SRM MADURAI
COLLEGE FOR ENGINEERING AND TECHNOLOGY
Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# LABORATORY MANUAL

| | |
|---|---|
| **Sub.Code** | **:** CS3361 |
| **Sub.Name** | : DATA SCIENCE LABORATORY |
| **Regulation** | : R2021 |

**Prepared By,**
**Ms.G.Aninthitha/Mrs.M.Kanimozhi**
**AP/CSE**

**Approved By,**
**Dr.C.Callins Christiyana**
**Prof& Head /CSE**

| INSTITUTE MISSION |
|---|
| To become a centre of excellence in preparing engineering with excellent technical, scientific research and entrepreneurial abilities to contribute to the society. |

| INSTITUTE MISSION | |
|---|---|
| 1 | Providing comprehensive learning environment |
| 2 | Imparting state-of-the-art technology to fulfil the needs of the students and Industry |
| 3 | Establishing Industry-Institute alliance for bilateral benefits |
| 4 | Promoting Research and Development activities |
| 5 | Offering student lead activities to inculcate ethics, social responsibilities, entrepreneurial, and leadership skills |

| DEPARTMENT VISION |
|---|
| To become a centre of excellence in technical education and scientific research in the field of Computer Science and Engineering for the wellbeing of the society. |

| DEPARTMENT MISSION | |
|---|---|
| 1 | Producing graduates with a strong theoretical and practical in computer technology to meet the Industry expectation. |
| 2 | Offering holistic learning ambience for faculty and students to investigate, apply and transfer knowledge. |
| 3 | Inculcating interpersonal traits among the students leading to employability and entrepreneurship. |
| 4 | Establishing effective linkage with the Industries for the mutual benefits |
| 5 | Strengthening Research activities to solve the problems related to industry and society. |

## SYLLABUS

| COURSE CODE | COURSE NAME | L | T | P | C |
|---|---|---|---|---|---|
| **CS3361** | **DATA SCIENCE LABORATORY** | **0** | **0** | **4** | **2** |

**COURSE OBJECTIVES :**

- To understand the python libraries for data science
- To understand the basic Statistical and Probability measures for data science.
- To learn descriptive analytics on the benchmark data sets.
- To apply correlation and regression analytics on standard data sets.
- To present and interpret data using visualization packages in Python.

**EXPERIMENTS**

1. Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages.
2. Working with Numpy arrays
3. Working with Pandas data frames
4. Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.
5. Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following:
   a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.
   b. Bivariate analysis: Linear and logistic regression modeling
   c. Multiple Regression analysis
   d. Also compare the results of the above analysis for the two data sets.
6. Apply and explore various plotting functions on UCI data sets.
   a. Normal curves
   b. Density and contour plots
   c. Correlation and scatter plots
   d. Histograms
   e. Three dimensional plotting
7. Visualizing Geographic Data with Basemap

**TOTAL: 60 Periods**

**CONTENT BEYOND SYLLABI: Hadoop, Apache spark**

**COURSE OUTCOMES:**

On completion of the course, students will be able to:

**CO1:** Make use of the python libraries for data science

**CO2:** Make use of the basic Statistical and Probability measures for data science.

**CO3:** Perform descriptive analytics on the benchmark data sets.

**CO4:** Perform correlation and regression analytics on standard data sets

**CO5:** Present and interpret data using visualization packages in Python.

**EQUIPMENT / SOFTWARE AND HARDWARE REQUIREMENT**

- INTEL based desktop PC with min. 8GB RAM and 500 GB HDD, 17" or higher TFT Monitor, Keyboard and mouse
- Windows 10 or higher operating system / Linux Ubuntu 20 or higher
- Python3.9 and above, Python, Numpy, Scipy, Matplotlib, Pandas, seaborn, Pycharm

## List of Experiments

| EX.NO:1 | **Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages.** |
|---|---|

## AIM

To Download and install python and its packages using pip installation

## PROCEDURE

**Install Python Data Science Packages**

Python is a high-level and general-purpose programming language with data science and machine learning packages. Use the video below to install on Windows, MacOS, or Linux. As a first step, install Python for Windows, MacOS, or Linux.

**Python Packages**

The power of Python is in the packages that are available either through the pip or conda package managers. This page is an overview of some of the best packages for machine learning and data science and how to install them.

We will explore the Python packages that are commonly used for data science and machine learning. You may need to install the packages from the terminal, Anaconda prompt, command prompt, or from the Jupyter Notebook. If you have multiple versions of Python or have specific dependencies then use an environment manager such as pyenv. For most users, a single installation is typically sufficient. The Python package manager pip has all of the packages (such as gekko) that we need for this course. If there is an administrative access error, install to the local profile with the --user flag.

**pip install gecko**

**Gekko**

Gekko provides an interface to gradient-based solvers for machine learning and optimization of mixed-integer, differential algebraic equations, and time series models.

Gekko provides exact first and second derivatives through automatic differentiation and discretization with simultaneous or sequential methods.

**pip install gecko**

**Keras**

Keras provides an interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Other backend packages were supported until version 2.4. TensorFlow is now the only backend and is installed separately with pip install tensorflow.

**pip install keras**

**Matplotlib**

The package matplotlib generates plots in Python.

**pip install matplotlib**

**Numpy**

Numpy is a numerical computing package for mathematics, science, and engineering. Many data science packages use Numpy as a dependency.

**pip install numpy**

**OpenCV**

OpenCV (Open Source Computer Vision Library) is a package for real-time computer vision and developed with support from Intel Research.

**pip install opencv-python**

**Pandas**

Pandas visualizes and manipulates data tables. There are many functions that allow efficient manipulation for the preliminary steps of data analysis problems.

**pip install pandas**

**Plotly**

Plotly renders interactive plots with HTML and JavaScript. Plotly Express is included with Plotly.

**pip install plotly**

**PyTorc**

PyTorch enables deep learning, computer vision, and natural language processing.Development is led by Facebook's AI Research lab (FAIR).

**pip install torch**

**Scikit-Learn**

Scikit-Learn (or sklearn) includes a wide variety of classification, regression and clustering algorithms including neural network, support vector machine, random forest, gradient boosting, k-means clustering, and other supervised or unsupervised learning methods.

**pip install scikit-learn**

**SciPy**

SciPy is a general-purpose package for mathematics, science, and engineering and extends the base capabilities of NumPy.

**pip install scipy**

**Seaborn**

Seaborn is built on matplotlib, and produces detailed plots in few lines of code.

**pip install seaborn**

**Statsmodels**

Statsmodels is a package for exploring data, estimating statistical models, and performing statistical tests. It include descriptive statistics, statistical tests, plotting functions, and result statistics.

**pip install statsmodels**

**TensorFlow**

TensorFlow is an open source machine learning platform with particular focus on training and inference of deep neural networks. Development is led by the Google Brain team.

**pip install tensorflow**

**Augmented Questions :**

1. How would you approach optimizing a Python program for performance? Discuss techniques for profiling, identifying bottlenecks, and improving efficiency. Provide examples of how you might apply these techniques to a data processing task.
2. In the context of data analysis, what are some best practices for ensuring data quality and integrity? Explain how you would handle missing data, outliers, and data inconsistencies in a dataset before performing any analysis.

**Viva Questions:**

1. How do you install NumPy, SciPy, Jupyter, Statsmodels, and Pandas in a Python environment?
2. Can you explain the primary functionalities of NumPy and how it is useful in scientific computing?
3. What are some common functions and features provided by SciPy, and how does it extend NumPy's capabilities?
4. What is Jupyter Notebook, and how does it facilitate interactive computing and data analysis?
5. Describe how Pandas and Statsmodels are used for data analysis and statistical modeling in Python.

**RESULT:**

Thus the download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages was successfully completed.

| EX.NO:2 | Working with Numpy arrays |
|---|---|

**AIM**

The aim is to create a NumPy array with a specified number of dimensions using the argument and verify the resulting number of dimensions.

**ALGORITHM:**

1. Import the NumPy library.
2. Create an array with a specified set of elements.
3. Use the **ndmin** argument to set the desired number of dimensions for the array.
4. Print the created array.
5. Print the number of dimensions of the array using the **ndim** attribute.

**CREATE A NUMPY NDARRAY OBJECT**

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

**Example**

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

To create an ndarray, we can pass a list, tuple or any array-like object into the array()

method, and it will be converted into anndarray:

**Example**

**Use a tuple to** create a NumPy array:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

**Dimensions in Arrays**

A dimension in arrays is one level of array depth (nested arrays).

**0-D Arrays**

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

**Example**

Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)
```

```
print(arr)
```

**1-D Arrays**

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

**Example**

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

**2-D Arrays**

An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix or 2nd order tensors.

**Example**

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

**3-D arrays**

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

**Example**

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

**Check Number of Dimensions?**

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

**Example**

Check how many dimensions the arrays have:

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

**Higher Dimensional Arrays**

An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the ndmin argument.

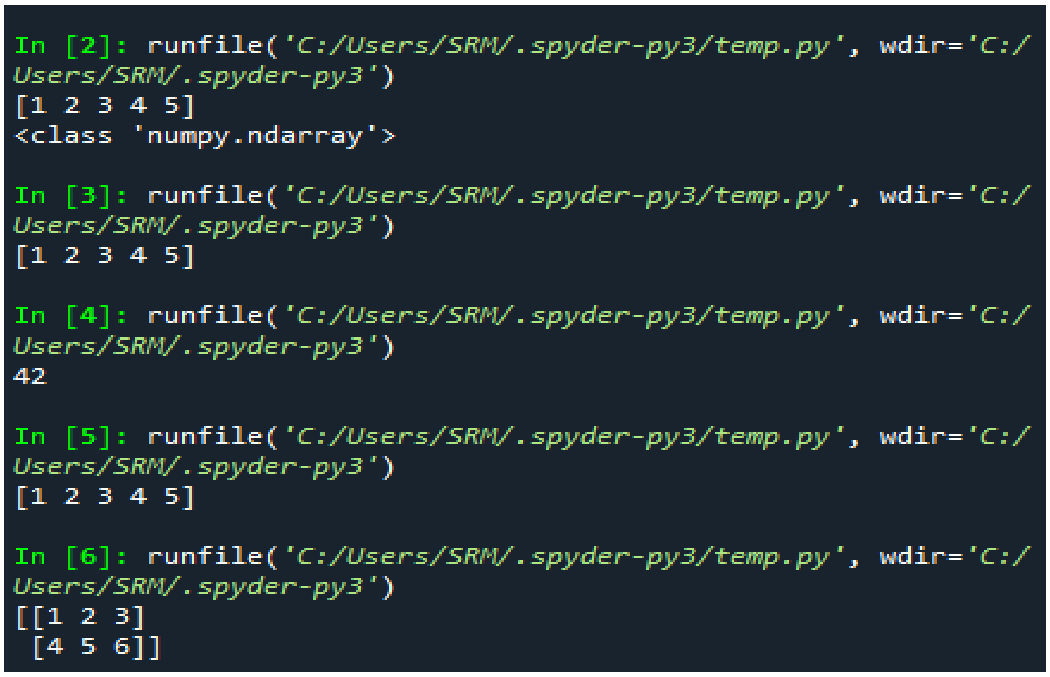
**Example**

Create an array with 5 dimensions and verify that it has 5 dimensions:

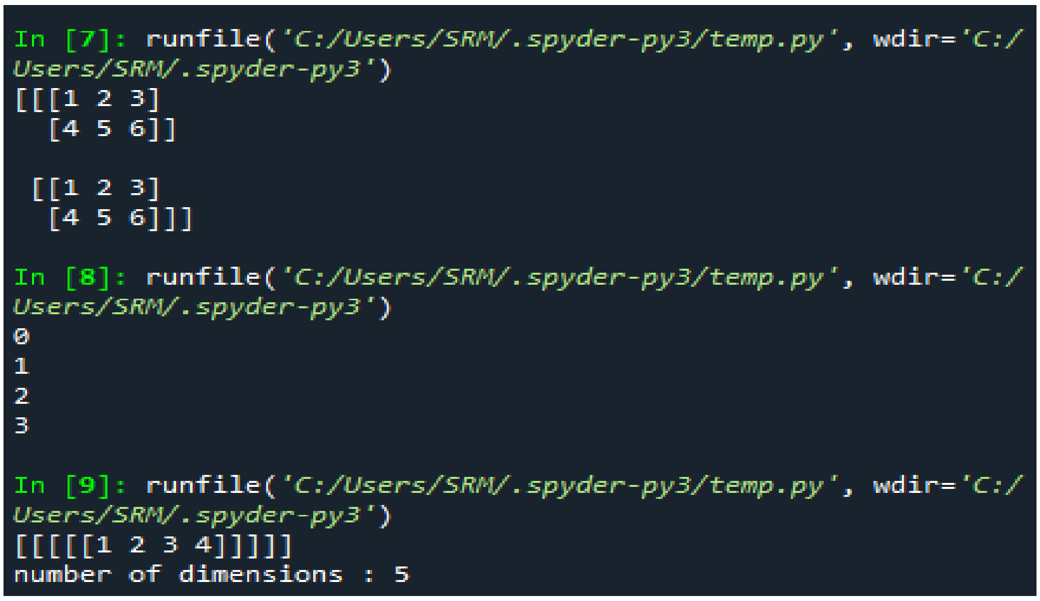
```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

**OUTPUT:**



```
In [2]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
[1 2 3 4 5]
<class 'numpy.ndarray'>

In [3]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
[1 2 3 4 5]

In [4]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
42

In [5]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
[1 2 3 4 5]

In [6]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
[[1 2 3]
 [4 5 6]]
```



```
In [7]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]

In [8]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
0
1
2
3

In [9]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

**Augmented Questions :**

1. **Write a Python program using NumPy to create a 3D array of shape `(4, 3, 2)` with random integers between 0 and 10. Perform the following tasks:**

- Compute the sum along the first axis (axis=0).
- Compute the mean along the second axis (axis=1).
- Flatten the 3D array into a 1D array and find the maximum value

**2. Write a Python program using NumPy to perform the following tasks with two 1D arrays:**

- Create two 1D arrays of length 10 with random integers between 1 and 20.
- Compute and print their dot product.
- Compute the element-wise product of the two arrays.
- Normalize the element-wise product by dividing it by the maximum value of the product

**Viva Questions:**

2. What is the difference between a NumPy array and a Python list, and why would you use a NumPy array for numerical computations?
3. How can you create a NumPy array from a Python list or tuple? Provide an example.
4. Describe the various methods to access and manipulate elements in a NumPy array. How can you perform slicing and indexing on a 2D array?
5. How do you perform basic arithmetic operations (such as addition, subtraction, multiplication, and division) on NumPy arrays? What are some benefits of using NumPy's vectorized operations over traditional loops?
6. Explain how broadcasting works in NumPy and give an example of how it can be used to perform operations on arrays of different shapes.

**RESULT**

Thus the working of Numpy arrays was executed successfully.

| EX.NO:3 | Working with Pandas data frames |
|---|---|

**AIM:**

The aim is to illustrate the basic operations of creating, indexing, and loading data into a Pandas DataFrame. This includes creating a simple DataFrame, locating specific rows using index labels, adding named indexes, and loading data from an external CSV file into a DataFrame.

**ALGORITHM:**

1. Import the Pandas library.
2. Create a simple DataFrame using a Python dictionary.
3. Print the DataFrame to display its structure.
4. Use the **loc** attribute to locate and print specific rows based on their index.
5. Add named indexes to the DataFrame using the **index** argument.
6. Print the DataFrame with named indexes.
7. Use the **loc** attribute with named indexes to locate and print specific rows.
8. Import the Pandas library again for the file loading example.
9. Use the **read_csv** function to load data from a CSV file into a DataFrame.
10. Print the resulting DataFrame.

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

**Example**

**Create a simple Pandas DataFrame**:

```
import pandas as pd

data = {

"calories": [420, 380, 390],

"duration": [50, 40, 45]

}

#load data into a DataFrame object:

df = pd.DataFrame(data)

print(df)
```

**Locate Row**

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

**Example**

**to print row 0 alone**

```
#refer to the row index:
print(df.loc[0])
```

**Example**

**Return row 0 and 1:**

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

**Named Indexes**

With the index argument, you can name your own indexes.

**Example**

**Add a list of names to give each row a name:**

```
import pandas as pd
data = {
"calories": [420, 380, 390],
"duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

**Locate Named Indexes**

Use the named index in the loc attribute to return the specified row(s).

**Example**

Return "day2":

```
#refer to the named index:
print(df.loc["day2"])
```

**Load Files Into a DataFrame**

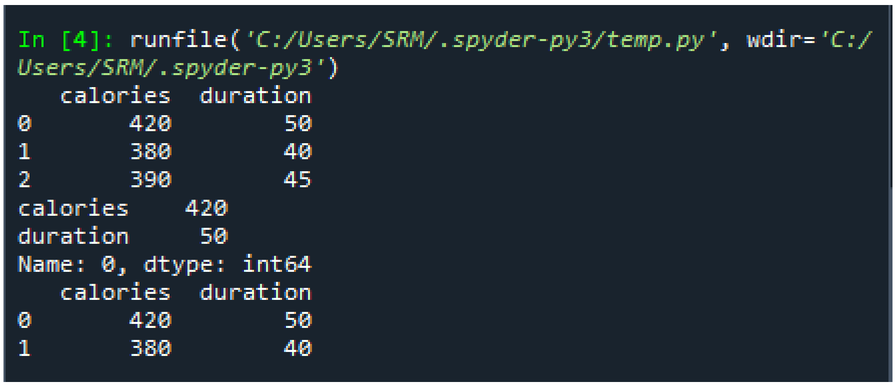
If your data sets are stored in a file, Pandas can load them into a DataFrame.

**Example**

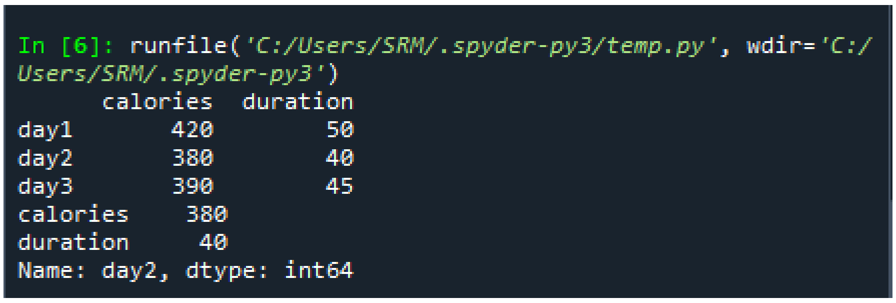
**Load a comma separated file (CSV file) into a DataFrame:**

```
import pandas as pd
file_path = r'C:\Users\SRM\Downloads\iris.csv'
df = pd.read_csv(file_path)
print(df)
```

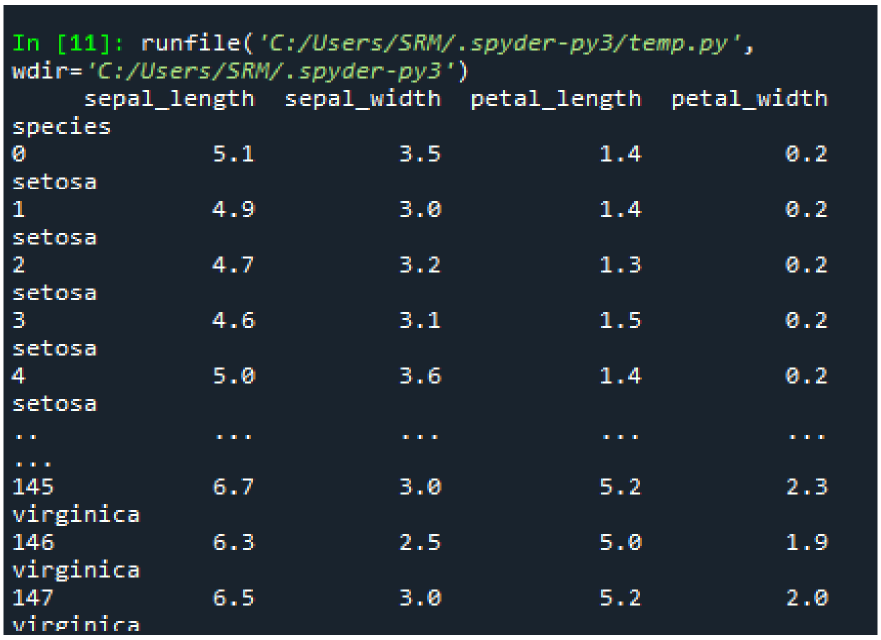
**OUTPUT:**



```
In [4]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
   calories  duration
0       420        50
1       380        40
2       390        45
calories    420
duration     50
Name: 0, dtype: int64
   calories  duration
0       420        50
1       380        40
```



```
In [6]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
      calories  duration
day1       420        50
day2       380        40
day3       390        45
calories    380
duration     40
Name: day2, dtype: int64
```



```
In [11]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
     sepal_length  sepal_width  petal_length  petal_width
species
0             5.1          3.5           1.4          0.2
setosa
1             4.9          3.0           1.4          0.2
setosa
2             4.7          3.2           1.3          0.2
setosa
3             4.6          3.1           1.5          0.2
setosa
4             5.0          3.6           1.4          0.2
setosa
..            ...          ...           ...          ...
...
145           6.7          3.0           5.2          2.3
virginica
146           6.3          2.5           5.0          1.9
virginica
147           6.5          3.0           5.2          2.0
virginica
```

**AUGMENTED QUESTIONS:**

1. Write a Python program using Pandas to analyze a dataset of customer transactions. The dataset includes columns for 'CustomerID', 'TransactionDate', 'Amount', and 'Category'. Perform the following tasks:

- Load the dataset from a CSV file.
- Convert the 'TransactionDate' column to a datetime format and set it as the index of the DataFrame.
- Filter the DataFrame to include only transactions that occurred in the last 30 days and sort them by 'Amount' in descending order.
- Export the filtered DataFrame to a new CSV file.

2. Create a Python program using Pandas to work with a dataset of employee records. The dataset contains columns 'EmployeeID', 'Name', 'Department', 'JoiningDate', and 'Salary'. Perform the following operations:

- Load the dataset and inspect its basic structure.
- Calculate the number of employees and the average salary for each department.
- Add a new column 'YearsWithCompany' to the DataFrame, representing the number of years each employee has been with the company.
- Identify and display employees who have been with the company for more than 5 years and have a salary above the median salary of their department.
- Save the results to an Excel file with separate sheets for each department.

**VIVA QUESTIONS:**

1. How can you create a Pandas DataFrame from a dictionary, and what are the key methods to inspect the structure and content of the DataFrame? Provide an example.
2. Describe how you would handle missing data in a Pandas DataFrame. What methods are available for detecting, removing, or imputing missing values?
3. How can you perform data filtering and selection in a Pandas DataFrame? Explain how to select rows based on certain conditions and how to access specific columns.
4. What are some common operations for data aggregation and grouping in Pandas? How would you use the `groupby` method to calculate summary statistics for different groups within a DataFrame?
5. Explain how to merge and join DataFrames in Pandas. What are the different types of joins available, and how do you handle conflicts and overlapping column names during the merge process?

**RESULT**

Thus the working with pandas data frames was executed successfully.

| EX.NO:4 | **Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.** |
|---|---|

**AIM**

Perform descriptive analytics on the Iris dataset using Pandas and Seaborn, including data reading, exploration, manipulation, summary statistics, visualization, and handling missing values.

**ALGORITHM**

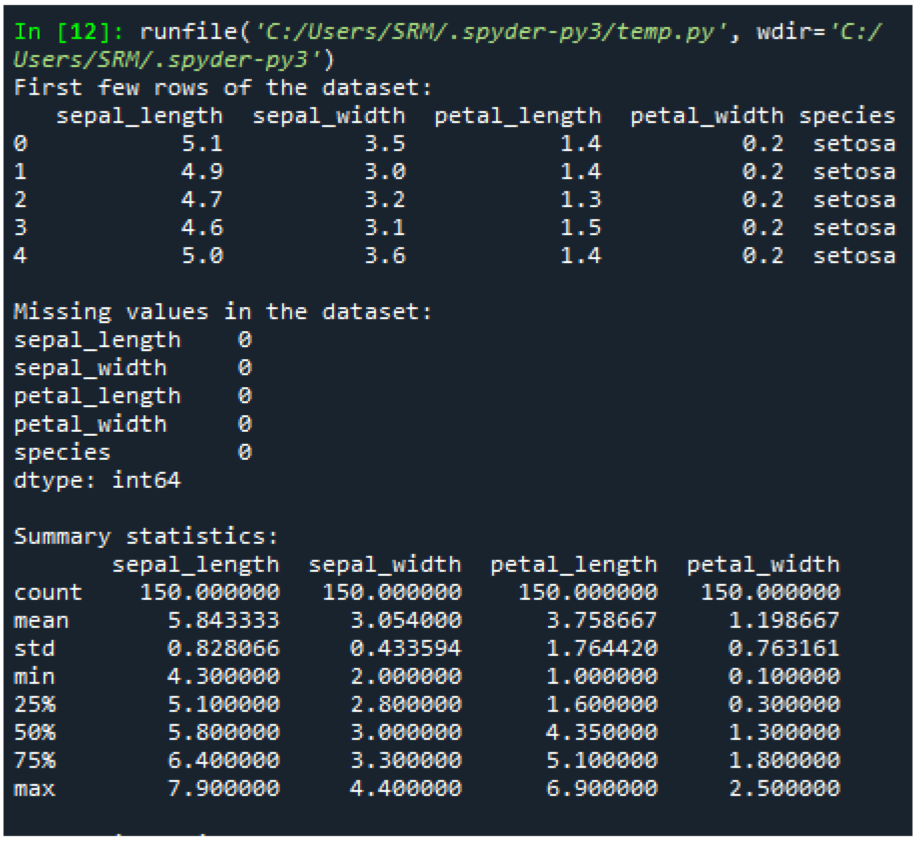
1. Read the Iris dataset from a CSV file using Pandas.
2. Explore the dataset's structure and content, perform data slicing, and select specific columns.
3. Calculate summary statistics, handle missing values, and manipulate the data.
4. Apply styling and visualization techniques using Seaborn for better insights.

**PROGRAM:**

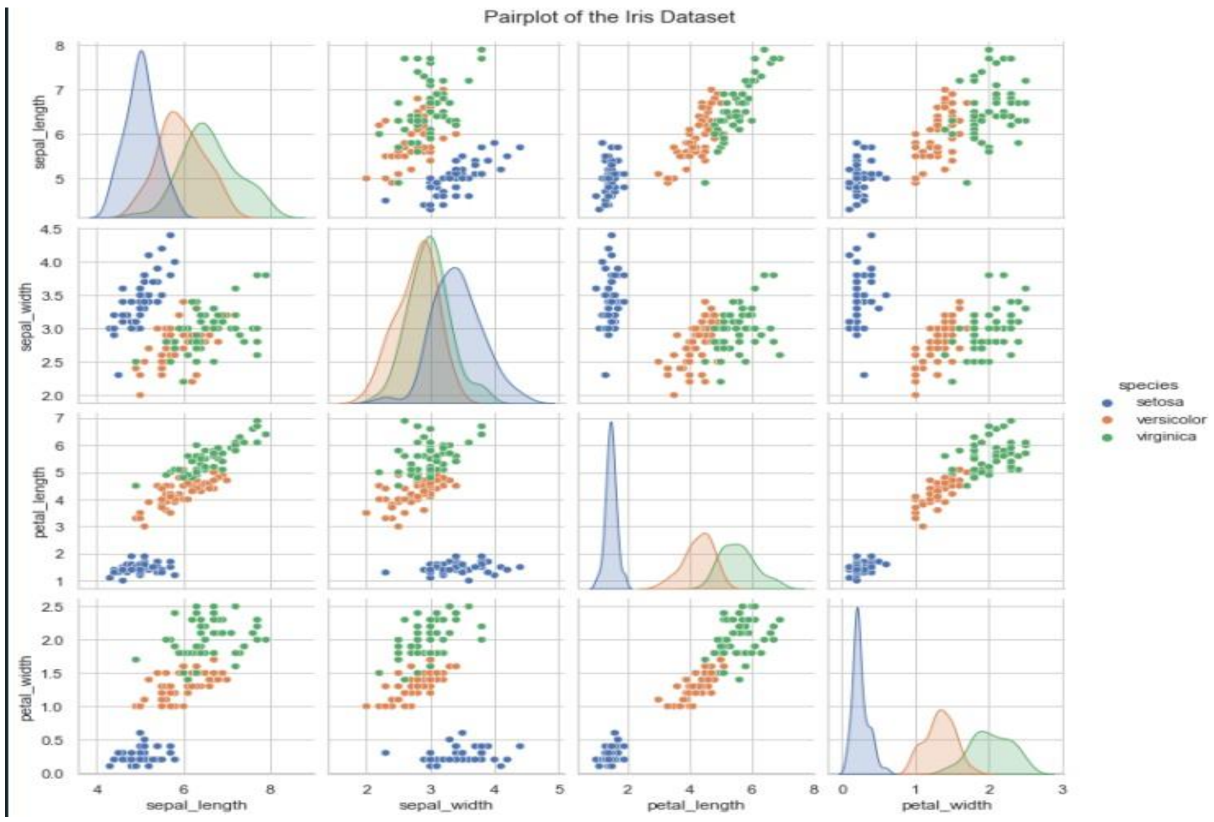
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Load the Iris dataset
file_path = r'C:\Users\SRM\Downloads\iris.csv'
df = pd.read_csv(file_path)
# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())
# Check for missing values
print("\nMissing values in the dataset:")
print(df.isnull().sum())
# Handling missing values (if any)
df = df.dropna()  # Drop rows with missing values
# Summary statistics
print("\nSummary statistics:")
print(df.describe())
# Data exploration and visualization
sns.set(style="whitegrid")
# Pairplot to see the pairwise relationships
print("\nGenerating pairplot...")
sns.pairplot(df, hue='species')
plt.suptitle("Pairplot of the Iris Dataset", y=1.02)
plt.show()
# Boxplot to see the distribution of each feature
print("\nGenerating boxplot...")
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, width=0.5, palette="colorblind")
```

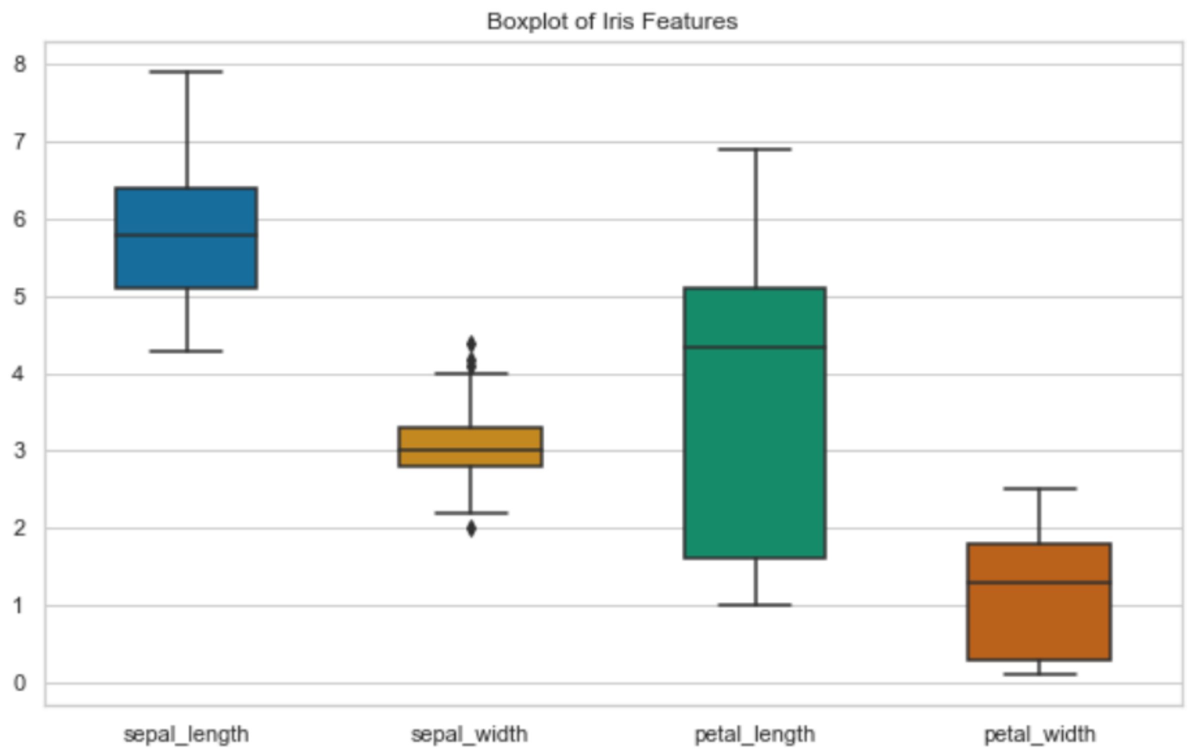
```
plt.title("Boxplot of Iris Features")
plt.show()

# Correlation heatmap
print("\nGenerating correlation heatmap...")
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
# Distribution of each species
print("\nGenerating countplot for species distribution...")
sns.countplot(x='species', data=df, palette="Set2")
plt.title("Species Distribution")
plt.show()
```

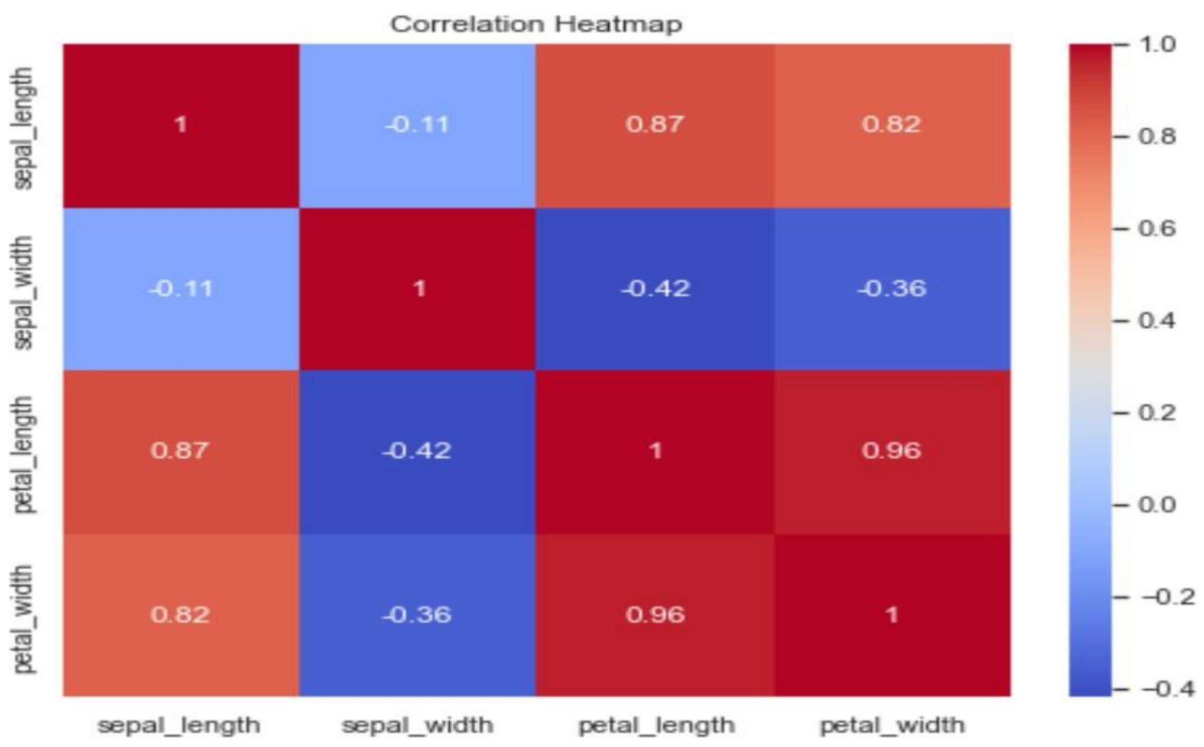
**OUTPUT:**

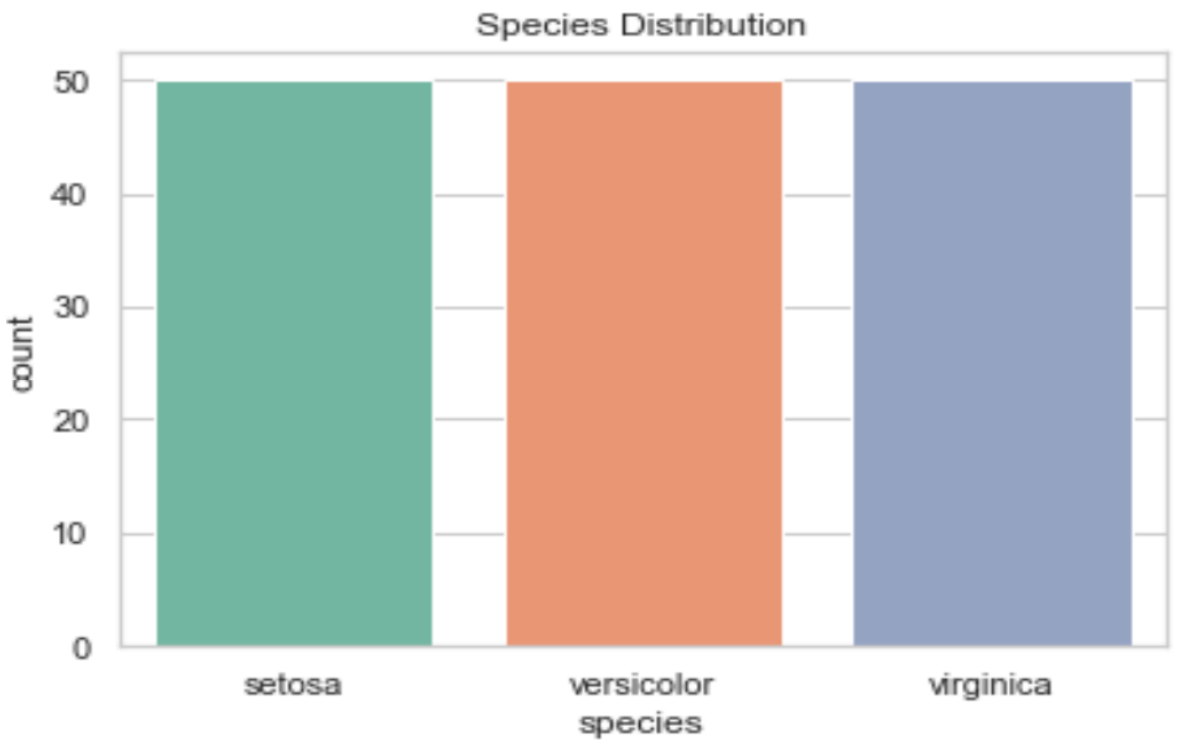


```
In [12]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
First few rows of the dataset:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa

Missing values in the dataset:
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64

Summary statistics:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```


Pairplot of the Iris Dataset
sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica


Boxplot of Iris Features
sepal_length
sepal_width
petal_length
petal_width


Correlation Heatmap
sepal_length
sepal_width
petal_length
petal_width
1
-0.11
0.87
0.82
-0.11
1
-0.42
-0.36
0.87
-0.42
1
0.96
0.82
-0.36
0.96
1
sepal_length
sepal_width
petal_length
petal_width
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Species Distribution
count
50
40
30
20
10
0
setosa
versicolor
virginica
species

**AUGMENTED QUESTIONS :**

**1. Write a Python program to read data from a CSV file named data.csv and perform the following tasks:**

- Print the first 5 rows of the DataFrame.
- Display the column names of the DataFrame

**2. Write a Python program to read data from an Excel file named data.xlsx and perform the following tasks:**

- Print the summary statistics of the DataFrame.
- Filter and display rows where the value in the 'Age' column is greater than 30.

**VIVA QUESTIONS:**

1. How do you read a CSV file into a Pandas DataFrame?
2. What function would you use to read data from an Excel file in Pandas, and how can you specify a particular sheet to load?
3. How can you load data directly from a URL into a Pandas DataFrame?
4. What Pandas functions can you use to get summary statistics (like mean, median, and standard deviation) for the Iris dataset?
5. How can you create a scatter plot of two features from the Iris dataset using Matplotlib or Seaborn?

**RESULT**

Thus the Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set was executed successfully.

| EX.NO:5.1 | Standard Deviation, Skewness and Kurtosis of Pima Indians Diabetes Dataset. |
|---|---|

**AIM:**

Perform basic data exploration and descriptive statistics on the diabetes dataset using Pandas and the Statistics module. This includes examining data structure, calculating mean, mode, median, variance, standard deviation, value counts, skewness, and kurtosis.

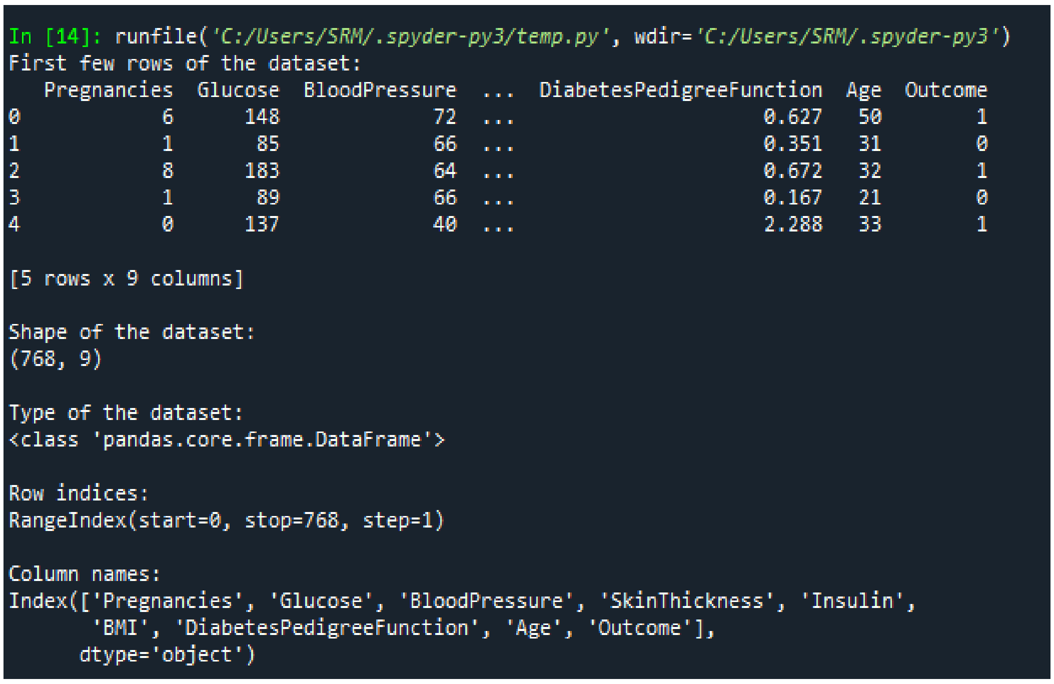
**ALGORITHM**

1. Read the diabetes dataset from a CSV file using Pandas.
2. Display the first few rows, shape, and data type of the dataset.
3. Calculate descriptive statistics using the Statistics module:
4. Calculate mean, mode, median, variance, and
5. Calculate value counts for the "Outcome" column.
6. Calculate skewness and kurtosis for the entire dataset.

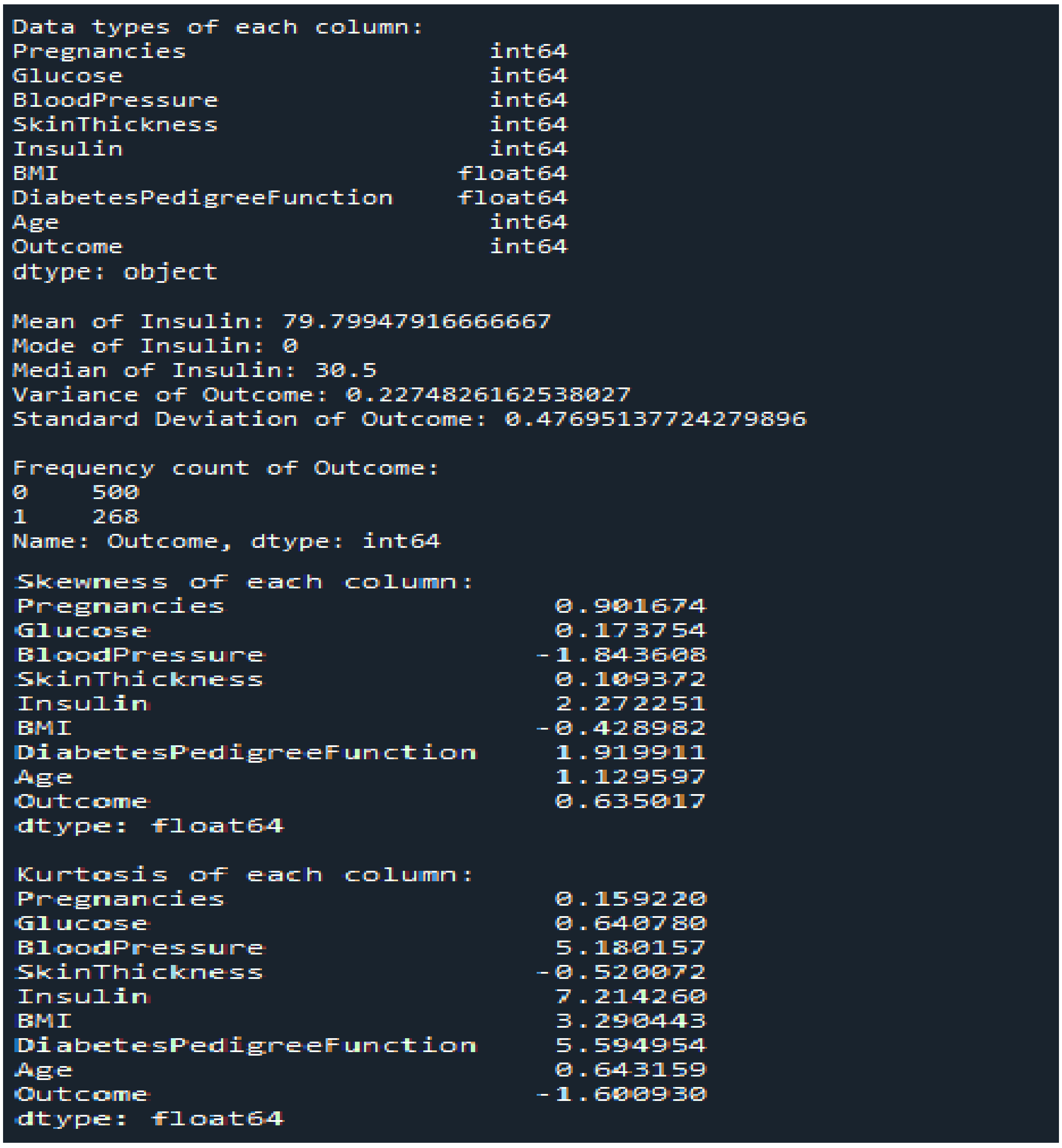
**PROGRAM**

```
import pandas as pd
import statistics
# Load the dataset
file_path = r'C:\Users\SRM\Downloads\diabetes.csv'  # Adjust the path to your file location
pima = pd.read_csv(file_path)
# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(pima.head())
# Print the shape of the dataset
print("\nShape of the dataset:")
print(pima.shape)
# Print the type of the dataset
print("\nType of the dataset:")
print(type(pima))
# Print the index of the dataset
print("\nRow indices:")
pima_row_idx = pima.index
print(pima_row_idx)
# Print the columns of the dataset
print("\nColumn names:")
pima_col_idx = pima.columns
print(pima_col_idx)
# Print the data types of the columns
print("\nData types of each column:")
print(pima.dtypes)
# Calculate statistical measures
mean = statistics.mean(pima["Insulin"])
mode = statistics.mode(pima["Insulin"])
median = statistics.median(pima["Insulin"])
variance = statistics.variance(pima["Outcome"])
standard_deviation = statistics.stdev(pima["Outcome"])
fre_count = pima["Outcome"].value_counts()
skew = pima.skew(axis=0, skipna=True)
kurt = pima.kurtosis(skipna=True)
# Print the calculated statistical measures
print("\nMean of Insulin:", mean)
print("Mode of Insulin:", mode)
print("Median of Insulin:", median)
print("Variance of Outcome:", variance)
print("Standard Deviation of Outcome:", standard_deviation)
print("\nFrequency count of Outcome:")
print(fre_count)
print("\nSkewness of each column:")
print(skew)
print("\nKurtosis of each column:")
print(kurt)
```

**OUTPUT:**



```
In [14]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/Users/SRM/.spyder-py3')
First few rows of the dataset:
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1

[5 rows x 9 columns]

Shape of the dataset:
(768, 9)

Type of the dataset:
<class 'pandas.core.frame.DataFrame'>

Row indices:
RangeIndex(start=0, stop=768, step=1)

Column names:
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```



```
Data types of each column:
Pregnancies                    int64
Glucose                        int64
BloodPressure                  int64
SkinThickness                  int64
Insulin                        int64
BMI                          float64
DiabetesPedigreeFunction     float64
Age                            int64
Outcome                        int64
dtype: object

Mean of Insulin: 79.79947916666667
Mode of Insulin: 0
Median of Insulin: 30.5
Variance of Outcome: 0.2274826162538027
Standard Deviation of Outcome: 0.47695137724279896

Frequency count of Outcome:
0    500
1    268
Name: Outcome, dtype: int64

Skewness of each column:
Pregnancies                 0.901674
Glucose                     0.173754
BloodPressure              -1.843608
SkinThickness               0.109372
Insulin                     2.272251
BMI                        -0.428982
DiabetesPedigreeFunction    1.919911
Age                         1.129597
Outcome                     0.635017
dtype: float64

Kurtosis of each column:
Pregnancies                 0.159220
Glucose                     0.640780
BloodPressure               5.180157
SkinThickness              -0.520072
Insulin                     7.214260
BMI                         3.290443
DiabetesPedigreeFunction    5.594954
Age                         0.643159
Outcome                    -1.600930
dtype: float64
```

**RESULT:**

Thus the Standard Deviation, Skewness and Kurtosis of Pima Indians Diabetes Dataset was executed successfully**.**

| EX.NO:5.2 | **Univariate analysis: Frequency, Mean, Median, Mode, Variance,Standard Deviation, Skewness and Kurtosis of UCI Diabetes Dataset.** |
|---|---|

**AIM:**

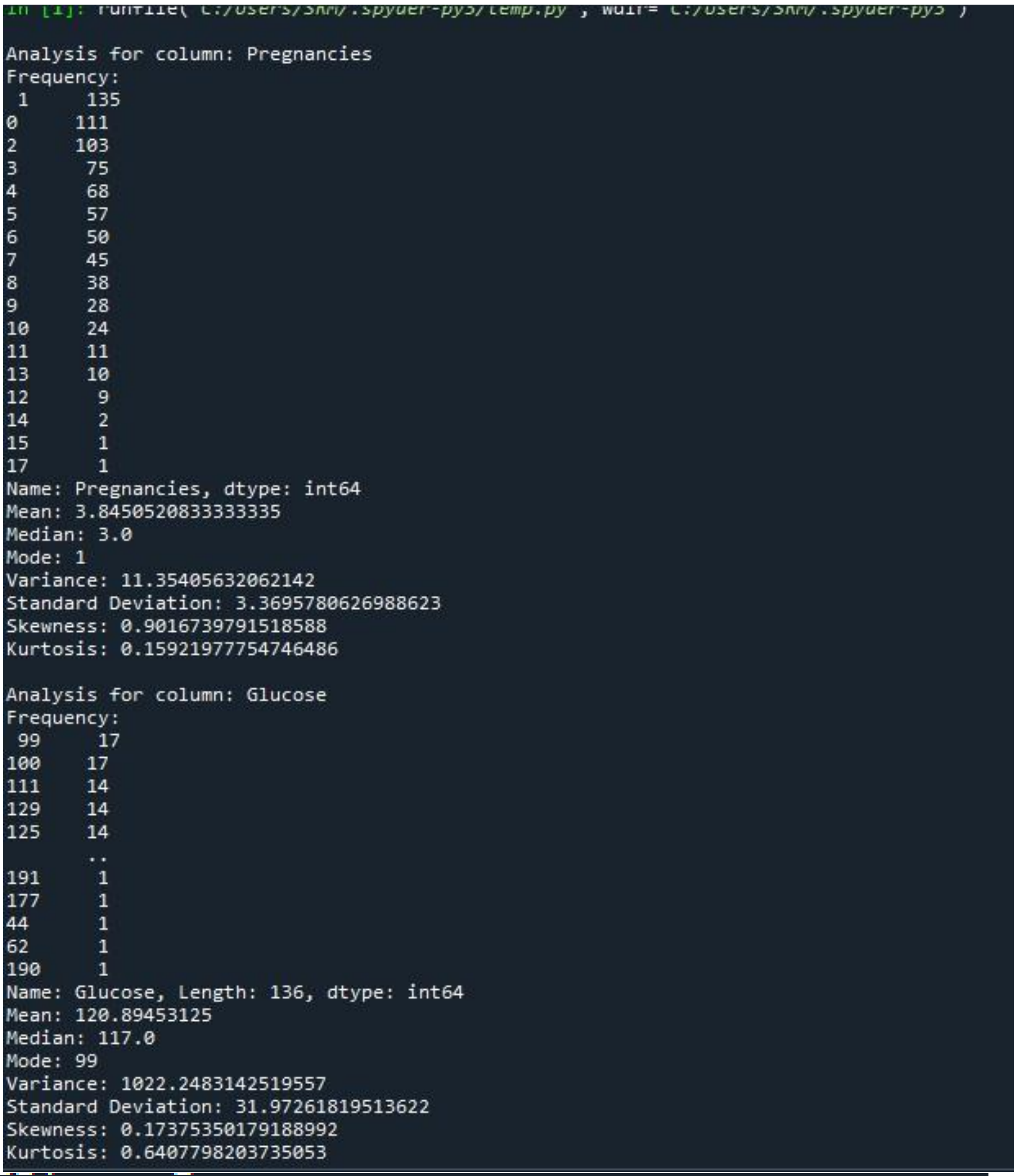
Conduct basic data exploration and compute descriptive statistics on the "num_lab_procedures" column of the diabetic dataset using Pandas and the Statistics module.

**ALGORITHM:**

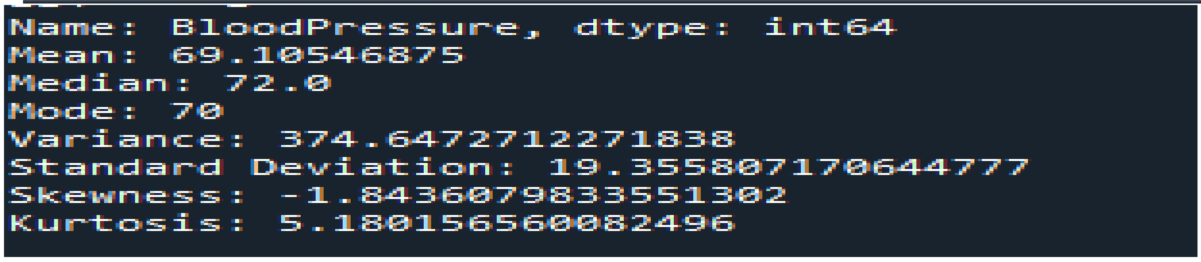
1. Read the diabetic dataset from a CSV file using Pandas.
2. Display the first few rows, shape, and data type of the dataset.
3. Retrieve and print row and column indices.
4. Calculate descriptive statistics using the Statistics module.

**PROGRAM**

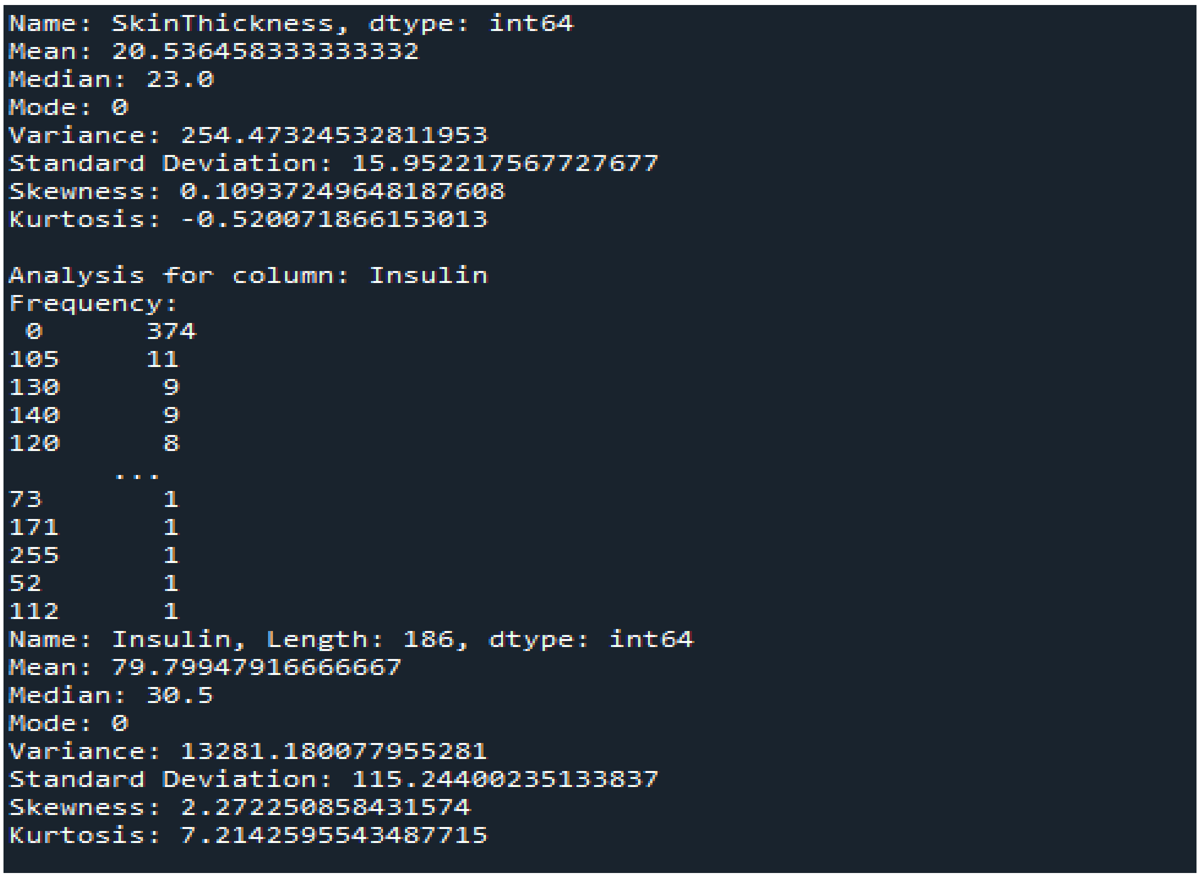
```
import pandas as pd
import statistics
# Load the dataset
file_path = r'C:\Users\SRM\Downloads\diabetes.csv'  # Adjust the path to your file location
pima = pd.read_csv(file_path)
# List of columns to analyze
columns = pima.columns
# Univariate analysis
for column in columns:
    print(f"\nAnalysis for column: {column}")
    # Frequency
    frequency = pima[column].value_counts()
    print("Frequency:\n", frequency)
    # Mean
    mean = pima[column].mean()
    print("Mean:", mean)
    # Median
    median = pima[column].median()
    print("Median:", median)
    # Mode
    mode = pima[column].mode()[0] if not pima[column].mode().empty else "No mode"
    print("Mode:", mode)
    # Variance
    variance = pima[column].var()
    print("Variance:", variance)
    # Standard Deviation
    std_dev = pima[column].std()
    print("Standard Deviation:", std_dev)

    # Skewness
    skewness = pima[column].skew()
    print("Skewness:", skewness)
    # Kurtosis
    kurtosis = pima[column].kurt()
    print("Kurtosis:", kurtosis)
```
**OUTPUT:**



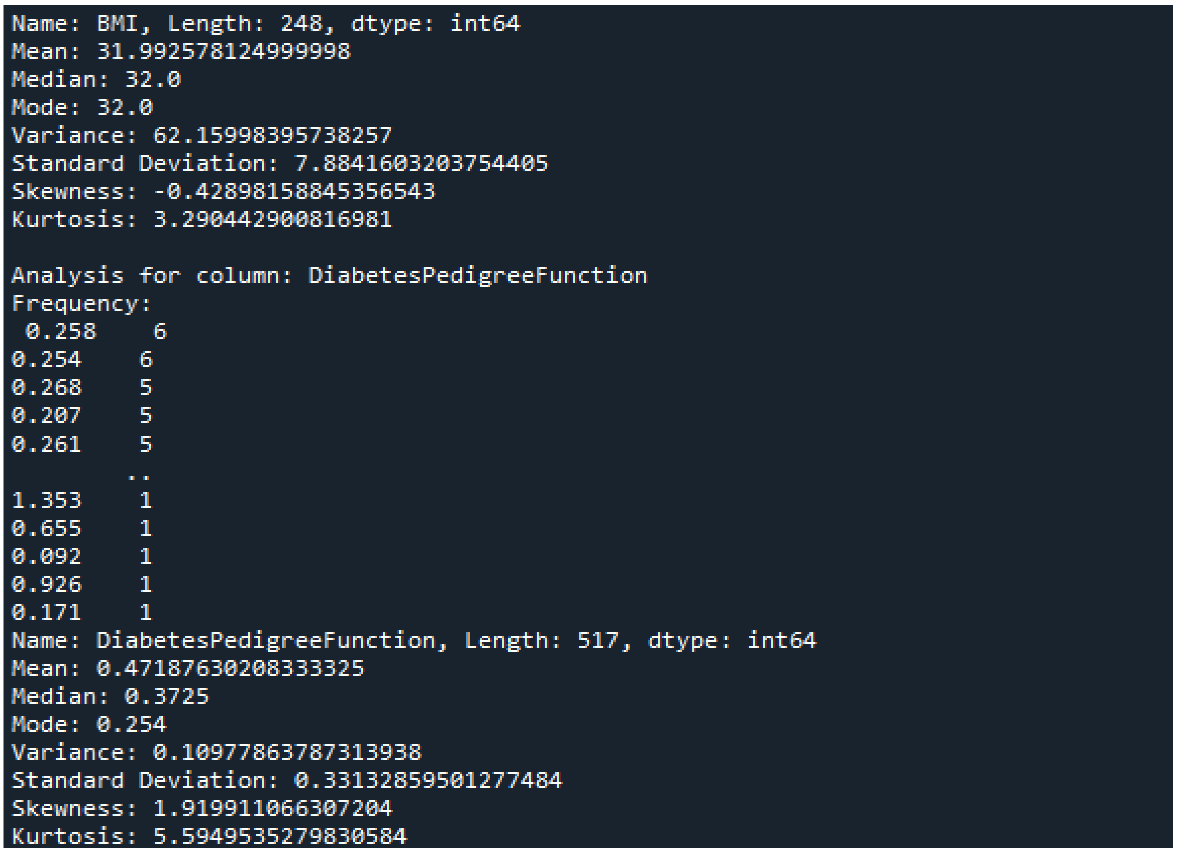
```
In [1]: runfile('C:/Users/SKM/.spyder-py3/temp.py', wdir='C:/Users/SKM/.spyder-py3')

Analysis for column: Pregnancies
Frequency:
 1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
15      1
17      1
Name: Pregnancies, dtype: int64
Mean: 3.8450520833333335
Median: 3.0
Mode: 1
Variance: 11.35405632062142
Standard Deviation: 3.3695780626988623
Skewness: 0.9016739791518588
Kurtosis: 0.15921977754746486

Analysis for column: Glucose
Frequency:
 99     17
100    17
111    14
129    14
125    14
       ..
191     1
177     1
44      1
62      1
190     1
Name: Glucose, Length: 136, dtype: int64
Mean: 120.89453125
Median: 117.0
Mode: 99
Variance: 1022.2483142519557
Standard Deviation: 31.97261819513622
Skewness: 0.17375350179188992
Kurtosis: 0.6407798203735053
```



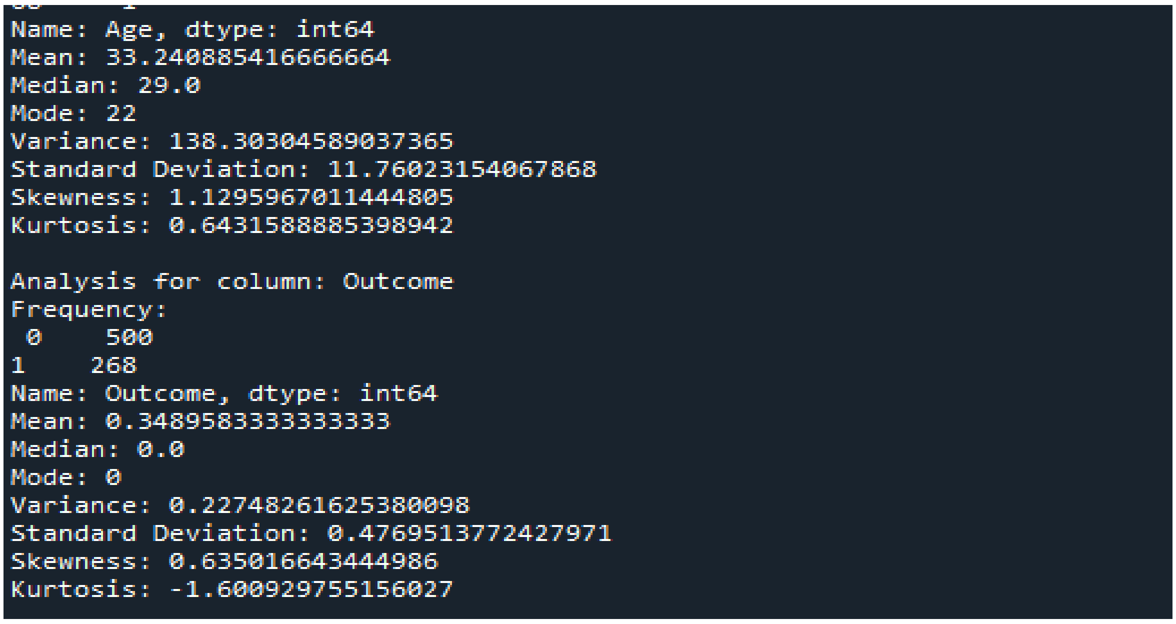
```
Name: BloodPressure, dtype: int64
Mean: 69.10546875
Median: 72.0
Mode: 70
Variance: 374.6472712271838
Standard Deviation: 19.355807170644777
Skewness: -1.8436079833551302
Kurtosis: 5.180156560082496
```



```
Name: SkinThickness, dtype: int64
Mean: 20.536458333333332
Median: 23.0
Mode: 0
Variance: 254.47324532811953
Standard Deviation: 15.952217567727677
Skewness: 0.10937249648187608
Kurtosis: -0.520071866153013

Analysis for column: Insulin
Frequency:
 0      374
105     11
130      9
140      9
120      8
       ...
73       1
171      1
255      1
52       1
112      1
Name: Insulin, Length: 186, dtype: int64
Mean: 79.79947916666667
Median: 30.5
Mode: 0
Variance: 13281.180077955281
Standard Deviation: 115.24400235133837
Skewness: 2.272250858431574
Kurtosis: 7.2142595543487715
```



```
Name: BMI, Length: 248, dtype: int64
Mean: 31.992578124999998
Median: 32.0
Mode: 32.0
Variance: 62.15998395738257
Standard Deviation: 7.8841603203754405
Skewness: -0.42898158845356543
Kurtosis: 3.290442900816981

Analysis for column: DiabetesPedigreeFunction
Frequency:
 0.258    6
0.254    6
0.268    5
0.207    5
0.261    5
        ..
1.353    1
0.655    1
0.092    1
0.926    1
0.171    1
Name: DiabetesPedigreeFunction, Length: 517, dtype: int64
Mean: 0.47187630208333325
Median: 0.3725
Mode: 0.254
Variance: 0.10977863787313938
Standard Deviation: 0.33132859501277484
Skewness: 1.919911066307204
Kurtosis: 5.5949535279830584
```



```
Name: Age, dtype: int64
Mean: 33.240885416666664
Median: 29.0
Mode: 22
Variance: 138.30304589037365
Standard Deviation: 11.76023154067868
Skewness: 1.1295967011444805
Kurtosis: 0.6431588885398942

Analysis for column: Outcome
Frequency:
 0    500
1    268
Name: Outcome, dtype: int64
Mean: 0.3489583333333333
Median: 0.0
Mode: 0
Variance: 0.22748261625380098
Standard Deviation: 0.4769513772427971
Skewness: 0.635016643444986
Kurtosis: -1.600929755156027
```

**RESULT:**

Thus the Univariate analysis: Frequency, Mean, Median, Mode, Variance,Standard Deviation, Skewness and Kurtosis of UCI Diabetes Dataset was executed successfully.

| EX.NO:5.3 | Bivariate Analysis-Program for linear regression |
|---|---|

**AIM:**

Explore the relationship between Glucose and Blood Pressure in the diabetes dataset using a scatter plot and create a linear regression model to predict Age based on BMI.

**ALGORITHIM:**

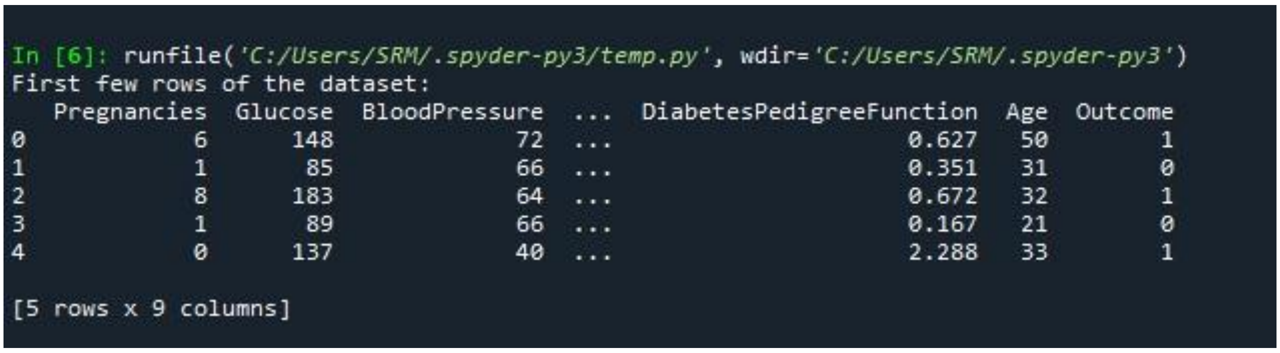
1. Import necessary libraries: NumPy, Pandas, Seaborn, Statistics, Matplotlib, scikit- learn, and statsmodels.
2. Read the diabetes dataset from a CSV file.
3. Select relevant columns for analysis (Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age).
4. Create a scatter plot to visualize the relationship between Glucose and Blood Pressure.
5. Extract features (X) and target variable (Y) for linear regression (e.g., Age vs BMI).
6. Use scikit-learn'sLinearRegression to fit a linear model.
7. Display the scatter plot.
8. Fit a linear regression model to predict Age based on BMI using statsmodels.

**PROGRAM**

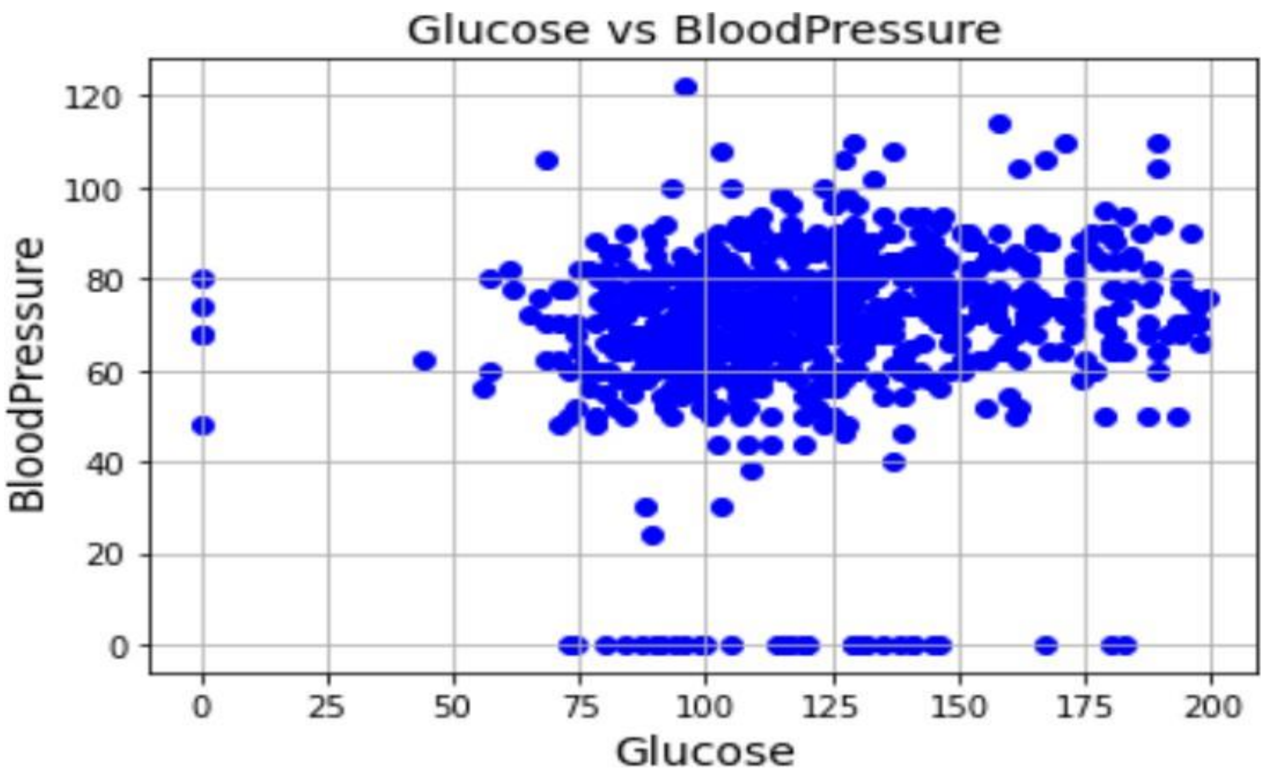
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Load the dataset
file_path = r'C:\Users\SRM\Downloads\diabetes.csv' # Adjust the path to your file location
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
head = df.head()
print("First few rows of the dataset:")
print(head)

# Define feature columns and target column
cols = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI",
"DiabetesPedigreeFunction", "Age"]
X = df[['BMI']]  # Features need to be a 2D array
Y = df['Age']    # Target variable

# Scatter plot of Glucose vs BloodPressure
plt.scatter(df['Glucose'], df['BloodPressure'], color='blue')
plt.title('Glucose vs BloodPressure', fontsize=14)
plt.xlabel('Glucose', fontsize=14)
plt.ylabel('BloodPressure', fontsize=14)
plt.grid(True)
plt.show()
# Fit a linear regression model
model = LinearRegression()
```

```
model.fit(X, Y)



# Print model coefficients
print(f"Intercept: {model.intercept_}")
print(f"Coefficient for BMI: {model.coef_[0]}")
```

**OUTPUT:**



```
In [6]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/Users/SRM/.spyder-py3')
First few rows of the dataset:
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1

[5 rows x 9 columns]
```



**RESULT:**

Thus the Bivariate Analysis-Program for linear regression was executed successfully.

| EX.NO:5.4 | **Bivariate Analysis Logistic regression** |
|---|---|

**AIM:**

Perform logistic regression analysis on the diabetes dataset to predict the likelihood of diabetes based on various independent variables. Evaluate the model's performance using classification report and confusion matrix.

**ALGORITHM**

1. Import necessary libraries: NumPy, Pandas, Seaborn,Matplotlib, statsmodels, and scikit-learn.

2. Read the diabetes dataset from a CSV file.

3. Define a logistic regression model using statsmodels withdifferent sets of independent variables (cols2, cols3cols4).

4. Import LogisticRegression from scikit-learn for additionalevaluation metrics.

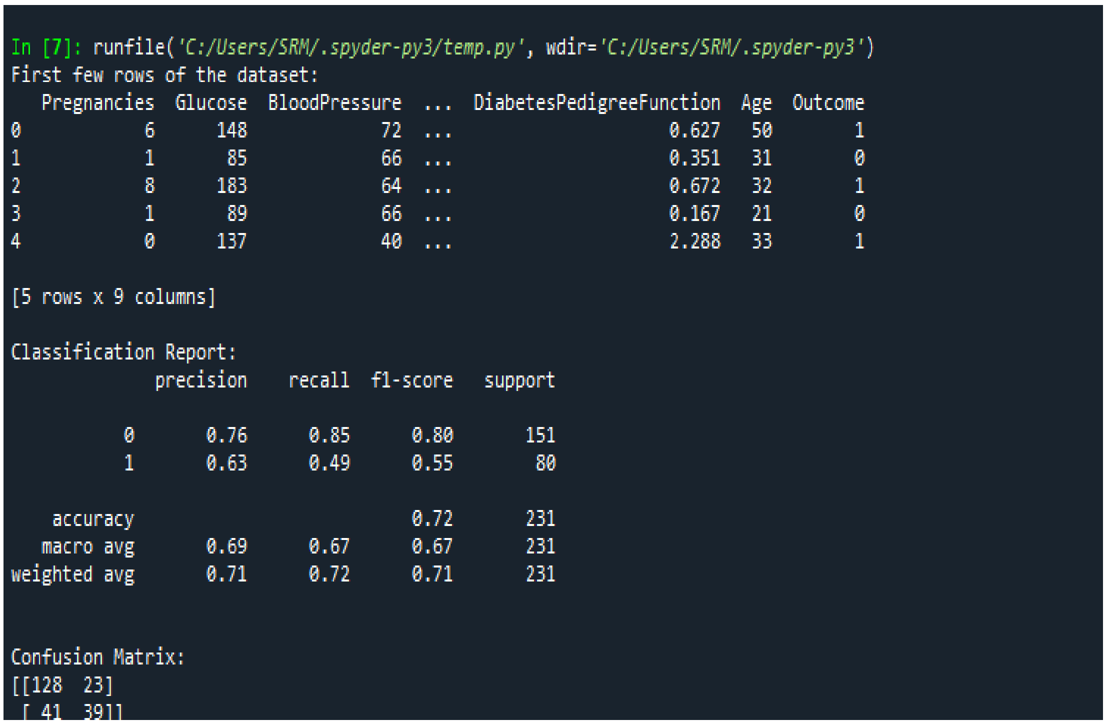
5.Calculate and print the confusion matrix to evaluate themodel's accuracy.

**PROGRAM**

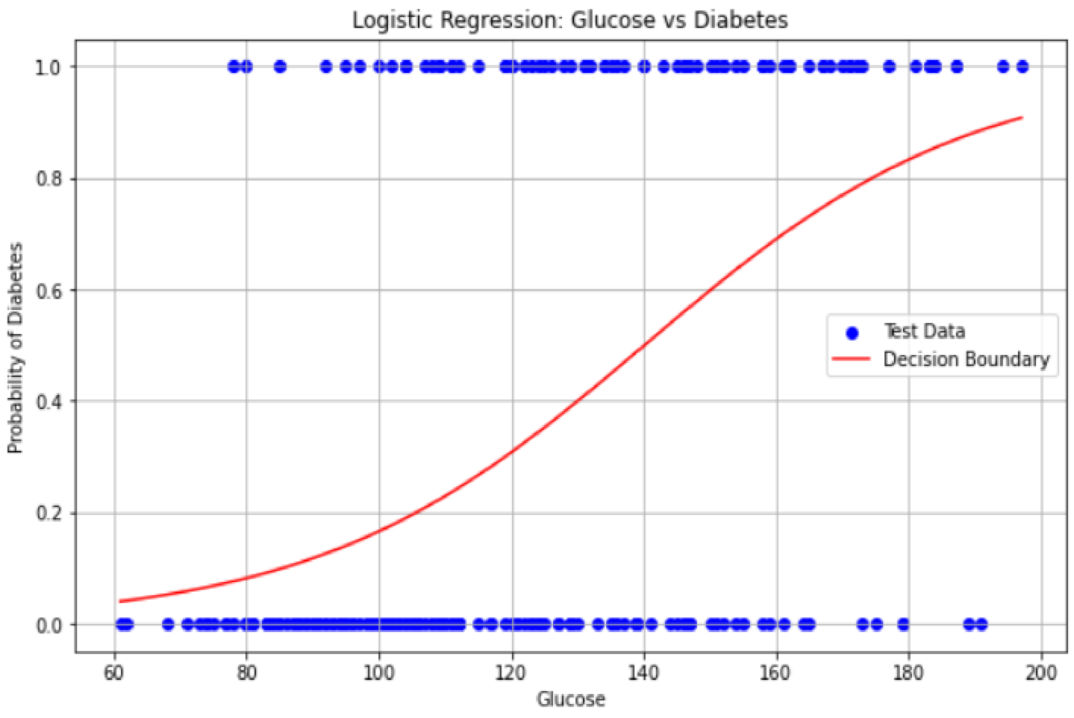
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Load the dataset
file_path = r'C:\Users\SRM\Downloads\diabetes.csv'  # Adjust the path to your file location
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())

# Define the feature and target variables
X = df[['Glucose']]  # Feature: Glucose
Y = df['Outcome']   # Target: Diabetes outcome

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# Initialize and fit the logistic regression model
model = LogisticRegression()
model.fit(X_train, Y_train)

# Predict on the test set
Y_pred = model.predict(X_test)

# Print classification report and confusion matrix
```

```
print("\nClassification Report:")
print(classification_report(Y_test, Y_pred))

print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(Y_test, Y_pred)
print(conf_matrix)

# Visualization: Scatter plot with decision boundary
plt.figure(figsize=(10, 6))

# Plot the data points
plt.scatter(X_test, Y_test, color='blue', label='Test Data')

# Plot decision boundary
x_values = np.linspace(X_test.min(), X_test.max(), 100)
y_values = model.predict_proba(x_values.reshape(-1, 1))[:, 1]
plt.plot(x_values, y_values, color='red', label='Decision Boundary')

# Labels and title
plt.xlabel('Glucose')
plt.ylabel('Probability of Diabetes')
plt.title('Logistic Regression: Glucose vs Diabetes')
plt.legend()
plt.grid(True)
plt.show()
```

**OUTPUT:**



```
In [7]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/Users/SRM/.spyder-py3')
First few rows of the dataset:
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1

[5 rows x 9 columns]

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.85      0.80       151
           1       0.63      0.49      0.55        80

    accuracy                           0.72       231
   macro avg       0.69      0.67      0.67       231
weighted avg       0.71      0.72      0.71       231


Confusion Matrix:
[[128  23]
 [ 41  39]]
```



**RESULT:**

Thus the Bivariate Analysis Logistic regression was executed successfully.

| EX.NO:5.5 | MULTIPLE REGRESSION ANALYSIS |
|---|---|

**AIM:**

The aim of the provided code is to analyze and visualize a dataset related to diabetes using Python and various libraries such as pandas, seaborn, matplotlib, and statsmodels. The code explores the dataset, calculates and visualizes the correlation matrix, generates a quantile-quantile (QQ) plot for the 'Age' variable, and produces scatter matrices for different subsets of the data.

**ALGORITHM:**

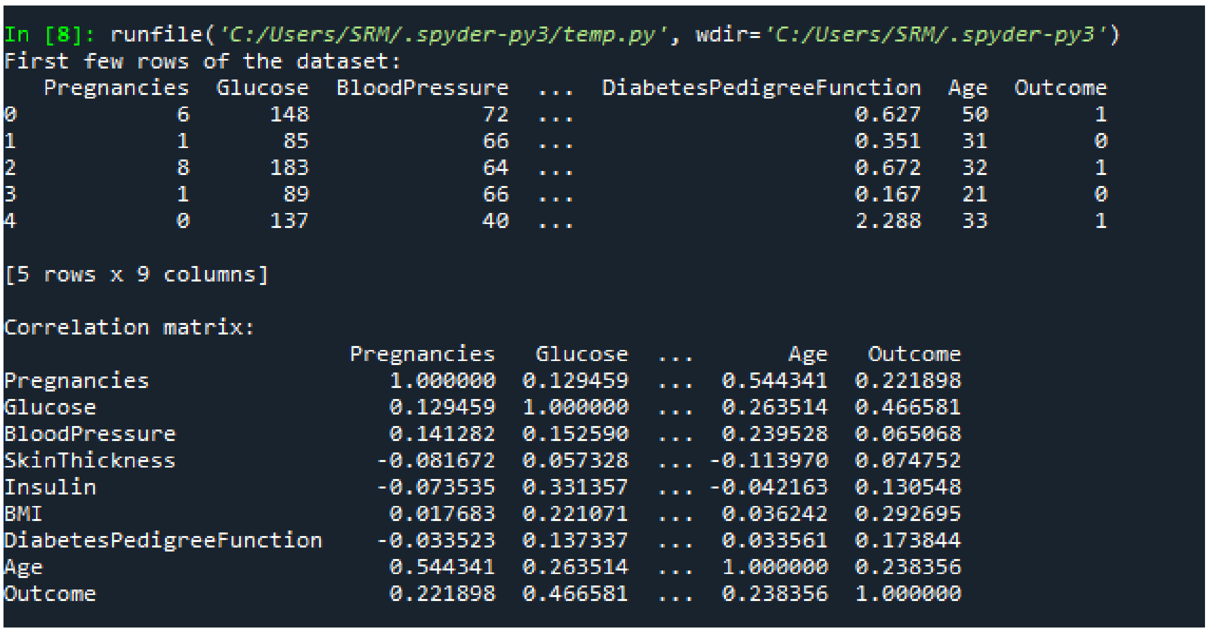
1. Import necessary libraries: pandas, seaborn, matplotlib, statsmodels, and pylab.
2. Read the diabetes dataset from the 'diabetes.csv' file into a pandas DataFrame (df).
3. Display the first few rows of the dataset using `df.head()`.
4. Calculate the correlation matrix (`corr`) for the variables in the dataset.
5. Create a heatmap (`hm`) using seaborn to visualize the correlation matrix.
6. Generate a quantile-quantile (QQ) plot for the 'Age' variable using statsmodels.
7. Display the scatter matrix for all variables in the dataset using matplotlib and seaborn.
8. Extract a subset of the data (`data`) containing the columns 'Pregnancies', 'Glucose', and 'BloodPressure'.
9. Display the scatter matrix for the subset of data.

**PROGRAM:**

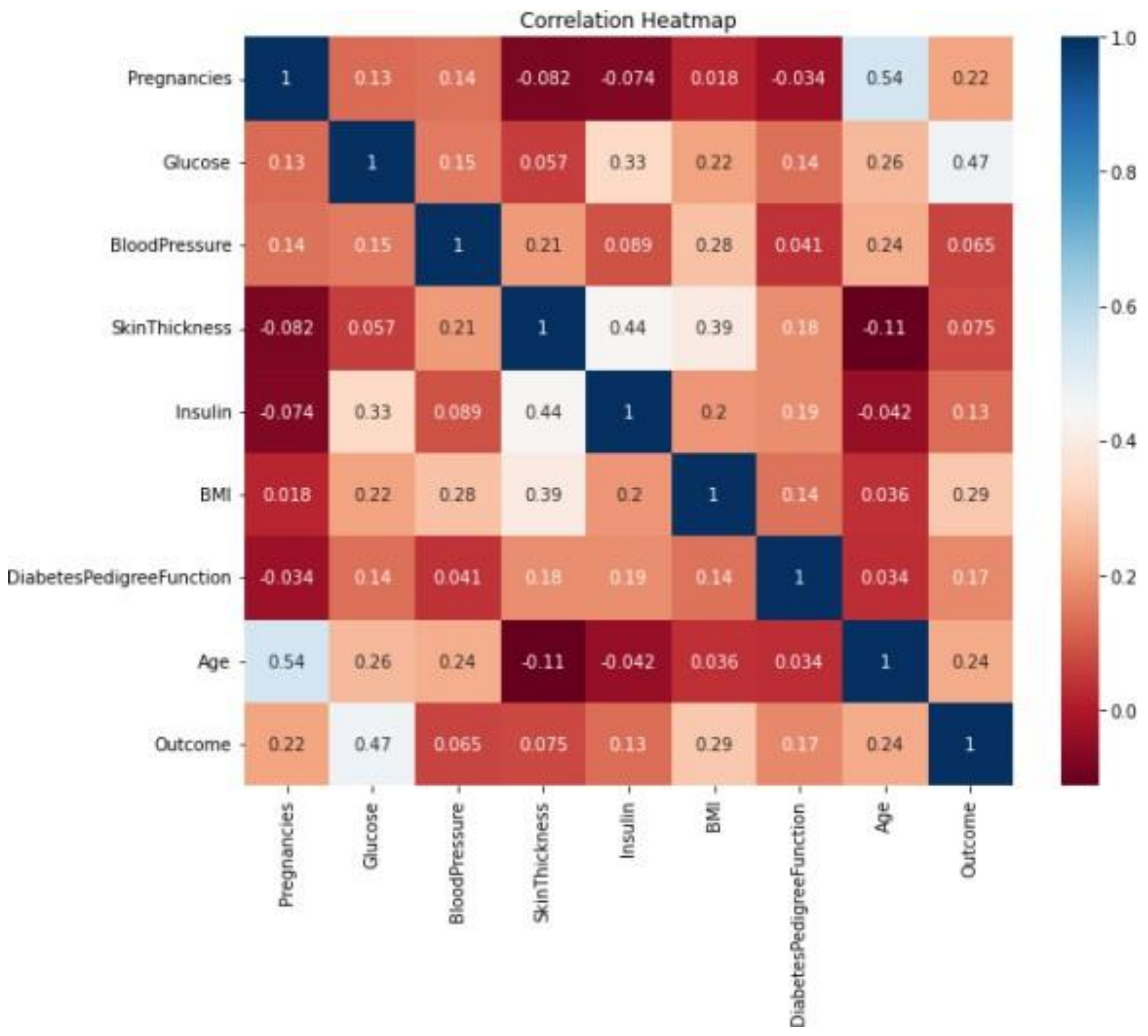
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pylab

# Load the dataset
file_path = r'C:\Users\SRM\Downloads\diabetes.csv'  # Adjust the path to your file location
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())

# Calculate and print the correlation matrix
corr = df.corr()
print("\nCorrelation matrix:")
print(corr)

# Heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
hm = sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='RdBu', annot=True)
```

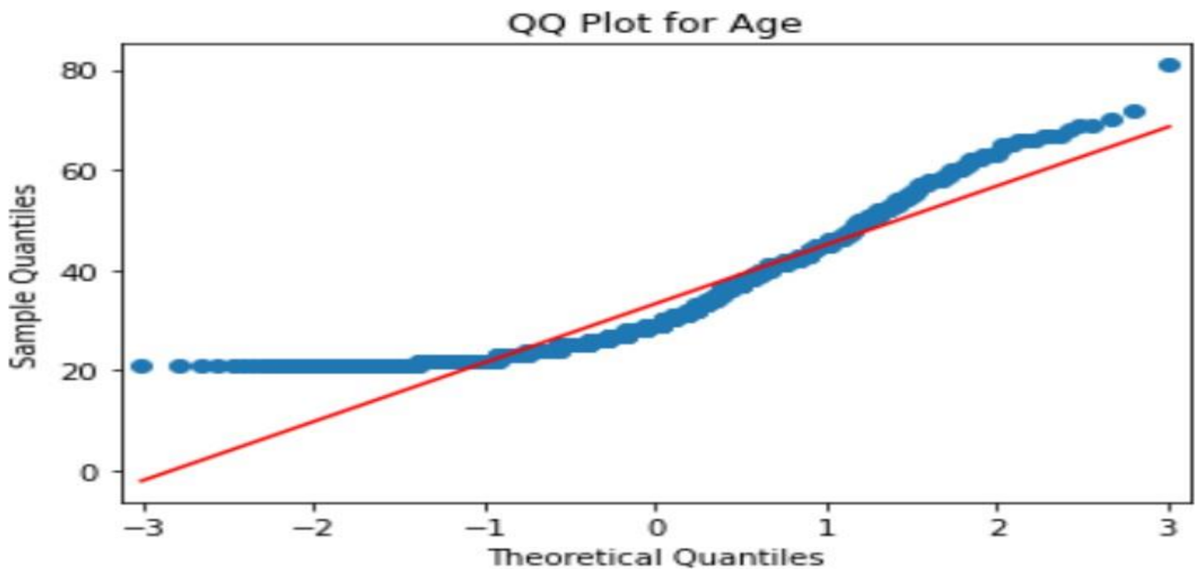
```
plt.title('Correlation Heatmap')
plt.show()




# QQ plot for the 'Age' column
data = df['Age']
plt.figure(figsize=(8, 6))
sm.qqplot(data, line='s')
plt.title('QQ Plot for Age')
plt.show()

# Scatter matrix for the entire DataFrame
plt.figure(figsize=(15, 10))
pd.plotting.scatter_matrix(df, alpha=0.8, figsize=(15, 15), diagonal='kde')
plt.suptitle('Scatter Matrix for Diabetes Dataset')
plt.show()

# Scatter matrix for selected features
data = df[["Pregnancies", "Glucose", "BloodPressure"]]
plt.figure(figsize=(10, 7))
pd.plotting.scatter_matrix(data, alpha=0.8, figsize=(10, 7), diagonal='kde')
plt.suptitle('Scatter Matrix for Selected Features')
plt.show()
```

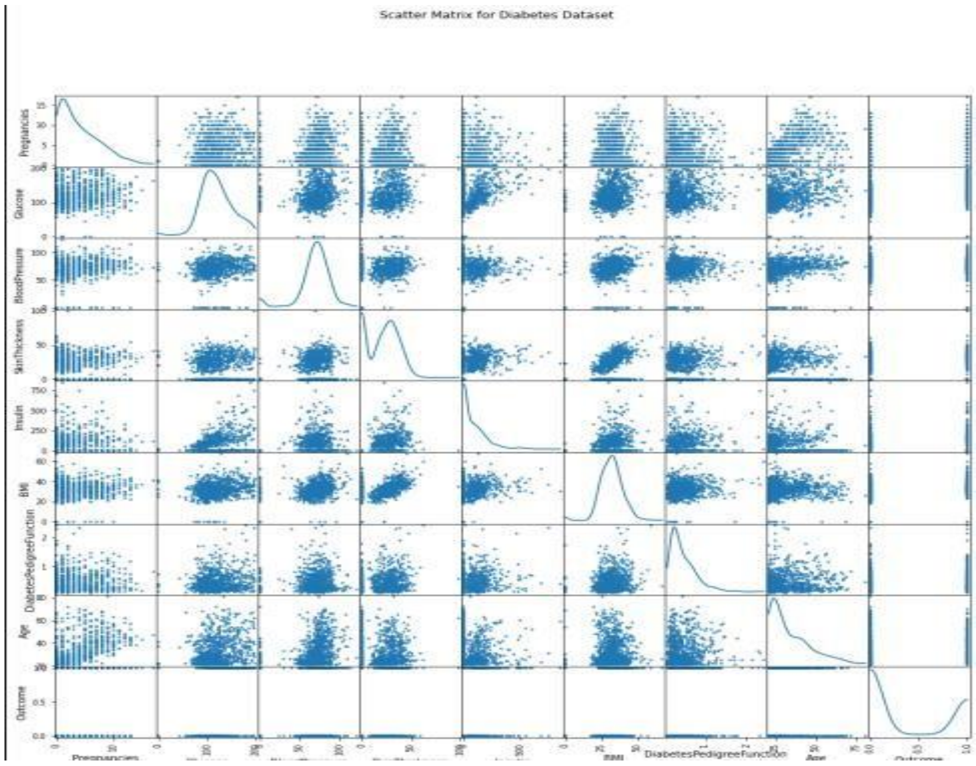
**OUTPUT:**

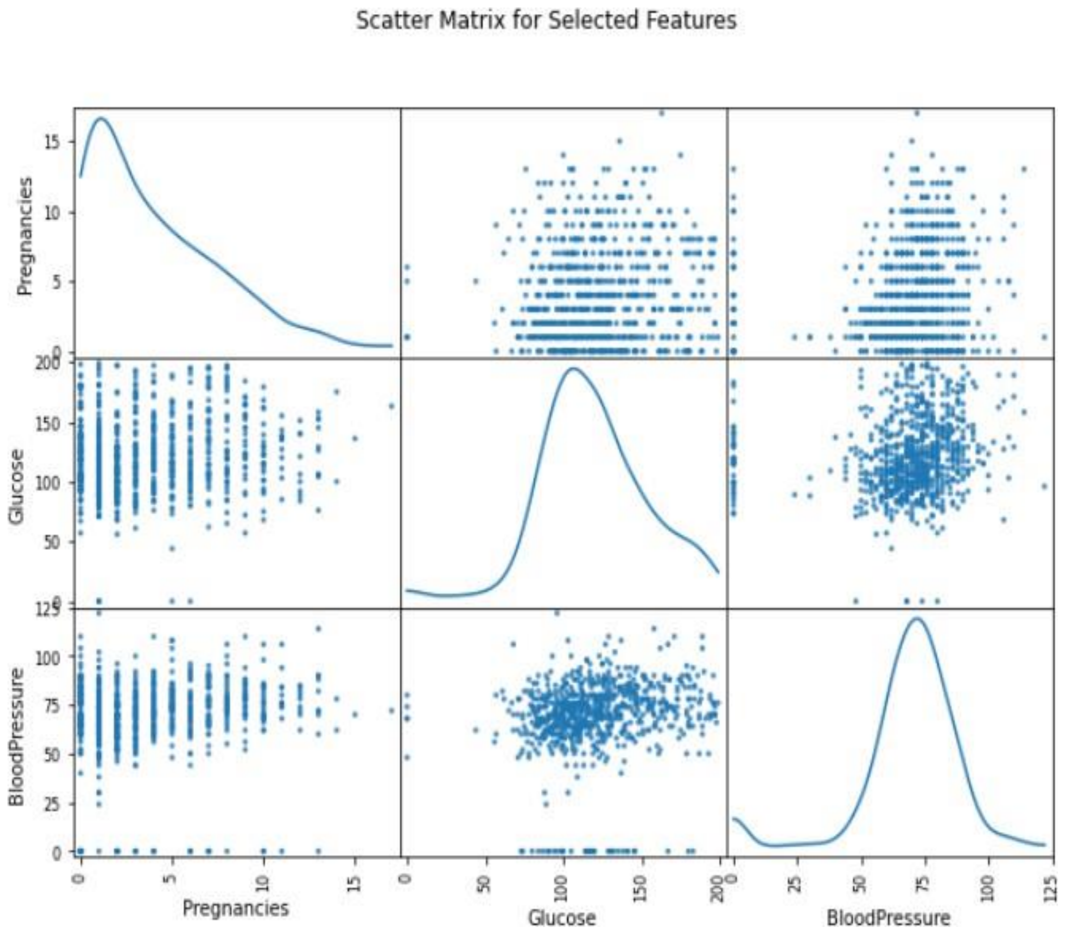


```
In [8]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/Users/SRM/.spyder-py3')
First few rows of the dataset:
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1

[5 rows x 9 columns]

Correlation matrix:
                          Pregnancies   Glucose  ...       Age   Outcome
Pregnancies                  1.000000  0.129459  ...  0.544341  0.221898
Glucose                      0.129459  1.000000  ...  0.263514  0.466581
BloodPressure                0.141282  0.152590  ...  0.239528  0.065068
SkinThickness               -0.081672  0.057328  ... -0.113970  0.074752
Insulin                     -0.073535  0.331357  ... -0.042163  0.130548
BMI                          0.017683  0.221071  ...  0.036242  0.292695
DiabetesPedigreeFunction    -0.033523  0.137337  ...  0.033561  0.173844
Age                          0.544341  0.263514  ...  1.000000  0.238356
Outcome                      0.221898  0.466581  ...  0.238356  1.000000
```


Correlation Heatmap
Pregnancies 1 0.13 0.14 -0.082 -0.074 0.018 -0.034 0.54 0.22
Glucose 0.13 1 0.15 0.057 0.33 0.22 0.14 0.26 0.47
BloodPressure 0.14 0.15 1 0.21 0.089 0.28 0.041 0.24 0.065
SkinThickness -0.082 0.057 0.21 1 0.44 0.39 0.18 -0.11 0.075
Insulin -0.074 0.33 0.089 0.44 1 0.2 0.19 -0.042 0.13
BMI 0.018 0.22 0.28 0.39 0.2 1 0.14 0.036 0.29
DiabetesPedigreeFunction -0.034 0.14 0.041 0.18 0.19 0.14 1 0.034 0.17
Age 0.54 0.26 0.24 -0.11 -0.042 0.036 0.034 1 0.24
Outcome 0.22 0.47 0.065 0.075 0.13 0.29 0.17 0.24 1
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0


QQ Plot for Age
Sample Quantiles
Theoretical Quantiles
80
60
40
20
0
−3
−2
−1
0
1
2
3


Scatter Matrix for Diabetes Dataset
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome


Scatter Matrix for Selected Features
Pregnancies
Glucose
BloodPressure

**AUGMENTED QUESTIONS:**

1. Write a Python program to perform the following tasks using the UCI Diabetes dataset:

   - Calculate and print the mean, median, and standard deviation of the 'Glucose' column.
   - Fit a linear regression model to predict 'Outcome' based on 'Glucose' and print the model's R-squared value.

2. Write a Python program to perform the following tasks using the Pima Indians Diabetes dataset:

   - Calculate and print the frequency of unique values in the 'Outcome' column.
   - Fit a logistic regression model to predict 'Outcome' based on 'Glucose' and print the model's accuracy.

**VIVA QUESTIONS :**

1. How do you calculate the mean, median, and standard deviation for a column in the Diabetes dataset?
2. What is the purpose of performing linear regression, and how would you apply it to the Diabetes dataset?
3. How can you fit a logistic regression model to the Diabetes dataset and evaluate its performance?
4. What is multiple regression analysis, and how would you use it to analyze the Diabetes dataset?
5. How would you compare the results of your analysis between the UCI Diabetes dataset and the Pima Indians Diabetes dataset?

**RESULT:**

The code produces visualizations and analyses, including a correlation matrix heatmap, a QQ plot for 'Age,' and scatter matrices, offering insights into variable relationships and distributions within the diabetes dataset.

| EX.NO:6 | Apply and explore various plotting functions on UCI data sets. |
|---|---|

**a. Normal curves**

**AIM:**

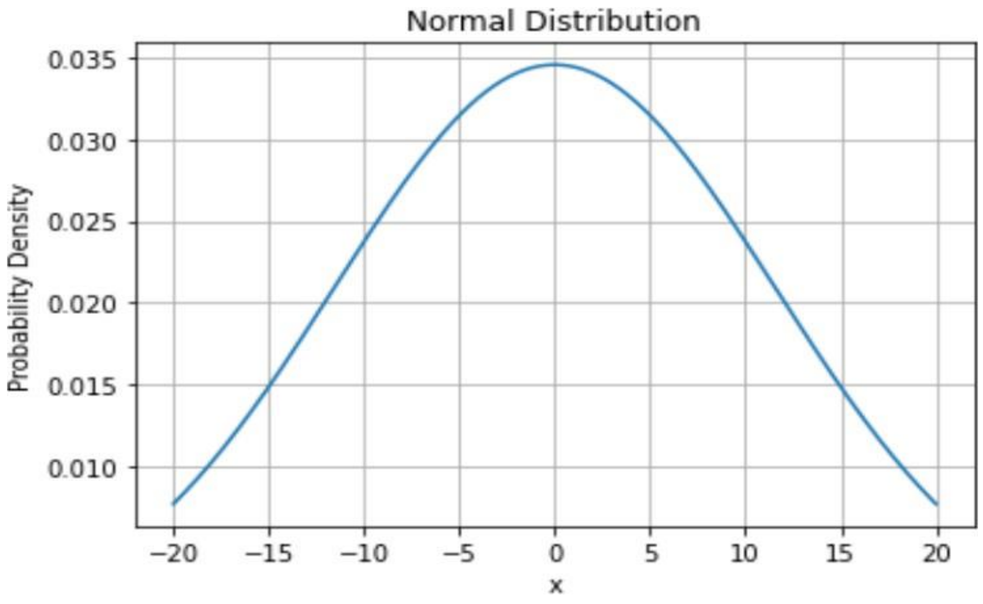
The aim of the provided Python code is to generate and visualize the probability density function (PDF) of a normal distribution. The range for the x-axis values is set from -20 to 20 with a step size of 0.01.

**ALGORITHM:**

1. Import necessary libraries: **numpy** for numerical operations, **matplotlib.pyplot** for plotting, **scipy.stats.norm** for the normal distribution, and **statistics** for calculating mean and standard deviation (although, this is later corrected to use **numpy** functions).
2. Create an array **x_axis** with values ranging from -20 to 20 with a step size of 0.01.
3. Incorrect calculation of mean and standard deviation:
    - The initial code attempts to use **statistics.mean** and **statistics.stdev** on the **x_axis** array, which is not the correct approach. The correct approach is to use **numpy.mean** and **numpy.std**.
4. Plot the normal distribution PDF:
    - The code uses **plt.plot** to plot the normal distribution PDF using **norm.pdf** from the **scipy.stats** module. The mean and standard deviation are used as parameters for the normal distribution.
5. Display the plot:
    - The code uses **plt.show()** to display the generated plot.

**PROGRAM**

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import statistics

# Generate x values from -20 to 20 with a step of 0.01
x_axis = np.arange(-20, 20, 0.01)

# Calculate mean and standard deviation of the x_axis values
mean = np.mean(x_axis)
sd = np.std(x_axis)

# Plot the normal distribution
plt.plot(x_axis, norm.pdf(x_axis, mean, sd))
plt.title('Normal Distribution')
plt.xlabel('x')
plt.ylabel('Probability Density')
plt.grid(True)
plt.show()
```



**RESULT:**

The corrected code will generate a plot displaying the probability density function of a normal distribution with mean and standard deviation calculated from the range of values specified on the x-axis (-20 to 20 with steps of 0.01). The resulting plot visually represents the distribution of the random variable within that range.

**b. Density and contour plots**

**AIM:**

The aim of this program is to use Python code is to create a 2D contour plot with filled contours and an overlaid image of a mathematical function. The function `f(x, y)` is defined, and the contours of this function are plotted on a grid using Matplotlib. Additionally, an image of the function is displayed using `plt.imshow`, and a color bar is added for reference.

**ALGORITHM:**

1. Import necessary libraries:
   - `matplotlib.pyplot` for plotting.
   - `numpy` for numerical operations.

2. Set the plotting style to 'seaborn-white' using `plt.style.use('seaborn-white')`.

3. Define the function `f(x, y)` which represents a mathematical expression involving sine, cosine, and exponentiation.

4. Generate evenly spaced values for `x` and `y` using `np.linspace`.

5. Create a mesh grid (`X`, `Y`) using `np.meshgrid` based on the generated `x` and `y` values.

6. Evaluate the function `f(X, Y)` for each point on the grid and store the result in the variable `Z`.

7. Plot black contours of the function using `plt.contour` with a single line.

8. Plot filled contours with a colormap ('RdGy') using `plt.contour` to emphasize different levels of the function.

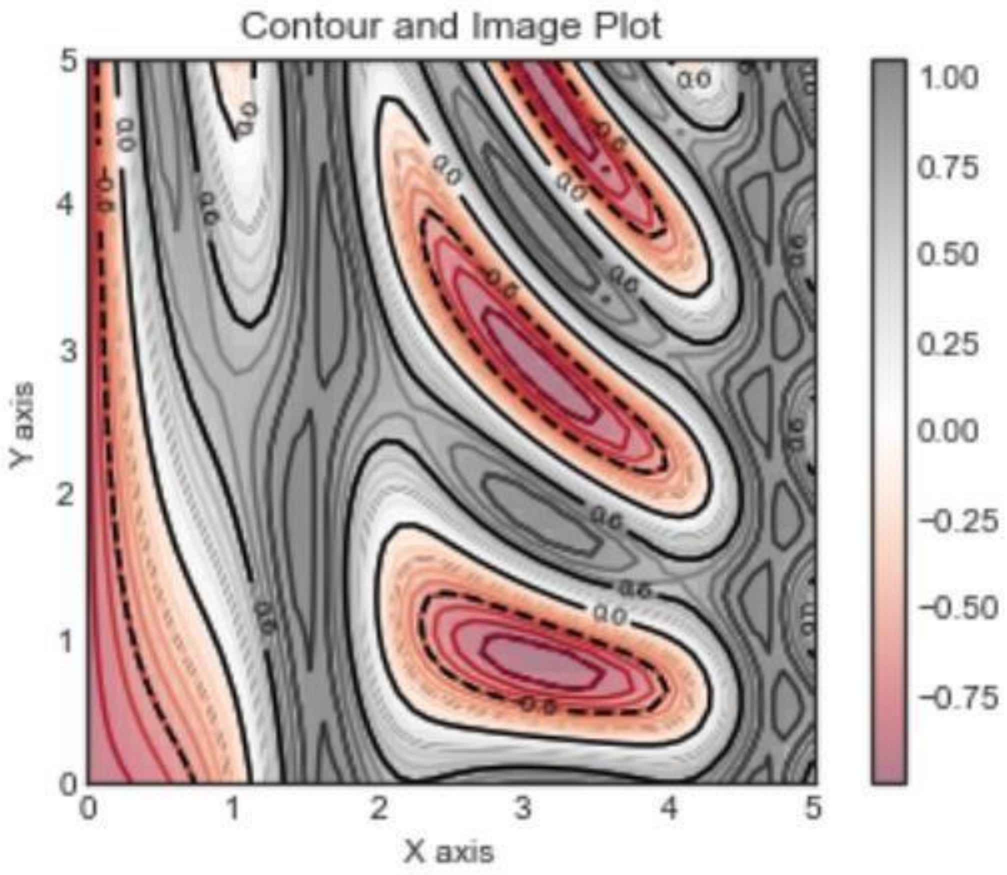
9. Add labeled contour lines using `plt.clabel` to provide information about the contour levels.

10. Display an image of the function using `plt.imshow` with transparency (alpha=0.5) and a specified colormap ('RdGy').

11. Add a color bar to the plot for reference using `plt.colorbar()`.

**PROGRAM:**

```
import numpy as np
import matplotlib.pyplot as plt

# Set the style for the plot
plt.style.use('seaborn-white')

# Define the function
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)

# Create grid values
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

# Create contour plots
plt.contour(X, Y, Z, colors='black')  # Basic contour plot
plt.contour(X, Y, Z, 20, cmap='RdGy')  # Contour plot with color map

# Add contour labels
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)

# Add an image plot of the data with color map
plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower', cmap='RdGy', alpha=0.5)

# Add color bar
plt.colorbar()

# Display the plot
plt.title('Contour and Image Plot')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```



**Result:**

The result of the code execution is a 2D contour plot with filled contours and an overlaid image of the function defined by `f(x, y)`. The contours provide a visual representation of the function's behavior, and the color bar helps to interpret the values associated with the colormap. The overall plot combines different elements to present a comprehensive view of the mathematical function in the specified range.

**c. Correlation and scatter plots**

## AIM:

The aim of the provided Python code is to analyze and visualize the relationship between the 'BloodPressure' and 'BMI' columns in a diabetes dataset using the Pandas, Seaborn, and SciPy libraries. It includes loading the dataset, displaying its headers, creating scatter plots, fitting a regression line, and calculating the correlation coefficient and correlation matrix.

## ALGORITHM:

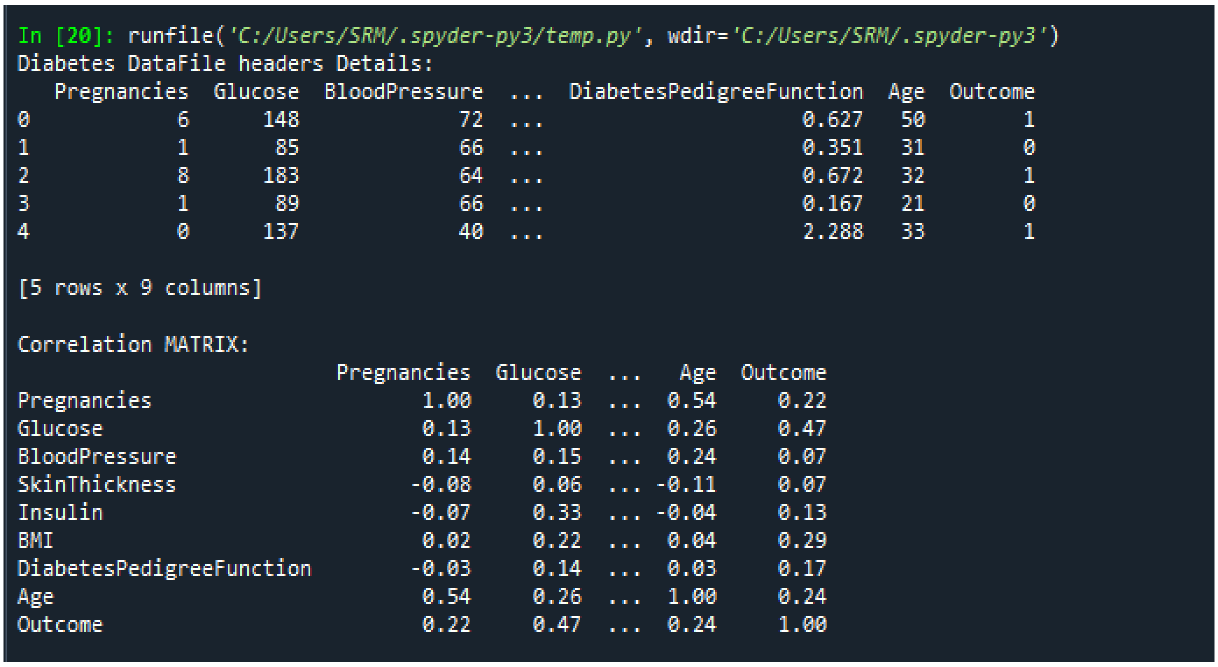
1. Import necessary libraries:
    - **pandas** for data manipulation and analysis.
    - **seaborn** for statistical data visualization.
    - **scipy.stats** for statistical functions.
2. Load the diabetes dataset from a CSV file using **pd.read_csv**.
3. Display the first few rows of the dataset using **print(diab.head())**.
4. Create a scatter plot using **sns.scatterplot** to visualize the relationship between 'BloodPressure' and 'BMI'.
5. Create another scatter plot with a regression line using **sns.lmplot** to show the linear relationship between 'BloodPressure' and 'BMI'.
6. Create a scatter plot with hue based on 'BloodPressure' using **sns.lmplot** to visualize the distribution of points across different 'BloodPressure' values.
7. Calculate and print the Pearson correlation coefficient between 'BloodPressure' and 'BMI' using **stats.pearsonr**.
8. Calculate and print the correlation matrix for all columns in the dataset using **diab.corr()**.
9. Visualize the correlation matrix using a heatmap with **sns.heatmap**.

## PROGRAM:

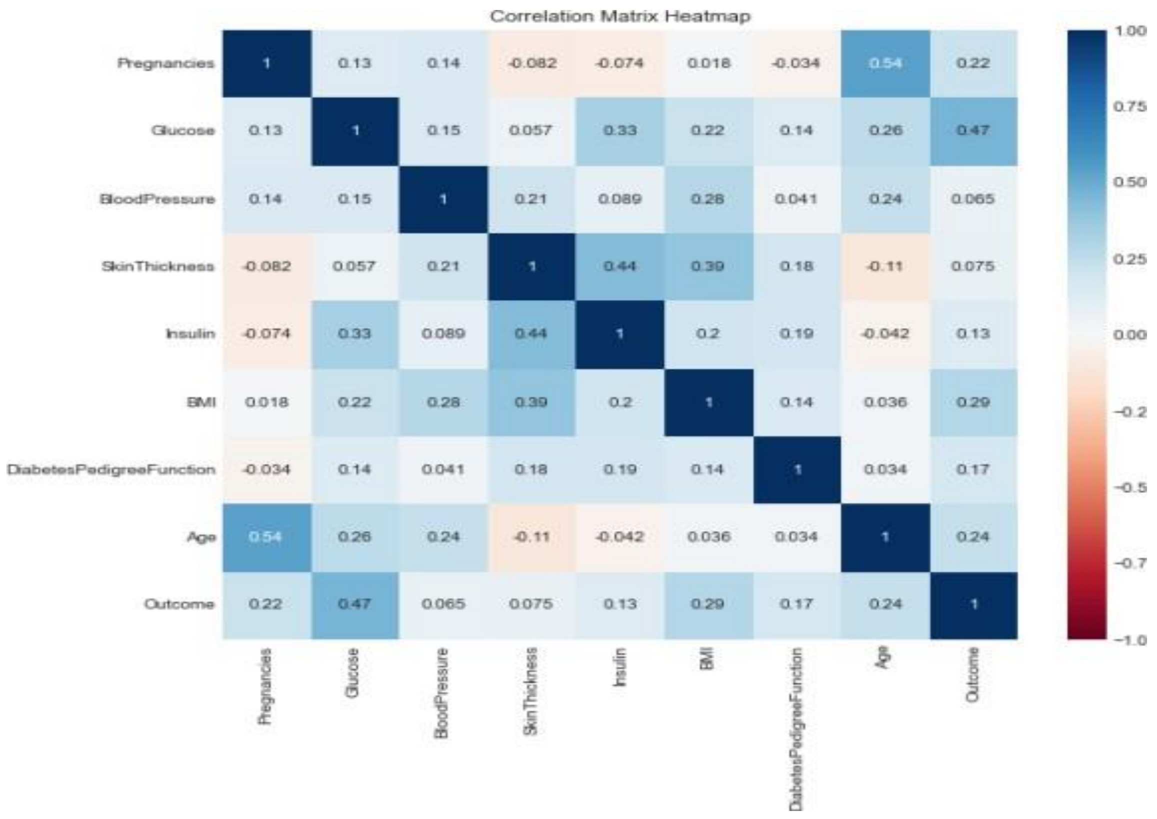
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Load the dataset
file_path = r'C:\Users\SRM\Downloads\diabetes.csv'
df = pd.read_csv(file_path)

# 1. Display the first few rows of the dataset
print("Diabetes DataFile headers Details:")
print(df.head())

# 2. Calculate and visualize the correlation matrix
cormat = df.corr()
```

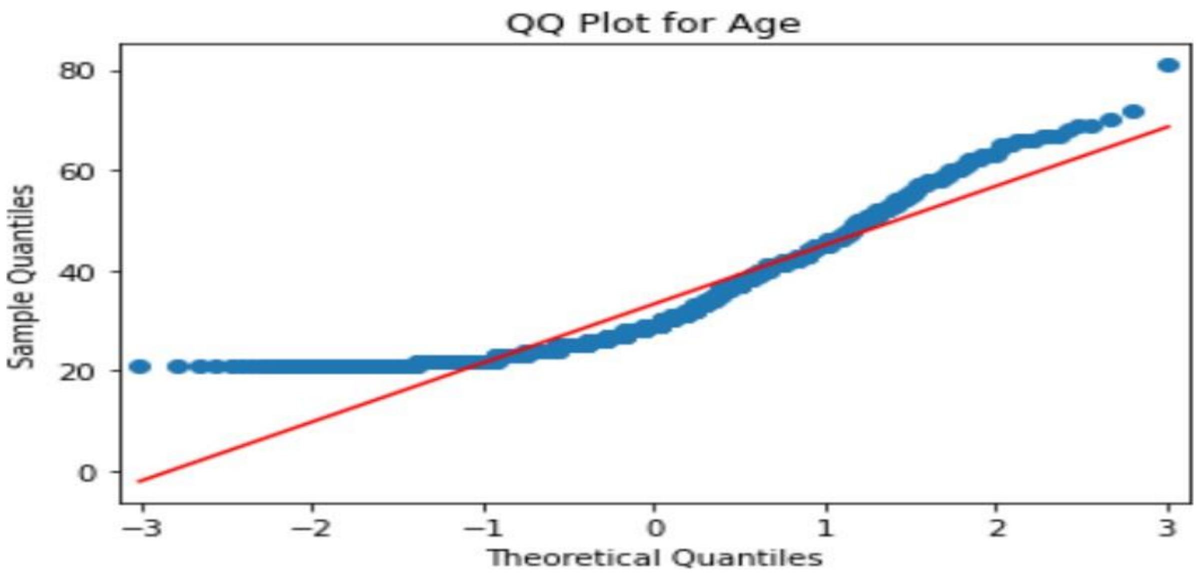
```
print("\nCorrelation MATRIX:")
print(round(cormat, 2))

# Heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cormat, annot=True, cmap='RdBu', vmin=-1, vmax=1)
plt.title("Correlation Matrix Heatmap")
plt.show()

# 3. Generate a QQ plot for the 'Age' variable
data = df['Age']
plt.figure(figsize=(8, 6))
sm.qqplot(data, line='s')
plt.title('QQ Plot for Age')
plt.show()

# 4. Produce scatter matrices
# Scatter matrix for the entire DataFrame
plt.figure(figsize=(15, 10))
pd.plotting.scatter_matrix(df, alpha=0.8, figsize=(15, 15), diagonal='kde')
plt.suptitle('Scatter Matrix for Diabetes Dataset')
plt.show()

# Scatter matrix for selected features
selected_features = df[["Pregnancies", "Glucose", "BloodPressure"]]
plt.figure(figsize=(10, 7))
pd.plotting.scatter_matrix(selected_features, alpha=0.8, figsize=(10, 7), diagonal='kde')
plt.suptitle('Scatter Matrix for Selected Features')
plt.show()
```

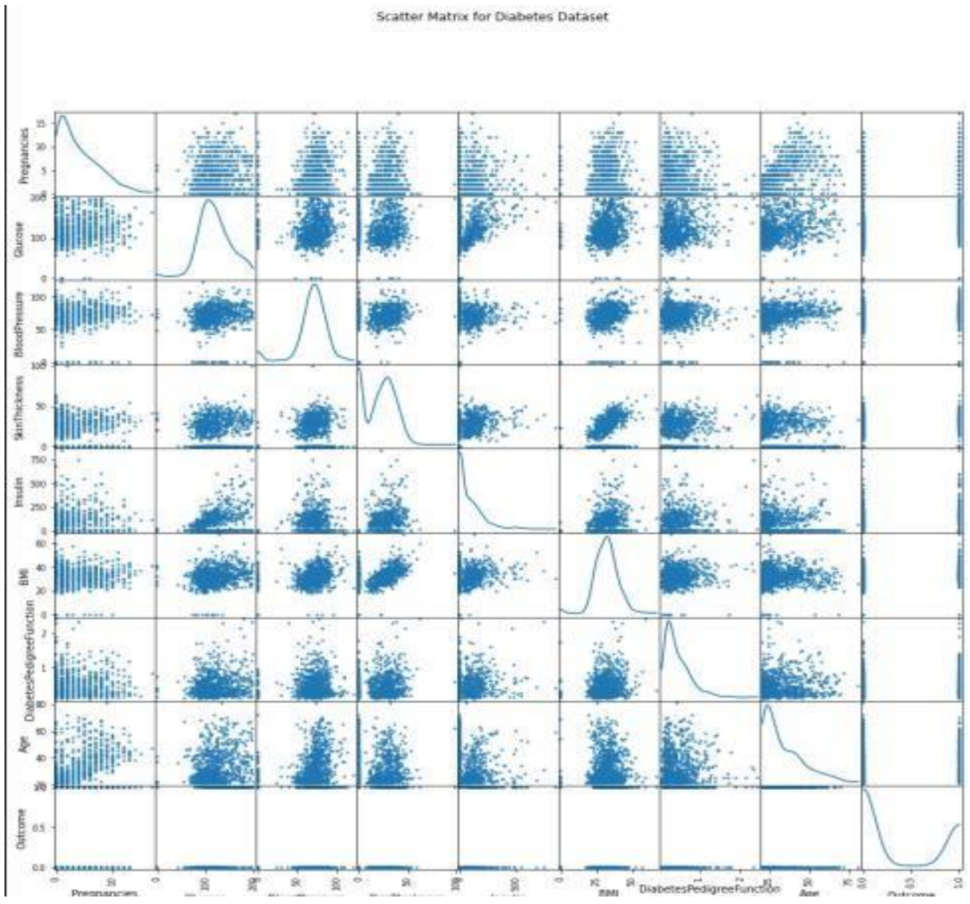
Output:

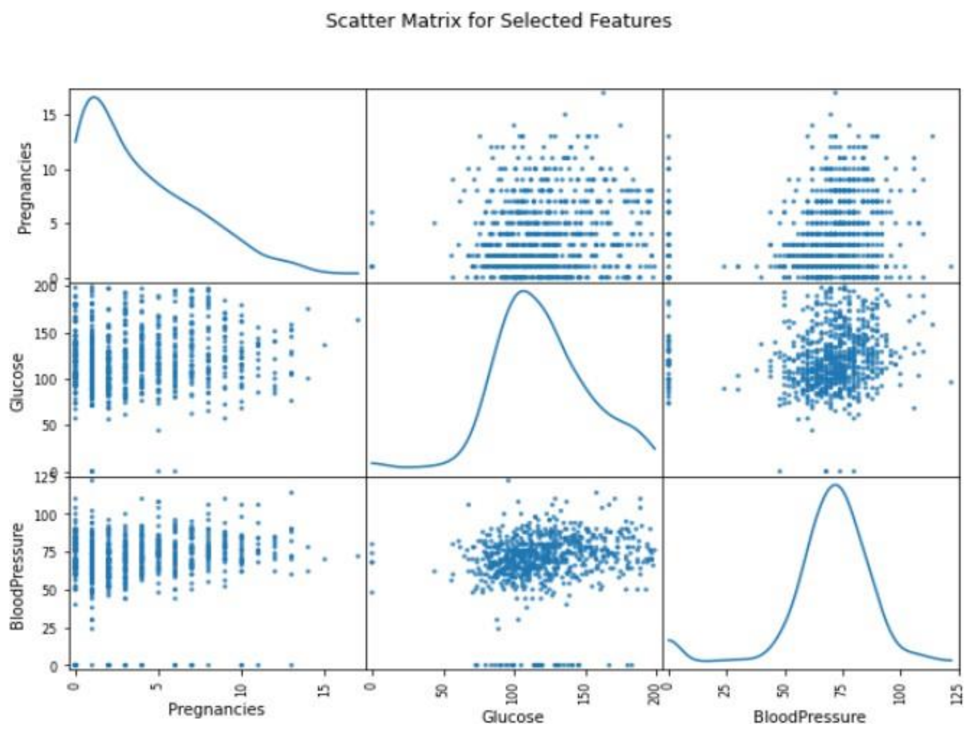


```
In [20]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/Users/SRM/.spyder-py3')
Diabetes DataFile headers Details:
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1

[5 rows x 9 columns]

Correlation MATRIX:
                          Pregnancies  Glucose  ...   Age  Outcome
Pregnancies                      1.00     0.13  ...  0.54     0.22
Glucose                          0.13     1.00  ...  0.26     0.47
BloodPressure                    0.14     0.15  ...  0.24     0.07
SkinThickness                   -0.08     0.06  ... -0.11     0.07
Insulin                         -0.07     0.33  ... -0.04     0.13
BMI                              0.02     0.22  ...  0.04     0.29
DiabetesPedigreeFunction        -0.03     0.14  ...  0.03     0.17
Age                              0.54     0.26  ...  1.00     0.24
Outcome                          0.22     0.47  ...  0.24     1.00
```




QQ Plot for Age
Sample Quantiles
Theoretical Quantiles
80
60
40
20
0
−3
−2
−1
0
1
2
3


Scatter Matrix for Diabetes Dataset
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome



**RESULT:**

The code generates visualizations and statistical measures for understanding the association between 'BloodPressure' and 'BMI' in the diabetes dataset. This includes scatter plots, regression lines, correlation coefficient, and a correlation matrix heatmap.

## Histograms

**AIM:**

The aim of the provided Python code is to create a histogram of a given dataset (`x`) and display its distribution.

## ALGORITHM:

1. Import the necessary library: `matplotlib.pyplot` for plotting.

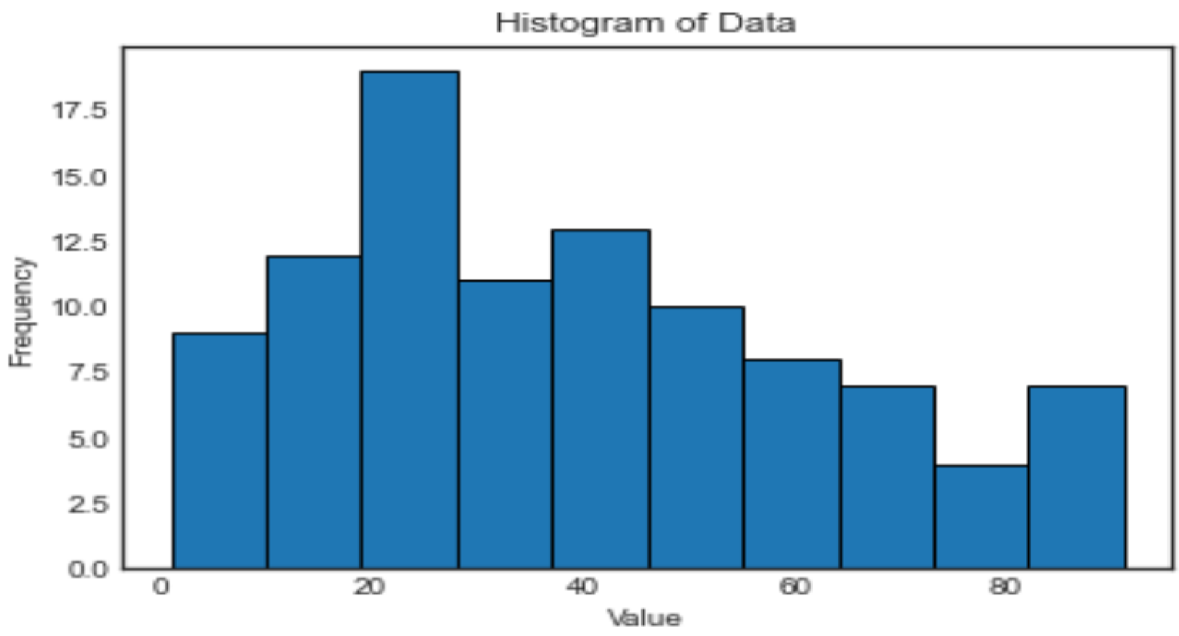
2. Define a list `x` containing numerical data.

3. Use `plt.hist(x, bins=10)` to create a histogram with 10 bins (intervals).

4. Display the histogram using `plt.show()`.

## PROGRAM:

```
import matplotlib.pyplot as plt
# Data
x = [1,1,2,3,3,5,7,8,9,10,
     10,11,11,13,13,15,16,17,18,18,
     18,19,20,21,21,23,24,24,25,25,
     25,25,26,26,26,27,27,27,27,27,
     29,30,30,31,33,34,34,34,35,36,
     36,37,37,38,38,39,40,41,41,42,
     43,44,45,45,46,47,48,48,49,50,
     51,52,53,54,55,55,56,57,58,60,
     61,63,64,65,66,68,70,71,72,74,
     75,77,81,83,84,87,89,90,90,91]

# Create the histogram
plt.hist(x, bins=10, edgecolor='black')

# Add title and labels
plt.title('Histogram of Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
# Display the plot
plt.show()
```



**RESULT:**

The result of the code execution is a histogram that visualizes the distribution of the data in the list `x`. The histogram is divided into 10 bins, providing insights into the frequency of values within each interval. The visualization allows for a quick understanding of the data's central tendency and spread.

## E. THREE DIMENSIONAL PLOTTING

### AIM:

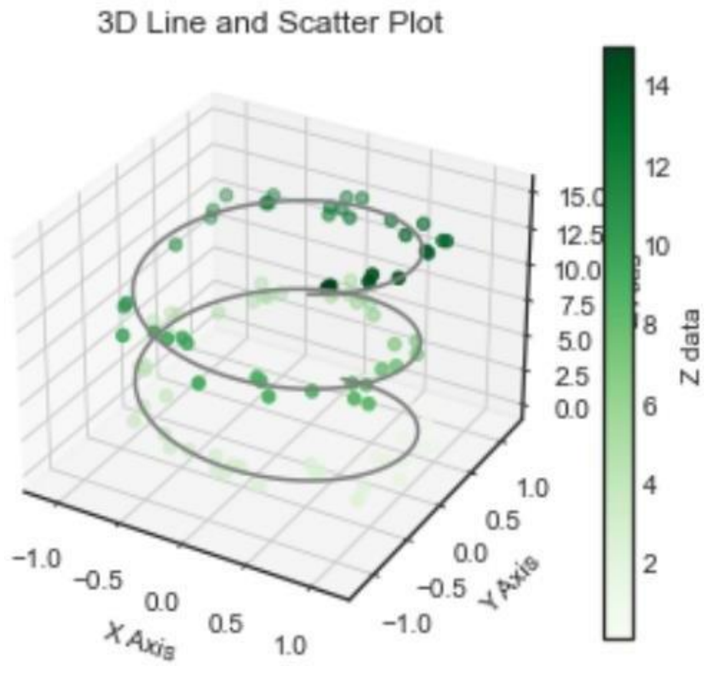
The aim of this provided Python code is to create a three-dimensional (3D) plot using Matplotlib. The code includes plotting a three-dimensional line and scattered points in a 3Dspace.

## ALGORITHM:

1. Import necessary libraries: `mpl_toolkits.mplot3d`, `numpy`, and
2. `matplotlib.pyplot`.
3. Create a figure and 3D axes using `plt.figure()` and `plt.axes(projection='3d')`.
   a. Generate data for a three-dimensional line: `zline`, `xline`, and `yline`.
   b. Plot the three-dimensional line using `ax.plot3D`.
   c. Generate data for three-dimensional scattered points: `zdata`, `xdata`, and `ydata`.
   d. Plot the scattered points using `ax.scatter3D`. The color of the points (`c`) is determined by the `zdata` values, and a colormap ('Greens') is applied for visual representation.

**PROGRAM:**

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create a new figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot(xline, yline, zline, color='gray')

# Data for three-dimensional scatter points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn(100)
ydata = np.cos(zdata) + 0.1 * np.random.randn(100)
sc = ax.scatter(xdata, ydata, zdata, c=zdata, cmap='Greens')

# Add color bar and labels
plt.colorbar(sc, ax=ax, label='Z data')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')
ax.set_title('3D Line and Scatter Plot')

# Show plot
plt.show()
```



3D Line and Scatter Plot
X Axis
Y Axis
Z data

**AUGMENTED QUESTIONS :**

**1. Write a Python program to perform the following tasks using the UCI dataset:**

- **Normal Curves:** Plot a normal distribution curve over a histogram for a numerical column, such as 'Glucose'. Calculate and display the mean and standard deviation used for plotting the normal curve.
- **Density and Contour Plots:** Create a density plot for the 'Glucose' and 'BMI' columns, and overlay a contour plot to visualize the density regions.

**2. Write a Python program to create the following visualizations using the UCI dataset:**

- **Correlation Plot:** Generate a heatmap showing the correlation matrix of all numerical columns in the dataset.
- **Scatter Plot Matrix:** Create a scatter plot matrix for a subset of columns (e.g., 'Glucose', 'BMI', 'Age') to explore pairwise relationships.
- **Three-Dimensional Plot:** Plot a 3D scatter plot using 'Glucose', 'BMI', and 'Age' as the three dimensions.

**VIVA QUESTIONS**:

1. How would you plot a normal distribution curve for a numerical column in the UCI dataset? Which Python libraries and functions can you use for this?
2. Explain how you can create a density plot and a contour plot for two numerical columns in the UCI dataset. What insights can these plots provide?
3. Describe the process for generating a correlation plot and scatter plot between two features in the UCI dataset. How do these plots help in understanding the relationship between features?
4. What steps would you take to create a histogram of a numerical feature in the UCI dataset? What information can be derived from a histogram?
5. How can you create a three-dimensional plot using three numerical features from the UCI dataset? Which Python functions are used for 3D plotting and what do these plots represent?

## RESULT:

The code generates a 3D plot with a three-dimensional line and scattered points. The line is defined by the functions `np.sin` and `np.cos`, and the scattered points have random coordinates influenced by sine and cosine functions. The color of the scattered points varies based on the `zdata` values, creating a visually appealing representation of the data in 3D space

| EX.NO:7 | VISUALIZING GEOGRAPHIC DATA WITH BASEMAP |
|---|---|

## AIM:

The goal of the provided code is to visualize geographic data using the Basemap toolkitin Matplotlib. Specifically, it creates maps with different projections, includes topographic features, and marks the location of Seattle.

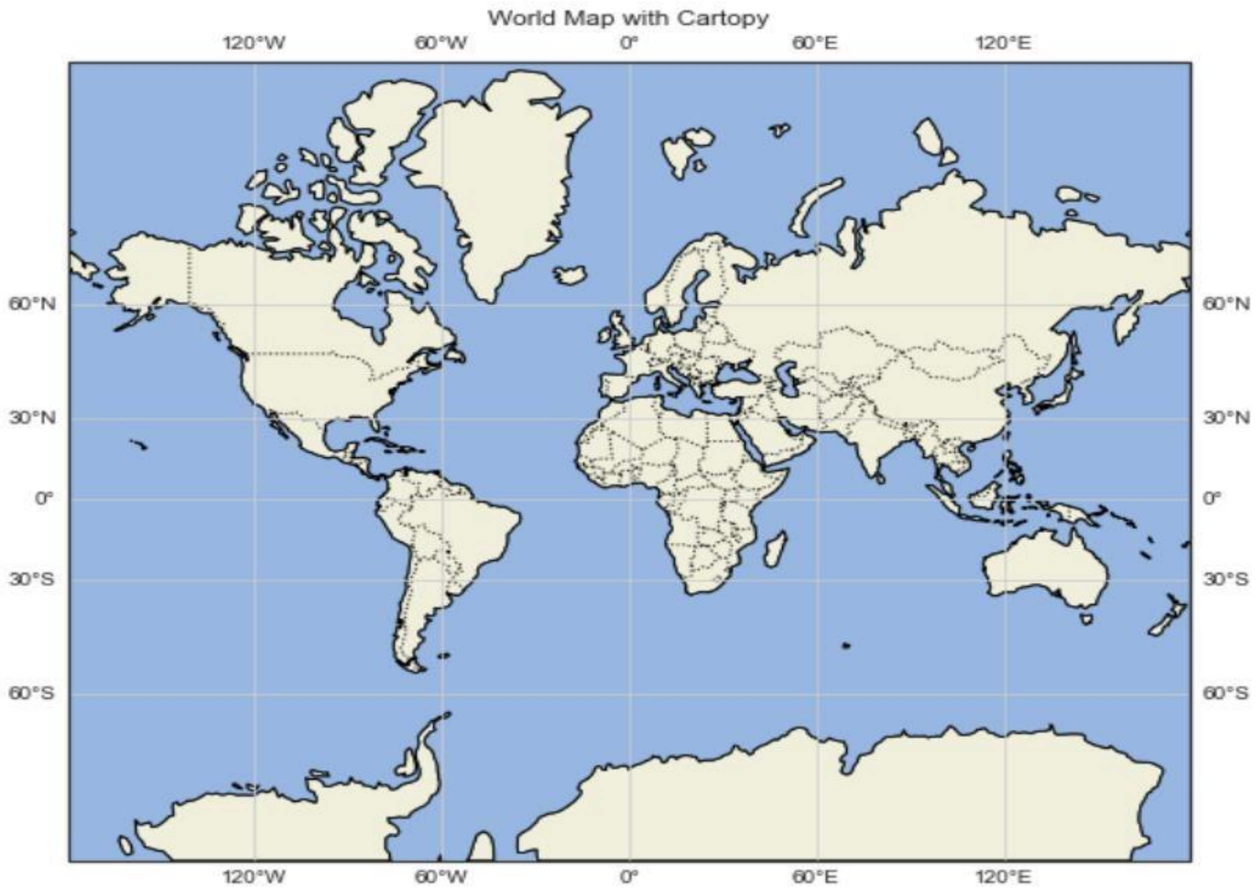
**ALGORITHM:**

1. Set up a Matplotlib figure and create a Basemap with Lambert conformal conic projection, specifying parameters like width, height, central latitude, and longitude.
2. Overlay topographic features using the `etopo` method and add a point on the map corresponding to Seattle.
3. Define a function `draw_map` to draw shaded-relief images and latitude/longitude lines with specified styles.
4. Generate three different maps with varying projections: cylindrical projection covering the entire world, repeated cylindrical projection, and Lambert conformal conic projection focused on a specific region.
5. Display the maps using Matplotlib.

## PROGRAM:

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

# Create a new figure
fig = plt.figure(figsize=(12, 8))

# Set up the Cartopy projection
ax = fig.add_subplot(111, projection=ccrs.Mercator())

# Add geographic features
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS, linestyle=':')
ax.add_feature(cfeature.LAND, edgecolor='black')
ax.add_feature(cfeature.OCEAN)

# Add gridlines
ax.gridlines(draw_labels=True)

# Set title
plt.title('World Map with Cartopy')

# Show the plot
plt.show()
```

**OUTPUT:**



**AUGMENTED QUESTIONS :**

1. Write a Python program using Basemap to create an interactive map that displays the locations of major cities around the world. Include functionality to zoom in and out, and add labels to each city. How would you integrate Basemap with other libraries to enhance interactivity?

2. Develop a Python script to visualize global climate data using Basemap. Create a map that displays temperature anomalies with a color gradient. Integrate Basemap with data from a CSV file containing latitude, longitude, and temperature anomaly values. How would you handle large datasets and ensure efficient rendering of the map?

**VIVA QUESTIONS**:

1. What is the Basemap toolkit, and how is it used for visualizing geographic data in Python?
2. How would you plot a simple map of a specific region or country using Basemap? What are the basic steps involved in creating such a map?
3. Explain how to overlay markers or data points on a Basemap. What functions or methods are used to add these elements to the map?
4. How can you display geographic data such as country borders, rivers, or cities on a map using Basemap? What are some common map features you can add?
5. Describe how to customize the appearance of a map created with Basemap, such as changing the map's projection, adding gridlines, or adjusting the map's color scheme.

**RESULT:**

The code produces geographic visualizations, including a Lambert conformal conic projection with topographic features, a cylindrical projection covering the entire world, and a repeated cylindrical projection. The resulting maps showcase the versatility of the Basemap toolkit for visualizing geographic data in Matplotlib.