Ex. No: 1	PLOTTING WITH MATPLOTLIB LIBRARY
09 / 03 / 22	

AIM:

To learn plotting with ‘MatPlotLib’ library in Python.

QUESTION:

Create data and plot the following:

- Line plot.
- Stem plot.
- Box plot.
- Scatter plot.
- Bar chart.
- Pie chart.
- Histogram.

Create a sine wave and introduce some noise into it. Also, print a wave with the noise wave concatenated to the sine wave.

ALGORITHM:

Step 1: Create datapoints
Step 2: Plot a line plot, stem plot, box plot and scatter plot.
Step 3: Plot a bar and pie chart.
Step 4: Plot a histogram.
Step 5: Create a sine wave.
Step 6: Create a noise wave.
Step 7: Append the noise wave to the end of sine wave (concatenated wave).
Step 8: Add individual datapoints of sine and noise wave (added wave).
Step 9: Plot all 4 waves

Program:

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

# Line, Stem, Box, Scatter Plot
# Bar, Pie Chart
# Histogram

np.random.seed(42)

'''## Line plot '''

ypoints = np.arange(1000)

plt.plot(ypoints, linestyle = 'dotted')
plt.show()


'''## Stem plot '''

# Make data
x = 0.5 + np.arange(8)
y = np.random.uniform(2, 7, len(x))

# Plot
fig, ax = plt.subplots()
ax.stem(x, y)
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()


'''## Box Plot '''

# Make data
np.random.seed(10)
D = np.random.normal((3, 5, 4), (1.25, 1.00, 1.25), (100, 3))

# Plot
fig, ax = plt.subplots()
VP = ax.boxplot(D, positions=[2, 4, 6], widths=1.5, patch_artist=True,
                showmeans=True, showfliers=True,
                medianprops={"color": "white", "linewidth": 0.5},
                boxprops={"facecolor": "C0", "edgecolor": "white",
                          "linewidth": 0.5},
                whiskerprops={"color": "C0", "linewidth": 1.5},
                capprops={"color": "C0", "linewidth": 1.5})

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))
```

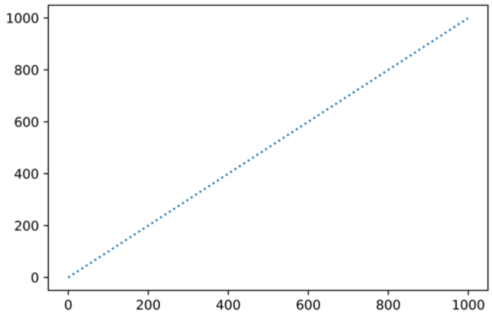
```
plt.show()


'''## Scatter Plot '''

# Make data
x = 4 + np.random.normal(0, 2, 24)
y = 4 + np.random.normal(0, 2, len(x))

# Size and color
sizes = np.random.uniform(15, 80, len(x))
colors = np.random.uniform(15, 80, len(x))

# Plot
fig, ax = plt.subplots()
ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()


'''## Bar Chart '''

# Make data
x = 0.5 + np.arange(8)
y = np.random.uniform(2, 7, len(x))

# Plot
fig, ax = plt.subplots()
ax.bar(x, y, width=1, edgecolor="white", linewidth=0.7)
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()


'''## Pie Chart '''

# Make data
x = [1, 2, 3, 4]
colors = plt.get_cmap('Blues')(np.linspace(0.2, 0.7, len(x)))

# Plot
fig, ax = plt.subplots()
ax.pie(x, colors=colors, radius=3, center=(4, 4),
       wedgeprops={"linewidth": 1, "edgecolor": "white"}, frame=True)
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```

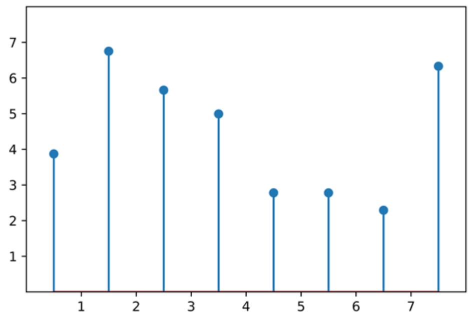
```
'''## Histogram '''

# Make data
x = 4 + np.random.normal(0, 1.5, 10000)

# Plot
fig, ax = plt.subplots()
ax.hist(x, bins=100, linewidth=0.5)

plt.show()


'''## Sine wave '''

# Index values
n = np.arange(1000)

signal = np.sin(2 * np.pi * 50 * n / 1000)
# x(n) = sin((2 * pi * wave_freq * n) / sampling_freq)
# Note: sampling_freq >= 2 * wave_freq
plt.plot(signal[100:200], linewidth=2.0)
plt.show()

# Create noise wave
noise = np.random.normal(0, 0.43, 1000)
plt.plot(noise[100:200])
plt.show()

# Append noise to signal wave
signal_noise_concat = np.concatenate([signal,noise])
plt.plot(signal_noise_concat[850: 1150])
plt.show()

# Add signal & noise
signal_noise_add = np.add(signal,noise)
plt.plot(signal_noise_add)
plt.show()

# Plot all figures
plt.subplot(2,2,1).title.set_text("Signal")
plt.plot(signal[100:200],color = 'violet')

plt.subplot(2,2,2).title.set_text("Noise")
plt.plot(noise[100:200],color = 'brown')

plt.subplot(2,2,3).title.set_text("Concatenated wave")
plt.plot(signal_noise_concat[850:1200], color = 'blue')

plt.subplot(2,2,4).title.set_text("Added wave")
plt.plot(signal_noise_add[100:200], color = 'green')
```

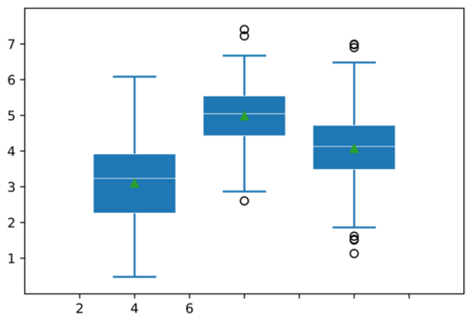
```
plt.subplots_adjust(wspace=0.5, hspace=0.7)
plt.show()
```

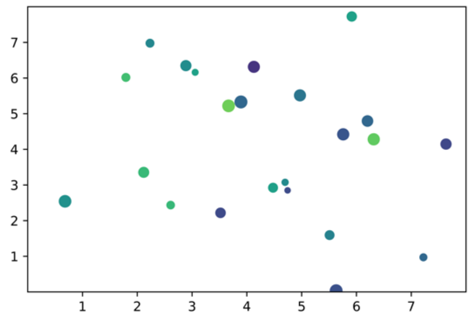
RESULT:

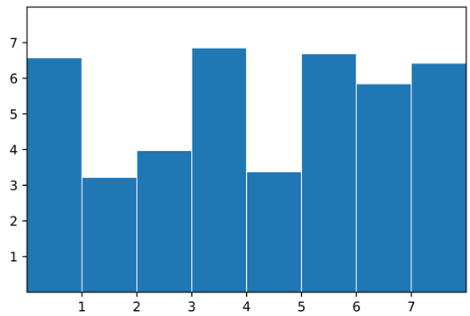
Concepts related plotting with ‘MatPlotLib’ library were studied.

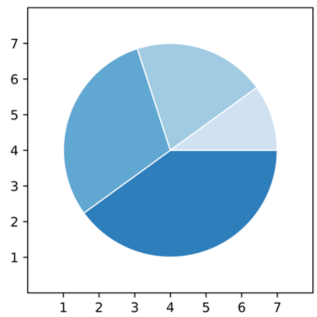

0
200
400
600
800
1000

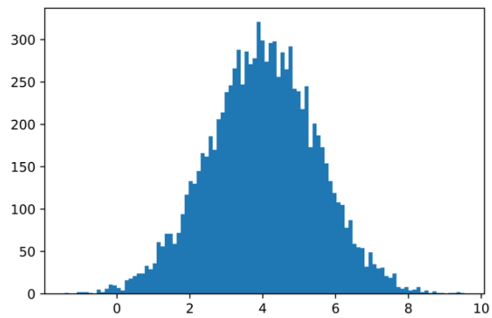

1
2
3
4
5
6
7

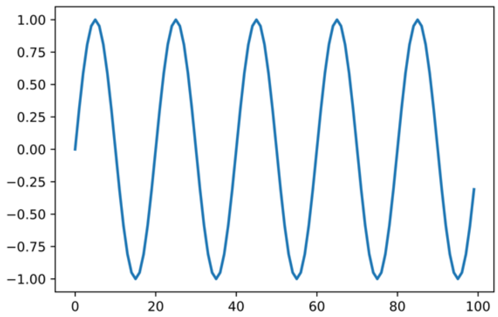

1
2
3
4
5
6
7

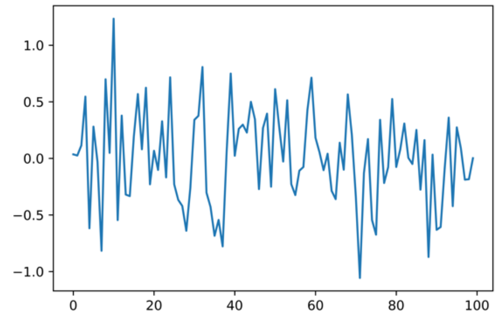

1 2 3 4 5 6 7
1 2 3 4 5 6 7

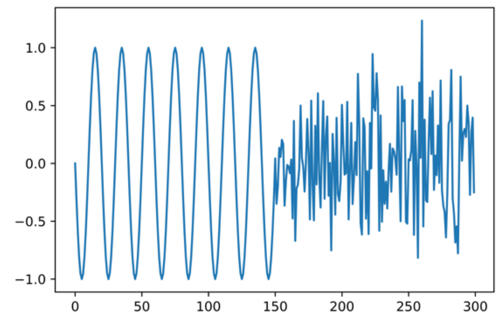

1 2 3 4 5 6 7
1 2 3 4 5 6 7

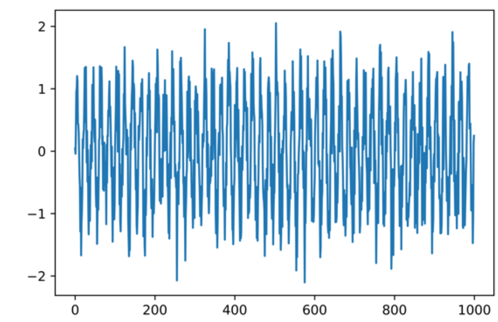

1 2 3 4 5 6 7
1 2 3 4 5 6 7

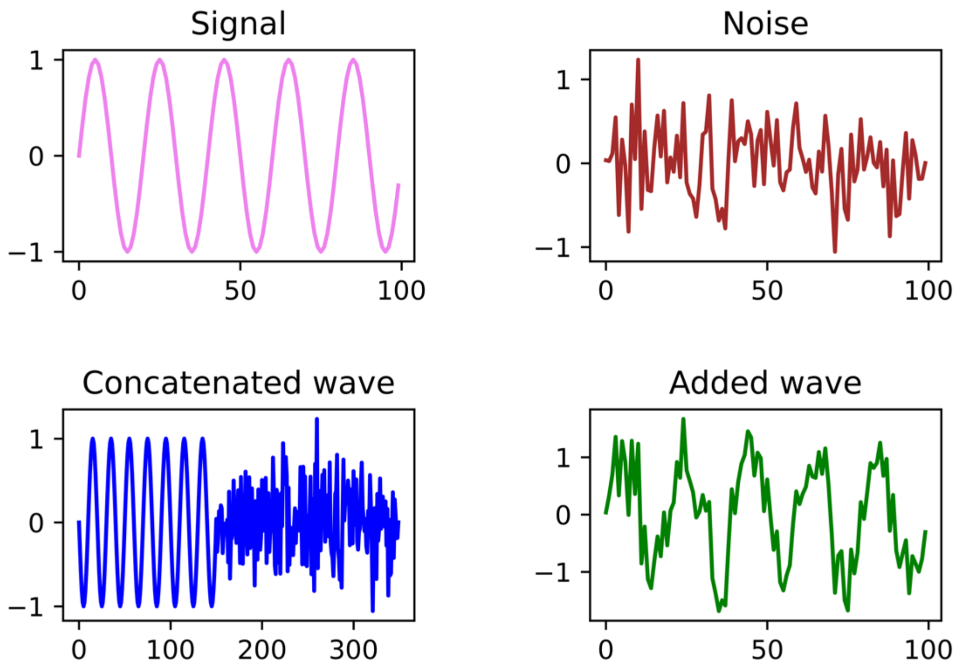

0 50 100 150 200 250 300
0 2 4 6 8 10


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0 20 40 60 80 100


1.0
0.5
0.0
−0.5
−1.0
0 20 40 60 80 100


1.0
0.5
0.0
−0.5
−1.0
0 50 100 150 200 250 300


2
1
0
−1
−2
0 200 400 600 800 1000



Signal
Noise
Concatenated wave
Added wave

Ex. No: 2	REGRESSION – SIMPLE REGRESSION
23 / 03 / 22	

AIM:

To learn about simple regression in Python.

QUESTION:

Create N random numbers for the independent variable, x_i, using the following command:

x = np.random.randn(N)

Create N random number for the dependent variable, y_i, using the following command, with different values for δ (for example, 0.1 < δ < 2):

y = x + δ ∗ np.random.randn(N)

Implement the following two equations, where $\bar{x}$ and $\bar{y}$ are the mean values of x_i and y_i, and c and m are the y-intercept and the slope of the straight line (Y = mX + c):

$$c = \bar{y} - m\bar{x}$$

$$m = \frac{\sum_{i=1}^{N}(\mathrm{xiyi} - \bar{y}\mathrm{xi})}{\sum_{i=1}^{N}(\mathrm{xi}^2 - \bar{x}\mathrm{xi})}$$

Compute the sum of squared error (SSE) using the following equation:

$$e = \sum_{i=1}^{N}(\bar{y}_i - \mathrm{yi})^2$$

Print the values of m, c, and e.

Vary the value of δ, in the following equation, between 0 and 2, in steps of 0.2, tabulate the m, c, and e, and give your observations.

Perform the above steps for the following equations of ‘y’:

y = x + δ ∗ np.random.randn(N)

y= x + 2

y = 2x – 3

ALGORITHM:

Step 1: Create independent variable ‘x’ & dependent variable ‘y’ using equations in question.
Step 2: Find slope ‘m’ and ‘y-intercept’.
Step 3: Plot regression line.
Step 4: Calculate SSE (sum-squared error) between actual and predicted values of ‘y’.
Step 5: Print slope, intercept, and sum-squared error for increasing values of delta (deviation of ‘y’ values from equation).
Step 6: Plot the graph for all values of delta (deviation).

Program A:

```
"""
Exercise 1: Linear Regression
Equation 1: y = x
"""

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(100)

# Number of samples
N = 100

plot = 0

# Print delta, slope, intercept and error in a table
print("Delta\t\tSlope\t\t\t\t\tIntercept\t\t\t\t\t\tSSE")

for num in range(1, 21, 2):
    # Variation -> Delta (0.1 to 1.9 with step increase of 0.2 )
    delta = num / 10

    # Independent variable
    x = np.random.randn(N)

    # Dependent variable
    y = x + delta * np.random.randn(N)

    # Mean
    xMean = sum(x) / N
    yMean = sum(y) / N

    # Slope 'm'
    numerator = 0
    denominator = 0

    for i in range(N):
        numerator += x[i] * y[i] - x[i] * yMean
```

Program B:

```
"""
Exercise 1: Linear Regression
Equation 2: y = x + 2
"""

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(100)

# Number of samples
N = 100

plot = 0

# Print delta, slope, intercept and error in a table
print("Delta\t\tSlope\t\t\t\t\tIntercept\t\t\t\t\t\tSSE")

for num in range(1, 21, 2):
    # Variation -> Delta (0.1 to 1.9 with step increase of 0.2 )
    delta = num / 10

    # Independent variable
    x = np.random.randn(N)

    # Dependent variable
    y = x + 2 + delta * np.random.randn(N)

    # Mean
    xMean = sum(x) / N
    yMean = sum(y) / N

    # Slope 'm'
    numerator = 0
    denominator = 0

    for i in range(N):
        numerator += x[i] * y[i] - x[i] * yMean

    for i in range(N):
        denominator += x[i] * x[i] - x[i] * xMean

    m = numerator / denominator

    # y - Intercept
    c = yMean - m * xMean

    # Line - Points on line are predicted values of 'y' for each 'x'
    xLine = x
    yLine = m * xLine + c
```

```
    # SSE -> Sum Squared Error (Error between actual 'y' & predicted 'y')
    e = 0

    for i in range(N):
        e += (yLine[i] - y[i]) ** 2

    # Print Table
    print(delta, m, c, e, sep = '\t\t')

    # Subplot
    plot += 1
    plt.subplot(5, 2, plot)
    plt.subplots_adjust(top=1,hspace=2, wspace=0.3)
    plt.grid(linewidth = 0.2, linestyle = '--')
    plt.title("\u0394 = %.1f" % delta)

    # Plot data points
    plt.scatter(x,y, s = 0.7)

    # Plot line
    plt.plot(xLine, yLine, color = 'red', linewidth=0.5)

# Save graph as an image
plt.savefig("linearRegression2.png", dpi = 1200, bbox_inches='tight')
```

Program C:

```
"""
Exercise 1: Linear Regression
Equation 3: y = 2x - 3
"""

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(100)

# Number of samples
N = 100

plot = 0

# Print delta, slope, intercept, and error in a table
print("Delta\t\tSlope\t\t\t\t\tIntercept\t\t\t\t\t\tSSE")

for num in range(1, 21, 2):
    # Variation -> Delta (0.1 to 1.9 with step increase of 0.2 )
    delta = num / 10

    # Independent variable
    x = np.random.randn(N)

    # Dependent variable
    y = 2 * x - 3 + delta * np.random.randn(N)

    # Mean
    xMean = sum(x) / N
    yMean = sum(y) / N

    # Slope 'm'
    numerator = 0
    denominator = 0

    for i in range(N):
        numerator += x[i] * y[i] - x[i] * yMean

    for i in range(N):
        denominator += x[i] * x[i] - x[i] * xMean

    m = numerator / denominator

    # y - Intercept
    c = yMean - m * xMean

    # Line - Points on line are predicted values of 'y' for each 'x'
    xLine = x
    yLine = m * xLine + c
```

```
    # SSE -> Sum Squared Error (Error between actual 'y' & predicted 'y')
    e = 0

    for i in range(N):
        e += (yLine[i] - y[i]) ** 2

    # Print Table
    print(delta, m, c, e, sep = '\t\t')

    # Subplot
    plot += 1
    plt.subplot(5, 2, plot)
    plt.subplots_adjust(top=1,hspace=2, wspace=0.3)
    plt.grid(linewidth = 0.2, linestyle = '--')
    plt.title("\u0394 = %.1f" % delta)

    # Plot data points
    plt.scatter(x,y, s = 0.7)

    # Plot line
    plt.plot(xLine, yLine, color = 'red', linewidth=0.5)

# Save graph as an image
plt.savefig("linearRegression3.png", dpi = 1200, bbox_inches='tight')
```

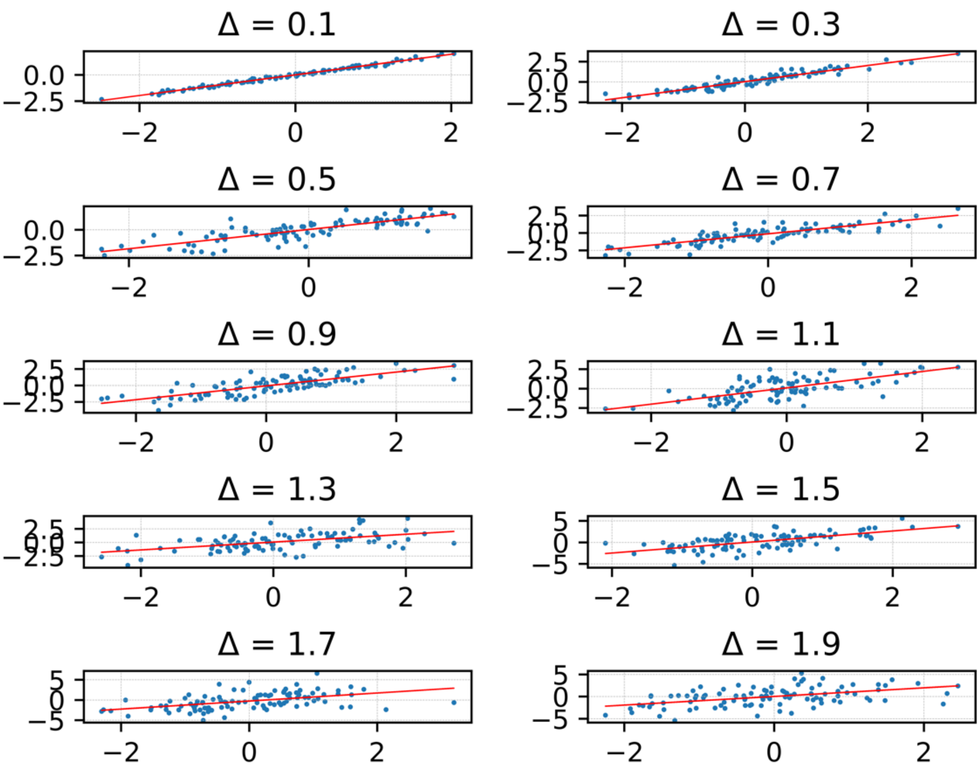
RESULT:

Concepts related simple linear regression were studied.

INPUT / OUTPUT:

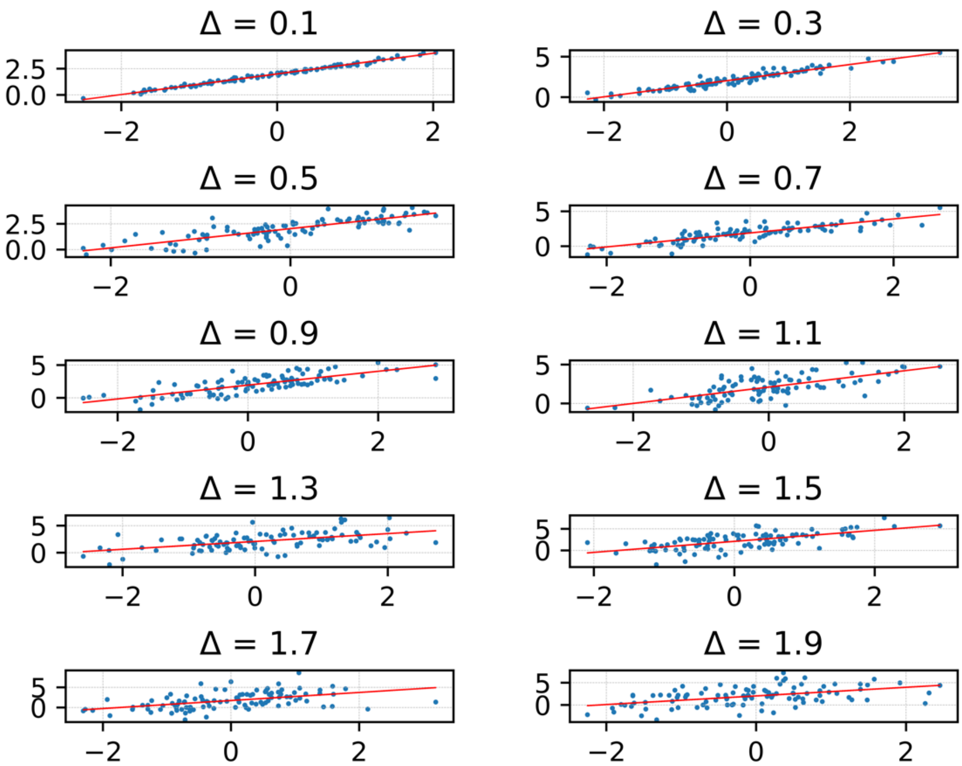
Program A:

```
In [7]: runfile('/Users/sheriffabdullah/College @ Shiv Nadar/Subjects/Semester 2/Foundations of Data Science/FDS Lab
Assignments/linearRegression1.py', wdir='/Users/sheriffabdullah/College @ Shiv Nadar/Subjects/Semester 2/Foundations
of Data Science/FDS Lab Assignments')
Delta       Slope                   Intercept                   SSE
0.1     0.9822431022755359      -0.008874278799616206       1.1025738346621319
0.3     0.9963353627195124      0.02587480604415024     10.793843088249993
0.5     0.939694732853961       0.027847913176580644        36.958230182202314
0.7     1.0032450167186568      -0.1197018179210592     42.66777522070362
0.9     1.053059188713496       -0.07443361079451724        82.92287659732465
1.1     1.052918145370536       0.08656418948022206     109.49120899224994
1.3     0.723157997796598       0.05958499976078034     181.13043281656738
1.5     1.282201935391324       0.07728838896084256     211.38661096339726
1.7     0.9950836861559095      -0.2809103631751024     324.4184602047594
1.9     0.9743245615812586      0.016036397511429448        338.11061254014515

In [8]: 
```



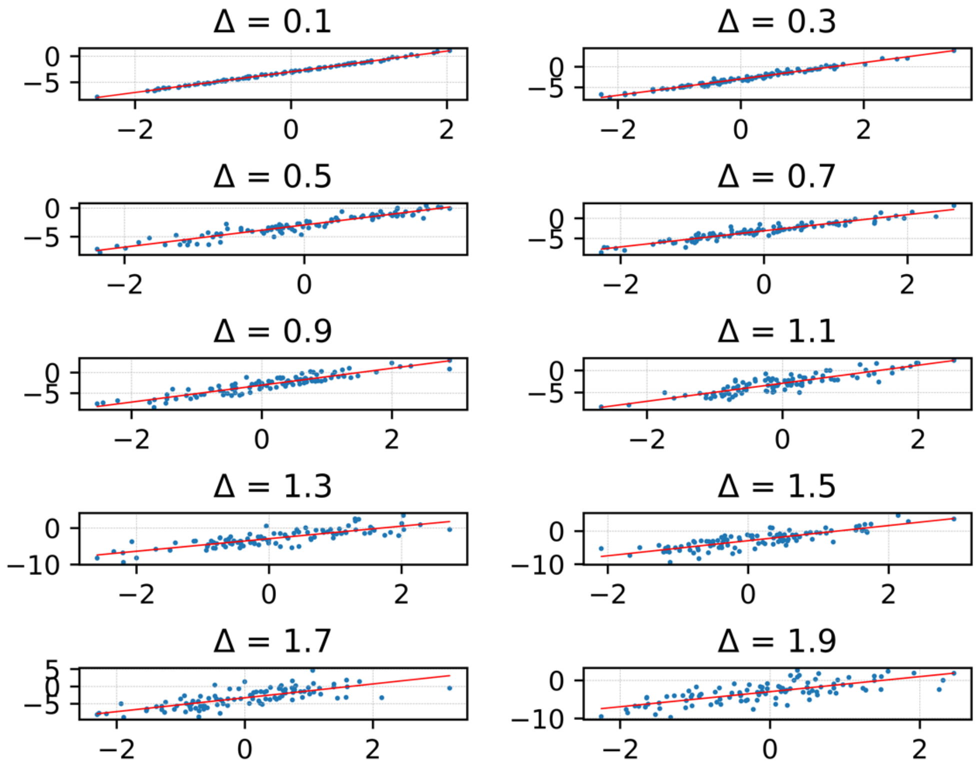
Program B:

```
In [1]: runfile('/Users/sheriffabdullah/College @ Shiv Nadar/Subjects/Semester 2/Foundations of Data Science/FDS Lab
Assignments/linearRegression3.py', wdir='/Users/sheriffabdullah/College @ Shiv Nadar/Subjects/Semester 2/Foundations
of Data Science/FDS Lab Assignments')
Delta       Slope                    Intercept                  SSE
0.1     1.9822431022755362      -3.0088742787996163     1.1025738346621323
0.3     1.996335362719512       -2.974125193955849      10.793843088249996
0.5     1.9396947328539602      -2.972152086823419      36.95823018220233
0.7     2.0032450167186564      -3.119701817921059      42.66777522070362
0.9     2.053059188713496       -3.074433610794517      82.92287659732465
1.1     2.0529181453705347      -2.913435810519778      109.49120899224994
1.3     1.7231579977965974      -2.940415000239219      181.13043281656738
1.5     2.2822019353913245      -2.9227116110391576     211.38661096339723
1.7     1.9950836861559096      -3.280910363175103      324.4184602047594
1.9     1.9743245615812586      -2.983963602488571      338.11061254014515
```



Program C:

```
In [1]: runfile('/Users/sheriffabdullah/College @ Shiv Nadar/Subjects/Semester 2/Foundations of Data Science/FDS Lab Assignments/linearRegression3.py', wdir='/Users/sheriffabdullah/College @ Shiv Nadar/Subjects/Semester 2/Foundations of Data Science/FDS Lab Assignments')
Delta       Slope                   Intercept                   SSE
0.1     1.9822431022755362      -3.0088742787996163     1.1025738346621323
0.3     1.996335362719512       -2.974125193955849      10.793843088249996
0.5     1.9396947328539602      -2.972152086823419      36.95823018220233
0.7     2.0032450167186564      -3.119701817921059      42.66777522070362
0.9     2.053059188713496       -3.074433610794517      82.92287659732465
1.1     2.0529181453705347      -2.913435810519778      109.49120899224994
1.3     1.7231579977965974      -2.940415000239219      181.13043281656738
1.5     2.2822019353913245      -2.9227116110391576     211.38661096339723
1.7     1.9950836861559096      -3.280910363175103      324.4184602047594
1.9     1.9743245615812586      -2.983963602488571      338.11061254014515
```



Ex. No: 3	CLASSIFICATION USING KNN
30 / 03 / 22	

AIM:

To learn classification using KNN algorithm in Python.

QUESTION:

Generate training data using the following parameters:

- ➔ Generate 100 2-dimensioanl data points (x, y) from a normal-distribution with $\mu_x = 2$, $\mu_y = 2$, $\sigma_x = 1$, and $\sigma_y = 1$. Let us consider these data points as class-1 training data.
- ➔ Generate 100 2-dimensioanl data points (x, y) from a normal-distribution with $\mu_x = 5$, $\mu_y = 5$, $\sigma_x = 1$, and $\sigma_y = 1$. Let us consider these data points as class-2 training data.

Generate testing data using the following parameters:

- ➔ Generate 20 2-dimensioanl data points (x, y) from a normal-distribution with $\mu_x = 2.5$, $\mu_y = 2.5$, $\sigma_x = 1$, and $\sigma_y = 1$. Let us consider these data points as class-1 testing data.
- ➔ Generate 20 2-dimensioanl data points (x, y) from a normal-distribution with $\mu_x = 4.5$, $\mu_y = 4.5$, $\sigma_x = 1$, and $\sigma_y = 1$. Let us consider these data points as class-2 testing data.

Save these training and testing data, separately.

For each of the testing data points, find out *K* nearest neighbour along with their class labels. Using voting method, make a decision on whether the test data point belongs to class-1 or class-2.

Calculate the classification accuracy.

Here, vary the value of *K* from 1 to 7 in steps of 1, and find out the optimal value for *K*.

ALGORITHM:

Step 1: Read string input ‘str1’ from user.
Step 2: Append the first character of ‘str1’ to ‘str2’.
Step 3: Replace all occurrences of first character of ‘str1’ with ‘$’.
Step 4: Append remaining characters of ‘str1’ to ‘str2’.
Step 5: Print output.

Program A:

```
"""
Exercise 2: K Nearest Neighbour
"""
import numpy as np
import matplotlib.pyplot as plt
'''
# Training Data - Class 1
x_train1 = np.random.normal(2, 1, 100)
y_train1 = np.random.normal(2, 1, 100)

# Training Data - Class 2
x_train2 = np.random.normal(5, 1, 100)
y_train2 = np.random.normal(5, 1, 100)

# Testing Data - Class 1
x_test1 = np.random.normal(2.5, 1, 20)
y_test1 = np.random.normal(2.5, 1, 20)

# Testing Data - Class 2
x_test2 = np.random.normal(4.5, 1, 20)
y_test2 = np.random.normal(4.5, 1, 20)

# Save training data in a '.npy' file
np.save("Saved Data/Training/x_train1.npy", x_train1)
np.save("Saved Data/Training/y_train1.npy", y_train1)
np.save("Saved Data/Training/x_train2.npy", x_train2)
np.save("Saved Data/Training/y_train2.npy", y_train2)

# Save testing data in a '.npy' file
np.save("Saved Data/Testing/x_test1.npy", x_test1)
np.save("Saved Data/Testing/y_test1.npy", y_test1)
np.save("Saved Data/Testing/x_test2.npy", x_test2)
np.save("Saved Data/Testing/y_test2.npy", y_test2)

# 2D Design Matrix for Training Data
train1_designMatrix = [[x_train1[i], y_train1[i]] for i in range(100)]
train2_designMatrix = [[x_train2[i], y_train2[i]] for i in range(100)]

# 2D Design Matrix for Testing Data
test1_designMatrix = [[x_test1[i], y_test1[i]] for i in range(20)]
test2_designMatrix = [[x_test2[i], y_test2[i]] for i in range(20)]

# Output
print("Training + testing data successfully generated.")
'''
```

Program B:

```
"""
Exercise 2: K Nearest Neighbour
Testing Data
"""

import math
from kNearestNeighbours_train import *

# Load training data from '.npy' file
x_train1 = np.load("Saved Data/Training/x_train1.npy")
y_train1 = np.load("Saved Data/Training/y_train1.npy")
x_train2 = np.load("Saved Data/Training/x_train2.npy")
y_train2 = np.load("Saved Data/Training/y_train2.npy")

# Load testing data from '.npy' file
x_test1 = np.load("Saved Data/Testing/x_test1.npy")
y_test1 = np.load("Saved Data/Testing/y_test1.npy")
x_test2 = np.load("Saved Data/Testing/x_test2.npy")
y_test2 = np.load("Saved Data/Testing/y_test2.npy")

print("Training + testing data successfully loaded.")
print()

# Plot Training Data
plt.scatter(x_train1, y_train1, color = 'black', s = 15)
plt.scatter(x_train2, y_train2, color = 'red', s = 15)

# Save scatter plot - Initial
plt.savefig("Graph/kNearestNeighbour_train.png", dpi = 1200)

# Number of test data points in 1 Class
numOfTestData = 20

# k Nearest Neighbours
for k in range(1, 8):

    # 3-Dimensional Distance Array
    distArr1 = []
    # Contains 20 arrays (1 array per test data point)
    # Every test data point array contains 200 '[distance, index]' arrays.

    trueClass1 = 0
    trueClass2 = 0

    # 20 'Class - 1' Test Data Points
    for pt in range(numOfTestData):
        # Append empty list for 1 test data point's data
        distArr1.append([])

        # Take 'pt'th data point from test data
```

```
        x_pt = x_test1[pt]
        y_pt = y_test1[pt]

        # Find distance of test point from each training point
        for i in range(100):
            dist = math.sqrt((x_pt - x_train1[i]) ** 2 + (y_pt - 
y_train1[i]) ** 2)
            distArr1[pt].append([dist, i])
        # 0 - 99 Index = Class 1

        for i in range(100):
            dist = math.sqrt((x_pt - x_train2[i]) ** 2 + (y_pt - 
y_train2[i]) ** 2)
            distArr1[pt].append([dist, i + 100])
        # 100 - 199 Index = Class 2

        # Sort the distances for 'pt'th test data point
        # Bubble Sort - Ascending Order
        for i in range(199):
            for j in range(199):
                if distArr1[pt][j][0] > distArr1[pt][j+1][0]:
                    # Swap values
                    temp = distArr1[pt][j]
                    distArr1[pt][j] = distArr1[pt][j+1]
                    distArr1[pt][j+1] = temp

        # Count number of points in class 1 & 2 within first 'k' points
        class1Count  = 0
        class2Count = 0

        for i in range(k):
            if distArr1[pt][i][1] >= 100:
                class2Count += 1
            else:
                class1Count += 1

        # Decision -  Class 1 or Class 2
        if class1Count > class2Count:
            trueClass1 += 1
            #print("Point belongs to Class 1")
        else:
            pass
            #print("Point belongs to Class 2")

        plt.scatter(x_pt, y_pt, color = "green", s = 7, marker = "<")

    distArr2 = []

    # 20 'Class - 2' Test Data Points
    for pt in range(numOfTestData):
        # Append empty list for 1 test data point's data
        distArr2.append([])
```

```
        # Take 'pt'th data point from test data
        x_pt = x_test2[pt]
        y_pt = y_test2[pt]

        # Find distance of test point from each training point
        for i in range(100):
            dist = math.sqrt((x_pt - x_train1[i]) ** 2 + (y_pt -
y_train1[i]) ** 2)
            distArr2[pt].append([dist, i])
        # 0 - 99 Index = Class 1

        for i in range(100):
            dist = math.sqrt((x_pt - x_train2[i]) ** 2 + (y_pt -
y_train2[i]) ** 2)
            distArr2[pt].append([dist, i + 100])
        # 100 - 199 Index = Class 2

        # Sort the distances for 'pt'th test data point
        # Bubble Sort - Ascending Order
        for i in range(199):
            for j in range(199):
                if distArr2[pt][j][0] > distArr2[pt][j+1][0]:
                    # Swap values
                    temp = distArr2[pt][j]
                    distArr2[pt][j] = distArr2[pt][j+1]
                    distArr2[pt][j+1] = temp

        # Count number of points in class 1 & 2 within first 'k' points
        class1Count  = 0
        class2Count = 0

        for i in range(k):
            if distArr2[pt][i][1] >= 100:
                class2Count += 1
            else:
                class1Count += 1

        # Decision -  Class 1 or Class 2
        if class1Count < class2Count:
            trueClass2 += 1
            #print("Point belongs to Class 1")
        else:
            pass
            #print("Point belongs to Class 2")

        plt.scatter(x_pt, y_pt, color = "blue", s = 7, marker = '>')

    # Accuracy
    accu = ((trueClass1 + trueClass2) / (numOfTestData * 2)) * 100
    print("k =", k, "Accuracy is", accu, "%")
```

```
# Output
print("\nOptimal value for 'k' is 5.")

# Save scatter plot - Final
plt.savefig("Graph/kNearestNeighbour_test.png", dpi = 1200)
```

RESULT:

The concepts related to KNN classifications were studied.

INPUT / OUTPUT:

Program A:

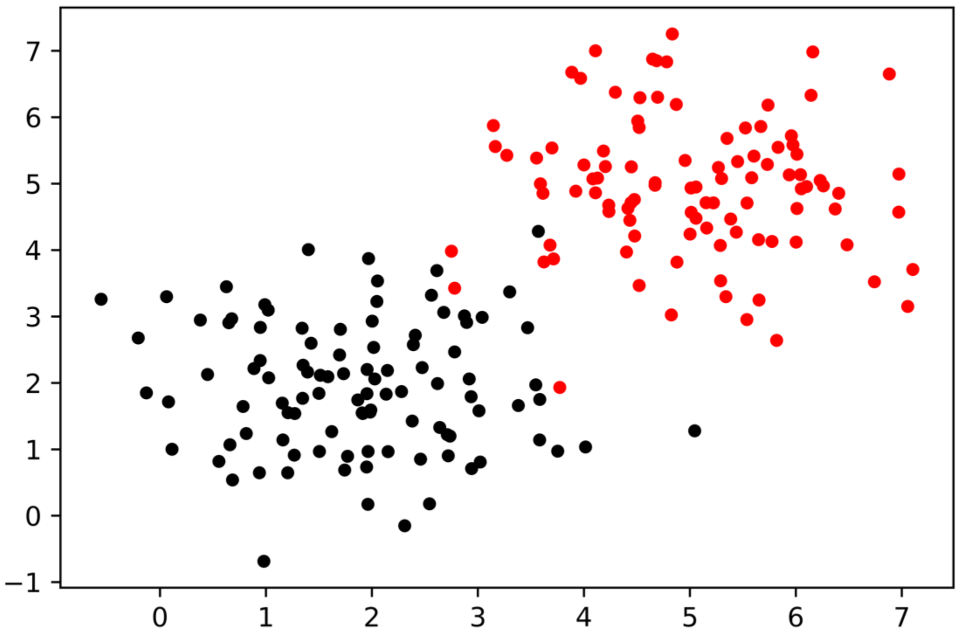
Training + testing data successfully generated.

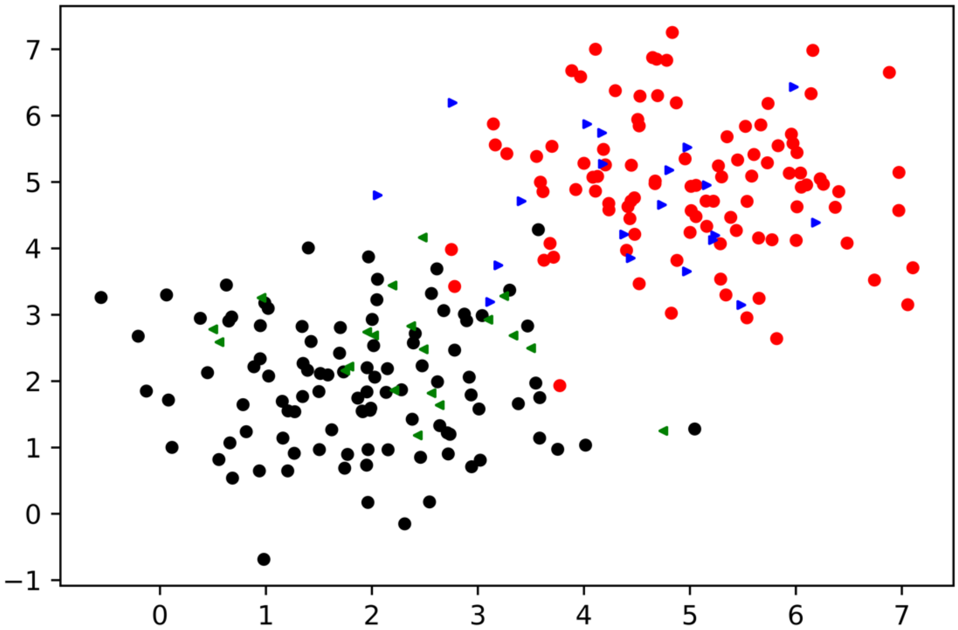
Program B:

Reloaded modules: kNearestNeighbours_train
Training + testing data successfully generated.
Training + testing data successfully loaded.

k = 1 Accuracy is 87.5%
k = 2 Accuracy is 85.0%
k = 3 Accuracy is 85.0%
k = 4 Accuracy is 85.0 %
k = 5 Accuracy is 90.0%
k = 6 Accuracy is 85.0 %
k = 7 Accuracy is 87.5%

Optimal value for 'k' is 5.




7
6
5
4
3
2
1
0
−1
0
1
2
3
4
5
6
7

Ex. No: 4	VECTORISATIONS WITH NUMPY LIBRARY
06 / 04 / 22	

AIM:

To learn vectorisations with 'NumPy' library in Python.

QUESTION:

Generate first 'N' Fibonacci series using closed-form solution and sum of even Fibonacci numbers within those first 'N' sequence. Solve using NumPy vectorization.

The Fibonacci series is a sequence of integers starting with zero, where each number is the sum of the previous two, except (of course) the first two numbers, zero and one (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 ...).

Usually Fibonacci sequence is estimated using a recurrence relation, but in this exercise you are going to solve it using a closed form solution based on the golden ratio.

The golden ratio 'φ' is defined as:

$$\varphi = \frac{1 + \sqrt{5}}{2}$$

The n^{th} Fibonacci number F_n is defined as:

$$F_n = \frac{\varphi^n - (-\varphi^{-n})}{\sqrt{5}}$$

Use the following Numpy methods to develop the solution:

- np.sqrt()
- np.arange()
- obj.astype()

Assume the value of 'N' as 50.

ALGORITHM:

Step 1: Estimate the value of 'φ'.
Step 2: Create a rank-1 array '***n***' using 'arange()' containing value from '0' to '49'.
Step 3: Generate Fibonacci sequence using the formula for F_n.

Step 4: The sequence generated will be '**float**' type. Convert it into '**int**' type using the astype() method.
Step 5: Print first 10 Fibonacci numbers.
Step 6: Estimate the sum of all even terms in the sequence generated above.
Step 7: The answer expected is: **6293134512.**

Program:

```
import numpy as np

# Number of Fibonacci numbers
N = 50
arr = np.arange(0, N, dtype="float")
goldenRatio=((1 + np.sqrt(5)) / 2)
print("Golden ratio:", goldenRatio)

for i in range(0, N):
    arr[i] = (((goldenRatio ** arr[i]) - (-goldenRatio ** (-arr[i]))) /
np.sqrt(5))

# Convert all numbers from 'float' to 'int' type
arr = np.ndarray.astype(arr, int)
print("\n'N' Fibonacci numbers: \n", arr)

# Sum of even numbers within first 'N' Fibonacci numbers.
sum = 0

for i in range(0, 50):
    if int(arr[i]) % 2==0:
        sum += int(arr[i])

print("\nSum of even Fibonacci numbers within 'N':", sum)

'''
Assume a rank-1 array
    x = np.arange(10)
    print(x)   # [0 1 2 3 4 5 6 7 8 9]

We can create a new array 'x_even' which comprises only the even numbers of
'x' as
    x_even = x[x%2 == 0]
    # Finds those entries in x which are divisible by 2.
    print(x_even)   # [0 2 4 6 8]
'''
```

RESULT:

Concepts related to vectorisations with NumPy library were studied.

INPUT / OUTPUT:

Golden ratio: 1.618033988749895

'N' Fibonacci numbers:

```
[         0          1          1          2          3          5
          8         13         21         34         55         89
        144        233        377        610        987       1597
       2584       4181       6765      10946      17711      28657
      46368      75025     121393     196418     317811     514229
     832040    1346269    2178309    3524578    5702887    9227465
   14930352   24157817   39088169   63245986  102334155  165580141
  267914296  433494437  701408733 1134903170 1836311903 2971215073
 4807526976 7778742049]
```

Sum of even Fibonacci numbers within 'N': 6293134512

Ex. No: 5	**DATA PREPROCESSING USING PANDAS**
06 / 04 / 22	

AIM:

To learn about data pre-processing using 'Pandas' library in Python.

QUESTION:

Refer to the CSV file ("*AdvertisingV2.csv*") provided along with this exercise.

The head() shows there are some missing values in the CSV file for which 'NaN' is displayed.

Find out which all attributes contain missing values?
Hint: Infer this detail by looking at the output of 'info()' method. Refer to 'fillna()' method for Pandas and apply the following operation:

1. a) Replace the **NaN** with **zero** for the column
2. b) Replace the **NaN** with **mean** of the corresponding column
3. c) Replace the **NaN** with **median** of the corresponding column

For each of the modification applied above, look at the descriptive statistics using describe() method, take a screenshot of the output and include them in the submission.

Write down how many data were missing corresponding to each column, what was the **mean** and **median** value which were replaced with **NaN**.

ALGORITHM:

Step 1: Import 'NumPy' and 'Pandas' library.
Step 2: Create dataframe.
Step 3: Replace missing values with '0' & print statistics.
Step 4: Replace missing values with mean & print statistics.
Step 5: Replace missing values with median & print statistics.

Program:

```
import numpy as np
import pandas as pd

data = pd.read_csv("AdvertisingV2.csv")
x = pd.DataFrame(data)
y = pd.DataFrame(data)
z = pd.DataFrame(data)
```

```
x.head()
print("Total number of datapoints: ", z.shape[0])
print("Number of null values:", x["TV"].isna().sum())

# Replace missing values with '0'
x = x.fillna(0)
print("\nMissing values filled with '0'.")
print("\n* Descriptive statistics *")

# Descriptive statistics
print(x.describe())

# Replace missing values with mean
print("\n** Mean **")
mean1 = y["TV"].mean()
print("TV:", mean1)
y["TV"] = y["TV"].replace(0, mean1)

mean2 = y["radio"].mean()
print("Radio:", mean2)
y["radio"] = y["radio"].replace(0, mean2)

mean3 = y["newspaper"].mean()
print("Newspaper:", mean1)
y["newspaper"] = y["newspaper"].replace(0, mean3)

mean4 = y["sales"].mean()
print("Sales:", mean1)
y["sales"] = y["sales"].replace(0, mean4)

print("Missing values filled with mean.")
print("\n* Descriptive statistics *")

# Descriptive statistics
print(y.describe())

# Replace missing values with median
print("\n** Median **")
median1 = z["TV"].median()
print("TV:", median1)
z["TV"] = z["TV"].replace(0, median1)

median2 = z["radio"].median()
print("Radio:", median2)
z["radio"] = z["radio"].replace(0, median2)

median3 = z["newspaper"].median()
print("Newspaper:", median1)
z["newspaper"] = z["newspaper"].replace(0, median3)

median4 = z["sales"].median()
```

```
print("Sales:", median1)
z["sales"] = z["sales"].replace(0, median4)

print("Missing values filled with median.")
print("\n* Descriptive statistics *")

# Descriptive statistics
print(z.describe())
```

RESULT:

The concepts related to data pre-processing using ‘Pandas’ were studied.

INPUT / OUTPUT:

Total number of datapoints: 200
Number of null values: 5

Missing values filled with '0'.

* Descriptive statistics *

	Unnamed: 0	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.00000	200.000000
mean	100.500000	143.349000	23.264000	29.91500	14.022500
std	57.879185	88.465442	14.846809	21.72123	5.217457
min	1.000000	0.000000	0.000000	0.00000	1.600000
25%	50.750000	68.850000	9.975000	12.27500	10.375000
50%	100.500000	140.800000	22.900000	25.60000	12.900000
75%	150.250000	218.825000	36.525000	43.55000	17.400000
max	200.000000	296.400000	49.600000	114.00000	27.000000

** Mean **
TV: 147.02461538461537
Radio: 23.264000000000024
Newspaper: 147.02461538461537
Sales: 147.02461538461537
Missing values filled with mean.

* Descriptive statistics *

	Unnamed: 0	TV	radio	newspaper	sales
count	200.000000	195.000000	200.000000	198.000000	200.000000
mean	100.500000	147.024615	23.380320	30.217172	14.022500
std	57.879185	86.513857	14.754473	21.620006	5.217457
min	1.000000	0.700000	0.300000	0.300000	1.600000
25%	50.750000	74.050000	10.075000	12.650000	10.375000
50%	100.500000	149.700000	23.282000	25.600000	12.900000
75%	150.250000	220.050000	36.525000	44.050000	17.400000
max	200.000000	296.400000	49.600000	114.000000	27.000000

** Median **
TV: 149.7
Radio: 23.28200000000001
Newspaper: 149.7
Sales: 149.7
Missing values filled with median.

* Descriptive statistics *

	Unnamed: 0	TV	radio	newspaper	sales
count	200.000000	195.000000	200.000000	198.000000	200.000000
mean	100.500000	147.024615	23.380320	30.217172	14.022500
std	57.879185	86.513857	14.754473	21.620006	5.217457
min	1.000000	0.700000	0.300000	0.300000	1.600000
25%	50.750000	74.050000	10.075000	12.650000	10.375000
50%	100.500000	149.700000	23.282000	25.600000	12.900000
75%	150.250000	220.050000	36.525000	44.050000	17.400000
max	200.000000	296.400000	49.600000	114.000000	27.000000

Ex. No: 6	LINEAR REGRESSION USING SCIKIT-LEARN
27 / 04 / 22	

AIM:

To learn linear regression using 'scikit-learn' library in Python.

QUESTION 1:

Implement Linear Regression model using Scikit-Learn. Predict the expected sales when spending the money for advertisement in the following medias:
- TV
- Radio
- Newspaper

Download the dataset from the following URL:
- https://www.visioncog.com/data/data.zip
(Use `Advertising.csv`)

ALGORITHM:

Step 1: Read 'Advertising.csv' file and create a dataframe.
Step 2: Create independent variable 'x' and store attributes.
Step 3: Create dependent variable 'y' and store attributes.
Step 4: Visualise the input attributes in a 3D plot.
Step 5: Spilt data into training and testing data.
Step 6: Visualise the training and testing in a 3D plot.
Step 7: Use the `LinearRegression` class from Scikit-Learn and create an object to this class.
Step 8: Call the `fit()` function by passing training data as its argument.
Step 9: Calculate and print R^2 score for the regression model.
Step 10: Print regression model parameters and independent variable 'x'.
Step 11: Print actual values predicted values of 'y'.

Program:

```
"""
Implement Linear Regression model using Scikit-Learn
Predict the expected sales when spending the money for
advertisement in the following medias:

- TV
- Radio
- Newspaper

Download the dataset from the following URL:
```

```
- https://www.visioncog.com/data/data.zip
- Use `Advertising.csv`
"""

# Get the data from VisionCog website
#! wget https://www.visioncog.com/data/data.zip
#! unzip data.zip

# Import all relevant scientific libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Loading the CSV file using Pandas library
df_Adv = pd.read_csv('data/Advertising.csv')

# Prints the first five entries by default.
print(df_Adv.head())

# Store the input attributes under the variable `X` and output under `Y`
X = df_Adv[['TV', 'radio', 'newspaper']]
X.head()

Y = df_Adv['sales']
Y.head()

# Visualize the input attributes in 3D plot
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 6))
ax = fig.gca(projection='3d')

ax.scatter(X.iloc[:,0], X.iloc[:,1], X.iloc[:,2], color="b");
# <data_frame>.iloc[<index_pos/row>, <title/col>]


ax.set_xlabel("TV")
ax.set_ylabel("Radio")
ax.set_zlabel("Newspaper")

plt.tight_layout()
plt.show()

"""
Split the data into training and testing.
- 80% for training
- 20% for testing
"""

from sklearn.model_selection import train_test_split
```

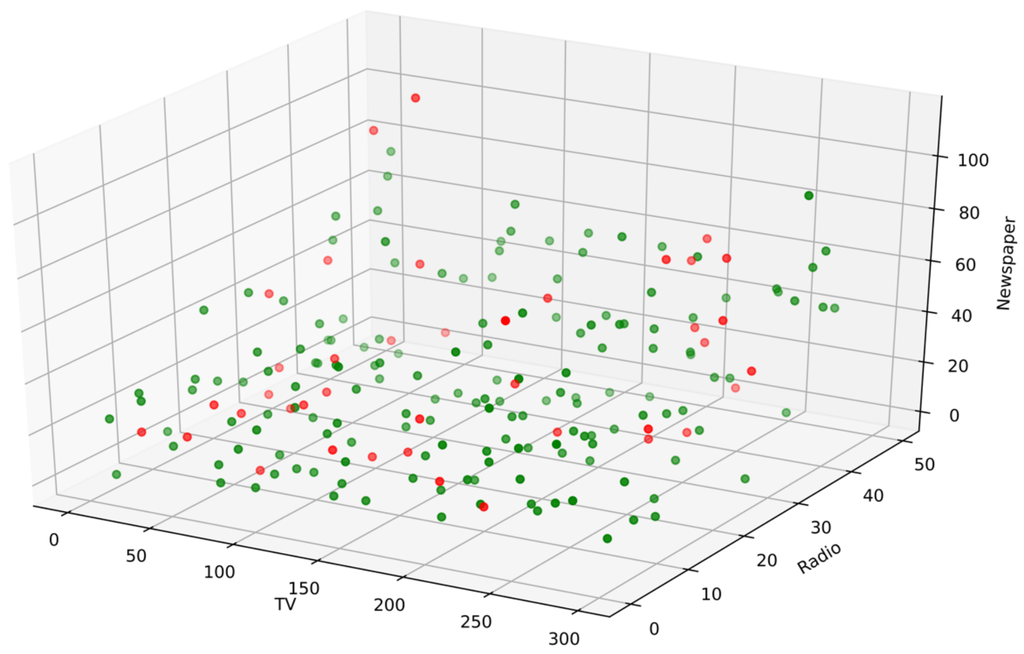
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)


print("\n'X' training dataset shape:", X_train.shape)
print("'X' testing dataset shape:", X_test.shape)

print("'Y' training dataset shape:", Y_train.shape)
print("'Y' testing dataset shape:", Y_test.shape)


"""
Visualize training and testing data
- Green --> training data
- Red   --> testing data
"""

fig = plt.figure(figsize=(10, 6))
ax = fig.gca(projection='3d')

# Plot traning data in green
ax.scatter(X_train.iloc[:,0], X_train.iloc[:,1], X_train.iloc[:,2],
color="g");

# Plot testing data in red
ax.scatter(X_test.iloc[:,0], X_test.iloc[:,1], X_test.iloc[:,2],
color="r");

ax.set_xlabel("TV")
ax.set_ylabel("Radio")
ax.set_zlabel("Newspaper")

plt.tight_layout()
plt.show()

"""
Use the `LinearRegression` class from Scikit-Learn
- Create an object to this class.
- Call the `fit()` function by passing training data as its argument.
"""

from sklearn.linear_model import LinearRegression

linearRegression = LinearRegression()
linearRegression.fit(X_train, Y_train)

"""
To evaluate the mode, use the R^2 test defined as follows:

R^2 = 1 - (SS[reg] / SS[tot])
```

```
- SS[reg] --> (Sum Squared Regression Error) the error of linear regresion
model trained on the training dataset.

- SS[tot] --> (Sum Squared Total Error) the error of a simple average
prediction model
"""

r2_Score = linearRegression.score(X_test, Y_test)

print("\nR^2 value:", r2_Score)
print("Adding 'X' (thereby, making it linear regression prediction) reduced
the error in prediction (by average of 'y' method) by 89.94%.")

"""
The model parameters can be accessed as member variables:
    y' =  c0 + c1 * x1 + c2 * x2 + ... + cn * xn
- obj.intercept_  --> co
- obj.coef_       --> c1, c2 , ... , cn
"""

print("\nIntercept:", linearRegression.intercept_)
print("Coefficients (c1, c2 & c3):", linearRegression.coef_)

print("\n** 'X' Test Dataset Values **\n", X_test[:10])

# Predict the sales for money spent on different advertising media

print("\n** 'Y' Prediction Values **")
for i in linearRegression.predict(X_test[:10]):
    print(i)

# Actual sales values"
print("\n** 'Y' Actual Values **")
for i in Y_test[:10]:
    print(i)
```

RESULT:

The concepts related to linear regression using ‘scikit-learn’ were studied.

PLOT:



TV
Radio
Newspaper
0
50
100
150
200
250
300
0
10
20
30
40
50
0
20
40
60
80
100

INPUT / OUTPUT:

	Unnamed: 0	TV	radio	newspaper	sales
0	1.	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3.	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

'X' training dataset shape: (160, 3)
'X' testing dataset shape: (40, 3)
'Y' training dataset shape: (160,)
'Y' testing dataset shape: (40,)

R^2 value: 0.899438024100912
Adding 'X' (thereby, making it linear regression prediction) reduced the error in prediction (by average of 'y' method) by 89.94%.

Intercept: 2.9790673381226274
Coefficients (c1, c2 & c3): [0.04472952 0.18919505 0.00276111]

** 'X' Test Dataset Values **

	TV	radio	newspaper
95	163.3	31.6	52.9
15	195.4	47.7	52.9
30	292.9	28.3	43.2
158	11.7	36.9	45.2
128	220.3	49.0	3.2
115	75.1	35.0	52.7
69	216.8	43.9	27.2
170	50.0	11.6	18.4
174	222.4	3.4	13.1
45	175.1	22.5	31.5

** 'Y' Prediction Values **
16.408024203228628
20.88988208714789
21.55384317908956
10.608502561984903
22.11237325985767
13.105591724016458
21.05719191631465
7.461010344558368
13.60634580543393
15.155069668921398

** 'Y' Actual Values **
16.9
22.4
21.4
7.3
24.7
12.6
22.3
8.4
11.5
14.9

Ex. No: 7	LOGISTIC REGRESSION AND KNN CLASSIFICATION USING SCIKIT-LEARN
04 / 05 / 22	

AIM:

To learn logistic regression and KNN classification using ‘scikit-learn’ library in Python.

QUESTION:

Multi-class classification using Scikit-Learn:
- KNN Classifier.
- Logistic Regression classifier.

The data must be synthetically created using `make_blob` function from Scikit-Learn.

ALGORITHM:

Step 1: Synthetically create data – datapoints with classes.
Step 2: Visualise the data.
Step 3: Split data into training and testing.
Step 4: Use KNN Classifier to classify datapoints using ‘x’ & ‘y’ values of the datapoints.
Step 5: Calculate classification accuracy.
Step 6: Use logistic regression to classify datapoints using ‘x’ & ‘y’ values of the datapoints.
Step 7: Calculate classification accuracy.

Program 1:

```
"""
Multi-class classification using Scikit-Learn
- KNN Classifier
- Logistic Regression classifier

The data is synthetically created using `make_blob` function from Scikit-
Learn.
"""

from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

"""## Synthetically creating data"""
centers = [[5,3], [5,5], [3,8]]
n_classes = len(centers)

data, labels = make_blobs(n_samples = 150,
```

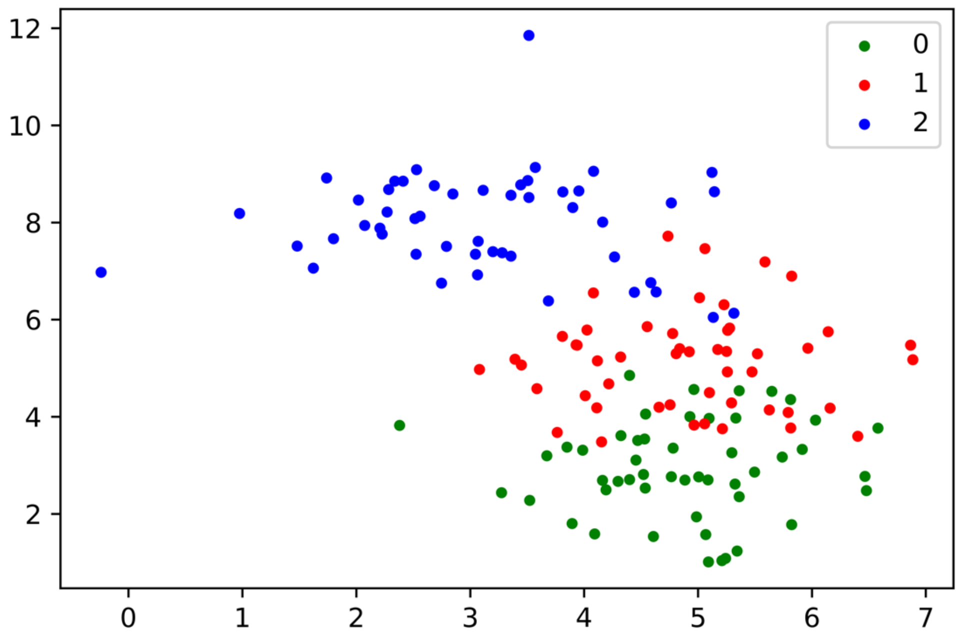
```
                          centers = np.array(centers),
                          random_state = 42)

"""## Visualize the data"""
colors = ('green', 'red', 'blue')

for nc in range(0, n_classes):
    plt.scatter(data[labels==nc, 0],
                data[labels==nc, 1],
                c=colors[nc], s=10,
                label=str(nc))

plt.legend()
plt.show()

print("** Sample Datapoints **")
print("  x - Value | y - Value")
print(data[:5, :5])
print("\nClass labels:")

for label in labels[:5]:
    print(label)

"""## Split the data into training and testing:
- 80% training
- 20% testing
"""

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels,
                                                    test_size = 0.2,
                                                    random_state=42)

print("\n'X' training data points:", len(X_train), "\n'X' testing data
points:", len(X_test))
print("'y' training data points:", len(y_train), "\n'y' testing data
points:", len(y_test))


"""## Use KNN classifier """

from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier()

knn_clf.fit(X_train, y_train)

predicted = knn_clf.predict(X_test)

print("\n\n** KNN Classifier **")
print("\nPredicted classes:", predicted[:10])
print("Actual classes   :", y_test[:10])
```

```
"""## Evaluate the model based on classification accuracy"""
score = knn_clf.score(X_test, y_test)
print("Classification accuracy: ", score)


"""## Use `LogisticRegression` model to perform classification.
Note:
    It is essential to include `multi_class='multinomial'` as argument
    to adapt it for multi-class classification. By default,
`LogisticRegression`
    performs only binary classification.
"""

from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(multi_class='multinomial')

lr_clf.fit(X_train, y_train)

predicted = lr_clf.predict(X_test)

print("\n** Logistic Regression Classification **")
print("\nPredicted classes:", predicted[:10])
print("Actual classes   :", y_test[:10])

score_lr_clf = lr_clf.score(X_test, y_test)
print("Classification accuracy: ", score_lr_clf)
```

RESULT:

The concepts related to logistic regression and KNN classification using ‘scikit-learn’ were studied.

PLOT:



INPUT / OUTPUT:

```
** Sample Datapoints **
   x - Value     |   y - Value
[[4.92798988   4.0035329 ]
 [4.53936123.  4.05712223]
 [3.58462926.  4.57935468]
 [4.00946367.  4.43370227]
 [5.01300189.  6.45353408]]

Class labels:
0
0
1
1
1

'X' training data points: 120
'X' testing data points: 30
'y' training data points: 120
'y' testing data points: 30


** KNN Classifier **

Predicted classes: [2 2 2 0 1 0 2 0 0 0]
Actual classes     : [1 2 2 0 0 1 2 0 0 0]
Classification accuracy:  0.8666666666666667

** Logistic Regression Classification **

Predicted classes: [2 2 2 0 0 1 2 0 0 0]
Actual classes     : [1 2 2 0 0 1 2 0 0 0]
Classification accuracy:  0.96666x66666666667
```