P.T.Lee Chengalvaraya Naicker College of Engineering & Technology

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)

Vallal P.T.Lee Chengalvaraya Naicker Nagar, Oovery, Kanchipuram – 631 502

Department of Computer Science and Engineering

Regulation – 2021

CS3361 – Data Science Laboratory

Record

Name :

Reg. No :

Year/ Semester :

P.T.Lee Chengalvaraya Naicker College of Engineering & Technology

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)
Vallal P.T.Lee Chengalvaraya Naicker Nagar, Oovery,
Kanchipuram – 631 502.

Department of Computer Science and Engineering

BONAFIDE CERTIFICATE

RegNo: _____

Certified that this is a bonafide record of work done by the candidate Mr\Mrs._____ in **CS3361- Data Science Laboratory** during the academic year _____ .

Lab-in-charge Head of the Department

Submitted for the Practical Examination held on _____.

Internal Examiner External Examiner

INDEX

Ex. No	Date	Title	Page No	Mark	Sign
1		Download, install and explore the features of Numpy, Scipy, Pandas, Jupyter and Statsmodels packages.			
2		Working with Numpy Arrays			
3		Working with Pandas DataFrame			
4		Descriptive Analysis on Iris Dataset			
5a		Univariate Analysis using the UCI diabetes data set			
5b		Bivariate Analysis using the UCI diabetes data set			
5c		Multiple Regression Analysis using the UCI diabetes data set			
6		Apply and explore various plotting functions on UCI data sets			
7		Visualizing Geographic Data with Basemap			

Ex. No.: 1	Download, install and explore the features of Numpy, Scipy, Pandas, Jupyter and Statsmodels packages

Aim:

To download, install the anaconda software and to explore the features of Numpy, Scipy, Jupyter, statsmodels and pandas packages.

Description:

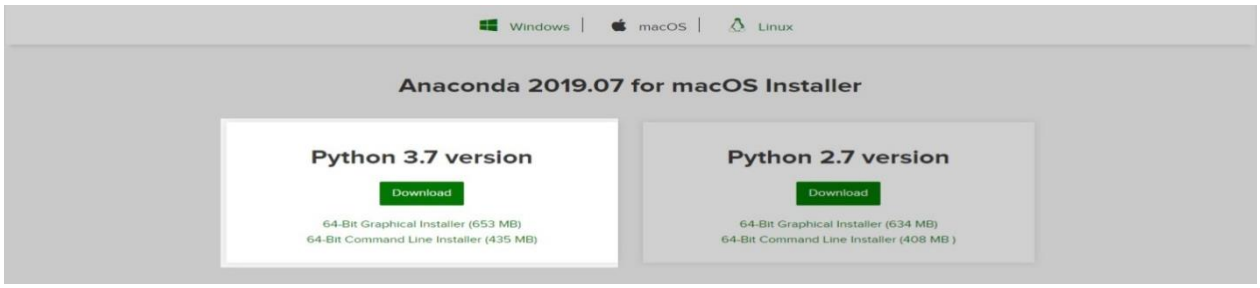
Python is an open-source, object-oriented and cross-platform programming language. Compared to programming languages like C++ or Java, Python is very concise. It allows us to build a working software prototype in a very short time. It has become the most used language in the data scientist's toolbox. It is also a general-purpose language and it is very flexible due to variety of available packages that solve a wide spectrum of problems and necessities. To install the necessary packages use 'pip'.

Anaconda(http://continuum.io/downloads) is a Python distribution offered by Continuum Analytics that includes nearly 200 packages, which comprises Numpy, Scipy, Pandas, Jupyter, Matplotlib, Scikit-learn and NLTK. It is a cross platform distribution (Windows, Linux and Mac OS X) that can be installed on machines with other existing Python distributions and versions. Its base version is free; instead, add-ons that contain advanced features are charged separately. Anaconda introduces 'conda', a binary package manager, as a command-line tool to manage package installations. Anaconda's goal is to provide enterprise-ready Python distribution for large-scale processing, predictive analytics and scientific computing.

Installation Steps for Anaconda:

Step #1: Go to Anaconda.com, and download the Anaconda version for Windows.

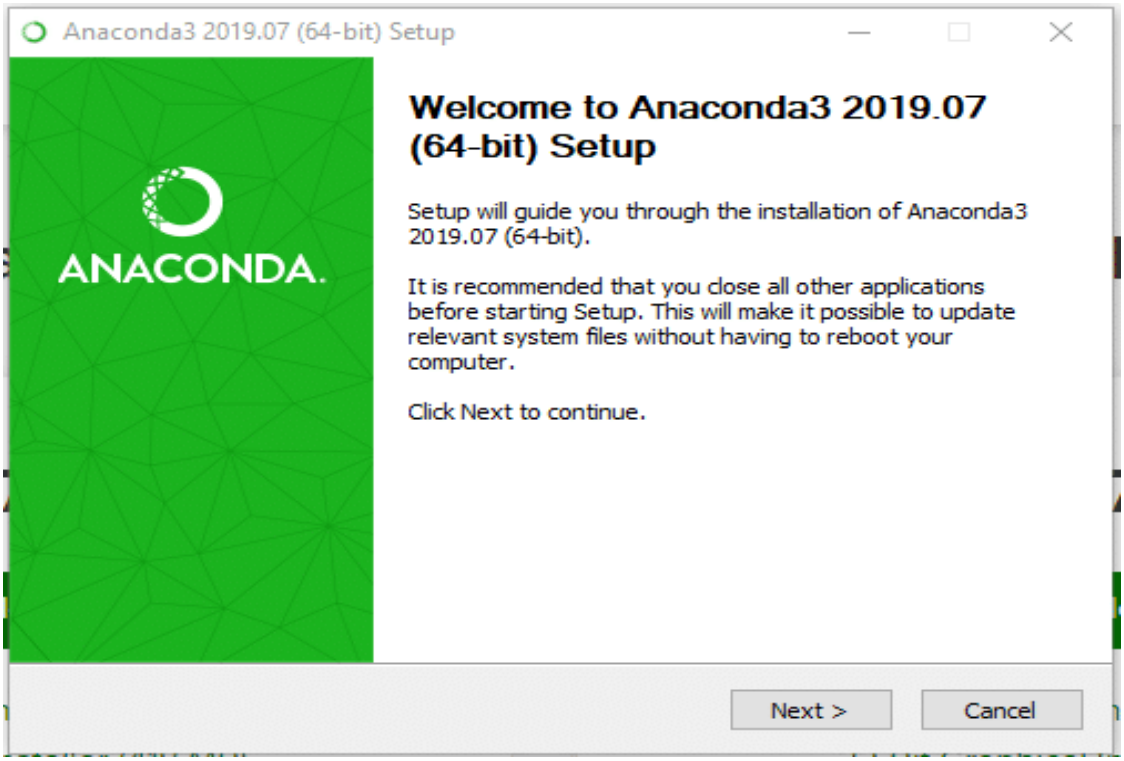
Step #2: Download the Python 3 version for Windows.



Step #3: Double-click on the executable file.

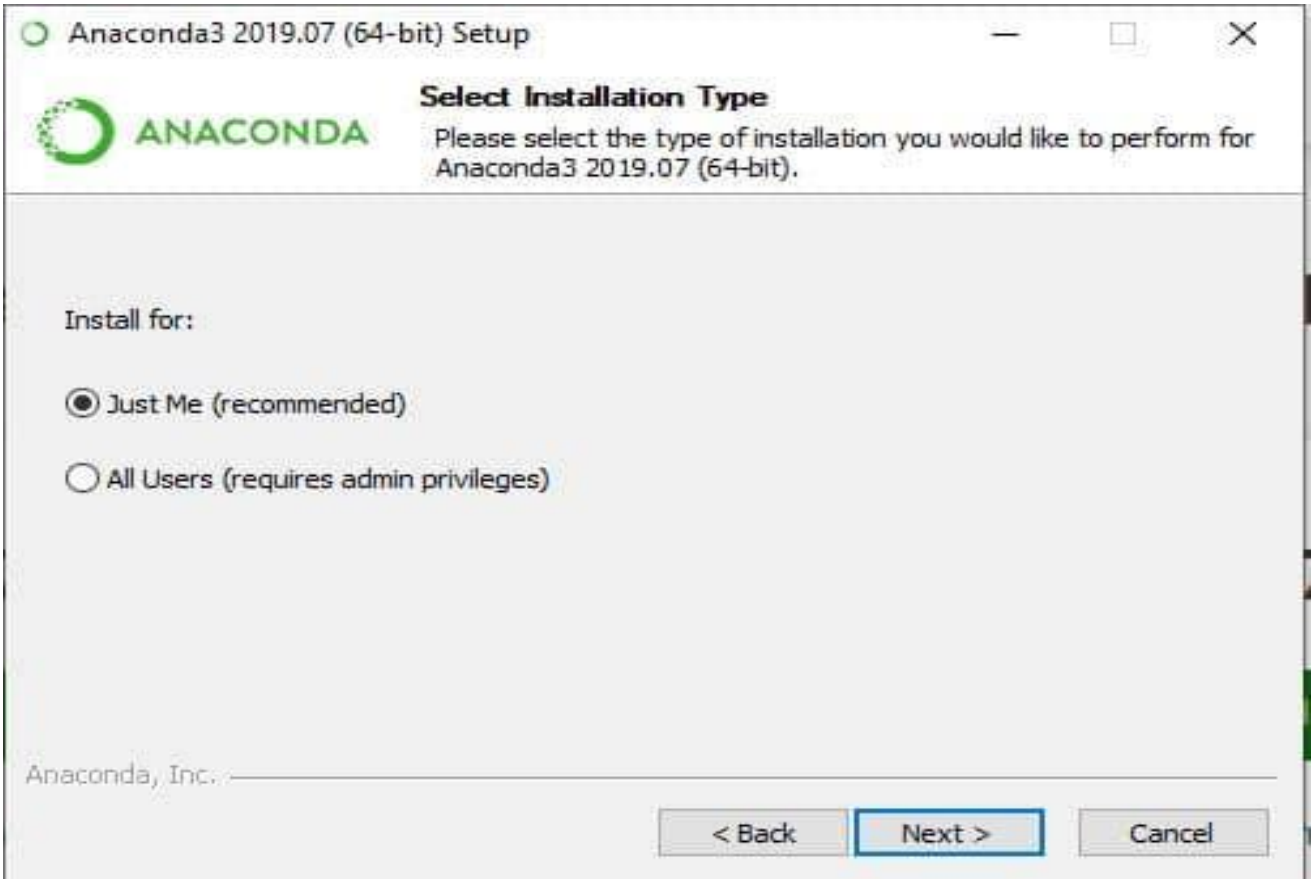
To get the installation of Anaconda started on operating system open the executable file in Download folder.

Step #4: Click Next



Step #5: Click I agree to the terms and conditions.

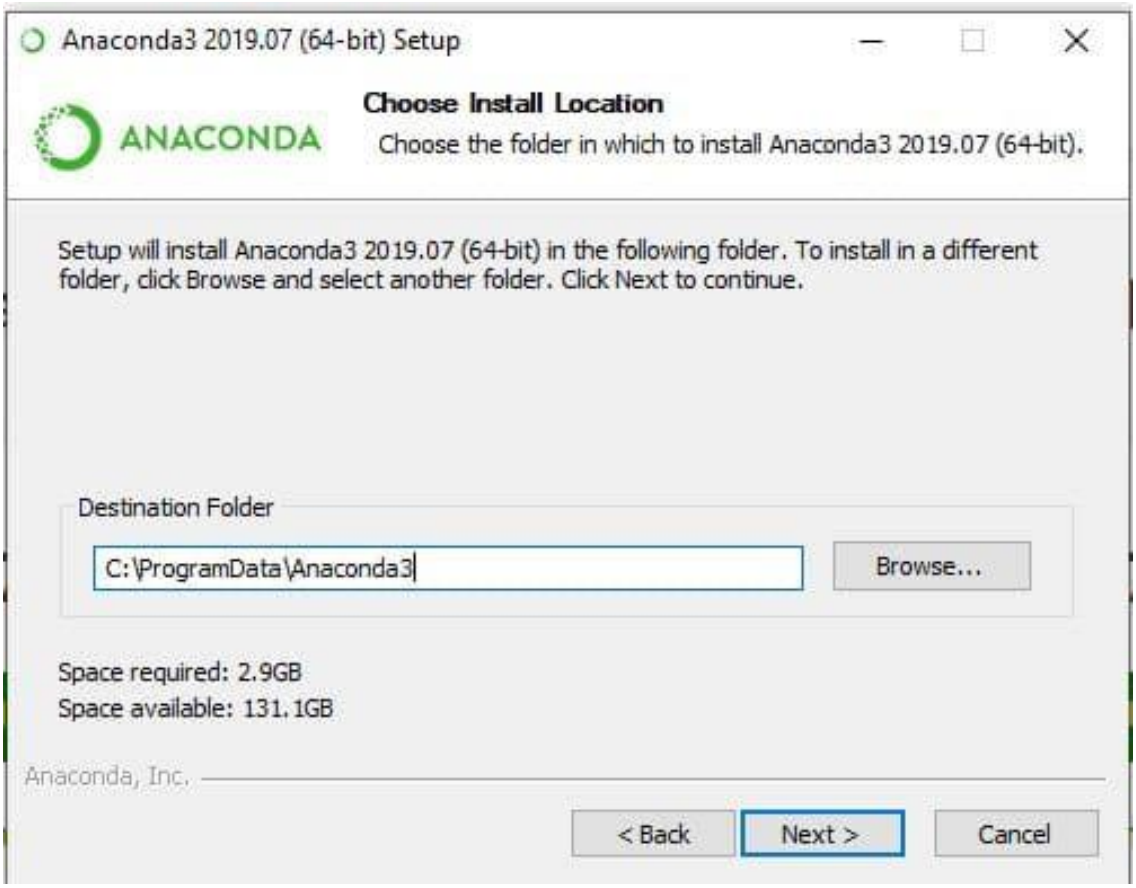
Step #6: Select who you want to give Anaconda to



This step will ask you if you want to install Anaconda just for you or for all the users using this PC. Click “Just-Me”, or “All users”, depending on your preference. Both options will do but to select “all users” you will need admin privileges.

Step #7: Select the installation location

If you have selected “All users”, by default, Anaconda will get installed in the *C:\ProgramData\Anaconda3* folder. So make sure that you have at least the right amount of space available to install the subdirectory comparing it the space required.



Step #8: Select the environment variables

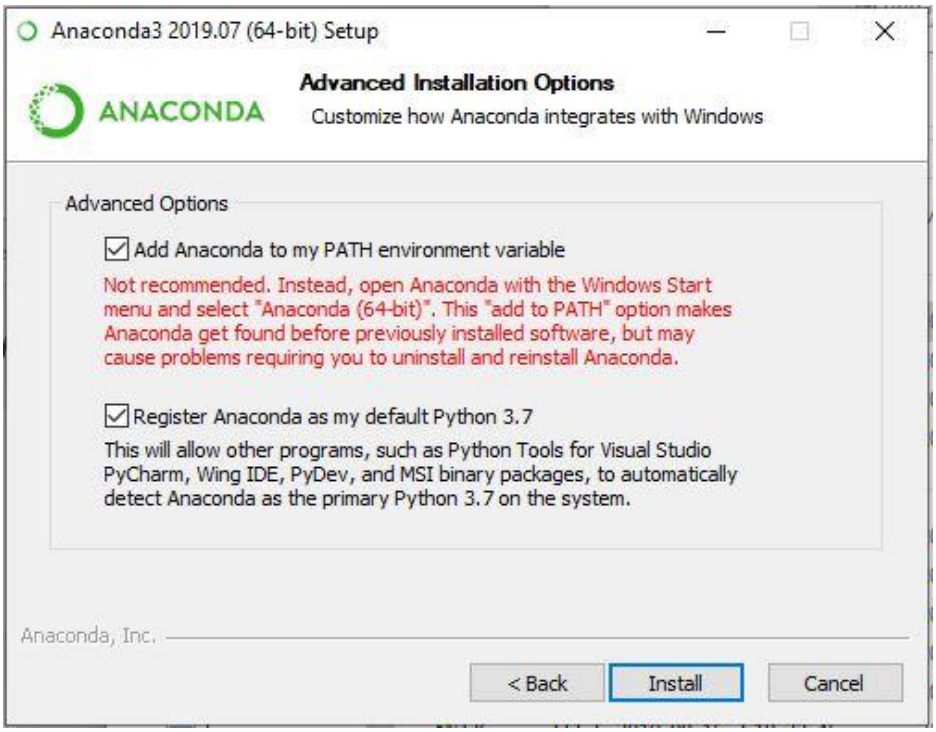
Depending on if you have any version of Python already installed on your operating system, or not, to do different set-up.

If You Are Installing Python For The First Time

Check the *Add Anaconda to my PATH environment variable.* This will let you use Anaconda in your command prompt.

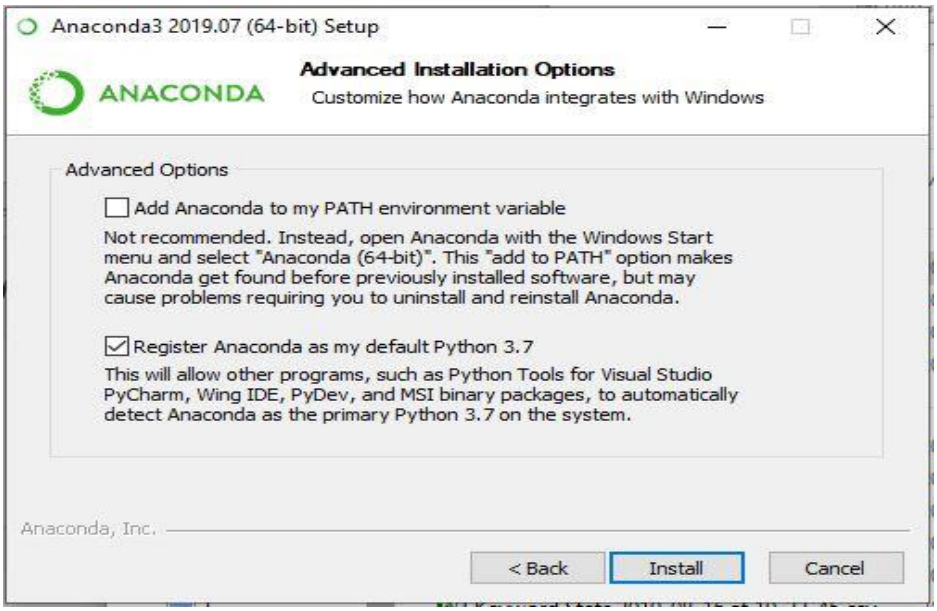
If You Already Have Python Installed

Leave *Add Anaconda to my PATH environment variable* unchecked.



Leaving it unchecked means that you will have to use Anaconda Command Prompt in order to use Anaconda.

So, unless you add the PATH later, you will not be able to use Python from your command prompt.



Python is not usually included by default on Windows, however we can check if any version exists on the system.

To know if you have Python Installed.

1. Go to Start Menu and type "Command Prompt" to open it.
2. Type the following command and hit the Enter key "python --version"
3. If nothing happens, you don't have Python installed. Otherwise, you will get this result.

$ python --version
Python 3.7.0

Step #9: Click Next and then "Finish".

Features of Python Packages:

1. Numpy

Numpy stands for Numerical Python. Numpy is an open-source library for the python programming language. It is used for scientific computing and working with arrays. The source code for Numpy is located at this github repository https://github.com/numpy/numpy.

Features:

1. High-performance N-dimensional array object.
2. It contains tools for integrating code from c/c++ and Fortran.
3. It contains a multidimensional container for generic data.
4. Additional linear algebra, Fourier transform and random number capabilities.
5. It consists of broadcasting functions.
6. It has data type capability to work with varied databases.

2. Scipy

Scipy stands for Scientific python. Scipy is a scientific computation library that uses Numpy underneath. The source code for Scipy is located at this github repository https://github.com/scipy/scipy.

Features:

1. Scipy provides algorithm for optimization, integration, interpolation, eigenvalue problems, algebraic equation, differential equations, statistics and many other classes of

problems.

2. It provides more utility functions for optimization, stats and signal processing.

Numpy Vs Scipy

Numpy and Scipy both are used for mathematical and numerical analysis. Numpy is suitable for basic operations such as sorting, indexing and many more because it contains array data, wherasscipy consists of all numeric data.

Numpy contains many functions that are used to resolve the linear algebra, Fourier Transforms, etc., whereas Scipy contains full featured version of the linear algebra module as well many other numerical algorithms.

3. Pandas

Python pandas is defined as an open-source library that provides high performance data manipulation in python. The name of pandas is derived from the word Panel Data, which means an Econometrics from Multidimensional data. It is used for data analysis in Python. Pandas is built on top of the numpy package, means numpy is required for operating pandas.

Features:

1. Group by data for aggregations and transformations.
2. It has a fast and efficient DataFrame object with the default and customizing indexing.
3. Used for reshaping and pivoting of data sets.
4. It is used for data alignment and integration of the missing data.
5.Provides the functionality of time series.
6. Process a variety of data sets in different formats like matrix data, tabular heterogeneous, time series.
7. Handle multiple operations of the data sets such as subsetting, slicing, filtering, groupby, re-ordering and re-shaping.

4. Statsmodels

Statsmodels is a python module that provides classes and functions for the estimation of

many different statistical methods as well as for conducting statistical tests and statistical data explorations. The package is released under the open source Modified BSD(3-clause) license. The online documentation is hosted at statsmodels.org.

Features:

1. Linear regression models like Ordinary least squares, Generalized least squares, Weighted least squares, Least squares with autoregressive errors.

2. Bayesian Mixed GLM for Binomial and Poisson

3. GEE: Generalized Estimating Equations for one-way clustered or longitudinal data

4. Nonparametric statistics: Univariate and Bivariate kernel density estimators

5. Datasets: Datasets used for examples and in testing

6. Sandbox: Statsmodels contains a Sandbox folder with code in various stages of development and testing.

5. Jupyter

A scientific approach requires the fast experimentation of different hypotheses in a reproductible fashion. Initially named IPython and limited to working only with the python language, Jupyter was created to address the need for an interactive command shell for several languages (based on the shell, web browser and application interface), featuring graphical integration, customizable commands, rich history (in the JSON format) and computational parallelism for enhanced performance.

Steps to install Jupyter using Anaconda

- Launch Anaconda Navigator
- Click on the Install Jupyter Notebook Button

Result:

Thus the anaconda software has been successfully installed and features of Numpy, Scipy, Jupyter, statsmodels and pandas packages has been explored.

Ex. No.: 2	Working with Numpy Arrays

Aim:

To write a Numpy arrays program to demonstrate the basic array concepts in Jupyter Notebook.

Algorithm:

Step 1: Start the program.

Step 2: Import the Numpy library.

Step 3: Define the 1-D array, 2-D array, 3-D array.

Step 4: Print the memory address, the shape, the datatype and perform some basic operations.

Step 5: Then create an array using built-in Numpy functions and perform some mathematical functions.

Step 6: Print the output.

Step 7: Stop the program.

Program:

1. NumPy program to print the NumPy version in your system.

Code:

```
import numpy as np
print(np.__version__)
```

Output:

1.21.5

2. Numpy program to print different types of arrays.

```
#one-dimensional array
```

Code:

```
import numpy as np
x=np.arange(2,6).reshape(4)
print(x)
```

Output:

```
[2, 3, 4, 5]
#two-dimensional array
```

Code:

```
import numpy as np
x=np.arange(2,10).reshape(2,4)
print(x)
```

Output:

```
[[2 3 4 5]
 [6 7 8 9]]
#three-dimensional array
```

Code:

```
import numpy as np
x=np.arange(24).reshape(4,3,2)
print(x)
```

Output:

```
[[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]

 [[12 13]
  [14 15]
  [16 17]]

 [[18 19]
  [20 21]
  [22 23]]]
```

3. Numpy program to perform Array slicing.

Code:

```
import numpy as np
arr=np.array([1,2,3,4])
print(arr[1:3:2])
print(arr[:3])
print(arr[::2])
```

Output:

```
[2]
[1 2 3]
[1 3]
```

4.Numpy program to perform trigonometric functions.

Code:

```
import numpy as np
arr=np.array([0,30,60,90])
print(np.sin(arr))
print(np.cos(arr))
print(np.tan(arr))
```

Output:

```
[ 0. -0.98803162 -0.30481062 0.89399666]
[ 1. 0.15425145 -0.95241298 -0.44807362]
[ 0. -6.4053312 0.32004039 -1.99520041]
```

5.NumPy program to convert a list of numeric value into a one-dimensional NumPy array.

Code:

```
import numpy as np
l = [12.23, 13.32, 100, 36.32]
```

```
print("Original List:",l)
a = np.array(l)
print("One-dimensional NumPy array: ",a)
```

Output:

```
Original List: [12.23, 13.32, 100, 36.32]
One-dimensional NumPy array: [ 12.23 13.32 100. 36.32]
```

6. NumPy program to create an array with values ranging from 12 to 38.

Code:

```
import numpy as np
x = np.arange(12, 38)
print(x)
```

Output:

```
[12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37]
```

7.NumPy program to reverse an array (first element becomes last).

Code:

```
import numpy as np
import numpy as np
x = np.arange(12, 38)
print("Original array:")
print(x)
print("Reverse array:")
x = x[::-1]
print(x)
```

Output:

```
Original array:
[12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37]
Reverse array:
 [37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12]
```

8. NumPy program to append values to the end of an array.

Code:

```
import numpy as np
x = [10, 20, 30]
print("Original array:")
print(x)
x = np.append(x, [[40, 50, 60], [70, 80, 90]])
print("After append values to the end of the array:")
print(x)
```

Output:

```
Original array:
[10, 20, 30]
After append values to the end of the array:
[10 20 30 40 50 60 70 80 90]
```

9.NumPy program to find common values between two arrays.

Code:

```
import numpy as np
array1 = np.array([0, 10, 20, 40, 60])
print("Array1: ",array1)
array2 = [10, 30, 40]
print("Array2: ",array2)
print("Common values between two arrays:")
print(np.intersect1d(array1, array2))
```

Output:

```
Array1: [ 0 10 20 40 60]
Array2: [10, 30, 40]
Common values between two arrays: [10 40]
```

10. NumPy program to find the union of two arrays.

Code:

```
import numpy as np
array1 = np.array([0, 10, 20, 40, 60, 80])
print("Array1: ",array1)
array2 = [10, 30, 40, 50, 70]
print("Array2: ",array2)
print("Unique sorted array of values that are in either of the two input arrays:")
print(np.union1d(array1, array2))
```

Output:

Array1: [0 10 20 40 60 80]

Array2: [10, 30, 40, 50, 70]

Unique sorted array of values that are in either of the two input arrays: [0 10 20 30 40 50 60 70 80]

11. NumPy program to concatenate two 2-dimensional arrays.

Code:

```
import numpy as np
a = np.array([[0, 1, 3], [5, 7, 9]])
b = np.array([[0, 2, 4], [6, 8, 10]])
c = np.concatenate((a, b), 1)
print(c)
```

Output:

```
[[ 0 1 3 0 2 4]
 [ 5 7 9 6 8 10]]
```

12.Write a NumPy program to convert the values of Centigrade degrees into Fahrenheit degrees and vice versa.

#Fahrenheit degree to Centigrade degree.

Code:

```
import numpy as np
fvalues = [0, 12, 45.21, 34, 99.91, 32]
F = np.array(fvalues)
print("Values in Fahrenheit degrees:")
print(F)
print("Values in  Centigrade degrees:")
print(np.round((5*F/9 - 5*32/9),2))
```

Output:

```
Values in Fahrenheit degrees:
[ 0. 12. 45.21 34. 99.91 32. ]
Values in Centigrade degrees:
[-17.78 -11.11 7.34 1.11 37.73 0. ]
```

#Centigrade degree to Fahrenheit degree.

Code:

```
import numpy as np
cvalues = [-17.78, -11.11, 7.34, 1.11, 37.73, 0]
C = np.array(cvalues)
print("Values in Centigrade degrees:")
print(C)
print("Values in  Fahrenheit degrees:")
print(np.round((9*C/5 + 32),2))
```

Output:

```
Values in Centigrade degrees:
[-17.78 -11.11 7.34 1.11 37.73 0. ]
Values in Fahrenheit degrees:
[-0. 12. 45.21 34. 99.91 32. ]
```

13.Numpy Program to perform arithmetic operations.

Code:

```
import numpy as np
```

```
a=np.arange(9,dtype=np.float_).reshape(3,3)
print("First Array:")
print(a)
b=np.array([10,20,30])
print("Second Array:")
print(b)
print("Addition of Arrays:")
print(np.add(a,b))
print("Subtraction of Arrays:")
print(np.subtract(a,b))
print("Multiplication of Arrays:")
print(np.multiply(a,b))
print("Division of Arrays:")
print(np.divide(a,b))
```

Output:

```
First Array:
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
Second Array:
[10 20 30]
Addition of Arrays:
[[10. 21. 32.]
 [13. 24. 35.]
 [16. 27. 38.]]
Subtraction of Arrays:
[[-10. -19. -28.]
 [ -7. -16. -25.]
 [ -4. -13. -22.]]
Multiplication of Arrays:
```

```
[[ 0.  20.  60. ]
 [ 30. 80.  150.]
 [ 60. 140. 240.]]
Division of Arrays:
[[0.  0.05 0.06666667]
 [0.3 0.2  0.16666667]
 [0.6 0.35 0.26666667]]
```

Result:

Thus the Numpy array programs was successfully executed and verified.

Ex. No.: 3	Working of Pandas DataFrame

Aim:

To write a Pandas program usingsample dataframe to perform operations in its dataframe.

Algorithm:

Step 1: Open an IDE such as Jupyter Notebook.

Step 2: Import pandas packages.

Step 3: Create a DataFrame using pandas and extract details of the dataset such as number of datas in each columns, their data types, memory usage, etc…

Step 4: Perform operations on rows and columns and renaming them.

Step 5: Selecting, Adding and Deleting datas on the dataset.

Step 6: Handle the missing datas on the DataFrame.

Sample DataFrame:

```
exam_data={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Matthew','Laura','Kevin','Jonas'],
'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
'attempts':[1,3,2,3,2,3,1,1,2,1],
'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels =['a','b','c','d','e','f','g','h','i','j']
```

Program:

1.Pandas program to create and display a DataFrame from a specified dictionary data which has the index labels.

Code:

```
import pandas as pd
import numpy as np
exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
    'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
    'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df = pd.DataFrame(exam_data , index=labels)
print(df)
```

Output:

```
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

2. Pandas program to display a summary of the basic information about a specified DataFrame and its data.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("Summary of the basic information about this DataFrame and its data:")
print(df.info())
```

Output:

```
Summary of the basic information about this DataFrame and its data:
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   name      10 non-null     object
 1   score     8 non-null      float64
 2   attempts  10 non-null     int64
 3   qualify   10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
None
```

3. Pandas program to get the first 3 rows of a given DataFrame.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("First three rows of the data frame:")
print(df.iloc[:3])
```

Output:

```
First three rows of the data frame:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
```

4. Pandas program to select the 'name' and 'score' columns from the following DataFrame.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("Select specific columns:")
print(df[['name', 'score']])
```

Output:

```
Select specific columns:
        name  score
a  Anastasia   12.5
b       Dima    9.0
c  Katherine   16.5
d      James    NaN
e      Emily    9.0
f    Michael   20.0
g    Matthew   14.5
h      Laura    NaN
i      Kevin    8.0
j      Jonas   19.0
```

5. Pandas program to select the rows where the number of attempts in the examination is greater than 2.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("Number of attempts in the examination is greater than 2:")
print(df[df['attempts'] > 2])
```

Output:

```
Number of attempts in the examination is greater than 2:
      name  score  attempts qualify
b     Dima    9.0         3      no
d    James    NaN         3      no
f  Michael   20.0         3     yes
```

6.Pandas program to count the number of rows and columns of a DataFrame.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
total_rows=len(df.axes[0])
total_cols=len(df.axes[1])
print("Number of Rows: "+str(total_rows))
print("Number of Columns: "+str(total_cols))
```

Output:

Number of Rows: 10

Number of Columns: 4

7. Pandas program to select the rows where the score is missing, i.e. is NaN.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("Rows where score is missing:")
print(df[df['score'].isnull()])
```

Output:

```
Rows where score is missing:
    name  score  attempts qualify
d  James    NaN         3      no
h  Laura    NaN         1      no
```

8. Pandas program to append a new row 'k' to DataFrame with given values for each column. Now delete the new row and return the original data frame.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("Original rows:")
print(df)
print("\nAppend a new row:")
df.loc['k'] = ['Suresh',15.5, 1, 'yes']
print("Print all records after insert a new record:")
print(df)
print("\nDelete the new row and display the original  rows:")
df = df.drop('k')
print(df)
```

Output:

```
Original rows:
         name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
Append a new row:
Print all records after insert a new record:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
k     Suresh   15.5         1     yes
Delete the new row and display the original  rows:
         name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

9. Pandas program to sort the DataFrame first by 'name' in descending order, then by 'score' in ascending order.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("Orginal rows:")
print(df)
df.sort_values(by=['name', 'score'], ascending=[False, True])
print("Sort the data frame first by ‘name’ in descending order, then by ‘score’ in ascending order:")
print(df)
```

Output:

```
Orginal rows:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

```
Sort the data frame first by ‘name’ in descending order, then by ‘score’ in ascending order:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

10. Pandas program to change the name 'James' to 'Suresh' in name column of the data frame.

Code:

```
df = pd.DataFrame(exam_data , index=labels)
print("Original rows:")
print(df)
print("\nChange the name 'James' to ‘Shyam’:")
df['name'] = df['name'].replace('James', 'Shyam')
print(df)
```

Output:

```
Original rows:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes

Change the name 'James' to ‘Suresh’:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      Shyam    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

Result:

Thus the working of Pandas dataframe was executed and verified successfully.

Ex. No.: 4	Descriptive Analysis on Iris Dataset

Aim:

To read data from text files, Excel and the web for doing descriptive analysis on the Iris data set.

Algorithm:

Step 1: Start the program.

Step 2: Create an Iris Data Set in Excel Sheet.

Step 3: Import data from files, excel and web.

Step 4: Import iris data set as an ".csv file".

Step 5: Read the imported csv file into pandas dataframe.

Step 6: Create different analysis function.

Step 7: Perform descriptive analysis on Iris dataset.

Step 8: Present the results using matplotlib visualization.

Procedure:

Download the iris.csv file from https://www.kaggle.com/datasets/uciml/iris and use the pandas library to load this CSV file and convert it into the dataframe. read_csv() method is used to read CSV files.

Program:

1.Reading the dataset "Iris.csv".

Code:

```
import pandas as pd
data = pd.read_csv("E:\iris.csv")
```

print(data)

print(data.dtypes)

Output:

```
     sepal_length  sepal_width  petal_length  petal_width         species
0             5.1          3.5           1.4          0.2     Iris-setosa
1             4.9          3.0           1.4          0.2     Iris-setosa
2             4.7          3.2           1.3          0.2     Iris-setosa
3             4.6          3.1           1.5          0.2     Iris-setosa
4             5.0          3.6           1.4          0.2     Iris-setosa
..            ...          ...           ...          ...             ...
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

2.The function head() will display the top rows of the dataset, the default value of this function is 5, that is it will show top 5 rows when no argument is given to it.

Code:

data.head()

Output:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

3.Displaying the number of rows randomly.

In sample() function, it will also display the rows according to arguments given, but it will display the rows randomly.

Code:

```
data.sample(10)
```

Output:

	sepal_length	sepal_width	petal_length	petal_width	species
75	6.6	3.0	4.4	1.4	Iris-versicolor
26	5.0	3.4	1.6	0.4	Iris-setosa
83	6.0	2.7	5.1	1.6	Iris-versicolor
70	5.9	3.2	4.8	1.8	Iris-versicolor
94	5.6	2.7	4.2	1.3	Iris-versicolor
13	4.3	3.0	1.1	0.1	Iris-setosa
123	6.3	2.7	4.9	1.8	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica
92	5.8	2.6	4.0	1.2	Iris-versicolor
58	6.6	2.9	4.6	1.3	Iris-versicolor

4. Displaying the number of columns and names of the columns.

The column() function prints all the columns of the dataset in a list form.

Code:

```
data.columns
```

Output:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
dtype='object')
```

5. Displaying the shape of the dataset.

The shape of the dataset means to print the total number of rows or entries and the total number of columns or features of that particular dataset.

Code:

```
data.shape
```

Output:

(150, 5)

6.Display the whole dataset

Code:

```
print(data)
```

Output:

```
     sepal_length  sepal_width  petal_length  petal_width         species
0             5.1          3.5           1.4          0.2     Iris-setosa
1             4.9          3.0           1.4          0.2     Iris-setosa
2             4.7          3.2           1.3          0.2     Iris-setosa
3             4.6          3.1           1.5          0.2     Iris-setosa
4             5.0          3.6           1.4          0.2     Iris-setosa
..            ...          ...           ...          ...             ...
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

7. Code: Slicing the rows.

Slicing means if we want to print or work upon a particular group of lines that is from 10th row to 20th row.

Code:

```
print(data[10:21])
```

Output:

	sepal_length	sepal_width	petal_length	petal_width	species
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa
20	5.4	3.4	1.7	0.2	Iris-setosa

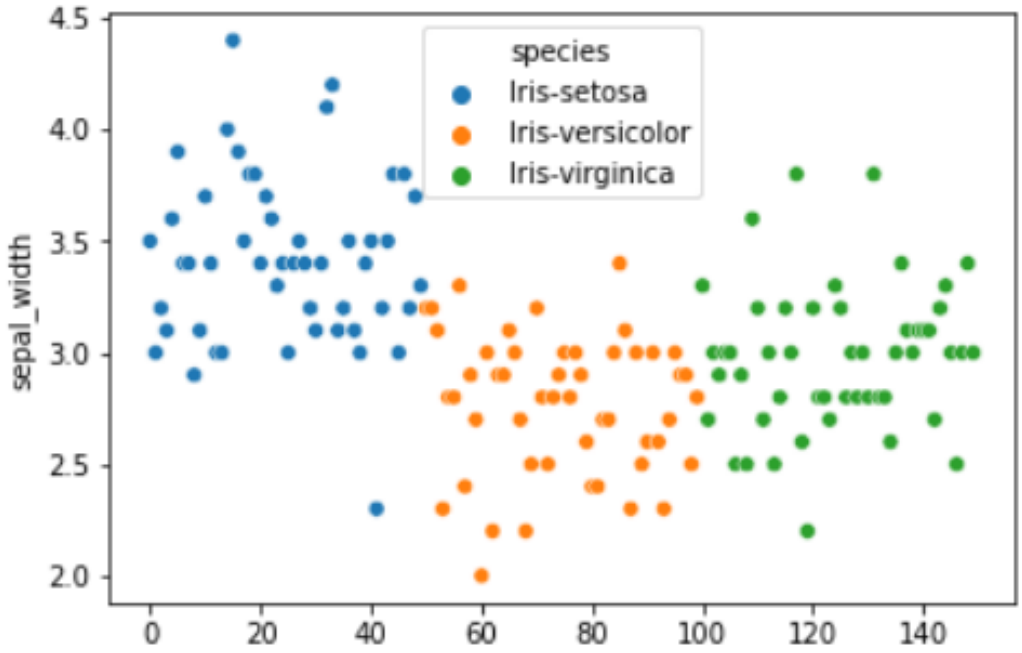
8. Plotting the datas using scatter plot from iris dataset.

Code:

```
import seaborn as sns
sns.scatterplot(x=data.index,y=data['sepal_width'],hue=data['species'])
```

Output:

```
<AxesSubplot:ylabel='sepal_width'>
```



Result:

Thus the descriptive analysis on the iris data set was successfully executed and verified

Ex. No.:5a	Univariate Analysis using the UCI diabetes data set

Aim:

To read data from csv files and to explore various commands for doing Univariate analysis using the UCI diabetes data set.

Algorithm:

Step 1: Start the program.

Step 2: Import and install necessary packages for diabetes data analysis and visualization.

Step 3: Read the dataset from Kaggle.

Step 4:Perform univariate analysis on the diabetes dataset.

Step 5: Stop the program.

Procedure:

Download the Pima_indian_diabetesdata as csv file from the https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database and use the Pandas library to load this CSV file and convert it into the dataframe. The read_csv() method is used to read CSV files.

Program:

To display data

Code:

```
import pandas as pd
df=pd.read_csv("D:\diabetes.csv")
print(df)
```

Output:

```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148            72             35        0  33.6
1            1      85             66             29        0  26.6
2            8      183            64             0         0  23.3
3            1      89             66             23       94  28.1
4            0      137            40             35      168  43.1
..         ...      ...           ...            ...      ...  ...
763         10      101            76             48      180  32.9
764          2      122            70             27        0  36.8
765          5      121            72             23      112  26.2
766          1      126            60              0        0  30.1
767          1       93            70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...

763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

To display data types

Code:

```
print(df.dtypes)
```

Output:

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction    float64
Age                         int64
Outcome                     int64
dtype: object
```

To print first five rows

Code:

```
print(df.head())
```

Output:

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
```

3 0.167 21 0

4 2.288 33 1

To get the shape of the dataset

Code:

```
print(df.shape)
```

Output:

(768, 9)

To calculate the mean

Code:

```
print("Mean of Pregnancies:%f" %df ['Pregnancies'].mean())
print("Mean of Blood Pressure:%f" %df ['BloodPressure'].mean())
print("Mean of Glucose:%f" %df ['Glucose'].mean())
print("Mean of Age:%f" %df ['Age'].mean())
```

Output:

Mean of Pregnancies:3.845052

Mean of Blood Pressure:69.105469

Mean of Glucose:120.894531

Mean of Age:33.240885

To calculate the median

Code:

```
print("Median of Pregnancies:%f" %df ['Pregnancies'].median())
print("Median of Blood Pressure:%f" %df ['BloodPressure'].median())
print("Median of Glucose:%f" %df ['Glucose'].median())
print("Median of Age:%f" %df ['Age'].median())
```

Output:

Median of Pregnancies:3.000000

Median of Blood Pressure:72.000000

Median of Glucose:117.000000

Median of Age:29.000000

To calculate the standard deviation

Code:

```
print("Standard Deviation for Pregnancies:%f" %df ['Pregnancies'].std())
print("Standard Deviation for Blood Pressure:%f" %df ['BloodPressure'].std())
print("Standard Deviation for Glucose:%f" %df ['Glucose'].std())
print("Standard Deviation for Age:%f" %df ['Age'].std())
```

Output:

Standard Deviation for Pregnancies:3.369578

Standard Deviation for Blood Pressure:19.355807

Standard Deviation for Glucose:31.972618

Standard Deviation for Age:11.760232

To calculate the Skewness

Code:

```
print("Skewness of Pregnancies:%f" %df ['Pregnancies'].skew())
print("Skewness of Blood Pressure:%f" %df ['BloodPressure'].skew())
print("Skewness of Glucose:%f" %df ['Glucose'].skew())
print("Skewness of Age:%f" %df ['Age'].skew())
```

Output:

Skewness of Pregnancies:0.901674

Skewness of Blood Pressure:-1.843608

Skewness of Glucose:0.173754

Skewness of Age:1.129597

To calculate the kurtosis

Code:

```
print("Kurtosis of Pregnancies:%f" %df ['Pregnancies'].kurt())
```

```
print("Kurtosis of Blood Pressure:%f" %df ['BloodPressure'].kurt())
print("Kurtosis of Glucose:%f" %df ['Glucose'].kurt())
print("Kurtosis of Age:%f" %df ['Age'].kurt())
```

Output:

```
Kurtosis of Pregnancies:0.159220
Kurtosis of Blood Pressure:5.180157
Kurtosis of Glucose:0.640780
Kurtosis of Age:0.643159
```

To describe the data

Code:

```
df.Glucose.describe()
```

Output:

```
count    768.000000
mean     120.894531
std       31.972618
min        0.000000
25%       99.000000
50%      117.000000
75%      140.250000
max      199.000000
Name: Glucose, dtype: float64
```

To create frequency table for pregnancies data

Code:

```
df['Pregnancies'].value_counts()
```

Output:

```
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

To create frequency table for glucose data

Code:

```
df['Glucose'].value_counts()
```

Output:

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

To find frequency count for outcome

Code:

```
pd.crosstab(index=df['Outcome'],columns='count')
```

Output:

col_0	count
Outcome	
0	500
1	268

To find frequency count for pregnancies

Code:

```
df[‘Pregnancies’].value_counts()
```

Output:

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

To find frequency table for pregnancy

Code:

```
pd.crosstab(index=df['Pregnancies'],columns='count')
```

Output:

col_0	Count
Pregnancies	
0	111
1	135
2	103
3	75
4	68
5	57
6	50
7	45
8	38
9	28
10	24
11	11
12	9

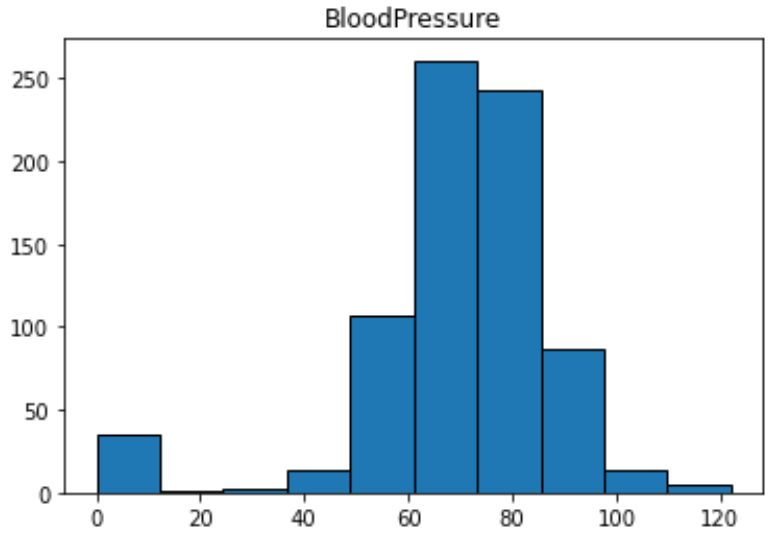
col_0	Count
Pregnancies	
13	10
14	2
15	1
17	1

To plot the histogram for blood pressure.

Code:

```
import matplotlib.pyplot as plt
df.hist(column='BloodPressure',grid=False,edgecolor='black')
array([[<AxesSubplot:title={'center':'BloodPressure'}>]], dtype=object)
```

Output:

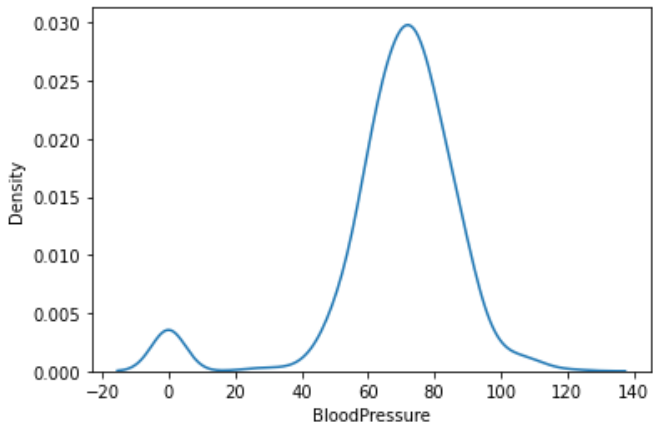


To plot the line curve for blood pressure and density.

Code:

```
import seaborn as sns
sns.kdeplot(df['BloodPressure'])
<AxesSubplot:xlabel='BloodPressure', ylabel='Density'>
```

Output:

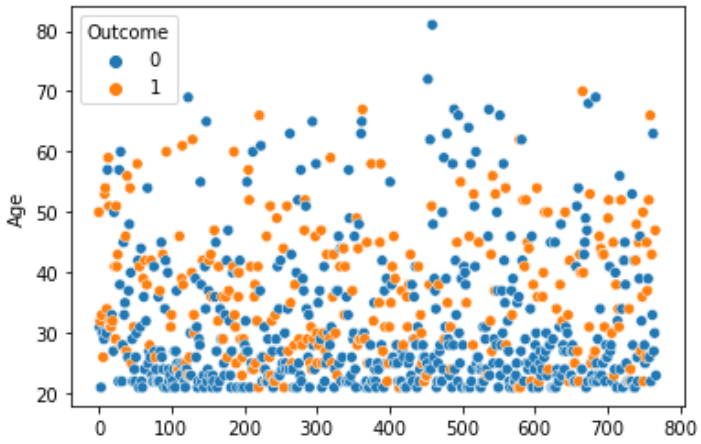


To plot the scatterplot for age and outcome.

Code:

```
import seaborn as sns
sns.scatterplot(x=df.index,y=df['Age'],hue=df['Outcome'])
<AxesSubplot:ylabel='Age'>
```

Output:



To calculate pregnancy proportion and its percentage.

Code:

```
import numpy as np
import pandas as pd
preg_proportion=np.array(df['Pregnancies'].value_counts())
preg_month=np.array(df['Pregnancies'].value_counts().index)
preg_proportion_perc=np.array(np.round(preg_proportion/sum(preg_proportion),3)*100,dtype=i
nt)
preg=pd.DataFrame({'month':preg_month,'count_of_preg_prop':preg_proportion,'percentage_pr
oportion':preg_proportion_perc})
preg.set_index(['month'],inplace=True)
preg.head(10)
```

Output:

month	count_of_preg_prop	percentage_proportion
1	135	17
0	111	14
2	103	13
3	75	9
4	68	8
5	57	7
6	50	6
7	45	5
8	38	4
9	28	3

To analyze the diabetes dataset

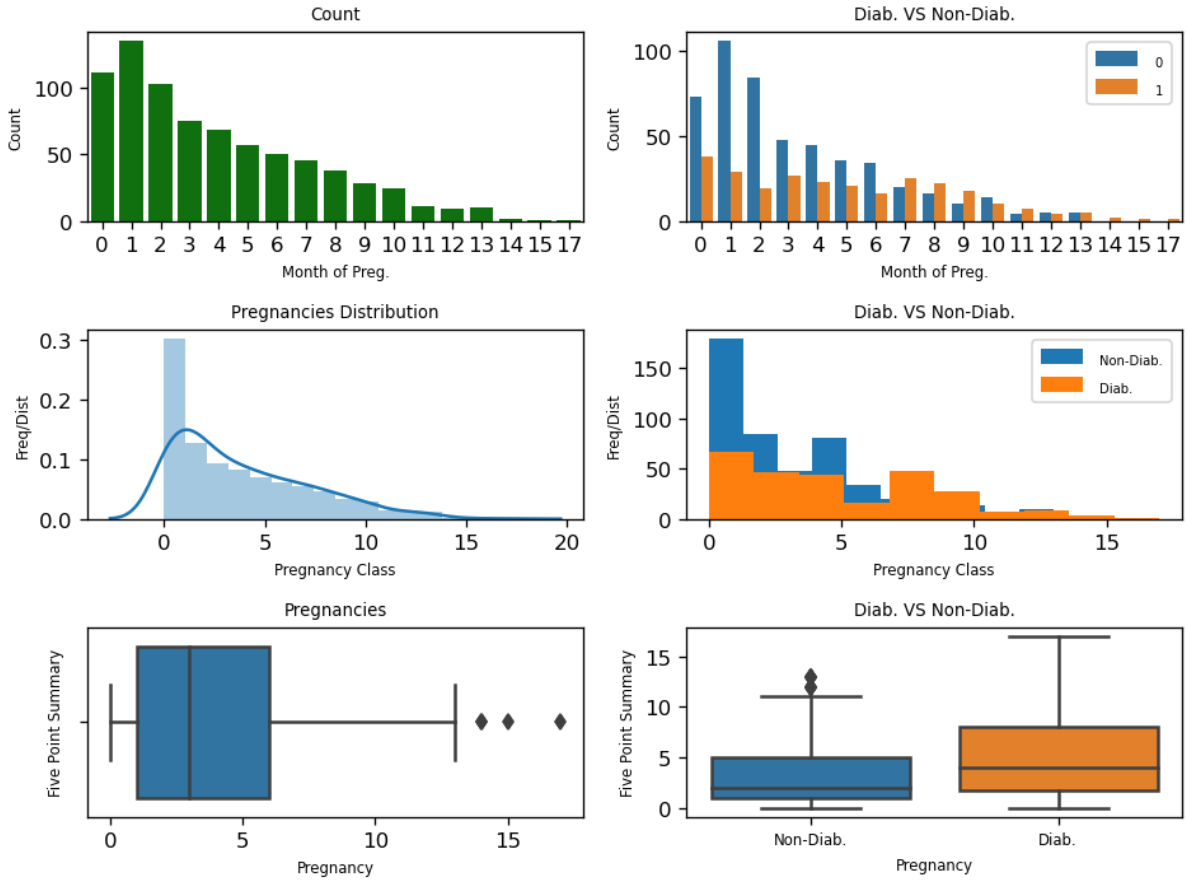
Code:

```
import matplotlib.pyplot as plt
fig,axes = plt.subplots(nrows=3,ncols=2,dpi=120,figsize = (8,6))
```

```
plot00=sns.countplot('Pregnancies',data=df,ax=axes[0][0],color='green')
axes[0][0].set_title('Count',fontdict={'fontsize':8})
axes[0][0].set_xlabel('Month of Preg.',fontdict={'fontsize':7})
axes[0][0].set_ylabel('Count',fontdict={'fontsize':7})
plt.tight_layout()

plot01=sns.countplot('Pregnancies',data=df,hue='Outcome',ax=axes[0][1])
axes[0][1].set_title('Diab. VS Non-Diab.',fontdict={'fontsize':8})
axes[0][1].set_xlabel('Month of Preg.',fontdict={'fontsize':7})
axes[0][1].set_ylabel('Count',fontdict={'fontsize':7})
plot01.axes.legend(loc=1)
plt.setp(axes[0][1].get_legend().get_texts(), fontsize='6')
plt.setp(axes[0][1].get_legend().get_title(), fontsize='6')
plt.tight_layout()

plot10 = sns.distplot(df['Pregnancies'],ax=axes[1][0])
axes[1][0].set_title('Pregnancies Distribution',fontdict={'fontsize':8})
axes[1][0].set_xlabel('Pregnancy Class',fontdict={'fontsize':7})
axes[1][0].set_ylabel('Freq/Dist',fontdict={'fontsize':7})
plt.tight_layout()

plot11 = df[df['Outcome']==False]['Pregnancies'].plot.hist(ax=axes[1][1],label='Non-Diab.')
plot11_2=df[df['Outcome']==True]['Pregnancies'].plot.hist(ax=axes[1][1],label='Diab.')
axes[1][1].set_title('Diab. VS Non-Diab.',fontdict={'fontsize':8})
axes[1][1].set_xlabel('Pregnancy Class',fontdict={'fontsize':7})
axes[1][1].set_ylabel('Freq/Dist',fontdict={'fontsize':7})
plot11.axes.legend(loc=1)
plt.setp(axes[1][1].get_legend().get_texts(), fontsize='6') # for legend text
plt.setp(axes[1][1].get_legend().get_title(), fontsize='6') # for legend title
plt.tight_layout()
```

```
plot20 = sns.boxplot(df['Pregnancies'],ax=axes[2][0],orient='v')
axes[2][0].set_title('Pregnancies',fontdict={'fontsize':8})
axes[2][0].set_xlabel('Pregnancy',fontdict={'fontsize':7})
axes[2][0].set_ylabel('Five Point Summary',fontdict={'fontsize':7})
plt.tight_layout()

plot21 = sns.boxplot(x='Outcome',y='Pregnancies',data=df,ax=axes[2][1])
axes[2][1].set_title('Diab. VS Non-Diab.',fontdict={'fontsize':8})
axes[2][1].set_xlabel('Pregnancy',fontdict={'fontsize':7})
axes[2][1].set_ylabel('Five Point Summary',fontdict={'fontsize':7})
plt.xticks(ticks=[0,1],labels=['Non-Diab.','Diab.'],fontsize=7)
plt.tight_layout()
plt.show()
```

Output:



Result:

Thus, the Univariate analysis using the UCI diabetes data set was successfully executed and practically verified.

Ex. No.:5b	Bivariate analysis using the UCI diabetes data set

Aim:

To read data from text files, Excel and the web and to explore the various commands for Bivariate analysis using the UCI diabetes data set.

Algorithm:

Step 1: Start the program.

Step 2: Import and install necessary packages for diabetes data analysis and visualization.

Step 3: Read the dataset from Kaggle.

Step 4: Perform bivariate analysis on the diabetes dataset.

Step 5: Stop the program.

Procedure:

Download the Pima_indian_diabetes data as csv file from the https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database and use the Pandas library to load this CSV file and convert it into the dataframe. The read_csv() method is used to read CSV files.

Program:

Linear Regression Modelling:

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model
```

```
from sklearn.metrics import mean_squared_error,r2_score
#Load the diabetes dataset
diabetes_X,diabetes_y=datasets.load_diabetes(return_X_y=True)
#Use only one feature
diabetes_X=diabetes_X[:,np.newaxis,2]
#Split the data into training and testing sets
diabetes_X_train=diabetes_X[:-50]
diabetes_X_test=diabetes_X[:-50]
#Split the targets into training and testing data sets
diabetes_y_train=diabetes_y[:-50]
diabetes_y_test=diabetes_y[:-50]
#Create linear regression object
regr=linear_model.LinearRegression()
#Train the model using the training sets.
regr.fit(diabetes_X_train,diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred=regr.predict(diabetes_X_test)
print("Coefficients:\n",regr.coef_)
```

Output:

```
Coefficients:
 [945.4992184]
```

#To calculate the mean squared error

Code:

```
print("Mean Squared Error:%2f"%mean_squared_error(diabetes_y_test,diabetes_y_pred))
```

Output:

```
Mean Squared Error:3943.926210
```

To calculate the coefficients of determination

Code:

```
print("Coefficients of Detemination:%2f"%r2_score(diabetes_y_test,diabetes_y_pred))
```

Output:

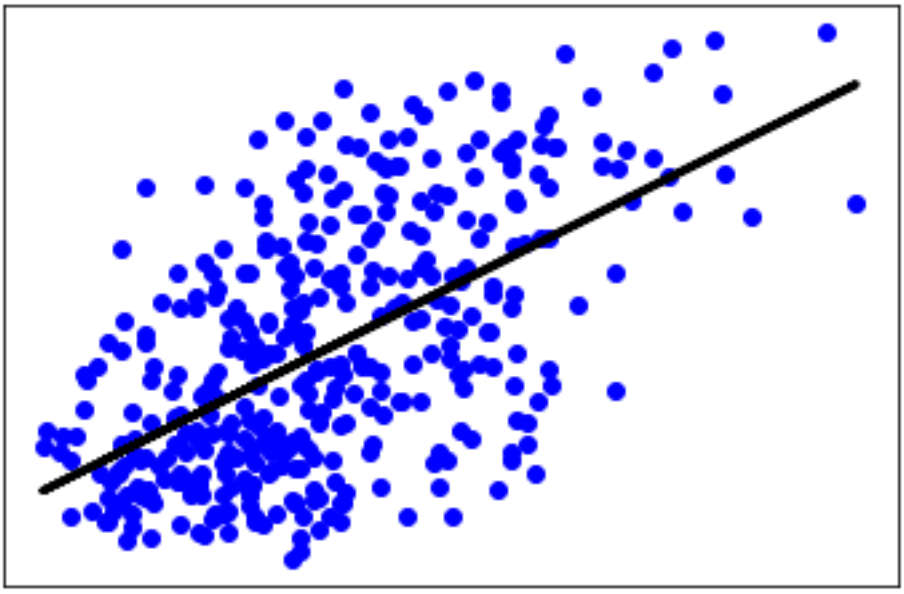
Coefficients of Detemination:0.335150

#To plot the outputs

Code:

```
plt.scatter(diabetes_X_test,diabetes_y_test,color="blue")
plt.plot(diabetes_X_test,diabetes_y_pred,color="black",linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Output:



Linear Regression Modelling:

Code:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
diabetes=pd.read_csv("D:\diabetes.csv")
diabetes
```

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Code:

```
X=df.drop("Outcome",axis=1)
Y=df[["Outcome"]]
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=0)
```

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X_train, y_train)
y_pred= model.predict(X_test)
print("Accuracy for test set is {}.".format(round(metrics.accuracy_score(y_test,y_pred),4)))
print("Precision for test set is {}.".format(round(metrics.precision_score(y_test,y_pred),4)))
print("Recall for test set is {}.".format(round(metrics.recall_score(y_test,y_pred),4)))
```

Output:

Accuracy for test set is 0.7917.

Precision for test set is 0.7115.

Recall for test set is 0.5968.

Code:

```
print(metrics.classification_report(y_test,y_pred))
```

Output:

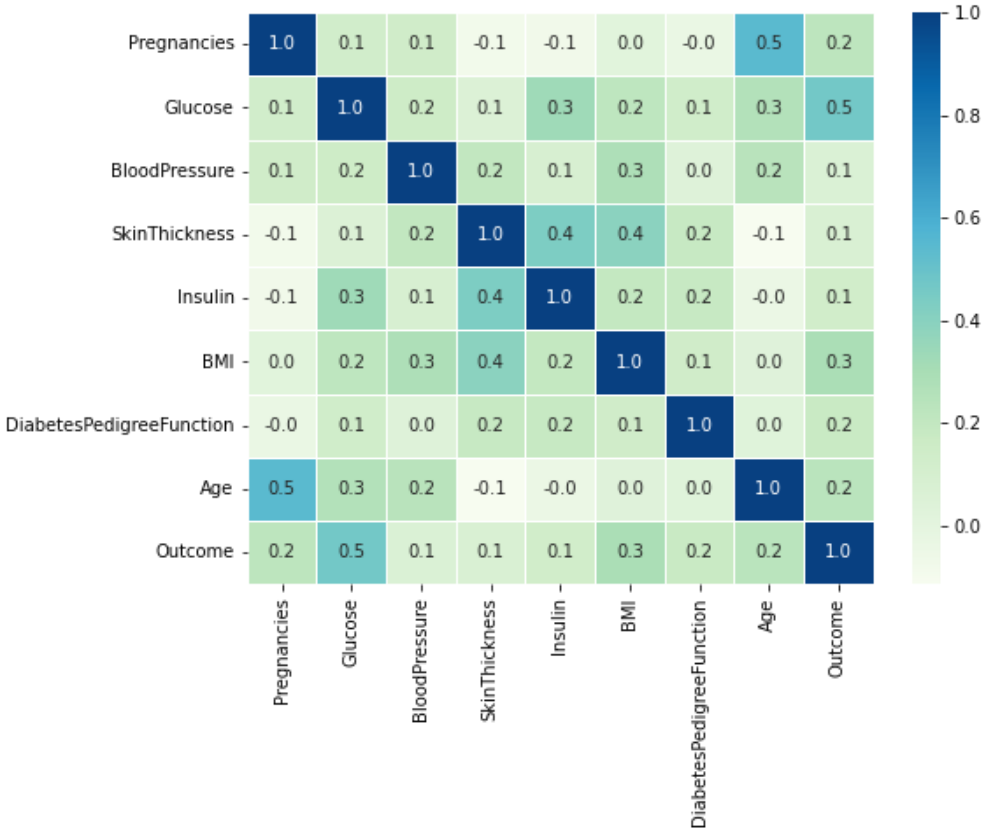
```
precision    recall  f1-score   support

           0       0.82      0.88      0.85       130
           1       0.71      0.60      0.65        62

    accuracy                           0.79       192
   macro avg       0.77      0.74      0.75       192
weighted avg       0.79      0.79      0.79       192
```

#Visualization

Code:

```
f,ax=plt.subplots(figsize=(8,6))
sns.heatmap(df.corr(),cmap="GnBu",annot=True,linewidths=0.5,fmt='.1f',ax=ax)
plt.show()
```

Output:



Result:

Thus the bivariate analysis using the UCI diabetes data set was successfully executed and verified.

Ex. No.: 5c	Multiple Regression Analysis using the UCI diabetes data set

Aim:

To read data from csv files and to explore various commands for doing multiple regression analysis using the UCI diabetes data set.

Algorithm:

Step 1: Start the program.

Step 2: Import and install necessary packages for diabetes data analysis and visualization.

Step 3: Read the dataset from Kaggle.

Step 4: Perform multiple regression analysis on the diabetes dataset.

Step 5: Stop the program.

Procedure:

Download the UCI diabetes data as csv file from the https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database and use the Pandas library to load this CSV file and convert it into the dataframe. The read_csv() method is used to read CSV files.

Program:

```
#import libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
#Loading data
df=pd.read_csv("E:\diabetes.csv")
df
```

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

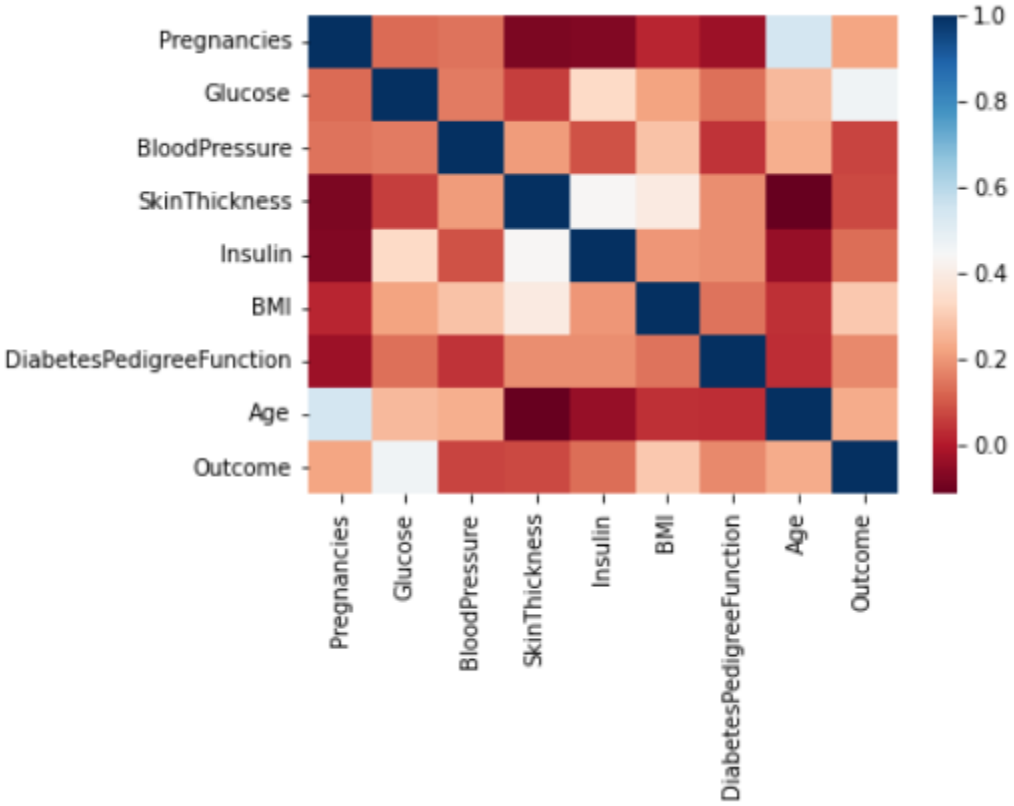
```
#Calculate the correlation matrix
corr=diabetes.corr()
#Display the correlation matrix
display(corr)
```

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

```
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,cmap='RdBu')
<AxesSubplot:>
```

Output:




```
#Train/test split
X=df.drop("Outcome",axis=1)
Y=df[["Outcome"]]
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25,random_state=0)
from sklearn.linear_model import LinearRegression
#create a linear regression model object
model=LinearRegression()
#pass through X_train and Y_train data set
model.fit(X_train,Y_train)
#grab the coefficient of our model and the intercept
intercept=model.intercept_[0]
```

```
coefficient=model.coef_[0][0]
print("The intercept for our model is {:.4}".format(intercept))
print(X.columns)
print("The Coefficients for the above given data are:")
print(model.coef_[0])
```

Output:

```
The intercept for our model is -0.879
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
dtype='object')
The Coefficients for the above given data are:
[ 0.0149227   0.0057196  -0.00214665  0.00101156 -0.00017079  0.0134799
0.14124496  0.00382917]
```

Code:

```
#To get multiple predictions
Y_predict=model.predict(X_test)
#To show first five predictions
Y_predict[:5]
```

Output:

```
array([[1.01391226],
       [0.21532924],
       [0.09157383],
       [0.60583158],
       [0.15988782]])
```

Code:

```
#define input
```

```
import statsmodels.api as sm
X2=sm.add_constant(X)
#create OLS model
model=sm.OLS(Y,X2)
#fit the data
est=model.fit()
#print out a summary
print(est.summary())
```

Output:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Outcome   R-squared:                       0.303
Model:                            OLS   Adj. R-squared:                  0.296
Method:                 Least Squares   F-statistic:                     41.29
Date:                Thu, 01 Dec 2022   Prob (F-statistic):           7.36e-55
Time:                        18:24:41   Log-Likelihood:                -381.91
No. Observations:                 768   AIC:                             781.8
Df Residuals:                     759   BIC:                             823.6
Df Model:                           8
Covariance Type:            nonrobust
===============================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------------
const                       -0.8539      0.085     -9.989      0.000      -1.022      -0.686
Pregnancies                  0.0206      0.005      4.014      0.000       0.011       0.031
Glucose                      0.0059      0.001     11.493      0.000       0.005       0.007
BloodPressure               -0.0023      0.001     -2.873      0.004      -0.004      -0.001
SkinThickness                0.0002      0.001      0.139      0.890      -0.002       0.002
Insulin                     -0.0002      0.000     -1.205      0.229      -0.000       0.000
BMI                          0.0132      0.002      6.344      0.000       0.009       0.017
DiabetesPedigreeFunction     0.1472      0.045      3.268      0.001       0.059       0.236
Age                          0.0026      0.002      1.693      0.091      -0.000       0.006
==============================================================================
Omnibus:                       41.539   Durbin-Watson:                   1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               31.183
Skew:                           0.395   Prob(JB):                     1.69e-07
Kurtosis:                       2.408   Cond. No.                     1.10e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.1e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

#estimate confidence intervals

Code:

```
est.conf_int()
```

Output:

	0	1
const	-1.021709	-0.686079
Pregnancies	0.010521	0.030663
Glucose	0.004909	0.006932
BloodPressure	-0.003925	-0.000739
SkinThickness	-0.002029	0.002338
Insulin	-0.000475	0.000114
BMI	0.009146	0.017343
DiabetesPedigreeFunction	0.058792	0.235682
Age	-0.000419	0.005662

#estimatepvalues

Code:

est.pvalues

Output:

```
const                      3.707465e-22
Pregnancies                6.561462e-05
Glucose                    2.691192e-28
BloodPressure              4.178788e-03
SkinThickness              8.895424e-01
Insulin                    2.285711e-01
BMI                        3.853484e-10
DiabetesPedigreeFunction   1.131733e-03
Age                        9.092163e-02
dtype: float64
```

Code:

from sklearn.metrics import mean_squared_error,mean_absolute_error

import math

```
#calculate the mean squared error
model_mse=mean_squared_error(Y_test,Y_predict)
#calculate the mean absolute error
model_mae=mean_absolute_error(Y_test,Y_predict)
#calculate the root mean squared error
model_rmse=math.sqrt(model_mse)

print("Mean Squared Error {:.3}".format(model_mse))
print("Mean Absolute Error {:.3}".format(model_mae))
print("Root Mean Squared Error {:.3}".format(model_rmse))
```

Output:

```
Mean Squared Error 0.148
Mean Absolute Error 0.322
Root Mean Squared Error 0.384
```

Result:

Thus the multiple regression analysis using the UCI diabetes data set was successfully executed and verified.

Ex. No.:6	Apply and explore various plotting functions on UCI data sets

Aim:

To read data from csv files and to explore various commands for plotting functionson the UCI diabetes data set.

Algorithm:

Step 1: Download the UCI dataset from UCI machine learning repository.

Step 2: Read the dataset by importing the file in Jupyter notebook.

Step 3: Import the iris dataset as an ".csv" file.

Step 4: Read and import the csv file into pandas dataframe.

Step 5: Apply and explore the various plotting functions.

Step 6: Visualize the various curves, plots and three-dimensional plotting.

Procedure:

Download the UCI diabetes data as csv file from the https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database and use the Pandas library to load this CSV file and convert it into the dataframe. The read_csv() method is used to read CSV files.

Program:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(color_codes=True)
%matplotlib inline
```

```
import warnings
warnings.filterwarnings("ignore")
diabetes=pd.read_csv("E:\diabetes.csv")
diabetes
```

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Code:

```
diabetes.describe()
```

Output:

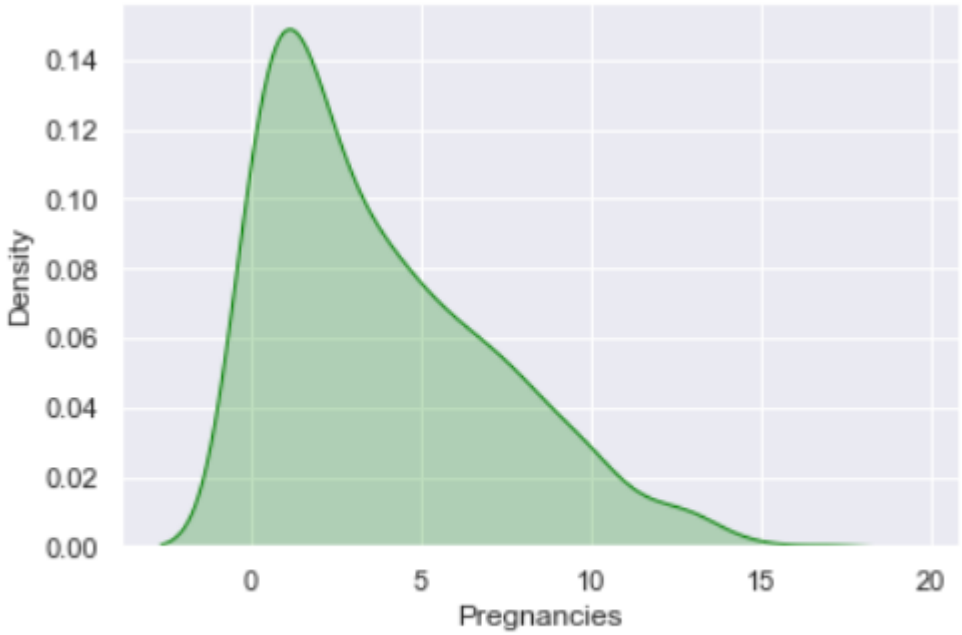
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Normal Curves:

Code:

```
sns.kdeplot(diabetes["Pregnancies"],color="green",shade=True)
plt.show()
plt.figure()
```

Output:

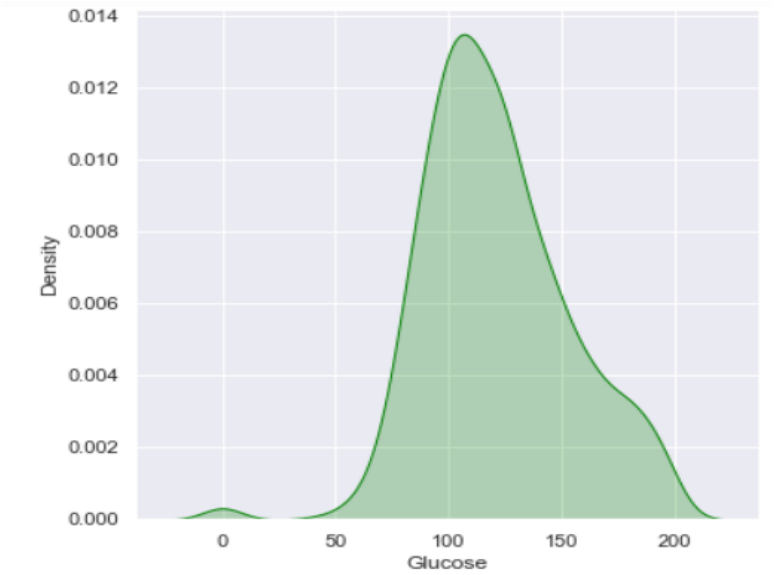



```
<Figure size 432x288 with 0 Axes>
```

Code:

```
plt.figure(figsize=(6,6))
sns.kdeplot(diabetes["Glucose"],color="green",shade=True)
plt.show()
plt.figure()
```

Output:



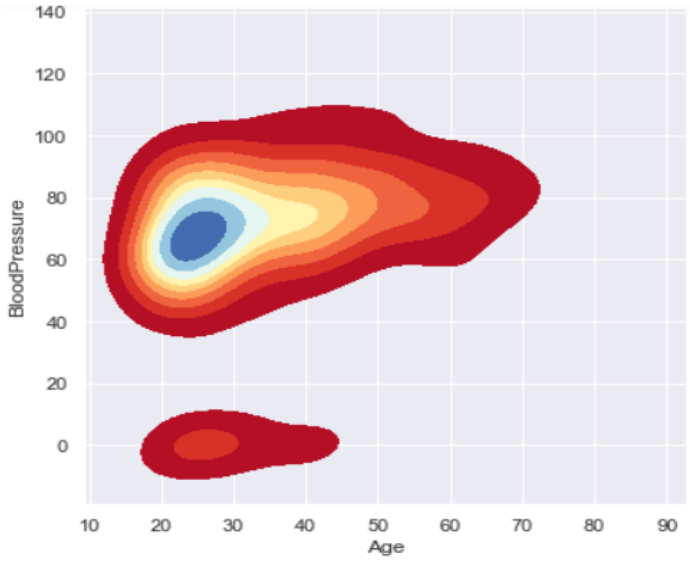
<Figure size 432x288 with 0 Axes>

Density and Contour plots:

Code:

```
plt.figure(figsize=(6,6))
sns.kdeplot(diabetes["Age"],diabetes["BloodPressure"],cmap="RdYlBu",shade=True)
plt.show()
plt.figure()
```

Output:

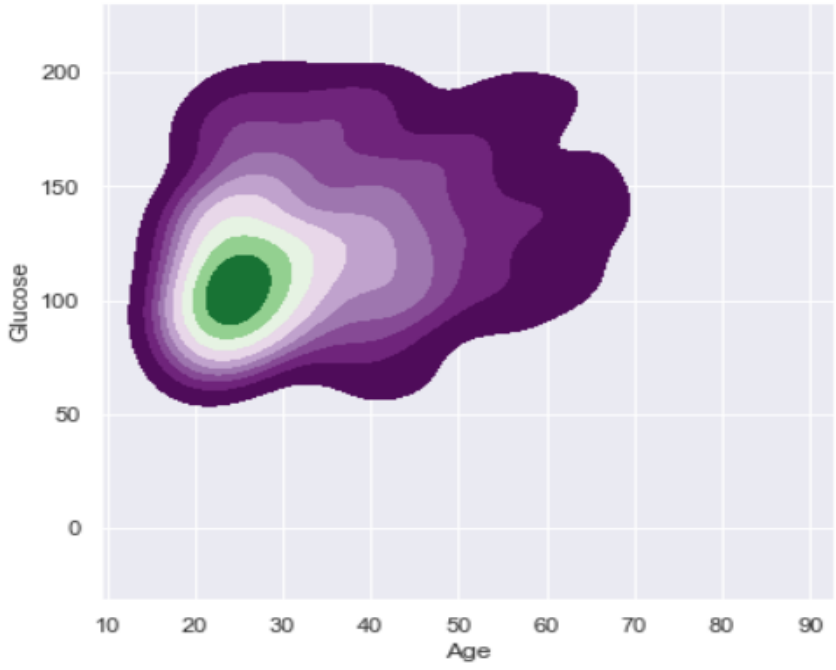


<Figure size 432x288 with 0 Axes>

Code:

```
plt.figure(figsize=(6,6))
sns.kdeplot(x=diabetes.Age,y=diabetes.Glucose,cmap="PRGn",shade=True,bw_adjust=1)
plt.show()
```

Output:



Correlation and Scatter plots:

Code:

```
corr=diabetes.corr()
display(corr)
```

Output:

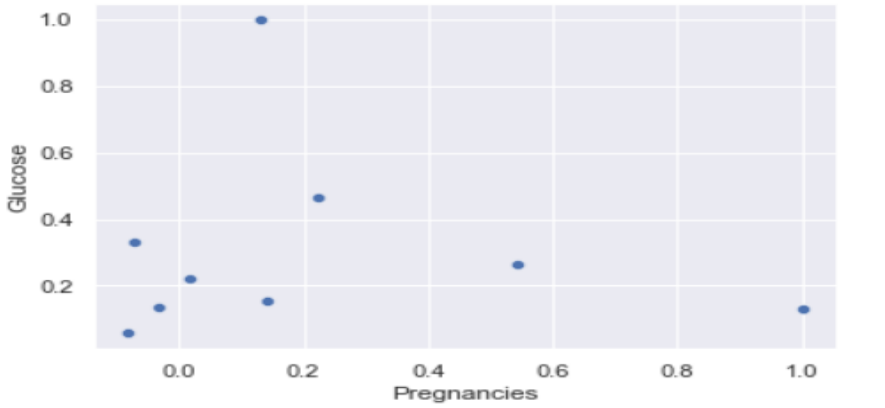
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844

Code:

```
import seaborn as sns
sns.scatterplot(x="Pregnancies",y="Glucose",data=corr)
```

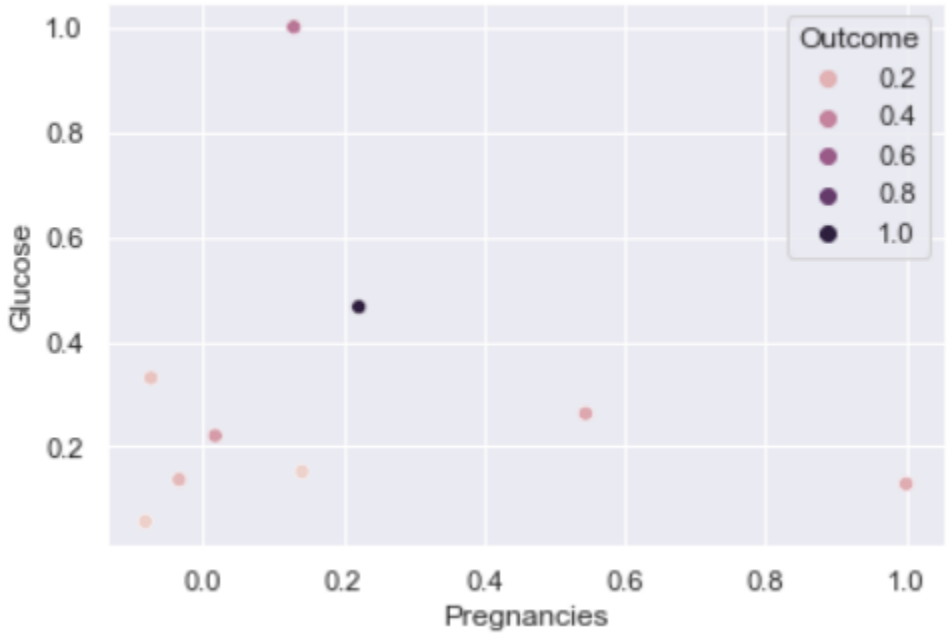
Output:

```
<AxesSubplot:xlabel='Pregnancies', ylabel='Glucose'>
```

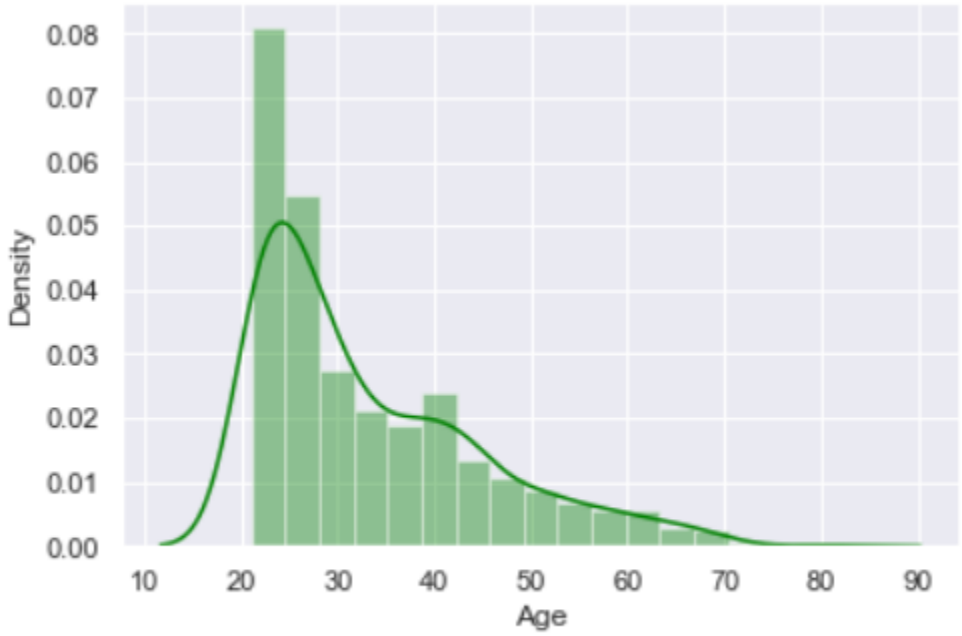


```
sns.scatterplot(x="Pregnancies",y="Glucose",hue="Outcome",data=corr)
```

```
<AxesSubplot:xlabel='Pregnancies', ylabel='Glucose'>
```

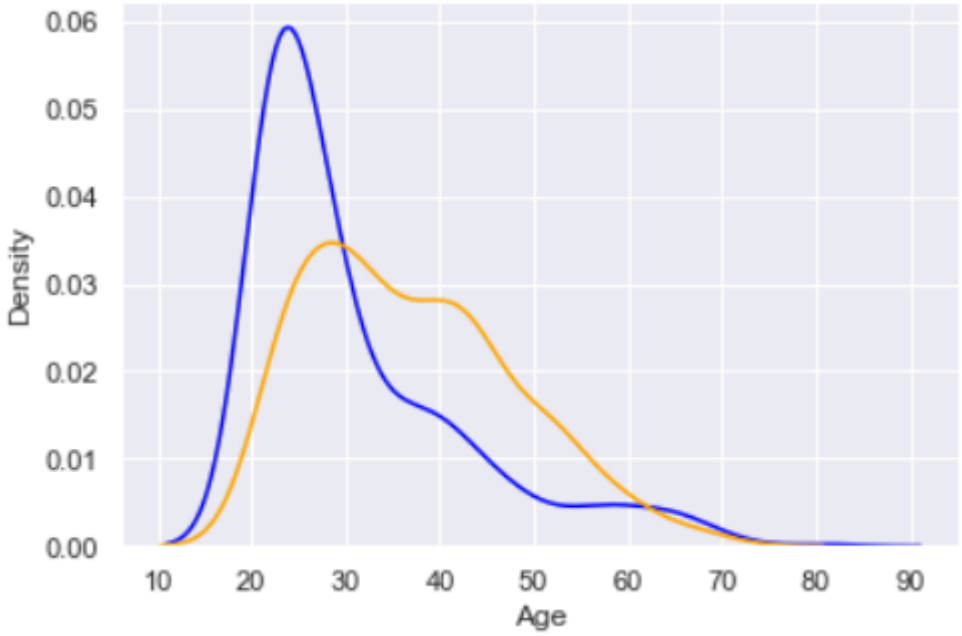



```
sns.distplot(diabetes["Age"],color="green")
plt.show()
plt.figure()
```




```
<Figure size 432x288 with 0 Axes>
```

```
sns.kdeplot(diabetes[diabetes.Outcome==0]['Age'],color="blue")
sns.kdeplot(diabetes[diabetes.Outcome==1]['Age'],color="orange")
plt.show()
```



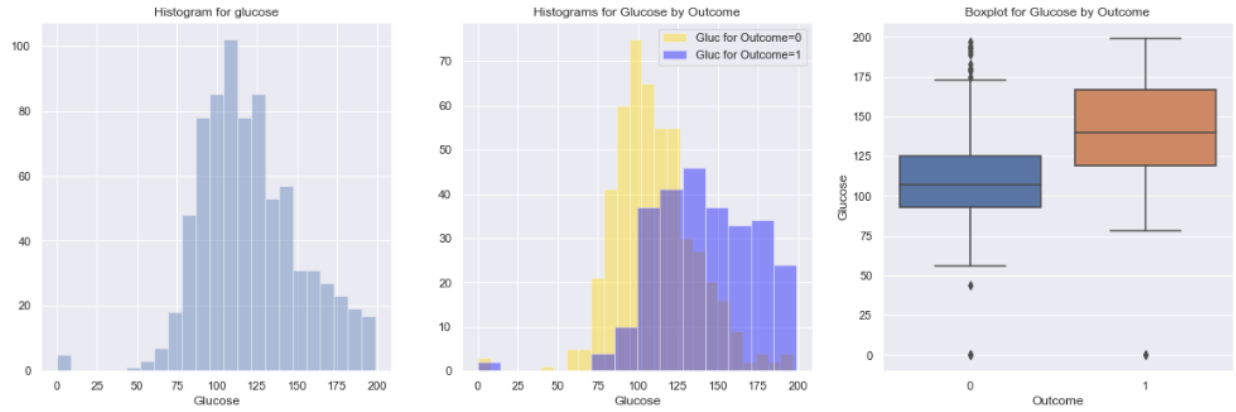
Histogram:

Code:

```
dia1=diabetes[diabetes.Outcome==1]
dia0=diabetes[diabetes.Outcome==0]
plt.figure(figsize=(20,6))
plt.subplot(1,3,1)
plt.title("Histogram for glucose")
sns.distplot(diabetes.Glucose,kde=False)
plt.subplot(1,3,2)
sns.distplot(dia0.Glucose,kde=False,color="Gold",label="Gluc for Outcome=0")
sns.distplot(dia1.Glucose,kde=False,color="Blue",label="Gluc for Outcome=1")
plt.title("Histograms for Glucose by Outcome")
plt.legend()
plt.subplot(1,3,3)
sns.boxplot(x=diabetes.Outcome,y=diabetes.Glucose)
plt.title("Boxplot for Glucose by Outcome")
```

Output:

```
Text(0.5, 1.0, 'Boxplot for Glucose by Outcome')
```



Three Dimensional plotting:

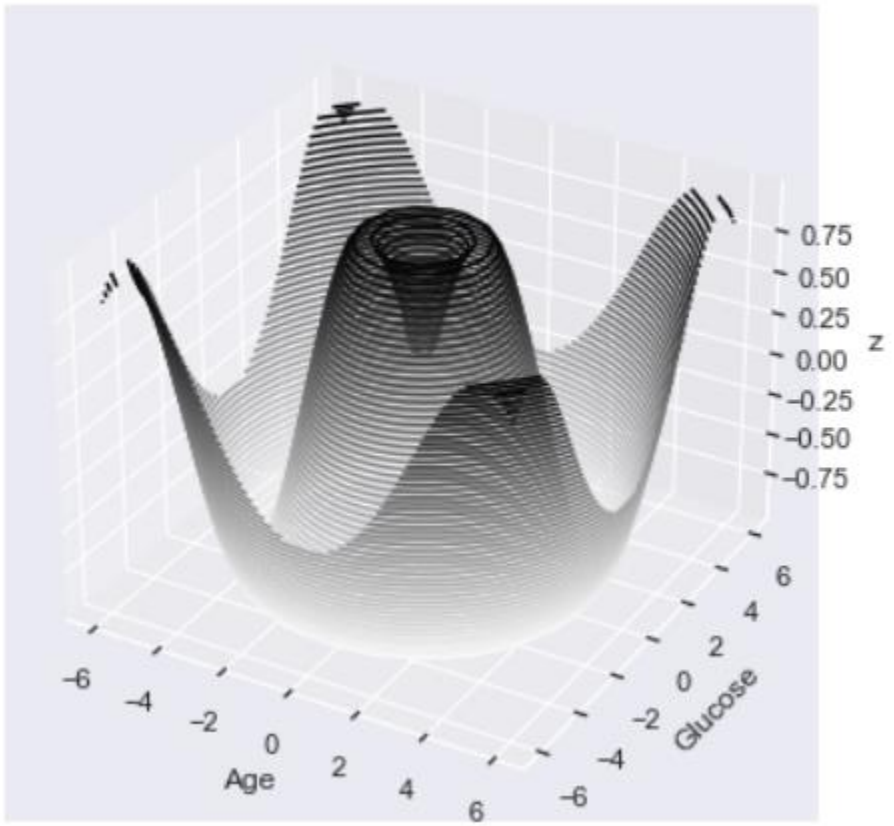
Code:

```
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import matplotlib
import functools
import plotly.express as px
import plotly.graph_objects as go
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
diabetes=pd.read_csv("E:\diabetes.csv")
x=diabetes.Age[:20]
y=diabetes.Glucose[:20]
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
```

```
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = plt.figure(figsize=(6,6))
ax = plt.axes(projection ='3d')
ax.contour3D(X, Y, Z,50,cmap='binary')
ax.set_xlabel('Age')
ax.set_ylabel('Glucose')
ax.set_zlabel('z')
```

Output:

```
Text(0.5, 0, 'z')
```



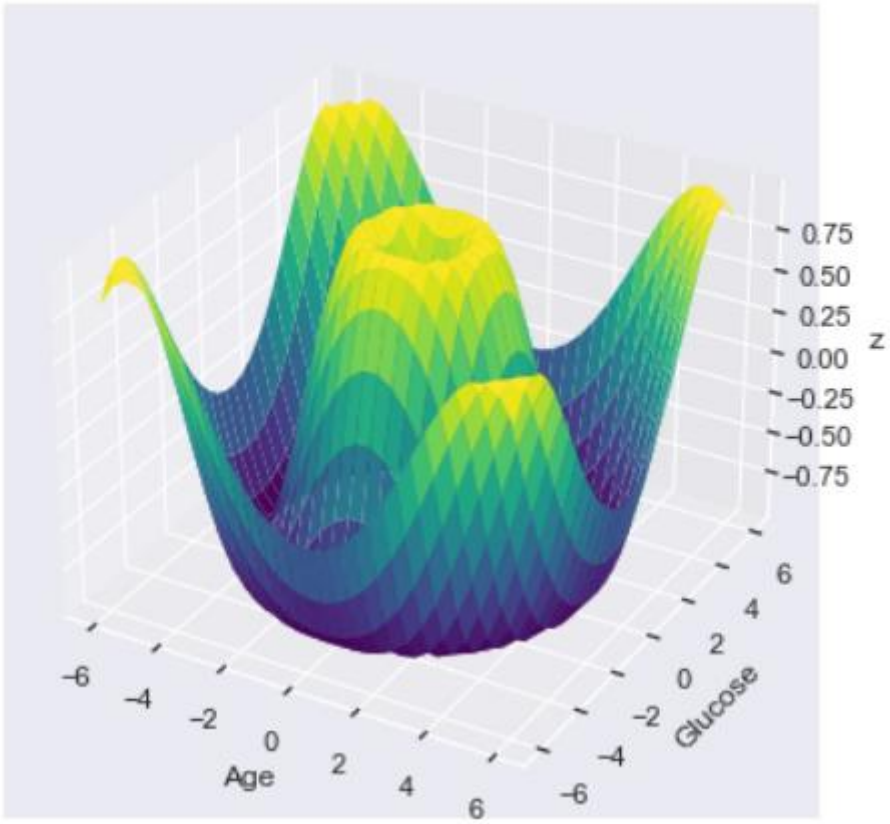
Code:

```
fig = plt.figure(figsize=(6,6))
ax = plt.axes(projection ='3d')
```

```
ax.plot_surface(X, Y, Z,rstride=1,cstride=1,cmap='viridis',edgecolor='none')
ax.set_xlabel('Age')
ax.set_ylabel('Glucose')
ax.set_zlabel('z')
```

Output:

```
Text(0.5, 0, 'z')
```

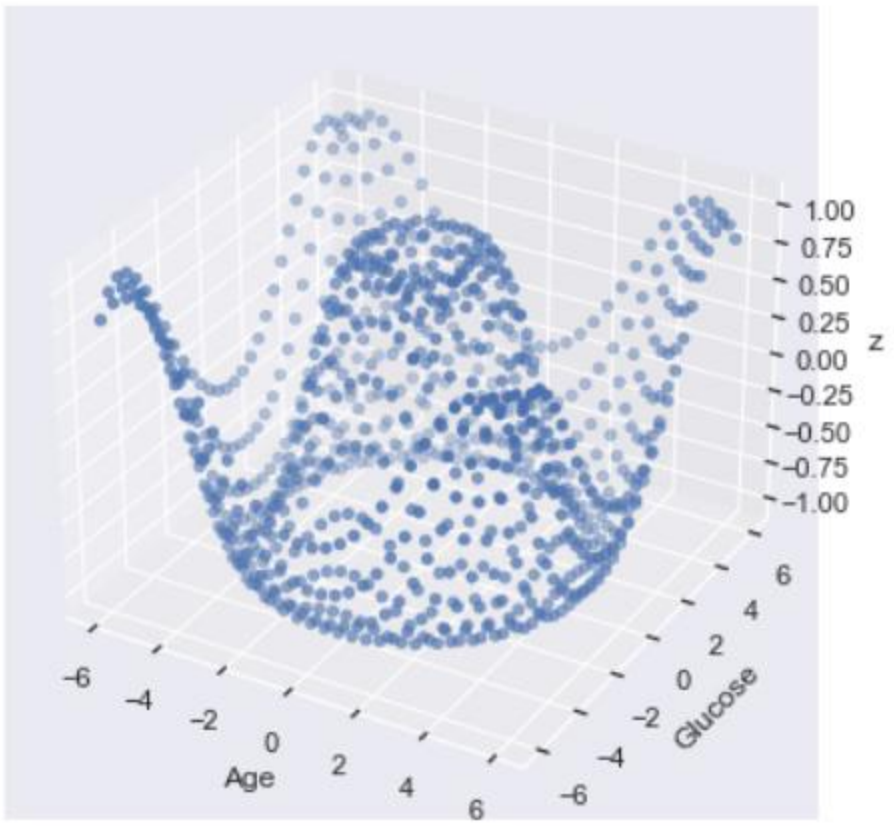


Code:

```
fig = plt.figure(figsize=(6,6))
ax = plt.axes(projection ='3d')
ax.scatter(X, Y, Z,cmap='viridis',linewidth=0.5)
ax.set_xlabel('Age')
ax.set_ylabel('Glucose')
ax.set_zlabel('z')
```

Output:

```
Text(0.5, 0, 'z')
```



Result:

Thus the various plotting functions on UCI diabetes data set has been executed successfully and verified.

Ex. No.:7	Visualizing Geographic Data with Basemap

Aim:

To visualize the geographic data with basemap python library.

Procedure:

1. Install the basemap package

Install the below package:

Use google colab (in anaconda prompt ,conda version is need to change, it may affect our other packages compatability)

pip install basemap

(or)

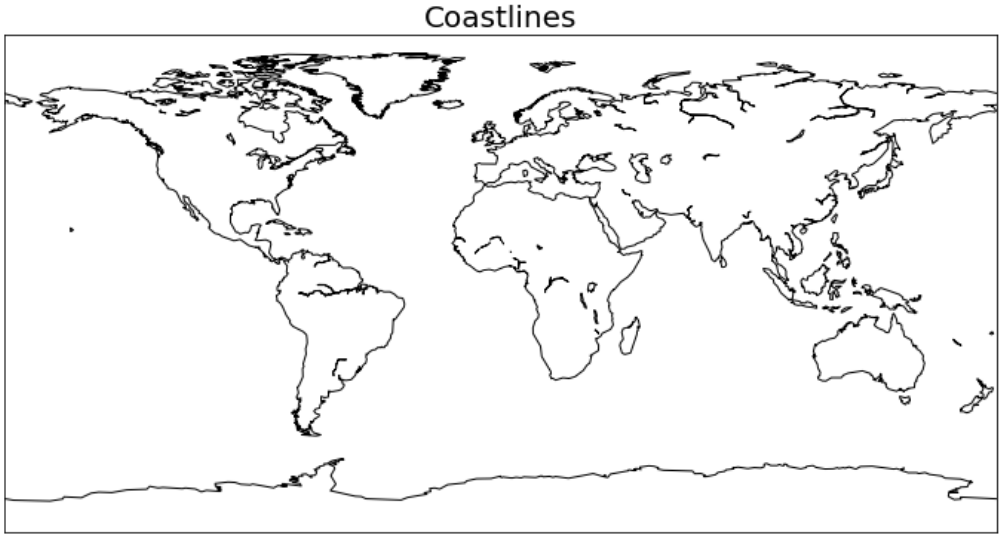
conda install -c https://conda.anaconda.org/anaconda basemap

2. Explore on various projection options example: ortho, lcc.
3. Mark the location using longitude and latitude

Program:

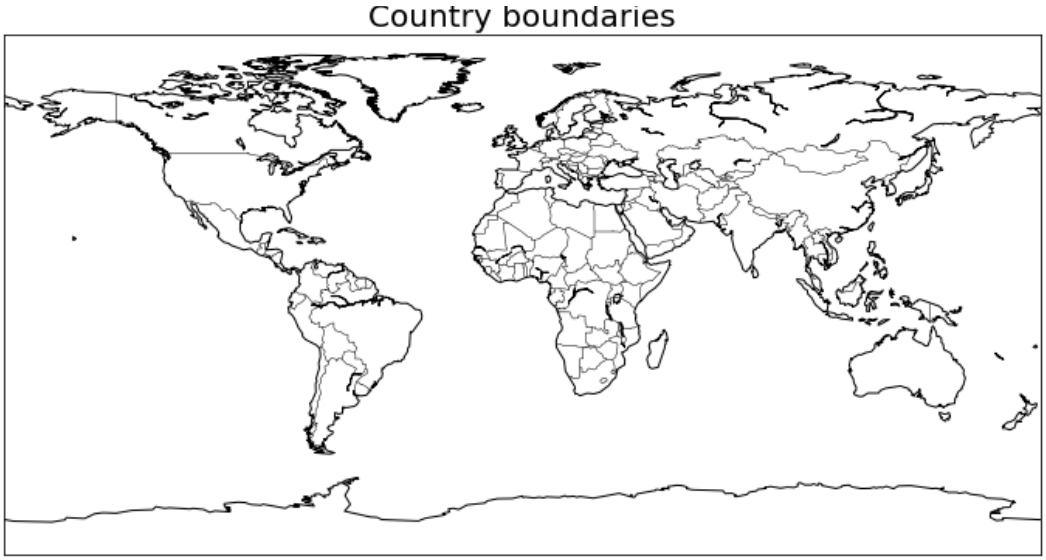
#Visualization of coastlines

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (12,12))
m = Basemap()
m.drawcoastlines()
plt.title("Coastlines", fontsize=20)
plt.show()
```



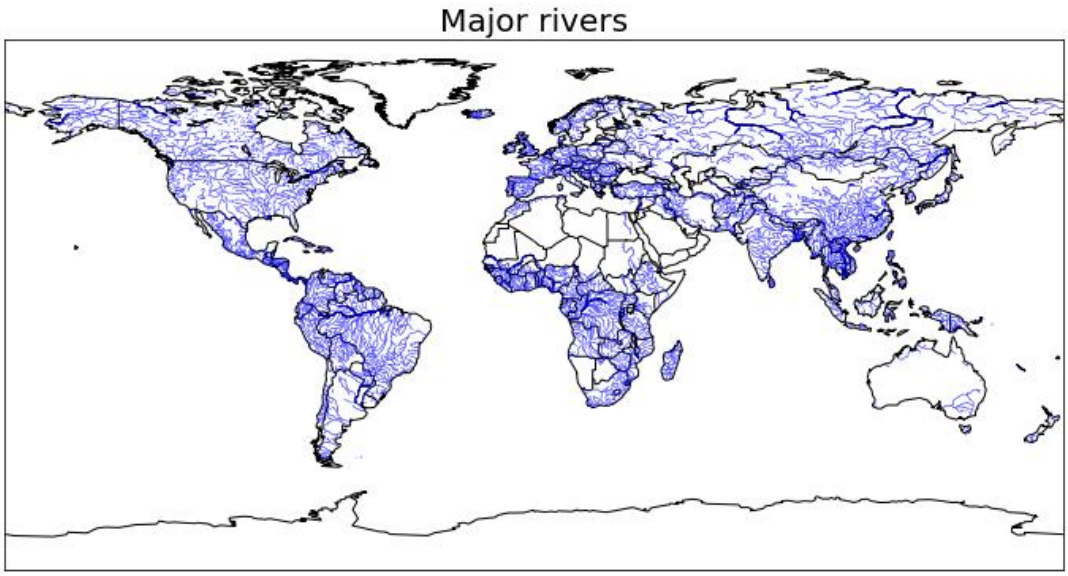
#Visualization of country boundaries

```
fig = plt.figure(figsize = (12,12))
m = Basemap()
m.drawcoastlines(linewidth=1.0, linestyle='solid', color='black')
m.drawcountries()
plt.title("Country boundaries", fontsize=20)
plt.show()
```



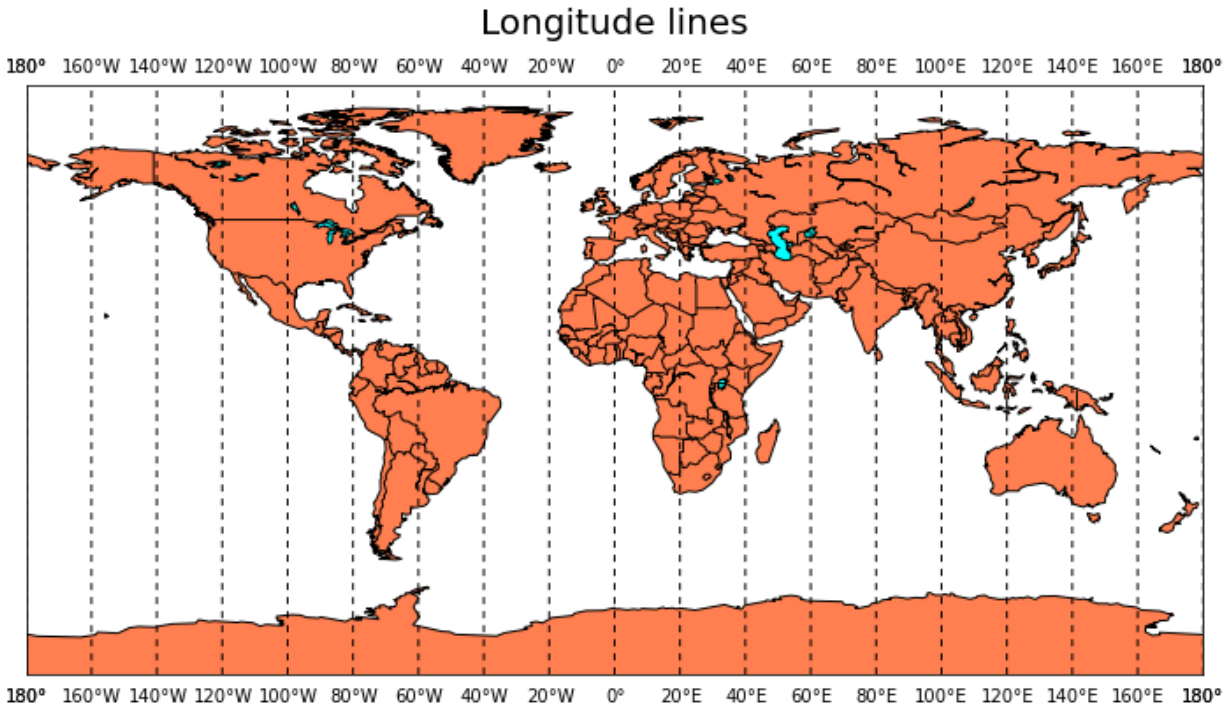
#Visualization of major rivers

```
fig = plt.figure(figsize = (12,12))
m = Basemap()
m.drawcoastlines(linewidth=1.0, linestyle='solid', color='black')
m.drawcountries(linewidth=1.0, linestyle='solid', color='k')
m.drawrivers(linewidth=0.5, linestyle='solid', color='#0000ff')
plt.title("Major rivers", fontsize=20)
plt.show()
```



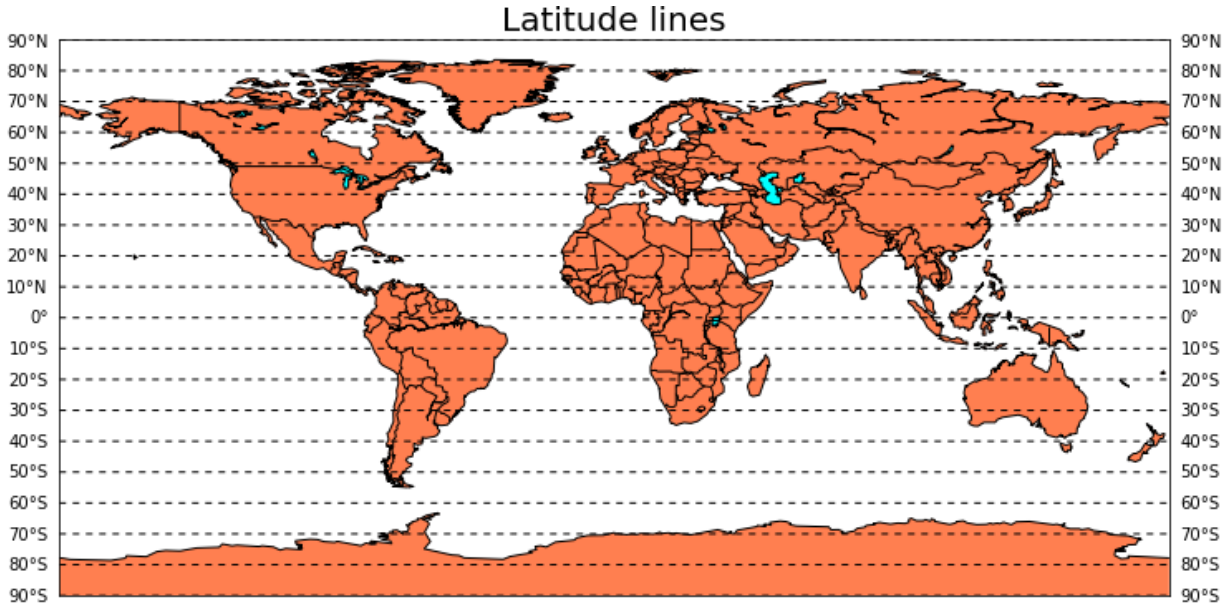
#Visualization of longitudinal lines

```
fig = plt.figure(figsize = (12,12))
m = Basemap()
m.drawcoastlines(linewidth=1.0, linestyle='solid', color='black')
m.drawcountries(linewidth=1.0, linestyle='solid', color='k')
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmeridians(range(0, 360, 20), color='k', linewidth=1.0, dashes=[4, 4], labels=[0, 0, 1, 1])
plt.title("Longitude lines", fontsize=20, pad=30)
plt.show()
```



#Visualization of latitudinal lines

```
fig = plt.figure(figsize = (12,12))
m = Basemap()
m.drawcoastlines(linewidth=1.0, linestyle='solid', color='black')
m.drawcountries(linewidth=1.0, linestyle='solid', color='k')
m.fillcontinents(color='coral',lake_color='aqua')
m.drawparallels(range(-90, 100, 10), color='k', linewidth=1.0, dashes=[4, 4], labels=[1, 1, 0, 0])
plt.title("Latitude lines", fontsize=20)
plt.show()
```



#Visualization of the country India in the map.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig=plt.figure(figsize=(8,8))
m=Basemap(projection='lcc',resolution=None,width=8E6,height=8E6,lat_0=20,lon_0=78)
m.etopo(scale=0.5,alpha=0.5)
x,y=m(78.9,20.5)
plt.plot(x,y,'ok',markersize=5)
plt.text(x,y,'INDIA',fontsize=12);
```

#Cylindrical projections

```
from itertools import chain
def draw_map(m, scale=0.2):
    # draw a shaded-relief image
    m.shadedrelief(scale=scale)
    # lats and longs are returned as a dictionary
    lats = m.drawparallels(np.linspace(-90, 90, 13))
    lons = m.drawmeridians(np.linspace(-180, 180, 13))
    # keys contain the plt.Line2D instances
    lat_lines = chain(*(tup[1][0] for tup in lats.items()))
    lon_lines = chain(*(tup[1][0] for tup in lons.items()))
    all_lines = chain(lat_lines, lon_lines)
    # cycle through these lines and set the desired style
    for line in all_lines:
        line.set(linestyle='-', alpha=0.3, color='w')
```

```
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.basemap import Basemap

fig=plt.figure(figsize=(8,6),edgecolor='w')

m=Basemap(projection='cyl',resolution=None,llcrnrlat=-90,urcrnrlat=90,llcrnrlon=-180,urcrnrlon=180,)

draw_map(m)
```

#Conic projections

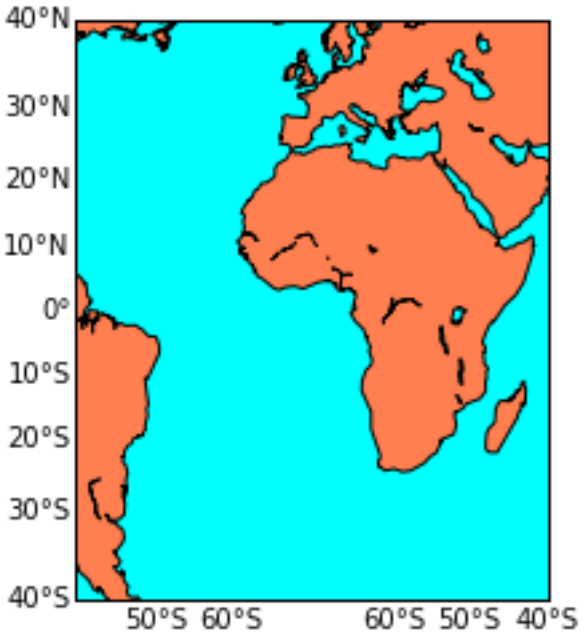
```
fig = plt.figure(figsize=(8, 8))

m = Basemap(projection='lcc', resolution=None,

        lon_0=0, lat_0=50, lat_1=45, lat_2=55,

        width=1.6E7, height=1.2E7)

draw_map(m)
```

#Perspective projections

```
map = Basemap(projection='ortho',lat_0=50, lon_0=-100)
map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color= 'coral',lake_color='aqua')
map.drawcoastlines()
map.drawcountries()
plt.show()
```

#Projection of latitude and longitude information of a particular region

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
map = Basemap(projection='poly', lon_0=0.0, lat_0=0,
llcrnrlon=-80.,llcrnrlat=-40,urcrnrlon=80.,urcrnrlat=40.)
map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='coral',lake_color='aqua')
map.drawcoastlines()
map.drawparallels(range(-90, 100, 10), linewidth=2, dashes=[4, 2], labels=[1,0,0,1], color='r', zorder=0 )
plt.show()
```



Result:

Thus the visualization of geographical data using basemap has been executed successfully.