Data Science Lab

(LAB MANUAL)

B.Tech. II Year I Semester

Department of Artificial Intelligence & Data Science

Chennai Institute of Technology

An AICTE Recognized- Accredited by NBA, NAAC - Affiliated to Anna University

LIST OF PROGRAMS

1. Write a python program for Basic Calculator.
2. Write a python program in Basic Statistics functions.
3. Write a python program for Data Visualizations.
4. Write a python program Application Program Linear Regression.
5. Write a python program Application Program to perform Classification using Logistic Regression.
6. Write a python program Application Program for plot a normal distribution with Matplotlib.
7. Write a python program Application Program for KNN Classifier with IRIS Dataset.
8. Write a python Application Program to demonstrate the Principal Component Analysis using Breast Cancer dataset.
9. Write a python Application Program to demonstrate the Singular Value Decomposition to find the document and string relativity.
10. Write a python Application Program to demonstrate the K-means Clustering.
11. Write a python Application Program to demonstrate the correlation with two set of values and display whether it is highly correlated or low correlated.
12. Write a python Application Program to demonstrate the Analysis of covariance (ANOVA).
13. Write a python Application Program to demonstrate the Multiple Regression.

1. BASIC CALCULATOR

1. **Problem Statement:** Write a python program to create a simple calculator which can perform basic arithmetic operations like addition, subtraction, multiplication or division depending upon the user input.

2. **Expected Learning Outcomes:** To train the students to understand the basics structure of python program and basic mathematical operations.

3. **Problem Analysis:** Simple calculation lime addition, subtraction, multiplication or division can be performed.

4. **A. Input:** Get the option for performing operation, get the numbers for doing basic operations. For example it takes the input as follows

   ```
   Please select operation -
   1. Add
   2. Subtract
   3. Multiply
   4. Divide
   Select operations form 1, 2, 3, 4: 1
   Enter first number: 20
   Enter second number: 13
   ```

   B. Output: It will print the basic operations and output. It do perform the operations as follows for the above operation selection.

   ```
   20 + 13 = 33
   ```

5. **Algorithm:**

   ```
   Step1:  Get the Option for Operation Selection.
   Step 2: Get the First Number.
   Step 3: Get the Second Number.
   Step 4: Perform the selected Operation.
   ```

6. **Coding:**

```
# Python program for simple calculator

# Function to add two numbers
def add(num1, num2):
    return num1 + num2

# Function to subtract two numbers
def subtract(num1, num2):
```

```
    return num1 - num2

# Function to multiply two numbers
def multiply(num1, num2):
    return num1 * num2

# Function to divide two numbers
def divide(num1, num2):
    return num1 / num2

print("Please select operation -\n" \
        "1. Add\n" \
        "2. Subtract\n" \
        "3. Multiply\n" \
        "4. Divide\n")


# Take input from the user
select = int(input("Select operations form 1, 2, 3, 4 :"))

number_1 = int(input("Enter first number: "))
number_2 = int(input("Enter second number: "))

if select == 1:
    print(number_1, "+", number_2, "=",
                    add(number_1, number_2))

elif select == 2:
    print(number_1, "-", number_2, "=",
                    subtract(number_1, number_2))

elif select == 3:
    print(number_1, "*", number_2, "=",
                    multiply(number_1, number_2))

elif select == 4:
    print(number_1, "/", number_2, "=",
                    divide(number_1, number_2))
else:
    print("Invalid input")
```

7. Test case :

a.	Expected Result:	Please select operation - 1. Add 2. Subtract 3. Multiply 4. Divide Select operations form 1, 2, 3, 4 : 3 Enter first number: 20 Enter second number: 13 20 * 13 = 260
b.	Actual Result:	

8. **Viva**:

2. BASIC STATISTICAL FUNCITONS

1. **Problem Statement:** Write a python program in Basic Statistics functions.

2. **Expected Learning Outcomes:** To train the students to understand the basics functions of statistics.

3. **Problem Analysis:** Compute the Mean, Median, Mode, Quartiles and Standard deviation using formulas.

4. **A. Input:** Take a set of eight numbers as input.

B. Output:

```
Set of frequencies: [11, 21, 34, 22, 27, 11, 23, 21]
Mean= 21.25
Mean= 21.25
Median 21.5
Mode [11, 21]
Mode 11
Quartiles (16.0, 21.5, 25.0)
Standard Deviation Simple Method: 7.1545440106270926
Standard Deviation using Numpy: 7.1545440106270926
```

5. **Algorithm:**

```
Step1:  Take a list of 8 Numbers.
Step2:  Compute the Mean value by simple Computation and print it.
Step3:  Compute the Mean value using numpy method and print it.
Step4:  Compute the Median value by simple Computation and print it.
Step5:  Compute the Mode value by simple Computation and print it.
Step6:  Compute the Mode value using numpy method and print it.
Step7:  Compute the IQR (Inter Quartile Range) by simple Computation and print it.
Step8:  Compute the Standard Deviation by simple Computation and print it.
Step9:  Compute the Standard Deviation using Numpy and print it.
```

6. **Coding:**

```
# Write a program to compute mean, median, mode, IQR and Standard Deviation
import numpy as n
from collections import Counter
from scipy import stats
# Finding Mean by simple Computation
a= [11, 21, 34, 22, 27, 11, 23, 21]
mean = sum(a)/len(a)
print (mean)
# Finding Mean using numpy method
```

```
mean = np.mean(a)
print (mean)
# Finding Median by simple Computation.
def median(nums):
   nums.sort()
   if len(nums)%2 == 0:
      return int(nums[len(nums)//2-1]+nums[len(nums)//2])/2
   else:
      return nums[len(nums)//2]
print (median(a))
# Finding Mode by simple Computation
data = dict(Counter(a))
mode = [k for k, v in data.items() if v == max(list(data.values()))]
print (mode)
# Finding Mode using numpy method
print (stats.mode(a)[0][0])
# Finding Quartiles by simple method
def quartiles(nums):
   nums=sorted(nums)
   Q1 = median(nums[:len(nums)//2])
   Q2 = median(nums)
   if len(nums)%2 == 0:
        Q3 = median(nums[len(nums)//2:])
   else:
        Q3 = median(nums[len(nums)//2+1:])
   return Q1,Q2,Q3
def median(nums):
   nums.sort()
   if len(nums)%2 == 0:
        return int(nums[len(nums)//2-1]+nums[len(nums)//2])/2
   else:
      return nums[len(nums)//2]
      print (quartiles(a))

# Find Standard deviation by simple computation
n=len(a)
std=(sum(map(lambda x: (x-sum(a)/n)**2,a))/n )**0.5
print(std)
# Find Standard deviation using numpy method
print (np.std(a))
```

Test case:

a.	Expected Result:	Set of frequencies: [11, 21, 34, 22, 27, 11, 23, 21] Mean= 21.25 Mean= 21.25 Median 21.5 Mode [11, 21] Mode 11

		Quartiles (16.0, 21.5, 25.0) Standard Deviation Simple Method: 7.1545440106270926 Standard Deviation using Numpy: 7.1545440106270926
b.	Actual Result:	

7. Viva:

3. DATA VISUALIZATIONS

1. **Problem Statement:** Write a python program for Data Visualizations.

2. **Expected Learning Outcomes:** To train the students to understand the different data visualization methods.

3. **Problem Analysis:** Take two different datasets and plotting different charts and graphs.

   A. Input: IRIS Dataset and Wine Review Data set taken as the Input.
   B. Output: Different graph plotting.

4. **Algorithm:**

```
Step1:  Load the IRIS Dataset and Wine Review Dataset
Step 2: Create the Color Scatter Plot of IRIS Dataset.
Step 3: Create the Line chart for each attributes of IRIS Dataset.
Step 4: Create the Histogram for Wine Review Scores.
Step 5: Create the Bar Chart for Wine Review Scores.
Step 6: Create the multiple histogram for attributes of IRIS Dataset.
Step 7: Create the vertical bar chart for Wine Review Scores using plot.bar().
Step 8: Create the horizontal bar chart for Wine Review Scores using plot.bar().
Step 9: Create the bar chart for Wine Review with highest cost five different
Counties.
```

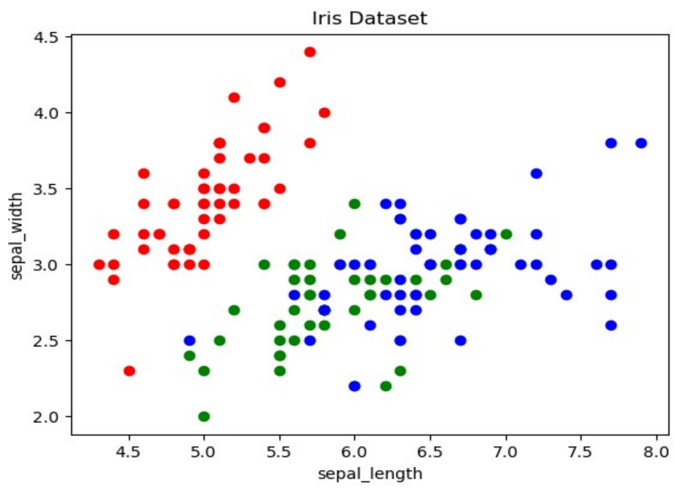
5. **Coding:**

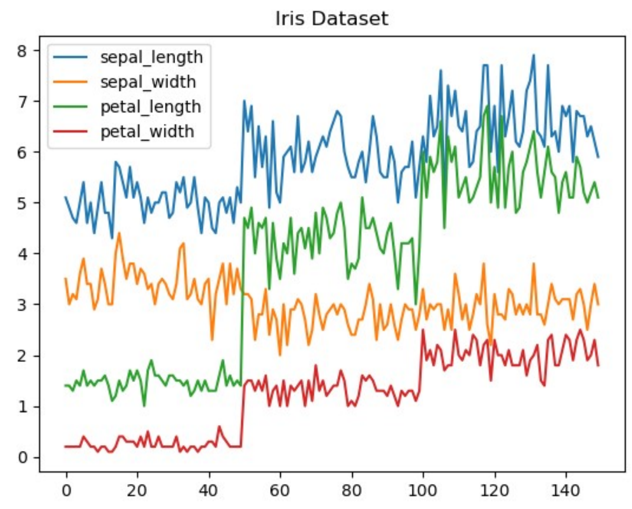
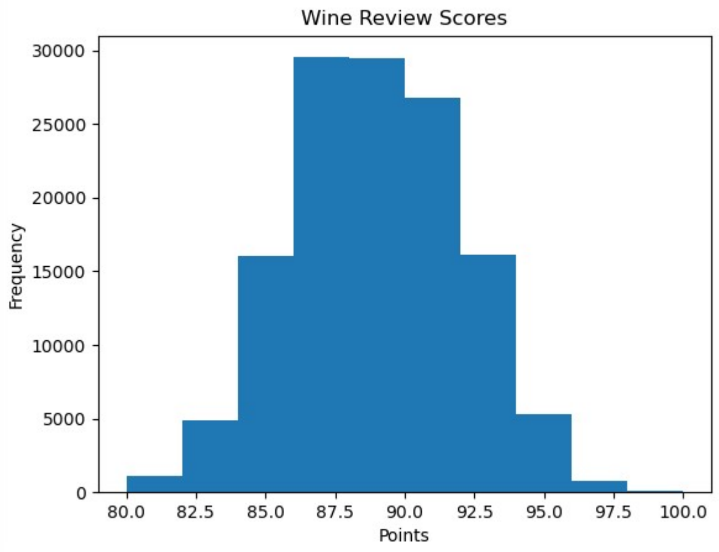
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
iris = pd.read_csv('iris.csv', names=['sepal_length', 'sepal_width', 'petal_length',
'petal_width', 'class'])
print(iris.head())
wine_reviews = pd.read_csv('winemag-data-130k-v2.csv', index_col=0)
wine_reviews.head()

# Create Color Scatter Plotting
colors = {'Iris-setosa':'r', 'Iris-versicolor':'g', 'Iris-virginica':'b'}
# create a figure and axis
fig, ax = plt.subplots()
# plot each data-point
for i in range(len(iris['sepal_length'])):
    ax.scatter(iris['sepal_length'][i], iris['sepal_width'][i],color=colors[iris['class'][i]])
# set a title and labels
ax.set_title('Iris Dataset')
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
```

```
plt.show()

# Create Line Chart Plotting
columns = iris.columns.drop(['class'])
# create x data
x_data = range(0, iris.shape[0])
# create figure and axis
fig, ax = plt.subplots()
# plot each column
for column in columns:
    ax.plot(x_data, iris[column], label=column)
# set title and legend
ax.set_title('Iris Dataset')
ax.legend()
plt.show()

# create figure and axis
fig, ax = plt.subplots()
# plot histogram
ax.hist(wine_reviews['points'])
# set title and labels
ax.set_title('Wine Review Scores')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
plt.show()

# create a figure and axis
fig, ax = plt.subplots()
# count the occurrence of each class
data = wine_reviews['points'].value_counts()
# get x and y data
points = data.index
frequency = data.values
# create bar chart
ax.bar(points, frequency)
# set title and labels
ax.set_title('Wine Review Scores')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
plt.show()



iris.plot.hist(subplots=True, layout=(2,2), figsize=(10, 10), bins=20)
plt.show()

wine_reviews['points'].value_counts().sort_index().plot.bar()
plt.show()
```

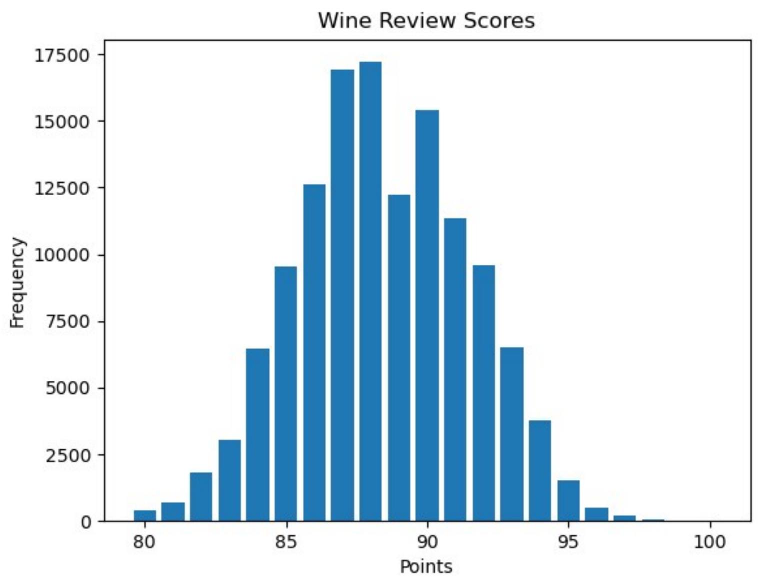
```
wine_reviews['points'].value_counts().sort_index().plot.barh()
plt.show()

wine_reviews.groupby("country").price.mean().sort_values(ascending=False)[:5].plo
t.bar()
plt.show()

# Correlation Matrix
corr = iris.corr()
fig, ax = plt.subplots()
# create heatmap
im = ax.imshow(corr.values)

# set labels
ax.set_xticks(np.arange(len(corr.columns)))
ax.set_yticks(np.arange(len(corr.columns)))
ax.set_xticklabels(corr.columns)
ax.set_yticklabels(corr.columns)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
        rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(len(corr.columns)):
    for j in range(len(corr.columns)):
        text = ax.text(j, i, np.around(corr.iloc[i, j], decimals=2),
                       ha="center", va="center", color="black")
plt.show()
```

6. Test case :

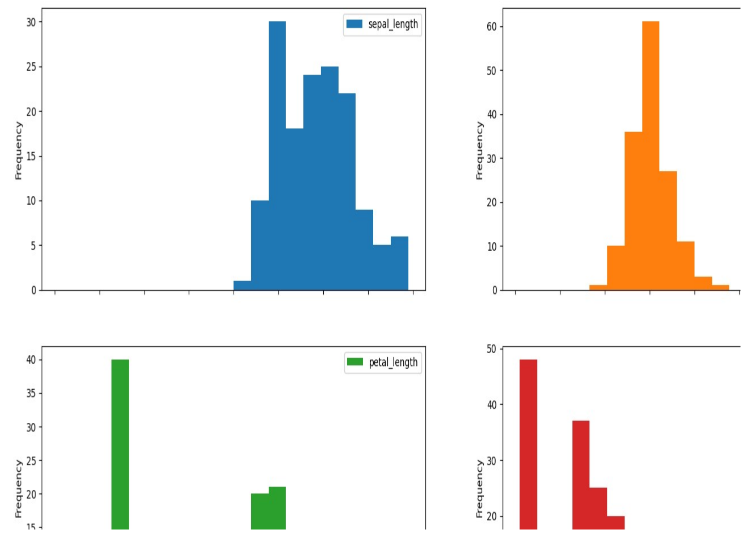
a.	Expected Result:	Scatter Plot of IRIS Dataset  Line chart for each attributes of IRIS Dataset

		 Histogram for Wine Review Scores. 

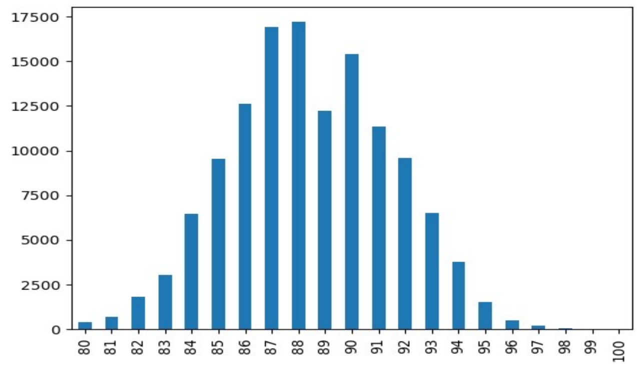
Bar Chart for Wine Review Scores

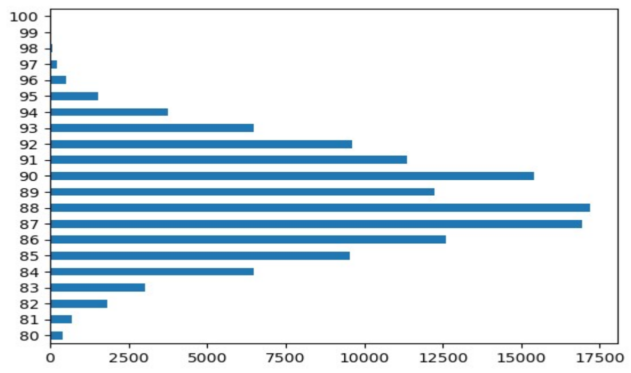
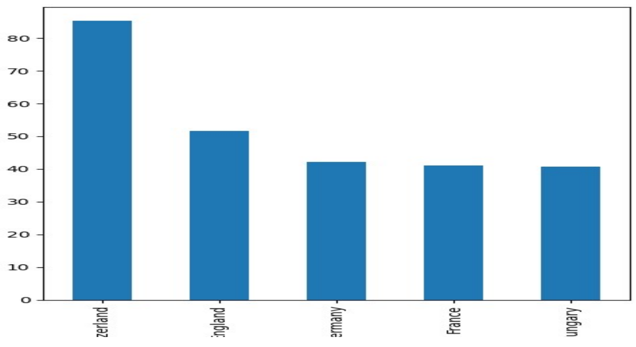
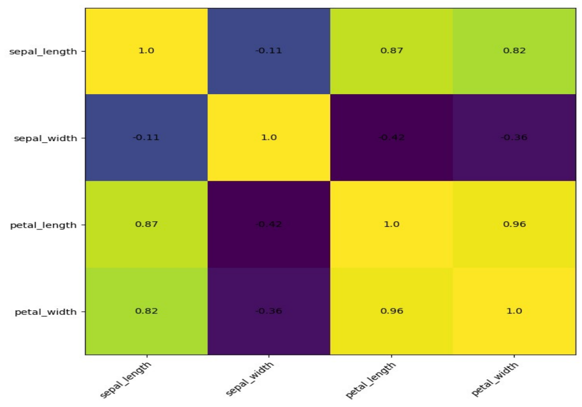


Multiple histogram for attributes of IRIS Dataset



Vertical bar chart for Wine Review Scores



		Horizontal bar chart for Wine Review Scores  Bar chart for Wine Review with highest cost five different Counties.  **Correlation Matrix** 
b.	Actual Result:	

7. **Viva**:

4. LINEAR REGRESSION

1. **Problem Statement:** Write a python program Application Program Linear Regression.

2. **Expected Learning Outcomes:** To train the students to understand the concept linear regression in python.

1. **Problem Analysis:** Consider the linear line y = a + bx. Find the value of b = n∑xy − (∑x) (∑y) / n∑x^2−$(∑x)^2$, a = ∑y−b (∑x)n, implement the value of a and b in y = a + bx and solve the equation. Once if you get the value of a, b and can regress y value for any x.

2. **A. Input:** Get any value of x.

   B.Output: Find the value of y for any x.

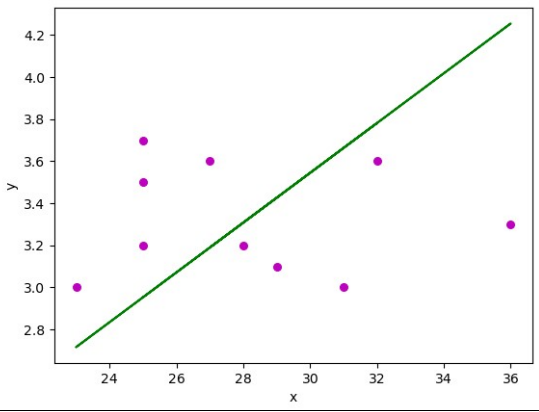
3. **Algorithm:**

   Step1: Consider a set of values x, y.
   Step2: Take the linear set of equation y = a+bx.
   Step3: Computer value of a, b with respect to the given values, b = n∑xy − (∑x) (∑y) / n∑x^2−$(∑x)^2$, a = ∑y−b (∑x)n.
   Step4: Implement the value of a, b in the equation y = a+ bx.
   Step5: Regress the value of y for any x.

4. **Coding:**

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)
    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)
    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x - n*m_y*m_x)
    SS_xx = np.sum(x*x - n*m_x*m_x)
    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    return(b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)
    # predicted response vector
    y_pred = b[0] + b[1]*x
    # plotting the regression line
    plt.plot(x, y_pred, color = "g")
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()

def main():
    # observations
    x = np.array([25, 23, 25, 31, 32, 25, 36, 27, 28, 29])
    y = np.array([3.2, 3, 3.5, 3, 3.6, 3.7, 3.3, 3.6, 3.2, 3.1])
        # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {} nb_1 = {}".format(b[0], b[1]))
    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

5. Test case :

a.	Expected Result:	Estimated coefficients: a = -0.006776599644680914 b = 0.11839062632187473 
b.	Actual Result:	

6. **Viva**:

5. CLASSIFICATION USING LOGISTIC REGRESSION

1. **Problem Statement:** Write a python program Application Program to perform Classification using Logistic Regression.

2. **Expected Learning Outcomes:** To train the students to understand the basics logistics regression using python.

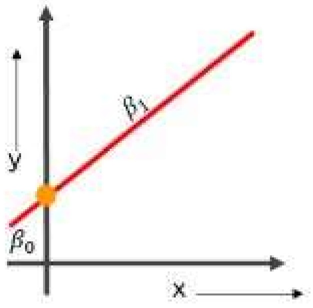
3. **Problem Analysis:**

Let's say that your goal is to build a logistic regression model in Python in order to determine whether candidates would get admitted to a prestigious university.
To understand logistic regression, let's go over the odds of success.
Odds (θ) = Probability of an event happening / Probability of an event not happening.

θ = p / 1 - p

The values of odds range from zero to ∞ and the values of probability lies between zero and one.

Consider the equation of a straight line:
$y = \beta 0 + \beta 1 * x$



4.

Here, $\beta 0$ is the y-intercept
$\beta 1$ is the slope of the line
x is the value of the x coordinate
y is the value of the prediction
Now to predict the odds of success, we use the following formula:

$$\log\left(\frac{p(x)}{1-P(x)}\right) = \beta_0 + \beta_1 x$$

Exponentiations both the sides, we have:

$$e^{ln\left(\frac{p(x)}{1-p(x)}\right)} = e^{\beta_0+\beta_1 x}$$

$$\left(\frac{p(x)}{1-p(x)}\right) = e^{\beta_0+\beta_1 x}$$

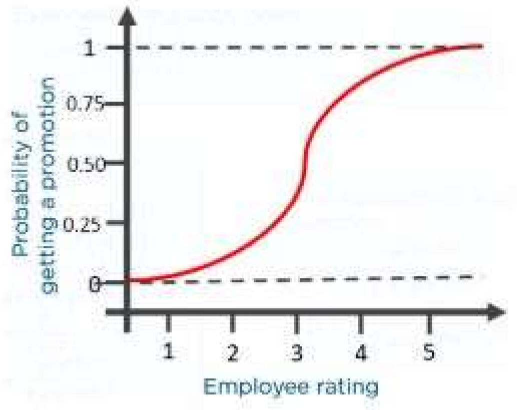
Let Y = e $\beta 0+\beta 1$ * x
Then p(x) / 1 - p(x) = Y
p(x) = Y(1 - p(x))
p(x) = Y - Y(p(x))
p(x) + Y(p(x)) = Y
p(x)(1+Y) = Y
p(x) = Y / 1+Y

$$p(x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}$$

The equation of the sigmoid function is:

$$p(x) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}}$$

The sigmoid curve obtained from the above equation is as follows:



5. **A. Input:** GMAT score, GPA and Years of work experience directly given in the Program as input.

   B. Output: Aspiring candidate get admitted or not.

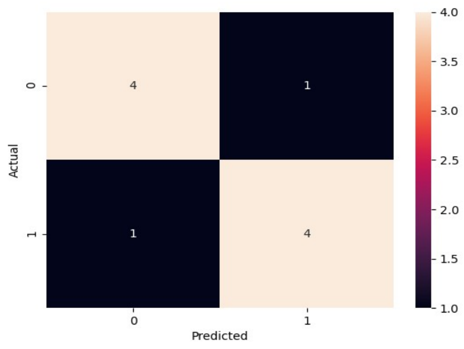
6. Algorithm:

Step1: Initialize the variables
Step2: Set the Data frame
Step3: Spilt data set into training and testing.
Step4: Fit the data into logistic regression function.
Step5: Predict the test data set.
Step6: Print the results.

7. Coding:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
candidates = {'gmat':
[780,750,690,710,680,730,690,720,740,690,610,690,710,680,770,610,580,650,540,590,620
,600,550,550,570,670,660,580,650,660,640,620,660,660,680,650,670,580,590,690],
          'gpa':
[4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.7,2.3,3.3,2,2.3,2.7,3,3.3,
3.7,2.3,3.7,3.3,3,2.7,4,3.3,3.3,2.3,2.7,3.3,1.7,3.7],
          'work_experience':
[3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4,2,6,5,1,2,4,6,5,1,2,1,4,5],
          'admitted':
[1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,0,0,0,1]
          }

df = pd.DataFrame(candidates,columns= ['gmat', 'gpa','work_experience','admitted'])

print (df)
X = df[['gmat', 'gpa','work_experience']]
y = df['admitted']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)
print (X_train)
print (y_train)
logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
print (X_test) #test dataset
print (y_pred) #predicted values
print('confusion_matrix:', confusion_matrix, sep='\n', end='\n\n')
plt.show()
```

8. Test case :

a.	Expected Result:	
b.	Actual Result:	

9. **Viva**:

6. NORMAL DISTRIBUTIONS

1. **Problem Statement:** Write a python program Application Program for plot a normal distribution with Matplotlib and Scipy.

2. **Expected Learning Outcomes:** To train the students to understand the basics Normal distribution.

3. **Problem Analysis:** Normal distribution is a symmetric probability distribution with equal number of observations on either half of the mean.

   The parameters representing the shape and probabilities of the normal distribution are mean and standard deviation

   Python Scipy stats module can be used to create a normal distribution with mean and standard deviation parameters using method norm.

   Standard normal distribution is normal distribution with mean as 0 and standard deviation as 1.

   In normal distribution, 68% of observations lie within 1 standard deviation, 95% of observations lie within 2 standard deviations and 99.7% observations lie within 3 standard deviations from the mean.

4. **A. Output:** Display of Normal Distribution curve in the range of -5 to 5.

5. **Algorithm:**

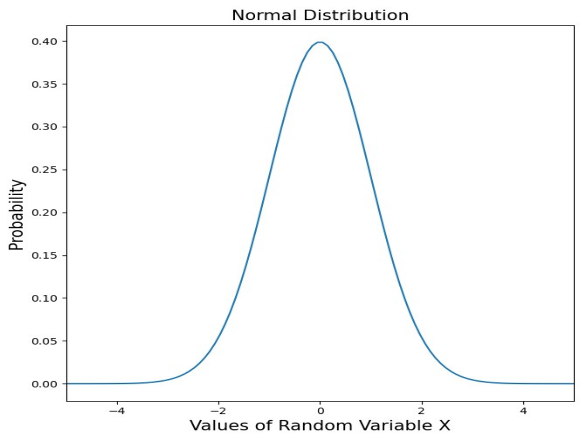
Step1: Set the Mean as 0 and Standard Deviation as 1.
Step2: Generate the set x of 100 random numbers in the range of -5 to 5.
Step3: Define the probability density function using x.
Step4: Plot the Normal Distribution.

6. **Coding:**

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
#
# Create a standard normal distribution with mean as 0 and standard deviation as 1
#
mu = 0
std = 1
snd = stats.norm(mu, std)
#
# Generate 100 random values between -5, 5
#
x = np.linspace(-5, 5, 100)
#
```

```
# Plot the standard normal distribution for different values of random variable
# falling in the range -5, 5
#
plt.figure(figsize=(7.5,7.5))
plt.plot(x, snd.pdf(x))
plt.xlim(-5, 5)
plt.title('Normal Distribution', fontsize='15')
plt.xlabel('Values of Random Variable X', fontsize='15')
plt.ylabel('Probability', fontsize='15')
plt.show()
```

7. Test case :

a.	Expected Result:	
b.	Actual Result:	

8. Viva:

7. KNN CLASSIFICATION

1. **Problem Statement:** Write a python program Application Program for KNN Classifier with IRIS Dataset.

2. **Expected Learning Outcomes:** To train the students to understand the basics of IRIS dataset and KNN Classification Program.

3. **Problem Analysis:**

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Distance functions

Euclidean

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^{k}\left|x_i - y_i\right|$$

Minkowski

$$\left(\sum_{i=1}^{k}(\left|x_i - y_i\right|)^q\right)^{1/q}$$

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.

4. **A. Input :** IRIS Dataset and test data.

B. Output: Category of the new test data item.

5. Algorithm:

Step 1: Load and Train the IRIS data
Step 2: Initialize K to your chosen number of neighbors
Step 3: For each example in the data

i. Calculate the distance between the query example and the current example from the data.
ii. Add the distance and the index of the example to an ordered collection.
iii. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
iv. Pick the first K entries from the sorted collection
v. Get the labels of the selected K entries
vi. Classify the new category as the mode of the K labels and return type.

6. Coding:

```
# Make Predictions with k-nearest neighbors on the Iris Flowers Dataset
from csv import reader
from math import sqrt

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
            csv_reader = reader(file)
            for row in csv_reader:
                    if not row:
                            continue
                    dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
            row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
            lookup[value] = i
            print('[%s] => %d' % (value, i))
    for row in dataset:
            row[column] = lookup[row[column]]
    return lookup
```

```
# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

# Make a prediction with KNN on Iris Dataset
filename = 'iris.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# define model parameter
```

```
num_neighbors = 5
# define a new record
row = [5.1,3.7,1.5,0.4]
# predict the label
label = predict_classification(dataset, row, num_neighbors)
print('Data=%s, Predicted: %s' % (row, label))
```

7. Test case :

a.	Expected Result:	[Iris-setosa] => 0 [Iris-virginica] => 1 [Iris-versicolor] => 2 Data=[5.1, 3.7, 1.5, 0.4], Predicted: 0
b.	Actual Result:	

8. Viva:

8. PRINCIPAL COMPONENT ANALYSIS

1. **Problem Statement:** Write a python Application Program to demonstrate the Principal Component Analysis.

2. **Expected Learning Outcomes:** To train the students to understand the basics of Principal Component Analysis in python program.

3. **Problem Analysis:**

Principal components analysis (PCA) is a dimensionality reduction technique that enables you to identify correlations and patterns in a data set so that it can be transformed into a data set of significantly lower dimension without loss of any important information.

Step 1: Standardization of the data

Standardization is all about scaling your data in such a way that all the variables and their values lie within a similar range.

Therefore, standardizing the data into a comparable range is very important. Standardization is carried out by subtracting each value in the data from the mean and dividing it by the overall deviation in the data set.

It can be calculated like so:

$$Z = \frac{\text{Variable value - mean}}{\text{Standard deviation}}$$

Post this step, all the variables in the data are scaled across a standard and comparable scale.

Step 2: Computing the covariance matrix

Mathematically, a covariance matrix is a p × p matrix, where p represents the dimensions of the data set. Each entry in the matrix represents the covariance of the corresponding variables.

Consider a case where we have a 2-Dimensional data set with variables a and b, the covariance matrix is a 2×2 matrix as shown below:

$$\begin{bmatrix} Cov(a,a) & Cov(a,b) \\ Cov(b,a) & Cov(b,b) \end{bmatrix}$$

In the above matrix:

- Cov(a, a) represents the covariance of a variable with itself, which is nothing but the variance of the variable 'a'
- Cov(a, b) represents the covariance of the variable 'a' with respect to the variable 'b'. And since covariance is commutative, Cov(a, b) = Cov(b, a)

Here are the key takeaways from the covariance matrix:

- The covariance value denotes how co-dependent two variables are with respect to each other
- If the covariance value is negative, it denotes the respective variables are indirectly proportional to each other
- A positive covariance denotes that the respective variables are directly proportional to each other

Simple math, isn't it? Now let's move on and look at the next step in PCA.

Step 3: Calculating the Eigenvectors and Eigenvalues

Eigenvectors and eigenvalues are the mathematical constructs that must be computed from the covariance matrix in order to determine the principal components of the data set.

But first, let's understand more about principal components

What are Principal Components?

Simply put, principal components are the new set of variables that are obtained from the initial set of variables. The principal components are computed in such a manner that newly obtained variables are highly significant and independent of each other. The principal components compress and possess most of the useful information that was scattered among the initial variables.

If your data set is of 5 dimensions, then 5 principal components are computed, such that, the first principal component stores the maximum possible information and the second one stores the remaining maximum info and so on, you get the idea.

Now, where do Eigenvectors fall into this whole process?

Assuming that you all have a basic understanding of Eigenvectors and eigenvalues, we know that these two algebraic formulations are always computed as a pair, i.e, for every eigenvector there is an eigenvalue. The dimensions in the data determine the number of eigenvectors that you need to calculate.

Consider a 2-Dimensional data set, for which 2 eigenvectors (and their respective eigenvalues) are computed. The idea behind eigenvectors is to use the Covariance matrix to understand where in the data there is the most amount of variance. Since more variance in the data denotes more information about the data, eigenvectors are used to identify and compute Principal Components.

Eigenvalues, on the other hand, simply denote the scalars of the respective eigenvectors. Therefore, eigenvectors and eigenvalues will compute the Principal Components of the data set.

Step 4: Computing the Principal Components

Once we have computed the Eigenvectors and eigenvalues, all we have to do is order them in the descending order, where the eigenvector with the highest eigenvalue is the most significant and thus forms the first principal component. The principal components of lesser significances can thus be removed in order to reduce the dimensions of the data.

The final step in computing the Principal Components is to form a matrix known as the feature matrix that contains all the significant data variables that possess maximum information about the data.

Step 5: Reducing the dimensions of the data set

The last step in performing PCA is to re-arrange the original data with the final principal components which represent the maximum and the most significant information of the data set. In order to replace the original data axis with the newly formed Principal Components, you simply multiply the transpose of the original data set by the transpose of the obtained feature vector.

So that was the theory behind the entire PCA process. It’s time to get your hands dirty and perform all these steps by using a real data set.

4. **A. Input:** Import the dataset from the python library sci-kit-learn. Breast cancer data can be downloaded.
   B. Output: After performing PCA we will get the dataset with reduced dimension.

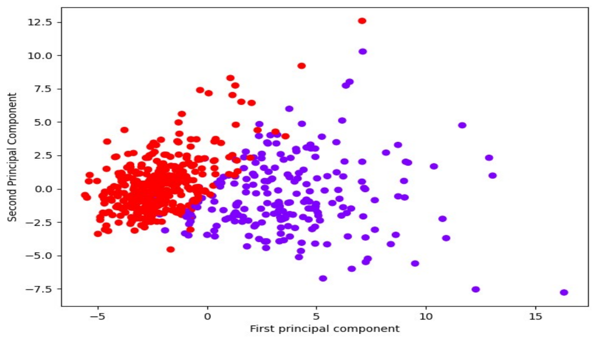
   Algorithm:

   Step-01: Get data.
   Step-02: Compute the mean vector (μ).
   Step-03: Subtract mean from the given data.
   Step-04: Calculate the covariance matrix.
   Step-05: Calculate the eigen vectors and eigen values of the covariance matrix.
   Step-06: Choosing components and forming a feature vector.
   Step-07: Deriving the new data set.

5. Coding:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
cancer.keys()
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df.head()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
print("Actual size",scaled_data.shape)
print("After PCA",x_pca.shape)
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='rainbow')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
plt.show()
map= pd.DataFrame(pca.components_,columns=cancer['feature_names'])
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')
plt.show()
```

6. Test case :

a.	Expected Result:	Actual size (569, 30) After PCA (569, 2) 
b.	Actual Result:	

7. **Viva**:

9. SINGULAR VALUE DECOMPOSITION

1. **Problem Statement:** Write a python Application Program to demonstrate the Singular Value Decomposition to find the document and string relativity.

2. **Expected Learning Outcomes:** To train the students to understand the basics of Singular Value Decomposition in python program.

3. **Problem Analysis:**

In this case, Singular Value Decomposition (SVD) is applied to the matrix; this is a form of a factor or more properly the mathematical generalization of which factor analysis is a special case. In general it is used for finding document relativity, dimensional reduction, Image Compression, Image Recovery, Spectral Clustering Background Removal from Videos, etc. In SVD, a rectangular matrix is decomposed into the product of three other matrices. One component matrix describes the original column entries in the same way, and the third is a diagonal matrix containing scaling values. When the three components are matrix-multiplied, the original matrix is reconstructed. The reconstructed two-dimensional matrix approximates the original matrix and a few highest values representing the top diagonal entries of the diagonal in the decomposition are selected to reconstruct the original matrix.
Each document in the particular sub-hierarchy represents the rows and each phrase with respect to the document is represented as the columns. Learning human like knowledge consists in formulating a bivariate frequency table with row i representing the ith phrase and column j representing the jth document (or between any two entities) and f_{ij} evaluated by the Shannon's measure of information $\sum p \log p$. This together with the dimension reduction will constitute the constraint satisfaction for prediction between the observed and the expected values to make classification. Actual data pertaining to any two measurable entities (phrases and sentences, text classification in digital libraries, etc.) will have to be collected. Sets of examples pertaining to each of the two entities can be exhibited in a bivariate frequency table for determining the relationships between any two examples. Tables can be formulated for comparison and valid conclusions.
SVD is a powerful technique employed for solving a linear system of equations **AX=B**, in M equations of N unknowns with M > = < N in order to get unique set of solutions; a set of singular solutions, infinite number of solutions; non trivial solutions or trivial solutions based upon the nature of the coefficient matrix A. Whatever maybe the vectors X, B concepts of rank, null space, range space of linear algebra are essential in formulating the computer program for any practical problem in conformity with the decomposition of the matrix A

$$[A] = [U][W][V^T]$$

When more equations than the unknowns are given, relevant solutions can be obtained by least squares method.

After the reconstruction of the original matrix we find the correlation between new document and the existing document in the sub-hierarchy. If the correlation is high then it is decided that the new document belongs to the particular sub-hierarchy category. The same process is repeated in all agents, if none of them finds correlation in its sub-hierarchy then it will be put under the miscellaneous category.

4. **A. Input:**

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

B. Output:

```
[0.1620579738985781, 0.40049828310613467, 0.37895454032072606,
0.4675662611791295, 0.17595367421580052, -0.0526549465800342, -
0.11514284281628315, -0.1591019817989112, -0.09183826787548349]

[0.258049326845599, 0.8411234345123202, 0.6057199488104255,
0.6973571707970627, 0.39231794947503684, 0.0331180051639017,
0.08324490705232306, 0.12177238577838864, 0.18737972500482497]

[-0.04308204300740035, 0.2539056647697741, -0.0966669539763707, -
0.20785820666699353, 0.1519133999898108, 0.22122703140656144,
0.5029448763149064, 0.706911627157333, 0.6155043995374361]

0.9385326876266149

The Given Line of text is belongs to Human Computer Interface Category
```

5. Algorithm:

1. Compose the matrix with row with text string and column with document.
2. SVD is the decomposition of a matrix A into 3 matrices – U, S, and V
3. S is the diagonal matrix of singular values. Think of singular values as the importance values of different features in the matrix
4. The rank of a matrix is a measure of the unique information stored in a matrix. Higher the rank, more the information

5. Eigenvectors of a matrix are directions of maximum spread or variance of data.
6. Find the relativity amount the data with a new document, if related print the related other print not related.

6. Coding:

```
import pandas as pd

# Python Program to find correlation coefficient.
import math

# function that returns correlation coefficient.
def correlationCoefficient(X, Y, n) :
    sum_X = 0
    sum_Y = 0
    sum_XY = 0
    squareSum_X = 0
    squareSum_Y = 0


    i = 0
    while i < n :
            # sum of elements of array X.
            sum_X = sum_X + X[i]

            # sum of elements of array Y.
            sum_Y = sum_Y + Y[i]

            # sum of X[i] * Y[i].
            sum_XY = sum_XY + X[i] * Y[i]

            # sum of square of array elements.
            squareSum_X = squareSum_X + X[i] * X[i]
            squareSum_Y = squareSum_Y + Y[i] * Y[i]

            i = i + 1

    # use formula for calculating correlation
    # coefficient.
    corr = (float)(n * sum_XY - sum_X * sum_Y)/ ( float )(math.sqrt((n * squareSum_X
-sum_X * sum_X)* (n * squareSum_Y - sum_Y * sum_Y)))
    return corr


c_names = ['c1','c2','c3','c4','c5','m1','m2','m3','m4']
words = ['human','interface',
'computer','user','system','response','time','EPS','survey','trees','graph','minors']
post_words = pd.DataFrame([[1, 0, 0, 1, 0, 0, 0, 0, 0],
                          [1, 0, 1, 0, 0, 0, 0, 0, 0],
                          [1, 1, 0, 0, 0, 0, 0, 0, 0],
                          [0, 1, 1, 0, 1, 0, 0, 0, 0],
                          [0, 1, 1, 2, 0, 0, 0, 0, 0],
                          [0, 1, 0, 0, 1, 0, 0, 0, 0],
                          [0, 1, 0, 0, 1, 0, 0, 0, 0],
                          [0, 0, 1, 1, 0, 0, 0, 0, 0],
                          [0, 1, 0, 0, 0, 0, 0, 0, 1],
                          [0, 0, 0, 0, 0, 1, 1, 1, 0],
                          [0, 0, 0, 0, 0, 0, 1, 1, 1],
                          [0, 0, 0, 0, 0, 0, 0, 1, 1]],
                         index = words,
                         columns = c_names)
#post_words.index.names = ['word:']
print(post_words)


import numpy as np

U, sigma, V = np.linalg.svd(post_words)
print("V = ")
print(np.round(V, decimals=2))

#V_df = pd.DataFrame(V, columns=c_names)
#print(V_df)

print(sigma)

print("U = ")
print(np.round(U, decimals=2))

A_approx = np.matrix(U[:, :2]) * np.diag(sigma[:2]) * np.matrix(V[:2, :])

print("A calculated using only the first two components:\n")
A_approxind = pd.DataFrame(A_approx, index=words, columns=c_names)
print("\n Recomposed value::\n")
print(A_approxind)

# Driver function
X = A_approx[:1].tolist()
X1 = []
Y1 = []
Z1 = []
print(type(X))
for i in X:
    for j in i:
        X1.append(j)

print(X1)
Y = A_approx[3:4].tolist()
for i in Y:
    for j in i:
        Y1.append(j)
print(Y1)

Z = A_approx[11:12].tolist()
for i in Z:
    for j in i:
        Z1.append(j)
print(Z1)
# Find the size of array.
n = len(X1)

# Function call to correlationCoefficient.
m = correlationCoefficient(X1, Y1, n)
print(m)
m1 = 0.5
if m > m1:
    print("The Given Line of text  is belongs to Human Computer Interface Category")
else:
    print("The Given Line of text does not belongs to Human Computer Interface
Category")
```

7. Test case :

a.	Expected Result:	[0.1620579738985781, 0.40049828310613467, 0.37895454032072606, 0.4675662611791295, 0.17595367421580052, -0.0526549465800342, -0.11514284281628315, -0.1591019817989112, -0.09183826787548349] [0.258049326845599, 0.8411234345123202, 0.6057199488104255, 0.6973571707970627, 0.39231794947503684, 0.0331180051639017, 0.08324490705232306, 0.12177238577838864, 0.18737972500482497] [-0.04308204300740035, 0.2539056647697741, -0.0966669539763707, -0.20785820666699353, 0.1519133999898108, 0.22122703140656144, 0.5029448763149064, 0.706911627157333, 0.6155043995374361] 0.9385326876266149 The Given Line of text is belongs to Human Computer Interface Category
b.	Actual Result:	

8. **Viva**:

10. K-MEANS CLUSTERING

1. **Problem Statement:** Write a C# Console Application Program to display and call-a-method using a reflection namespace.

2. **Expected Learning Outcomes:** To train the students to understand the basics of reflection namespace in C# Program.

3. **Problem Analysis:** K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

   It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

   It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

   It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

   The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

   The k-means clustering algorithm mainly performs two tasks:

   Determines the best value for K center points or centroids by an iterative process.

   Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

   Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

   The below diagram explains the working of the K-means Clustering Algorithm:

4. **A. Input:** X and Y values, K values initially Assumed

   B. Output: Different cluster of points grouped

5. **Algorithm:**

   Step-1: Select the number K to decide the number of clusters.
   Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.
Step-4: Calculate the variance and place a new centroid of each cluster.
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
Step-7: The model is ready.

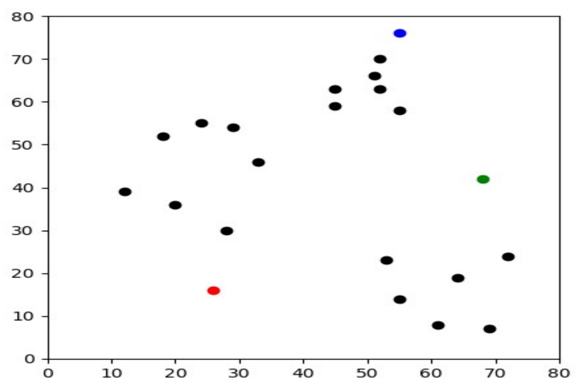
6. Coding:

```
# K-means algorithms
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame({
   'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72],
   'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24]
})


np.random.seed(200)
k = 3
# centroids[i] = [x, y]
centroids = {
   i+1: [np.random.randint(0, 80), np.random.randint(0, 80)]
   for i in range(k)
}

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color='k')
colmap = {1: 'r', 2: 'g', 3: 'b'}
for i in centroids.keys():
   plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

## Assignment Stage

def assignment(df, centroids):
   for i in centroids.keys():
      # sqrt((x1 - x2)^2 - (y1 - y2)^2)
      df['distance_from_{}'.format(i)] = (
         np.sqrt(
            (df['x'] - centroids[i][0]) ** 2
            + (df['y'] - centroids[i][1]) ** 2
         )
      )
   centroid_distance_cols = ['distance_from_{}'.format(i) for i in centroids.keys()]
```

```
    df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)
    df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_')))
    df['color'] = df['closest'].map(lambda x: colmap[x])
    return df

df = assignment(df, centroids)
print(df.head())

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()


## Update Stage

import copy

old_centroids = copy.deepcopy(centroids)

def update(k):
    for i in centroids.keys():
        centroids[i][0] = np.mean(df[df['closest'] == i]['x'])
        centroids[i][1] = np.mean(df[df['closest'] == i]['y'])
    return k

centroids = update(centroids)

fig = plt.figure(figsize=(5, 5))
ax = plt.axes()
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
for i in old_centroids.keys():
    old_x = old_centroids[i][0]
    old_y = old_centroids[i][1]
    dx = (centroids[i][0] - old_centroids[i][0]) * 0.75
    dy = (centroids[i][1] - old_centroids[i][1]) * 0.75
    ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i],
ec=colmap[i])
plt.show()

## Repeat Assigment Stage

df = assignment(df, centroids)
```

```
# Plot results
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

# Continue until all assigned categories don't change any more
while True:
    closest_centroids = df['closest'].copy(deep=True)
    centroids = update(centroids)
    df = assignment(df, centroids)
    if closest_centroids.equals(df['closest']):
        break

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

7. **Test case :**

a.	Expected Result:	Different Clusters: 
b.	Actual Result:	

8. **Viva**:

11. CORRELATION ANALYSIS

1. **Problem Statement:** Write a python Application Program to demonstrate the correlation with two set of values and display whether it is highly correlated or low correlated.

2. **Expected Learning Outcomes:** To train the students to understand the correlation between two attributes using python program.

3. **Problem Analysis:**

The formula for Pearson correlation coefficient for sample of size **n** (written as $\mathbf{r_{xy}}$) is given as:

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{\Sigma_i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{\Sigma_i=1}^{n}(y_i-\bar{y})^2}}$$

where **n** is the sample size, $\mathbf{x_i}$ & $\mathbf{y_i}$ are the ith sample points and $\bar{x}$ & $\bar{y}$ are the sample means for the random variables X and Y respectively.

A. Input: Set of Value X and Y given as Input
X = [15, 18, 21, 24, 27]
Y = [25, 25, 27, 31, 32]

B. Output: It displays whether the X and Y Correlated or Not.

4. **Algorithm:**

Step1: Compute the value of $\bar{x}$ & $\bar{y}$.

Step 2: Compute $\sum_{n=1}^{n}(X-\bar{x})(Y-\bar{y})$.

Step 3: Compute $r_{x,y} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{\Sigma_i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{\Sigma_i=1}^{n}(y_i-\bar{y})^2}}$

Step 4: Find it is highly correlated or low correlated and display the result.

5. Coding:

```
# Python Program to find correlation coefficient.
import math

# function that returns correlation coefficient.
def correlationCoefficient(X, Y, n) :
    sum_X = 0
    sum_Y = 0
    sum_XY = 0
    squareSum_X = 0
    squareSum_Y = 0
    i = 0
    while i < n :
            # sum of elements of array X.
            sum_X = sum_X + X[i]

            # sum of elements of array Y.
            sum_Y = sum_Y + Y[i]

            # sum of X[i] * Y[i].
            sum_XY = sum_XY + X[i] * Y[i]

            # sum of square of array elements.
            squareSum_X = squareSum_X + X[i] * X[i]
            squareSum_Y = squareSum_Y + Y[i] * Y[i]

            i = i + 1

    # use formula for calculating correlation
    # coefficient.
    corr = (float)(n * sum_XY - sum_X * sum_Y)/ ( float )(math.sqrt((n * squareSum_X
-sum_X * sum_X)* (n * squareSum_Y - sum_Y * sum_Y)))
    return corr

# Driver function
X = [15, 18, 21, 24, 27]
Y = [25, 25, 27, 31, 32]

print(X)
print(Y)
# Find the size of array.
n = len(X)

# Function call to correlationCoefficient.
z = correlationCoefficient(X, Y, n)
if(abs(z) > 0.5):
   print ('{0:.6f}'.format(z), "Highly COrrelated")
else:
   print('{0:.6f}'.format(z),"Low Correlated")
```

6. **Test case :**

a.	Expected Result:	[15, 18, 21, 24, 27] [25, 25, 27, 31, 32] 0.953463 Highly Correlated
b.	Actual Result:	

7. **Viva**:

12. VARIANCE ANALYSIS (ANNOVA)

1. **Problem Statement:** Write a python Application Program to demonstrate the Analysis of covariance (ANOVA).

2. **Expected Learning Outcomes:** To train the students to understand the basics Covariance (ANOVA) using python Program.

3. **Problem Analysis:**

   An Analysis of Variance Test, or ANOVA, can be thought of as a generalization of the t-tests for more than 2 groups. The independent t-test is used to compare the means of a condition between two groups. ANOVA is used when we want to compare the means of a condition between more than two groups.

   ANOVA tests if there is a difference in the mean somewhere in the model (testing if there was an overall effect), but it does not tell us where the difference is (if there is one). To find where the difference is between the groups, we have to conduct post-hoc tests.

   To perform any tests, we first need to define the null and alternate hypothesis:

   Null Hypothesis – There is no significant difference among the groups
   Alternate Hypothesis – There is a significant difference among the groups

   Basically, ANOVA is performed by comparing two types of variation, the variation between the sample means, as well as the variation within each of the samples. The below-mentioned formula represents one-way Anova test statistics.

   The result of the ANOVA formula, the F statistic (also called the F-ratio), allows for the analysis of multiple groups of data to determine the variability between samples and within samples.

$$F = \frac{\text{Explained Variance}}{\text{Unexplained variance}} \; or \; \frac{\text{Variance between groups}}{\text{Variance within groups}} = \frac{\text{Sum of squares between groups}}{\text{Sum of squares for error}}$$

$$= \frac{MST}{MSE} = \frac{\textit{Mean squared error treatments}}{\textit{Mean squared error}} = \frac{SS_B/Df_T}{SS_W/Df_E} = \frac{SS_B/(k-1)}{SS_W/(N-k)}$$

Where,

$$SS_B = \sum_i n_i\,(\bar{y}_i - \bar{y})^2$$

$$SS_w = \sum_{ij} (y_{ij} - \bar{y}_i)^2$$

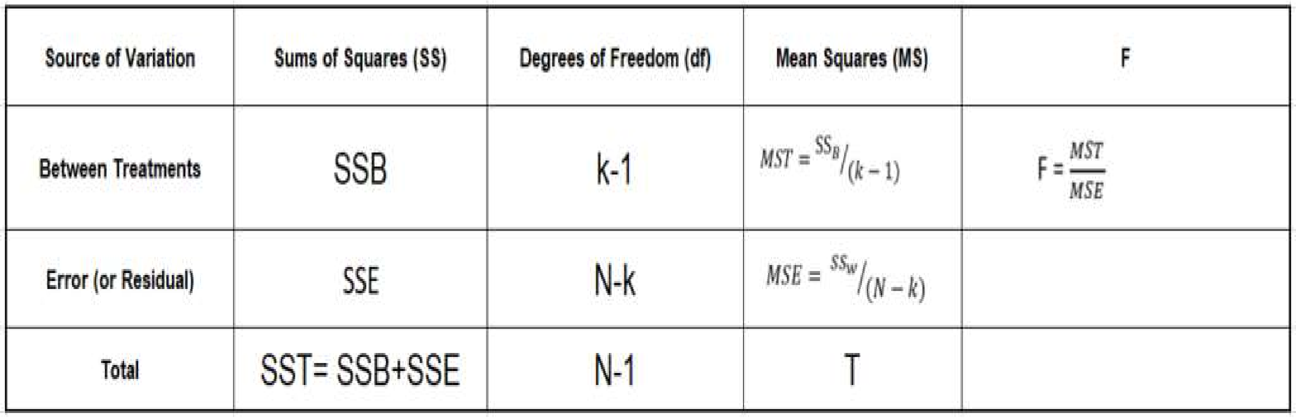
The formula for one-way ANOVA test can be written like this:

Where,

y_i $-$ *sample mean in the* i^{th} *group*
n_i $-$ *number of observation in the* i^{th} *group*
$\bar{y}$ $-$ *total mean of the data*
k $-$*total number of the groups*

y_{ij} $-$ j^{th} *observation in the out of k groups*
N $-$*Overall sample size*

When we plot the ANOVA table, all the above components can be seen in it as below:



Source of Variation	Sums of Squares (SS)	Degrees of Freedom (df)	Mean Squares (MS)	F
Between Treatments	SSB	k-1	$MST = {SS_B}/{(k-1)}$	$F = \frac{MST}{MSE}$
Error (or Residual)	SSE	N-k	$MSE = {SS_W}/{(N-k)}$	
Total	SST= SSB+SSE	N-1	T	

In general, if the p-value associated with the F is smaller than 0.05, then the null hypothesis is rejected and the alternative hypothesis is supported. If the null hypothesis is rejected, we can conclude that the means of all the groups are not equal.

Types of ANOVA Test**s**

1. **One-Way ANOVA:** A one-way ANOVA has just one independent variable
   - For example, differences in Corona cases can be assessed by Country, and a Country can have 2, 20, or more different categories to compare
2. **Two-Way ANOVA:** A two-way ANOVA (also called factorial ANOVA) refers to an ANOVA using two independent variables
   - Expanding the example above, a two-way ANOVA can examine differences in Corona cases (the dependent variable) by Age group (independent variable 1) and Gender (independent variable 2). Two-way ANOVA can be used to examine the interaction between the two independent variables. Interactions indicate that differences are not uniform across all categories of the independent variables
   - For example, Old Age Group may have higher Corona cases overall compared to the Young Age group, but this difference could be greater (or less) in Asian countries compared to European countries
3. **N-Way ANOVA**: A researcher can also use more than two independent variables, and this is an n-way ANOVA (with n being the number of independent variables you have), aka MANOVA Test.

- For example, potential differences in Corona cases can be examined by Country, Gender, Age group, Ethnicity, etc, simultaneously
- An ANOVA will give you a single (univariate) f-value while a MANOVA will give you a multivariate F-value.

4. **A. Input:**

   A bunch of students from different colleges taking the same exam. You want to see if one college outperforms the other, hence your null hypothesis is that the means of GPAs in each group are equivalent to those of the other groups. To keep it simple, we will consider 3 groups (college 'A', 'B', 'C') with 6 students each.

   A=[25,25,27,30,23,20]　　　B=[30,30,21,24,26,28]
   C=[18,30,29,29,24,26]

   Null Hypothesis: GPAs in each group are equivalent to those of the other groups.

   Alternate Hypothesis – There is a significant difference among the groups

   B. Output:

   To find the null hypothesis or alternate hypothesis acceptable.

5. **Algorithm:**

1. Rows are grouped according to their value in the category column.
2. The total mean value of the value column is computed.
   $$\bar{x}_{tot} = \frac{1}{n}\sum_{i=1}^{n} x_i$$
3. The mean within each group is computed.
4. The difference between each value and the mean value for the group is calculated and squared.
5. The squared difference values are added. The result is a value that relates to the total deviation of rows from the mean of their respective groups. This value is referred to as the *sum of squares within groups*, or **S2Wthn**.
6. For each group, the difference between the total mean and the group mean is squared and multiplied by the number of values in the group. The results are added. The result is referred to as the *sum of squares between groups*, or **S2Btwn**.
   $$S2Btwn = N_1(\bar{x}_1 - \bar{x}_{tot})^2 + N_2(\bar{x}_2 - \bar{x}_{tot})^2 + \cdots + N_N(\bar{x}_N - \bar{x}_{tot})^2$$
7. The two sums of squares are used to obtain a statistic for testing the null hypothesis, the so called F-statistic. The F-statistic is calculated as:

$$F = \frac{S2Btwn/dfBtwn}{S2Wthn/dfWthn}$$

where *dfBtwn* (degree of freedom between groups) equals the number of groups minus 1, and *dfWthn* (degree of freedom within groups) equals the total number of values minus the number of groups.

8. The F-statistic is distributed according to the F-distribution (commonly presented in mathematical tables/handbooks). The F-statistic, in combination with the degrees of freedom and an F-distribution table, yields the p-value.

   The p-value is the probability of the actual *or a more extreme* outcome under the null-hypothesis. The lower the p-value, the larger the difference.

6. Coding:

```
import pandas as pd
import numpy as np
import scipy.stats as stats
a=[25,25,27,30,23,20]
b=[30,30,21,24,26,28]
c=[18,30,29,29,24,26]
list_of_tuples = list(zip(a, b,c))
df = pd.DataFrame(list_of_tuples, columns = ['A', 'B', 'C'])
df
m1=np.mean(a)
m2=np.mean(b)
m3=np.mean(c)

print('Average mark for college A: {}'.format(m1))
print('Average mark for college B: {}'.format(m2))
print('Average mark for college C: {}'.format(m3))
m=(m1+m2+m3)/3
print('Overall mean: {}'.format(m))
SSb=6*((m1-m)**2+(m2-m)**2+(m3-m)**2)
print('Between-groups Sum of Squared Differences: {}'.format(SSb))
MSb=SSb/2
print('Between-groups Mean Square value: {}'.format(MSb))
err_a=list(a-m1)
err_b=list(b-m2)
err_c=list(c-m3)
err=err_a+err_b+err_c
ssw=[]
for i in err:
   ssw.append(i**2)
SSw=np.sum(ssw)
print('Within-group Sum of Squared Differences: {}'.format(SSw))
MSw=SSw/15
print('Within-group Mean Square value: {}'.format(MSw))
F=MSb/MSw
print('F-score: {}'.format(F))
print(stats.f_oneway(a,b,c))
```

7. Test case :

a.	Expected Result:	Average mark for college A: 25.0 Average mark for college B: 26.5 Average mark for college C: 26.0 Overall mean: 25.833333333333332 Between-groups Sum of Squared Differences: 6.999999999999999 Between-groups Mean Square value: 3.4999999999999996 Within-group Sum of Squared Differences: 223.5 Within-group Mean Square value: 14.9 F-score: 0.23489932885906037 F_onewayResult(statistic=0.2348993288590604, pvalue=0.793504662732833) We see that p-value > 0.05. Hence, we have to accept the Null Hypothesis – there are differences among different density groups for the given data.
b.	Actual Result:	

8. **Viva**:

13. MULTIPLE-LINEAR REGRESSION

1. **Problem Statement:** Write a python Application Program to demonstrate the Multiple Linear Regression.

2. **Expected Learning Outcomes:** To train the students to understand the Multi-Linear Regression using python program.

3. **Problem Analysis:**
   Multiple linear regression attempts to model the relationship between **two or more features** and a response by fitting a linear equation to the observed data. Clearly, it is nothing but an extension of simple linear regression.

   Consider a dataset with **p** features (or independent variables) and one response (or dependent variable).

   Also, the dataset contains **n** rows/observations.
   We define:

   X (**feature matrix**) = a matrix of size **n X p** where x_{ij} denotes the values of jth feature for ith observation.

   So,
$$\begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \vdots & x_{np} \end{pmatrix}$$
   and

   y (**response vector**) = a vector of size **n** where y_{i} denotes the value of response for ith observation.

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix}$$

   The regression line for **p** features is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}$$
   where h(x_i) is predicted response value for ith observation and b_0, b_1, ..., b_p are the regression coefficients**.**

   Also, we can write:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i$$
$$or$$
$$y_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

where e_i represents residual error in ith observation.
We can generalize our linear model a little bit more by representing feature matrix **X** as:

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

So now, the linear model can be expressed in terms of matrices as:

$$y = X\beta + \varepsilon$$

Where,

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ \beta_p \end{bmatrix}$$

And

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_n \end{bmatrix}$$

Now, we determine an estimate of b**,** i.e. b' using the Least Squares method**.** As already explained, the Least Squares method tends to determine b' for which total residual error is minimized.

We present the result directly here:

$$\hat{\beta} = (X'X)^{-1}X'y$$

where ' represents the transpose of the matrix while -1 represents the matrix inverse. Knowing the least square estimates, b', the multiple linear regression model can now be estimated as:

$$\hat{y} = X\hat{\beta}$$

where y’ is the estimated response vector**.**

4. **A. Input:** Boston house pricing dataset using Scikit-learn.
   B. Output: List of all Coefficients**,** variance score and residual error plots

5. **Algorithm:**

Step1: Get the multi-attribute dataset using the Scikit-learn data source.

Step 2: Create a regression object.

Step 3: Train the dataset with the regression model fit.

Step 4: Get and print the regression coefficients and variance.

Step 5. Plot the residual error.

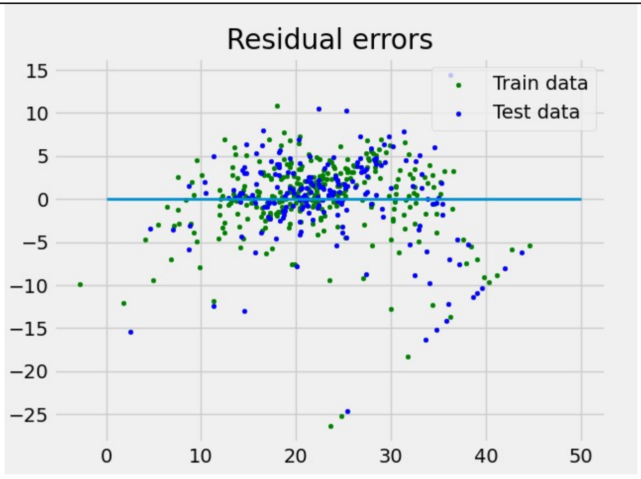
6. **Coding:**

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

# load the boston dataset
boston = datasets.load_boston(return_X_y=False)

# defining feature matrix(X) and response vector(y)
X = boston.data
y = boston.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=1)

# create linear regression object
reg = linear_model.LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# regression coefficients
print('Coefficients: ', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))

# plot for residual error
```

```
## setting plot style
plt.style.use('fivethirtyeight')

## plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
          color = "green", s = 10, label = 'Train data')

## plotting residual errors in test data
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,
          color = "blue", s = 10, label = 'Test data')

## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

## plotting legend
plt.legend(loc = 'upper right')

## plot title
plt.title("Residual errors")

## method call for showing the plot
plt.show()
```

7. **Test case :**

a.	Expected Result:	Coefficients: [-8.95714048e-02 6.73132853e-02 5.04649248e-02 2.18579583e+00 -1.72053975e+01 3.63606995e+00 2.05579939e-03 -1.36602886e+00 2.89576718e-01 -1.22700072e-02 - 8.34881849e-01 9.40360790e-03 -5.04008320e-01] Variance score: 0.720905667266178

		
b.	Actual Result:	

8. **Viva**: