EXP NO: 1 **DATE:**

PACKAGES FOR DATA SCIENCE IN PYTHON

AIM:

To download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels, and Pandas packages in Python.

PROCEDURE:

1. **NumPy**

NumPy (Numerical Python) is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

Installation: You can install NumPy using pip: pip install Numpy
Features: Arrays: Creating and manipulating arrays (np.array()).
Mathematical Functions: Array operations, linear algebra (np.linalg), random number generation (np.random). **File I/O:** Reading/writing data from/to disk.

2. **SciPy**

SciPy builds on NumPy and provides a collection of algorithms and functions for scientific and engineering applications. It includes modules for optimization, integration, interpolation, linear algebra, statistics, and more.
Installation: Install SciPy using pip: pip install scipy
Features: Integration and Optimization: Integration (scipy.integrate), optimization (scipy.optimize).
Statistics: Statistical functions (scipy.stats).
Interpolation: Interpolation methods (scipy.interpolate).
Signal Processing: Signal processing tools (scipy.signal).

3. **Jupyter**

Jupyter is a powerful tool for interactive computing. It allows you to create and share documents containing live code, equations, visualizations, and narrative text. Jupyter Notebook is the most popular interface.
Installation: Install Jupyter using pip: pip install jupyter
Features: Notebooks: Create and run notebooks with live code.
Markdown Support: Add formatted text, equations, and images.
Kernel Support: Run code in different programming languages (Python, R, Julia, etc.).
Visualization: Inline plotting with libraries like Matplotlib.

4. **Statsmodels**

Statsmodels is a Python module that provides classes and functions for estimating many different statistical models and performing statistical tests. It complements SciPy's statistical capabilities with more specialized models.
Installation: Install Statsmodels using pip: pip install statsmodels

Features: Statistical Models: Regression models (statsmodels.regression), time series analysis (statsmodels.tsa), ANOVA, etc. **Statistical Tests:** Hypothesis tests (statsmodels.stats).

Visualization: Plotting capabilities for diagnostics and model results.

5. **Pandas**

Pandas is a powerful data analysis and manipulation library for Python. It provides easyto- use data structures and data analysis tools for handling structured data.

Installation: Install Pandas using pip: pip install pandas

Features: DataFrame: Data structure for labeled data with rows and columns.

Data Manipulation: Filtering, merging, reshaping (groupby, pivot_table).

Data Input/Output: Read and write data from various file formats (CSV, Excel, SQL databases).

Time Series: Handling time series data effectively.

Getting Started:

Install Packages: Use pip to install the packages (numpy, scipy, jupyter, statsmodels, pandas).

Explore Documentation: Visit the official documentation for each package to explore their functionalities and usage examples.

Practice: Start coding with simple examples to familiarize yourself with each package's capabilities.

Combine: Often, these packages work together. For example, use Pandas for data manipulation, NumPy for numerical operations, SciPy for statistical tests, Statsmodels for modeling, and Jupyter for interactive analysis.

RESULT:

Thus, the features of NumPy, SciPy, Jupyter, Statsmodels, and Pandas packages were downloaded, installed, and explored.

EXP NO: 2 **DATE:**

BASIC NUMPY OPERATIONS

AIM:

To perform basic NumPy operations in python for
a) creating a NumPy program to create Arrays
b) creating a NumPy program to work with Arrays
c) creating a NumPy program to work with Arrays
d) creating a NumPy program to operations with matrices

A) **Write a NumPy program to create Arrays**

a) **One Dimensional Array:**

PROGRAM:

```
import numpy as np
arr = np.array([3,4,5,6,7])
print(arr)
```

OUTPUT:

```
[3 4 5 6 7]
```

b) **Two Dimensional Array:**

PROGRAM:

```
import numpy as np
arr = np.array([[4, 6, 8], [2, 10, 12]])
print(arr)
```

OUTPUT:

```
[[4 6 8] [2 10 12]]
```

c) **Index the Value at Position 0:**

PROGRAM:

```
import numpy as np
arr = np.array([2,3,4,5])
print(arr[0])
```

OUTPUT:

```
2
```

d) Add the Values at Position 2 and 3:

PROGRAM:

```
import numpy as np
arr = np.array([1,3,5,7,9])
print(arr[1] + arr[2])
```

OUTPUT:

```
8
```

e) **Access the Element on the 2nd Row, 5th Column:**

PROGRAM:

```
import numpy as np
arr = np.array([[0,2,4,6,8], [1,3,5,7,9]])
print(‘2nd row 5th element: ', arr[1, 4])
```

OUTPUT:

2nd row 5th element: 9

f) **Access the element on the 1st row, 2nd column:**

PROGRAM:

```
import numpy as np
arr = np.array([[0,2,4,6,8], [1,3,5,7,9]])
print('1st row 2nd element: ', arr[0, 1])
```

OUTPUT:

1st row 2nd element: 2

B) Write a Numpy program to work with Arrays

a) **Indexing Array**

PROGRAM:

```
import numpy as np
arr = np.array([3,5,7,9,11])
print(arr[1])
```

OUTPUT:

5

b) **Use negative indexing to access an array from the end:**
PROGRAM:

```
import numpy as np
arr = np.array([[0,2,4,6,8], [1,3,5,7,9]]
print('Last element from 2nd row: ', arr[1, -1])
```

OUTPUT:

Last element from 2nd row: 9

c) Slice elements from index 1 to index 5 from the array:

PROGRAM:

```
import numpy as np
arr = np.array([1,3,5,7,9,11,12])
print(arr[1:5])
```

OUTPUT:

[3 5 7 9]

d) Negative Slicing - Slice from the index 3 from the end to index 1 from the end:

PROGRAM:

```
import numpy as np
 arr = np.array([1,3,5,7,9,11,12])
print(arr[-3:-1])
```

OUTPUT:

[9 11]

e)Print the shape of an array:

PROGRAM:

```
import numpy as np
arr = np.array([[9, 3, 1], [2, 4, 6]])
print(arr.shape)
```

OUTPUT:

```
(2, 3)
```

e) Split the array in 3 parts:

PROGRAM:

```
import numpy as np
arr = np.array([[9, 3, 1], [2, 4, 6]])
print('1st row 2nd element: ', arr[0, 1])
```

OUTPUT:

```
1st row 2nd element: 3
```

C) **Write a Numpy program to work with arrays**

PROGRAM:

```
import numpy as np
a = np.array([[1, 3], [5, 7]])
b = np.array([[2, 4], [6, 8]])
while True:
    print("1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5. Dot product\n6. Exponentiation\n7. Logarithm\n8. Natural logarithm\n9. Exit")
    user_input = input("Enter the option number: ")
    try:
        n = int(user_input)
    except ValueError:
        print("Invalid input. Please enter a valid option number.")
        continue
    if 1 <= n <= 8:
        if n == 1:
            c = np.add(a, b)
            print("Sum:\n", c)
            print("\n")
        elif n == 2:
            d = np.subtract(a, b)
            print("Difference:\n", d)
            print("\n")
        elif n == 3:
            e = np.multiply(a, b)
            print("Product:\n", e)
             print("\n")
        elif n == 4:
            f = np.divide(a, b)
```

```
            print("Divide:\n", f)
            print("\n")
        elif n == 5:
            g = np.dot(a, b)
            print("Dot product:\n", g)
            print("\n")
        elif n == 6:
            h, i = np.exp(a), np.exp(b)
            print("Exponentiation for array a:\n", h)
            print("Exponentiation for array b:\n", i)
            print("\n")
        elif n == 7:
            l, m = np.log(a), np.log(b)
            print("Logarithm for array a:\n", l)
            print("Logarithm for array b:\n", m)
            print("\n")
        elif n == 8:
            x, y = np.log10(a), np.log10(b)
            print("Natural logarithm for array a:\n", x)
            print("Natural logarithm for array b:\n", y)
            print("\n")
        elif n == 9:
            break
        else:
            print("No such option exists. Please enter an existing
        option.\n")
```

OUTPUT:

```
1. Add
2. Subtract
3. Multiply
4. Divide
5. Dot product
6. Exponentiation
7. Logarithm
8. Natural logarithm
9. Exit
Enter the option number: 1
Sum:
[[ 3 7]
[11 15]]
 1. Add
 2. Subtract
 3. Multiply
```

4. Divide
5. Dot product
6. Exponentiation
7. Logarithm
8. Natural logarithm
9. Exit

Enter the option number: 8
Natural logarithm for array a:
[[0. 0.47712125]
[0.69897 0.84509804]]
Natural logarithm for array b:
[[0.30103 0.60205999]
[0.77815125 0.90308999]]
1.Add
2. Subtract
3. Multiply
4. Divide
5. Dot product
6. Exponentiation
7. Logarithm
8. Natural logarithm
9. Exit Enter the option number: 3
Product:
[[2 12]
[30 56]]
1. Add
2. Subtract
3. Multiply
4. Divide
5. Dot product
6. Exponentiation
7. Logarithm
8. Natural logarithm
9. Exit
Enter the option number: 9

D. **Write a Numpy program to operations with matrices**.

PROGRAM:

```
import numpy as np
def get_start_end_nmbr():
    s = int(input("Enter the starting value: "))
    e = int(input("Enter the end value: "))
    nmbr = int(input("Enter the number of values to be printed: "))
    return s, e, nmbr
```

```
def get_rows_columns():
        r = int(input("Enter the number of rows: "))
        c = int(input("Enter the number of columns: "))
        return r, c
while True:
        print("1. Create a sequence with linspace function")
        print("2. Create an n-dimensional array using random function")
        print("3. Create an n-dimensional array of zeros")
        print("4. Create an n-dimensional array of ones")
        print("5. Create an n-dimensional array using fill function")
        print("6. Exit")
        n = int(input("Enter the option: "))
        if 1 <= n <= 5:
                if n == 1:
                        s, e, nmbr = get_start_end_nmbr()
                         l = np.linspace(s, e, nmbr)
                        print("Generated sequence:\n", l)
                        print()
                elif n == 2:
                        r, c = get_rows_columns()
                        rndm = np.random.random((r, c))
                        print("Randomly created n-dimensional array:\n",
                rndm)
                        print()
                elif n == 3:
                        r, c = get_rows_columns()
                        z = np.zeros((r, c), dtype="int")
                        print("n-dimensional array of zeros:\n", z)
                        print()
                elif n == 4:
                        r, c = get_rows_columns()
                        o = np.ones((r, c), dtype="int")
                        print("n-dimensional array of ones:\n", o)
                        print()
                elif n == 5:
                        r, c = get_rows_columns()
                        f = np.full((r, c), 6)
                        print("n-dimensional array of given number:\n", f)
                        print()
                elif n == 6:
                        break
                else:
```

print("No such option exists. Please enter an existing option.\n")

OUTPUT:

```
1. Create a sequence with linspace function
2. Create an n-dimensional array using random function
3. Create an n-dimensional array of zeros
4. Create an n-dimensional array of ones
5. Create an n-dimensional array using fill function
6. Exit
Enter the option: 1
Enter the starting value: 1
Enter the end value: 12
Enter the number of values to be printed: 5
Generated sequence:
[ 1. 3.75 6.5 9.25 12. ]
1.Create a sequence with linspace function
2. Create an n-dimensional array using random function
3. Create an n-dimensional array of zeros
4. Create an n-dimensional array of ones
5. Create an n-dimensional array using fill function
6. Exit
Enter the option: 2
Enter the number of rows: 3
Enter the number of columns: 3
Randomly created n-dimensional array:
[[0.62388664 0.71755544 0.60403254]
[0.02656711 0.134686 0.1324508 ]
[0.33487147 0.40065194 0.3307965 ]]
1. Create a sequence with linspace function
2. Create an n-dimensional array using random function
3. Create an n-dimensional array of zeros
4. Create an n-dimensional array of ones
5. Create an n-dimensional array using fill function
6. Exit
Enter the option: 3
Enter the number of rows: 4
Enter the number of columns: 5
n-dimensional array of zeros:
[[0 0 0 0 0]
 [0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]]
1. Create a sequence with linspace function
```

2. Create an n-dimensional array using random function
3. Create an n-dimensional array of zeros
4. Create an n-dimensional array of ones
5. Create an n-dimensional array using fill function
6. Exit

Enter the option: 4
Enter the number of rows: 3
Enter the number of columns: 2
n-dimensional array of ones:
[[1 1]
[1 1] [
1 1]]

1. Create a sequence with linspace function
2. Create an n-dimensional array using random function
3. Create an n-dimensional array of zeros
4. Create an n-dimensional array of ones
5. Create an n-dimensional array using fill function
6. Exit Enter the option: 5

Enter the number of rows: 2
Enter the number of columns: 6
n-dimensional array of given number:
[[6 6 6 6 6 6]
[6 6 6 6 6 6]]

1. Create a sequence with linspace function
2. Create an n-dimensional array using random function
3. Create an n-dimensional array of zeros
4. Create an n-dimensional array of ones
5. Create an n-dimensional array using fill function
6. Exit

Enter the option: 6

RESULT:

Thus, the program to implement NumPy operations with arrays using Python has been executed and the output was verified successfully.

EXP NO: 3 **DATE:**

WORKING WITH PANDAS DATAFRAMES

AIM:

i) To create a dataframe from a series
ii) To create a datafram from a dictionary
iii) To create a datafram from n-dimensional arrays
iv) To load a dataset from an external source into a pandas dataframe

CREATION OF A DATAFRAME FROM A SERIES

PROGRAM:

```
import numpy as np
import pandas as pd
print("Pandas Version:", pd.__version__)
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
series = pd.Series([2, 3, 7, 11, 13, 17, 19, 23])
print("Series:\n", series)
series_df = pd.DataFrame({
        'A': range(1, 5),
        'B': pd.Timestamp('2024-09-22'),
        'C': pd.Series(5, index=list(range(4)), dtype='float64'),
        'D': np.array([3] * 4, dtype='int64'),
        'E': pd.Categorical(["Depression", "Social Anxiety", "Bipolar
Disorder", "Eating Disorder"]),
        'F': 'Mental health',
        'G': 'is challenging' })
print("\nDataFrame:\n", series_df)
```

OUTPUT:

```
Pandas Version: 2.1.4
Series:
0 2
1 3
2 7
3 11
4 13
5 17
6 19
7 23
dtype: int64
DataFrame:
A  B           C        D           E                   F                G
0  1  2024-09-22  5.0 3        Depression       Mental health    is challenging
1  2  2024-09-22  5.0 3    Social Anxiety       Mental health    is challenging
2  3  2024-09-22  5.0 3 Bipolar Disorder     Mental health    is challenging
3  4  2024-09-22  5.0 3    Eating Disorder    Mental health    is challenging
```

ii) CREATION OF A DATAFRAME FROM DICTIONARY

PROGRAM:

```
import numpy as np
import pandas as pd
dict_df = [{'A': 'Axe', 'B': 'Ball'},{'A': 'Ant', 'B':'Bat', 'C': 'Car'}]
dict_df = pd.DataFrame(dict_df)
print(dict_df)
```

OUTPUT:

```
    A     B     C
0  Axe   Ball   NaN
1  Ant   Bat    Car
```

iii) CREATION OF A DATAFRAME FROM N-DIMENSIONAL ARRAYS

PROGRAM:

```
import numpy as np
import pandas as pd
sdf = {
    'County': ['India', 'Ireland', 'Argentina', 'Australia', 'Brazil', 'Canada'],
    'ISO-Code': ['IND', 'IRE', 'ARG', 'AUS', 'BRA', 'CAN'],
    'Area': [4180.69, 4917.94, 454.07, 27397.76, 25192.10, 14910.94],
  'Administrative centre': ["New Delhi", "Dublin", "Buenos Aires",
"Canberra", "Brasília", "Ottawa"] }
sdf_df = pd.DataFrame(sdf)
print(sdf_df)
```

OUTPUT:

```
   County  ISO-Code   Area      Administrative centre
0  India       IND      4180.69     New Delhi
1  Argentina  ARG     4917.94      Buenos Aires
2  Ireland     IRE      454.07       Dublin
3  Brazil     BRA      27397.76     Brasília
4  Australia  AUS     25192.10     Canberra
5  Canada    CAN    14910.94     Ottawa
```

LOADING A DATASET FROM AN EXTERNAL SOURCE INTO A PANDAS DATAFRAME

PROGRAM:

```
import numpy as np
import pandas as pd
columns = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
'marital_status', 'occupation', 'relationship', 'ethnicity', 'gender', 'capital_gain',
'capital_loss', 'hours_per_week', 'country_of_origin', 'income']
df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data', names=columns)
print(df.head(10))
```

RESULT:

Thus, the programs for creating and loading pandas dataframes using Python have been implemented and the output was verified successfully.

EXP NO: 4 **DATE: <u>DESCRIPTIVE ANALYTICS WITH PANDAS ON IRIS DATA</u>**

<u>AIM</u>:

To perform descriptive analytics on the iris dataset by reading iris data from a CSV file, the web, and the sklearn datasets module

i) **<u>Descriptive analytics on the Iris dataset by reading data from a specific location in the computer or from web</u>**

CODE: for importing pandas to use in code as pd.

```
import pandas as pd
```

CODE: for reading data from CSV file

```
iris = pd.read_csv('iris.csv', delimiter = ',')
```

CODE: for reading data from URL

a.Create csv_url and pass to it the URL where the data set is available 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'.

```
csv_url='https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data'
```

b. Create a list of column names "col_names" using the iris attribute information.

```
# using the attribute information as the column names
col_names=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width',
'Class']
```

c. Create a panda's DataFrame object called iris.

```
iris = pd.read_csv(csv_url, names = col_names)
```

CODE: to display the top rows of the dataset with their columns

```
# Default value of head() function is 5, that is, it shows top 5 rows when no
argument is given
iris.head()
```

OUTPUT:

	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

CODE: to display the specified number of rows randomly

```
iris.sample(10)
```

OUTPUT:

	sepal_length	sepal_width	petal_length	petal_width	species
61	5.9	3.0	4.2	1.5	versicolor
103	6.3	2.9	5.6	1.8	virginica
102	7.1	3.0	5.9	2.1	virginica
31	5.4	3.4	1.5	0.4	setosa
105	7.6	3.0	6.6	2.1	virginica
92	5.8	2.6	4.0	1.2	versicolor
128	6.4	2.8	5.6	2.1	virginica
109	7.2	3.6	6.1	2.5	virginica
113	5.7	2.5	5.0	2.0	virginica
51	6.4	3.2	4.5	1.5	versicolor

CODE: to display the number of columns and names of the columns.

iris.columns

OUTPUT:

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

CODE: to display the shape of the dataset

Displays number of rows and columns.

iris.shape

OUTPUT:

(150, 5)

CODE: to display the whole dataset

iris

OUTPUT:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
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

CODE: to slice the rows

Prints the rows from 10 to 20

iris[10:21]

OUTPUT:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
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

CODE: Display the number of instances and attributes in the dataset

```
# Demonstrates a complete dataset - no null values
iris.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

CODE: Display the number of instances of each species

```
# Shows a balanced dataset - each type is equally represented
iris.groupby('species').size()
```

OUTPUT:

```
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

CODE: Display the datatypes of each of the attributes

```
#The columns of the resulting DataFrame have different dtypes.
iris.dtypes
```

OUTPUT:

```
Sepal_Length    float64
Sepal_Width     float64
Petal_Length    float64
Petal_Width     float64
Class            object
dtype: object
```

CODE: Display basic statistical features of the dataset

```
iris.describe()
```

OUTPUT:

	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

CODE: Count the number of rows in the dataset

```
iris.count()
```

OUTPUT:

```
sepal_length    150
sepal_width     150
petal_length    150
petal_width     150
species         150
dtype: int64
```

CODE: Number of counts of unique values using "value_counts()"

```
iris["species"].value_counts()
```

OUTPUT:

```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

CODE: To calculate sample mean for every numeric column

```
# Sample mean for every numeric column
iris.mean()
```

OUTPUT:

```
sepal_length    5.843333
sepal_width     3.054000
petal_length    3.758667
petal_width     1.198667
dtype: float64
```

CODE: To calculate sample median for every numeric column

```
# Sample mean for every numeric column
iris.median()
```

OUTPUT:

```
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
dtype: float64
```

ii) <u>Descriptive analytics on the Iris dataset by reading data from the scikit-learn datasets module</u>

CODE: Load iris dataset from scikit learn datasets

module from sklearn.datasets import load_iris

iris= load_iris()

CODE: Store features matrix in X

X= iris.data

CODE: Store target vector in y

y= iris.target

CODE: Names of features/columns in iris dataset

iris.feature_names

OUTPUT:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

CODE: Display names of target/output in iris dataset

print(iris.target_names)

OUTPUT:

```
['setosa' 'versicolor' 'virginica']
```

CODE: Examine the size of feature matrix

print(iris.data.shape)

OUTPUT:

(150, 4)

CODE: Display the size of target vector

print(iris.target.shape)

OUTPUT:

(150,)

CODE: Display the contents of the data

print(iris.data)

OUTPUT:

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
```

```
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
```

```
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
```

```
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

CODE: Display target vector iris species: 0 = setosa, 1 = versicolor, 2 = virginica

```
print(iris.target)
```

OUTPUT:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

CODE: Convert into dataframe

```
import pandas as pd
import numpy as np
```

```
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],columns= iris['feature_names'] + ['Species'])
# Distribution of each Iris species
df['Species'].value_counts()
```

OUTPUT:

```
0.0    50
1.0    50
2.0    50
Name: Species, dtype: int64
```

CODE: Display basic statistical features of the dataset

```
df.describe()
```

OUTPUT:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

RESULT:

Thus, the code for reading data from CSV file, web, and sklearn package was executed and various commands for doing descriptive analytics on the Iris data set were executed and the output is verified.

EX.NO.:5a **DATE:**

UNIVARIATE STATISTICAL ANALYSIS ON DIABETES DATA

AIM:

To perform the univariate statistical analysis by calculating frequency, mean, median, mode, variance, standard deviation, skewness, and kurtosis on the Pima diabetes dataset.

ALGORITHM:

Step 1: Open the Anaconda prompt and type “jupyter notebook”.
Step 2: Create a new notebook and save it.
Step 3: Import the required packages.
Step 4: Read the Pima diabetes dataset (pima_diabetes.csv).
Step 5: Type the commands for statistical analysis of the diabetes data.
Step 6: Display the output.
Step 7: Terminate the program.

PROGRAM:

Code: Import the packages

```
import pandas as pd
import numpy as np
import statistics as st
```

Code: Load the pima_diabetes data

```
df=pd.read_csv("pima_diabetes.csv")
```

Code: Shape of the dataset

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

Output:

```
(768, 9)
```

Code: Display the number of instances and attributes in the dataset

```
print(df.info())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

Code: Mean of the numerical variables in the data

```
df.mean()
```

Output:

```
Pregnancies                   3.845052
Glucose                     120.894531
BloodPressure                69.105469
SkinThickness                20.536458
Insulin                      79.799479
BMI                          31.992578
DiabetesPedigreeFunction      0.471876
Age                          33.240885
Outcome                       0.348958
dtype: float64
```

Code: Calculate the mean of the variables 'Pregnancies' and 'Glucose'

```
print(df.loc[:,'Pregnancies'].mean())
```

print(df.loc[:,'Glucose'].mean())

Output:
3.8450520833333335
120.89453125

Code: Calculate the mean of the first five rows
df.mean(axis = 1)[0:5]

Output:
```
0    38.469667
1    26.550111
2    34.663556
3    35.807444
4    51.043111
dtype: float64
```

Code: Median of the numerical variables in the data
df.median()

Output:
```
Pregnancies                   3.0000
Glucose                     117.0000
BloodPressure                72.0000
SkinThickness                23.0000
Insulin                      30.5000
BMI                          32.0000
DiabetesPedigreeFunction      0.3725
Age                          29.0000
Outcome                       0.0000
dtype: float64
```

Code: Calculate median of a particular column
print(df.loc[:,'Pregnancies'].median())
print(df.loc[:,'Glucose'].median())

Output:
3.0
117.0

Code: Calculate the median of the first five rows
df.median(axis = 1)[0:5]

Output:
```
0    33.6
1    26.6
2     8.0
3    23.0
4    35.0
dtype: float64
```

Code: Compute the mode of all the variables in the data
df.mode()

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1.0	99	70.0	0.0	0.0	32.0	0.254	22.0	0.0
1	NaN	100	NaN	NaN	NaN	NaN	0.258	NaN	NaN

Code: Compute standard deviation of all the numerical variables in the data
df.std()

Output:
```
Pregnancies                   3.369578
Glucose                      31.972618
BloodPressure                19.355807
SkinThickness                15.952218
Insulin                     115.244002
BMI                           7.884160
DiabetesPedigreeFunction      0.331329
Age                          11.760232
Outcome                       0.476951
dtype: float64
```

Code: Calculate the standard deviation of a particular variable print(df.loc[:,' Pregnancies'].std())
print(df.loc[:,' Glucose '].std())

Output:
3.3695780626988623
31.97261819513622

Code: Calculate the standard deviation for the first five rows
df.std(axis = 1)[0:5]

Output:
```
0    48.296112
1    31.119744
2    59.585320
3    37.639873
4    60.541569
dtype: float64
```

# Print the variance of all the numerical variables in the dataset
df.var()

Output:

```
Pregnancies                    11.354056
Glucose                      1022.248314
BloodPressure                 374.647271
SkinThickness                 254.473245
Insulin                     13281.180078
BMI                            62.159984
DiabetesPedigreeFunction        0.109779
Age                           138.303046
Outcome                         0.227483
dtype: float64
```

Code: Calculate the skewness of the numerical variables using the skew() function

print(df.skew())

Output:

```
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
```

Code: Calculate the kurtosis of the numerical variables using the kurtosis() function

print(df.kurtosis())

Output:

```
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 code for performing univariate statistical analysis on the Pima diabetes dataset was executed using python and the output is verified.

EX. NO.:5b **DATE:**

BIVARIATE ANALYSIS ON DIABETES DATA

(i) BIVARIATE ANALYSIS USING LINEAR REGRESSION

AIM:

To perform the bivariate analysis using Linear Regression on the Pima diabetes dataset.

ALGORITHM:

Step 1: Load the relevant libraries, such as pandas and statsmodels.
Step 2: Read the data using the pandas read_csv( ) function from local storage and save it in a variable called "data".
Step 3: Create a correlation matrix.
Step 4: Define the response variable and explanatory variable
Step 5: Fit a linear regression model.
Step 6: Generate the model summary table and interpret the model coefficients.

PROGRAM:

```
import pandas as pd
import statsmodels.api as sm
data = pd.read_csv("pima_diabetes.csv")
#create correlation matrix
data.corr()

#Bivariate Analysis of Glucose-Insulin features
#define response variable 1
y1 = data['Glucose']

#define explanatory variable 1
x1 = data[['Insulin']]

#add constant to predictor variables
x1 = sm.add_constant(x1)

#fit linear regression model
model1 = sm.OLS(y1, x1).fit()

#view model summary
print(model1.summary())

#Bivariate Analysis of Age-Pregnancies features
#define response variable 2
```

```
y2 = data['Age']

#define explanatory variable 2
x2 = data['Pregnancies']

#add constant to predictor variables x
2 = sm.add_constant(x2)

#fit linear regression model
model2 = sm.OLS(y2, x2).fit()

#view model summary
print(model2.summary())
```

#Bivariate Analysis of SkinThickness-BMI features

```
#define response variable 3
y3 = data['SkinThickness']

#define explanatory variable 3
x3 = data[['BMI']]

#add constant to predictor variables
x3 = sm.add_constant(x3)

#fit linear regression model
Model3 = sm.OLS(y3, x3).fit()

#view model summary
print(model3.summary())
```

OUTPUT:

a. Correlation Matrix

	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

b. Bivariate Analysis of Glucose-Insulin features

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Glucose   R-squared:                       0.110
Model:                            OLS   Adj. R-squared:                  0.109
Method:                 Least Squares   F-statistic:                     94.48
Date:                Mon, 17 Oct 2022   Prob (F-statistic):           3.88e-21
Time:                        18:55:34   Log-Likelihood:                -3705.6
No. Observations:                 768   AIC:                             7415.
Df Residuals:                     766   BIC:                             7425.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        113.5586      1.325     85.694      0.000     110.957     116.160
Insulin        0.0919      0.009      9.720      0.000       0.073       0.110
==============================================================================
Omnibus:                       24.112   Durbin-Watson:                   1.910
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               35.923
Skew:                           0.279   Prob(JB):                     1.58e-08
Kurtosis:                       3.901   Cond. No.                         170.
==============================================================================
```

c. Bivariate Analysis of Age-Pregnancies features

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    Age   R-squared:                       0.296
Model:                            OLS   Adj. R-squared:                  0.295
Method:                 Least Squares   F-statistic:                     322.5
Date:                Mon, 17 Oct 2022   Prob (F-statistic):           1.86e-60
Time:                        19:05:57   Log-Likelihood:                -2847.2
No. Observations:                 768   AIC:                             5698.
Df Residuals:                     766   BIC:                             5708.
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const          25.9360      0.541     47.970      0.000      24.875      26.997
Pregnancies     1.8998      0.106     17.959      0.000       1.692       2.107
==============================================================================
Omnibus:                      245.469   Durbin-Watson:                   2.065
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              628.234
Skew:                           1.664   Prob(JB):                    3.81e-137
Kurtosis:                       5.924   Cond. No.                         7.93
==============================================================================
```

d. Bivariate Analysis of SkinThickness-BMI features

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          SkinThickness   R-squared:                       0.154
Model:                            OLS   Adj. R-squared:                  0.153
Method:                 Least Squares   F-statistic:                     139.6
Date:                Mon, 17 Oct 2022   Prob (F-statistic):           1.05e-29
Time:                        19:10:07   Log-Likelihood:                -3152.0
No. Observations:                 768   AIC:                             6308.
Df Residuals:                     766   BIC:                             6317.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -4.8753      2.215     -2.201      0.028      -9.224      -0.526
BMI            0.7943      0.067     11.814      0.000       0.662       0.926
==============================================================================
Omnibus:                        9.542   Durbin-Watson:                   1.938
Prob(Omnibus):                  0.008   Jarque-Bera (JB):                9.612
Skew:                          -0.273   Prob(JB):                      0.00818
Kurtosis:                       3.045   Cond. No.                         138.
==============================================================================
```

RESULT: Thus the bivariate analysis using Linear Regression was performed successfully on the Pima diabetes dataset and the output is verified.

(ii) BIVARIATE ANALYSIS USING LOGISTIC REGRESSION

AIM:

To perform the bivariate analysis using Logistic Regression on the Pima diabetes dataset.

ALGORITHM:

Step 1: Load the relevant libraries, such as pandas and statsmodels.
Step 2: Read the data using the pandas read_csv( ) function from local storage and save it in a variable called "data".
Step 3: Fit a logistic regression model.
Step 4: Generate the model summary table and interpret the model coefficients.

PROGRAM:

```
# importing libraries
import statsmodels.api as sm
import pandas as pd

# loading the training dataset
data = pd.read_csv('pima_diabetes.csv', index_col = 0)

# defining the dependent and independent variables
Xtrain = data[['Glucose', 'BloodPressure' , 'SkinThickness' , 'Insulin' , 'BMI',
    'DiabetesPedigreeFunction','Age']]
ytrain = data[['Outcome']]

# building the model and fitting the data
log_reg = sm.Logit(ytrain, Xtrain).fit()

# printing the summary table
print(log_reg.summary())
```

OUTPUT:

```
Optimization terminated successfully.
         Current function value: 0.622121
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:                Outcome   No. Observations:                  768
Model:                          Logit   Df Residuals:                      761
Method:                           MLE   Df Model:                            6
Date:                Mon, 17 Oct 2022   Pseudo R-squ.:                 0.03815
Time:                        19:32:45   Log-Likelihood:                -477.79
converged:                       True   LL-Null:                       -496.74
Covariance Type:            nonrobust   LLR p-value:                 1.172e-06
============================================================================================
                               coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Glucose                      0.0122      0.003      4.579      0.000       0.007       0.017
BloodPressure               -0.0298      0.005     -6.404      0.000      -0.039      -0.021
SkinThickness             1.809e-05      0.006      0.003      0.998      -0.012       0.012
Insulin                      0.0006      0.001      0.772      0.440      -0.001       0.002
BMI                         -0.0059      0.011     -0.562      0.574      -0.027       0.015
DiabetesPedigreeFunction     0.2486      0.237      1.051      0.293      -0.215       0.712
Age                          0.0040      0.007      0.573      0.567      -0.010       0.018
============================================================================================
```

RESULT:

Thus the bivariate analysis using Logistic Regression was performed successfully on the Pima diabetes dataset and the output is verified.

EX. NO.:5c **DATE:**

MULTIPLE REGRESSION ANALYSIS ON DIABETES DATA

AIM:

To perform the multiple regression analysis on the Pima diabetes dataset.

ALGORITHM:

Step 1: Import the modules and packages.
Step 2: Read the diabetes data using the pandas read_csv( ) function and save it in variable "df".
Step 3: Create feature variables.
Step 4: Define the response variable and explanatory variables.
Step 5: Fit a linear regression model.
Step 6: Generate the model summary table and interpret the model coefficients.

PROGRAM:

```
# importing modules and packages
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import statsmodels.api as ssm

# importing data
df = pd.read_csv('pima_diabetes.csv')

# creating feature variables
X = df.drop('Outcome', axis=1)
Y = df['Outcome']

X=ssm.add_constant(X)           #to add constant value in the model
model= ssm.OLS(Y,X).fit()       #fitting the model
predictions= model.summary()  #summary of the model

predictions
```

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:	Tue, 25 Oct 2022	**Prob (F-statistic):**	7.36e-55
Time:	19:11:28	**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

RESULT:

Thus the multiple regression analysis was performed successfully on the Pima diabetes dataset and the output is verified.

EX. NO.:6 **DATE:**

APPLICATION OF PLOTTING FUNCTIONS ON UCI DATASETS

AIM:

To apply and explore various plotting functions using matplotlib and seaborn packages on the Adult UCI dataset.

a. Normal curves
b. Density and contour plots
c. Correlation and scatter plots
d. Histograms
e. Three-dimensional plotting

a) NORMAL CURVES

ALGORITHM:

Step 1: Install seaborn using the command: $ conda install seaborn
Step 2: Load the relevant libraries, such as numpy, pandas, matplotlib and seaborn.
Step 3: Read the adult UCI data and save it as dataFrame(df).
Step 4: Generate curves using the categorical and relational plot functions.
Step 5: Show the output.

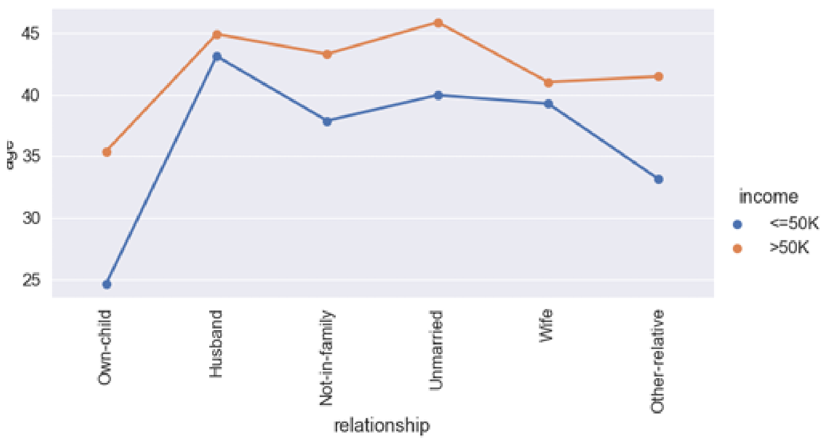
PROGRAM:

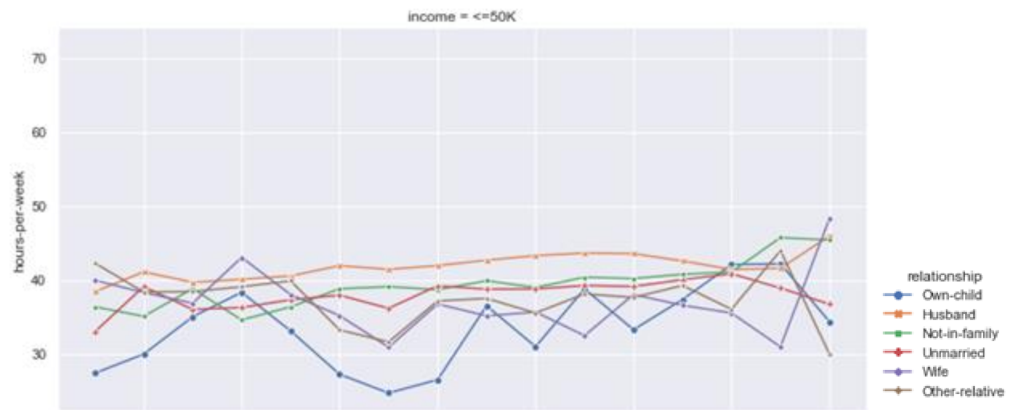
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

df = pd.read_csv("adult.csv")

#Check the structure of the data df.info()
sns.set(font_scale=1.5)
sns.catplot(x="relationship", y="age", data=df,
            kind="point",hue='income',
            capsize=0.4,ci=None,aspect=2)

# Show plot
plt.xticks(rotation=90)
plt.show()

 sns.set(font_scale=1)
```

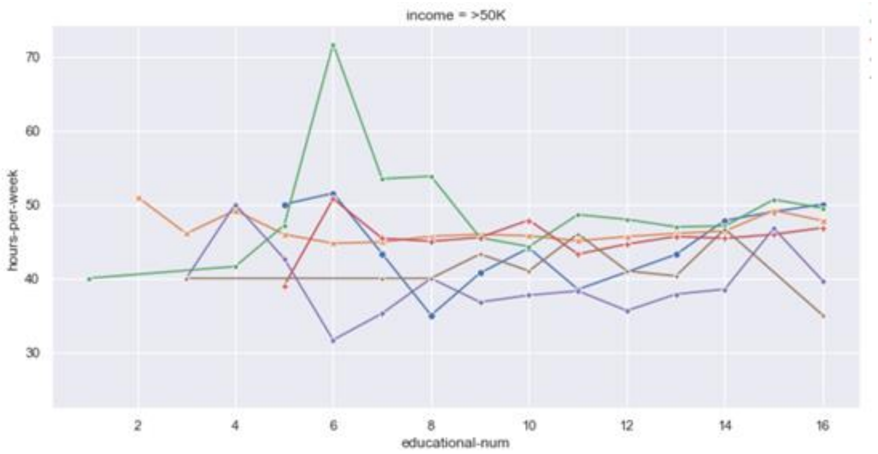
```
sns.relplot(x="educational-num", y="hours-per-week", data=df,
    kind="line",row='income' , ci=None, hue="relationship",
    style="relationship",markers=True,dashes=False,aspect=2)

# Show plot
plt.show()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```







b) DENSITY AND CONTOUR PLOTS

ALGORITHM:

Step 1: Load the relevant libraries, such as numpy, pandas, matplotlib and seaborn.

Step 2: Read the adult UCI data and save it as dataFrame(df).

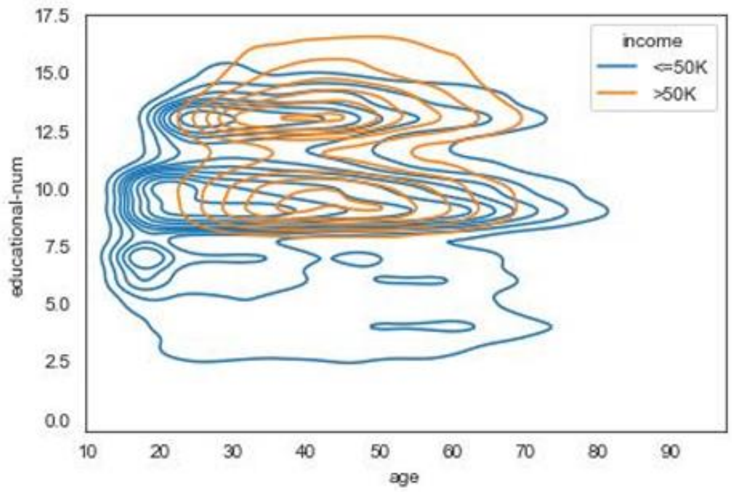
Step 3: Plot univariate or bivariate distributions using kernel density estimation.

Step 4: Map a third variable with a hue semantic to show conditional distributions.

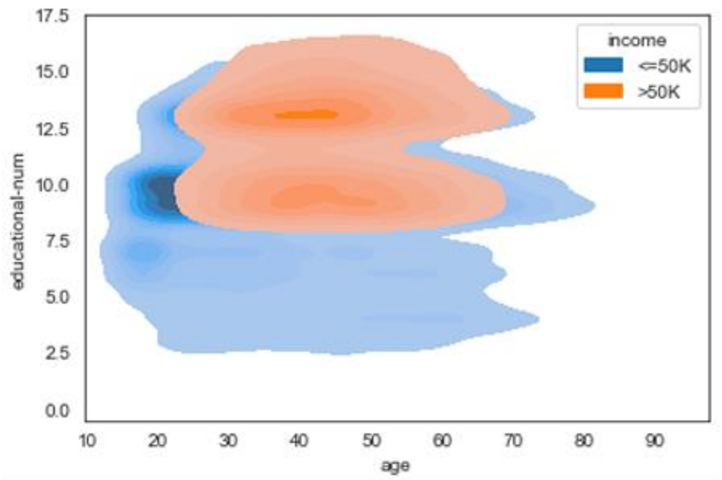
Step 5: Show the filled contours by setting fill=True. Step 6: Display the output.

PROGRAM:

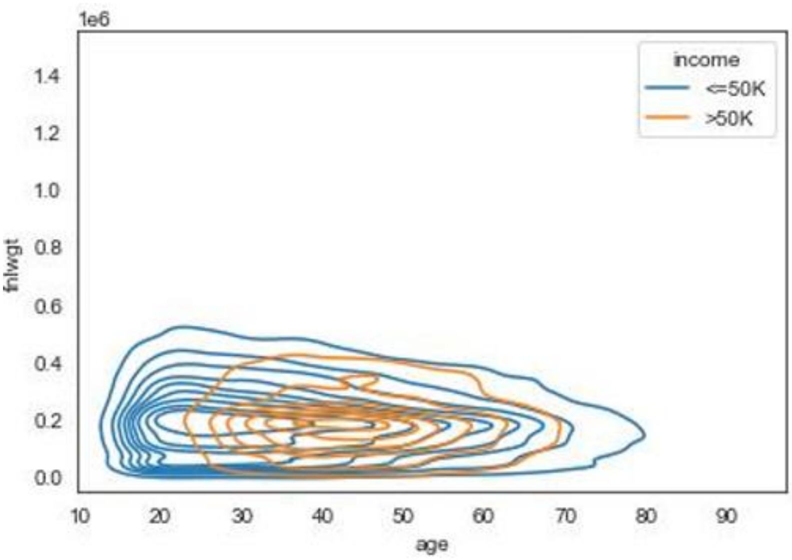
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.simplefilter(action="ignore", category=FutureWarning)
df = pd.read_csv("adult.csv")

# set seaborn style
sns.set_style("white")

#Map a third variable “income” with a hue semantic to show conditional distributions
sns.kdeplot(data=df, x="age", y="educational-num", hue="income")
```



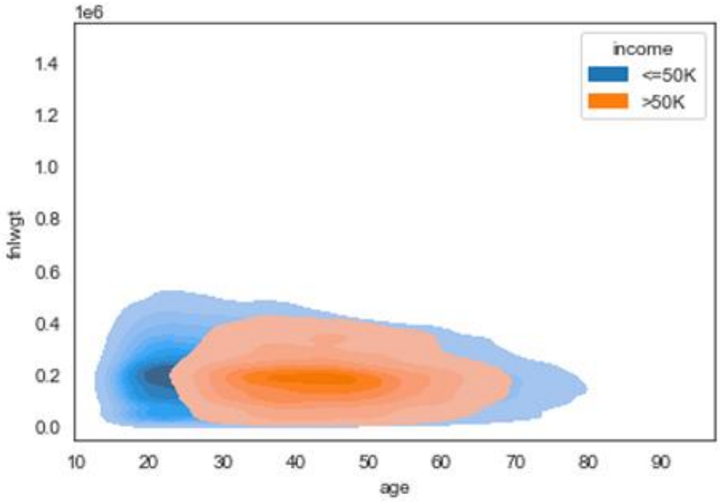
#Show filled contours
sns.kdeplot(data=df, x="age", y="educational-num", hue="income", fill=True)



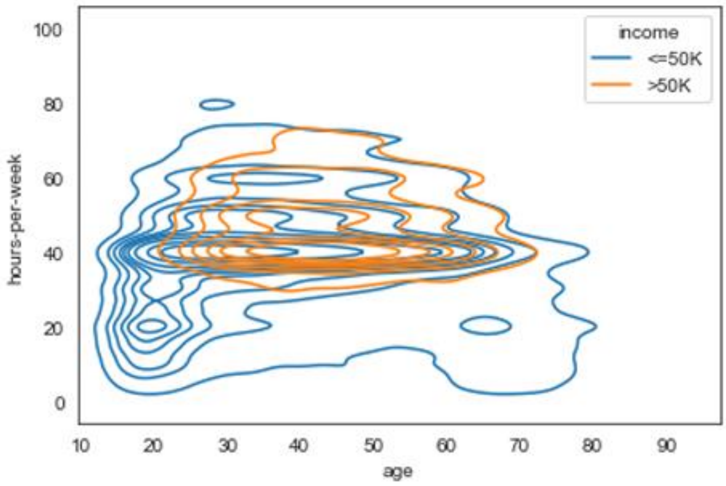
sns.kdeplot(data=df, x="age", y="fnlwgt", hue="income")



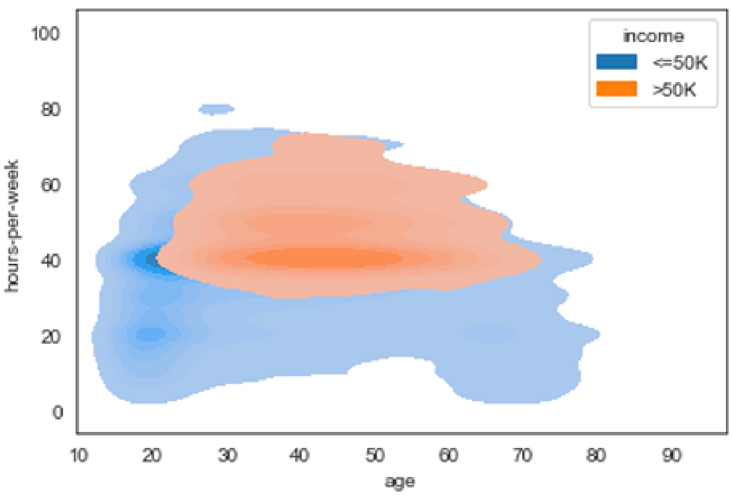
```
sns.kdeplot(data=df, x="age", y="fnlwgt", hue="income", fill=True)
```




```
sns.kdeplot(data=df, x="age", y="hours-per-week", hue="income")
```




```
sns.kdeplot(data=df, x="age", y="hours-per-week", hue="income", fill=True)
```



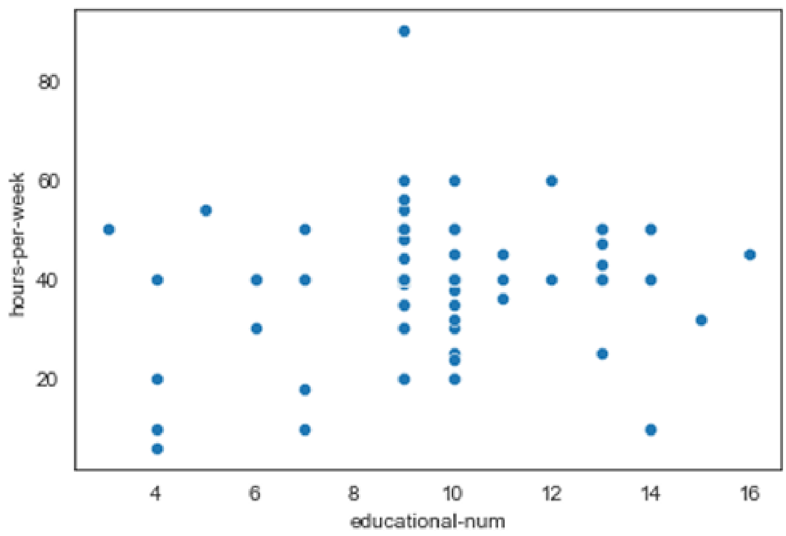
c) CORRELATION AND SCATTER PLOTS

ALGORITHM:

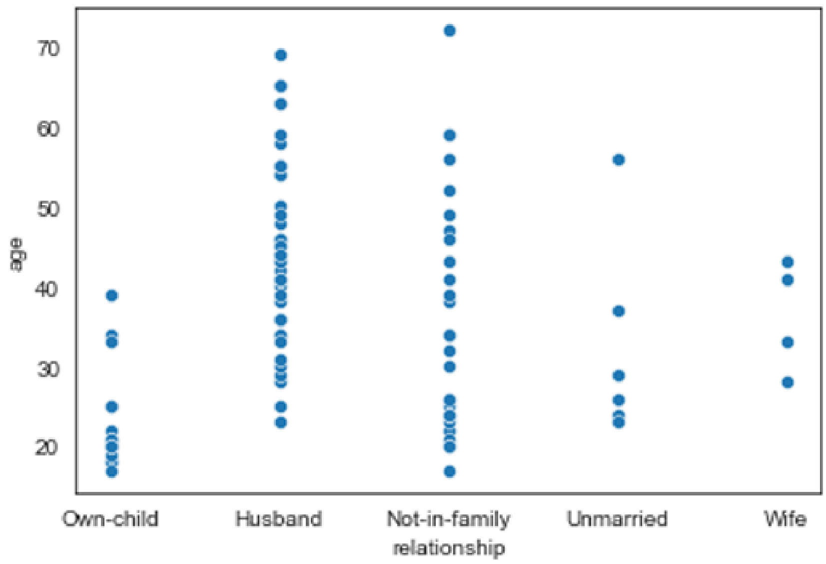
Step 1: Load the relevant libraries, such as numpy, pandas, matplotlib, and seaborn.
Step 2: Read the adult UCI data and save it as dataFrame(df).
Step 3: Assign x and y to draw a scatter plot between two variables.
Step 4: Assign a variable to "hue" that maps its levels to the color of the points.
Step 5: Create a heatmap to observe the correlation between two or more (numeric) variables.
Step 6: Display the output.

PROGRAM:

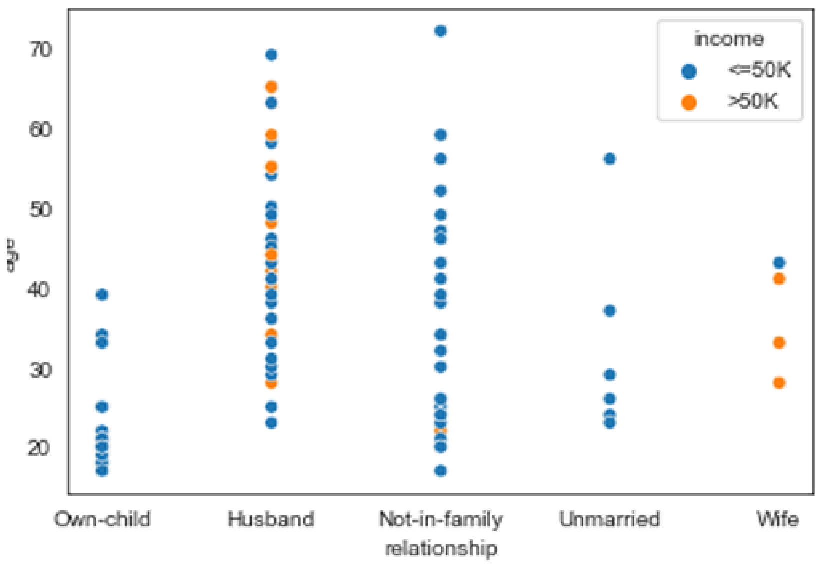
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

df = pd.read_csv("adult.csv")

# set seaborn style
sns.set_style("white")
sns.scatterplot(data=df[0:100], x="educational-num", y="hours-per-week")
```



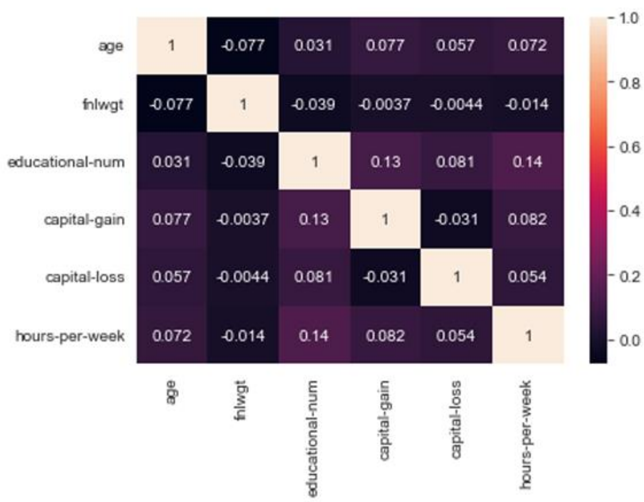

```
sns.scatterplot(data=df[0:100], x="relationship", y="age")
```



sns.scatterplot(data=df[0:100], x="relationship", y="age", hue="income")



cormat = df.corr()
sns.heatmap(cormat, annot=True);



d) HISTOGRAMS

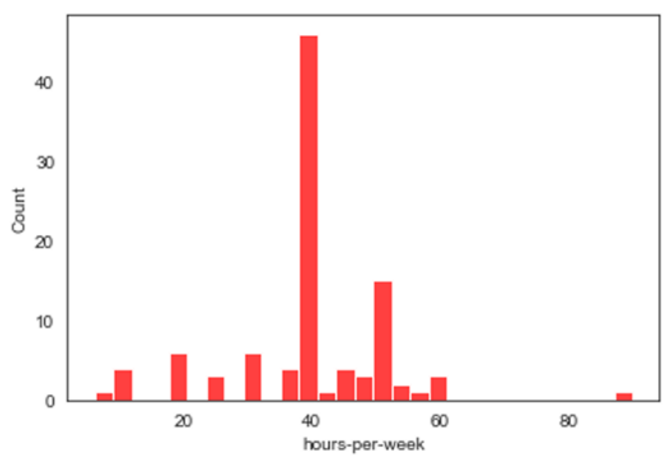
ALGORITHM:

Step 1: Load the relevant libraries, such as numpy, pandas, matplotlib, and seaborn.
Step 2: Read the adult UCI data and save it as dataFrame(df).
Step 3: Create a simple histogram by assigning a variable to x to plot a univariate distribution along the x-axis.
Step 4: To smooth the histogram, add a kde curve by setting the kde argument to True.
Step 5: Apply histplot and distplot functions to observe the distributions.
Step 6: Plot multiple distributions and "stack" them.
Step 7: Plot histograms for each of the continuous variables using hist function.
Step 8: Display the output.

PROGRAM:

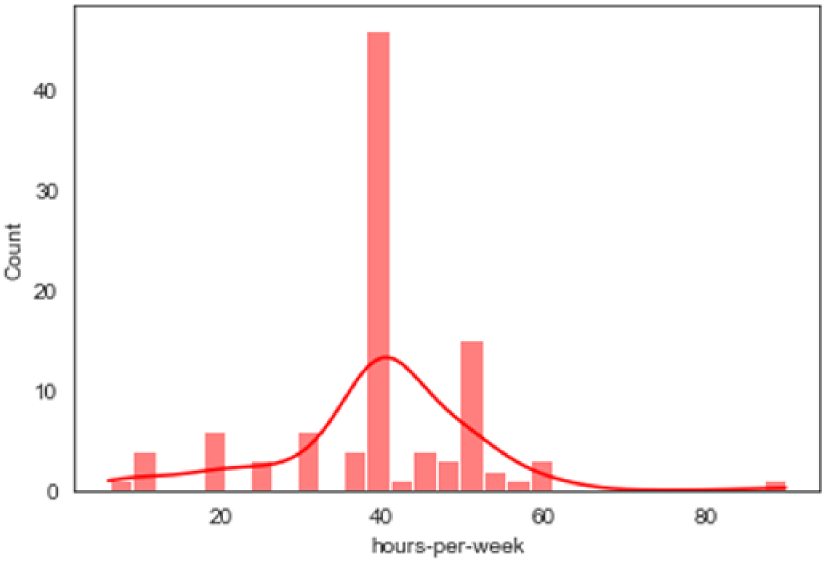
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

df = pd.read_csv("adult.csv")

# set seaborn style
sns.set_style("white")

# A simple histogram
sns.histplot(data=df[:100], x="hours-per-week", color="red")
```

```
<AxesSubplot:xlabel='hours-per-week', ylabel='Count'>
```



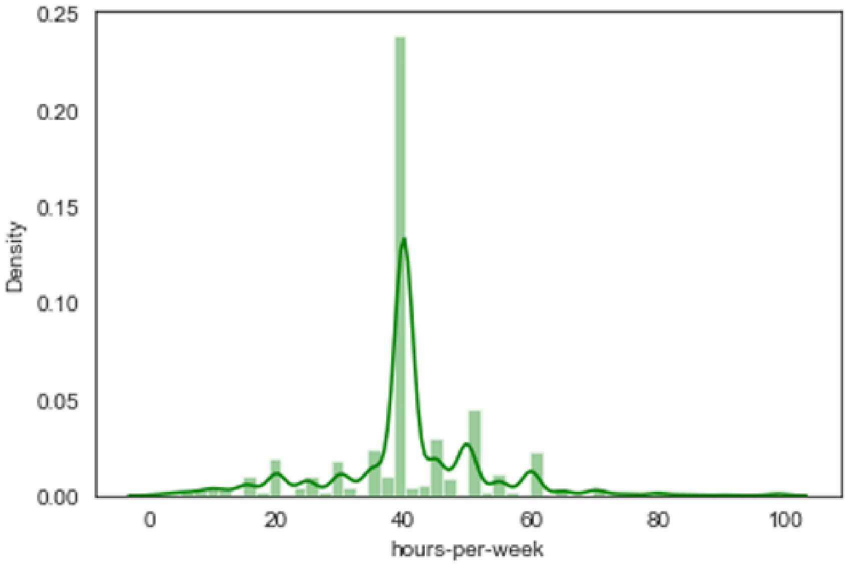
sns.histplot(data=df[:100], x="hours-per-week", kde=True, color="red")

<AxesSubplot:xlabel='hours-per-week', ylabel='Count'>



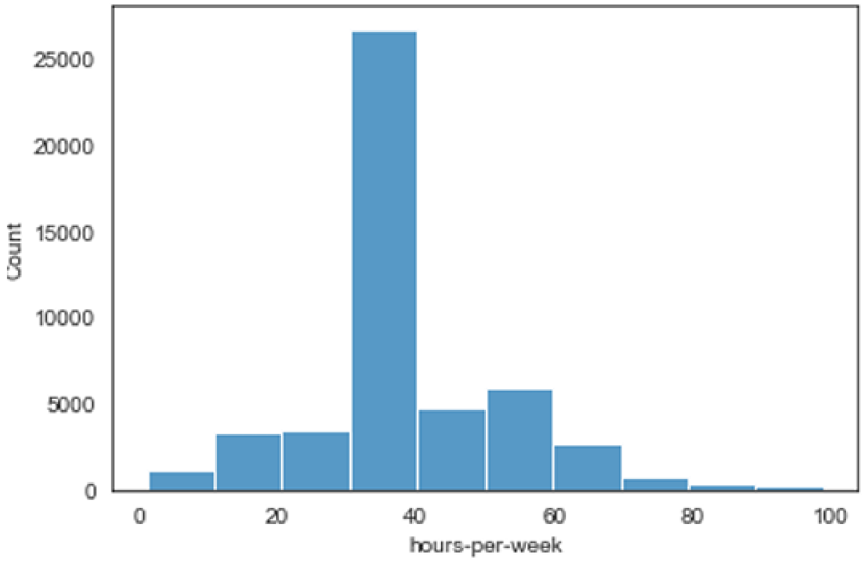
sns.distplot(df["hours-per-week"], color="green")

<AxesSubplot:xlabel='hours-per-week', ylabel='Density'>

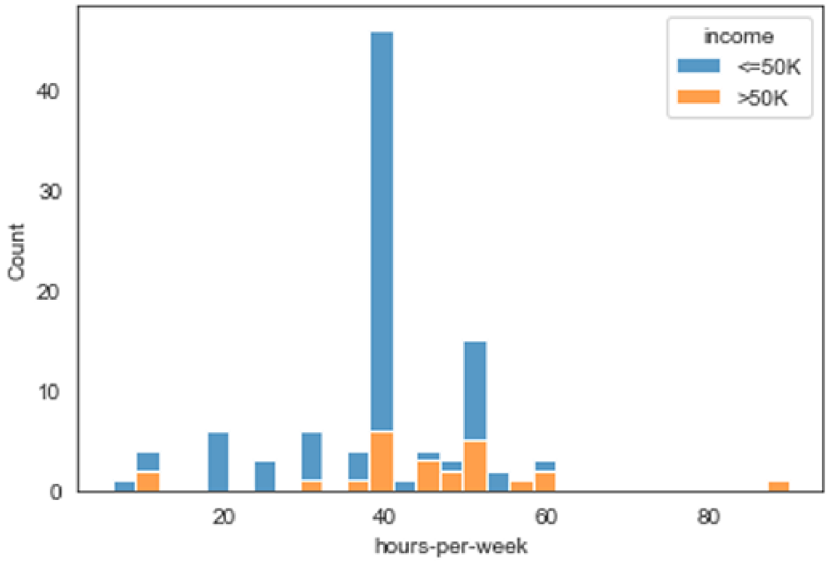


sns.histplot(data=df,
x="hours-per-week", bins=10)

<AxesSubplot:xlabel='hours-per-week', ylabel='Count'>

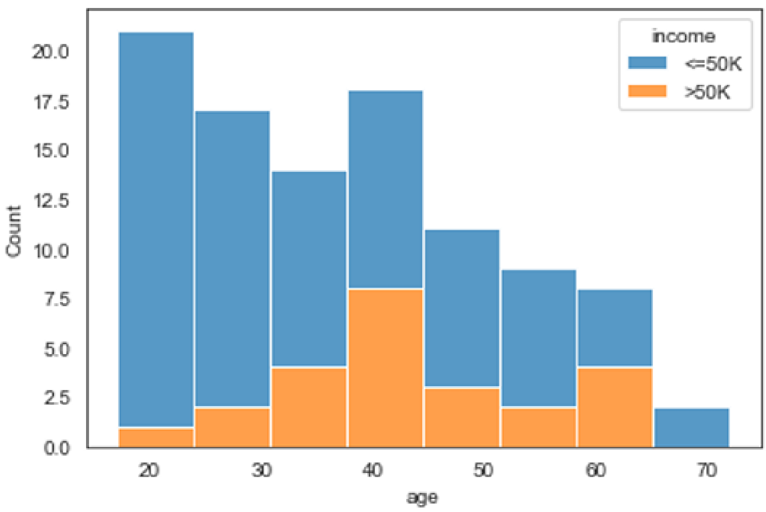


sns.histplot(data=df[:100], x="hours-per-week", hue="income", multiple="stack")



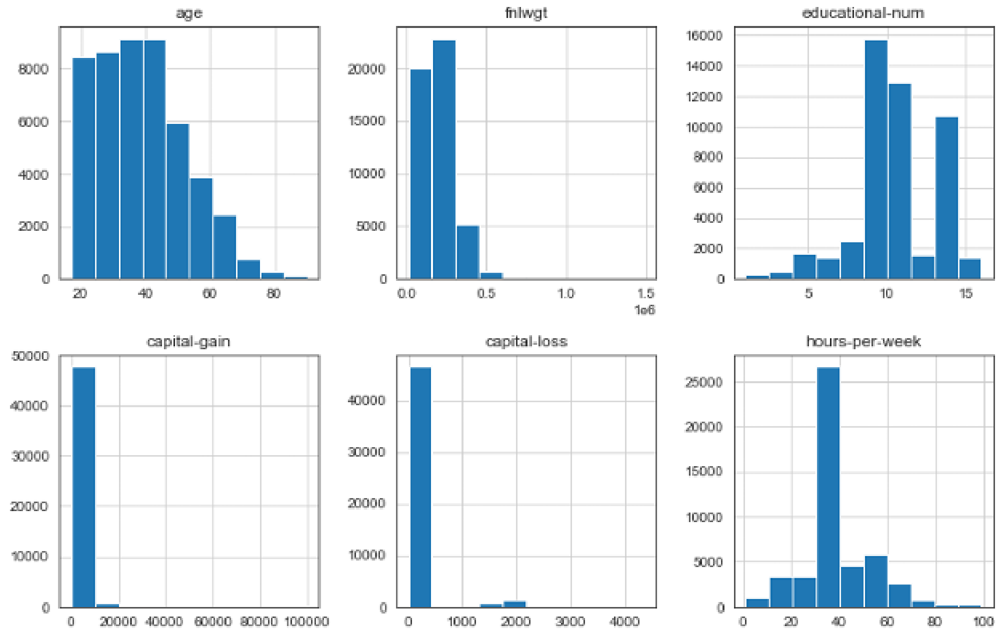
sns.histplot(data=df[:100], x="age", hue="income", multiple="stack")

```
<AxesSubplot:xlabel='age', ylabel='Count'>
```



df.hist(figsize=(12,12), layout=(3,3), sharex=False)

```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'fnlwgt'}>,
        <AxesSubplot:title={'center':'educational-num'}>],
       [<AxesSubplot:title={'center':'capital-gain'}>,
        <AxesSubplot:title={'center':'capital-loss'}>,
        <AxesSubplot:title={'center':'hours-per-week'}>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



e) THREE-DIMENSIONAL PLOTTING

ALGORITHM:

Step 1: Install plotly using the command: $ conda install plotly

Step 2: Load the relevant libraries, such as numpy, pandas, matplotlib, and plotly.

Step 3: Read the adult UCI data and save it as dataFrame(df).

Step 4: Plot individual data in three-dimensional space using px.scatter_3d function.

Step 5: Display the output.

PROGRAM:

```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px

df = pd.read_csv("adult.csv")
fig = px.scatter_3d(df[:200], x='age', y='capital-gain', z='hours-per-week',
color='income')
fig.show()
```

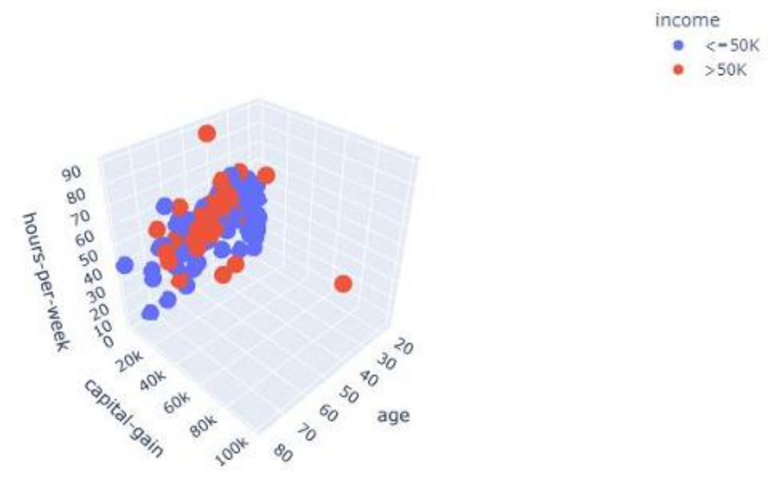
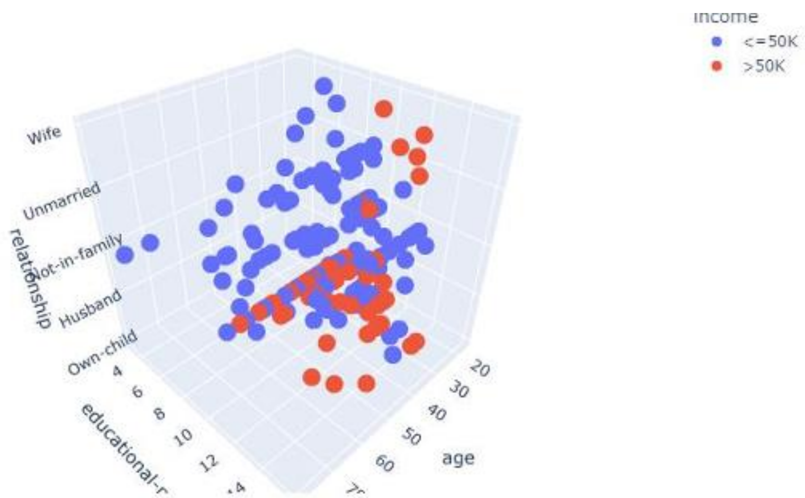


fig1 = px.scatter_3d(df[:200], x='age', y='educational-num', z='relationship', color='income')
fig1.show()



RESULT:

Thus, various plotting functions using matplotlib, seaborn, and plotly packages on the Adult UCI dataset were applied and explored.

EX. NO.:7 **DATE:**

VISUALIZING GEOGRAPHIC DATA WITH BASEMAP

AIM:

To visualize California cities data using basemap.

ALGORITHM:

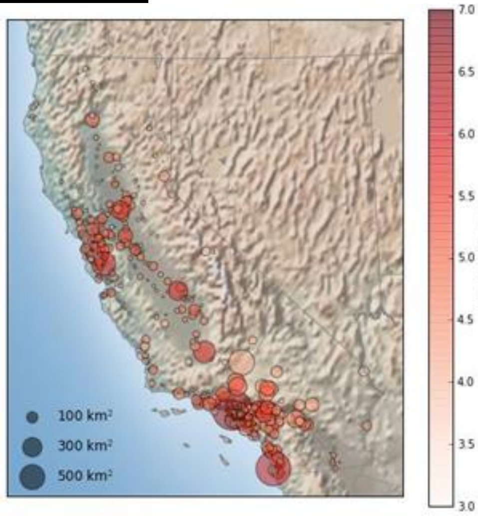
Step 1: Install basemap using the command: $ conda install basemap
Step 2: Load the relevant libraries, such as numpy, pandas, matplotlib, and basemap.
Step 3: Read the ‘california_cities.csv' data and extract it.
Step 4: Draw the map background.
Step 5: Scatter city data, with color reflecting population and size reflecting area.
Step 6: Make a legend with dummy points.
Step 7: Create a colorbar and legend.

PROGRAM:

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

cities = pd.read_csv('california_cities.csv')
# Extract the data
lat = cities['latd'].values
lon = cities['longd'].values
population = cities['population_total'].values
area = cities['area_total_km2'].values

# Draw the map background
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution='h', lat_0=37.5, lon_0=-119,
    width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')
m.drawstates(color='gray')
# scatter city data, with color reflecting population and size reflecting area
m.scatter(lon, lat, latlon=True, c=np.log10(population), s=area, cmap='Reds',
    alpha=0.5)
# create colorbar and legend
plt.colorbar(label=r'$\log_{10}({\rm population})$')
```

```
plt.clim(3, 7)
# make legend with dummy points
for a in [100, 300, 500]:
plt.scatter([], [], c='k', alpha=0.5, s=a, label=str(a) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False, labelspacing=1, loc='lower left');
```

OUTPUT:



RESULT:

Thus, the map showing information about the location, size, and population of California cities was displayed using the basemap package and the output is verified.