EXPERIMENT 1 – INTRODUCTION TO PYTHON PROGRAMMING

AIM: A brief introduction to Python with the implementation of basic commands.

SOFTWARE REQUIRED:

Spyder IDE 5.1.5

Anaconda3 2021.11 (Python 3.9.7 64-bit)

Anaconda Inc., 2021.11

NUMPY: (https://www.w3schools.com/python/numpy/default.asp)

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data types can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

```
import numpy as np # Support for large, multi-dimensional arrays and matrices
```

1. **NumPy Creating Arrays:** – NumPy is used to work with arrays. The array object in NumPy is called ndarray. We can create a NumPy ndarray object by using the array() function. For example,

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

(i) 0-D Arrays – 0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array. For example, create a 0-D array with the value 42.

```
arr = np.array(42)
print(arr)
```

(ii) 1-D Arrays – An array that has 0-D arrays as its elements are called a uni-dimensional or 1-D array. These are the most common and basic arrays. For example, create a 1-D array containing the values 1,2,3,4,5.

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

(iii) 2-D Arrays – An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix or 2nd order tensors. For example, create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6.

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

2. **NumPy Array Indexing:** –

(i) Access Array Elements – Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, the second has index 1, etc. For example,

```
arr = np.array([1, 2, 3, 4])
print(arr[0]) # Get the first element from the array
print(arr[1]) # Get the second element from the array
print(arr[2] + arr[3]) # Get third and fourth elements from the
array and add them
```

(ii) Access 2-D Arrays – To access elements from 2-D arrays we can use comma-separated integers representing the dimension and the index of the element. Think of 2-D arrays like a table with rows and columns, where the row represents the dimension and the index represents the column. For example,

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print("2nd element on 1st row: ", arr[0, 1]) # Access the element
on the 1st row, 2nd column
print("5th element on 2nd row: ", arr[1, 4]) # Access the element
on the 2nd row, 5th column
```

(iii) Access 3-D Arrays – To access elements from 3-D arrays we can use comma-separated integers representing the dimensions and the index of the element. For example, access the third element of the second array of the first array.

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11,
12]]])
print(arr[0, 1, 2])
```

3. **NumPy Array Slicing:** – Slicing in python means taking elements from one given index to another given index. We pass a slice instead of an index like this: [start:end].

We can also define the step, like this: [start:end:step]. If we don't pass the start it's considered 0. If we don't pass the end it's considered the length of the array in that dimension. If we don't pass a step it's considered 1. For example,

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5]) # Slice elements from index 1 to index 5 from the array
print(arr[4:]) # Slice elements from index 4 to the end of the array
print(arr[:4]) # Slice elements from the beginning to index 4 (not
included)
```

(i) Negative Slicing – Use the minus operator to refer to an index from the end. For example, slice from index 3 from the end to index 1 from the end.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

(ii) Step – Use the step value to determine the step of the slicing. For example,

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2]) # Return every other element from index 1 to
index 5
print(arr[::2]) # Return every other element from the entire
array
```

MATPLOTLIB: (https://www.w3schools.com/python/matplotlib_intro.asp)

Matplotlib is a low-level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, and a few segments are written in C, Objective-C and Javascript for Platform compatibility. Most of the Matplotlib utilities lie under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt # Provides an implicit way of plotting
```

1. **Matplotlib Pyplot:** – Now the Pyplot package can be referred to as plt. For example, draw a line in a diagram from position (0,0) to position (6,250).

```
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

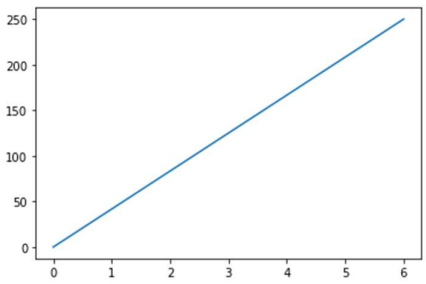

Figure 1 – Matplotplib Pyplot

2. **Matplotlib Plotting:** –
   (i) Plotting x and y points – The plot() function is used to draw points (markers) in a diagram. By default, the plot()

function draws a line from point to point. The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the x-axis. Parameter 2 is an array containing the points on the y-axis. If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function. For example, draw a line in a diagram from position (1, 3) to position (8, 10).

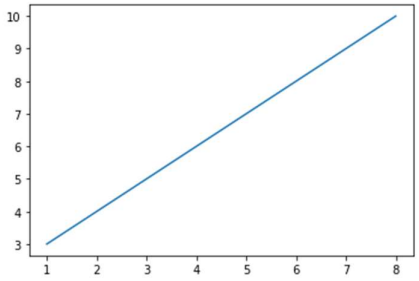

Figure 2 – Matplotlib Plotting

```
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints)
plt.show()
```

(ii) Plotting Without Line – To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'. For example, draw two points in the diagram, one at position (1, 3) and one in position (8, 10).

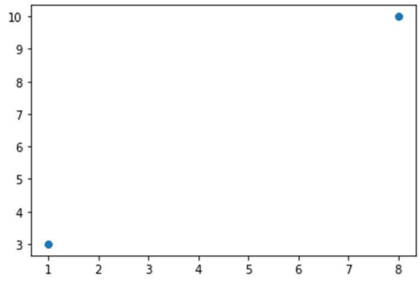

Figure 3 – Plotting Without Line

```
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints,
'o')
plt.show()
```

(iii) Multiple Points – You can plot as many points as you like, just make sure you have the same number of points on both axis. For example, draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10).

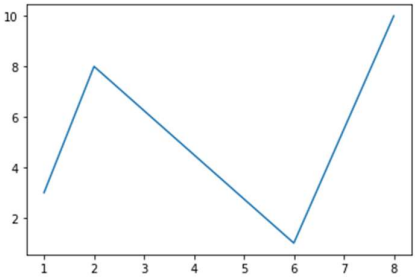

Figure 4 – Multiple Points

```
xpoints = np.array([1, 2, 6,
8])
ypoints = np.array([3, 8, 1,
10])
plt.plot(xpoints, ypoints)
plt.show()
```

3. **Matplotlib Markers:** –

(i) Markers – You can use the keyword argument marker to emphasize each point with a specified marker. For example, mark each point with a circle.

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```

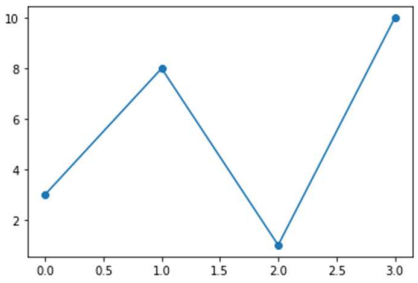

Figure 5 - Markers

(ii) Format Strings (fmt) – You can use also use the shortcut string notation parameter to specify the marker. This parameter is also called fmt, and is written with this syntax: marker|line|color. For example, mark each point with a circle.

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, 'o:r')
plt.show()
```

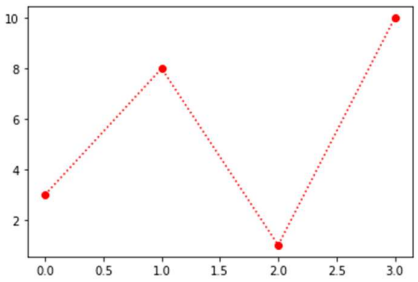

Figure 6 – Format Strings (fmt)

(iii) Marker Size – You can use the keyword argument marker size or the shorter version, ms to set the size of the markers. For example, set the size of the markers to 20.

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20)
plt.show()
```

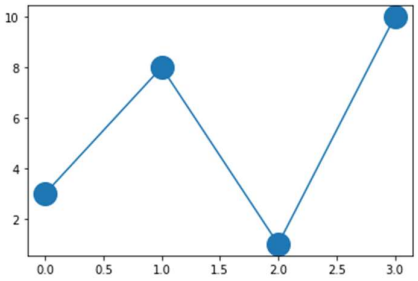

Figure 7 – Marker Size

SEABORN: (https://www.tutorialspoint.com/seaborn/index.htm)

It is an open-source, BSD-licensed Python library providing a high-level API for visualizing the data using Python programming language.

```
import seaborn as sb # Provides high-level API to visualize data
```

1. **Importing Data as Pandas DataFrame:** – In this section, we will import a dataset. This dataset loads as Pandas DataFrame by default. If there is any function in the Pandas

DataFrame, it works on this DataFrame. The following line of code will help you import the dataset,

```
df = sb.load_dataset('flights')
print (df.head())
```

2. **Histogram:** – Histograms represent the data distribution by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin. For example,

```
df = sb.load_dataset('iris')
sb.histplot(df['petal_length'], kde = False) # Here, kde flag is set to False. As a result, the representation of the kernel estimation plot will be removed and only the histogram is plotted.
plt.show()
```

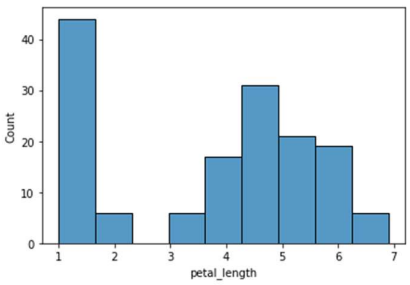

Figure 8 - Histogram

3. **Categorical Scatter Plots, stripplot():** – It is used when one of the variables under study is categorical. It represents the data in sorted order along any one of the axis. For example,

```
df = sb.load_dataset('iris')
sb.stripplot(x = "species", y = "petal_length", data = df)
plt.show()
```

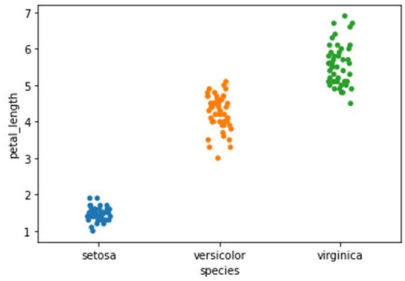

Figure 9 – Categorical Scatter Plots, stripplot()

4. **Statistical Estimation:** – In most situations, we deal with estimations of the whole distribution of the data. But when it comes to central tendency estimation, we need a specific way to summarize the distribution. Mean and median are the very often used techniques to estimate the central tendency of the distribution.

(i) Bar Plot – The barplot() shows the relation between a categorical variable and a continuous variable. The data is represented in rectangular bars where the length of the bar represents the proportion of the data in that category. The bar plot represents the estimate of central tendency. Let us use the 'titanic' dataset to learn bar plots.

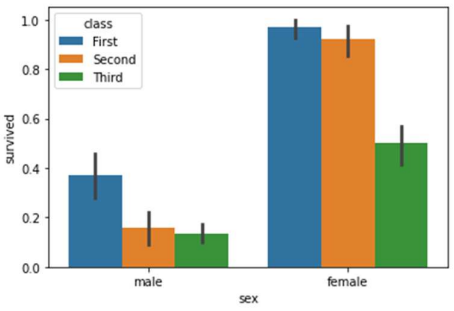

Figure 10 – Bar Plot

```
df = sb.load_dataset('titanic')
sb.barplot(x = "sex", y = "survived", hue = "class", data = df)
plt.show()
```

(ii) Point Plots – Point plots serve the same as bar plots but in a different style. Rather than the full bar, the value of the estimate is represented by the point at a certain height on the other axis. For example,

```
df = sb.load_dataset('titanic')
sb.pointplot(x = "sex", y = "survived", hue = "class", data = df)
plt.show()
```

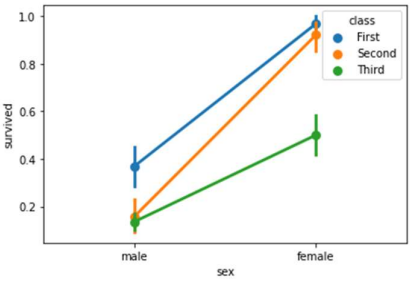

Figure 11 – Point Plots

5. **Plotting Wide Form Data:** – It is always preferable to use 'long-from' or 'tidy' datasets. But at times when we are left with no option rather but to use a 'wide-form' dataset, the same functions can also be applied to "wide-form" data in a variety of formats, including Pandas Data Frames or two-dimensional NumPy arrays. These objects should be passed directly to the data parameter the x and y variables must be specified as strings. For example,

```
df = sb.load_dataset('iris')
sb.boxplot(data = df, orient = "h")
plt.show()
```

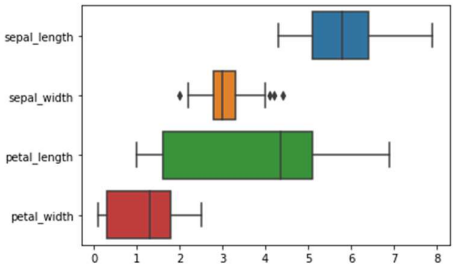

Figure 12 – Plotting Wide Form Data

PANDAS: (https://www.w3schools.com/python/pandas/default.asp)

It is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

```
import pandas as pd # Library for working with data sets
```

1. **Series:** – It is like a column in a table. It is a one-dimensional array holding data of any type. For example, create a simple Pandas Series from a list.

```
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

```
print(myvar[0]) # Return the first value of the series
```

(i) Create Labels – With the index argument, you can name your labels. For example,

```
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
print(myvar["y"]) # Return the value of "y"
```

(ii) Key/Value Objects as Series – You can also use a key/value object, like a dictionary, when creating a Series. For example, create a simple Pandas Series from a dictionary.

```
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

(iii) DataFrames – Data sets in Pandas are usually multi-dimensional tables, called DataFrames. Series is like a column, a DataFrame is a whole table. For example, create a DataFrame from two Series.

```
data = {
"calories": [420, 380, 390],
"duration": [50, 40, 45]
}
myvar = pd.DataFrame(data)
print(myvar)
```

2. **Read CSV Files:** – A simple way to store big data sets is to use CSV files (comma-separated files). CSV files contain plain text and are a well know format that can be read by everyone including Pandas. For example, load the CSV into a DataFrame.

```
df = pd.read_csv('data.csv')
print(df.to_string())
```

3. **Analyzing DataFrames:** –

(i) Viewing the Data – One of the most used methods for getting a quick overview of the DataFrame, is the head() method. The head() method returns the headers and a specified number of rows, starting from the top. For example, get a quick overview by printing the first 10 rows of the DataFrame.

```
df = pd.read_csv('data.csv')
print(df.head(10))
```

(ii) Info About the Data – The DataFrames object has a method called info(), that gives you more information about the data set. For example,

```
df = pd.read_csv('data.csv')
print(df.info())
```

SCIPY: (https://www.w3schools.com/python/scipy/index.php)

SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely. SciPy was created by NumPy's creator Travis Oliphant.

1. **Constants in SciPy:** – As SciPy is more focused on scientific implementations, it provides many built-in scientific constants. These constants can be helpful when you are working with Data Science. For example, print the constant value of PI.

```
from scipy import constants
print("\nConstants in SciPy:")
print(constants.pi)
```

2. **Roots of an Equation**: – NumPy is capable of finding roots for polynomials and linear equations, but it can not find roots for non-linear equations, like this one:

$$x + \cos(x)$$

For that, you can use SciPy's optimize.root function. This function takes two required arguments:

(i) fun - a function representing an equation,
(ii) x0 - an initial guess for the root.

The function returns an object with information regarding the solution. The actual solution is given under attribute x of the returned object. For example, find root of the equation x + cos(x).

```
from scipy.optimize import root
from math import cos
def eqn(x):
      return x + cos(x)
myroot = root(eqn, 0)
print(myroot.x) # Find root of the equation x + cos(x)
print("\n")
print(myroot) # Print all information about the solution (not just x
which is the root)
```

SCIKIT-LEARN: (https://www.tutorialspoint.com/scikit_learn/index.htm)

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistency interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

1. **Dataset Loading:** – A collection of data is called a dataset. It is having the following two components,

(i) Features – The variables of data are called their features. They are also known as predictors, inputs or attributes.
- Feature Matrix – It is the collection of features, in case there is more than one.
- Feature Names – It is the list of all the names of the features.

(ii) Response – It is the output variable that depends upon the feature variables. They are also known as target, label or output.
- Response Vector – It is used to represent the response column. Generally, we have just one response column.
- Target Names – These represent the possible values taken by a response vector.

Scikit-learn has a few example datasets like iris and digits for classification and the Boston house prices for regression. Following is an example to load an iris dataset,

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nFirst 10 rows of X:\n", X[:10])
```

2. **Splitting the Dataset**: – To check the accuracy of our model, we can split the dataset into two pieces-a training set and a testing set. Use the training set to train the model and the testing set to test the model. After that, we can evaluate how well our model did.

   The following example will split the data into a 70:30 ratio, i.e. 70% of data will be used as training data and 30% will be used as testing data. The dataset is the iris dataset as in the above example.

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

RESULT:

A brief introduction to Python is documented along with the syntaxes and examples from basic concepts and commands and all simulation results were verified successfully.

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

Restarting kernel...
```

```
In [1]:         'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/
19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 1 - Introduction to
Python Programming/Expt_1_Code_Matplotlib.py'      ='E:/Plan B/Amrita Vishwa Vidyapeetham/
Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial Intelligence/Lab/
Experiment 1 - Introduction to Python Programming'

Pyplot:

Plotting x and y points:

Plotting Without Line:

Multiple Points:

Markers:

Format Strings (fmt):

Marker Size:

In [2]:         'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/
19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 1 - Introduction to
Python Programming/Expt_1_Code_NumPy.py'      ='E:/Plan B/Amrita Vishwa Vidyapeetham/
Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial Intelligence/Lab/
Experiment 1 - Introduction to Python Programming'


[1 2 3 4 5]
<class 'numpy.ndarray'>


42


[1 2 3 4 5]


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


1
2
7
```

```
2nd element on 1st row:  2
5th element on 2nd row:  10


6


[2 3 4 5]
[5 6 7]
[1 2 3 4]


[5 6]


[2 4]
[1 3 5 7]
```

In [3]:
```
'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 1 - Introduction to Python Programming/Expt_1_Code_Pandas.py'      ='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 1 - Introduction to Python Programming'

Pandas Series:
0    1
1    7
2    2
dtype: int64
1

Create Labels:
x    1
y    7
z    2
dtype: int64
7

Key/Value Objects as Series:
day1    420
day2    380
day3    390
dtype: int64

DataFrames:
   calories  duration
0       420        50
1       380        40
2       390        45

Read CSV Files:
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
```

3	45	109	175	282.4
4	45	117	148	406.0
5	60	102	127	300.0
6	60	110	136	374.0
7	45	104	134	253.3
8	30	109	133	195.1
9	60	98	124	269.0
10	60	103	147	329.3
11	60	100	120	250.7
12	60	106	128	345.3
13	60	104	132	379.3
14	60	98	123	275.0
15	60	98	120	215.2
16	60	100	120	300.0
17	45	90	112	NaN
18	60	103	123	323.0
19	45	97	125	243.0
20	60	108	131	364.2
21	45	100	119	282.0
22	60	130	101	300.0
23	45	105	132	246.0
24	60	102	126	334.5
25	60	100	120	250.0
26	60	92	118	241.0
27	60	103	132	NaN
28	60	100	132	280.0
29	60	102	129	380.3
30	60	92	115	243.0
31	45	90	112	180.1
32	60	101	124	299.0
33	60	93	113	223.0
34	60	107	136	361.0
35	60	114	140	415.0
36	60	102	127	300.0
37	60	100	120	300.0
38	60	100	120	300.0
39	45	104	129	266.0
40	45	90	112	180.1
41	60	98	126	286.0
42	60	100	122	329.4
43	60	111	138	400.0
44	60	111	131	397.0
45	60	99	119	273.0
46	60	109	153	387.6
47	45	111	136	300.0
48	45	108	129	298.0
49	60	111	139	397.6
50	60	107	136	380.2
51	80	123	146	643.1
52	60	106	130	263.0
53	60	118	151	486.0
54	30	136	175	238.0
55	60	121	146	450.7
56	60	118	121	413.0
57	45	115	144	305.0

58	20	153	172	226.4
59	45	123	152	321.0
60	210	108	160	1376.0
61	160	110	137	1034.4
62	160	109	135	853.0
63	45	118	141	341.0
64	20	110	130	131.4
65	180	90	130	800.4
66	150	105	135	873.4
67	150	107	130	816.0
68	20	106	136	110.4
69	300	108	143	1500.2
70	150	97	129	1115.0
71	60	109	153	387.6
72	90	100	127	700.0
73	150	97	127	953.2
74	45	114	146	304.0
75	90	98	125	563.2
76	45	105	134	251.0
77	45	110	141	300.0
78	120	100	130	500.4
79	270	100	131	1729.0
80	30	159	182	319.2
81	45	149	169	344.0
82	30	103	139	151.1
83	120	100	130	500.0
84	45	100	120	225.3
85	30	151	170	300.0
86	45	102	136	234.0
87	120	100	157	1000.1
88	45	129	103	242.0
89	20	83	107	50.3
90	180	101	127	600.1
91	45	107	137	NaN
92	30	90	107	105.3
93	15	80	100	50.5
94	20	150	171	127.4
95	20	151	168	229.4
96	30	95	128	128.2
97	25	152	168	244.2
98	30	109	131	188.2
99	90	93	124	604.1
100	20	95	112	77.7
101	90	90	110	500.0
102	90	90	100	500.0
103	90	90	100	500.4
104	30	92	108	92.7
105	30	93	128	124.0
106	180	90	120	800.3
107	30	90	120	86.2
108	90	90	120	500.3
109	210	137	184	1860.4
110	60	102	124	325.2
111	45	107	124	275.0
112	15	124	139	124.2

113	45	100	120	225.3
114	60	108	131	367.6
115	60	108	151	351.7
116	60	116	141	443.0
117	60	97	122	277.4
118	60	105	125	NaN
119	60	103	124	332.7
120	30	112	137	193.9
121	45	100	120	100.7
122	60	119	169	336.7
123	60	107	127	344.9
124	60	111	151	368.5
125	60	98	122	271.0
126	60	97	124	275.3
127	60	109	127	382.0
128	90	99	125	466.4
129	60	114	151	384.0
130	60	104	134	342.5
131	60	107	138	357.5
132	60	103	133	335.0
133	60	106	132	327.5
134	60	103	136	339.0
135	20	136	156	189.0
136	45	117	143	317.7
137	45	115	137	318.0
138	45	113	138	308.0
139	20	141	162	222.4
140	60	108	135	390.0
141	60	97	127	NaN
142	45	100	120	250.4
143	45	122	149	335.4
144	60	136	170	470.2
145	45	106	126	270.8
146	60	107	136	400.0
147	60	112	146	361.9
148	30	103	127	185.0
149	60	110	150	409.4
150	60	106	134	343.0
151	60	109	129	353.2
152	60	109	138	374.0
153	30	150	167	275.8
154	60	105	128	328.0
155	60	111	151	368.5
156	60	97	131	270.4
157	60	100	120	270.4
158	60	114	150	382.8
159	30	80	120	240.9
160	30	85	120	250.4
161	45	90	130	260.4
162	45	95	130	270.0
163	45	100	140	280.9
164	60	105	140	290.8
165	60	110	145	300.0
166	60	115	145	310.2
167	75	120	150	320.4

```
168        75    125       150     330.4

Viewing the Data:
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.0
6        60    110       136     374.0
7        45    104       134     253.3
8        30    109       133     195.1
9        60     98       124     269.0

Info About the Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None

In [4]:          'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/
19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 1 - Introduction to
Python Programming/Expt_1_Code_SciPy_Sklearn.py'      ='E:/Plan B/Amrita Vishwa
Vidyapeetham/Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial
Intelligence/Lab/Experiment 1 - Introduction to Python Programming'

Constants in SciPy:
3.141592653589793

Roots of an Equation:
[-0.73908513]


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

Dataset Loading:
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```

```
Target names: ['setosa' 'versicolor' 'virginica']

First 10 rows of X:
 [[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]]

Splitting the Dataset:
(105, 4)
(45, 4)
(105,)
(45,)

In [5]:          'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/
19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 1 - Introduction to
Python Programming/Expt_1_Code_Seaborn.py'      ='E:/Plan B/Amrita Vishwa Vidyapeetham/
Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial Intelligence/Lab/
Experiment 1 - Introduction to Python Programming'

Importing Datasets and Libraries (Importing Data as Pandas DataFrame):
   year month  passengers
0  1949   Jan         112
1  1949   Feb         118
2  1949   Mar         132
3  1949   Apr         129
4  1949   May         121

Histogram:

Plotting Categorical Data (stripplot()):

Statistical Estimation (Bar Plot):

Statistical Estimation (Point Plots):

Plotting Wide Form Data:

In [6]:
```