressway,
Greater Noida, Gautam Buddha Nagar, U.P., India

Roll Number: ***Admission Number:***

BONAFIDE CERTIFICATE

Certified to be the bonafide record of the work done by ………………………………............................. of V Semester Third Year B.Tech. Degree course in Galgotias University of School of Computing Science and Engineering in Computer Science and Engineering Department in the ……………………………….……………….…………...... Laboratory during the academic year 2023-2024.

HEAD OF DEPARTMENT   **FACULTY IN-CHARGE**

Submitted for the University Examination held on ………………………………………...

INTERNAL EXAMNINER   **EXTERNAL EXAMINER**

Table of Contents

Ex. No.: 1	Perform basic data exploration tasks such as loading data, summary statistics, and handling missing values using R.
Date:	

Aim

To perform basic data exploration tasks such as loading data, summary statistics, and handling missing values using R.

Program and Output

```
data("mtcars")
>
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
>
> summary(mtcars)
      mpg             cyl             disp             hp             drat      
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930  
       wt             qsec             vs               am              gear      
 Min.   :1.513   Min.   :14.50   Min.   :0.0000   Min.   :0.0000   Min.   :3.000  
 1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000  
 Median :3.325   Median :17.71   Median :0.0000   Median :0.0000   Median :4.000  
 Mean   :3.217   Mean   :17.85   Mean   :0.4375   Mean   :0.4062   Mean   :3.688  
 3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000  
 Max.   :5.424   Max.   :22.90   Max.   :1.0000   Max.   :1.0000   Max.   :5.000  
      carb      
 Min.   :1.000  
 1st Qu.:2.000  
 Median :2.000  
 Mean   :2.812  
 3rd Qu.:4.000  
 Max.   :8.000  
>
>
> sum(is.na(mtcars))
[1] 0
>
>
> mtcars <- mtcars %>%
+   mutate(across(where(is.numeric), ~ifelse(is.na(.), mean(., na.rm = TRUE), .)))
>
>
> mtcars <- na.omit(mtcars)
>
>
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
>
> print("Agrima Saxena - 21scse1280023")
> Agrima Saxena - 21scse1280023
```

Result

Thus, the basic data exploration tasks such as loading data, summary statistics, and handling missing values using R is performed and output was verified.

Ex. No.: 2	Explore and utilize software tools like Pandas for data manipulation and analysis using R.
Date:	

Aim

To explore and utilize software tools like Pandas for data manipulation and analysis using R.

Program and Output

```
> # Loading necessary libraries
> if (!require(dplyr)) install.packages("dplyr")
> if (!require(tidyr)) install.packages("tidyr")
> if (!require(data.table)) install.packages("data.table")
> library(dplyr)
> library(tidyr)
> library(data.table)
>
> # Example: Using R's built-in mtcars dataset
> data("mtcars")
>
> # Viewing the first few rows of the dataset
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
>
> # Selecting specific columns
> selected_data <- select(mtcars, mpg, cyl, hp)
>
> # Filtering rows based on a condition
> filtered_data <- filter(mtcars, cyl == 6)
>
> # Creating a new column
> mtcars <- mutate(mtcars, power_to_weight = hp/wt)
>
> # Summarizing data
> summary_stats <- mtcars %>%
+   group_by(cyl) %>%
+   summarise(mean_mpg = mean(mpg),
+             mean_hp = mean(hp))
>
> # Reshaping data from wide to long format
> long_format <- pivot_longer(mtcars, cols = c(mpg, hp, wt), names_to = "measurement", values_to = "value")
>
> # Reshaping data from long to wide format
> wide_format <- pivot_wider(long_format, names_from = measurement, values_from = value)
>
> # Working with data.table
> mtcars_dt <- as.data.table(mtcars)
> mean_hp_per_cyl <- mtcars_dt[, .(mean_hp = mean(hp)), by = .(cyl)]
>
> # Printing results
> print(summary_stats)
# A tibble: 3 × 3
    cyl mean_mpg mean_hp
  <dbl>    <dbl>   <dbl>
1     4     26.7    82.6
2     6     19.7   122.
3     8     15.1   209.
> print(mean_hp_per_cyl)
   cyl   mean_hp
1:   6 122.28571
2:   4  82.63636
3:   8 209.21429
>
> print("Agrima Saxena - 21scse1280023")
> Agrima Saxena - 21scse1280023
```

Result

Thus, the software tools like Pandas for data manipulation and analysis using R was explored and utilized.

Ex. No.: 3	Create various visual aids, such as histograms, box plots, and scatter plots, to gain insights from data using R.
Date:	

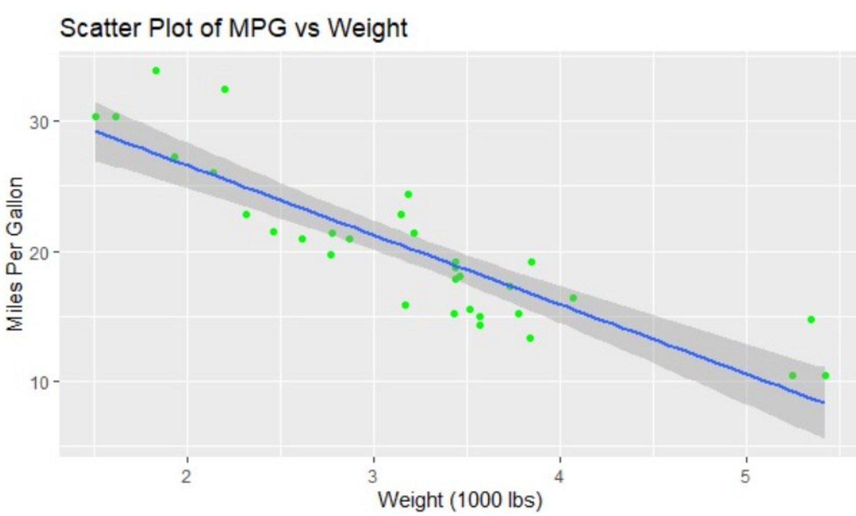
Aim

To create various visual aids, such as histograms, box plots, and scatter plots, to gain insights from data using R.

Program

```
# Loading necessary library
if (!require(ggplot2)) install.packages("ggplot2")
library(ggplot2)
# Using R's built-in mtcars dataset
data("mtcars")
# Scatter Plot of 'mpg' vs 'wt' (weight)
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(color = "green") +
  geom_smooth(method = lm) +  # Adding a linear regression line
  ggtitle("Scatter Plot of MPG vs Weight") +
  xlab("Weight (1000 lbs)") +
  ylab("Miles Per Gallon")
```

Output



Result

Thus, the various visual aids, such as histograms, box plots, and scatter plots, to gain insights from data using R was created and output was verified.

Ex. No.: 4	Merge multiple datasets using common keys using R.
Date:	

Aim

To merge multiple datasets using common keys using R.

Program and Output

```
> # Print merged dataframes
> inner_joined_df
  ID  Name Age        City Salary
1  1 Alice  25    New York  60000
2  2   Bob  30 Los Angeles  70000
3  4 David  40     Chicago  55000
> left_joined_df
  ID    Name Age        City Salary
1  1   Alice  25    New York  60000
2  2     Bob  30 Los Angeles  70000
3  3 Charlie  35        <NA>     NA
4  4   David  40     Chicago  55000
> right_joined_df
  ID  Name Age        City Salary
1  1 Alice  25    New York  60000
2  2   Bob  30 Los Angeles  70000
3  4 David  40     Chicago  55000
4  5  <NA>  NA     Houston  65000
> full_joined_df
  ID    Name Age        City Salary
1  1   Alice  25    New York  60000
2  2     Bob  30 Los Angeles  70000
3  3 Charlie  35        <NA>     NA
4  4   David  40     Chicago  55000
5  5    <NA>  NA     Houston  65000
>
> print("Agrima Saxena - 21scse1280023")
> Agrima Saxena - 21scse1280023
```

Result

Thus, the merge multiple datasets using common keys using R is performed and output was verified.

Ex. No.: 5	Reshape and pivot data using Pandas (Python).
Date:	

Aim

To reshape and pivot data using Pandas (Python).

Program

```
import pandas as pd
data = {'A': ['a', 'b', 'c', 'd'],
 'B': ['x', 'y', 'z', 'w'],
 'C': ['p', 'q', 'r', 's']}
df = pd.DataFrame(data)
print(df)
df = df.pivot(index='A', columns='B', values='C')
print(df)
```

Output

```
   A  B  C
0  a  x  p
1  b  y  q
2  c  z  r
3  d  w  s
B    w    x    y    z
A
a  NaN    p  NaN  NaN
b  NaN  NaN    q  NaN
c  NaN  NaN  NaN    r
d    s  NaN  NaN  NaN
```

Result

Thus, the reshape and pivot data using Pandas (Python) has done and output was verified.

Ex. No.: 6	Aggregate data using group by and aggregation functions in Pandas (Python).
Date:	

Aim

To aggregate data using group by and aggregation functions in Pandas (Python).

Program

```
import pandas as pd
data = {'Name': ['Tom', 'Nick', 'Tom', 'Julia', 'Nick', 'Tom'],
        'Age': [20, 21, 19, 18, 22, 20],
        'City': ['New York', 'London', 'Manchester', 'New York', 'London', 'Manchester']}
df = pd.DataFrame(data)
result_age = df.groupby('Name')['Age'].agg(['mean', 'min', 'max', 'count'])
result_city = df.groupby('Name')['City'].agg(['count'])
result = pd.concat([result_age, result_city], axis=1)
print(result)
print("Agrima Saxena - 21SCSE1280023")
```

OUTPUT

```
            mean  min  max  count  count
Name
Julia  18.000000   18   18      1      1
Nick   21.500000   21   22      2      2
Tom    19.666667   19   20      3      3
Agrima Saxena - 21SCSE1280023
```

Result

Thus, the aggregate data using group by and aggregation functions in Pandas (Python) has done and output was verified.

Ex. No.: 7	Create pivot tables and cross-tabulations using Pandas (Python).
Date:	

Aim

To create pivot tables and cross-tabulations using Pandas (Python).

Program

```
import pandas as pd
# Sample data
data = {'Name': ['Tom', 'Nick', 'Tom', 'Julia', 'Nick', 'Tom'],
        'Age': [20, 21, 19, 18, 22, 20],
        'City': ['New York', 'London', 'Manchester', 'New York', 'London', 'Manchester']}
df = pd.DataFrame(data)
# Pivot Table
pivot_table = df.pivot_table(values='Age', index='Name', columns='City', aggfunc='mean', fill_value=0)
# Cross-tabulation
cross_tab = pd.crosstab(df['Name'], df['City'])
print("Pivot Table:")
print(pivot_table)
print("\nCross-tabulation:")
print(cross_tab)
print("Agrima Saxena - 21SCSE1280023")
```

Output

```
Pivot Table:
City    London  Manchester  New York
Name
Julia      0.0         0.0        18
Nick      21.5         0.0         0
Tom        0.0        19.5        20

Cross-tabulation:
City    London  Manchester  New York
Name
Julia        0           0         1
Nick         2           0         0
Tom          0           2         1
Agrima Saxena - 21SCSE1280023
```

Result

Thus, the pivot tables and cross-tabulations using Pandas (Python) was created and output was verified.

Ex. No.: 8	**Create basic line plots to visualize trends in data using python.**
Date:	

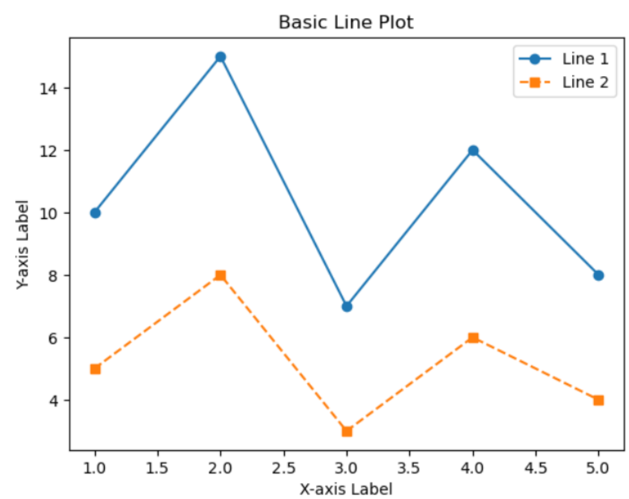
Aim

To create basic line plots to visualize trends in data using python.

Program

```
import matplotlib.pyplot as plt
# Sample data
x = [1, 2, 3, 4, 5]
y1 = [10, 15, 7, 12, 8]
y2 = [5, 8, 3, 6, 4]
# Plotting a basic line plot
plt.plot(x, y1, label='Line 1', marker='o')  # Line with markers
plt.plot(x, y2, label='Line 2', linestyle='--', marker='s')  # Dashed line with square markers
# Adding labels and title
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Basic Line Plot')
# Adding legend
plt.legend()
# Display the plot
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output




```
Agrima Saxena - 21SCSE1280023
```

Result

Thus, the basic line plots to visualize trends in data using python was created and output was verified.

Ex. No.: 9	Generate scatter plots to explore relationships between variables using python.
Date:	

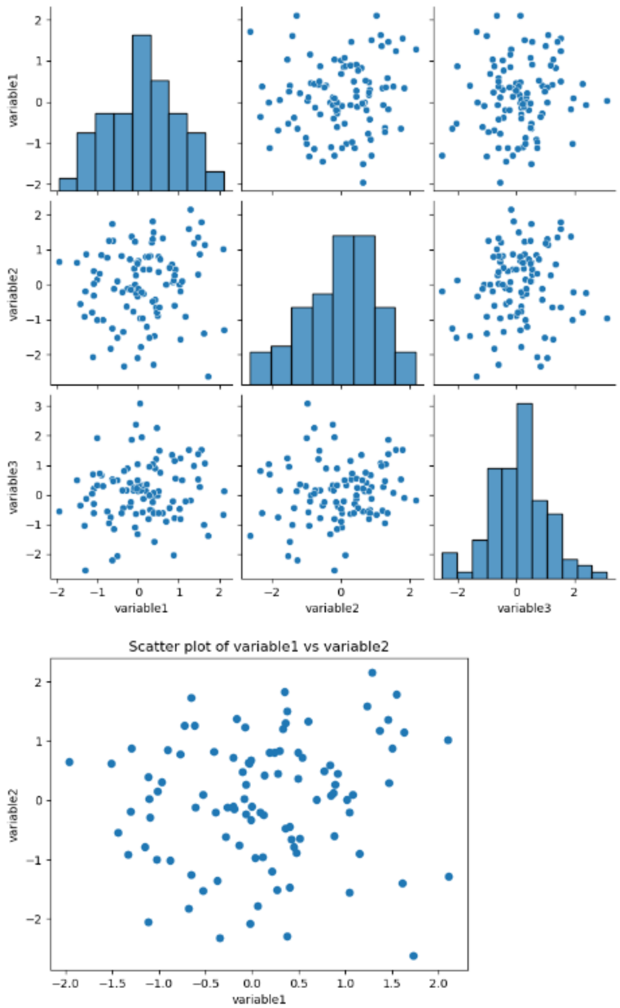
Aim

To generate scatter plots to explore relationships between variables using python.

Program

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = {
   'variable1': np.random.randn(100),
   'variable2': np.random.randn(100),
   'variable3': np.random.randn(100)
}
df = pd.DataFrame(data)
sns.pairplot(df, kind='scatter')
plt.show()
plt.scatter(df['variable1'], df['variable2'])
plt.xlabel('variable1')
plt.ylabel('variable2')
plt.title('Scatter plot of variable1 vs variable2')
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



Agrima Saxena - 21SCSE1280023

Result

Thus, the scatter plots to explore relationships between variables using python was generated and output was verified.

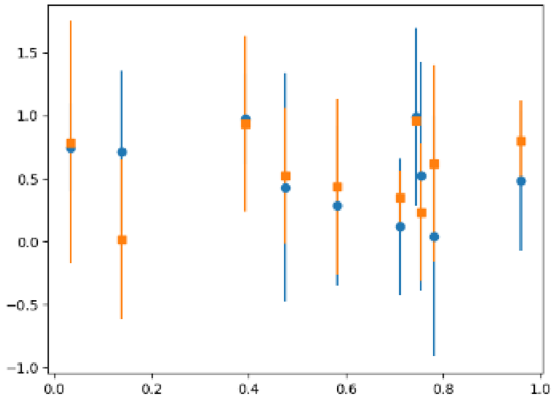
Ex. No.: 10	Create error bars and visualize uncertainty in data using python.
Date:	

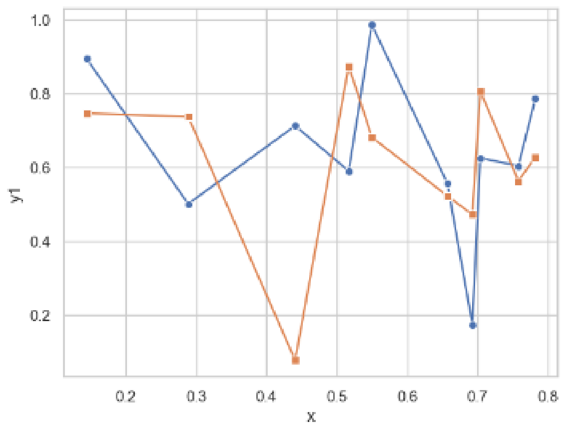
Aim
To create error bars and visualize uncertainty in data using python.

Program

```
import numpy as np
import matplotlib.pyplot as plt
# Assume you have a data array
data = np.random.rand(10, 3)
# Create error bars
errors = np.random.rand(10, 3)
fig, ax = plt.subplots()
ax.errorbar(data[:, 0], data[:, 1], yerr=errors[:, 1], fmt='o')
ax.errorbar(data[:, 0], data[:, 2], yerr=errors[:, 2], fmt='s')
plt.show()
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Assume you have a data array
data = np.random.rand(10, 3)
# Create a DataFrame
df = pd.DataFrame(data, columns=['x', 'y1', 'y2'])
# Create error bars
errors = np.random.rand(10, 3)
df['y1_err'] = errors[:, 1]
df['y2_err'] = errors[:, 2]
# Create error bars plot
sns.set(style="whitegrid")
fig, ax = plt.subplots()
ax = sns.lineplot(data=df, x="x", y="y1", marker="o", ax=ax)
ax = sns.lineplot(data=df, x="x", y="y2", marker="s", ax=ax)
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output





Agrima Saxena - 21SCSE1280023

Result

Thus, the error bars and visualize uncertainty in data using python was created and output was verified.

Ex. No.: 11	Create density and contour plots to visualize data distributions using python.
Date:	

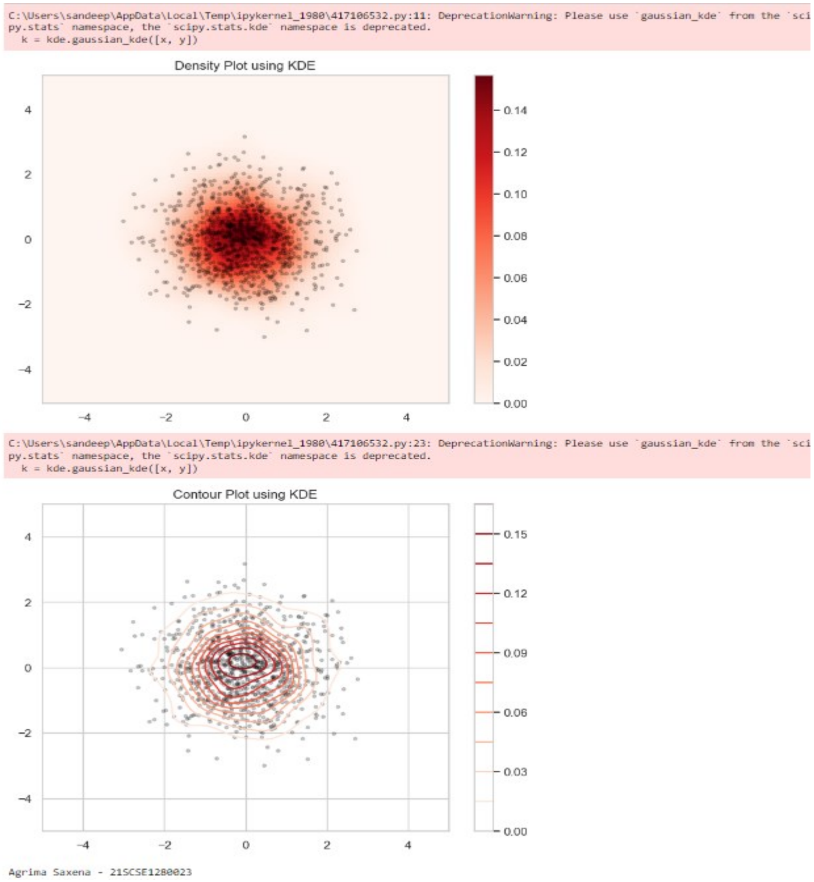
Aim

To create density and contour plots to visualize data distributions using python.

Program

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import kde
# Creating random data for the demonstration
np.random.seed(0)
x = np.random.normal(0, 1, 1000)
y = np.random.normal(0, 1, 1000)
# Density plot using KDE
k = kde.gaussian_kde([x, y])
xi, yi = np.mgrid[-5:5:100j, -5:5:100j]
zi = k(np.vstack([xi.flatten(), yi.flatten()]))
plt.figure(figsize=(8, 6))
plt.pcolormesh(xi, yi, zi.reshape(xi.shape), shading='auto', cmap='Reds')
plt.colorbar()
plt.scatter(x, y, c='black', marker='.', alpha=0.2)
plt.title('Density Plot using KDE')
plt.show()
# Contour plot using KDE
k = kde.gaussian_kde([x, y])
xi, yi = np.mgrid[-5:5:100j, -5:5:100j]
zi = k(np.vstack([xi.flatten(), yi.flatten()]))
plt.figure(figsize=(8, 6))
CS = plt.contour(xi, yi, zi.reshape(xi.shape), levels=14, cmap='Reds')
plt.colorbar()
plt.scatter(x, y, c='black', marker='.', alpha=0.2)
plt.title('Contour Plot using KDE')
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



Result

Thus, the density and contour plots to visualize data distributions using python was created and output was verified.

Ex. No.: 12	**Generate histograms to understand the distribution of numerical variables using python.**
Date:	

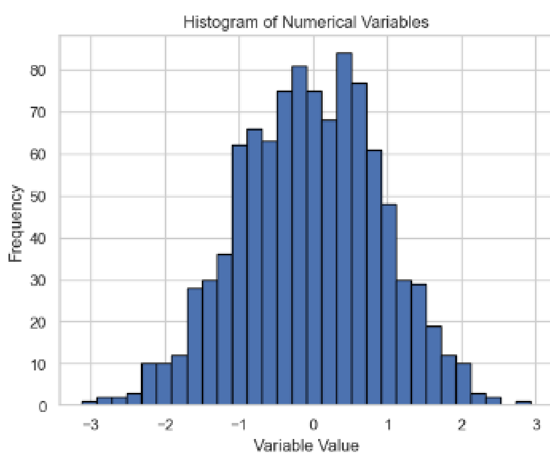
Aim

To generate histograms to understand the distribution of numerical variables using python.

Program

```
import matplotlib.pyplot as plt
import numpy as np
data = np.random.normal(size=1000)
plt.hist(data, bins=30, edgecolor='black')
plt.title('Histogram of Numerical Variables')
plt.xlabel('Variable Value')
plt.ylabel('Frequency')
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



Agrima Saxena - 21SCSE1280023

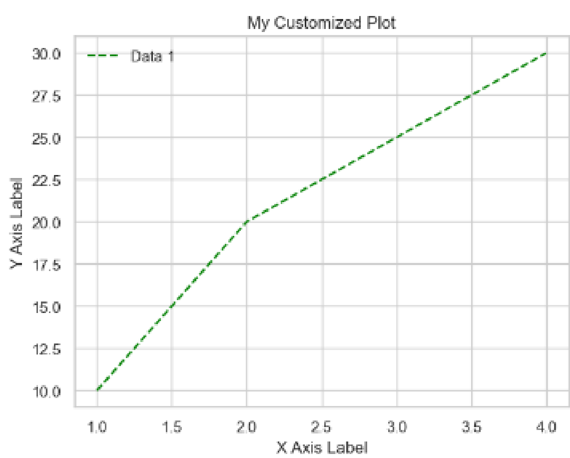
Result

Thus, the histograms to understand the distribution of numerical variables using python was generated and output was verified.

Ex. No.: 13	**Customize plots with legends, colors, and labels using python.**
Date:	

Aim
To customize plots with legends, colors, and labels using python.

Program
```
# Data
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
# Plot
plt.plot(x, y, color='green', linestyle='dashed', label='Data 1')
# Customization
plt.xlabel('X Axis Label')
plt.ylabel('Y Axis Label')
plt.title('My Customized Plot')
plt.legend(loc='upper left', frameon=False)
# Display
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



```
Agrima Saxena - 21SCSE1280023
```

Result
Thus, the customize plots with legends, colors, and labels using python has done and output was verified.

Ex. No.: 14	Create subplots to display multiple plots in a single figure using python.
Date:	

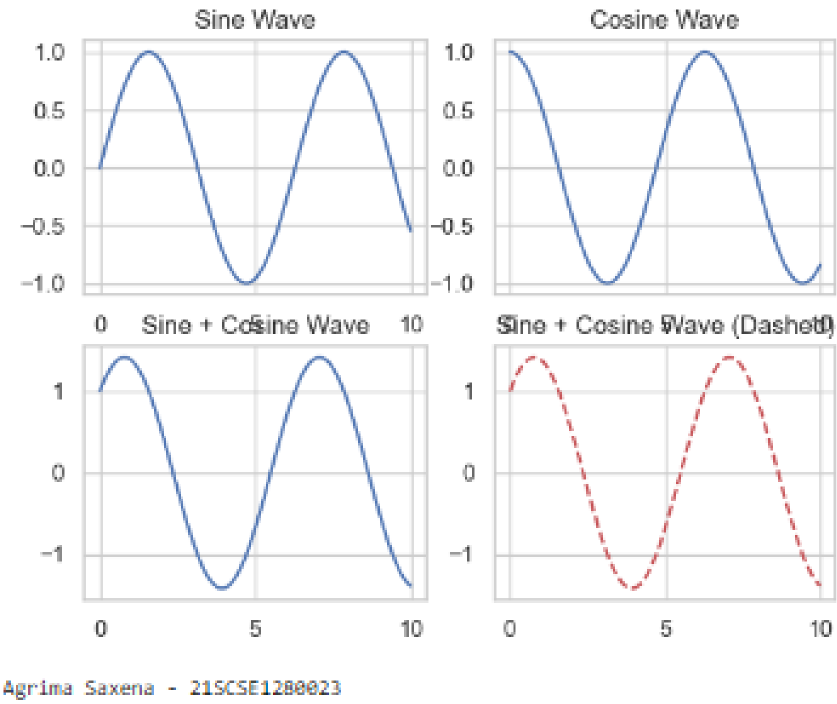
Aim

To create subplots to display multiple plots in a single figure using python.

Program

```
import matplotlib.pyplot as plt
import numpy as np
# Generate some data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.sin(x) + np.cos(x)
# Create subplots
fig, axs = plt.subplots(2, 2)
# Make the first plot in the top left subplot
axs[0, 0].plot(x, y1)
axs[0, 0].set_title('Sine Wave')
# Make the second plot in the top right subplot
axs[0, 1].plot(x, y2)
axs[0, 1].set_title('Cosine Wave')
# Make the third plot in the bottom left subplot
axs[1, 0].plot(x, y3)
axs[1, 0].set_title('Sine + Cosine Wave')
# Make the fourth plot in the bottom right subplot
axs[1, 1].plot(x, y3, 'r--')
axs[1, 1].set_title('Sine + Cosine Wave (Dashed)')
# Display the plot
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



Result

Thus, the subplots to display multiple plots in a single figure using python was created and output was verified.

Ex. No.: 15	Add text and annotations to your visualizations for clarity using python.
Date:	

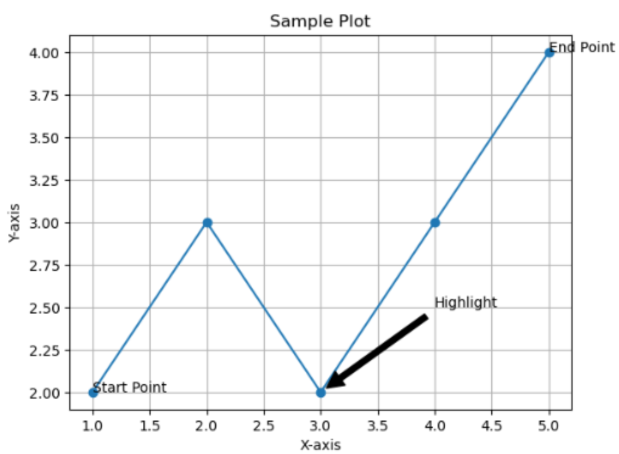
Aim

To add text and annotations to your visualizations for clarity using python.

Program

```
import matplotlib.pyplot as plt
# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 3, 2, 3, 4]
plt.plot(x, y, marker='o') # Basic line plot
# Adding text, title, and labels
plt.title("Sample Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.text(1, 2, "Start Point")
plt.text(5, 4, "End Point")
# Annotating a specific point
plt.annotate('Highlight', xy=(3, 2), xytext=(4, 2.5),
        arrowprops=dict(facecolor='black', shrink=0.05))
# Adding grid
plt.grid(True)
# Show plot
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



Agrima Saxena - 21SCSE1280023

Result

Thus, the add text and annotations to your visualizations for clarity using python has done and output was verified.

Ex. No.: 16	**Explore advanced customization options for Matplotlib plots using python.**
Date:	

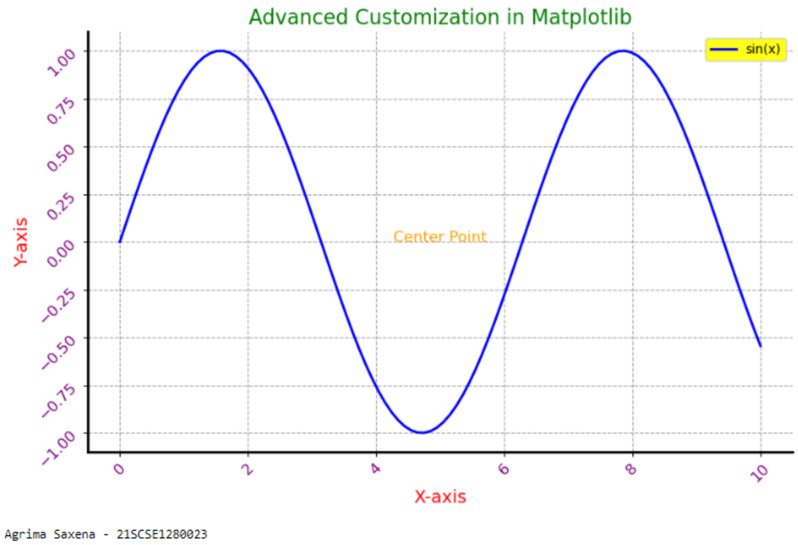
Aim

To explore advanced customization options for Matplotlib plots using python.

Program

```
import matplotlib.pyplot as plt
import numpy as np
# Generating some data
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.figure(figsize=(10, 6)) # Setting the size of the plot
# Creating a plot with a grid, labels, and a title
plt.plot(x, y, label='sin(x)', color='blue', linestyle='-', linewidth=2)
plt.title('Advanced Customization in Matplotlib', fontsize=16, color='green')
plt.xlabel('X-axis', fontsize=14, color='red')
plt.ylabel('Y-axis', fontsize=14, color='red')
# Customizing the tick labels
plt.xticks(fontsize=12, rotation=45, color='purple')
plt.yticks(fontsize=12, rotation=45, color='purple')
# Adding a legend with a customized location and frame
plt.legend(loc='upper right', frameon=True, framealpha=0.9, facecolor='yellow')
# Adding text inside the plot
plt.text(5, 0, 'Center Point', fontsize=12, color='orange', ha='center')
# Customizing the axes appearance
plt.gca().spines['top'].set_color('none')
plt.gca().spines['right'].set_color('none')
plt.gca().spines['left'].set_linewidth(2)
plt.gca().spines['bottom'].set_linewidth(2)
# Adding a grid
plt.grid(True, linestyle='--', color='gray', alpha=0.7)
# Show plot
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



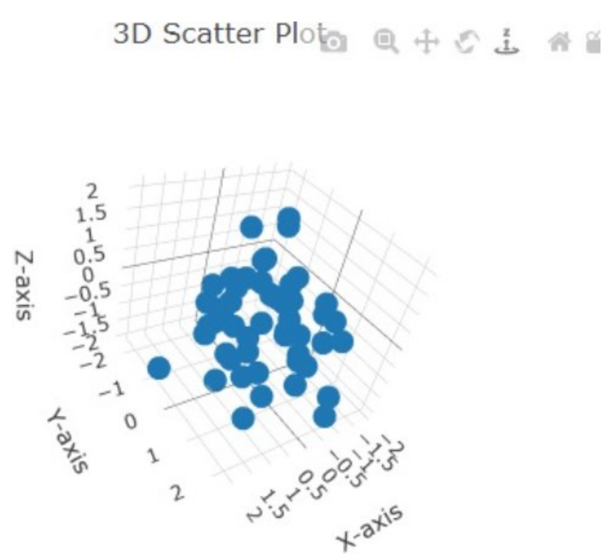
Result

Thus, the advanced customization options for Matplotlib plots using python was explored and output was verified.

Ex. No.: 17	Create 3D plots to visualize three-variable relationships using R.
Date:	

Aim
To create 3D plots to visualize three-variable relationships using R.

Program
```
# Installing and loading the plotly package
if (!require(plotly)) install.packages("plotly", dependencies = TRUE)
library(plotly)
# Creating sample data
set.seed(123)
x <- rnorm(50)
y <- rnorm(50)
z <- rnorm(50)
# Creating a 3D scatter plot
fig <- plot_ly(x = ~x, y = ~y, z = ~z, type = 'scatter3d', mode = 'markers')
# Adding layout
fig <- fig %>% layout(scene = list(xaxis = list(title = 'X-axis'),
                        yaxis = list(title = 'Y-axis'),
                        zaxis = list(title = 'Z-axis')),
               title = "3D Scatter Plot")
# Display the plot
fig
```

Output


Result
Thus, the 3D plots to visualize three-variable relationships using R was created and output was verified.

Ex. No.: 18	**Visualize geographic data using Basemap (Python).**
Date:	

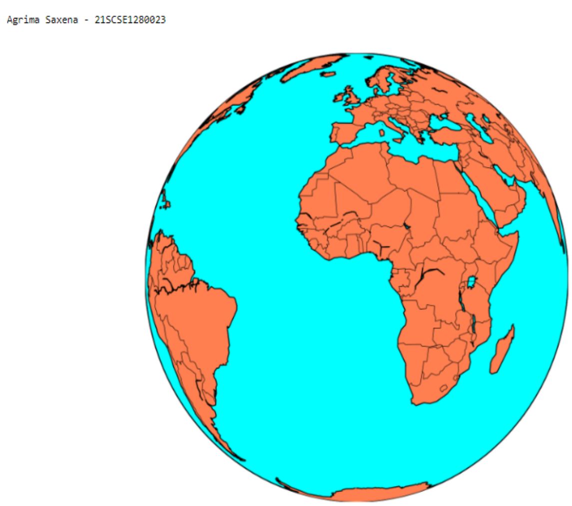
Aim

To visualize geographic data using Basemap (Python).

Program

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
# Create a new map
plt.figure(figsize=(8, 8))
# Set up Basemap: 'ortho' projection for a global view
# lat_0 and lon_0 are the center points
m = Basemap(projection='ortho', lat_0=0, lon_0=0)
# Draw coastlines, countries, and boundaries
m.drawcoastlines()
m.drawcountries()
m.drawmapboundary(fill_color='aqua')
# Fill the continents and the oceans
m.fillcontinents(color='coral', lake_color='aqua')
# Show the plot
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



Result

Thus, the visualize geographic data using Basemap (Python) has done and output was verified.

Ex. No.: 19	**Use Seaborn, a high-level data visualization library, to create stylish and informative plots using python.**
Date:	

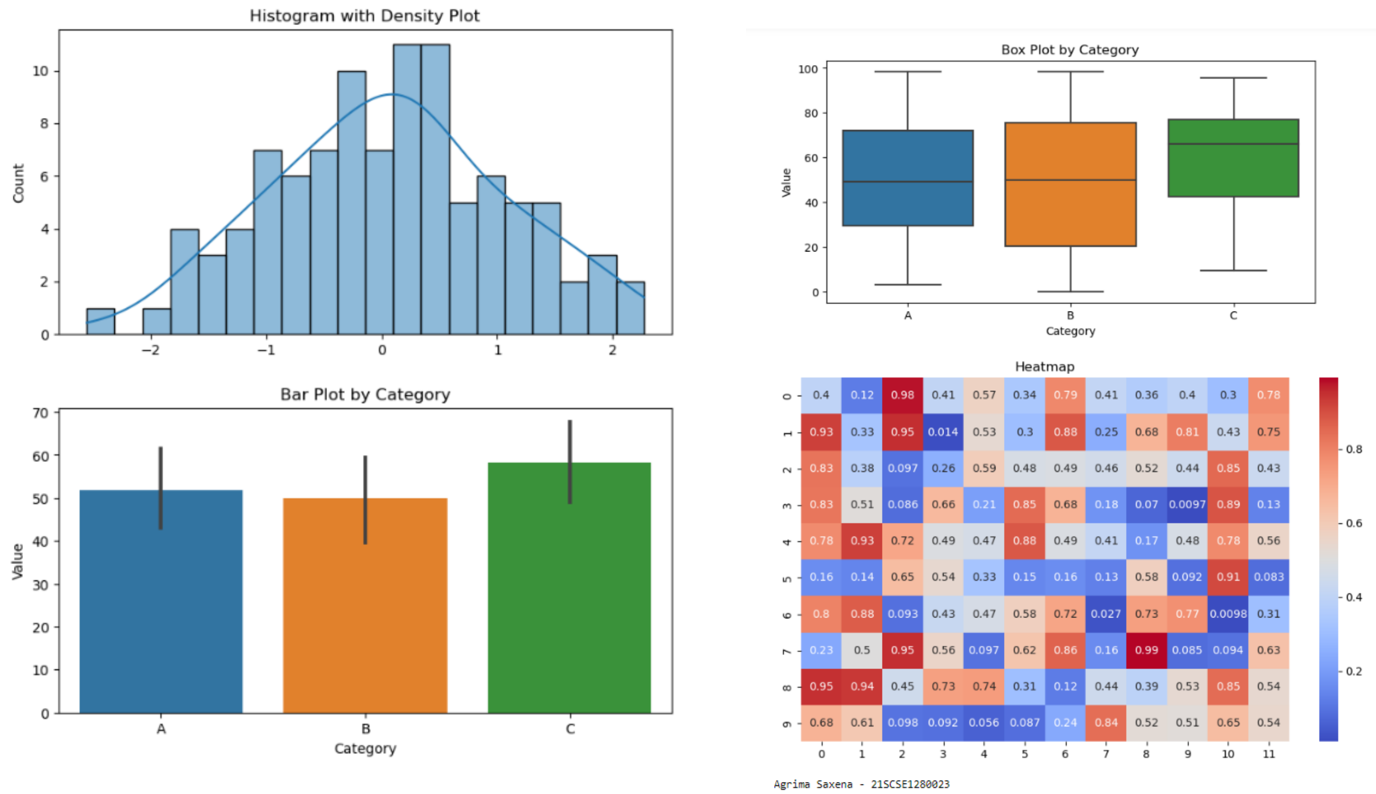
Aim

To use Seaborn, a high-level data visualization library, to create stylish and informative plots using python.

Program

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# Example data
np.random.seed(0)
data = np.random.randn(100)
df = pd.DataFrame({'Category': np.random.choice(['A', 'B', 'C'], size=100),
                   'Value': np.random.rand(100) * 100})
# Example 1: Histogram
plt.figure(figsize=(8, 4))
sns.histplot(data, bins=20, kde=True)
plt.title('Histogram with Density Plot')
plt.show()
# Example 2: Bar Plot
plt.figure(figsize=(8, 4))
sns.barplot(x='Category', y='Value', data=df)
plt.title('Bar Plot by Category')
plt.show()
# Example 3: Box Plot
plt.figure(figsize=(8, 4))
sns.boxplot(x='Category', y='Value', data=df)
plt.title('Box Plot by Category')
plt.show()
# Example 4: Heatmap
heatmap_data = np.random.rand(10, 12)
plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm')
plt.title('Heatmap')
plt.show()
print("Agrima Saxena - 21SCSE1280023")
```

Output



Result

Thus, the use Seaborn, a high-level data visualization library, to create stylish and informative plots using python has done and output was verified.

Ex. No.: 20	Calculate and visualize measures of central tendency and variability using R.
Date:	

Aim

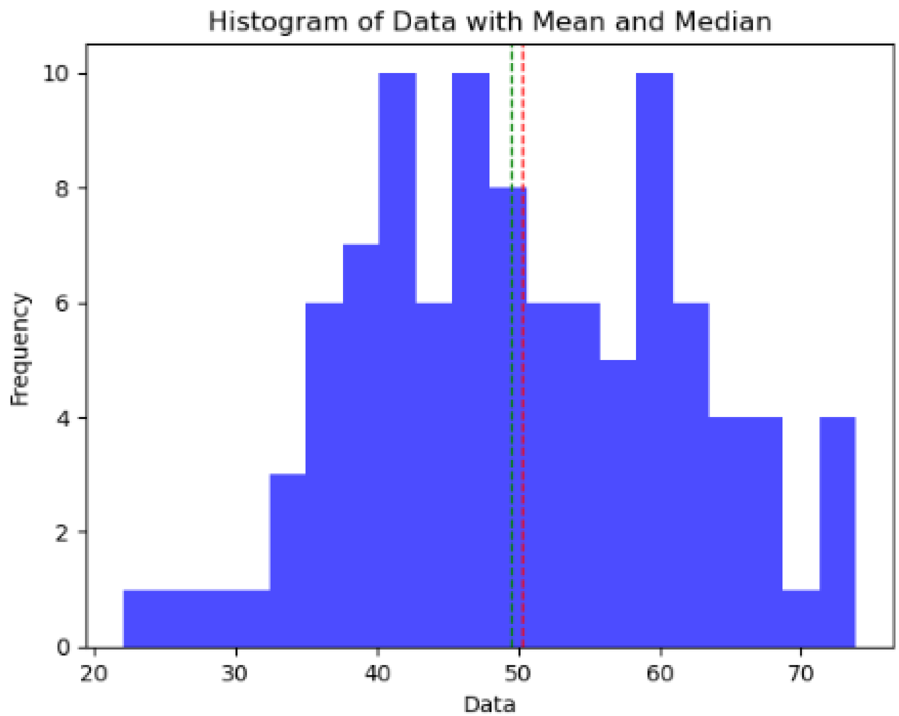
To calculate and visualize measures of central tendency and variability using R.

Program

```
install.packages("ggplot2")
# Sample Data
set.seed(123) # for reproducibility
data <- rnorm(100, mean = 50, sd = 10) # 100 random numbers, normal distribution
# Calculating central tendency
mean_data <- mean(data)
median_data <- median(data)
# Calculating variability
sd_data <- sd(data) # Standard deviation
var_data <- var(data) # Variance
# Printing the results
cat("Mean:", mean_data, "\n")
cat("Median:", median_data, "\n")
cat("Standard Deviation:", sd_data, "\n")
cat("Variance:", var_data, "\n")
library(ggplot2)
# Basic histogram with mean and median lines
ggplot(data = data.frame(data), aes(x = data)) +
  geom_histogram(binwidth = 5, fill = "blue", alpha = 0.7) +
  geom_vline(aes(xintercept = mean_data), color = "red", linetype = "dashed", size = 1) +
  geom_vline(aes(xintercept = median_data), color = "green", linetype = "dashed", size = 1) +
  labs(title = "Histogram of Data with Mean and Median",
       x = "Data",
       y = "Frequency") +
  theme_minimal()
car("Agrima Saxena – 21SCSE1280023")
```

Output

```
Mean: 50.2710907349036
Median: 49.46730268292144
Standard Deviation: 11.28240470477961
Variance: 127.2926559224331
```




```
Agrima Saxena - 21SCSE1280023
```

Result

Thus, the calculate and visualize measures of central tendency and variability using R has done and output was verified.

Ex. No.: 21	Scale and standardize variables to make them comparable using R.
Date:	

Aim

To scale and standardize variables to make them comparable using R.

Program

```
# Sample Data
set.seed(123) # for reproducibility
data <- data.frame(
  Variable1 = rnorm(100, mean = 50, sd = 10), # Normally distributed data
  Variable2 = runif(100, min = 0, max = 100)  # Uniformly distributed data
)
# Scaling and Standardizing
scaled_data <- scale(data)
# The 'scale' function by default centers and scales (standardizes) the data
# Viewing the first few rows of the scaled data
head(scaled_data)
# Mean should be approximately 0
sapply(scaled_data, mean)
# Standard Deviation should be 1
sapply(scaled_data, sd)
cat("Agrima Saxena – 21SCSE1280023")
```

Output

```
First 5 rows of scaled and standardized data:
 [[-0.98626109 -0.60960471]
 [ 0.8599553   0.13168825]
 [ 0.22678625  0.22250786]
 [-1.35911078  0.02471892]
 [-0.5368619  -1.77508232]]

Mean of scaled data (should be close to 0):
 [2.18214335e-15 2.02060590e-16]
Standard deviation of scaled data (should be 1):
 [1. 1.]
Agrima Saxena - 21SCSE1280023
```

Result

Thus, the scale and standardize variables to make them comparable using R has done and output was verified.

Ex. No.: 22	**Analyze income inequality using Gini coefficients using R.**
Date:	

Aim
To analyze income inequality using Gini coefficients using R.

Program
```
# Install and load the necessary package
if (!require(ineq)) install.packages("ineq")
library(ineq)

# Create a hypothetical dataset of incomes
income_data <- c(25000, 30000, 32000, 36000, 40000, 45000, 50000, 60000, 80000, 100000, 150000,
200000)

# Calculate the Gini coefficient
gini_coefficient <- ineq(income_data, type = "Gini")

# Print the Gini coefficient
print(paste("Gini Coefficient:", gini_coefficient))
cat("Agrima Saxena – 21SCSE1280023")
```

Output
```
Gini Coefficient: 0.3700864779874214
Agrima Saxena - 21SCSE1280023
```

Result
Thus, the income inequality using Gini coefficients using R has analyzed and output was verified.

Ex. No.: 23	**Apply smoothing techniques to time series data for trend analysis using R and python.**
Date:	

Aim

To apply smoothing techniques to time series data for trend analysis using R and python.

Program

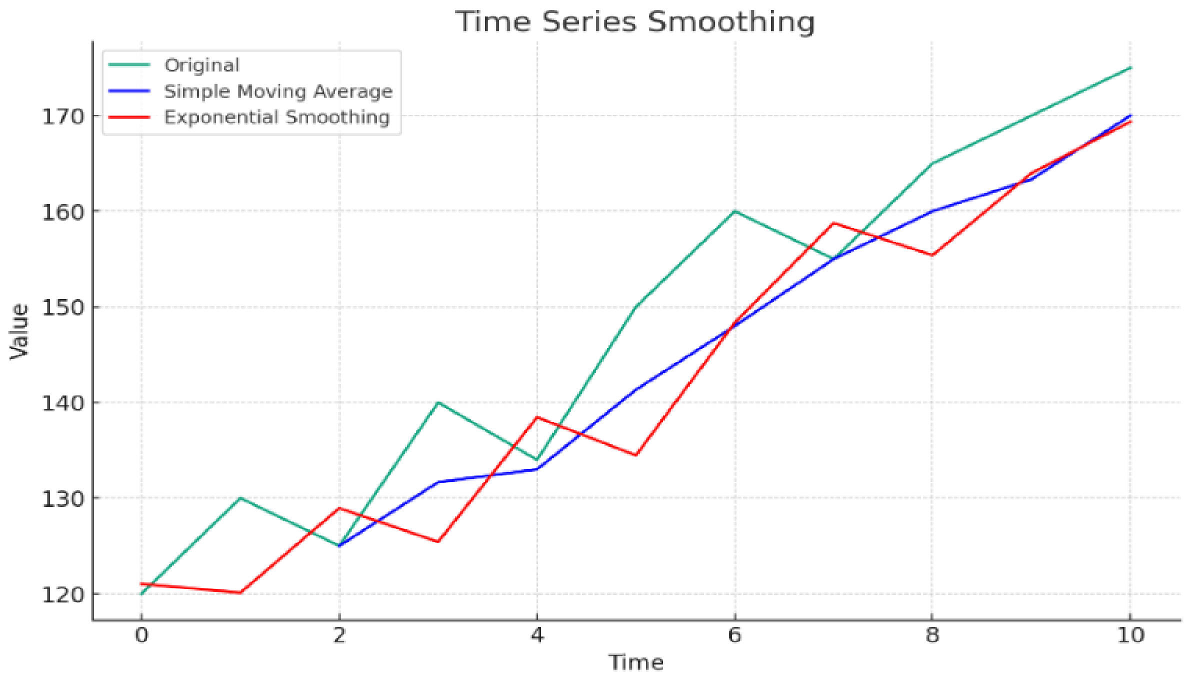
```
# Install and load necessary packages
if (!require(TTR)) install.packages("TTR")
if (!require(forecast)) install.packages("forecast")
library(TTR)
library(forecast)
# Hypothetical time series data
time_series_data <- ts(c(120, 130, 125, 140, 134, 150, 160, 155, 165, 170, 175), frequency = 12)

# Simple Moving Average
sma <- SMA(time_series_data, n = 3)

# Exponential Smoothing
exp_smoothing <- HoltWinters(time_series_data)

# Plotting
plot(time_series_data, main="Time Series Smoothing", xlab="Time", ylab="Value")
lines(sma, col="blue")
lines(exp_smoothing$fitted[,1], col="red")
cat("Agrima Saxena – 21SCSE1280023")
```

Output



Result

Thus, the smoothing techniques to time series data for trend analysis using R and python was applied and output was verified.

Ex. No.: 24	Create contingency tables and analyze percentages using R.
Date:	

Aim

To create contingency tables and analyze percentages using R.

Program

```
import numpy as np
import pandas as pd
np.random.seed(0)
example_data = pd.DataFrame({
 'Category1': np.random.choice(['Option1', 'Option2'], size=100),
 'Category2': np.random.choice(['OptionA', 'OptionB'], size=100)
})
# Creating the contingency table
contingency_table = pd.crosstab(example_data['Category1'],
example_data['Category2'], margins=True)
# Calculating the percentages
contingency_percentages = contingency_table.div(contingency_table.iloc[-1, :],
axis=1) * 100
print(contingency_table, contingency_percentages)
```

Output

```
Category2  OptionA  OptionB  All
Category1
Option1         24       20   44
Option2         31       25   56
All             55       45  100 Category2     OptionA     OptionB    All
Category1
Option1     43.636364   44.444444   44.0
Option2     56.363636   55.555556   56.0
All        100.000000  100.000000  100.0
```

Result

Thus, the contingency tables and analyze percentages using R has created and output was verified.

Ex. No.: 25	Perform chi-squared tests and visual analysis of contingency tables using R.
Date:	

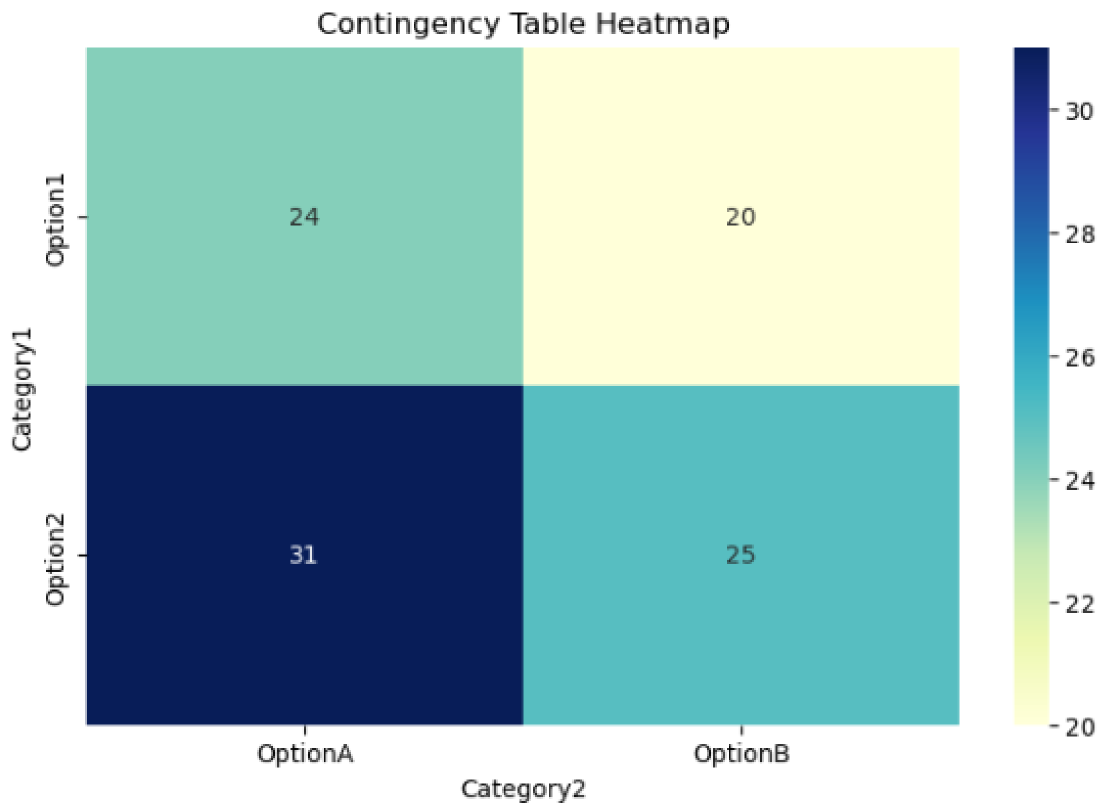
Aim

To perform chi-squared tests and visual analysis of contingency tables using R.

Program

```
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
# Example data for the contingency table
np.random.seed(0)
df = pd.DataFrame({
 'Category1': np.random.choice(['Option1', 'Option2'], size=100),
 'Category2': np.random.choice(['OptionA', 'OptionB'], size=100)
})
# Creating the contingency table
contingency_table = pd.crosstab(df['Category1'], df['Category2'], margins=True)
 #Performing the Chi-squared test
chi2, p_value, dof, expected = stats.chi2_contingency(contingency_table.iloc[:-1, :-1])
# Visualizing the contingency table with a heatmap
plt.figure(figsize=(8, 5))
sns.heatmap(contingency_table.iloc[:-1, :-1], annot=True, cmap="YlGnBu", fmt='g')
plt.title('Contingency Table Heatmap')
plt.ylabel('Category1')
plt.xlabel('Category2')
plt.show()
print(chi2, p_value, dof, expected)
```

Output




```
0.0 1.0 1 [[24.2 19.8]
 [30.8 25.2]]
```

Result

Thus, the chi-squared tests and visual analysis of contingency tables using R was performed and output was verified.

Ex. No.: 26	**Explore causality between variables using regression analysis using R.**
Date:	

Aim

To explore causality between variables using regression analysis using R.

Program

```
import statsmodels.api as sm
# Using the same example data
# Adding a numerical variable for regression analysis
df['Numerical'] = np.random.normal(0, 1, 100)
# Preparing the data for regression analysis
X = df['Numerical'] # Independent variable
y = df['Category1'].apply(lambda x: 1 if x == 'Option1' else 0) # Dependent variable
(binary)
# Adding a constant to the model (intercept)
X = sm.add_constant(X)
# Performing the regression analysis
model = sm.OLS(y, X).fit()
# Summary of the regression model
print(model.summary())
```

Output

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Category1   R-squared:                       0.010
Model:                            OLS   Adj. R-squared:                 -0.000
Method:                 Least Squares   F-statistic:                    0.9940
Date:                Tue, 16 Jan 2024   Prob (F-statistic):              0.321
Time:                        14:51:14   Log-Likelihood:                -71.349
No. Observations:                 100   AIC:                             146.7
Df Residuals:                      98   BIC:                             151.9
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.4477      0.050      8.868      0.000       0.347       0.548
Numerical      0.0507      0.051      0.997      0.321      -0.050       0.152
==============================================================================
Omnibus:                      906.187   Durbin-Watson:                   2.166
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               16.032
Skew:                           0.239   Prob(JB):                     0.000330
Kurtosis:                       1.097   Cond. No.                         1.17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

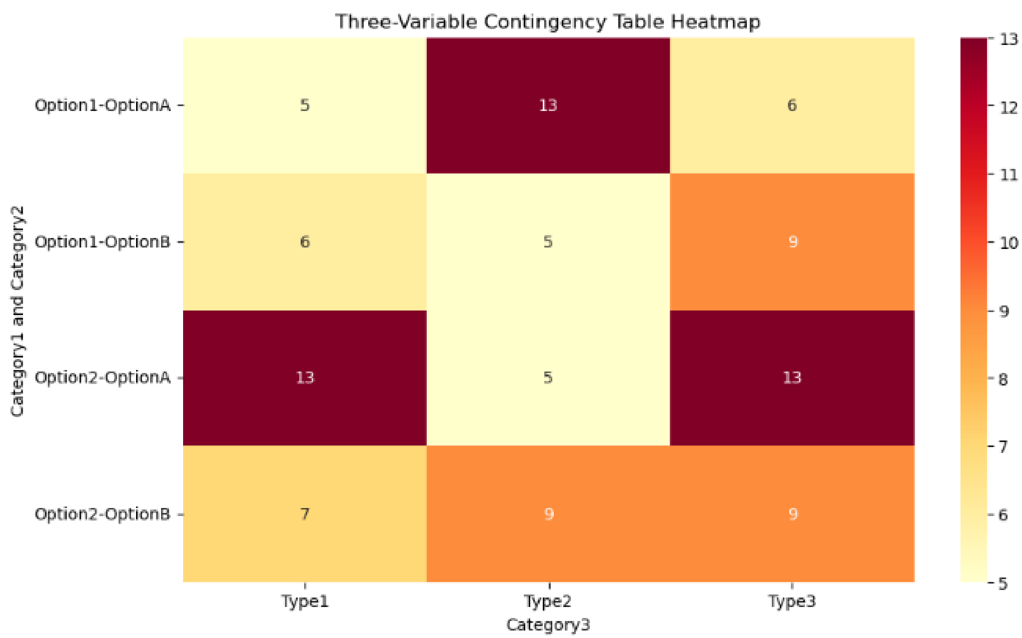
Result

Thus, the causality between variables using regression analysis using R has explored and output was verified.

Ex. No.: 27	Extend contingency tables to three-variable analysis using R.
Date:	

Aim
To extend contingency tables to three-variable analysis using R.

Program
```
df['Category3'] = np.random.choice(['Type1', 'Type2', 'Type3'], size=100)
# Creating a three-variable contingency table
three_var_contingency = pd.crosstab(index=[df['Category1'], df['Category2']],
columns=df['Category3'])
# Visualizing the three-variable contingency table
plt.figure(figsize=(10, 6))
sns.heatmap(three_var_contingency, annot=True, cmap="YlOrRd", fmt='g')
plt.title('Three-Variable Contingency Table Heatmap')
plt.ylabel('Category1 and Category2')
plt.xlabel('Category3')
plt.show()
print(three_var_contingency)
```

Output



```
Category3            Type1  Type2  Type3
Category1 Category2
Option1   OptionA        5     13      6
          OptionB        6      5      9
Option2   OptionA       13      5     13
          OptionB        7      9      9
```

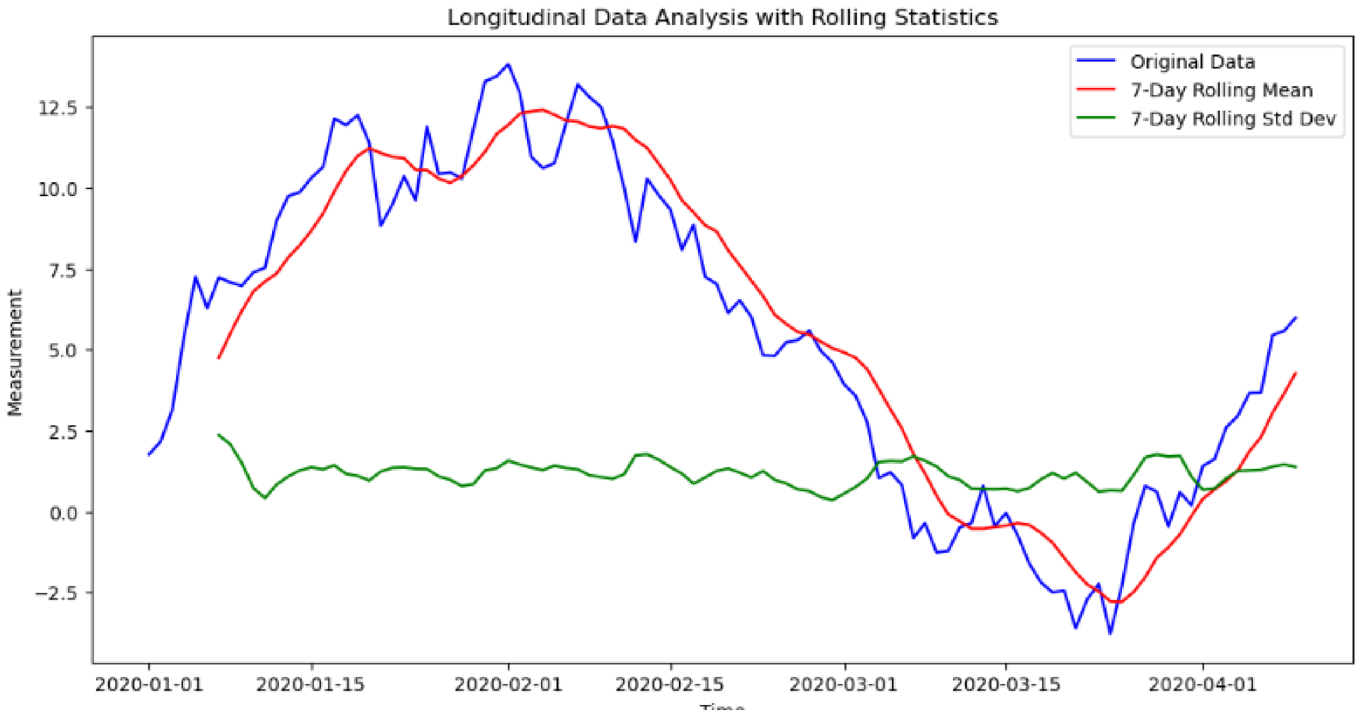
Result
Thus, the contingency tables to three-variable analysis using R have been extended and output was verified.

Ex. No.: 28	Analyze longitudinal data, growth curves, and trends using R.
Date:	

Aim
To analyze longitudinal data, growth curves, and trends using R.

Program
```
np.random.seed(0)
time_series_data = pd.DataFrame({
 'Time': pd.date_range(start='2020-01-01', periods=100, freq='D'),
 'Measurement': np.random.normal(loc=0, scale=1, size=100).cumsum()
})
# Setting 'Time' as the index
time_series_data.set_index('Time', inplace=True)
# Calculating rolling statistics for growth curves and trends
rolling_mean = time_series_data['Measurement'].rolling(window=7).mean()
rolling_std = time_series_data['Measurement'].rolling(window=7).std()
# Plotting the data
plt.figure(figsize=(12, 6))
plt.plot(time_series_data['Measurement'], label='Original Data', color='blue')
plt.plot(rolling_mean, label='7-Day Rolling Mean', color='red')
plt.plot(rolling_std, label='7-Day Rolling Std Dev', color='green')
plt.title('Longitudinal Data Analysis with Rolling Statistics')
plt.xlabel('Time')
plt.ylabel('Measurement')
plt.legend()
plt.show()
```

Output



Result

Thus, the longitudinal data, growth curves, and trends using R has analyzed and output was verified.

Ex. No.: 29	Perform data cleaning and preprocessing for time series data using R.
Date:	

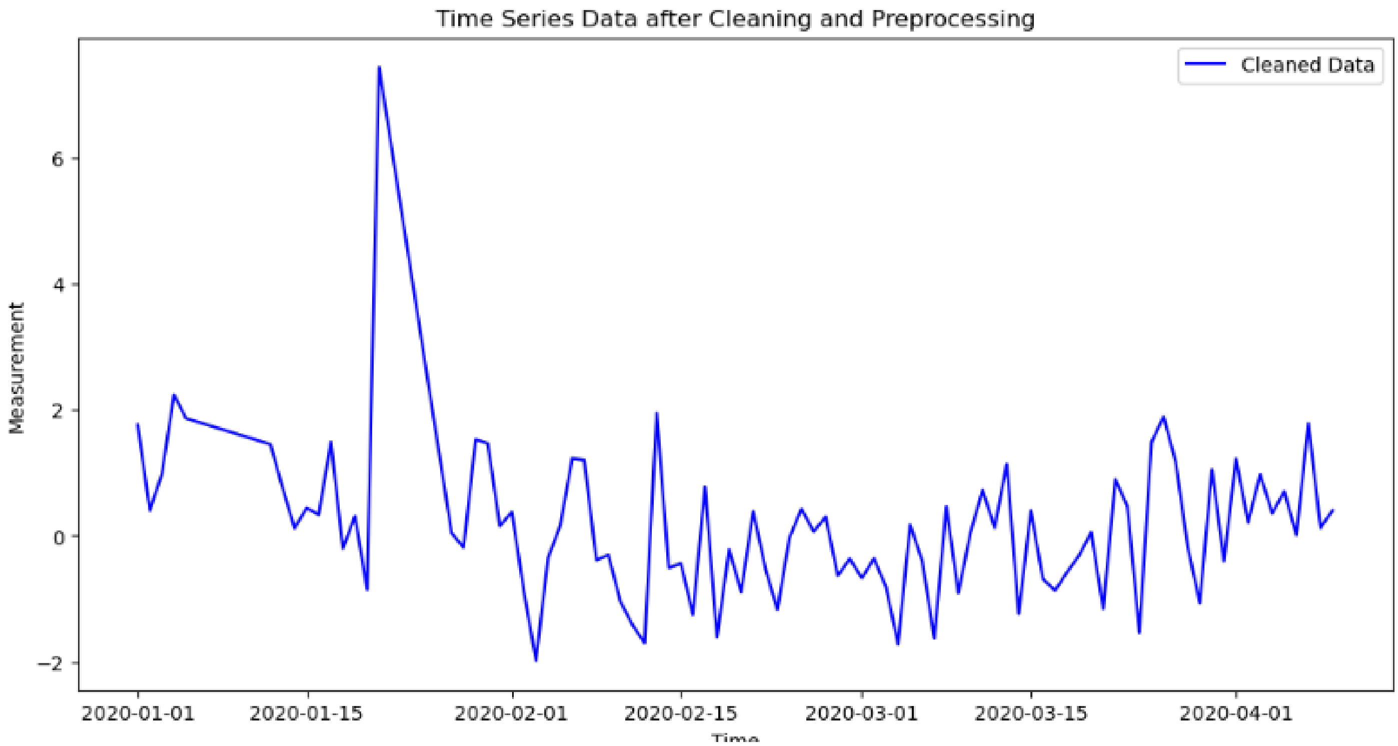
Aim

To perform data cleaning and preprocessing for time series data using R.

Program

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
# Generating example time series data with missing values and outliers
np.random.seed(0)
time_series_data = pd.DataFrame({
 'Time': pd.date_range(start='2020-01-01', periods=100, freq='D'),
 'Measurement': np.random.normal(loc=0, scale=1, size=100)
})
# Introducing missing values
time_series_data.loc[5:10, 'Measurement'] = np.nan
# Introducing outliers
time_series_data.loc[20:25, 'Measurement'] = time_series_data.loc[20:25,
'Measurement'] + 10
# Data cleaning: Handling missing values by interpolation
time_series_data['Measurement'] = time_series_data['Measurement'].interpolate()
# Data preprocessing: Removing outliers using Z-score
time_series_data['Z_Score'] = stats.zscore(time_series_data['Measurement'])
time_series_data = time_series_data[time_series_data['Z_Score'].abs() < 3]
# Dropping the Z-score column as it's no longer needed
time_series_data.drop('Z_Score', axis=1, inplace=True)
# Setting 'Time' as the index
time_series_data.set_index('Time', inplace=True)
# Visualizing the cleaned and preprocessed data
plt.figure(figsize=(12, 6))
plt.plot(time_series_data['Measurement'], label='Cleaned Data', color='blue')
plt.title('Time Series Data after Cleaning and Preprocessing')
plt.xlabel('Time')
plt.ylabel('Measurement')
plt.legend()
plt.show()
```

Output



Result

Thus, the data cleaning and preprocessing for time series data using R has performed and output was verified.

Ex. No.: 30	**Visualizing Time Series: Create time series plots and explore patterns in data using R.**
Date:	

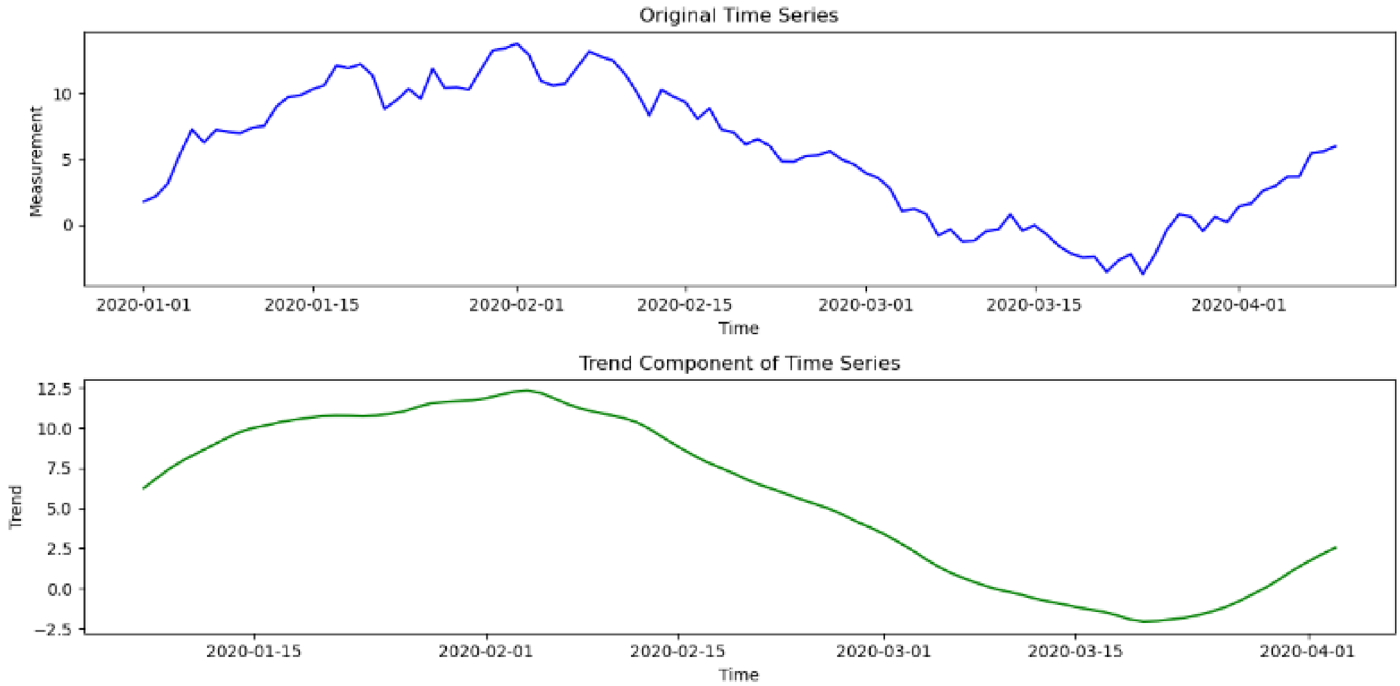
Aim

To visualizing Time Series: Create time series plots and explore patterns in data using R.

Program

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
np.random.seed(0)
time_series_data = pd.DataFrame({
 'Time': pd.date_range(start='2020-01-01', periods=100, freq='D'),
 'Measurement': np.random.normal(loc=0, scale=1, size=100).cumsum()
})
# Setting 'Time' as the index
time_series_data.set_index('Time', inplace=True)
# Creating time series plots
plt.figure(figsize=(12, 6))
# Original time series plot
plt.subplot(2, 1, 1)
plt.plot(time_series_data['Measurement'], label='Original Data', color='blue')
plt.title('Original Time Series')
plt.xlabel('Time')
plt.ylabel('Measurement')
# Seasonal Decomposition to explore patterns
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(time_series_data['Measurement'],
model='additive', period=12)
# Trend component
plt.subplot(2, 1, 2)
plt.plot(decomposition.trend, label='Trend', color='green')
plt.title('Trend Component of Time Series')
plt.xlabel('Time')
plt.ylabel('Trend')
plt.tight_layout()
plt.show()
```

Output



Result

Thus, the visualizing Time Series: Create time series plots and explore patterns in data using R has done and output was verified.