DELTA Pg No. Date / /

EXPERIMENT - 1

: To study the following python libraries
i) numpy ii) matplotlib iii) scikit-learn

Numpy :-

→ Fundamental package for numerical computations
→ Supports Multi-Dimensional arrays & matrices.
→ Offers mathematical functions for array operations.

Functions :- np.array(), np.zeroes(), np.ones(), np.sum() etc.

Matplotlib :-

→ Popular plotting library for creating visualizations in Python.
→ Supports static, interactive, and animated plots.
→ Offers wide range of functions for data visual tasks.

Functions :- plt.plot(), plt.scatter(), plt.bar(), plt.legend()

Scikit-learn :-

→ Machine Learning in Python
→ Contain various algorithms for classification, regression, clustering, etc.
→ Provides tools for data mining, analysis & model evaluation.

Functions :- LinearRegression(), model.predict(), train_test_split()

DELTA Pg No. Date / /

EXPERIMENT - 2

Aim: To create & work with numpy arrays

a) Create 1D, 2d, & 3D array

```
Import numpy as np
arr1 = np.array([4,5,8,7,1,3,6,2])
arr2 = np.array([[40,30,10], [-4,-6,-5]])
arr3 = np.array([[[1,4,2], [78,77,76]], [[4,6,5], [44,45,46]],
                 [[8,7,9], [11,22,33]]])
print("Arrays created successfully!")
```

b) Access Elements of 1d, 2d, & 3d array

```
print("\n\n Accessing elements from arrays :- \n")
print("1-D Array: \n", arr1)
print("\n 2-D Array: \n", arr2)
print("\n 3-D Array: \n", arr3)
```

c) Print the shape of the array

```
print("\n\n Shape of arrays :- ")
print("\n Shape:", np.shape(arr1))
print("\n Shape:", np.shape(arr2))
print("\n Shape:", np.shape(arr3))
```

d) Perform array slicing :

```
# -> select range 0:1 rows
print(arr2[0:1,:], "\n")
print("\n\n Array slicing :- \n")
```

DELTA Pg No. Date / /

```
# --> Specific row [0, 1]
print (arr2 [[0,1], : ], "\n")
# --> Normal slicing
print (arr3 [2] [0] [2], "\n")
```

Perform boolean Indexing

```
print ("\n\nPerforming boolean Indexing :-")
print ("Displaying Boolean list :- \n")
print (arr2 > 2, "\n")
print ("Displaying the values of the boolean list :-")
print (arr2 [arr2 > 2])
```

Reshape a 1-D Array into 2-D & 3-D array :

```
#reshaping into 2-D Array
print ("\n Reshaping of 1-D Array :- \n")
print ("Reshaping 1-D Array into 2-D : \n")
print (arr1.reshape (2,4))
#reshaping into 3-D Arrays
print ("\n Displaying 1-D Array into 3-D : \n")
arr4 = np.array ([1,2,3,4,5,6,7,8,9])
print (arr4.reshape (3,3))
```

Perform array Arithmetic Operations by using numpy functions :-

```
print ("\n\n Perform array Arithmetic operations :-")
print ("\n Addition :-"
```

```
arr01 = np.array([8,10,24,21,2,9,36,62])
print(arr01 + arr1)
print(arr01 - arr1)
print(arr1 * arr1)
print(arr01 / arr1)
print(arr01, '+', arr1, "=", np.add(arr01, arr1))

print("\nSubtraction:")
arr03 = np.array([[[2,4,1],[76,77,79]],
                  [[5,6,4],[14,54,65]],
                  [[9,7,8],[21,42,53]]])
print(arr01, '-', arr1, "=", np.add(arr03, arr3))

print("\nMultiplication:-")
print(arr1, '*', arr1, "=", np.multiply(arr1, arr1))

print("\nDivision:-")
print(arr01, '/', arr1, "=", np.divide(arr01, arr1))
```

b) Perform Matrix Multiplication using numpy dot function

```
A = [[2,3,3],[4,5,6],[7,8,9]]
B = [[5,8,2],[6,7,0],[4,5,1]]
print("\nMatrix A:\n", A)
print("\nMatrix B:\n", B, "\n")
print("Result of Matrix Multiplication is:")
print("\n", np.dot(A,B))
```

DELTA Pg No.
Date / /

i) To sort numpy arrays

```
print("\n\n Sorting numpy arrays :- \n Sorting 1-D array :-")
print(np.sort(arr2))
print("\n Sorting 2-D array :-")
print(np.sort(arr2))
print("\n Sorting 3-D array :-")
print(np.sort(arr3))
```

ii) To perform Assignment, Shallow Copy & Deep Copy:

```
print("\n\n Assignment :-")
arr5 = arr1
print(arr5)
print("\n Shallow Copy :-")
arr6 = arr5.view()
print(arr6)
print("\n Deep Copy :-")
arr7 = arr5.copy()
print(arr7)
```

k) To perform matrix multiplication using matrix method :-

```
print("\n\n Matrix Multiplication using matrix method:")
mat1 = np.matrix([[1,4,5],[6,8,9]])
print("mat 1 = ", mat1, "\n")
mat2 = np.matrix([[9,8,6],[5,4,1],[1,2,3]])
print("mat2 =", mat2, "\n")
print("Result =")
print(mat1 @ mat2)   # or np.dot(mat1, mat2)
```

1) Output

2023-01-31	10
2023-02-28	20
2023-3-31	30
2023-04-30	40
2023-05-31	50
⋮ ⋮ ⋮	⋮
2023-12-31	120

Using loc
30
60

Using iloc
30
60

2)

Passenger Id	Survived	Pclass	Name	Sex	Age
1	0		Brand	M	22
2	1		Cumings	F	38
3	1		Hoklner	F	26
4	1		Euchie	F	35
5	0		Allen	M	35

Sliced Data Using loc:

	Passenger ID	Age
2	3	26
3	4	35
4	5	35

DELTA Pg No.
Date / /

EXPERIMENT -3

Aim: Write a program in Python in Python to create a series with index as data range of 12, access the elements using 'loc' & 'iloc'

```
import pandas as pd
date_range = pd.date_range(start='2023-01-01', period=12,
                           freq='M')
data = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
series = pd.Series(data, index = date_range)
print(series)

print("Using loc")
print(series.loc['2023-03-31'])
print(series.loc['2023-04-30'])

print("Using iloc")
print(series.iloc[2])
```

2) Create a dataframe reading from a csv file & perform slicing using loc & iloc. Sort the data frame in descending order & describe the dataframe.

Sliced Data Using Iloc:

	PassengerID	Survived
2	3	1
3	4	1
4	5	0

Sorted Dataframe:-

P.Id	Survived	Pclass	Name	Sex	Age
630	631	1	1	Barkworth,	80
851	852	0	3	Svensson	74
493	494	0	1	Astagavaydh	71

Dataframe Description

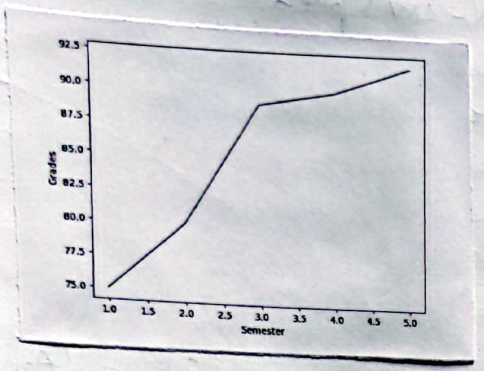
	P.Id	Survived	Pclass	Age
Count	891.000 000	891.000000	891.000000	714.000
Mean	446.000 000	0.383838	2.308642	29.69911
Std	257.353 842	0.486592	0.836071	14.52649
min	1.000 000	0.000000	1.000000	0.420
25%	223.500.000	0.000000	2.000 000	20.125
50%	446.000000	0.000000	3.000000	28.000
75%	668.500000	1.000000	3.000 000	38.000
Max	891.000000	1.000000	3.000000	80.000

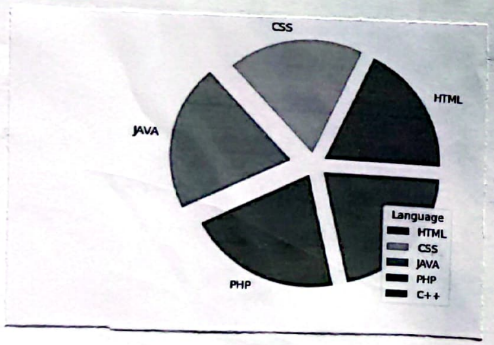
DELTA Pg No. Date / /

```
import pandas as pd
path = "D:\Machine Learning\Titanic-Dataset.csv"

df = pd.read_csv(path)

sliced_data_loc = df.loc[2:4, ['PassengerId', 'Age']]
print(sliced_data_loc)

sliced_data_iloc = df.iloc[2:5, 0:2]
print(sliced_data_iloc)

sorted_df = df.sort_values(by='Age', ascending=False)
print(sorted_df.head(3))

print(df.describe())
```

Observation

In Machine Learning, series is a one-dimensional labelled array for storing data of various types. loc & iloc is being used for label-based data & integer-based data respectively in dataframes. Dataframe is like a two-dimensional array labelled for storing data of various types.





DELTA Pg No.
Date / /

EXPERIMENT-4

Aim: Create plot using matplotlib pyplot

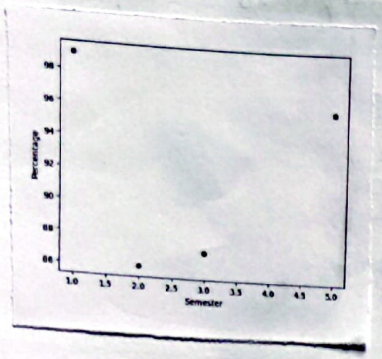
1) Semester - wise grade line graph

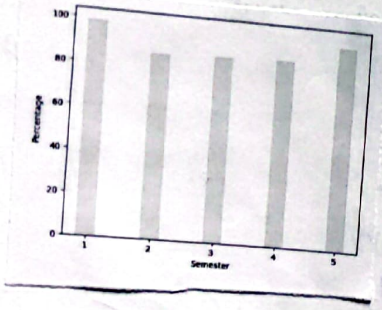
Import matplotlib.pyplot as plt

```
semester = [1, 2, 3, 4, 5]
grades = ['A', 'B', 'A+', 'B+', A]

plt.plot (semesters, grades, marker = 'o', linestyle = '-')
plt.title (Semester Wise Grades)
plt.xlabel ('Semester')
plt.ylabel ('Grade')
plt.show()
```

2) Pie Chart depicting your core competency in each of the programming Language

```
Import numpy as np
Language = ['HTML', 'CSS', 'JAVA', 'PHP', 'C++']
Competency = np.array ([80, 80, 89, 90, 92])
myexplode = [0.1, 0.1, 0.2, 0.1, 0.1]
plt.pie (Competency, labels = Language, explode = myexplode)
plt.legend (title = "Language")
plt.show()
```





iii) Scatter plot depicting your percentage till semester 5. Use different color for different semester.

```
Semester = np.array ([1,2,3,4,5])
Percentage = np.array ([99,86,87,88,96])
colors = ['red', 'green', 'blue', 'yellow', 'purple']

plt.xlabel ("Semester")
plt.ylabel ("Percentage")
plt.scatter (Semester, Percentage, alpha=0.8, c=colors)
plt.show()
```

iv) Bar chart (vertically & Horizontally depicting the percentage till 5th semester.

→ Use width & height attribute, different color for vertical & horizontal bar chart.

```
Semester = np.array ([1,2,3,4,5])
percentage = np.array ([99,86,87,88,96])

plt.xlabel ("Semester")
plt.ylabel ("Percentage")
plt.bar (Semester, percentage, color = "Pink", width=0.3)

plt.show ()
```

DELTA Pg No.
Date / /

Horizontal
plt.xlabel ("Percentage")
plt.ylabel ("Semester")
plt.barh (Semester, percentage, color = 'blue', alpha = 0.5, height = 0.3)
plt.show ()

1) Create a subplot in 3×2
Import matplotlib.pyplot as plt
Import numpy as np

x = np.array ([0, 1, 2, 3])
y = np.array ([3, 8, 1, 10])
plt.subplot (2, 3, 1)
plt.plot (x, y)

x = np.array ([0, 1, 2, 3])
y = np.array ([10, 20, 30, 40])
plt.subplot (2, 3, 2)
plt.plot (x, y)

x = np.array ([0, 1, 2, 3])
y = np.array ([2, 4, 6, 8])
plt.subplot (2, 3, 3)
plt.plot (x, y)

DELTA Pg No.
Date / /

```
x = np.array ([0, 1, 2, 3])
y = np.array ([3, 6, 9, 12])
plt.subplot (2,3,4)
plt.plot (x, y)

x = np.array ([0, 1, 23])
y = np.array ([3, 8, 23, 10])
plt.subplot (2,3,5)
plt.plot (x,y).

x = np.array ([0, 1, 2, 3])
y = np.array ([10, 15, 20, 25])
plt.subplot (2,3,6)
plt.plot (x,y)
```

Observation

Matplotlib helps with data visualization.
plot() function creates a single plot with customization options.
subplot() makes multiple subplots in a single figure.
show() display the plot

EXPERIMENT - 5

Working with the seaborn library to

1) Create a distribution plot with histogram

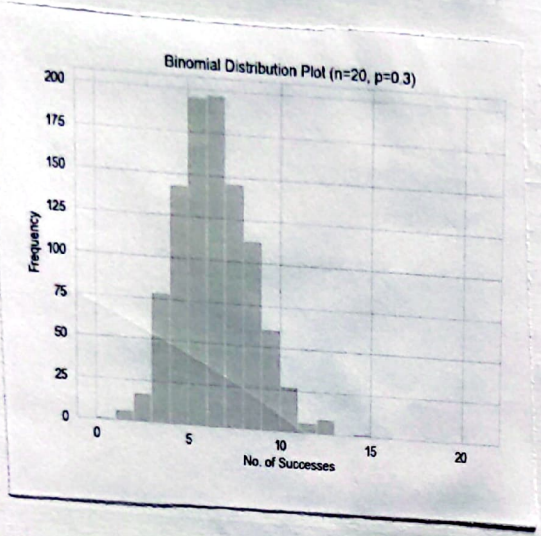
```
import matplotlib.pyplot as plt
import seaborn as sns
data = [2, 4, 6, 8, 10]
sns.distplot(data, hist=False)
plt.title("Distribution Plot without Histogram")
plt.show()
```

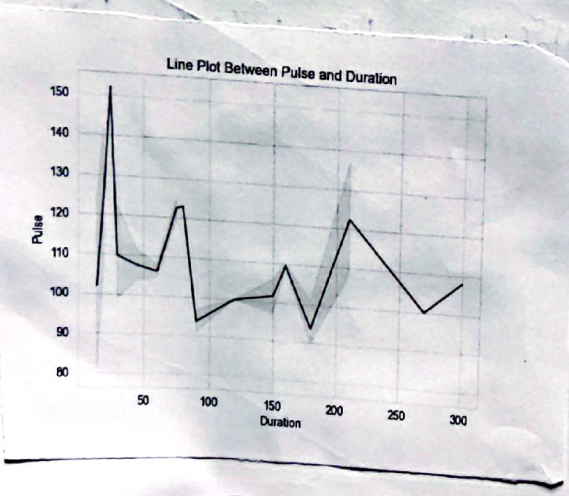
2) Create a distribution plot without histogram

```
import seaborn as sns
data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sns.distplot(data, hist=False)
plt.title("Distribution Plot without Histogram")
```

3) Create a Normal distribution Plot

```
import matplotlib.pyplot as plt
import seaborn as sns
data = [0, 5, 10, 15, 20, 25]
sns.distplot(data, Kde=True)
plt.title("Normal Distribution Plot")
```



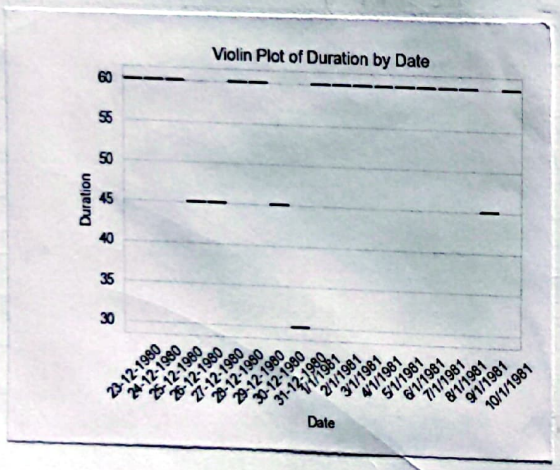


iv) Create a Binomial Distribution Plot. Add the corresponding labels.

```
Import matplotlib.pyplot as plt
Import numpy as np
Import seaborn as sns

n=20
p=0.3
data = np.random.binomial (n,p,1000)
sns.set_style("whitegrid")
sns.distplot (data, bins = range (0,n+2), kde = False)

plt.xlabel ("No of success")
plt.ylabel ("Frequency")
plt.title ("Binomial Distribution Plot(n={n}, p={p})")
plt.show()
```

v) Use of csv file to obtain the plot b/w two parameters.

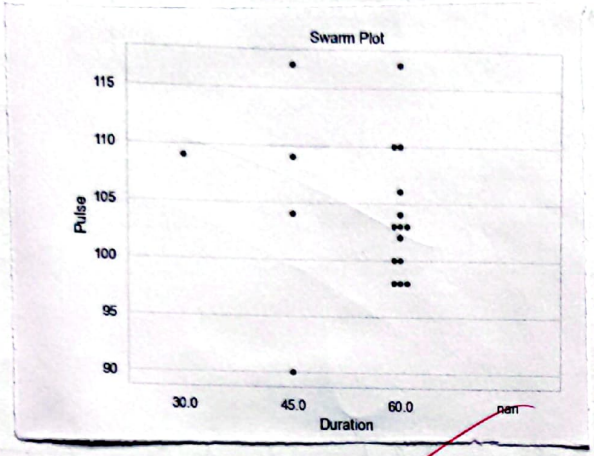
```
Import pandas as pd
Import matplotlib.pyplot as plt.
Import pandas as pd.

df = pd.read_csv('data_new.csv')
```

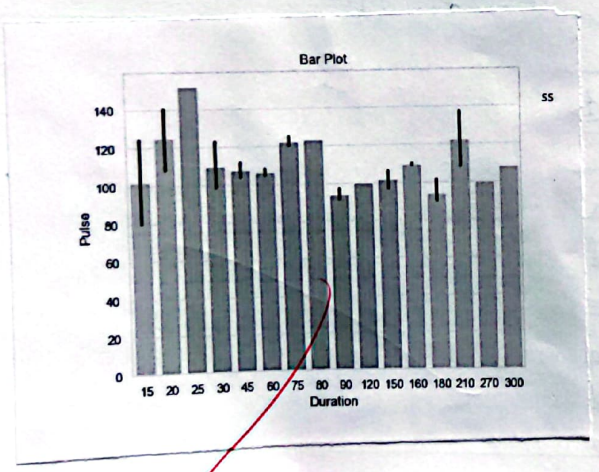



```
sns.set_style("whitegrid")
sns.scatterplot(x='Duration', y='Pulse', data=df,
                hue='Calories')
plt.xlabel('Duration')
plt.ylabel('Pulse')
plt.title('Scatter Plot with hue')

plt.legend(title='Calories')
plt.show()
```

Viva) Draw a violin plot b/w the parameter

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

df = pd.read_csv('data_new1.csv')

sns.violinplot(x='date', y='Duration', data=df)
plt.xlabel('Date')
plt.ylabel('Duration')
plt.title('Violin Plot of Duration by Date')
plt.xticks(rotation=45)
plt.show()
```





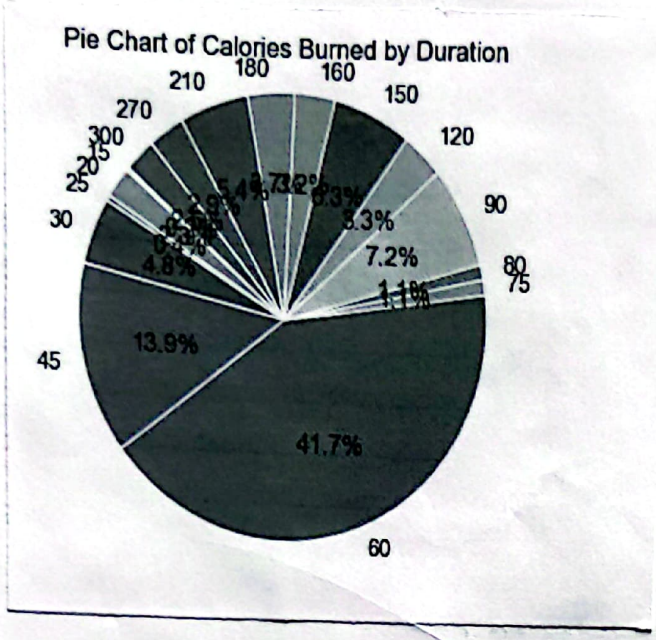
DELTA Pg No.
Date / /

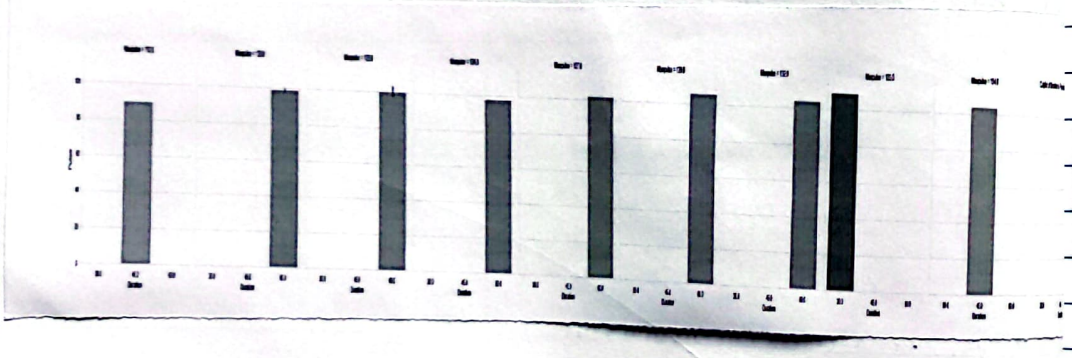
9) Draw a Swarm Plot

```
Import seaborn as sns.
Import matplotlib.pyplot as plt
Import pandas as pd.

data = pd.read_csv("data_new1.csv")
sns.set(style='whitegrid')
sns.swarmplot(x='Duration', y='Pulse', data=data)

plt.xlabel("Duration")
plt.ylabel("Pulse")
plt.title("Swarm plot")
plt.show()
```

x) Bar plot b/w the parameter

```
Import seaborn as sns.
Import matplotlib.pyplot as plt
Import pandas as pd.

data = pd.read_csv("data_new.csv")
sns.barplot(x:"Duration", y:"Pulse", data=data)
plt.xlabel("Duration")
plt.ylabel("Pulse")
plt.title("Barplot")
plt.show()
```





vi) Draw a pie-chart

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv('data_new.csv')
duration_sum = df.groupby('Duration')['Calories'].sum()

plt.pie(duration_sum, labels = duration_sum.index,
        autopct = '%1.1f%%');
plt.title('Pie Chart of Calories Burned by Duration')
plt.show
```

vii) Draw a bar plot

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

df = pd.read_csv('dataset1.csv')
sns.set_style('whitegrid')

sns.catplot(x='Duration', y='Pulse', col='Maxpulse',
            kind='bar', data='df')
```

plt.xlabels ('Duration')
plt.ylabels ('Pulse')

plt.title ("Catplot of Duration Vs Pulse with Maxpulse Column")

plt.show()

Observation

Boxplot were easier to understand & simple use of Median, Quartiles, Interquartile ranges, Outliers

On Other hands Violinplots are more informative they are like a shape of violin which helps in comparison.

Swarmplots / Strip plot used to compare the the heights of men & women, or the ages of students in different grades.

Output

. . . iris_dataset:

Iris plant dataset

** Data Set Characteristics **

: No. of Instances : 150 (50 in each of 3 class)

: No. of Instances : 4 numeric, predictive attributes & the class

: Attribute Information:

- Sepal length in cm
- Sepal width in cm
- petal length in cm
- petal ~~length~~ width in cm
- Class
  - Iris-setosa
  - Iris-versicolour
  - Iris-virginica

: Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
Sepal length:	4.3	7.9	5.84	0.83	0.7826
Sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	1.76	1.76	0.9490 (high)
petal width:	0.1	2.5	0.76	0.76	0.9565 (high)

DELTA Pg No. Date

Practical - 6

i) Import iris dataset from sklearn
ii) See its description
iii) Extract its target data
iv) Draw scatter plot b/w dependent & independent variable

```
# Import matplotlib.pyplot as plt.
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.DESCR)

x = iris.data
y = iris.target

dfx = pd.DataFrame(x)
dfy = pd.DataFrame(y)

feature_names = iris.feature_names
target_names = iris.target_names
print("Features names:", feature_names)
print("Target names:", target_names)
```

: Missing Attribute values : None
: Class Distribution : 33.3% for each of 3 classes
: Creator : R. A. Fisher
: Donor : Michael Marshall (MARSHALL%P.L.U@10.arc.nasa.gov)
: Date : July, 1988

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

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

Features Value :

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Target Value :

	0
0	0
1	0
2	0
3	0
4	0

DELTA Pg No. Date / /

```
print ("Feature value : \n", dfx.head())
print ("Target value : \n", dfy.head())

plt.figure (figsize = (8,6))
plt.scatter (x[:,0], x[:,1], c=y, cmap='viridis')

plt.xlabel ('Sepal Length (cm)')
plt.ylabel ('Sepal Width (cm)')

plt.title ("Scatter plot : Sepal length vs Sepal width")
plt.colorbar (label = 'Species')
plt.show()
```

Observation
Scikit learn is the ML library, using scikit learn we have in built datasets to work with like iris, diabetes, digit scikit-learn dataset helps to quickly illustrate the behaviour of various Algorithm implemented in scikit learn.

> numpy loadtxt

```
[[ 6.   148.   72   0.   33.6   0.627   50   1.]
 [ 1.    85.   66.  0.   26.6   0.351   31.  0.]
 [ 8.   183.   64.  0.   23.3   0.672   22   1.]]
```

> File with header

```
[ 'Gender', 'Group', 'License']
  [ ['Male', 'A', '0']
    ['female', 'A', '1']
    ['female', 'B', '1'] ]
```

DELTA Pg No. Date / /

Practical -7

Aim: To perform dataset loading using numpy, pandas, & csv module.

1) Using numpy loadtxt

```
from numpy import loadtxt
path = "D:\MACHINE LEARNING\pima-indians-diabetes.csv"
file = open('path', 'r')
data = nploadtxt('path', delimiter=",")
print (data)
```

2) file with header

```
import numpy as np
import csv
path = "D:\MACHINE LEARNING\gues.csv".

with open(path, 'r') as f:
    reader = csv.reader(f, delimeter=',')
    headers = next(reader)
    data = list(reader)
    data = np.array(data)
print(headers)
print(data[:3])
```

	Duration	Pulse	Maxpulse	Calories	date
0	60.0	110	Nan	NaN	23-12-1980
1	60.0	117	Nan	479.0	24-12-1980
2	60.0	103	Nan	340.0	25-12-1980
	60	115	NaN	310.2	7-6-1981
	75	120	150.0	320.4	8-6-1981
	75	125	NaN	330.4	9-6-1981

> Adding Headers

	Gender	Group	Uerse
0	Male	A	0
1	Female	A	1
2	Female	B	1
3	Male	A	1
4	Female	B	1

iii) Using pandas read_csv

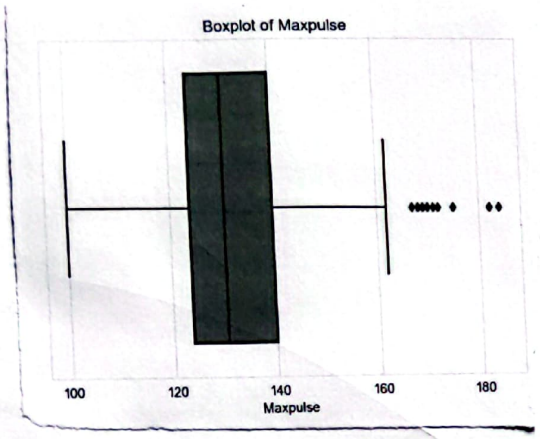
```
import pandas as pd.
df = pd.read_csv ('data_new1.csv')
print (df.head (3))
print (df.tail (3))
```

iv) Adding header in file and loading

```
from pandas import read_csv
header_name = ['Gender', 'Group', 'Uerse']
df = pd.read_csv('gues.csv', names = headername)

print(df)
```

Observations

Here we load data using numpy loadtxt, genfromtxt and readcsv, if we want to skip the row we use next() method, in read csv we can also add header names if not names in the file using names attribute.



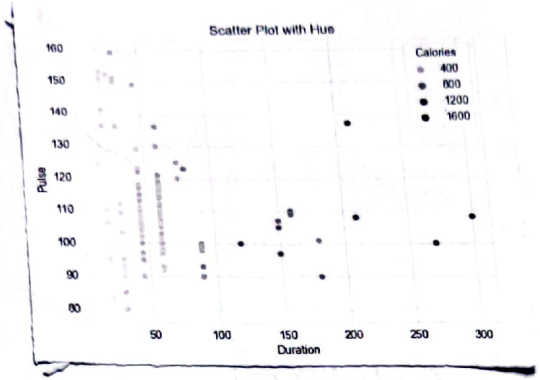
DELTA Pg No.
Date / /

Practical - 8

Aim: To perform the outlier analysis using z-score & IQR i.e. statistical visualization technique (box & scatter plot)

```
import numpy as np
import pandas as pd
import seaborn.pyplot as plt
from scipy import stats
iris = sns.load_dataset("iris")

numerical_columns = ['sepal_length', 'sepal_width', 'petal_length',
                     'petal_width']
data = iris[numerical_columns]

def detect_outliers_zscore(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3-Q1
    return ('data < (Q1 - 1.5* IQR))| (data > (Q3 +1.5* IQR))')

outliers_zscore = detect_outliers_zscore(data)
outliers_iqr = detect_outliers_iqr(data)

plt.figure(figsize=(12,6))
```

z-score.




```
plt.subplot(1,2,1)
sns.boxplot(data=data, orient='y', palette='Set2')
plt.title("Box plot of Iris Features")

plt.subplot(1,2,2)
for feature in numerical_columns:
    sns.scatterplot(data=data, x=feature,
                    y=Iris['species'], palette='Set2')
plt.title("Scatter plot of Iris features")

plt.tight_layout()
plt.show()

print("Outliers detected using z-score:")
print("Outliers zscore")
print("Outliers detected using IQR:")
print(Outliers_iqr)
```

Observations:

- Z-score method identified outliers with indices
- IQR method found outliers based on quartile ranges
- Box plot visually displayed data spread & outliers
- Scatter plot showed data distribution across Iris species.

Outliers analysis highlights potential anomalies in Iris dataset.

	Age	Income	Score
0	25.00	50000.0	85.0
1	30.00	53750.0	92.0
2	26.25	60000.0	78.0
3	22.00	75000.0	87.6
4	28.00	53750.0	88.0
5	26.25	30000.0	95.0

	Employee ID	Gender	Work Category	Gender new	...	1	2	3	4
0	1	Male	Engineering	1	...	1.0	1.0	0.0	0.0
1	2	Female	Sales	0	...	0.0	0.0	0.0	1.0
2	3	Male	Marketing	1	...	1.0	0.0	1.0	0.0

[3 rows x 10 columns]

DELTA Pg No.
Date / /

Practical - 9

Aim: To handle the numerical data & perform one-hot encoding on categorical data

1) Handling Numerical data

```
Import numpy as np
Import pandas as pd
data = { 'Age' : [25, 30, np.nan, 26.25, 28, np.nan],
         'Income' : [50000, np.nan, 60000, 75000, np.nan, 30,000 ]
         'Score' : [85, 92, 78, np.nan, 88, 95] }

df = pd.DataFrame (data)
mean_Imputer = df.mean()
df. fillna ( mean_Imputer, inplace=True )
print (df)
```

2) Perform One-hot Encoding

```
Import pandas as pd
from sklearn.preprocessing import OneHotEncoder

data = pd . read_csv ('Employee_data.csv')
```

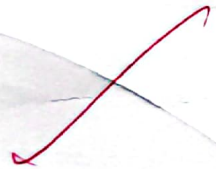
DELTA Pg No.
Date / /

```
# Convert columns with categorical value to datatype category
data['Gender'] = data['Gender'].astype('category')
data['Work_category'] = data['work_category'].astype('category')

# Obtain category codes
data['Gender_new'] = data['Gender'].cat.codes
data['Work_category_new'] = data['work_category'].cat.codes

enc = OneHotEncoder()

# Pass encodes (category code) columns to the instance
  & convert it to dataframe

enc_data = pd.DataFrame(enc.fit_transform(data[['Gender_new',
                'Work_category_new']]).toarray())

# merge with main

new_df = data.join(enc_data)
print(new_df)
```

Observations :-

- The functions are used in one hot encoding in Python are:-
  1) get_dummies() : It takes a dataframe & returns a new Dataframe with encoded columns of categorical variables
  2) OneHotEncoder() : This takes list of cat. variables & returns a one-hot encoded representation

```
64.88935388 64.88935388 67.82564561 80.67192189 64.88935388 70.39490086
64.88935388 72.96415612 66.35749974 73.69822905 68.926755   68.926755
66.72453621 66.72453621 66.35749974 70.76193733 67.82564561 66.35749974
71.86304672 68.55971854 79.57081249 67.45860914 72.96415612 65.62342681
64.52231742 77.00155724 65.25639035 66.72453621 71.86304672 67.82564561
65.99046328 69.66082793 73.69822905 64.88935388 79.57081249 65.62342681
64.52231742 64.88935388 65.99046328 67.82564561 67.09157268 64.52231742
81.03895835 67.09157268 75.16637491 64.52231742 73.33119258 65.62342681
68.19268207 65.62342681 66.72453621 65.25639035 67.82564561 67.09157268
64.88935388 67.45860914 64.52231742 65.99046328 72.59711965 65.25639035
68.55971854 65.25639035 65.62342681 64.52231742 75.16637491 74.79933845
66.35749974 68.55971854 71.49601026 67.09157268 68.19268207 72.23008319
65.25639035 65.25639035 72.59711965 70.0278644  71.12897379 65.99046328
75.53341138 67.45860914 67.82564561 73.69822905 64.52231742 64.88935388
65.62342681 67.09157268 71.86304672 71.12897379]
[ 62   0  52  68  64  70  82  62 104  84  74  72  94  85  70  90  70  60
  72  76  70  60  64  58  88  64  70  88  72  74  68  92  68  78  72  44
  64  70  70  62  58  65  70  60  94 110  72  86  74  50  88  80  66  58
  92   0  76  64  62  86  88  54   0  65  60  62   0  70   0 108  66  88
  84  76  50  62  98  70  65  88  68  64  68  88  66  74  70  78   0  78
  65  72  76  82  74  70  80  84  82  80  54   0  60  58  62  92  72  56
  74  58  74  60 100  52  88  88  72  70  70  70  82  80  75  90  80  60
  74  58  76  68  90  88   0  75  78  78   0  78  60  86  90  78  70  62
 106  70  64  72  78  84  68  74  80  54]
Accuracy: 0.038961038961038966
Precision: 0.038961038961038966
Recall: 0.038961038961038966
```



EXPERIMENT - 10

DELTA Pg No. Date / /

To implement Linear Regression (Dataset = blood-pressure_data.csv)

```
Import pandas as pd
Import numpy as np
Import matplotlib.pyplot as plt
from sklearn.model_selection Import train_test_split
from sklearn.linear_model Import LinearRegression
from sklearn.metrics Import accuracy_score,
                             precision_score, recall_score

# Obtain Dependent & Independent variables
  df = pd.read_csv('blood_pressure_data.csv')
  X = df['Age'].values.reshape(-1,1)
  y = df['BloodPressure'].values

# Split the dataset into training & testing sets
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

# Train & test the model
  model = LinearRegression()
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  print(y_pred)
  print(y_test)
```

DELTA Pg No.
Date / /

```
# Calculate accuracy, precision & recall
acc = accuracy_score(y_test, y_pred.round())  # no avg attribute
prec = precision_score(y_test, y_pred.round(), average='micro')
rec = recall_score(y_test, y_pred.round(), average='micro')

print(f'Accuracy: {acc}')
print(f'Precision: {prec}')
print(f'Recall: {rec}')
```

Observation

- We got a matrix of values along with our accuracy, precision & recall scores of Linear Regression.

=

- train_test_split(): This function splits a Dataframe into two Dataframes (training & testing)
- pred_round(): It rounds the predicted values of Linear Model.
- average = micro: argument specifies accuracy is calculated by averaging the correct no. of predictions

```
Accuracy: 0.7337662337662337
Precision: 0.7425742574257426
Recall: 0.8333333333333334
```

Observation:-

- Logistic Regression make predictions about binary outcomes i.e TRUE or FALSE etc
- It can be used to control confounding variables, which are variables that are related to both independent variable & binary outcome.

EXPERIMENT-11

Date

To Implement Logistic Regression (Dataset = blood pressure_data.csv)

```
Import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score

df = pd.read_csv('blood_pressure_data.csv')
X = df['Age'].values.reshape(-1,1)
y = (df['BloodPressure'] > df['BloodPressure'].mean()).astype(int)
# Binary classification, 1 is above mean else 0 if below

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size=0.2, random_state=42)

model = LogisticRegression() # By default active fun. is Sigmoid
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print(f'Accuracy : {accuracy}')
print(f'Precision : {precision}')
print(f'Recall : {recall}')
```

25/10/23