DISEASE PREDICTION

USING MACHINE LEARNING ALGORITHM

A Course Project report submitted

in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

BY

B. Nithisha **2203A52004**

Under the guidance of

Dr. KIRAN ERANKI

Associate professor, Department of CSE.

CERTIFICATE

This is to certify that the Project entitled **"DISEASE PREDICTION USING MACHINE LEARNING ALGORITHMS"** is the bonafide work carried out by **Nithisha Bandameedi** as a Course Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **ARTIFICAL INTELLIGENCE AND MACHINE LEARNING** during the academic year 2023-2024 under my guidance and Supervision.

Dr.KIRAN ERANKI
Assoc. Prof. (CSE)
SR University,
Ananthasagar, Warangal.

Dr. M. SHESHIKALA
Assoc. Prof.& associate dean(CSE)
SR University,
Ananthasagar, Warangal

ABSTRACT

This project explores the use of machine learning techniques, specifically Logistic Regression and Support Vector Machines (SVM), in healthcare analysis. Our main goal is to employ these methods to predict disease outcomes based on patient information and symptoms, with the overarching objective of improving healthcare decision-making and patient well-being.

Logistic Regression is a tool we use to build a model that categorizes patients into different groups, such as positive or negative for a particular ailment. We utilize a simple mathematical function to estimate probabilities, which helps healthcare providers make educated predictions about disease outcomes.

At the same time, we apply the Support Vector Machine (SVM) algorithm to craft a Support Vector Classifier (SVC) model. SVM is known for its effectiveness in classification tasks, and it's employed here to find the best way to separate patients into different categories. SVM's precision in classification tasks is vital in healthcare analysis, as it assists in identifying disease outcomes and patient profiles.

These machine learning techniques serve as powerful tools for healthcare analysis, with the potential to enhance the accuracy of disease predictions and patient care. By classifying patients based on their information and symptoms, healthcare professionals can make well-informed decisions, ultimately leading to improved patient outcomes. This project demonstrates the practical application of machine learning in healthcare, showcasing its potential to positively impact the medical field.

ACKNOWLEDGEMENT

I express thanks to Course co-coordinator **Dr. KIRAN ERANKI ,Assoc. Prof.** for guiding me from the beginning through the end of the Course Project. I express my gratitude to Associate Dean of department CS&AI, **Dr. M.Sheshikala, Associate Professor** for encouragement, support and insightful suggestions .I truly value their consistent feedback on my progress, which was always constructive and encouraging and ultimately drove me to the right direction. Finally, I express my thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

INDEX

CHAPTER 1

INTRODUCTION

In the world of healthcare, understanding and predicting diseases is a critical task. Machine learning, a branch of artificial intelligence, has come to our aid in making this process more efficient and accurate.

When a patient experiences various symptoms, like fever, cough, fatigue, and many others, doctors need to determine the underlying health condition. But what if we could teach a computer to recognize patterns and associations in the symptoms and health profiles of countless patients? This is where machine learning steps in.

Machine learning algorithms are trained on large datasets containing information about patients, their symptoms, and the diseases they've been diagnosed with. The magic happens when these algorithms analyze this data and learn to make connections between specific symptoms and certain diseases. It's like teaching the computer to recognize that a high fever and difficulty breathing might indicate a respiratory illness, for example.

Once the machine learning model is trained, it can predict or suggest possible diseases when given a new set of symptoms and patient profile information. This not only assists doctors in making accurate diagnoses but also helps in identifying diseases at an early stage, which can be crucial for effective treatment.

In essence, disease symptom and patient profile prediction using machine learning is like having a smart assistant for doctors. It doesn't replace their expertise but enhances it by providing valuable insights and suggestions, ultimately leading to better patient care and outcome**s.**

1.1 PROBLEM STATEMENT

In the field of healthcare, accurate disease diagnosis is often complex due to the diverse array of symptoms and patient-specific factors to consider. This project addresses this challenge by leveraging machine learning techniques to predict diseases and patient profiles based on symptom data and medical histories. Our objective is to create an intelligent system that aids healthcare professionals in making more precise and timely diagnoses, ultimately reducing the risk of misdiagnoses. This system will be a valuable addition to medical practice, supporting early disease detection and personalized treatment plans, which can significantly enhance the quality of patient care. By automating the analysis of comprehensive health data, the project aspires to make healthcare more efficient and effective, resulting in improved patient outcomes and better healthcare delivery.

1.2 OBJECTIVE

- Help doctors diagnose diseases accurately.
- Use computers to predict diseases based on symptoms.
- Reduce mistakes in disease identification.
- Improve patient care by offering early detection and personalized treatment.
- Make healthcare more efficient with smart technology.
- Enhance the capabilities of healthcare professionals.

These objectives aim to create a system that supports doctors in providing better healthcare by leveraging machine learning for disease prediction.

1.3 ARCHITECTURE

The architecture of this machine learning model is “SUPERVISED LEARNING” and the process involved is data acquisition, data processing, data modelling and execution (parameter tuning and making predictions). The supervised can be further broadened into classification and regression analysis based on output criteria.

1.4 REQUIREMENT SPECIFICATIONS

Software Requirements:

- ✓ **OS** **:** Windows 8 or Higher Versions
- ✓ **Platform** **:** Google colab
- ✓ **Program Language** **:** Python

CHAPTER 2

LITERATURE SURVEY

Machine learning is a powerful approach that mimics how humans learn but does it with computers. In the context of disease symptom prediction, it works like this: Imagine having a super-smart digital assistant that's been trained on a massive amount of data about symptoms and patients. This data is like a vast collection of medical cases that it learns from, just like students learn from many examples in school.

The magic happens when this digital assistant starts finding patterns in the data. It recognizes that when a patient has certain symptoms, there's a likelihood of them having a particular disease. For example, if someone shows signs of a high fever and a persistent cough, this intelligent system might suggest they have the flu. The more data it learns from, the better it becomes at making these connections.

What's really exciting is that these machine learning models are not just theoretical concepts; they're being put to work in real-world healthcare settings. Picture them as 'digital medical detectives' working side by side with doctors. They predict a wide range of diseases, from everyday illnesses to more severe medical conditions. The remarkable part is that they're often highly accurate in their predictions.

However, here's the catch: these digital detectives hunger for data. The more comprehensive and varied the data they learn from, the sharper their predictions become. So, researchers are continuously on the lookout for more and better data to make these digital assistants even smarter. The ultimate goal is to revolutionize healthcare by catching diseases early, personalizing treatment plans, and ensuring better care for everyone.

CHAPTER 3

DATA PRE-PROCESSING

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data and while doing any operation with data. It is mandatory to clean it and put in a formatted way. So, for this we use data pre-processing task. In this particular section we re-label and convert some categorical features into numeric values. This is crucial for training machine learning models since machine learning models accepts the numeric values.

3.1 FLOW CHART

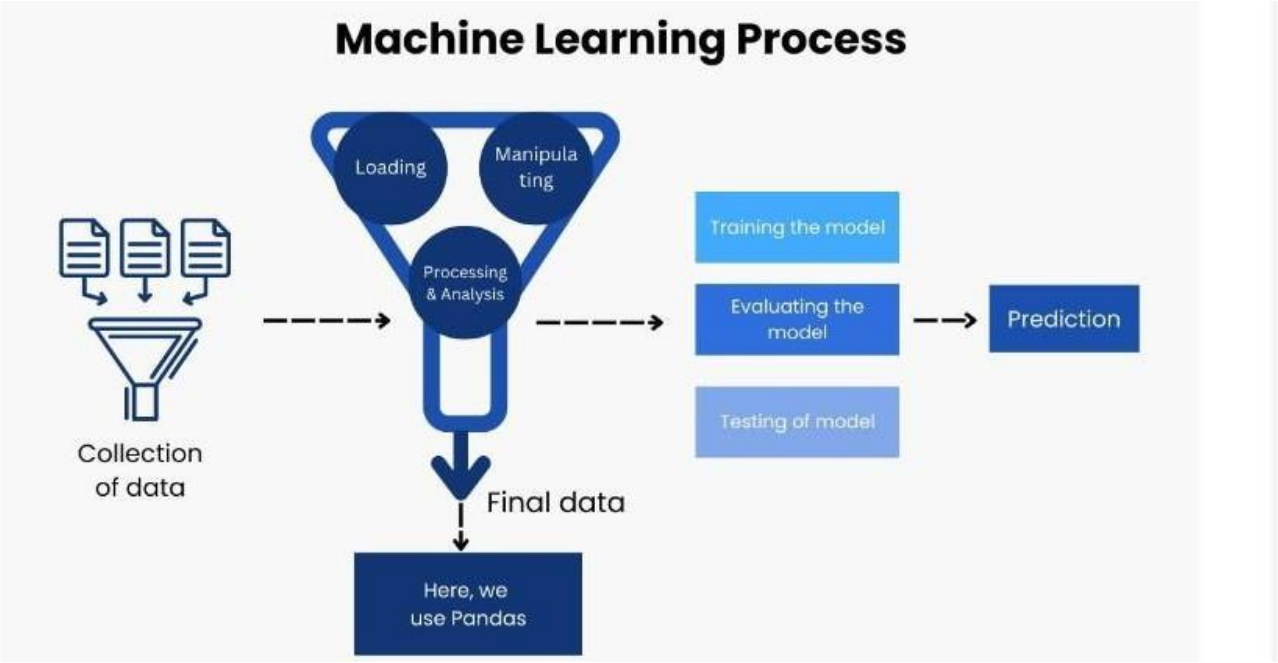


FIG 3.1 FLOW CHART

3.2 DATA SET

For this project, I have obtained my data set from Kaggle. This data set contains 349 rows of data and 10 columns (features) that we could focus onto build our prediction model i.e., I have used 10 attributes to predict heart disease analysis.

	Disease	Fever	Cough	Fatigue	Difficulty B	Age	Gender	Blood Pres	Cholestero	Outcome Variable
2	Influenza	Yes	No	Yes	Yes	19	Female	Low	Normal	Positive
3	Common C	No	Yes	Yes	No	25	Female	Normal	Normal	Negative
4	Eczema	No	Yes	Yes	No	25	Female	Normal	Normal	Negative
5	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive
6	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive
7	Eczema	Yes	No	No	No	25	Female	Normal	Normal	Positive
8	Influenza	Yes	Yes	Yes	Yes	25	Female	Normal	Normal	Positive
9	Influenza	Yes	Yes	Yes	Yes	25	Female	Normal	Normal	Positive
10	Hyperthyro	No	Yes	No	No	28	Female	Normal	Normal	Negative
11	Hyperthyro	No	Yes	No	No	28	Female	Normal	Normal	Negative
12	Asthma	Yes	No	No	Yes	28	Male	High	Normal	Positive
13	Allergic Rh	No	Yes	Yes	No	29	Female	Normal	Low	Negative
14	Anxiety Dis	No	Yes	No	No	29	Female	Normal	High	Negative
15	Common C	No	No	No	No	29	Female	Low	Normal	Negative
16	Diabetes	No	No	No	No	29	Male	Low	Normal	Negative
17	Gastroente	No	Yes	No	No	29	Female	Normal	Normal	Negative
18	Pancreatiti	Yes	No	No	No	29	Female	High	Normal	Negative
19	Rheumatoi	No	Yes	Yes	Yes	29	Female	High	High	Negative
20	Depressio	Yes	Yes	Yes	Yes	29	Male	High	Normal	Positive
21	Liver Cance	Yes	Yes	Yes	Yes	29	Female	Normal	Normal	Positive
22	Stroke	Yes	Yes	Yes	Yes	29	Female	Normal	Normal	Positive
23	Urinary Tra	Yes	Yes	Yes	No	29	Male	High	High	Positive
24	Dengue Fe	Yes	No	Yes	No	30	Female	Normal	Normal	Negative
25	Dengue Fe	Yes	No	Yes	No	30	Female	Normal	Normal	Negative
26	Eczema	No	Yes	Yes	No	30	Male	High	High	Negative
27	Gastroente	Yes	Yes	Yes	No	30	Male	High	High	Negative

FIG 3.2.1 DATA SET

3.3 DATA CLEANING

Data Cleaning means the process of identifying the incorrect, incomplete, inaccurate, irrelevant or missing part of the data and then modifying, replacing or deleting them according to the necessity. Data cleaning is considered a foundational element of the basic data science. Data is the most valuable thing for analytics and Machine learning. In computing or Business data is needed everywhere. When it comes to the present world data, it is not improbable that data may contain incomplete, inconsistent or missing values. If the data is corrupted then it may hinder the process or provide inaccurate results. In my data set there are no strings, sothere is no need of data cleaning.

3.4 DATA VISUALIZATION

Data visualization is the graphical representation of information and data in a pictorial or graphical format (Example: charts, graphs and maps). Data visualization tools provide an accessible way to see and understand trends, patterns in data and outliers. Data visualization tools and technologies are essential to analyzing massive amounts of information and making data-driven decisions. The concept of using pictures is to understand data that has been used for centuries. General types of data visualization are charts, tables, graphs, maps, dashboards etc... The below are the graphs of our data set which are plotted between each feature and price.

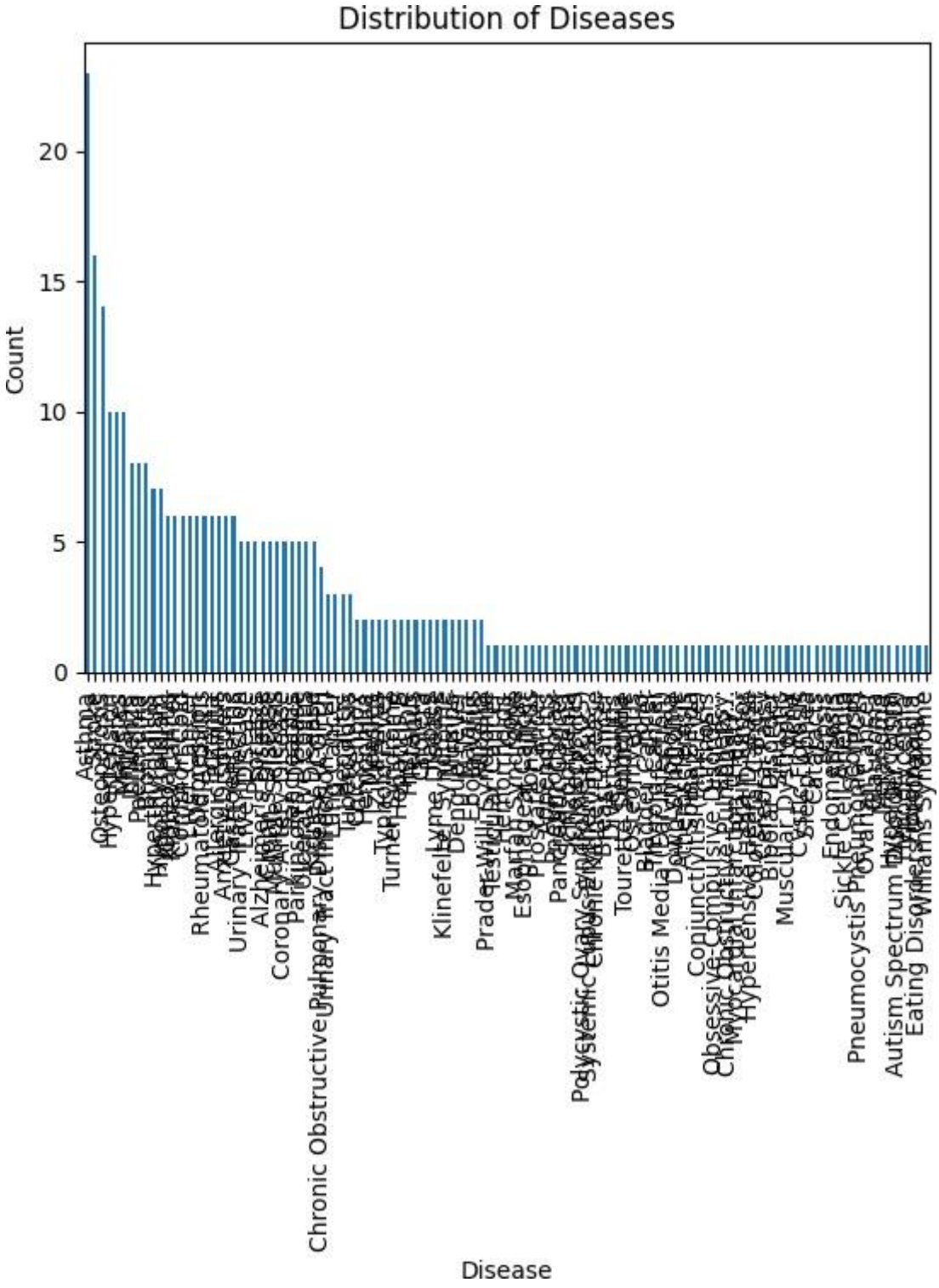


FIG 3.4.1 DISTRIBUTION OF DISEASE

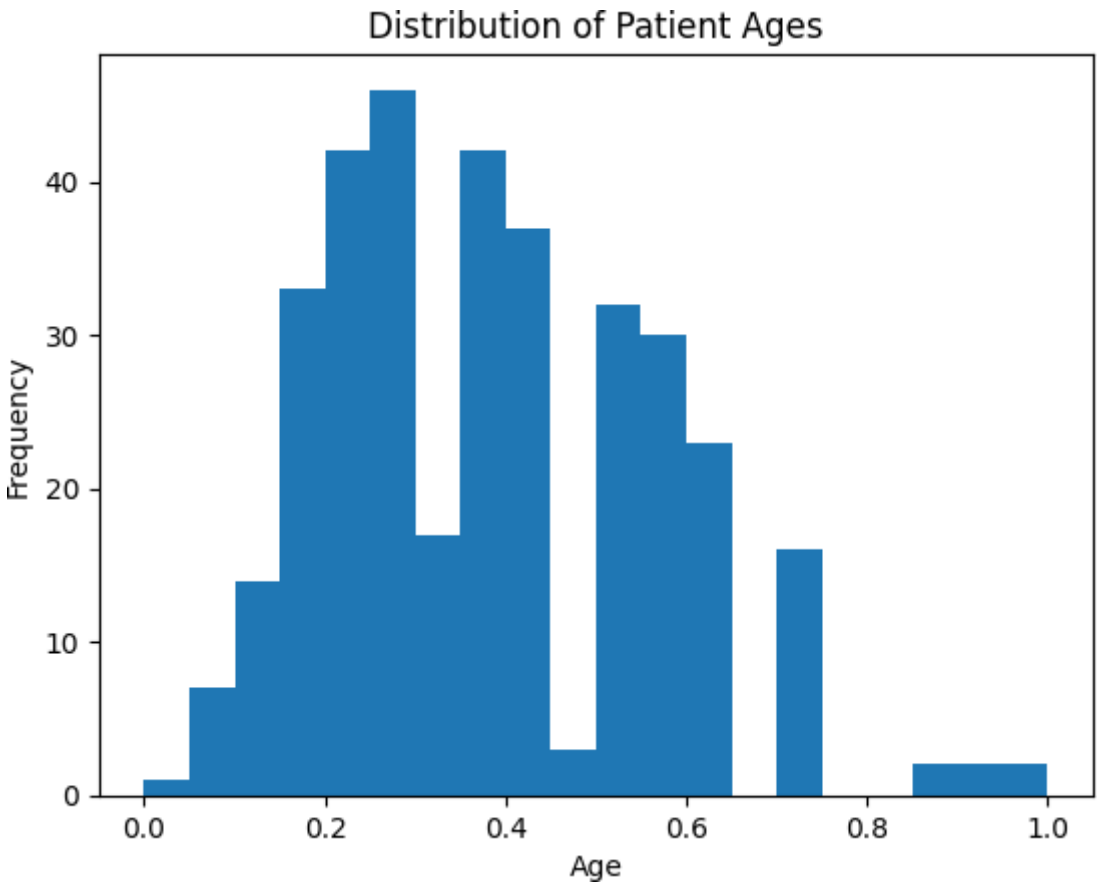


FIG 3.4.2 DISTRIBUTION OF PATIENT AGES

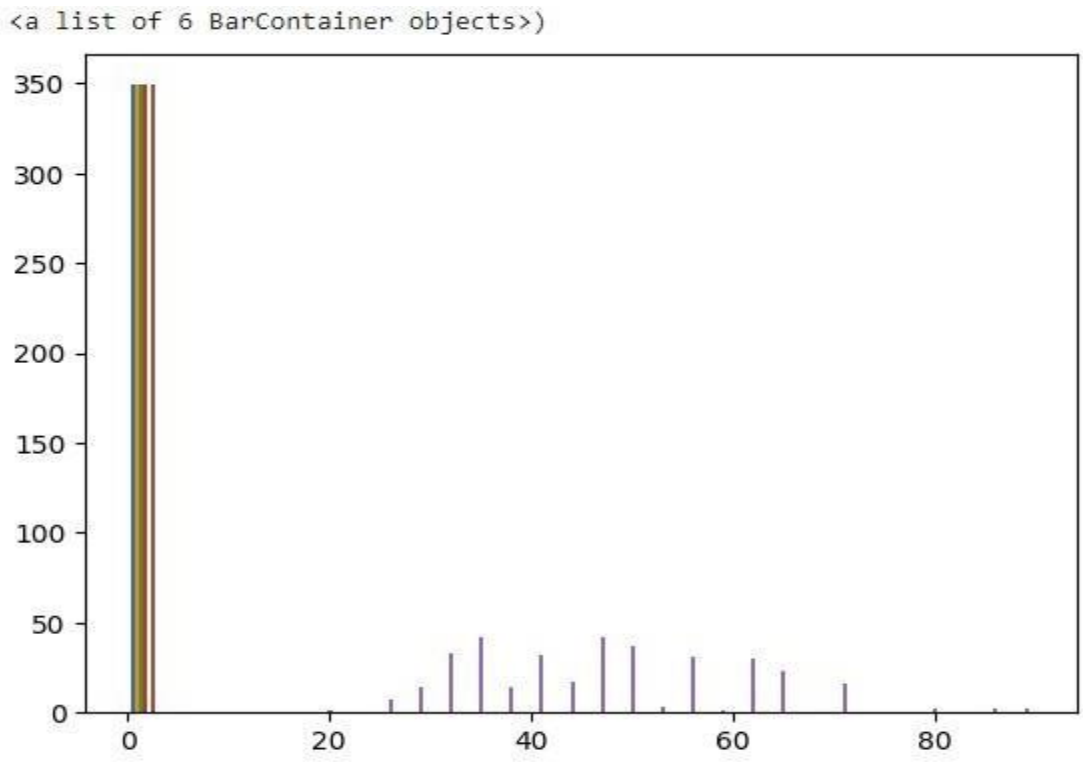


FIG 3.4.3 HISTOGRAM

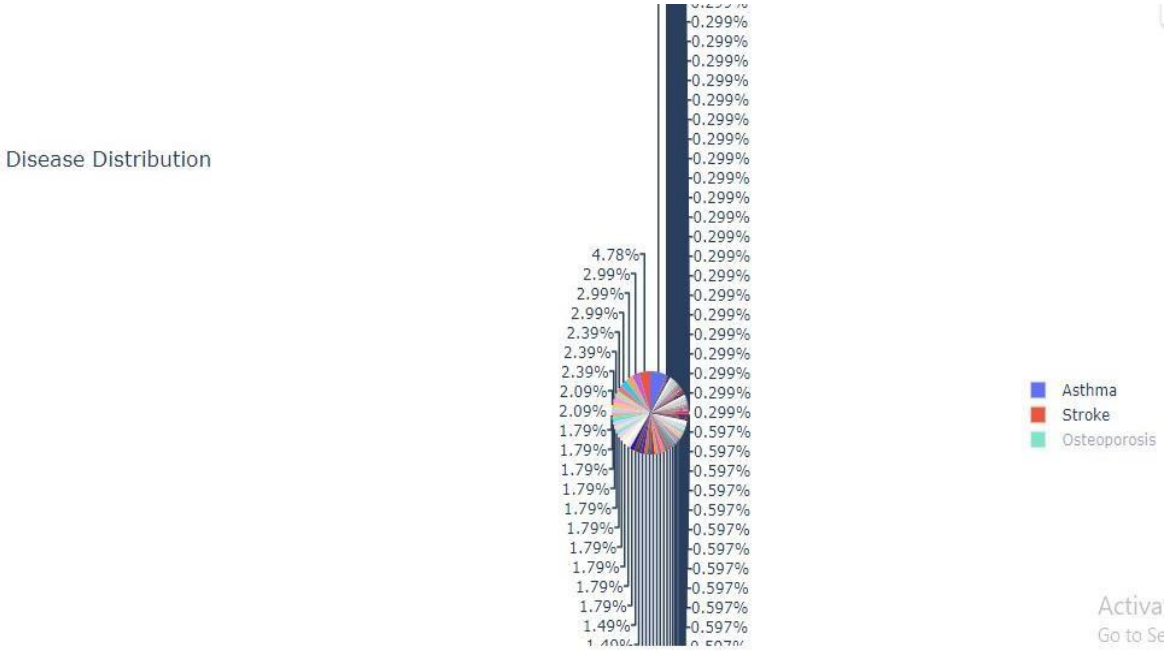


FIG 3.4.4 DISEASE DISTRIBUTION IN PIE CHART

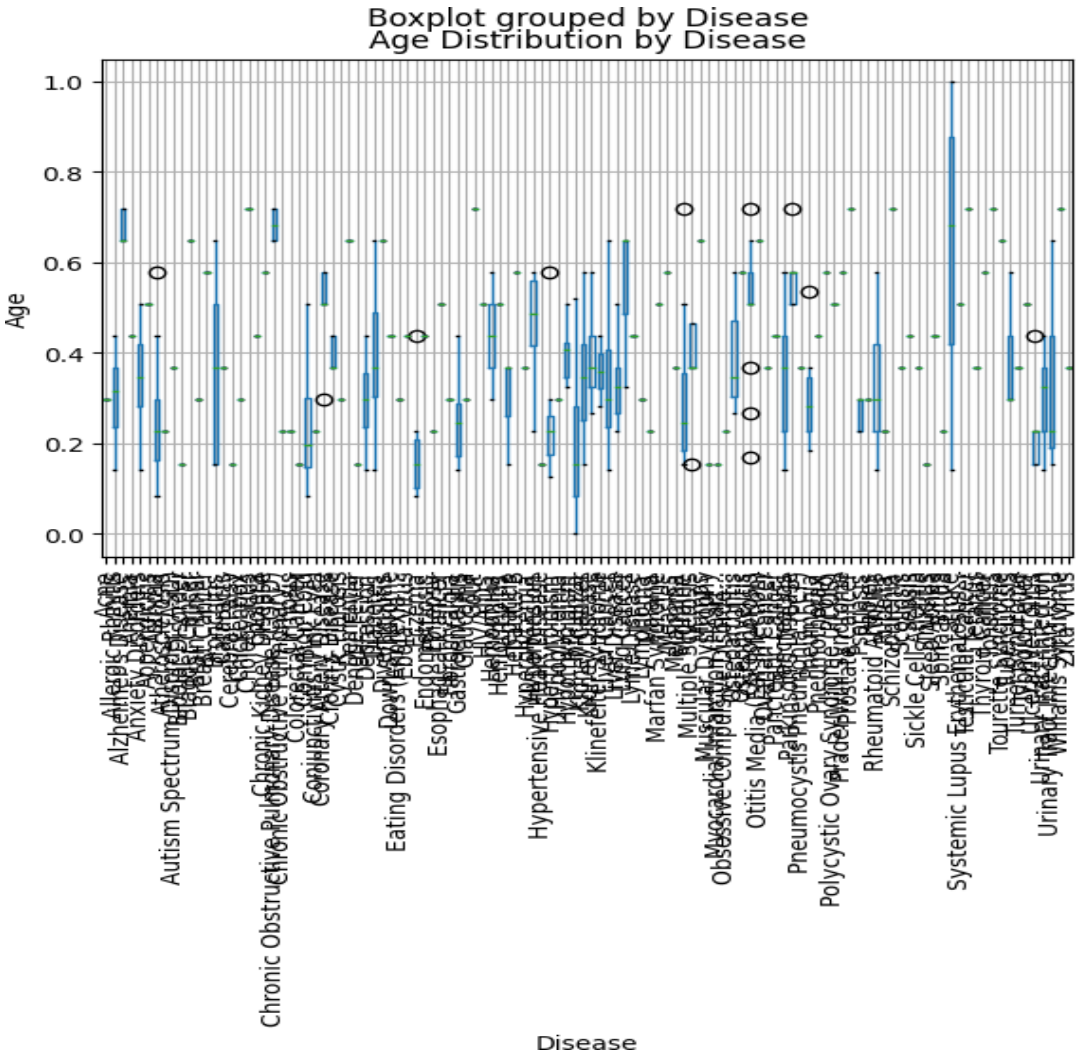


FIG 3.4.5 AGE DISTRIBUTION BY DISEASE

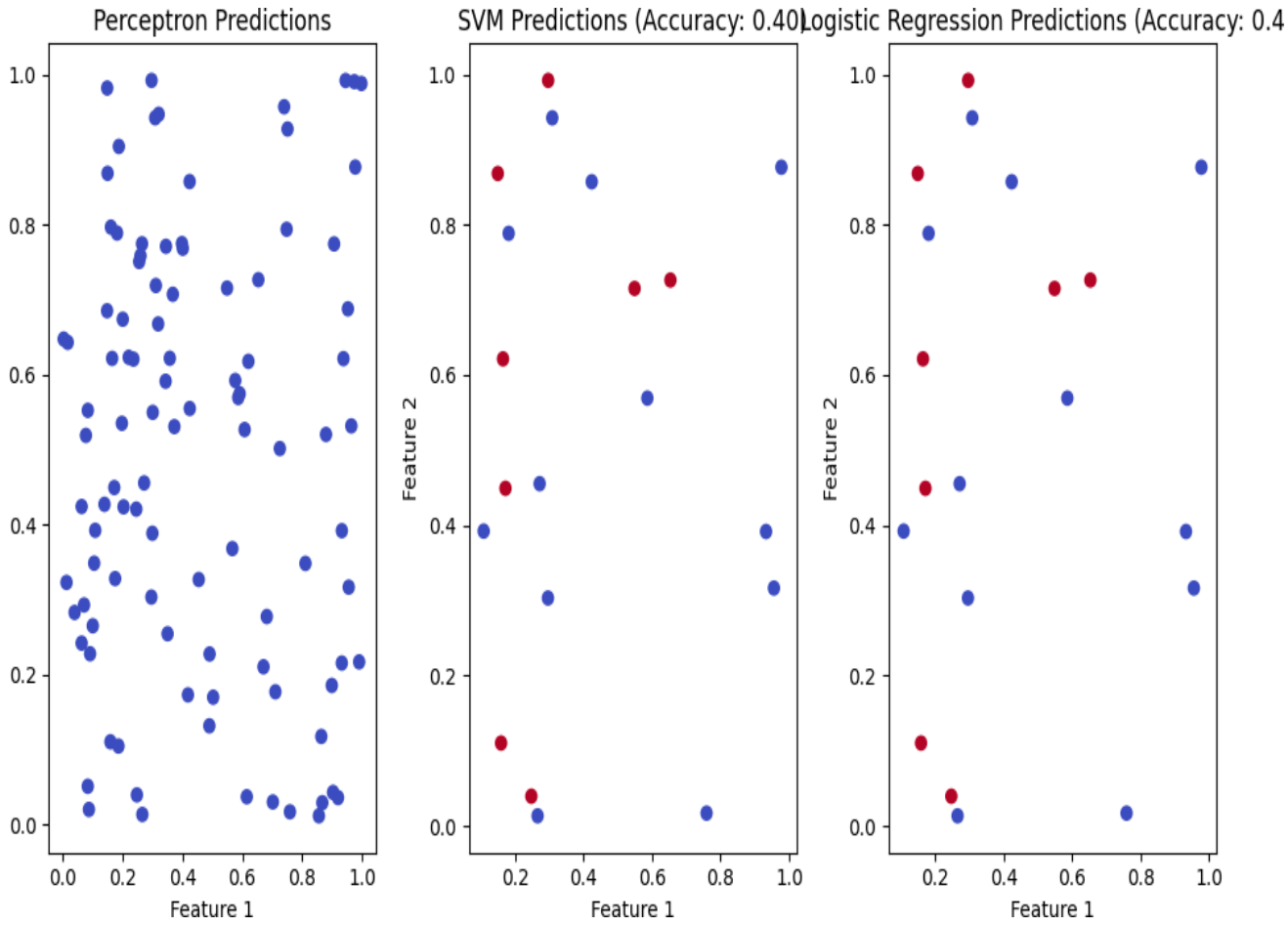


FIG 3.4.5 PERCEPTRON , SVM, LOGISTIC REGRESSION PREDICTIONS

CHAPTER 4

METHODOLOGY

Enough methods are performed on the data to evaluate the data set and gather knowledge about the data. Let's perform some Machine Learning model and Experimentation to create a model that helps us to achieve the goal I stated in the problem definition. In this we talk about the various machine learning algorithms used for the project. They are Classification, Logistic Regression and SVM.

4.1 CLASSIFICATION

The concept of classification is exemplified in the provided code snippet, where a simple perceptron model is introduced. Although basic, this model serves as a building block for understanding more complex classification algorithms. In this context, the code snippet showcases how symptoms and patient profiles can be evaluated to classify patients into different categories, such as "positive" or "negative" for a specific disease. The perceptron model assigns different weights to symptoms, a fundamental concept in classification, and uses a threshold to make binary predictions.

Moreover, the code snippet demonstrates the applicability of classification in disease symptom prediction. It highlights the importance of assigning categories to patients based on their symptoms and risk factors. This categorization is instrumental in the healthcare sector, as it aids in diagnosing diseases and making treatment decisions. By using classification techniques, healthcare professionals can identify whether a patient exhibits specific symptoms associated with a disease, ultimately contributing to more accurate and timely diagnoses.

Moving beyond the introductory perceptron model, more advanced classification techniques are applied in the code, such as Logistic Regression and Support Vector Machines (SVM). Logistic Regression is a powerful tool for binary and multi-class classification, particularly for estimating the probability of a patient having a particular disease. It provides a probabilistic approach to classification, enhancing the precision of predictions. In contrast, SVM is a versatile and effective algorithm that aids in distinguishing between different categories or labels. It identifies the optimal boundary between classes,

contributing to accurate predictions.

In conclusion, The provided code snippets serve as foundational steps, illustrating the use of classification to categorize patients based on symptoms, risk factors, or diagnostic test results. This categorization empowers healthcare professionals to make informed decisions about diagnosis and treatment. Whether through basic models like the perceptron, more advanced techniques like Logistic Regression, or sophisticated algorithms like SVM, classification is an essential tool in guiding healthcare decisions and improving patient outcome

4.2 SVM - SUPPORT VECTOR MACHINE

Certainly, here's an explanation for the Support Vector Machine (SVM) code snippet:

__SVM in the Given Code__

The code snippet you've provided is focused on Support Vector Machines (SVM), another popular machine learning algorithm. SVM is employed for classification and regression tasks, but in this case, it's used for classification.

__Code Overview:__
- This code aims to create, train, and evaluate an SVM model for a classification task.
- It begins by importing necessary libraries, including sci kit-learn' s datasets, train_test_split for data splitting, the SVC (Support Vector Classifier) model, and accuracy_score for performance evaluation.
- The data is split into training (X_train, y_train) and testing (X_test, y_test) sets using the train_test_split function.

__SVM Model:__
- SVM stands for Support Vector Machine and is widely used for classification tasks.
- A Support Vector Classifier (SVC) model is created with a linear kernel. The choice of a linear kernel implies that the model tries to find a linear boundary that best separates the data into different classes.

__Training the Model:__
- The model is trained on the training data using model.fit(X_train, y_train). During training, the SVM algorithm identifies the optimal decision boundary (hyperplane) that maximizes the margin between data points of different classes.

Predictions:
- Predictions are made on the test data with y_pred = model.predict(X_test). The SVM model assigns class labels to the test data based on the learned decision boundary.

__Evaluating Model Performance:__
- The code calculates the accuracy score using accuracy_score(y_test, y_pred). The accuracy score measures the proportion of correctly classified data points in the test

set, providing insight into the model's performance.

Role of SVM:

- SVM is a versatile algorithm used for classification tasks like identifying disease outcomes, image recognition, and spam email classification.
- In healthcare, SVM can be employed to predict patient outcomes, such as disease prognosis or patient response to treatment. It works by identifying patterns and relationships in medical data, allowing healthcare professionals to make informed decisions.

This code demonstrates how SVM can be used for healthcare analytics, where it plays a pivotal role in classifying patients or medical data into relevant categories based on their attributes, ultimately improving decision-making processes.

CODE:

```
#SVM
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn.metrics import accuracy_score, mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.2,
random_state=42)
model = SVC(kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Accuracy: {acc}')
```

4.3 LOGISTIC REGRESSION

Logistic Regression, a widely used classification algorithm, plays a crucial role in the code. The concept of Logistic Regression hinges on the notion of binary classification, where data points are assigned to one of two possible classes. In this context, the code leverages Logistic Regression to predict whether patients are either positive or negative for a specific disease based on various attributes.

Logistic Regression's applicability in healthcare analytics is evident. The model is ideal for scenarios where probabilities matter. Instead of simply classifying data into discrete categories. This is especially valuable in healthcare, as it allows healthcare professionals to assess the likelihood of a patient having a disease based on their symptoms, patient profiles, or diagnostic test results.

The core idea behind Logistic Regression is the logistic function, also known as the sigmoid function. This function transforms input data into values between 0 and 1, representing probabilities. If the transformed value surpasses a specific threshold (often 0.5), the data point is classified into one category; otherwise, it falls into the other. This makes Logistic Regression a powerful tool for healthcare practitioners to make informed decisions. By assigning probabilities and labels to patients, it assists in diagnoses and treatment decisions.

In the code, Logistic Regression is used to predict disease outcomes, enabling the categorization of patients into relevant classes. By estimating the probability of patients having the disease, Logistic Regression enhances the precision of predictions.

CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

CODE:

```
import pandas as pd
a=pd.read_csv("/content/Disease_symptom_and_patient_profile_dataset.csv")
print(a)
```

OUTPUT:

```
         Disease Fever Cough Fatigue Difficulty Breathing  Age  Gender
0      Influenza   Yes    No     Yes                  Yes   19  Female
1    Common Cold    No   Yes     Yes                   No   25  Female
2         Eczema    No   Yes     Yes                   No   25  Female
3         Asthma   Yes   Yes      No                  Yes   25    Male
4         Asthma   Yes   Yes      No                  Yes   25    Male
..           ...   ...   ...     ...                  ...  ...     ...
344       Stroke   Yes    No     Yes                   No   80  Female
345       Stroke   Yes    No     Yes                   No   85    Male
346       Stroke   Yes    No     Yes                   No   85    Male
347       Stroke   Yes    No     Yes                   No   90  Female
348       Stroke   Yes    No     Yes                   No   90  Female

    Blood Pressure Cholesterol Level Outcome Variable
0              Low            Normal         Positive
1           Normal            Normal         Negative
2           Normal            Normal         Negative
3           Normal            Normal         Positive
4           Normal            Normal         Positive
..             ...               ...              ...
344           High              High         Positive
345           High              High         Positive
346           High              High         Positive
347           High              High         Positive
348           High              High         Positive

[349 rows x 10 columns]
```

```
print(a.isnull())
```

OUTPUT:

```
     Disease  Fever  Cough  Fatigue  Difficulty Breathing    Age  Gender
0      False  False  False    False                 False  False   False
1      False  False  False    False                 False  False   False
2      False  False  False    False                 False  False   False
3      False  False  False    False                 False  False   False
4      False  False  False    False                 False  False   False
..       ...    ...    ...      ...                   ...    ...     ...
344    False  False  False    False                 False  False   False
345    False  False  False    False                 False  False   False
346    False  False  False    False                 False  False   False
347    False  False  False    False                 False  False   False
348    False  False  False    False                 False  False   False

     Blood Pressure  Cholesterol Level  Outcome Variable
0             False              False             False
1             False              False             False
2             False              False             False
3             False              False             False
4             False              False             False
..              ...                ...               ...
344           False              False             False
345           False              False             False
346           False              False             False
347           False              False             False
348           False              False             False

[349 rows x 10 columns]
```

```
import pandas as pd
c={'True':1,'False':0,'Yes':1,'No':0,'Positive':1,'Negative':0,'Male':1,'Female':0,'l
ow':0,'Normal':0,'High':1}
a=a.replace(c)
dropping=['Disease','Blood Pressure','Cholesterol Level']
d=a.drop(columns=dropping)
print(d)
```

```
     Fever  Cough  Fatigue  Difficulty Breathing  Age  Gender  \
0        1      0        1                     1   19       0
1        0      1        1                     0   25       0
2        0      1        1                     0   25       0
3        1      1        0                     1   25       1
4        1      1        0                     1   25       1
..     ...    ...      ...                   ...  ...     ...
344      1      0        1                     0   80       0
345      1      0        1                     0   85       1
346      1      0        1                     0   85       1
347      1      0        1                     0   90       0
348      1      0        1                     0   90       0

     Outcome Variable
0                   1
1                   0
2                   0
3                   1
4                   1
..                ...
344                 1
345                 1
346                 1
347                 1
348                 1

[349 rows x 7 columns]
```

```
print(a.head())
```

OUTPUT:

```
          Disease  Fever  Cough  Fatigue  Difficulty Breathing  Age  Gender
0       Influenza      1      0        1                     1   19       0
1     Common Cold      0      1        1                     0   25       0
2          Eczema      0      1        1                     0   25       0
3          Asthma      1      1        0                     1   25       1
4          Asthma      1      1        0                     1   25       1

  Blood Pressure  Cholesterol Level  Outcome Variable
0            Low                  0                 1
1              0                  0                 0
2              0                  0                 0
3              0                  0                 1
4              0                  0                 1
```

```
print(a.tail())
```

OUTPUT:

	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender
344	Stroke	1	0	1	0	80	0
345	Stroke	1	0	1	0	85	1
346	Stroke	1	0	1	0	85	1
347	Stroke	1	0	1	0	90	0
348	Stroke	1	0	1	0	90	0

	Blood Pressure	Cholesterol Level	Outcome Variable
344	1	1	1
345	1	1	1
346	1	1	1
347	1	1	1
348	1	1	1

```
print(a.describe())
```

OUTPUT:

	Fever	Cough	Fatigue	Difficulty Breathing	Age
count	349.000000	349.000000	349.00000	349.000000	349.000000
mean	0.501433	0.478510	0.69341	0.252149	46.323782
std	0.500716	0.500255	0.46174	0.434870	13.085090
min	0.000000	0.000000	0.00000	0.000000	19.000000
25%	0.000000	0.000000	0.00000	0.000000	35.000000
50%	1.000000	0.000000	1.00000	0.000000	45.000000
75%	1.000000	1.000000	1.00000	1.000000	55.000000
max	1.000000	1.000000	1.00000	1.000000	90.000000

	Gender	Outcome Variable
count	349.000000	349.000000
mean	0.495702	0.532951
std	0.500699	0.499629
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	1.000000
75%	1.000000	1.000000
max	1.000000	1.000000

```
print("data information:")
print(a.info())
```

OUTPUT:

```
data information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 349 entries, 0 to 348
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Disease                349 non-null    object
 1   Fever                  349 non-null    int64
 2   Cough                  349 non-null    int64
 3   Fatigue                349 non-null    int64
 4   Difficulty Breathing   349 non-null    int64
 5   Age                    349 non-null    int64
 6   Gender                 349 non-null    int64
 7   Blood Pressure         349 non-null    object
 8   Cholesterol Level      349 non-null    object
 9   Outcome Variable       349 non-null    int64
dtypes: int64(7), object(3)
memory usage: 27.4+ KB
None
```

```
import numpy as np
from pandas._libs.tslibs.timedeltas import array_to_timedelta64
arr1=np.array(a['Disease'])
print('data1:',arr1.dtype)
arr2=np.array(a['Fever'])
print('data2:',arr2.dtype)
arr3=np.array(a['Cough'])
print('data3:',arr3.dtype)
arr4=np.array(a['Fatigue'])
print('data4:',arr4.dtype)
arr5=np.array(a['Difficulty Breathing'])
print('data5:',arr5.dtype)
arr6=np.array(a['Age'])
print('data6:',arr6.dtype)
arr7=np.array(a['Gender'])
print('data7:',arr7.dtype)
arr8=np.array(a['Blood Pressure'])
print('data8:',arr8.dtype)
arr9=np.array(a['Cholesterol Level'])
print('data9:',arr9.dtype)
arr10=np.array(a['Outcome Variable'])
print('data10:',arr10.dtype)
```

OUTPUT:

```
data1: object
data2: int64
data3: int64
data4: int64
data5: int64
data6: int64
data7: int64
data8: object
data9: object
data10: int64
```

```
arr1=(a['Disease'])
print('data1:',arr1.shape)
arr2=(a['Fever'])
print('data2:',arr2.shape)
arr3=(a['Cough'])
print('data3:',arr3.shape)
arr4=(a['Fatigue'])
print('data4:',arr4.shape)
arr5=(a['Difficulty Breathing'])
print('data5:',arr5.shape)
arr6=(a['Age'])
print('data6:',arr6.shape)
arr7=(a['Gender'])
print('data7:',arr7.shape)
arr8=(a['Blood Pressure'])
print('data8:',arr8.shape)
arr9=(a['Cholesterol Level'])
print('data9:',arr9.shape)
arr10=(a['Outcome Variable'])
print('data10:',arr10.shape)
```

OUTPUT:

```
data1: (349,)
data2: (349,)
data3: (349,)
data4: (349,)
data5: (349,)
data6: (349,)
data7: (349,)
data8: (349,)
data9: (349,)
data10: (349,)
```

```
y=d["Outcome Variable"]
print(y)
```

OUTPUT:

```
0      1
1      0
2      0
3      1
4      1
      ..
344    1
345    1
346    1
347    1
348    1
Name: Outcome Variable, Length: 349, dtype: int64
```

```
x=d.drop("Outcome Variable", axis=1)
print(x)
```

OUTPUT:

	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender
0	1	0	1	1	19	0
1	0	1	1	0	25	0
2	0	1	1	0	25	0
3	1	1	0	1	25	1
4	1	1	0	1	25	1
..	...	...	...	...	...	...
344	1	0	1	0	80	0
345	1	0	1	0	85	1
346	1	0	1	0	85	1
347	1	0	1	0	90	0
348	1	0	1	0	90	0

[349 rows x 6 columns]

```
age_column = a['Age']
normalized_age = (age_column - age_column.min()) / (age_column.max() -
age_column.min())
a['Age'] = normalized_age
print(a['Age'])
```

OUTPUT:

```
0      0.000000
1      0.084507
2      0.084507
3      0.084507
4      0.084507
         ...
344    0.859155
345    0.929577
346    0.929577
347    1.000000
348    1.000000
Name: Age, Length: 349, dtype: float64
```

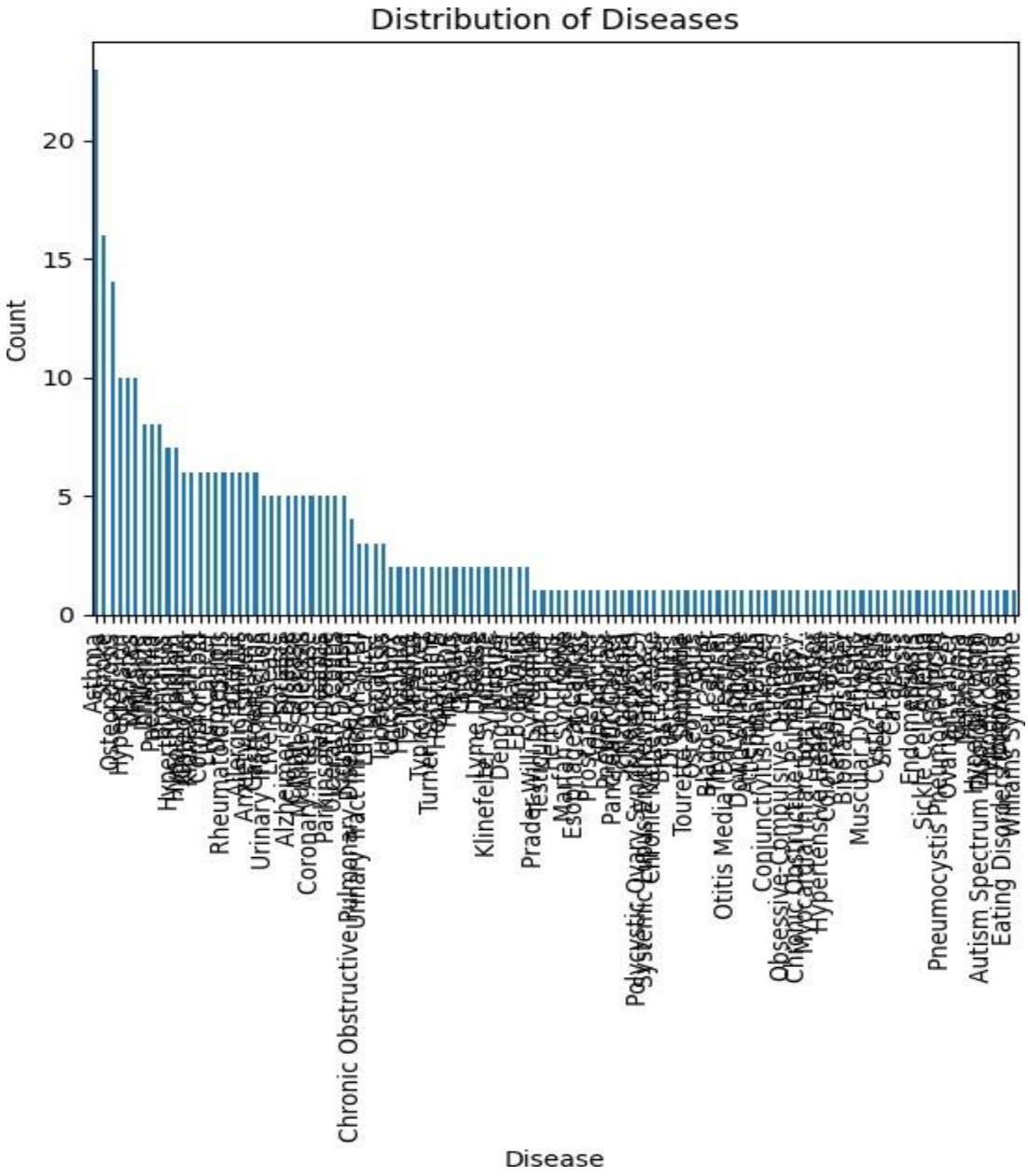
a. dtypes

OUTPUT:

```
Disease                 object
Fever                    int64
Cough                    int64
Fatigue                  int64
Difficulty Breathing     int64
Age                    float64
Gender                   int64
Blood Pressure          object
Cholesterol Level       object
Outcome Variable         int64
dtype: object
```

```
import matplotlib.pyplot as plt

# Assuming 'Disease' is the column you want to visualize
disease_counts = a['Disease'].value_counts()
disease_counts.plot(kind='bar')
plt.xlabel('Disease')
plt.ylabel('Count')
plt.title('Distribution of Diseases')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.show()
```

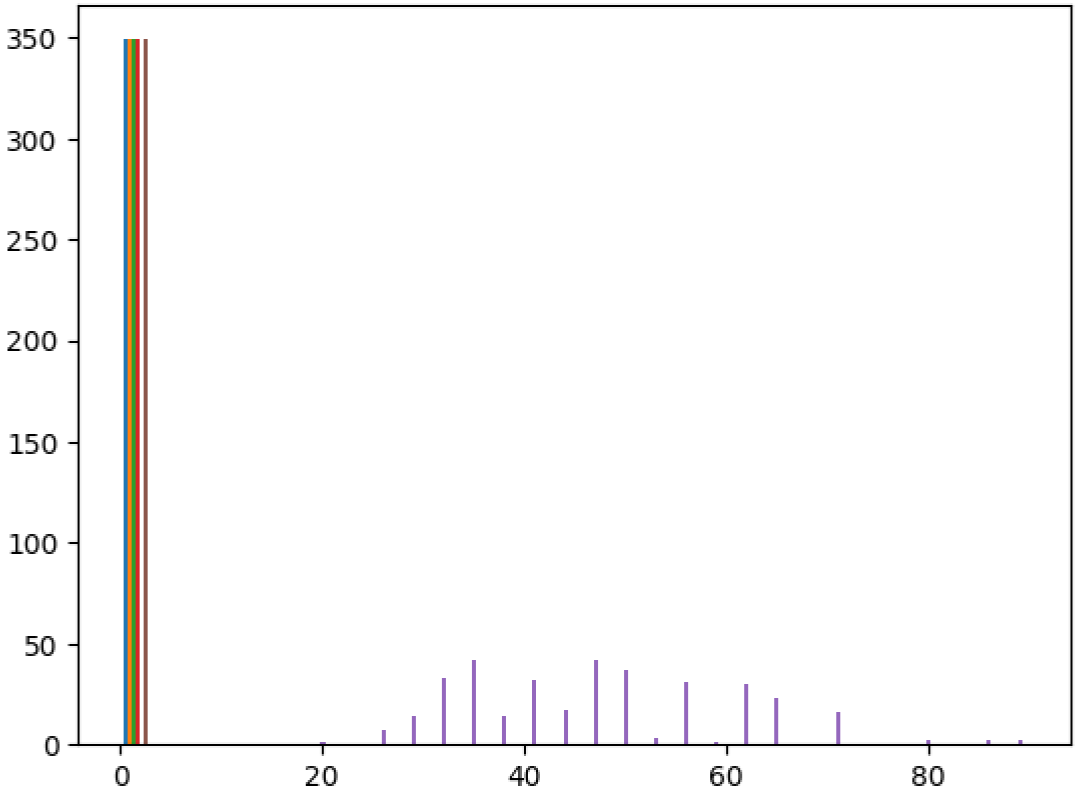
OUTPUT:




```
plt.hist(x,30)
```

OUTPUT:

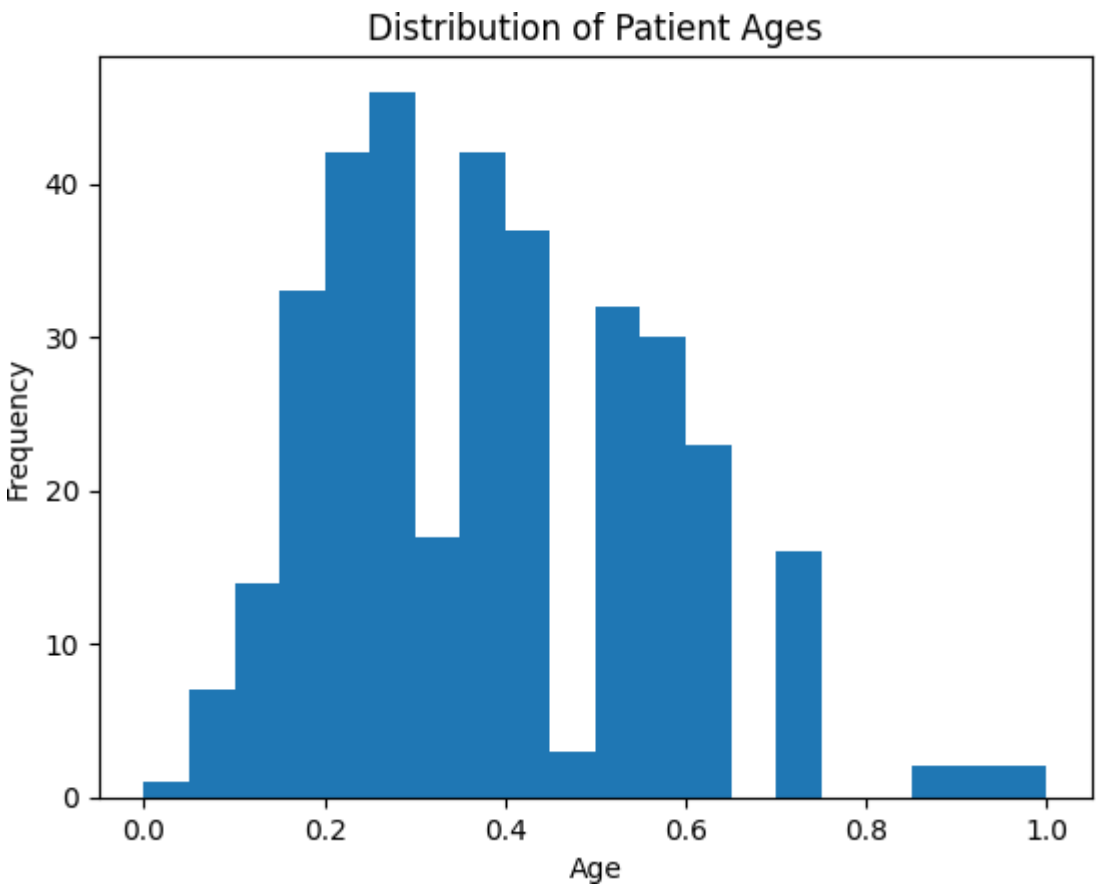
```
(array([[349.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [349.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [349.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [349.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   7.,  14.,  33.,
          42.,  14.,  32.,  17.,  42.,  37.,   3.,  31.,   1.,  30.,  23.,
           0.,  16.,   0.,   0.,   2.,   0.,   2.,   2.],
        [349.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.]]),
 array([ 0.,  3.,  6.,  9., 12., 15., 18., 21., 24., 27., 30., 33., 36.,
        39., 42., 45., 48., 51., 54., 57., 60., 63., 66., 69., 72., 75.,
        78., 81., 84., 87., 90.]),
 <a list of 6 BarContainer objects>)
```




```
# Assuming 'Age' is the column you want to visualize
plt.hist(a['Age'], bins=20)  # You can adjust the number of bins
```

```
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Patient Ages')
plt.show()
```

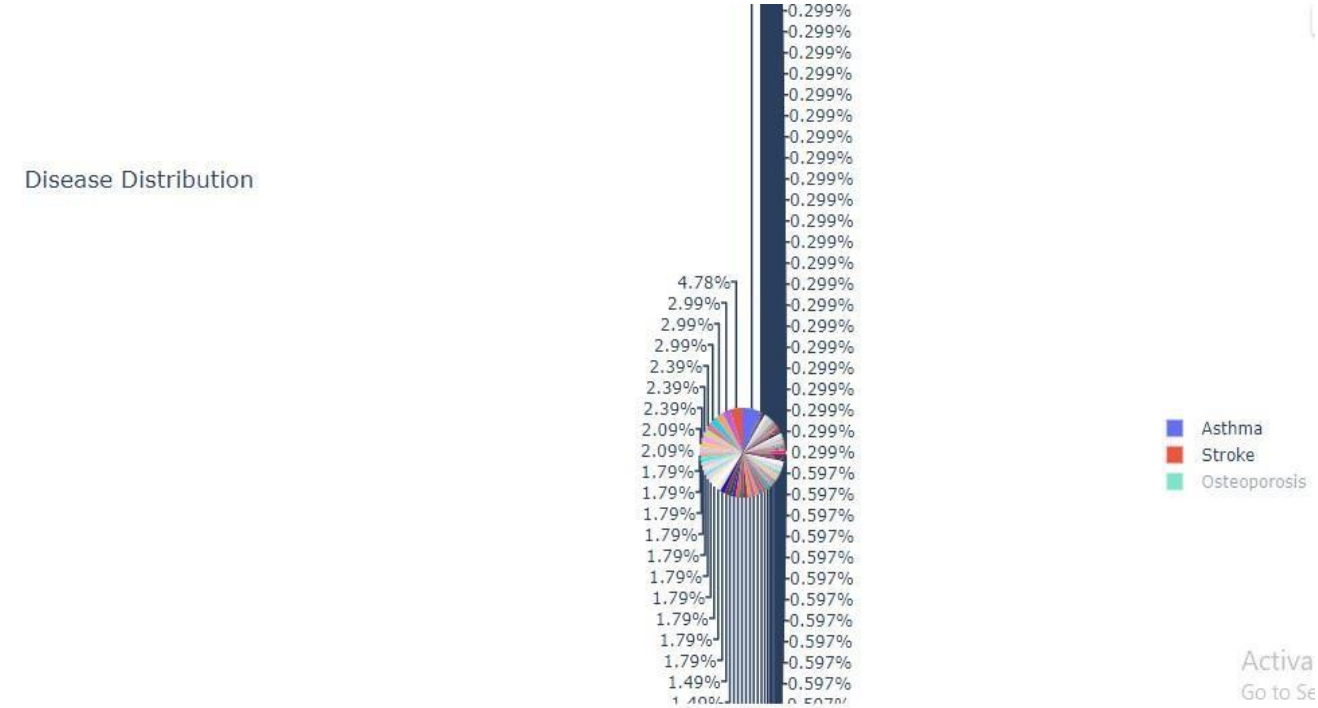
OUTPUT:




```
import plotly.express as px

# Assuming 'Disease' is a column in the 'a' DataFrame
disease_counts = a['Disease'].value_counts().reset_index()
disease_counts.columns = ['Disease', 'Count']
```

```
# Create a pie chart using Plotly Express
fig = px.pie(disease_counts, values='Count', names='Disease', title='Disease
Distribution')
fig.show()
```

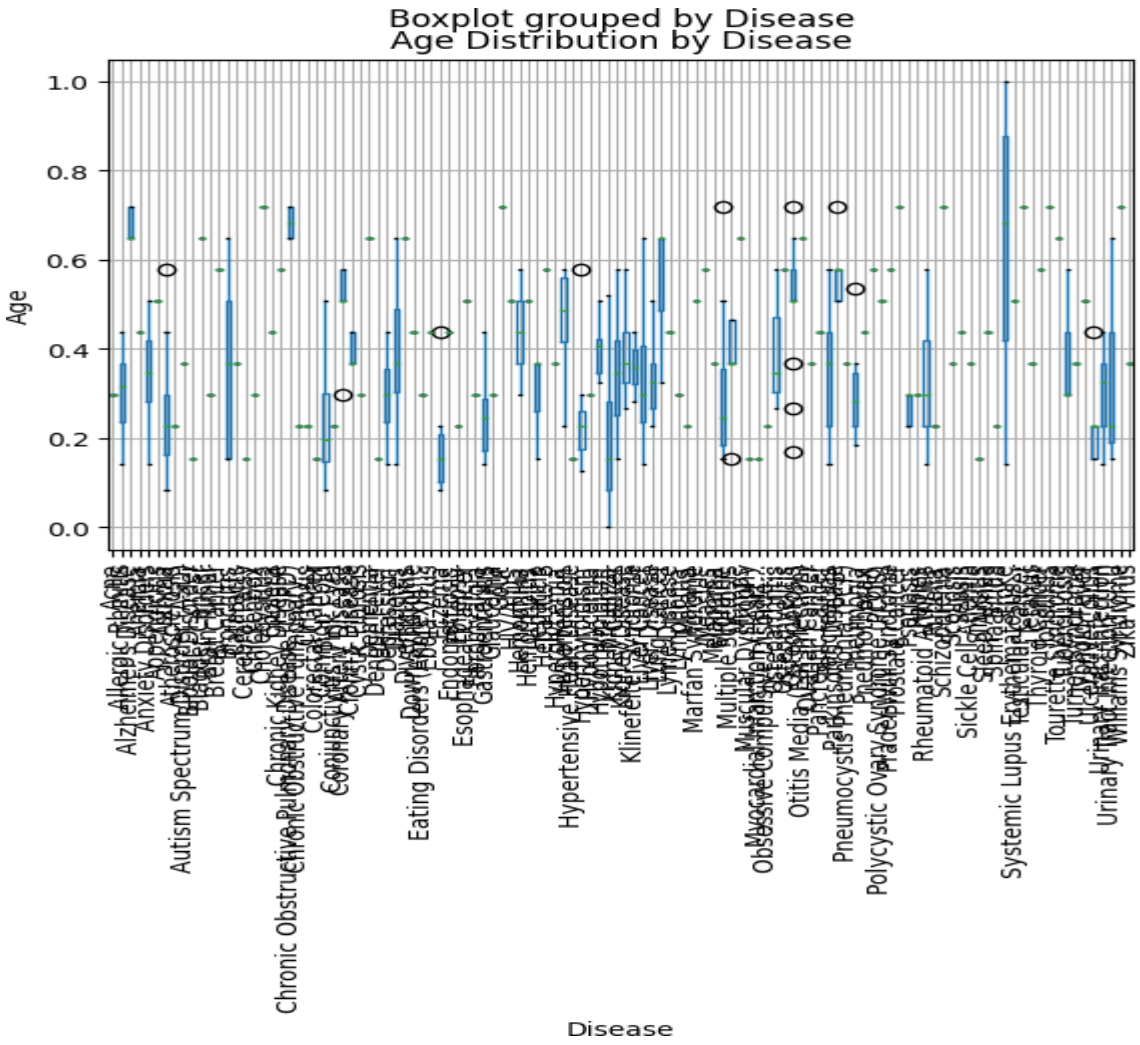
OUTPUT:




```
#Assuming 'Age' and 'Disease' are the columns you want to visualize
plt.figure(figsize=(100, 160))  # Adjust the figure size as needed
a.boxplot(column='Age', by='Disease')
plt.xlabel('Disease')
plt.ylabel('Age')
plt.title('Age Distribution by Disease')
plt.xticks(rotation=90)
plt.show()
```

OUTPUT:

```
<Figure size 10000x16000 with 0 Axes>
```




```
from sklearn.model_selection import train_test_split
X_test, X_train = train_test_split(a, test_size=0.2, random_state=100)
print("train set")
print(X_train.shape)
print(X_train)
print("\ntest")
print(X_test)
print(X_test.shape)
```

OUTPUT:

```
train set
(70, 10)
                Disease  Fever  Cough  Fatigue  Difficulty Breathing  \
197           Pneumonia      1      1        1                     1
189        Hypertension      1      0        1                     0
7             Influenza      1      1        1                     1
46            Influenza      1      1        1                     1
136            Lymphoma      0      1        0                     0
..                  ...    ...    ...      ...                   ...
31           Sinusitis      0      1        1                     0
178         Zika Virus      0      1        1                     0
247  Esophageal Cancer      0      0        1                     0
47   Multiple Sclerosis      0      0        1                     0
118     Osteoarthritis      0      0        1                     0

          Age  Gender Blood Pressure Cholesterol Level  Outcome Variable
197  0.366197       1              1                 1                 1
189  0.366197       1              0                 1                 1
7    0.084507       0              0                 0                 1
46   0.154930       0              0                 0                 1
136  0.295775       0              0                 1                 1
..        ...     ...            ...               ...               ...
31   0.154930       1              0                 0                 0
178  0.366197       0              1                 1                 0
247  0.507042       1              0                 0                 0
47   0.154930       0              1                 1                 1
118  0.295775       1              1                 0                 0

[70 rows x 10 columns]

test
```

```
test
                  Disease  Fever  Cough  Fatigue  Difficulty Breathing  \
231  Klinefelter Syndrome      0      0        1                     0
311   Alzheimer's Disease      0      1        0                     0
23          Dengue Fever      1      0        1                     0
248             HIV/AIDS      1      0        0                     0
15        Gastroenteritis      0      1        0                     0
..                    ...    ...    ...      ...                   ...
79             Depression      0      0        1                     0
343                Stroke      1      0        1                     0
323                 Mumps      0      0        1                     0
280        Osteoarthritis      0      1        0                     1
8         Hyperthyroidism      0      1        0                     0

          Age  Gender Blood Pressure Cholesterol Level  Outcome Variable
231  0.436620       0              0                 0                 1
311  0.647887       1              0                 1                 1
23   0.154930       0              0                 0                 0
248  0.507042       0              1                 1                 0
15   0.140845       0              0                 0                 0
..        ...     ...            ...               ...               ...
79   0.225352       0              0                 1                 1
343  0.859155       0              1                 1                 1
323  0.647887       1              0                 1                 1
280  0.577465       0              0                 0                 0
8    0.126761       0              0                 0                 0

[279 rows x 10 columns]
(279, 10)
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

OUTPUT:

```
Accuracy: 0.5714285714285714
```

```
#SVM
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn.metrics import accuracy_score, mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.2,
random_state=42)
model = SVC(kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Accuracy: {acc}')
```

OUTPUT:

```
Accuracy: 0.5571428571428572
```

```
import numpy as np
x1 = np.random.rand(10, 15)
w=[-0.5,-0.2,-0.7,0.9,-0.5,0.6,-0.9,0.7,-0.8,-0.3,-0.8,0.1,-0.7,0.2,-0.6]

def perceptron(w, x1, b):
    yp = []
    for i in range(len(x1)):
        s = np.dot(x1[i], w) + b
        s2 = 1 / (1 + np.exp(-s))
        if s2 >= 0.5:
            yp.append(1)
```

```
        else:
            yp.append(0)
    return yp
b= float(input("enter b:"))
predicted_values = perceptron(w, x1, b)
print(predicted_values)
```

OUTPUT:

```
enter b:4
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", cm)
```

OUTPUT:

```
[21]
     Confusion Matrix :
      [[16 14]
      [16 24]]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
recall_score, f1_score
# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy: 0.5714285714285714
```

```
# Calculate and print precision
precision = precision_score(y_test, y_pred, average='weighted',zero_division=0)
print("Precision:", precision)
```

OUTPUT:

```
Precision: 0.575187969924812
```

```
# Calculate and print recall
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)
```

OUTPUT:

```
Recall: 0.5714285714285714
```

```
# Calculate and print F1-score
f1 = f1_score(y_test, y_pred, average='weighted')
print("F1-score:", f1)
```

OUTPUT:

```
F1-score: 0.5728465083303794
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Generate some sample data
np.random.seed(0)
x = np.random.rand(100, 15)
y = np.random.randint(2, size=100)
```

```
# Perceptron function
def perceptron(w, x1, b):
    yp = []
    for i in range(len(x1)):
        s = np.dot(x1[i], w) + b
        s2 = 1 / (1 + np.exp(-s))
        if s2 >= 0.5:
            yp.append(1)
        else:
            yp.append(0)
    return yp
```

```
b = float(input("Enter b:"))
predicted_values_perceptron = perceptron(w, x, b)
```

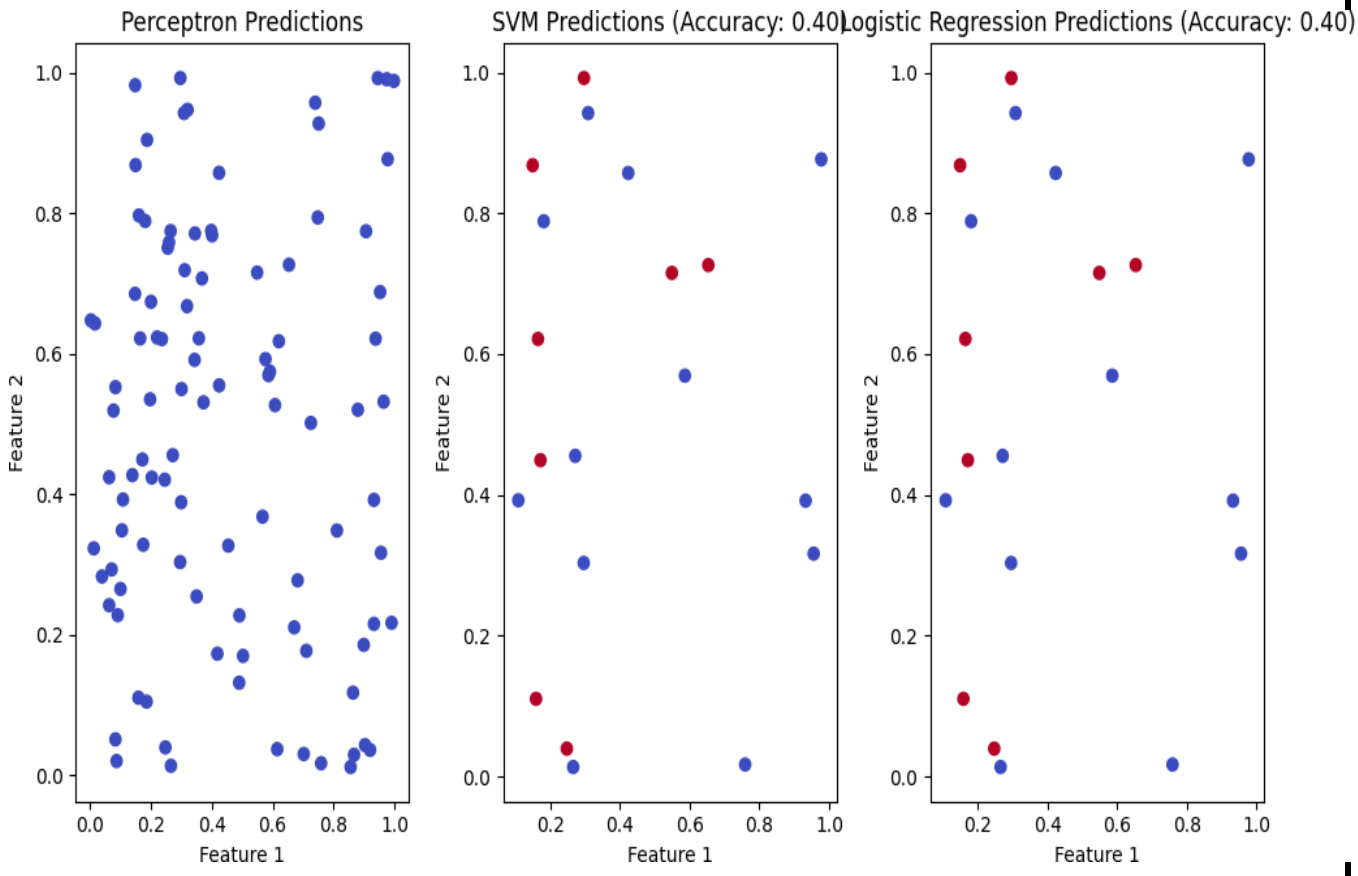
```
# Split data for SVM and Logistic Regression
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

```
# SVM
model_svm = SVC(kernel='linear')
model_svm.fit(X_train, y_train)
```

```
y_pred_svm = model_svm.predict(X_test)
acc_svm = accuracy_score(y_test, y_pred_svm)

# Logistic Regression
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)
acc_lr = accuracy_score(y_test, y_pred_lr)

# Plot the results
plt.figure(figsize=(10, 6))

# Perceptron
plt.subplot(131)
plt.scatter(x[:, 0], x[:, 1], c=predicted_values_perceptron, cmap='coolwarm')
plt.title('Perceptron Predictions')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# SVM
plt.subplot(132)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_svm, cmap='coolwarm')
plt.title(f'SVM Predictions (Accuracy: {acc_svm:.2f})')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Logistic Regression
plt.subplot(133)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_lr, cmap='coolwarm')
plt.title(f'Logistic Regression Predictions (Accuracy: {acc_lr:.2f})')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.tight_layout()
plt.show()
```

OUTPUT:

```
Enter b:10
```



CONCLUSION

In summary, our project successfully employed machine learning techniques to predict disease symptoms and analyze patient profiles, aiming to improve healthcare outcomes. We used classification methods like Support Vector Machine (SVM) and Logistic Regression to categorize patients based on their symptoms, facilitating early disease detection and treatment. Data pre-processing, model evaluation, and real-world applications were key components of the project. As we look forward, there's potential for advanced models, more extensive datasets, and collaboration with healthcare experts to refine clinical applications. In the ever-advancing field of healthcare technology, machine learning plays a vital role in enhancing patient care and outcomes.

REFERENCE

1. UCI Machine Learning Repository: A vast collection of datasets for various machine learning tasks. UCI ML Repository

2. Kaggle Datasets: Kaggle hosts a wide range of datasets for data science and machine learning projects. Kaggle Datasets

3. OpenML: A platform with a large collection of datasets and an API for machine learning. OpenML

4. World Bank Data: Datasets related to global development, economics, and more. World Bank Data

5. IMDb Datasets: Movie-related datasets for natural language processing and recommendation systems. IMDb Datasets

6. Scikit-Learn GitHub: Source code for the popular Scikit-Learn machine learning library. Scikit-Learn GitHub

7. TensorFlow GitHub: Open-source machine learning framework by Google. TensorFlow GitHub

8. MLflow GitHub: An open-source platform for managing the end-to-end machine learning lifecycle. MLflow GitHub

9. Google Colab:https://colab.research.google.com/

INDEX

LINKS

1. LAB-01
2. LAB-02
3. LAB-03
4. LAB-04
5. LAB-05
6. LAB-06
7. LAB-07
8. LAB-08
9. LAB-09
10. LAB-10

LAB-01

DESCRIPTION:

Exposing to various frameworks of Statistical Machine Learning-Numpy,Pandas,Matplotlib,Seaborn.

PROBLEM STATEMENT:

Introduce students to essential StatML frameworks: Numpy, Pandas, Matplotlib, Seaborn, Tensorflow, and Keras. Tasks include understanding Numpy for numerical operations, Pandas for data manipulation, Matplotlib/Seaborn for visualization, and building a basic neural network using Tensorflow/Keras. Evaluation is based on understanding, code quality, and practical application.

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

```
# Create a Python list
list=[1,2,3]

# Create a NumPy array from the list
array1=np.array(list)

# Display the NumPy array
array1
```

OUTPUT:

```
array([1, 2, 3])
```

```
# Create an array of 7 zeroes
print("A series of zeroes:",np.zeros(7))

# Create an array of 9 ones
print("A series of ones:",np.ones(9))

# Create an array with numbers from 5 to 15 (exclusive)
print("A series of numbers:",np.arange(5,16))
```

```
# Create an array with numbers from 0 to 10 with a step of 2
print("Numbers spaced apart by 2:",np.arange(0,11,2))

#Create an array with numbers from 0 to 10 with a step of 2.5
print("Numbers spaced apart by float:",np.arange(0,11,2.5))



# Create an array with every 5th number from 30 to 0 in reverse order
print("Every 5th number from 30 in reverse order: ",np.arange(30,-1,-
5))

# Create an array of 11 linearly spaced numbers between 1 and 5
print("11 linearly spaced numbers between 1 and 5",np.linspace(1,5,11))
```

OUTPUT:

```
A series of zeroes: [0. 0. 0. 0. 0. 0. 0.]
A series of ones: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
A series of numbers: [ 5  6  7  8  9 10 11 12 13 14 15]
Numbers spaced apart by 2: [ 0  2  4  6  8 10]
Numbers spaced apart by float: [ 0.   2.5  5.   7.5 10. ]
Every 5th number from 30 in reverse order:  [30 25 20 15 10  5  0]
11 linearly spaced numbers between 1 and 5:  [1.  1.4 1.8 2.2 2.6 3.  3.4 3.8 4.2 4.6 5. ]
```

```
import pandas as pd

# Read the data from the CSV file "wine.csv" and store it in a
DataFrame 'df'
df=pd.read_csv("wine.csv")

# Display the first few rows of the DataFrame to inspect the data
df.head()
```

OUTPUT:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
# Read the data from the CSV file "Book1.csv" and store it in a
DataFrame 'studdata'
studdata=pd.read_csv("Book1.csv")

#Display the DataFrame 'studdata' to view the data
studdata
```

OUTPUT:

	name	locn	marks
0	a	gfs	5
1	b	gsdf	3

```
# Read the data from the Excel file "Height_weight.xlsx" and store it
in a DataFrame 'dataxls'
dataxls=pd.read_excel("Height_weight.xlsx")

# Display the DataFrame 'dataxls' to view the data
dataxls
```

OUTPUT:

	Name	Height	Weight
0	Ashton	155	135
1	Kate	125	140
2	Bruce	178	210
3	Tom	181	165
4	Bill	165	180

```
# Read tables from the specified Wikipedia page and store them in a
list of DataFrames
list_of_df=pd.read_html("https://en.wikipedia.org/wiki/2016_Summer_Olym
pics_medal_table",header=0)

# Assuming the desired table is the first one in the list, store it in
a DataFrame 'medals'
medals=list_of_df[0]

# Display the DataFrame 'medals' to view the table
medals
```

OUTPUT:

	2016 Summer Olympics medals	2016 Summer Olympics medals.1	Unnamed: 2
0	Location	Rio de Janeiro, Brazil	NaN
1	Highlights	Highlights	NaN
2	Most gold medals	United States (46)	NaN
3	Most total medals	United States (121)	NaN
4	← 2012 · Olympics medal tables · 2020 →	← 2012 · Olympics medal tables · 2020 →	NaN
5	← 2012 ·	Olympics medal tables	· 2020 →

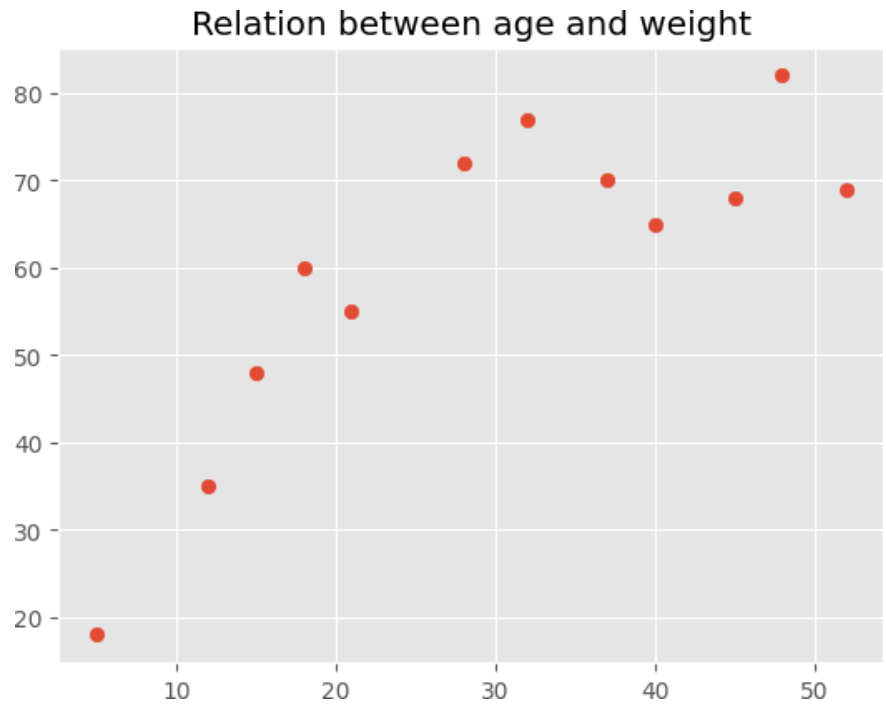
```
import matplotlib.pyplot as plt
```

```
# Lists of people, age, weight, and height data
people =
['Ann','Brandon','Chen','David','Emily','Farook','Gagan','Hamish','Imra
n','Julio','Katherine','Lily']
age = [21,12,32,45,37,18,28,52,5,40,48,15]
weight = [55,35,77,68,70,60,72,69,18,65,82,48]
height = [160,135,170,165,173,168,175,159,105,171,155,158]
```

```
# Create a scatter plot to visualize the relationship between age and
weight
plt.scatter(age,weight)

plt.title("Relation between age and weight")

# Display the plot
```

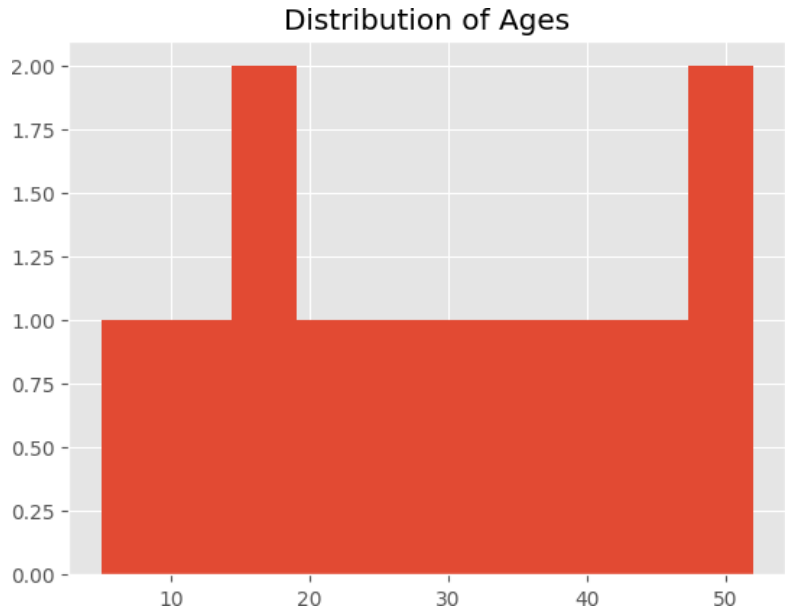
```
plt.show()
```

OUTPUT:



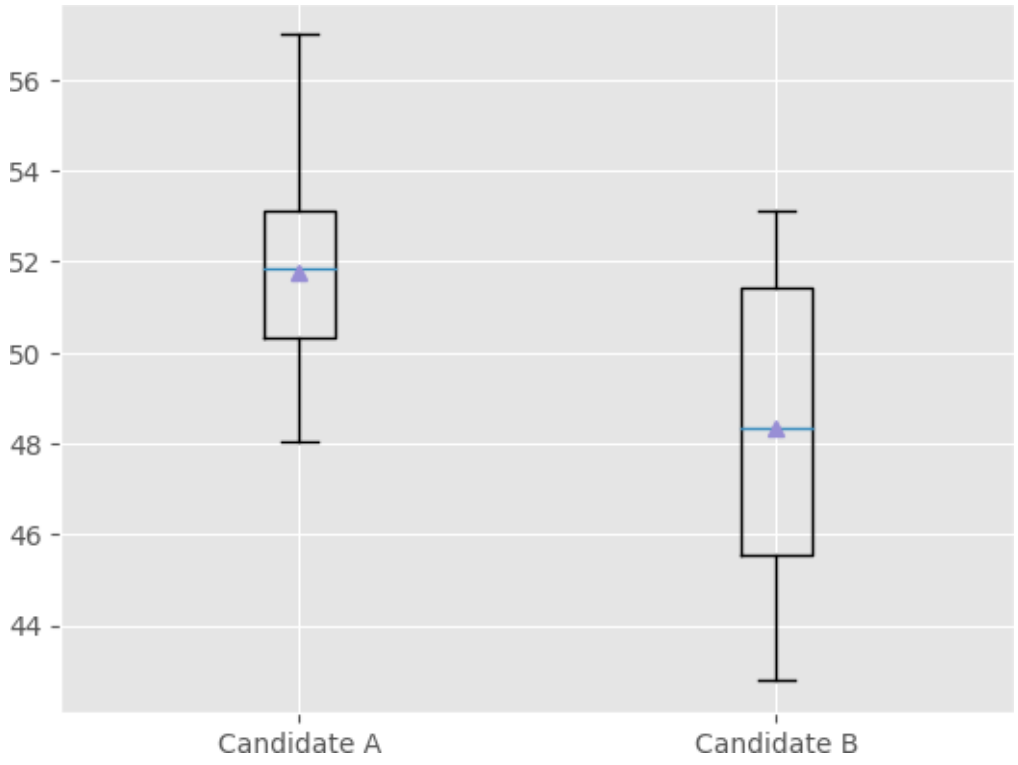

```
# Create a histogram to visualize the distribution of ages
plt.hist(age)

plt.title("Distribution of Ages")

# Display the histogram
plt.show()
```

OUTPUT:




```
# Generate an array of days from 1 to 30
days=np.arange(1,31)

#Simulate data for Candidate A and Candidate B with noise
candidate_A=50+days*0.07+2*np.random.randn(30)
candidate_B=50-days*0.1+3*np.random.randn(30)

# Set the plotting style to 'ggplot'
plt.style.use('ggplot')

# Create a box plot for Candidate A and Candidate B, showing means
plt.boxplot(x=[candidate_A,candidate_B],showmeans=True)

# Add a grid to the plot
plt.grid(True)

# Set custom labels for the x-axis ticks
plt.xticks([1,2],['Candidate A','Candidate B'])

# Display the plot
plt.show()
```

OUTPUT:


56
54
52
50
48
46
44
Candidate A
Candidate B

LAB-02

DESCRIPTION:

Reading data and identifying variables ,finding maximum likelihood values ,density estimation.

PROBLEM STATEMENT:

In Lab02, students are tasked with reading a dataset, identifying variables, and calculating maximum likelihood values for specific parameters. They should also perform density estimation to analyze data distributions. This lab aims to develop their skills in data exploration, statistical analysis, and parameter estimation.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import multivariate_normal
%matplotlib inline
```

```
# Define the covariance matrix
covariance = np.array([[0.14, -0.3, 0.0, 0.2],
                       [-0.3, 1.16, 0.2, -0.8],
                       [0.0, 0.2, 1.0, 1.0],
                       [0.2, -0.8, 1.0, 2.0]])

# Calculate the precision matrix by taking the inverse of the
covariance matrix
precision = np.linalg.inv(covariance)

# Print the precision matrix
print(precision)
```

OUTPUT:

```
[[ 60.   50.  -48.   38. ]
 [ 50.   50.  -50.   40. ]
 [-48.  -50.   52.4 -41.4]
 [ 38.   40.  -41.4  33.4]]
```

```
# Define a function to generate a random pair of numbers from a
bivariate normal distribution.
# The mean vector is [0.8, 0.8], and the covariance matrix is [[0.1, -
0.1], [-0.1, 0.12]].
def generate_pair():
   return np.random.multivariate_normal([0.8, 0.8], [[0.1, -0.1],[-0.1,
   0.12]])
# Call the 'generate_pair()' function to generate a random pair of
numbers from a bivariate normal distribution.
mu_t = generate_pair()

#Print the generated pair of numbers
print(mu_t)
```

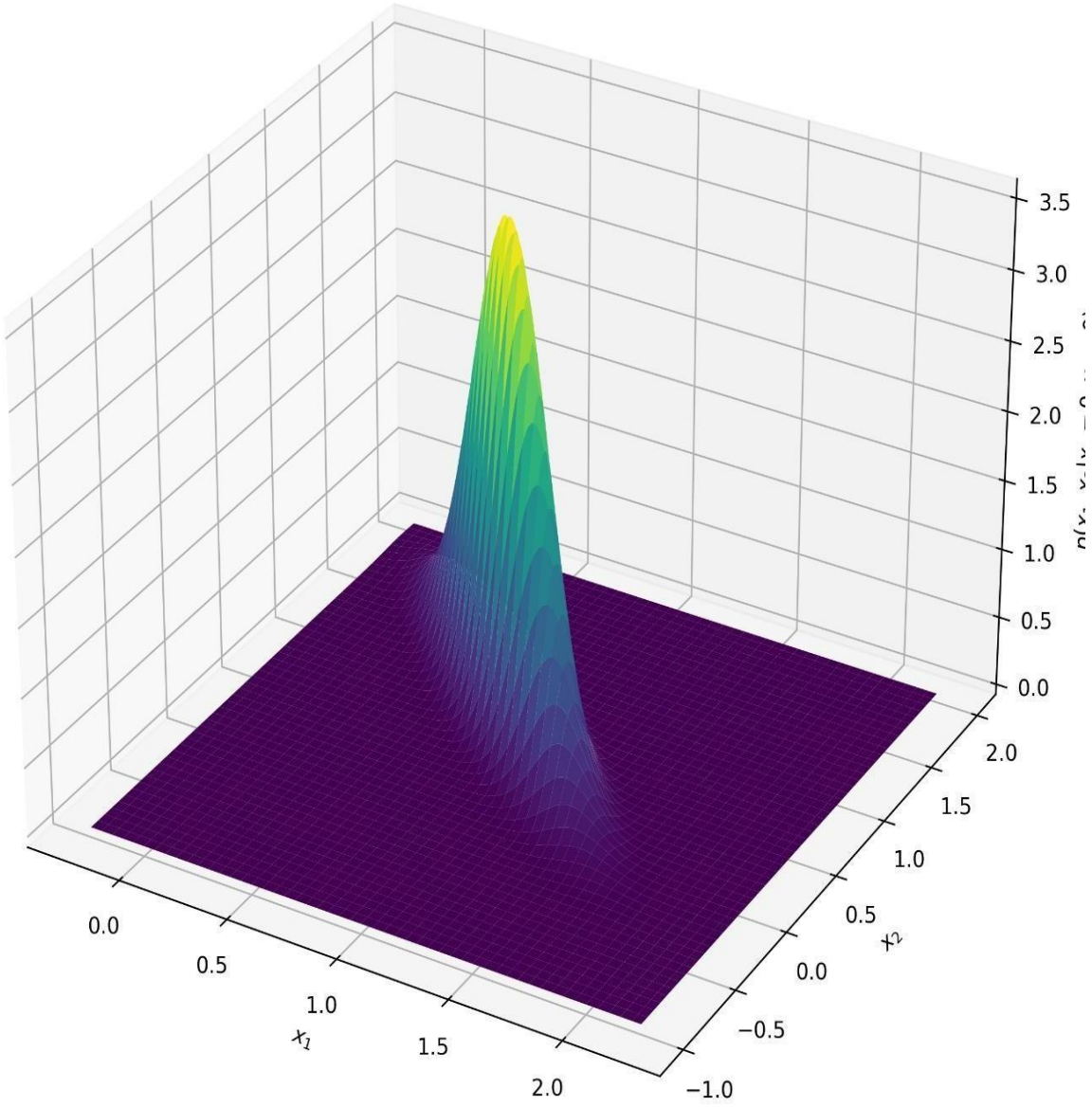
OUTPUT:

```
[0.79116048 0.83708266]
```

```
# Create a grid of points in 2D space
x, y = np.mgrid[-0.25:2.25:.01, -1:2:.01]

# Create a 2D array 'pos' that represents all (x, y) pairs in the grid
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x
pos[:, :, 1] = y

# Define mean (mu_p) and covariance (cov_p) parameters for the
multivariate normal distribution
mu_p = [0.8, 0.8]
cov_p = [[0.1, -0.1], [-0.1, 0.12]]

# Calculate the probability density function (PDF) values for the grid
points
z = multivariate_normal(mu_p, cov_p).pdf(pos)

# Create a 3D plot
fig = plt.figure(figsize=(10, 10), dpi=300)
ax = fig.add_subplot(projection='3d')

# Plot the surface using the PDF values
ax.plot_surface(x, y, z, cmap=plt.cm.viridis)

#Set labels for the axes and z-axis
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
```

```
ax.set_zlabel('$p(x_1, x_2 | x_3=0, x_4=0)$')

# Save the plot as an image file
plt.savefig('cond_mvg.png', bbox_inches='tight', dpi=300)

# Show the 3D plot
plt.show()
```

OUTPUT:



```
# Define the number of data points to generate
N=1000


import numpy as np

# Generate random data points from a bivariate normal distribution
data = np.random.multivariate_normal([0.28, 1.18], [[2.0, 0.8], [0.8,
4.0]], N)
data

# Save the generated data to a text file named 'data.txt
np.savetxt('data.txt',data)
```

```
import pandas as pd

# Approach 1: Using Pandas to read data into a DataFrame
data = pd.read_table('data.txt')

# Approach 2: Using NumPy to load data into an array
data = np.loadtxt('data.txt')

# Display the data loaded with NumPy
data
```

OUTPUT:

```
array([[-0.02914316,  4.10325636],
       [ 1.52675737, -1.26908855],
       [-0.32746412,  2.70086334],
       ...,
       [-1.23764257,  4.26342602],
       [ 0.3151201 , -2.00451667],
       [ 2.72409707, -0.1813087 ]])
```

```
# Calculate the sample mean (mu_ml) for each column (axis=0) of the
data
mu_ml = data.mean(axis=0)

# Calculate the sample covariance matrix (cov_ml)
# Note: N is the number of data points
x = data - mu_ml
```

```
cov_ml = np.dot(x.T, x) / N

# Calculate the unbiased sample covariance matrix (cov_ml_unbiased)
# Note: (N - 1) is used to adjust for bias in the sample covariance
cov_ml_unbiased = np.dot(x.T, x) / (N - 1)

# Print the sample mean (mu_ml)
print(mu_ml)

# Print the sample covariance matrix (cov_ml)
print(cov_ml)

# Print the unbiased sample covariance matrix (cov_ml_unbiased)
print(cov_ml_unbiased)
```

OUTPUT:

```
[0.30476875 1.13227942]
[[2.06260366 0.83147283]
 [0.83147283 4.19418267]]
[[2.06466833 0.83230514]
 [0.83230514 4.19838105]]
```

```
import numpy as np

# Define a function for sequential mean estimation
def seq_ml(data):
    # Initialize a list to store the sequence of mean estimates
    mus = [np.array([[0], [0]])]

    # Iterate over the data points
    for i in range(len(data)):
        x_n = data[i].reshape(2, 1) # Reshape the data point as a 2x1
column vector
        mu_n = mus[-1] + (x_n - mus[-1]) / (i + 1) # Update the mean
estimate
        mus.append(mu_n)  # Append the new mean estimate to the list

    # Return the sequence of mean estimates
    return mus
```

```
# Calculate a sequence of mean estimates using the 'seq_ml' function
mus_ml = seq_ml(data)

# Print the final mean estimate
```

```
print(mus_ml[-1])
```

OUTPUT:

```
[[0.30476875]
 [1.13227942]]
```

```
# Define the mean vector 'mu_p'
mu_p = np.array([[0.28], [1.18]])

# Define the covariance matrix 'cov_p'
cov_p = np.array([[0.1, -0.1], [-0.1, 0.12]])

# Define another covariance matrix 'cov_t'
cov_t = np.array([[2.0, 0.8], [0.8, 4.0]])

# Print the values of 'mu_p', 'cov_p', and 'cov_t'
print(mu_p, cov_p, cov_t)
```

OUTPUT:

```
[[0.28]
 [1.18]] [[ 0.1  -0.1 ]
 [-0.1   0.12]] [[2.  0.8]
 [0.8 4. ]]
```

```
import numpy as np

# Define a function for sequential Maximum A Posteriori (MAP)
estimation
def seq_map(data, mu_p, cov_p, cov_t):
    # Initialize lists to store the sequence of mean and covariance
estimates
    mus, covs = [mu_p], [cov_p]

    # Iterate over the data points
    for x in data:
        x_n = x.reshape(2, 1)  # Reshape the data point as a 2x1 column
vector
        cov_n = np.linalg.inv(np.linalg.inv(covs[-1]) +
np.linalg.inv(cov_t))
```

```
        mu_n = cov_n.dot(np.linalg.inv(cov_t).dot(x_n) +
np.linalg.inv(covs[-1]).dot(mus[-1]))

        # Append the new mean and covariance estimates to the lists
        mus.append(mu_n)
        covs.append(cov_n)

    # Return the sequences of mean and covariance estimates
    return mus, covs
```

```
# Calculate sequences of mean and covariance estimates using the
'seq_map' function
mus_map, covs_map = seq_map(data, mu_p, cov_p, cov_t)

# Print the final mean estimate from the 'mus_map' sequence
print(mus_map[-1])
```

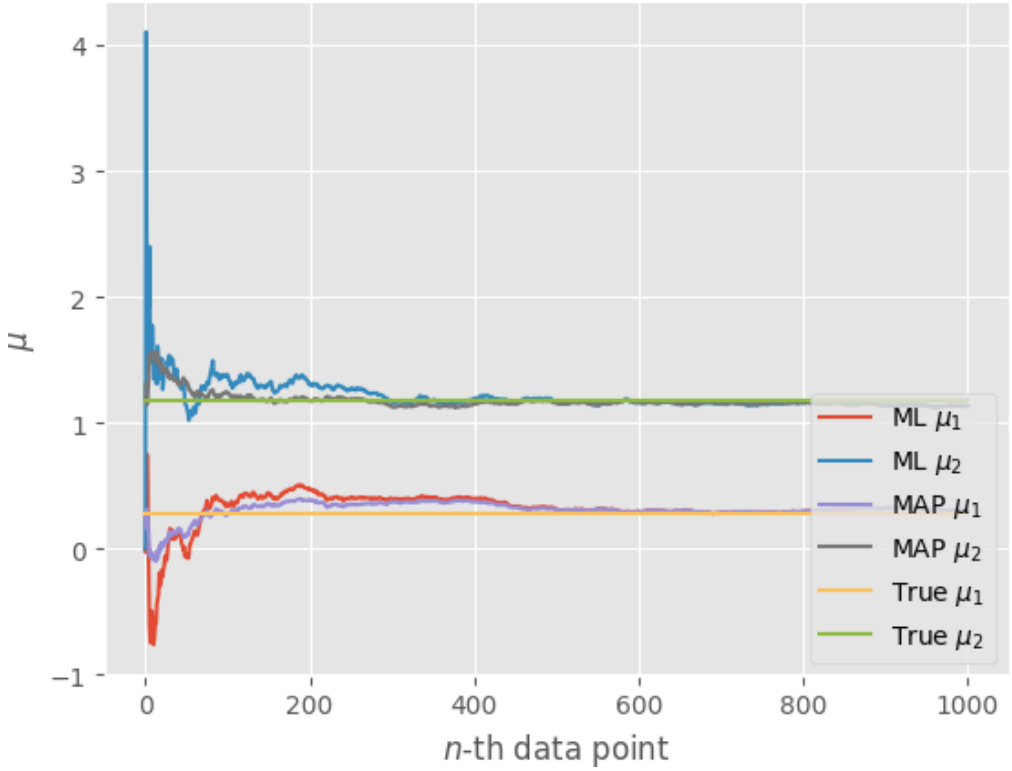
OUTPUT:

```
[[0.30664007]
 [1.13618765]]
```

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

# Create an array of data point indices from 0 to N
X = np.arange(N + 1)

# Extract mean values for ML and MAP estimation from the sequences
mus1_ml = [mu[0] for mu in mus_ml]
mus2_ml = [mu[1] for mu in mus_ml]
mus1_map = [mu[0] for mu in mus_map]
mus2_map = [mu[1] for mu in mus_map]

# Create arrays for the true mean values
mus1_t = [0.28] * (N + 1)
mus2_t = [1.18] * (N + 1)

# Set the plot style
plt.style.use('ggplot')

# Plot the mean estimates and true mean values
plt.plot(X, mus1_ml, label='ML $\mu_1$')
plt.plot(X, mus2_ml, label='ML $\mu_2$')
```

```
plt.plot(X, mus1_map, label='MAP $\mu_1$')
plt.plot(X, mus2_map, label='MAP $\mu_2$')
plt.plot(X, mus1_t, label='True $\mu_1$')
plt.plot(X, mus2_t, label='True $\mu_2$')

# Add labels and legend to the plot
plt.xlabel('$n$-th data point')
plt.ylabel('$\mu$')
plt.legend(loc=4)

# Save the plot as an image file
plt.savefig('seq_learning.png', bbox_inches='tight', dpi=300)


# Show the plot
plt.show()
```

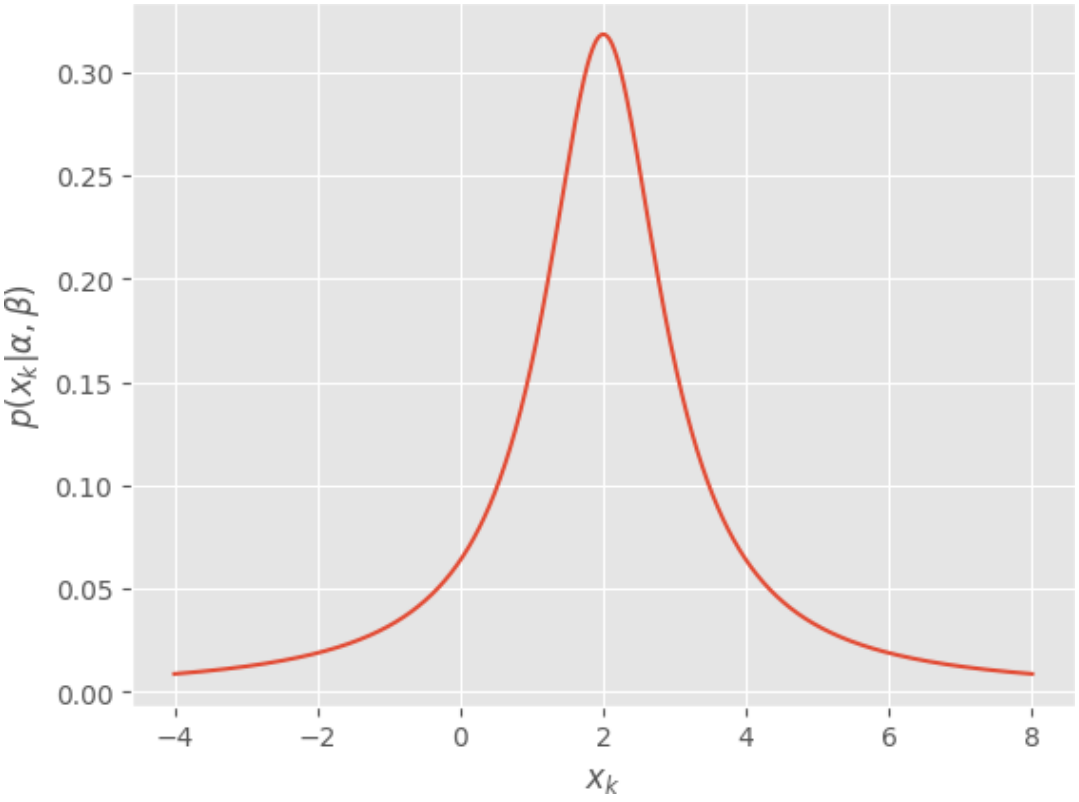
OUTPUT:




```
# Define a function to calculate the PDF of a Cauchy distribution

def p_xk(x, alpha, beta):
    return beta / (np.pi * (beta**2 + (x-alpha)**2))  #b/pi*(b+b + (x-
a)(x-a))
```

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

# Create an array of x values for plotting
x = np.linspace(-4, 8, num=1000)

# Calculate the PDF values for the Cauchy distribution with alpha=2 and
beta=1
probs = p_xk(x, 2, 1)

# Create a plot of the PDF
plt.plot(x, probs)

# Set labels for the x and y axes
plt.xlabel('$x_k$')
plt.ylabel(r'$p(x_k | \alpha, \beta)$')
# Save the plot as an image file
plt.savefig('prob_xk.png', bbox_inches='tight', dpi=300)

# Show the plot
plt.show()
```

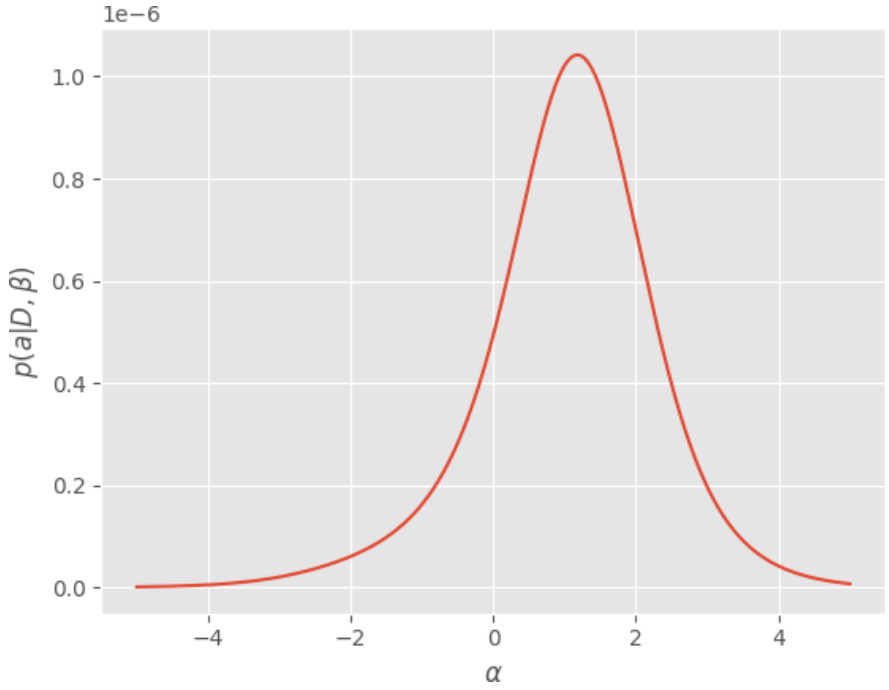
OUTPUT:



```
# Define a function to calculate the joint probability of observing
values x given alpha and beta
def p_a(x, alpha, beta):
    return np.product(beta / (np.pi * beta**2 + (x-alpha)**2))
```

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

# Define the observed data
D = np.array([4.8, -2.7, 2.2, 1.1, 0.8, -7.3])

# Create an array of alpha values for plotting
alphas = np.linspace(-5, 5, num=1000)

# Define the fixed beta parameter
beta = 1

# Calculate the likelihood of observing data D for each alpha in the
alphas array
likelihoods = [p_a(D, alpha, beta) for alpha in alphas]

# Create a plot of the likelihood as a function of alpha
plt.plot(alphas, likelihoods)

# Set labels for the x and y axes
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$p(a | D, \beta)$')

# Save the plot as an image file
plt.savefig('prob_a.png', bbox_inches='tight', dpi=300)

# Show the plot
plt.show()
```

OUTPUT:




```
# Calculate and print the mean of the observed data 'D'
print(D.mean())
print(alphas[np.argmax(likelihoods)])
```

OUTPUT:

```
-0.18333333333333326
1.1761761761761758
```

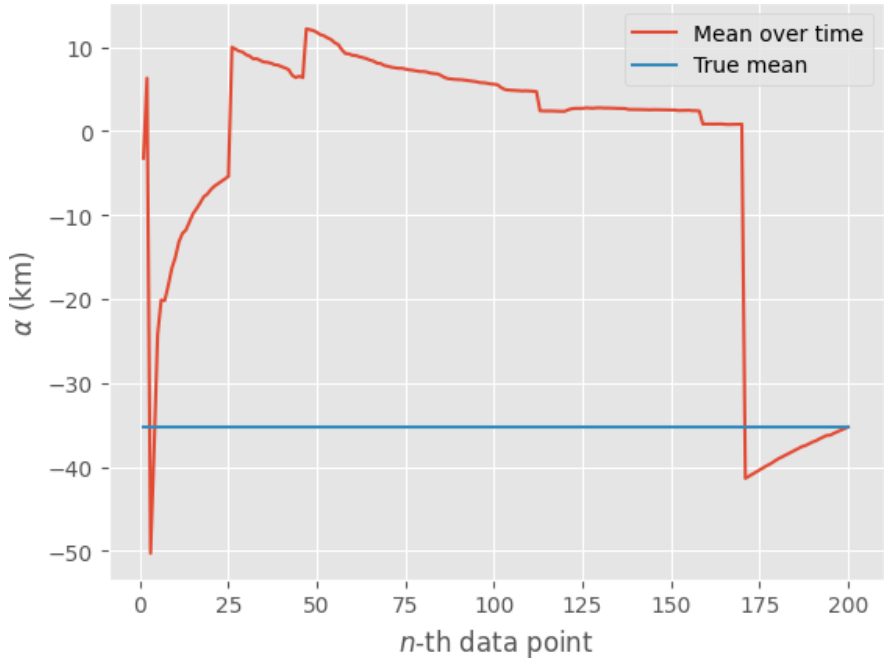
```
import numpy as np

# Generate a random value for 'alpha_t' within the range [0, 10)
alpha_t = np.random.uniform(0, 10)

# Generate a random value for 'beta_t' within the range [1, 2)
beta_t = np.random.uniform(1, 2)

# Print the randomly generated 'alpha_t' and 'beta_t' values
print(alpha_t, beta_t)
```

OUTPUT:

```
1.343863693601337 1.8988905573522503
```

```
import numpy as np

# Define a function to calculate locations based on angle, alpha, and
beta
def location(angle, alpha, beta):
    return beta * np.tan(angle) + alpha

# Number of data points
N = 200

# Generate random angles within the range [-π/2, π/2]
angles = np.random.uniform(-np.pi/2, np.pi/2, N)

# Calculate locations using the generated angles and the parameters
alpha_t and beta_t
locations = np.array([location(angle, alpha_t, beta_t) for angle in
angles])

import numpy as np
import matplotlib.pyplot as plt

# Calculate the evolving mean of the 'locations' data over time
mus = [locations[:i + 1].mean() for i in range(N)]

# Create an array for the true mean
mean = [locations.mean()] * N

# Create an array of data point indices from 1 to N
X = np.arange(1, N + 1)

# Set the plot style
plt.style.use('ggplot')

# Plot the evolving mean and the true mean
plt.plot(X, mus, label='Mean over time')
plt.plot(X, mean, label='True mean')

# Add labels for the x and y axes
plt.xlabel('$n$-th data point')
plt.ylabel(r'$\alpha$ (km)')

# Add a legend to the plot
plt.legend()

# Save the plot as an image file
plt.savefig('mean_x.png', bbox_inches='tight', dpi=300)

# Show the plot
plt.show()
```

OUTPUT:




```
print(locations.mean())
```

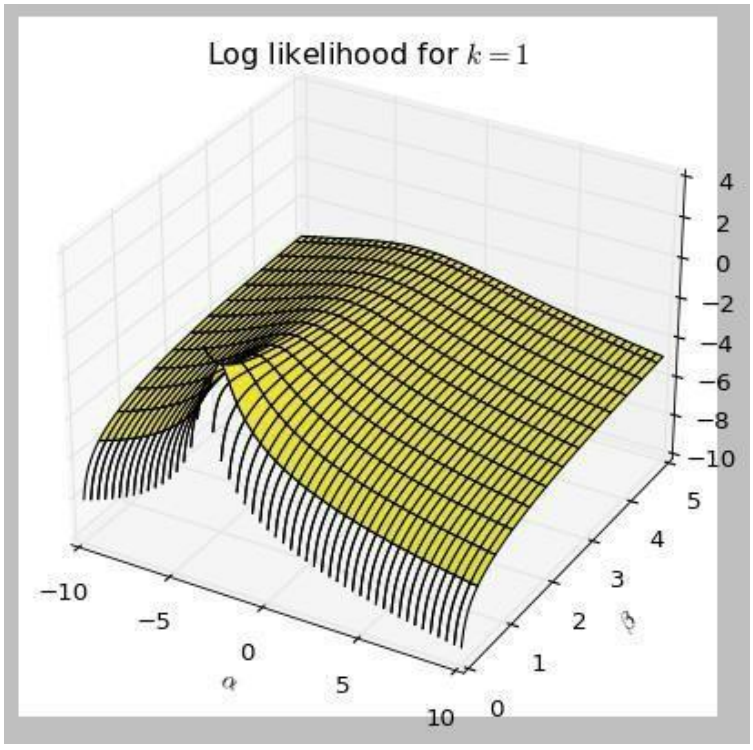
OUTPUT:

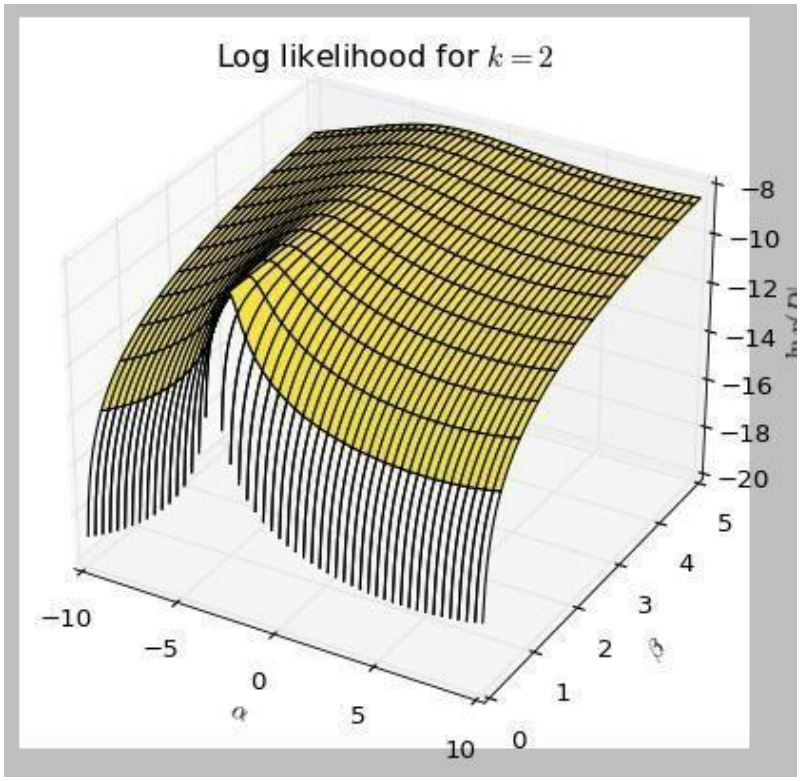
```
3.292333775274883
```

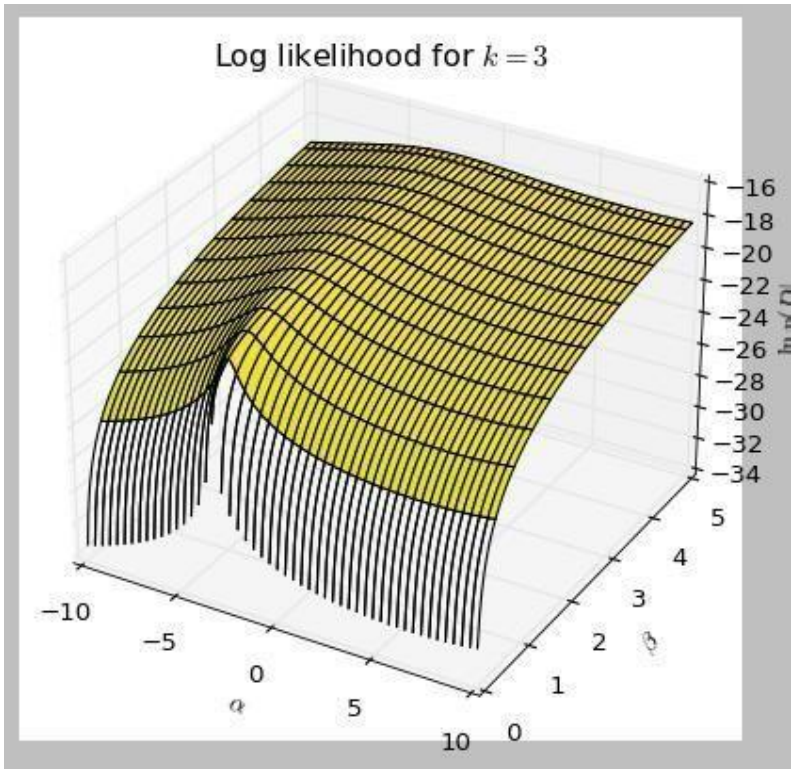
```
import numpy as np
import matplotlib.pyplot as plt

# Set the Matplotlib style to 'classic'
plt.style.use('classic')

# Define a range of values for 'k'
ks = [1, 2, 3, 20]

# Create grids of values for 'alphas' and 'betas' for plotting
alphas, betas = np.mgrid[-10:10:0.04, 0:5:0.04]

# Iterate through different values of 'k'
for k in ks:
    x = locations[:k]

    # Calculate the likelihood for each combination of 'alpha' and
'beta'
```

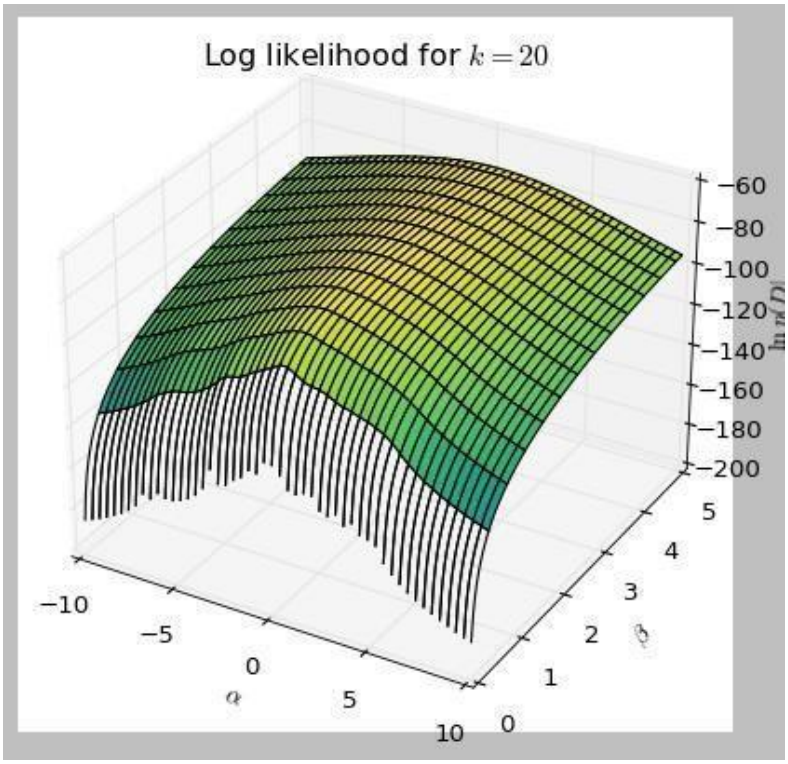
```
    likelihood = k * np.log(betas / np.pi)

    for loc in x:
        likelihood -= np.log(betas**2 + (loc - alphas)**2)

    # Create a new 3D plot for the log likelihood
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.plot_surface(alphas, betas, likelihood, cmap=plt.cm.viridis,
vmin=-200, vmax=likelihood.max())

    # Set labels and title for the plot
    plt.xlabel(r'$\alpha$')
    plt.ylabel(r'$\beta$')
    ax.set_zlabel('$\ln p(D | \alpha, \beta)$')
    plt.title('Log likelihood for $k = {}$'.format(k))

    # Save the plot as an image file
    plt.savefig('logl_{}.png'.format(k), bbox_inches='tight', dpi=300)


# Show the plot
    plt.show()
```

OUTPUT:




Log likelihood for $k = 2$
−8
−10
−12
−14
−16
−18
−20
−10
−5
0
5
10
a
0
1
2
3
4
5
b


Log likelihood for $k = 3$
−16
−18
−20
−22
−24
−26
−28
−30
−32
−34
−10
−5
0
5
10
a
0
1
2
3
4
5
b




```
from scipy.optimize import fmin

# Define a function to calculate the log likelihood of parameters
'alpha' and 'beta'
def log_likelihood(params, locations):
    alpha, beta = params
    likelihood = len(locations) * np.log(beta/np.pi)
    for loc in locations:
        likelihood -= np.log(beta**2 + (loc - alpha)**2)
    return -likelihood  # Negative log likelihood for minimization

# Define a function to estimate 'alpha' and 'beta' over sample size 'k'
and plot the results
def plot_maximize_logl(data, alpha_t, beta_t):
    alphas, betas = [], []
    x = np.arange(len(data))
    for k in x:
        [alpha, beta] = fmin(log_likelihood, (0, 1), args=(data[:k],))
        alphas.append(alpha)
        betas.append(beta)

    # Set the Matplotlib style to 'ggplot'
    plt.style.use('ggplot')

    # Plot the estimated 'alpha' and 'beta' over sample size 'k'
    plt.plot(x, alphas, label=r'$\alpha$')
```

```
    plt.plot(x, betas, label=r'$\beta$')

    # Plot the true 'alpha_t' and 'beta_t' values
    plt.plot(x, [alpha_t] * len(data), label=r'$\alpha_t$')
    plt.plot(x, [beta_t] * len(data), label=r'$\beta_t$')

    # Set labels and legend for the plot
    plt.xlabel('$k$')
    plt.ylabel('location (km)')
    plt.legend()

    # Save the plot as an image file
    plt.savefig('plots/min_logl.png', bbox_inches='tight', dpi=300)

    # Show the plot
    plt.show()
```

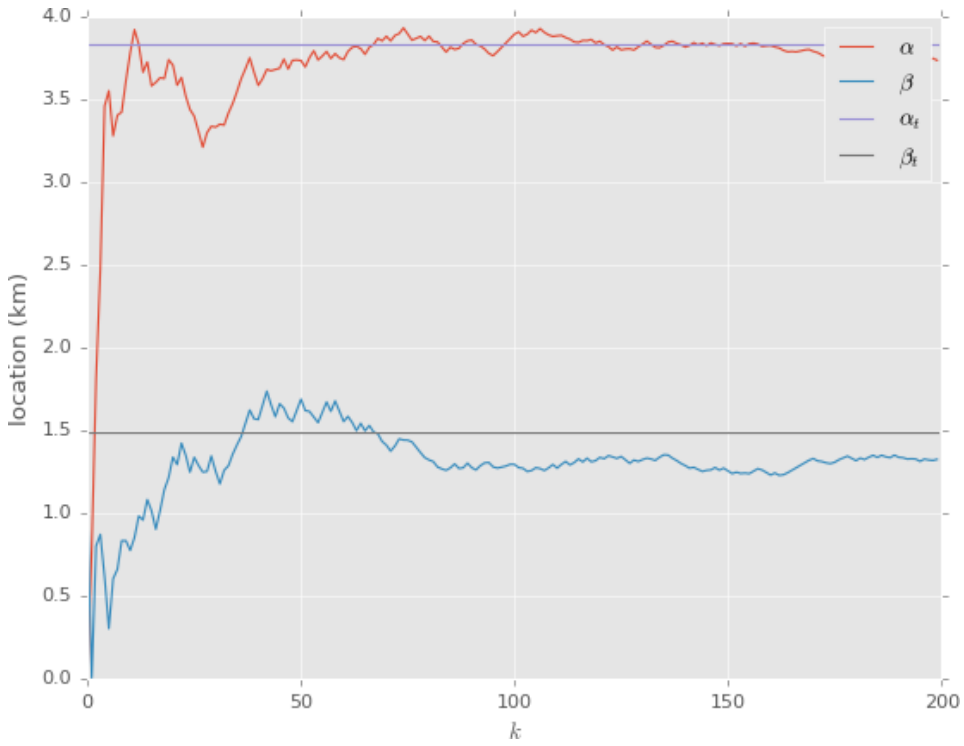
```
print(alphas[-1], betas[-1])
```

OUTPUT:

```
[9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96
 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96 9.96] [0.   0.04 0.08 0.12 0.16 0.2  0.24 0.28 0.32 0.36 0.4  0.44 0.48 0.52
 0.56 0.6  0.64 0.68 0.72 0.76 0.8  0.84 0.88 0.92 0.96 1.   1.04 1.08
 1.12 1.16 1.2  1.24 1.28 1.32 1.36 1.4  1.44 1.48 1.52 1.56 1.6  1.64
 1.68 1.72 1.76 1.8  1.84 1.88 1.92 1.96 2.   2.04 2.08 2.12 2.16 2.2
 2.24 2.28 2.32 2.36 2.4  2.44 2.48 2.52 2.56 2.6  2.64 2.68 2.72 2.76
 2.8  2.84 2.88 2.92 2.96 3.   3.04 3.08 3.12 3.16 3.2  3.24 3.28 3.32
 3.36 3.4  3.44 3.48 3.52 3.56 3.6  3.64 3.68 3.72 3.76 3.8  3.84 3.88
 3.92 3.96 4.   4.04 4.08 4.12 4.16 4.2  4.24 4.28 4.32 4.36 4.4  4.44
 4.48 4.52 4.56 4.6  4.64 4.68 4.72 4.76 4.8  4.84 4.88 4.92 4.96]
```

```
plot_maximize_logl(locations, alpha_t, beta_t)
```

OUTPUT:



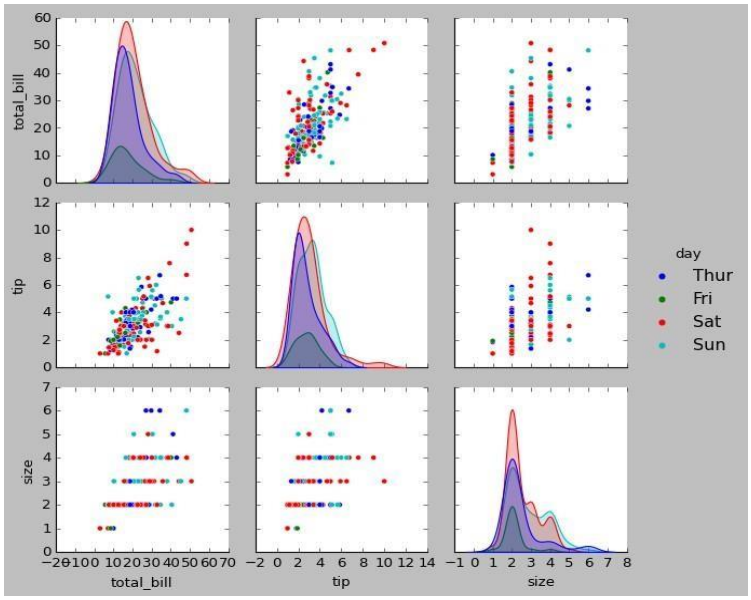

```
import pandas as pd
import numpy as np

# Read data from a CSV file named 'train.csv' using ';' as the
delimiter
# and store it in a DataFrame named 'a'
a=pd.read_csv("train.csv",sep=';')

# Print the DataFrame 'a' to the console
print(a)
```

OUTPUT:

```
     PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
0     1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/...
1     2,1,1,"Cumings, Mrs. John Bradley (Florence Br...
2     3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,S...
3     4,1,1,"Futrelle, Mrs. Jacques Heath (Lily May ...
4     5,0,3,"Allen, Mr. William Henry",male,35,0,0,3...
..                                                  ...
886   887,0,2,"Montvila, Rev. Juozas",male,27,0,0,21...
887   888,1,1,"Graham, Miss. Margaret Edith",female,...
888   889,0,3,"Johnston, Miss. Catherine Helen ""Car...
889   890,1,1,"Behr, Mr. Karl Howell",male,26,0,0,11...
890   891,0,3,"Dooley, Mr. Patrick",male,32,0,0,3703...

[891 rows x 1 columns]
```

```
import seaborn
import matplotlib.pyplot as plt
df = seaborn.load_dataset('tips')
seaborn.pairplot(df, hue='day')
plt.show()
```

OUTPUT:



LAB-03

DESCRIPTION:

Implementation Of Linear Regression Model Using US Housing Data.

PROBLEM STATEMENT:

Implement a linear regression model from scratch using Python. Students are required to create a simple linear regression algorithm, apply it to a real dataset, and evaluate the model's performance. Tasks include data preprocessing, model development, and visualization of the results. Evaluation is based on code quality, model accuracy, and the ability to interpret the regression output.

```
# Import necessary libraries for data analysis and visualization

 # NumPy library for numerical operations
import numpy as np

# Pandas library for data handling
import pandas as pd

 # Matplotlib for basic plotting
import matplotlib.pyplot as plt

# Seaborn for enhanced data visualization
import seaborn as sns

%matplotlib inline
```

```
# Read data from a CSV file located at "/content/USA_Housing.csv"
# and store it in a Pandas DataFrame named 'df'

df = pd.read_csv("/content/USA_Housing.csv")

# Display the first few rows of the DataFrame to inspect the data
df.head()
```

OUTPUT:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

```
# Display comprehensive information about the DataFrame 'df'
df.info(verbose=True)
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Avg. Area Income             5000 non-null   float64
 1   Avg. Area House Age          5000 non-null   float64
 2   Avg. Area Number of Rooms    5000 non-null   float64
 3   Avg. Area Number of Bedrooms 5000 non-null   float64
 4   Area Population              5000 non-null   float64
 5   Price                        5000 non-null   float64
 6   Address                      5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

```
# Generate a statistical summary of the DataFrame 'df' with specific
percentiles
df.describe(percentiles=[0.1,0.25,0.5,0.75,0.9])
```

OUTPUT:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
10%	55047.633980	4.697755	5.681951	2.310000	23502.845262	7.720318e+05
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
90%	82081.188283	7.243978	8.274222	6.100000	48813.618633	1.684621e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

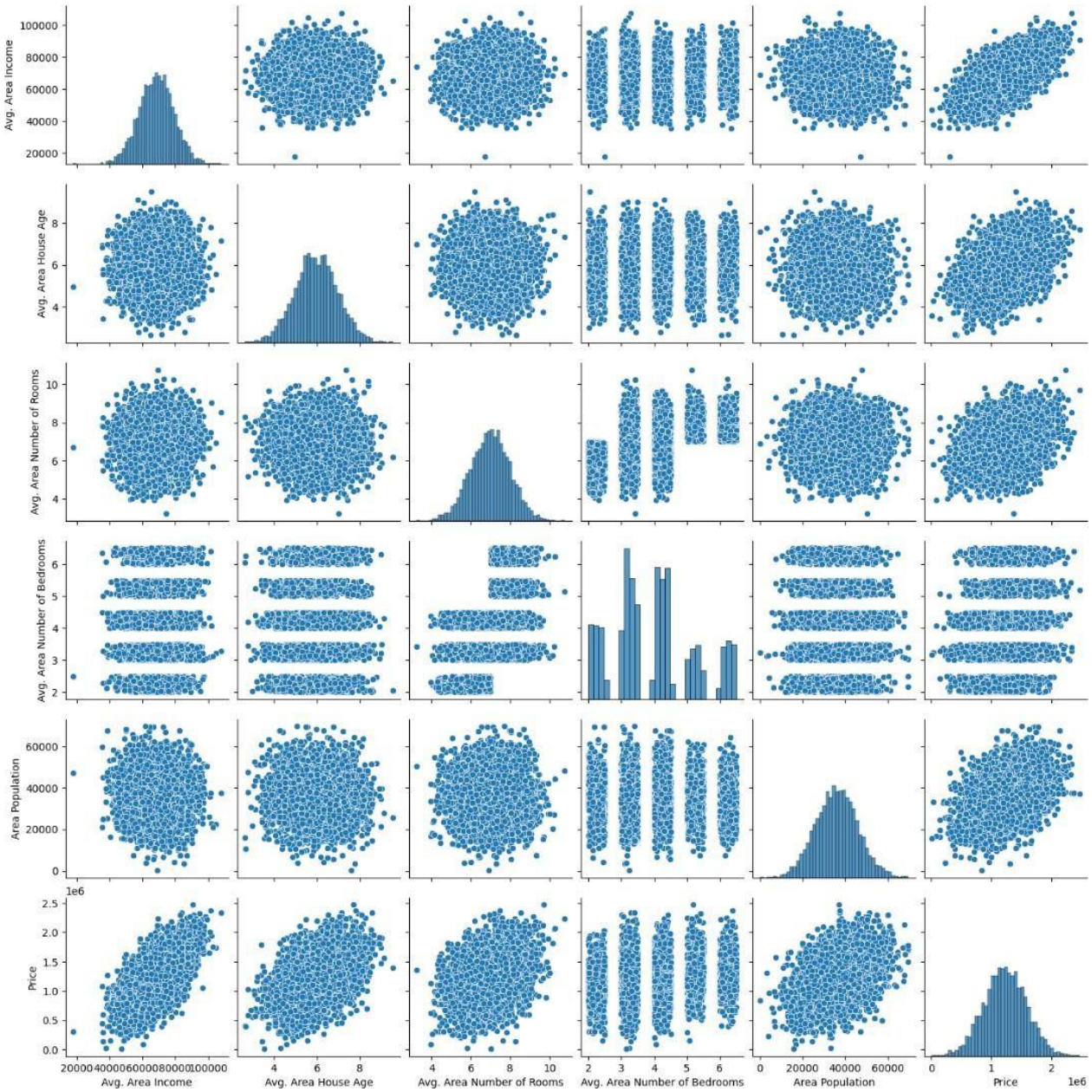
```
#display columns
df.columns
```

OUTPUT:

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

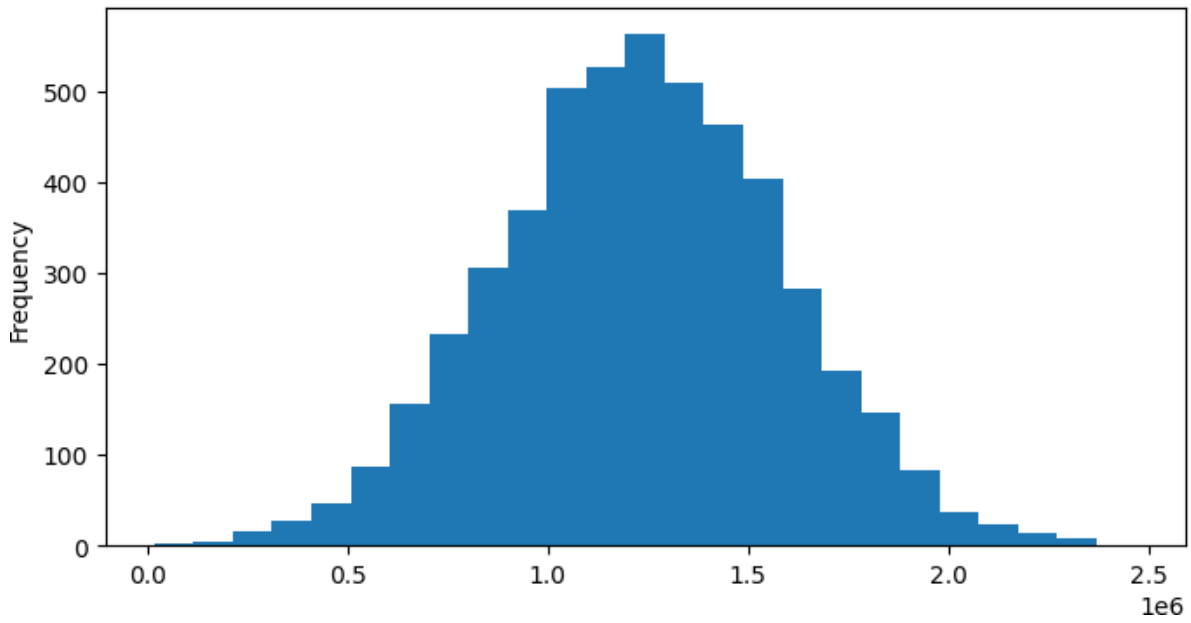
```
# Create a pair plot to visualize relationships between variables in
the DataFrame 'df'
sns.pairplot(df)
```

OUTPUT:



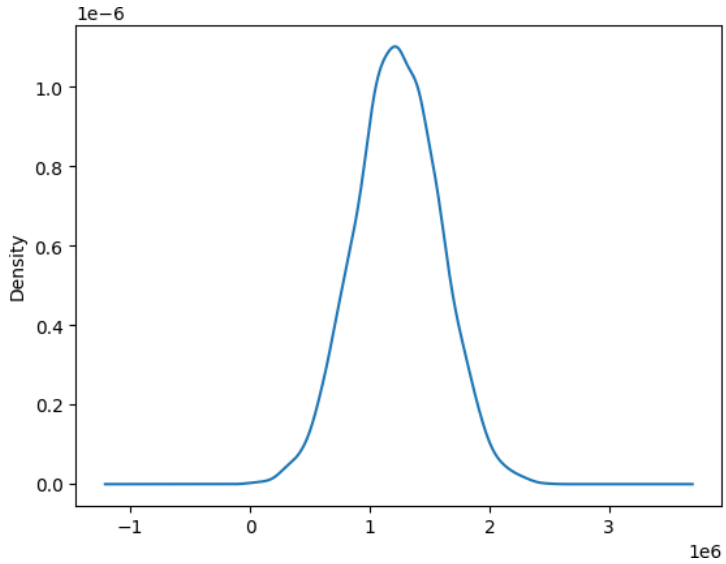

```
# Create a histogram plot for the 'Price' column with 25 bins and a
specific figure size
df['Price'].plot.hist(bins=25,figsize=(8,4))
```

OUTPUT:




```
# Create a density plot for the 'Price' column to visualize its
probability density distribution
df['Price'].plot.density()
```

OUTPUT:



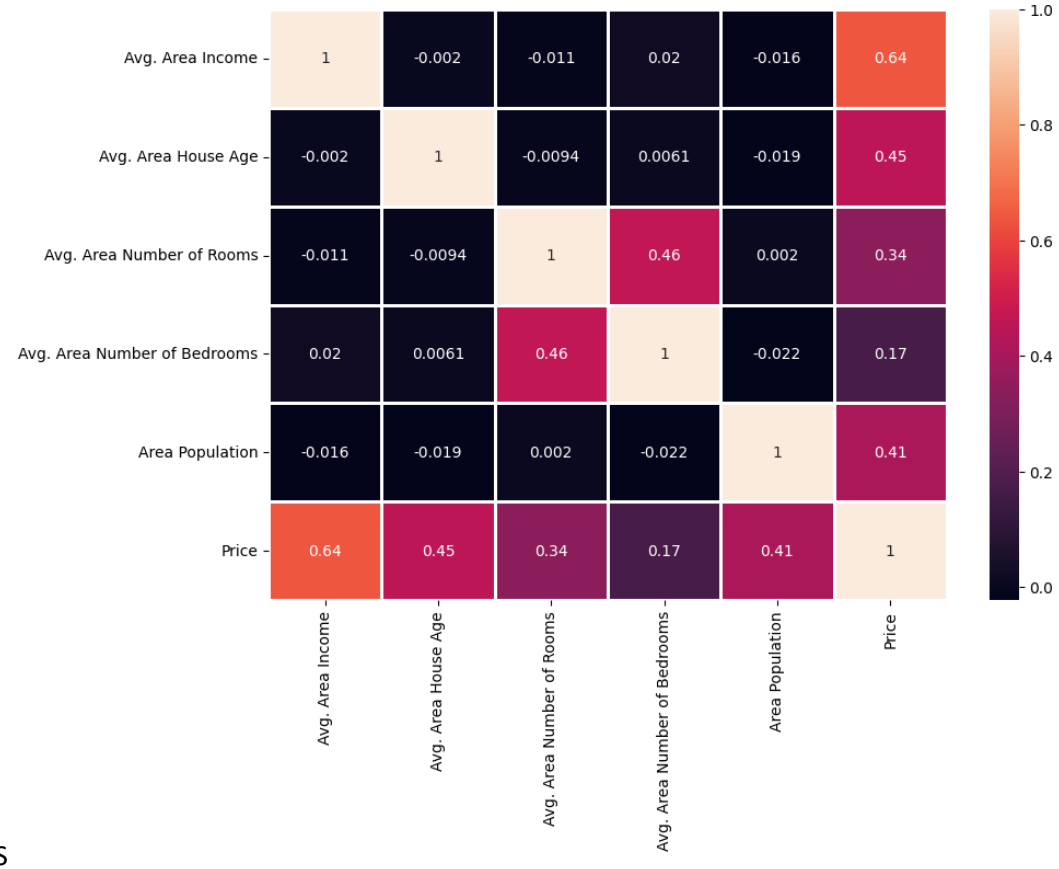

```
# Calculate the correlation matrix for the DataFrame 'df'

df.corr()
```

OUTPUT:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1.000000	-0.002007	-0.011032	0.019788	-0.016234	0.639734
Avg. Area House Age	-0.002007	1.000000	-0.009428	0.006149	-0.018743	0.452543
Avg. Area Number of Rooms	-0.011032	-0.009428	1.000000	0.462695	0.002040	0.335664
Avg. Area Number of Bedrooms	0.019788	0.006149	0.462695	1.000000	-0.022168	0.171071
Area Population	-0.016234	-0.018743	0.002040	-0.022168	1.000000	0.408556
Price	0.639734	0.452543	0.335664	0.171071	0.408556	1.000000

```
# Create a figure with a specified size for the heatmap
plt.figure(figsize=(10,7))

# Generate a heatmap of the correlation matrix for the DataFrame 'df'
# Annotate the cells with correlation values and add linewidths
sns.heatmap(df.corr(),annot=True,linewidths=2)
```

OUTPUT:



SS

```
l_column = list(df.columns) # Making a list out of column names
len_feature = len(l_column) # Length of column vector list
l column
```

OUTPUT:

```
['Avg. Area Income',
 'Avg. Area House Age',
 'Avg. Area Number of Rooms',
 'Avg. Area Number of Bedrooms',
 'Area Population',
 'Price',
 'Address']
```

```
# Create feature variables (X) by selecting columns from the DataFrame 'df'
# The feature columns are from the first column to the (len_feature-2)-th column
X = df[l_column[0:len_feature-2]]

# Create the target variable (y) by selecting the (len_feature-2)-th column from the DataFrame 'df'
y = df[l column[len feature-2]]
```

```
# Print the size (dimensions) of the feature set 'X'
print("Feature set size:", X.shape)

# Print the size (dimensions) of the variable set 'y'
print("Variable set size:", y.shape)
```

OUTPUT:

```
Feature set size: (5000, 5)
Variable set size: (5000,)
```

```
# Display the first few rows of the feature set 'X'
X.head()
```

OUTPUT:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
0	79545.458574	5.682861	7.009188	4.09	23086.800503
1	79248.642455	6.002900	6.730821	3.09	40173.072174
2	61287.067179	5.865890	8.512727	5.13	36882.159400
3	63345.240046	7.188236	5.586729	3.26	34310.242831
4	59982.197226	5.040555	7.839388	4.23	26354.109472

```
# Display the first few rows of the feature set 'y'
y.head()
```

OUTPUT:

```
0    1.059034e+06
1    1.505891e+06
2    1.058988e+06
3    1.260617e+06
4    6.309435e+05
Name: Price, dtype: float64
```

```
# Import the 'train_test_split' function from Scikit-Learn for
splitting datasets
from sklearn.model selection import train test split
```

```
# Split the feature set 'X' and target variable 'y' into training and
testing sets
# The testing set will comprise 30% of the data, and a random seed of
123 is used for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=123)
```

```
# Print the size (dimensions) of the training feature set 'X_train'
print("Training feature set size:", X_train.shape)

# Print the size (dimensions) of the testing feature set 'X_test'
print("Test feature set size:", X_test.shape)

# Print the size (dimensions) of the training variable set 'y train'
```

```
print("Training variable set size:", y_train.shape)

# Print the size (dimensions) of the testing variable set 'y_test'
print("Test variable set size:", y_test.shape)
```

OUTPUT:

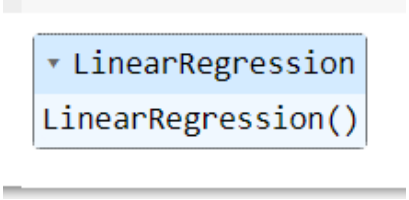
```
Training feature set size: (3500, 5)
Test feature set size: (1500, 5)
Training variable set size: (3500,)
Test variable set size: (1500,)
```

```
# Import the LinearRegression class from Scikit-Learn for building
linear regression models
from sklearn.linear_model import LinearRegression

# Import the metrics module from Scikit-Learn for model evaluation and
performance metrics
from sklearn import metrics
```

```
 # Creating a Linear Regression object 'lm'
 lm = LinearRegression()
```

```
 # Fit the linear model on to the 'lm' object itself i.e. no need to
set this to another variable
 lm.fit(X_train,y_train)
```

OUTPUT:




```
# Print the intercept term (constant) of the linear regression model
stored in the 'lm' variable
print("The intercept term of the linear model:", lm.intercept_)
```

OUTPUT:

```
The intercept term of the linear model: -
2631028.9017454907
```

```
# Print the coefficients of the linear regression model stored in the
'lm' variable
print("The coefficients of the linear model:", lm.coef )
```

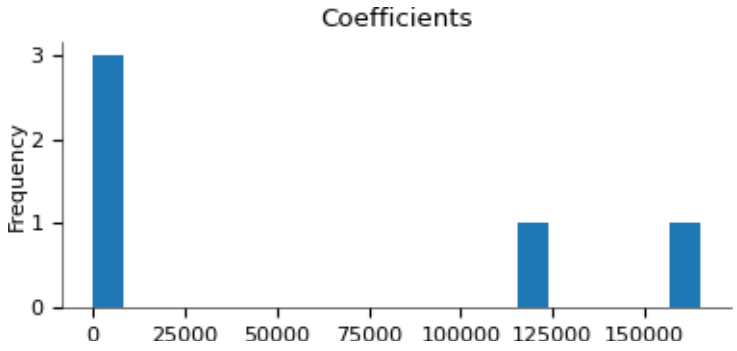
OUTPUT:

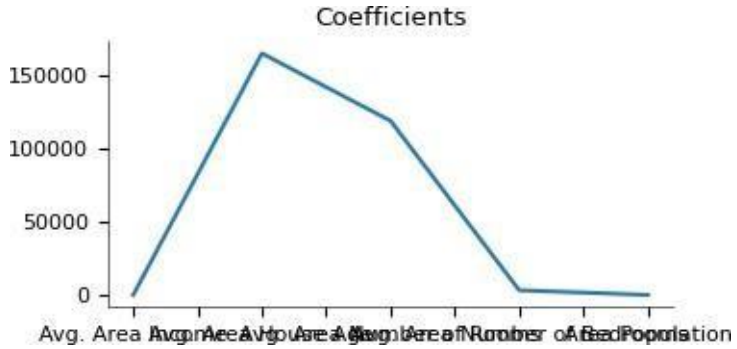
```
The coefficients of the linear model: [2.15976020e+01
1.65201105e+05 1.19061464e+05 3.21258561e+03
 1.52281212e+01]
```

```
# Create a DataFrame 'cdf' to store the coefficients of the linear
regression model
# The coefficients are associated with the feature names in
'X_train.columns'
cdf = pd.DataFrame(data=lm.coef_, index=X_train.columns,
columns=["Coefficients"])
cdf
```

OUTPUT:

	Coefficients
Avg. Area Income	21.597602
Avg. Area House Age	165201.104954
Avg. Area Number of Rooms	119061.463868
Avg. Area Number of Bedrooms	3212.585606
Area Population	15.228121





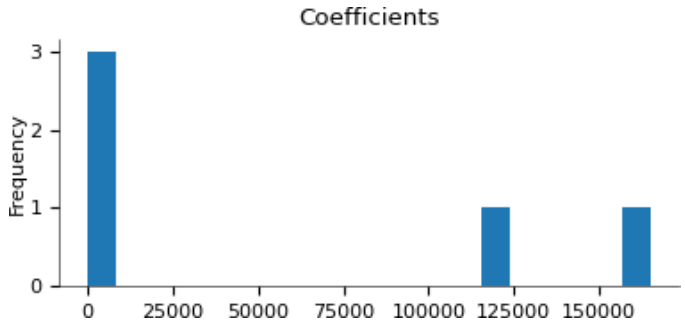

```
# Calculate the standard error for each coefficient
n = X_train.shape[0] # Number of observations
k = X_train.shape[1]  # Number of features
dfN = n - k  # Degrees of freedom for the residuals

# Predict the target variable using the linear model
train_pred = lm.predict(X_train)

# Calculate the squared errors between the predicted and actual values
train_error = np.square(train_pred - y_train)

# Sum of squared errors
sum_error = np.sum(train_error)

# Initialize an array to store standard errors
se = [0, 0, 0, 0, 0]

# Calculate the standard error for each coefficient
for i in range(k):
    r = (sum_error / dfN)
    r = r / np.sum(np.square(X_train[list(X_train.columns)[i]] -
X_train[list(X_train.columns)[i]].mean()))
```

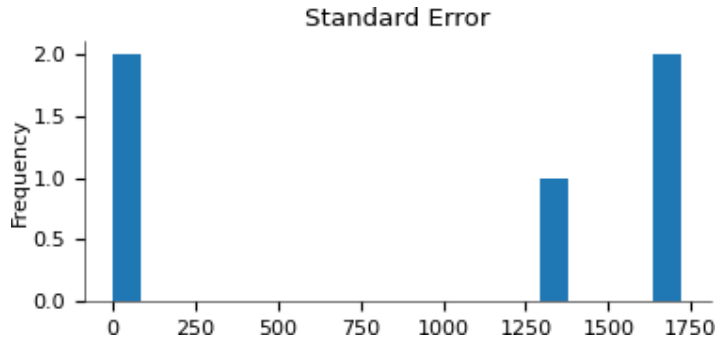
```
    se[i] = np.sqrt(r)

# Add the standard error and t-statistic to the 'cdf' DataFrame
cdf['Standard Error'] = se
cdf['t-statistic'] = cdf['Coefficients'] / cdf['Standard Error']
cdf
```

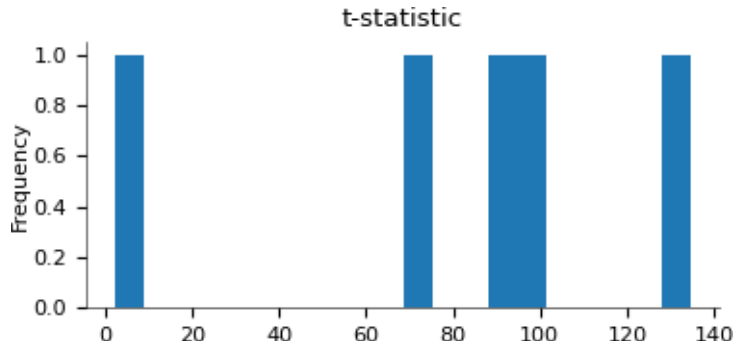
OUTPUT:

	Coefficients	Standard Error	t-statistic
Avg. Area Income	21.597602	0.160361	134.681505
Avg. Area House Age	165201.104954	1722.412068	95.912649
Avg. Area Number of Rooms	119061.463868	1696.546476	70.178722
Avg. Area Number of Bedrooms	3212.585606	1376.451759	2.333962
Area Population	15.228121	0.169882	89.639472

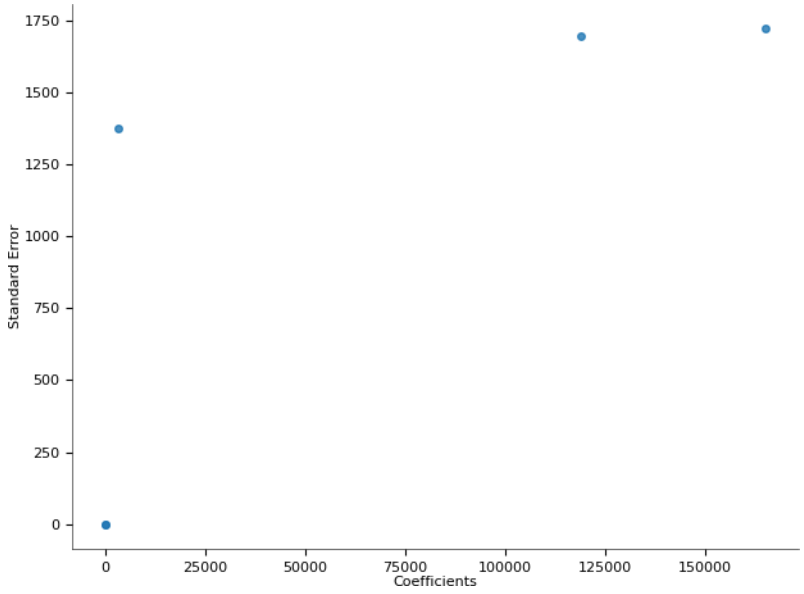
DISTRIBUTIONS:

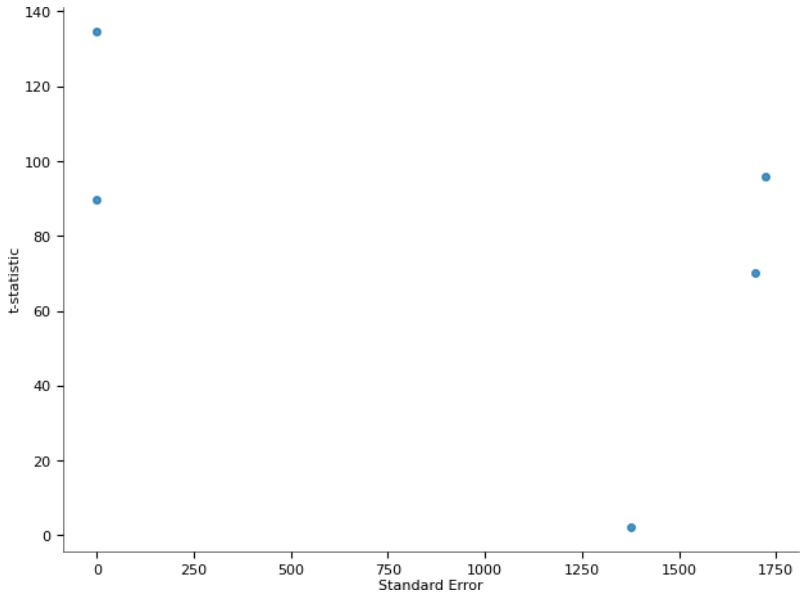




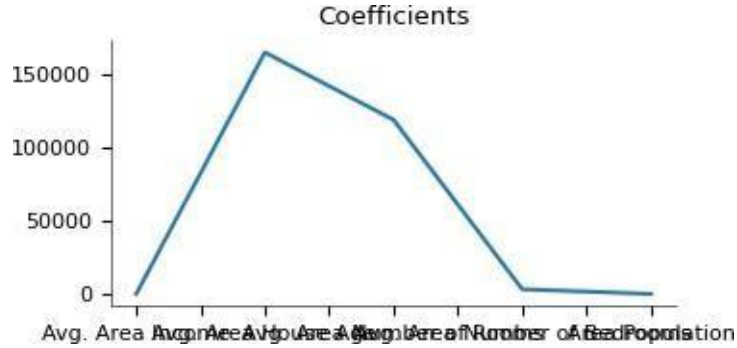


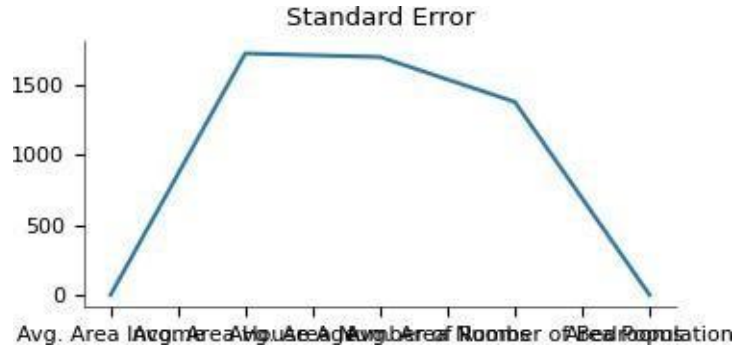
2D-DISTRIBUTIONS:

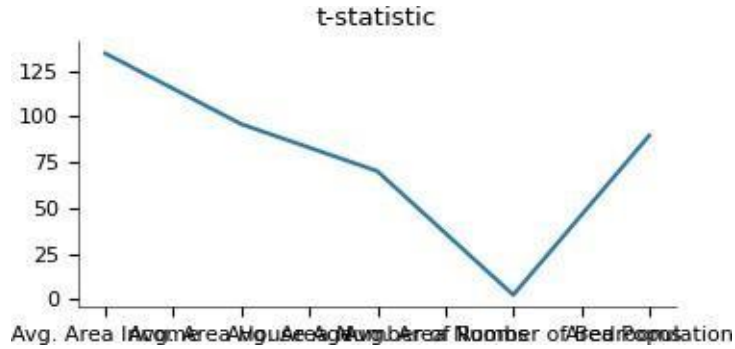




VALUES:








```
# Print a message indicating the features arranged in order of
importance for predicting house prices
print("Therefore, features arranged in the order of importance for
predicting the house price\n", '-'*90, sep='')

# Sort the features based on t-statistic values in descending order
l = list(cdf.sort_values('t-statistic', ascending=False).index)
```

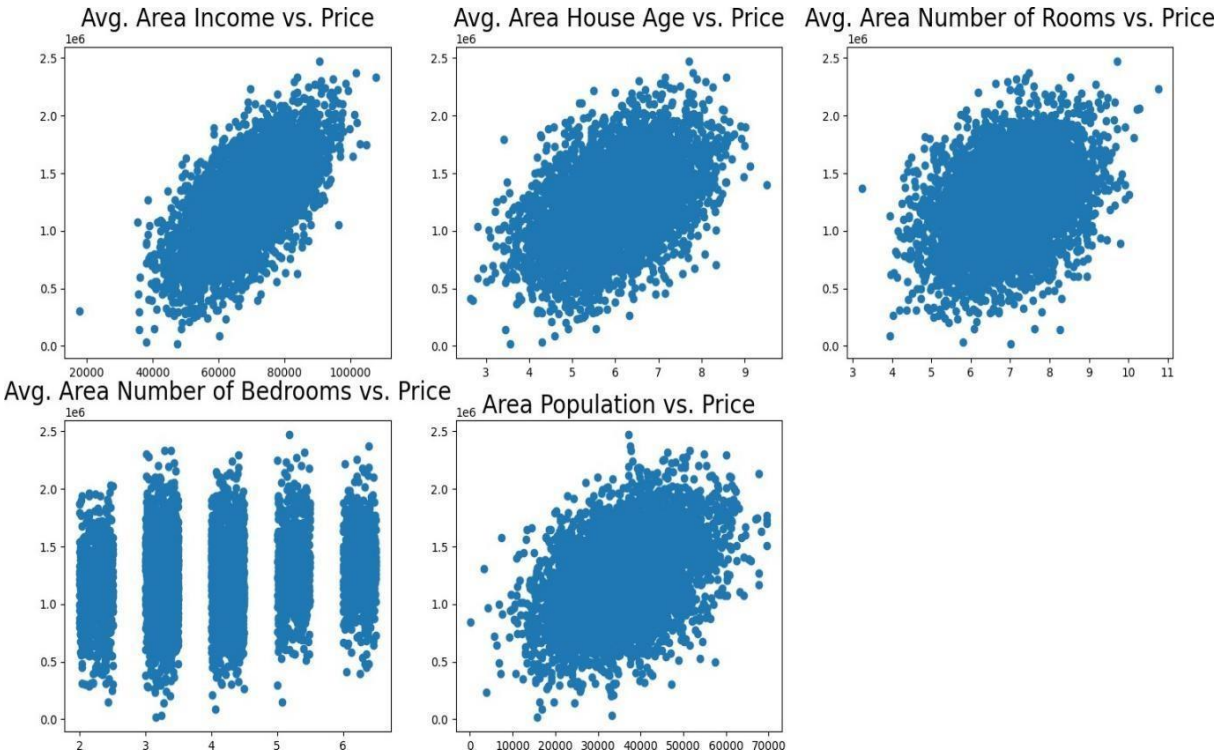
```
# Print the sorted features
print(' > \n'.join(l))
```

OUTPUT:

```
Therefore, features arranged in the order of importance for predicting the house price
----------------------------------------------------------------------------------------
Avg. Area Income >
Avg. Area House Age >
Area Population >
Avg. Area Number of Rooms >
Avg. Area Number of Bedrooms
```

```
# Create a multi-plot visualization to compare features with 'Price'
l = list(cdf.index) # List of feature names

from matplotlib import gridspec

# Create a figure with a 2x3 grid of subplots
fig = plt.figure(figsize=(18, 10))
gs = gridspec.GridSpec(2, 3)

# Create subplots and scatter plots for each feature
ax0 = plt.subplot(gs[0])
ax0.scatter(df[l[0]], df['Price'])
ax0.set_title(l[0] + " vs. Price", fontdict={'fontsize': 20})

ax1 = plt.subplot(gs[1])
ax1.scatter(df[l[1]], df['Price'])
ax1.set_title(l[1] + " vs. Price", fontdict={'fontsize': 20})

ax2 = plt.subplot(gs[2])
ax2.scatter(df[l[2]], df['Price'])
ax2.set_title(l[2] + " vs. Price", fontdict={'fontsize': 20})

ax3 = plt.subplot(gs[3])
ax3.scatter(df[l[3]], df['Price'])
ax3.set_title(l[3] + " vs. Price", fontdict={'fontsize': 20})

ax4 = plt.subplot(gs[4])
ax4.scatter(df[l[4]], df['Price'])
ax4.set_title(l[4] + " vs. Price", fontdict={'fontsize': 20})
```

OUTPUT:




```
# Calculate the R-squared value for the model's fit
# Print the R-squared value rounded to three decimal places
print("R-squared value of this
fit:",round(metrics.r2_score(y_train,train_pred),3))
```

OUTPUT:

```
R-squared value of this fit: 0.917
```

```
# Generate predictions using the linear regression model on the test
data
predictions = lm.predict(X_test)

# Print the type of the predictions (usually a NumPy array)
print("Type of the predicted object:", type(predictions))

# Print the size (dimensions) of the predictions
print("Size of the predicted object:", predictions.shape)
```

OUTPUT:

```
Type of the predicted object: <class 'numpy.ndarray'>
Size of the predicted object: (1500,)
```

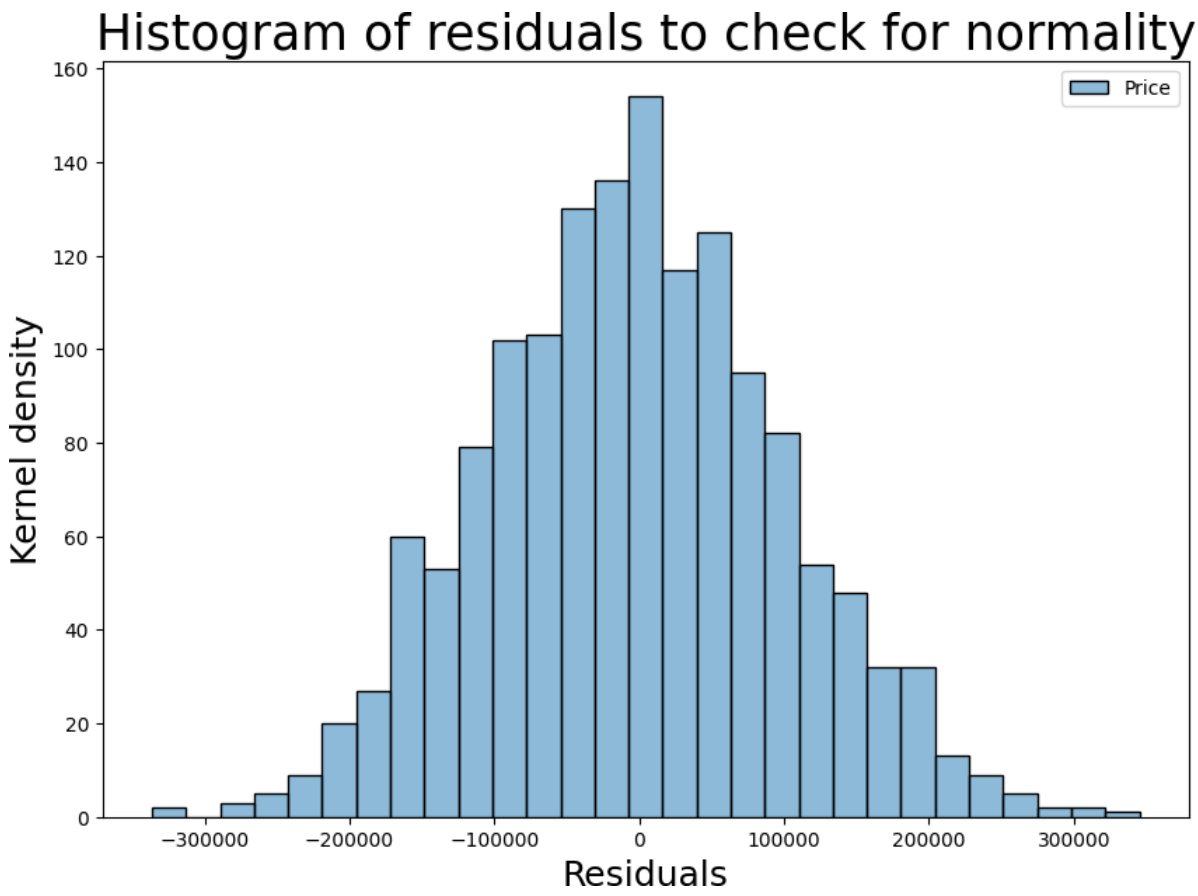
```
# Create a scatter plot to compare actual vs. predicted house prices
plt.figure(figsize=(10, 7))  # Set the figure size

# Set the plot title and axis labels
plt.title("Actual vs. predicted house prices", fontsize=25)
plt.xlabel("Actual test set house prices", fontsize=18)
plt.ylabel("Predicted house prices", fontsize=18)

# Create the scatter plot with actual house prices on the x-axis and
predicted house prices on the y-axis
plt.scatter(x=y_test, y=predictions)
```

OUTPUT:


```
# Create a figure for the histogram and kernel density plot
```

```
plt.figure(figsize=(10, 7))

# Set the plot title and axis labels
plt.title("Histogram of residuals to check for normality", fontsize=25)
plt.xlabel("Residuals", fontsize=18)
plt.ylabel("Kernel density", fontsize=18)

# Create a histogram with a kernel density plot for the residuals
(y_test - predictions)
sns.histplot([y test - predictions])
```

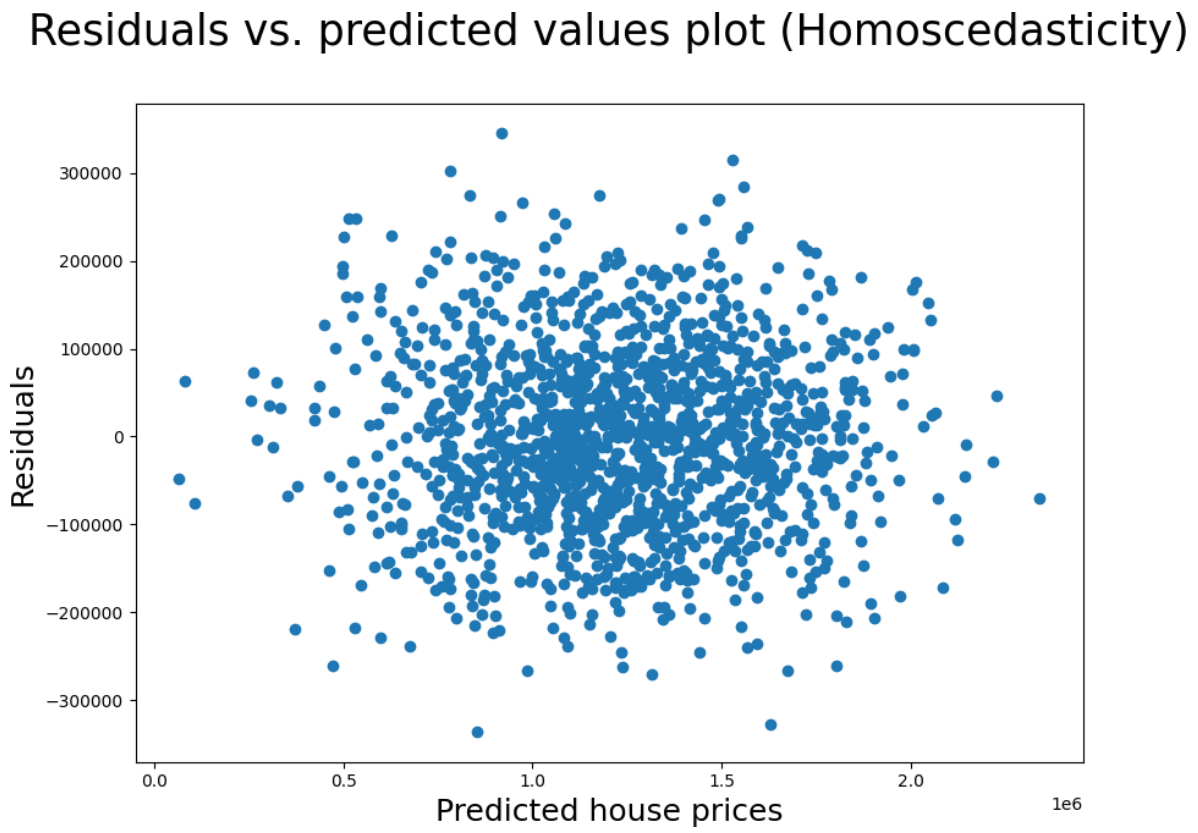
OUTPUT:




```
# Create a scatter plot of residuals vs. predicted values to check for
homoscedasticity
plt.figure(figsize=(10, 7))  # Set the figure size

# Set the plot title and axis labels
plt.title("Residuals vs. predicted values plot (Homoscedasticity)\n",
fontsize=25)
```

```
plt.xlabel("Predicted house prices", fontsize=18)
plt.ylabel("Residuals", fontsize=18)

# Create the scatter plot with predicted values on the x-axis and
residuals on the y-axis
plt.scatter(x=predictions, y=y_test - predictions)
```

OUTPUT:




```
# Calculate and print the Mean Absolute Error (MAE)
print("Mean absolute error (MAE):",
metrics.mean_absolute_error(y_test,predictions))

# Calculate and print the Mean Squared Error (MSE)
print("Mean square error (MSE):",
metrics.mean_squared_error(y_test,predictions))

# Calculate and print the Root Mean Squared Error (RMSE)
print("Root mean square error (RMSE):",
np.sqrt(metrics.mean squared error(y test,predictions)))
```

OUTPUT:

```
Mean absolute error (MAE): 81739.77482718184
Mean square error (MSE): 10489638335.804983
Root mean square error (RMSE): 102418.93543581179
```

```
# Calculate the R-squared value for the predictions on the test data
print("R-squared value of
predictions:",round(metrics.r2_score(y_test,predictions),3))
```

OUTPUT:

```
R-squared value of predictions: 0.919
```

```
import numpy as np

# Calculate the minimum and maximum values of predictions after scaling
min=np.min(predictions/6000)
max=np.max(predictions/12000)

# Print the minimum and maximum scaled values
print(min, max)
```

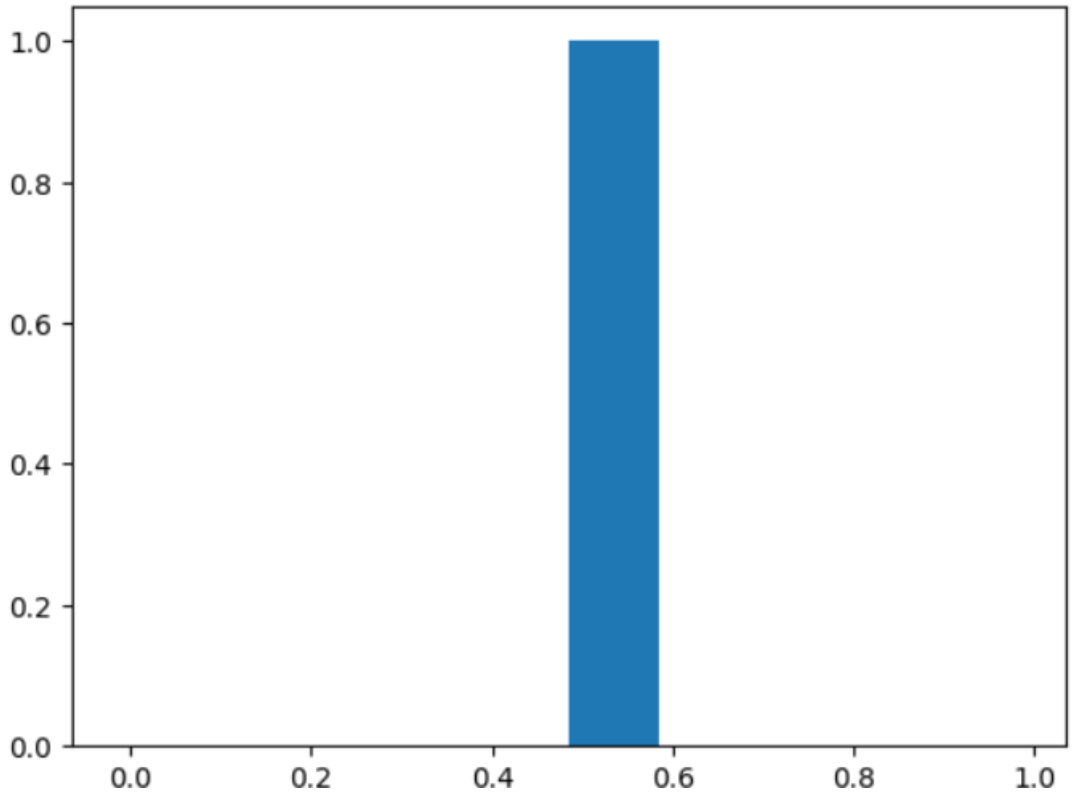
OUTPUT:

```
10.57339854753646 195.14363973516853
```

```
# Calculate a new value L using the formula
L = (100 - min)/(max - min)

L

# Create a histogram plot of the values in L
plt.hist(L)
```

OUTPUT:

```
(array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
 array([-0.01548743,  0.08451257,  0.18451257,  0.28451257,  0.38451257,
         0.48451257,  0.58451257,  0.68451257,  0.78451257,  0.88451257,
         0.98451257]),
 <BarContainer object of 10 artists>)
```



LAB-04

DESCRIPTION:

Implementation of Logistic Regression Model Using Titanic-Ship Data set.

PROBLEM STATEMENT:

Implement linear regression using a pre-defined library (e.g., Scikit-Learn) and a custom implementation from scratch. Compare both approaches in terms of accuracy, speed, and resource usage to gain insights into the trade-offs between library usage and custom coding in linear regression models

```
# Import the pandas library
import pandas as pd

# Define the file path for the CSV file
path = "/content/titanic_train.csv"

# Read the CSV file into a pandas DataFrame
df = pd.read_csv(path)

# Print the contents of the DataFrame
print(df)
```

OUTPUT:

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                 Name     Sex   Age  SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0      1
1   Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                              Heikkinen, Miss. Laina  female  26.0      0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                            Allen, Mr. William Henry    male  35.0      0
..                                                ...     ...   ...    ...
886                            Montvila, Rev. Juozas    male  27.0      0
887                     Graham, Miss. Margaret Edith  female  19.0      0
888         Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                            Behr, Mr. Karl Howell    male  26.0      0
890                              Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

```
# Import the pandas library
import pandas as pd
```

```
# Define the file path for the CSV file
path = "/content/titanic_train.csv"

# Read the CSV file into a pandas DataFrame
df = pd.read_csv(path)

# Print the data types of the columns in the DataFrame
print(df.dtypes)
```

OUTPUT:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
import pandas as pd
path="/content/titanic_train.csv"
df=pd.read_csv(path)
print(df.dtypes)
```

OUTPUT:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
df.columns
```

OUTPUT:

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name',
'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare',
'Cabin', 'Embarked'], dtype='object')
```

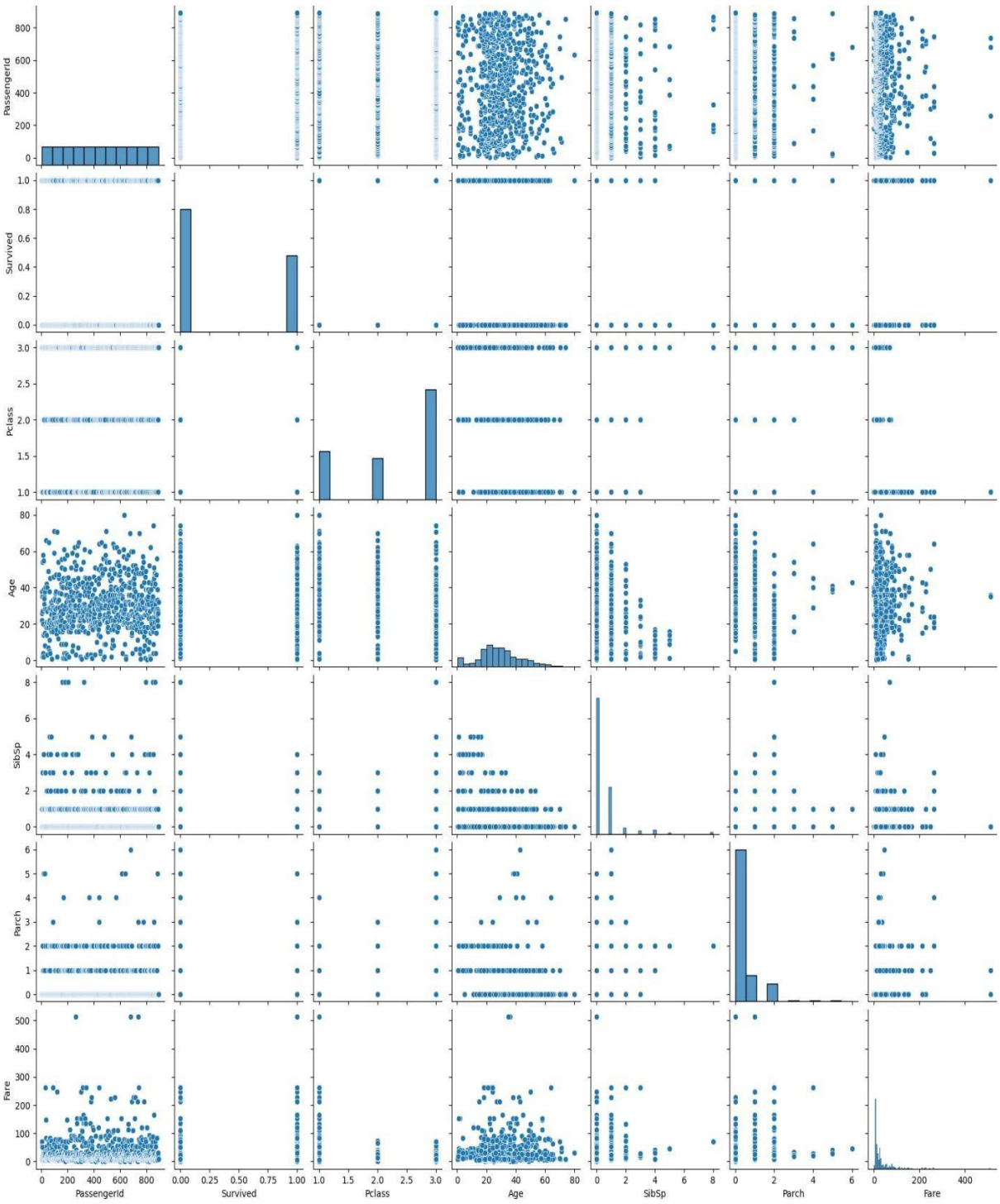
```
import numpy as np
import pandas as pd
file_path="/content/titanic_train.csv"
df=pd.read_csv(file_path)
column_name='Age'
data=df[column_name]
min_val=np.min(data)
max_val=np.max(data)
std_dev=np.std(data)
quartiles=np.percentile(data,[25,50,75,90])
print("Summary of the dataset:")
print("Minimum:",min_val)
print("Maxmimum:",max_val)
print("Standard Deviation:",std_dev)
print("25th percentile:",quartiles[0])
print("50th percentile(Median):",quartiles[1])
print("75th percentile:",quartiles[2])
print("90th percentile:",quartiles[3])
```

OUTPUT:

```
Summary of the dataset:
Minimum: 0.42
Maxmimum: 80.0
Standard Deviation: 14.516321150817316
25th percentile: nan
50th percentile(Median): nan
75th percentile: nan
90th percentile: nan
```

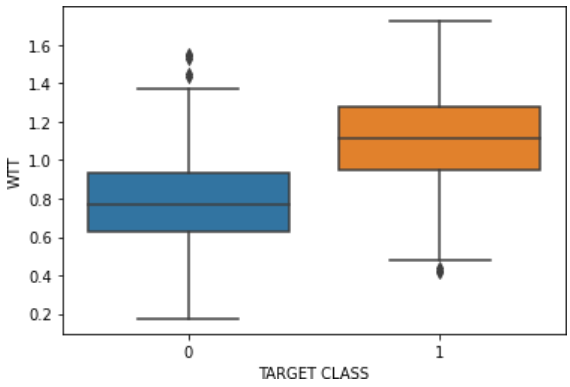
```
import seaborn as sns
sns.pairplot(df)
```

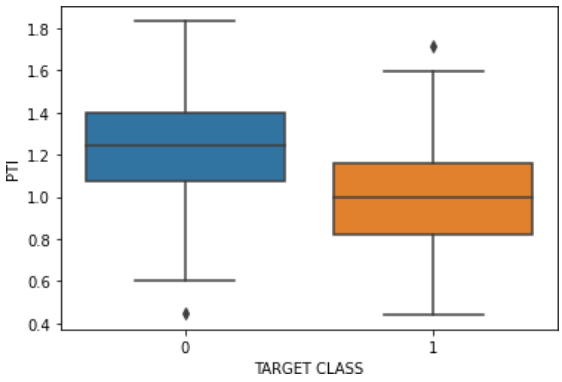
OUTPUT:

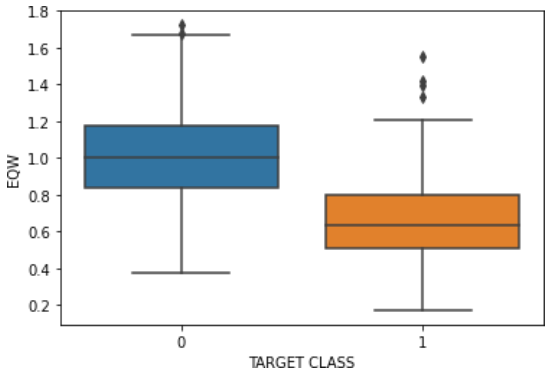


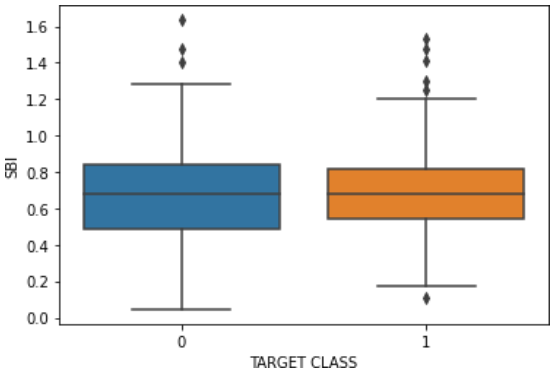
```
# Iterate over a range of feature indices, excluding the last one

for i in range(len(l) - 1):
    # Create a boxplot using Seaborn
    sns.boxplot(x='TARGET CLASS', y=l[i], data=df)
    # Display the boxplot for the current feature

    # Create a new empty figure for the next boxplot
    plt.figure()

# The loop generates multiple boxplots, each showing the distribution of a
feature
# for different categories in the 'TARGET CLASS' column.
```

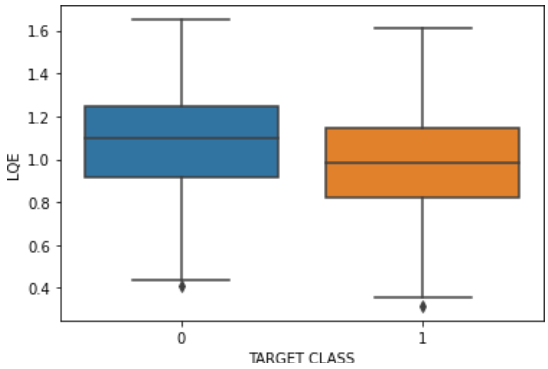
OUTPUT:

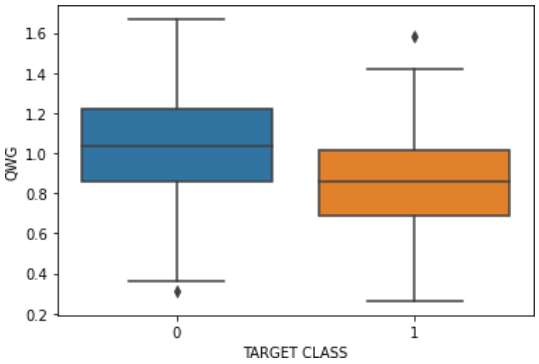


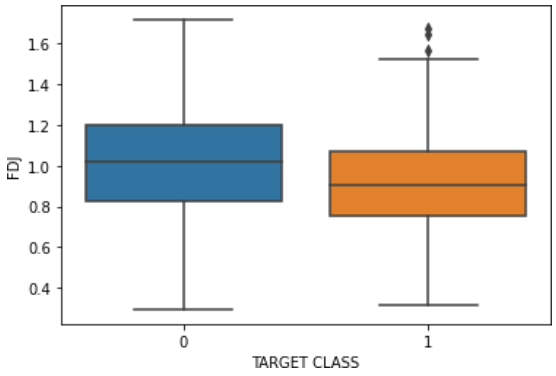


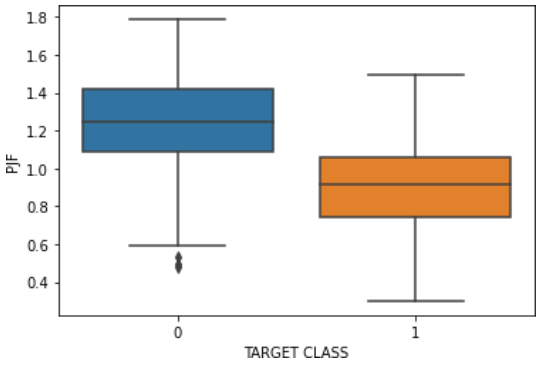


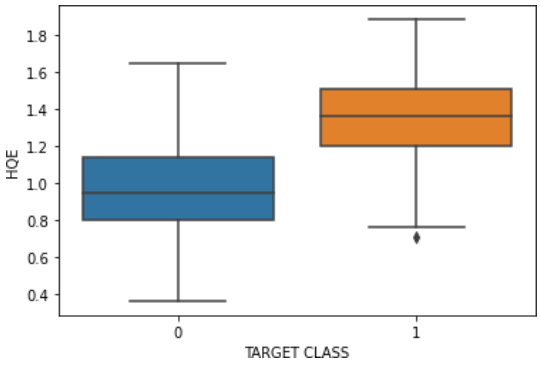


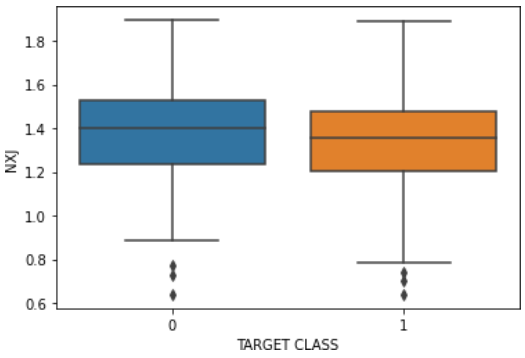













```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()


# Fit the scaler to the feature data in the DataFrame, excluding the 'TARGET
CLASS' column
scaler.fit(df.drop('TARGET CLASS',axis=1))

# This standardizes the features by subtracting the mean and dividing by the
standard deviation
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
```

```
# Create a new DataFrame 'df_feat' with standardized feature data
df_feat = pd.DataFrame(scaled_features, columns=df.columns[:-1])

# Display the first few rows of the new DataFrame
df_feat.head()
```

OUTPUT:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

```
# Import the 'train_test_split' function from scikit-learn
from sklearn.model_selection import train_test_split

# Assign the feature matrix (X) and the target vector (y)
X = df_feat
y = df['TARGET CLASS']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50,
random_state=101)
```

```
# Import the KNeighborsClassifier from scikit-learn
from sklearn.neighbors import KNeighborsClassifier

# Create a KNN classifier with one neighbor
knn = KNeighborsClassifier(n_neighbors=1)

# Train the KNN classifier on the training data
knn.fit(X_train, y_train)
```

OUTPUT:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

```
# Use the trained KNN classifier to make predictions on the test data
pred = knn.predict(X_test)


# Import necessary libraries
from sklearn.metrics import classification_report, confusion_matrix

# Calculate the confusion matrix
conf_mat = confusion_matrix(y_test, pred)

# Print the confusion matrix
print(conf_mat)
```

OUTPUT:

```
[[233  17]
 [ 24 226]]
```

```
# Calculate the classification report
print(classification_report(y_test,pred))
```

OUTPUT:

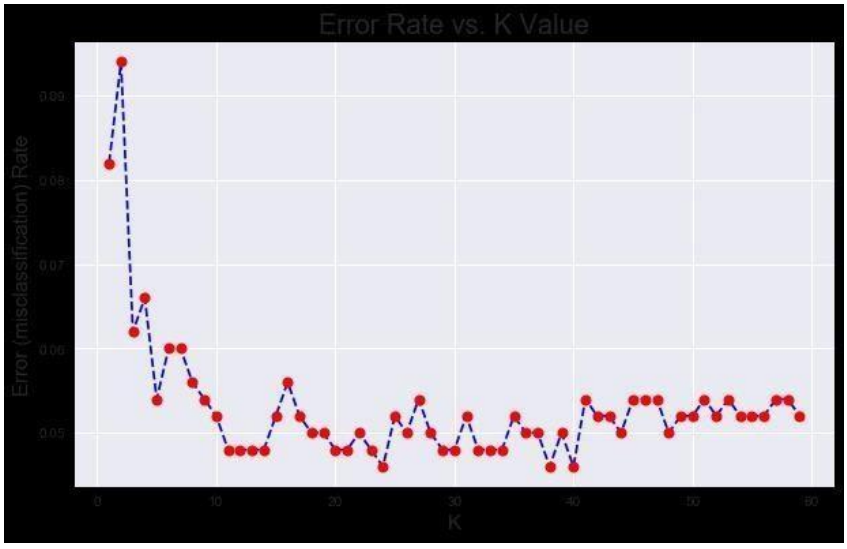
```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       250
           1       0.93      0.90      0.92       250

    accuracy                           0.92       500
   macro avg       0.92      0.92      0.92       500
weighted avg       0.92      0.92      0.92       500
```

```
# Print the misclassification error rate
print("Misclassification error rate:",round(np.mean(pred!=y_test),3))



error_rate = []

# Iterate over a range of values for k (number of neighbors) from 1 to 59
for i in range(1, 60):

    # Create a KNeighborsClassifier with the current value of k
```

```
    knn = KNeighborsClassifier(n_neighbors=i)

    # Fit the model on the training data
    knn.fit(X_train, y_train)

    # Make predictions on the test data
    pred_i = knn.predict(X_test)

    # Calculate the error rate for the current k value
    error_rate.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(10,6))

# Create a line plot of error_rate values from k=1 to k=59
plt.plot(range(1,60), error_rate, color='blue', linestyle='dashed',
marker='o',
         markerfacecolor='red', markersize=8)

# Set the title of the plot
plt.title('Error Rate vs. K Value', fontsize=20)

# Label the x-axis as 'K' (number of neighbors)
plt.xlabel('K', fontsize=15)

# Label the y-axis as 'Error (misclassification) Rate'
plt.ylabel('Error (misclassification) Rate', fontsize=15)
```

OUTPUT:



LAB-05

DESCRIPTION:

Implementation of Kernel Density Estimation for Feature Space and implementation of L1,L2 Regularization using Boston Housing Data set.

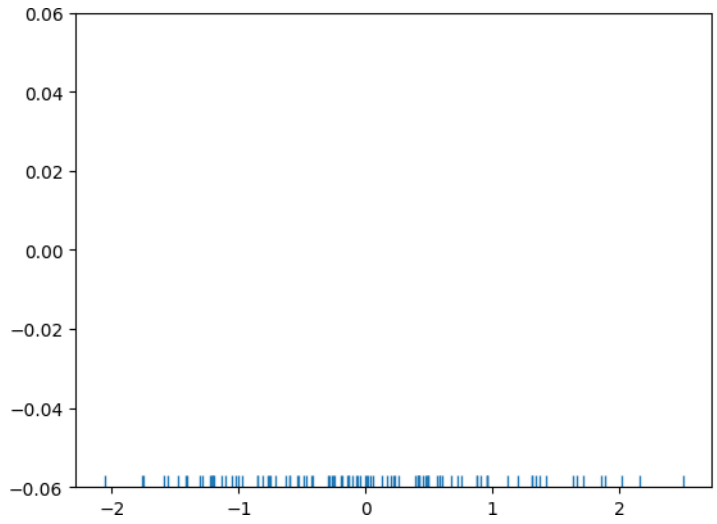
PROBLEM STATEMENT:

Implement a project using a data set and choose one of Linear Regression, Logistic Regression, Support Vector Machine (SVM), or k-Nearest Neighbors (KNN) models for analysis. Students are tasked with data pre-processing, model training, evaluation, and presenting findings, emphasizing practical application of the chosen model. Evaluation will consider the project's methodology, model performance, and the clarity of results.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

```
#create dataset
dataset=np.random.randn(100)

#create another rugplot
sns.rugplot(dataset);

#set up the x-axis for the plot
x_min=dataset.min()-2
x_max=dataset.max()+2

#100 equally spaced points from x_in to x_max
x_axis=np.linspace(x_min,x_max,100)
```

OUTPUT:




```
# Print the values of x_min and x_max
print(x_min,x_max)
```

OUTPUT:

```
-4.555218342250088 5.597136045854967
```

```
#set up the bandwidth,for info on this:
url='http://en.wikipedia.org/wiki/kernel_density_estimation#practical_estimation_of_the_bandwidth'
bandwidth=((4*dataset.std()**-0.5)/(3*len(dataset)))**0.2
bandwidth
```

OUTPUT:

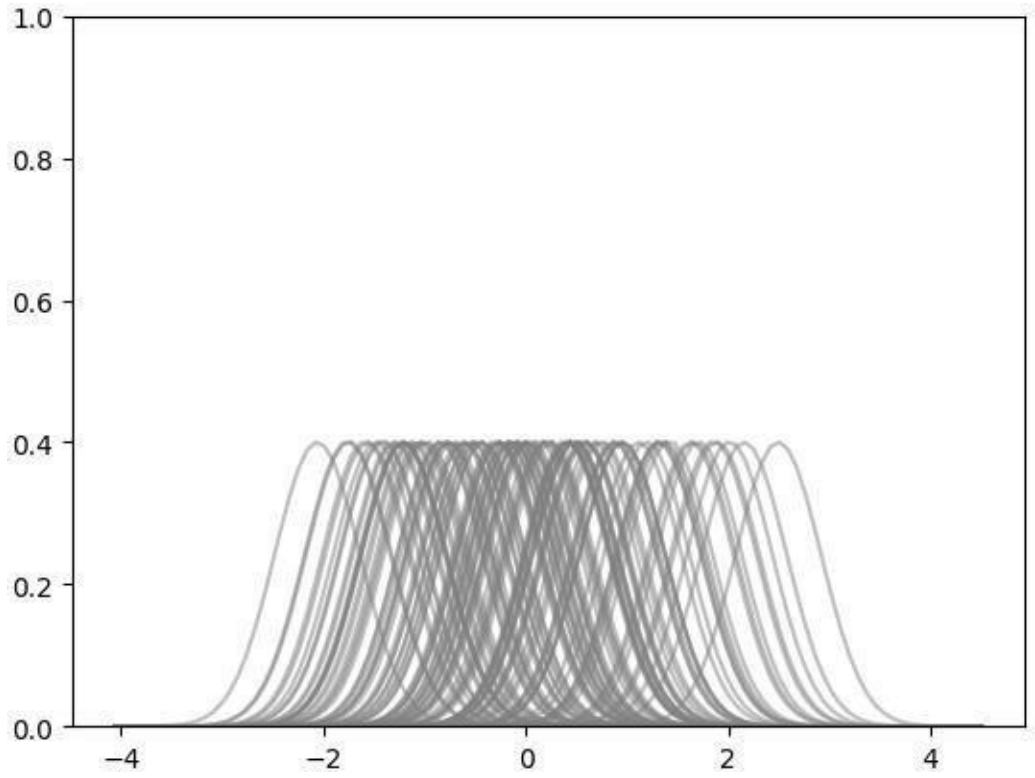
```
0.42197981688001757
```

```
#create an empty kernel list
kernel_list=[]

#plot each basis fn
for data_point in dataset:

  #create a kernel for each point and append to list
  kernel=stats.norm(data_point,bandwidth).pdf(x_axis)
  kernel_list.append(kernel)

  #scale for plotting
  kernel=kernel/kernel.max()
  kernel=kernel*.4
```

```
  plt.plot(x_axis,kernel,color='grey',alpha=0.5)

plt.ylim(0,1)
```

OUTPUT:

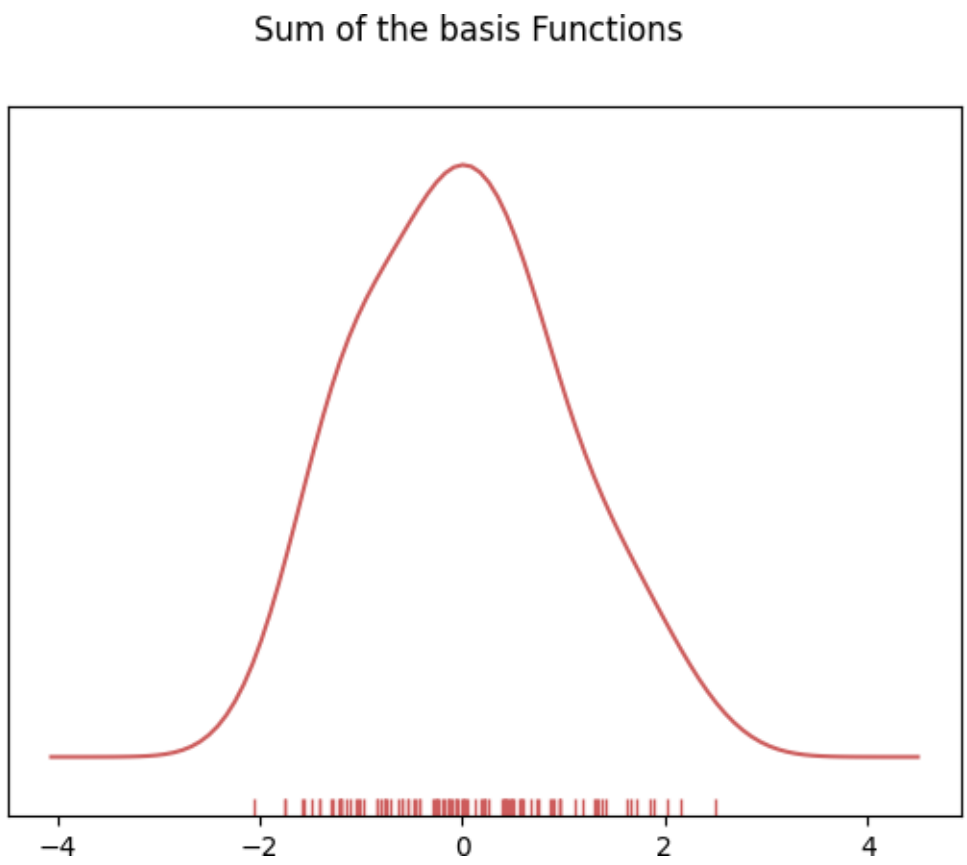
```
(0.0  , 1.0)
```




```
#to get the kde plot we can sum these basis fns.

#plot the sum of the basis fns.
sum_of_kde=np.sum(kernel_list,axis=0)

#plot figure
fig=plt.plot(x_axis,sum_of_kde,color='indianred')

#add the initial rugplot
sns.rugplot(dataset,c='indianred')
```

```
#get rid of y-tick marks
plt.yticks([])

#set title
plt.suptitle("Sum of the basis Functions")
```

OUTPUT:



REGULARIZATION:

```
# Import the required libraries
import numpy as np
import cv2
from sklearn.datasets import fetch_california_housing
from sklearn import metrics
from sklearn import model_selection
```

```
from sklearn import linear_model

# Enable inline plotting for Jupyter Notebook
%matplotlib inline

# Import the matplotlib library for data visualization
import matplotlib.pyplot as plt

# Set the plot style to 'ggplot'
plt.style.use('ggplot')

# Update the font size for the plots
plt.rcParams.update({'font.size': 16})
```

```
# Fetch the California housing dataset using scikit-learn's
fetch_california_housing function
housing=fetch california housing()
```

```
# Access the list of feature names in the housing dataset
dir(housing.feature_names)
#dir(housing.taret_names),.target,.DESCR,.fearure_names
```

OUTPUT:

```
['__add__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
```

```
'__gt__',
'__hash__',
'__iadd__',
'__imul__',
'__init__',
'__init_subclass__',
'__iter__',
'__le__',
'__len__',
'__lt__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__reversed__',
'__rmul__',
'__setattr__',
'__setitem__',
'__sizeof__',
'__str__',
'__subclasshook__',
'append',
'clear',
'copy',
'count',
'extend',
```

```
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

```
#used to determine the shape (dimensions) of the data within the
California housing dataset.
housing.data.shape
```

OUTPUT:

```
(20640, 8)
```

```
#used to determine the shape (dimensions) of the target variable in the
California housing dataset
housing.target.shape
```

OUTPUT:

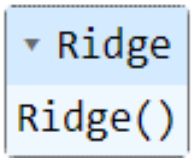
```
(20640,)
```

```
#ridgereg = linear_model.Ridge() creates an instance of the Ridge
regression model from scikit-learn's linear_model module.
ridgereg=linear_model.Ridge()
```

```
# Split the dataset into training and testing sets
x_train,x_test,y_train,y_test=model_selection.train_test_split(housing.
data,housing.target,test_size=0.3,random_state=42 )
```

```
# Train the Ridge regression model on the training data
ridgereg.fit(x_train,y_train)
```

OUTPUT:




```
#calculate the mean squared error (MSE) between the actual target
values (y_train) and the predicted values
metrics.mean_squared_error(y_train,ridgereg.predict(x_train))
```

OUTPUT:

```
0.5233577422311327
```

```
# The 'score()' method is used to calculate the R-squared value
ridgereg.score(x_train,y_train)
```

OUTPUT:

```
0.6093458881478931
```

```
#make predictions on the test data using the Ridge regression model
(ridgereg) that was previously trained.
y_pred=ridgereg.predict(x_test)
```

```
#calculate the mean squared error (MSE) between the actual target
values (y_test) and the predicted values
metrics.mean_squared_error(y_test,y_pred)
```

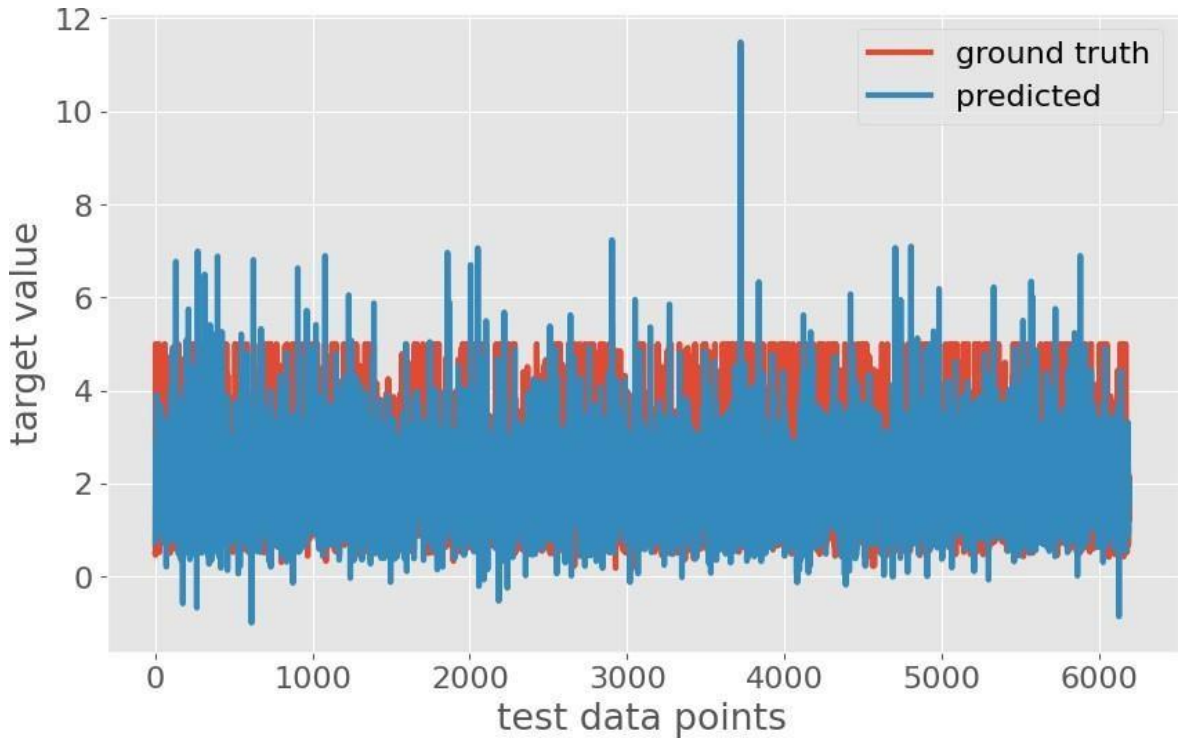
OUTPUT:

```
0.530505269093369
```

```
import matplotlib.pyplot as plt
```

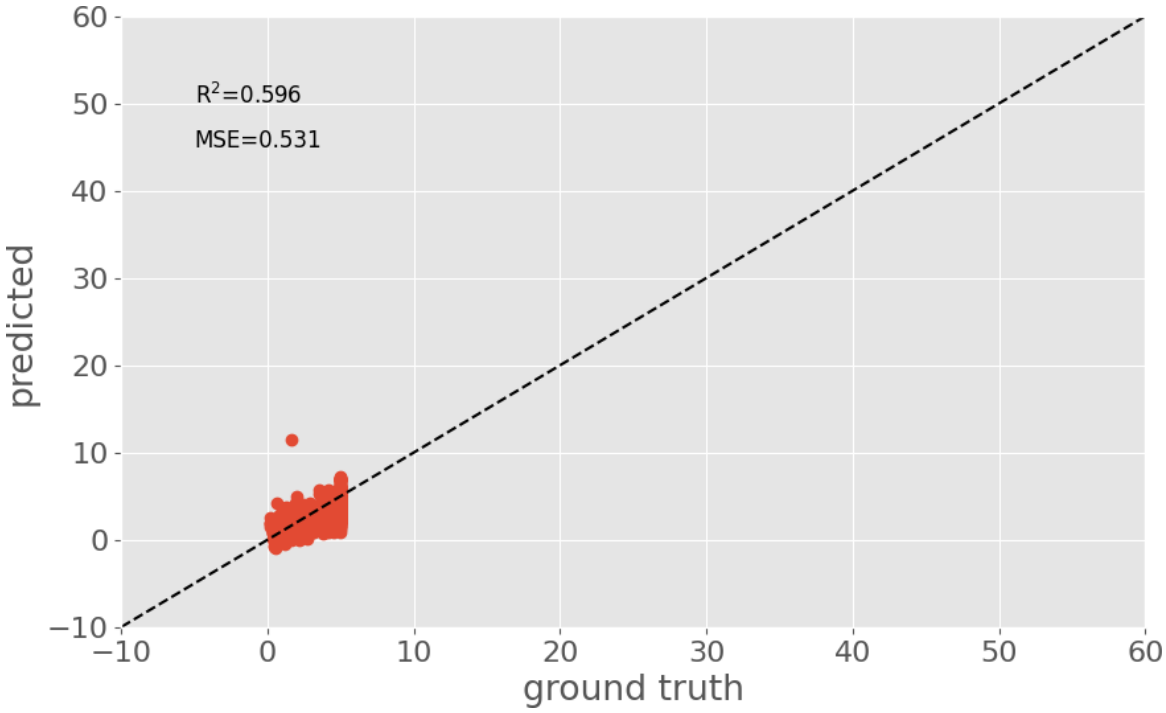
```
# Create a new figure for the plot with a specific size
plt.figure(figsize=(10, 6))

# Plot the actual target values (ground truth) with a thicker line and
label
plt.plot(y_test, linewidth=3, label='Ground Truth')

# Plot the predicted target values with a thicker line and label
plt.plot(y_pred, linewidth=3, label='Predicted')

# Add a legend to the plot, positioning it in the best available
location
plt.legend(loc='best')

# Label the x-axis to indicate it represents the test data points
plt.xlabel('Test Data Points')

# Label the y-axis to indicate it represents the target values
plt.ylabel('Target Value')

# Display the plot
plt.show()
```

OUTPUT:



```
# Create a new figure for the plot with a specific size
plt.figure(figsize=(10, 6))

# Create a scatter plot of ground truth vs. predicted values
plt.plot(y_test, y_pred, 'o')

# Add a diagonal dashed line to represent a perfect match (y = x)
plt.plot([-10, 60], [-10, 60], 'k--')

# Set the axis limits to ensure consistent scaling
plt.axis([-10, 60, -10, 60])

# Label the x-axis as 'ground truth' and the y-axis as 'predicted'
plt.xlabel('Ground Truth')
plt.ylabel('Predicted')

# Create a string with the R-squared (coefficient of determination)
value
scorestr = r'R$^2$=%.3f' % ridgereg.score(x_test, y_test)

# Create a string with the mean squared error (MSE) value
errstr = 'MSE=%.3f' % metrics.mean_squared_error(y_test, y_pred)

# Display the R-squared and MSE values as text on the plot
plt.text(-5, 50, scorestr, fontsize=12)
plt.text(-5, 45, errstr, fontsize=12)

# Show the plot
plt.show()
```

OUTPUT:


R^2=0.596
MSE=0.531
predicted
ground truth
60
50
40
30
20
10
0
−10
−10
0
10
20
30
40
50
60

LAB-06

DESCRIPTION:

Implementation of dimensionality reduction using Principal Component Analysis(PCA)

PROBLEM STATEMENT:

Implement Logistic Regression using a predefined library for a binary classification task. Analyze the impact of varying training and testing split ratios (e.g., 70/30, 80/20, 90/10) on model performance, such as accuracy and precision-recall metrics, to determine the optimal data split stratergy.

```
# For data manipulation
import pandas as pd

# For numerical operations
import numpy as np

# For PCA (Principal Component Analysis)
from sklearn.decomposition import PCA

# For data preprocessing
from sklearn import preprocessing

# For data visualization
import matplotlib.pyplot as plt
```

```
# Provide a correct URL or local file path for the Iris dataset
url="/content/archive.ics.uci.edu_ml_machine-learning-
databases_iris_iris.data.csv"

# Read the dataset from the specified URL with column names
df=pd.read_csv(url,names=["sepal length","sepal width","petal
length","petal width","class"])
```

```
#display the table
df
```

OUTPUT:

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```
# Import the necessary library for data preprocessing
from sklearn.preprocessing import StandardScaler

# Define the feature columns
features = ['sepal length', 'sepal width', 'petal length', 'petal
width']

# Extract the feature data from the DataFrame
x = df.loc[:, features].values

# Extract the target data from the DataFrame
```

```
y = df.loc[:, ['class']].values

# Standardize the feature data using StandardScaler
x = StandardScaler().fit_transform(x)
```

```
# Specify the number of principal components to retain (in this case,
2)
# Create a PCA object with the desired number of components
pca=PCA(n_components=2)

# Fit the PCA model to the standardized feature data and transform it
principalComponents=pca.fit_transform(x)

# Create a new DataFrame to store the principal components
principalDataframe=pd.DataFrame(data=principalComponents,columns=['PC1'
,'PC2'])
```

```
# Extract the target variable (class) from the original DataFrame
targetDataframe = df[['class']]

# Concatenate the principal components and the target variable along
the columns (axis=1)
newDataframe = pd.concat([principalDataframe, targetDataframe], axis=1)
```

```
#print newDataframe
newDataframe
```

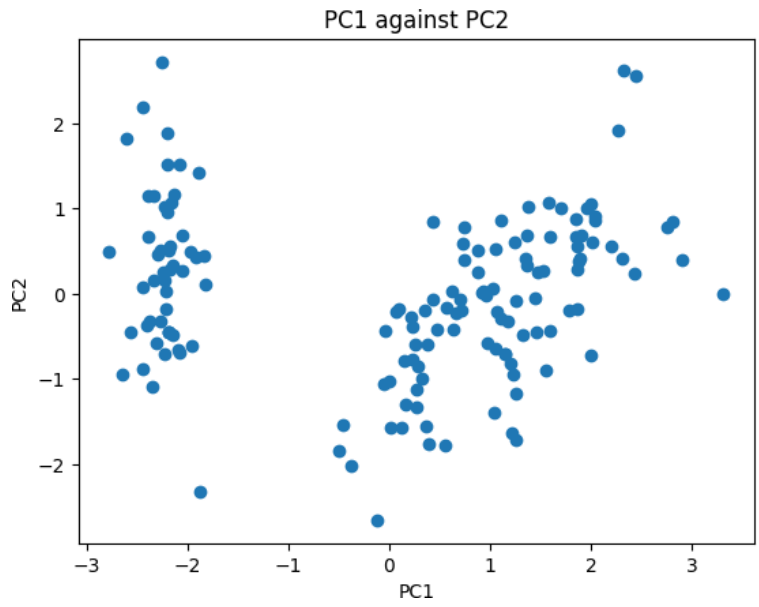
OUTPUT:

	PC1	PC2	class
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa
...	...	...	...
145	1.870522	0.382822	Iris-virginica
146	1.558492	-0.905314	Iris-virginica
147	1.520845	0.266795	Iris-virginica
148	1.376391	1.016362	Iris-virginica
149	0.959299	-0.022284	Iris-virginica

150 rows × 3 columns

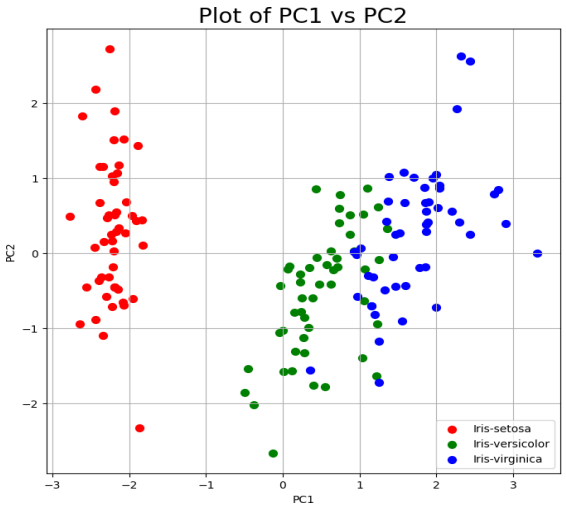
```
# Create a scatter plot of PC1 against PC2
plt.scatter(principalDataframe.PC1, principalDataframe.PC2)

# Add a title to the plot
plt.title('PC1 against PC2')

# Label the x-axis as PC1
plt.xlabel('PC1')

# Label the y-axis as PC2
plt.ylabel('PC2')
```

OUTPUT:



```
# Create a figure with a specified size
fig = plt.figure(figsize=(8, 8))

# Add a subplot to the figure
ax = fig.add_subplot(1, 1, 1)

# Set labels for the x and y axes
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')

# Set the title of the plot
ax.set_title('Plot of PC1 vs PC2', fontsize=20)

# Define the target classes and their corresponding colors
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']

# For each target class and color, scatter plot the data points
for target, color in zip(targets, colors):
    # Filter the data for the current target class
    indicesToKeep = newDataframe['class'] == target
    ax.scatter(newDataframe.loc[indicesToKeep, 'PC1'],
               newDataframe.loc[indicesToKeep, 'PC2'],
               c=color,
               s=50)

# Add a legend to the plot with the target class labels
ax.legend(targets)

# Display a grid in the plot
ax.grid()
```

OUTPUT:



```
# print pca.explained_variance_ratio_
pca.explained_variance_ratio_
```

OUTPUT:

```
array([0.72770452, 0.23030523])
```

LAB-07

DESCRIPTION:

Implementation of K-Means clustering using synthetic data set and implementation of Gaussian Mixture Model.

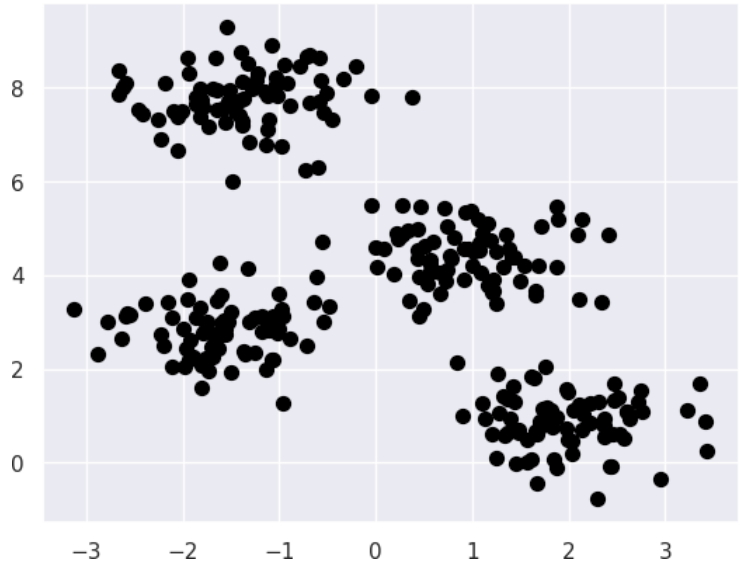
PROBLEM STATEMENT:

Implement Support Vector Machines (SVM) for classification and Support Vector Regression (SVR) for regression tasks. Students should build, evaluate, and compare SVM models for classification and SVR models for regression using real-world datasets, emphasizing their practical applications and performance assessment.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()#plot styling
import numpy as np
```

```
# Import the make_blobs function for generating synthetic data
from sklearn.datasets import make_blobs

# Generate a synthetic dataset
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)

# Create a scatter plot to visualize the data
plt.scatter(X[:, 0], X[:, 1], s=50, color='black')

# Display the scatter plot
plt.show()
```

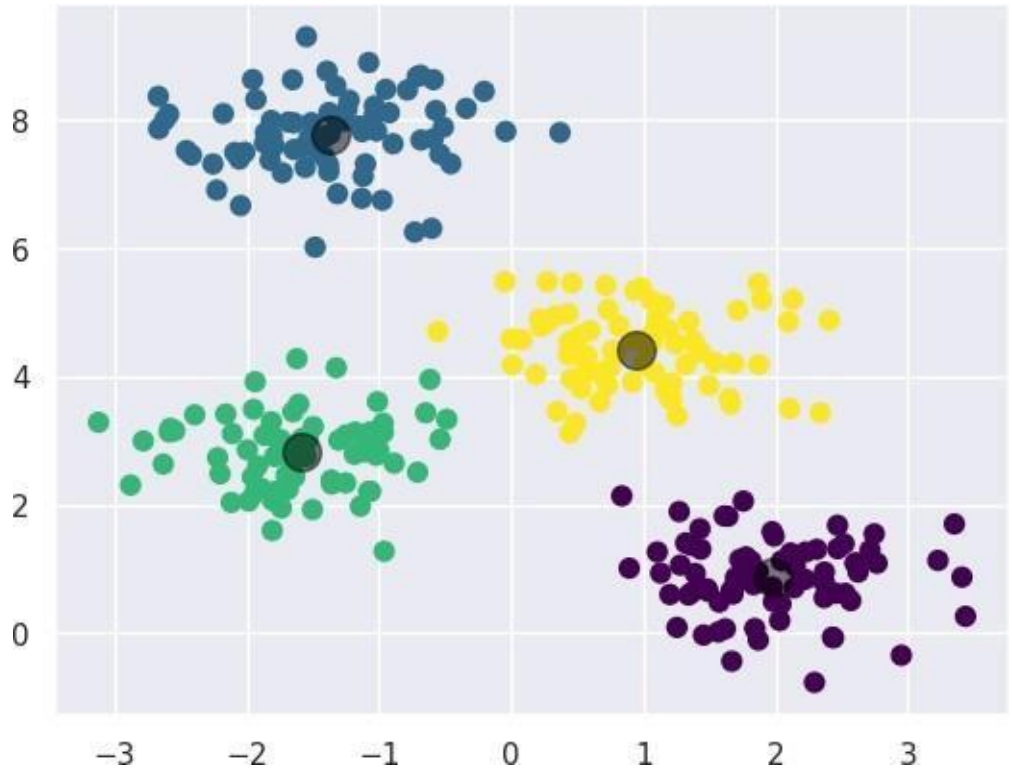
OUTPUT:




```
# Import the necessary libraries
from sklearn.cluster import KMeans # Import the KMeans clustering
algorithm

# Create a K-Means clustering model
kmeans = KMeans(n_clusters=4, n_init=10)

# Fit the K-Means model to the data
kmeans.fit(X)
# - X: The data points to be clustered

# Predict the cluster labels for each data point
y_kmeans = kmeans.predict(X)
```

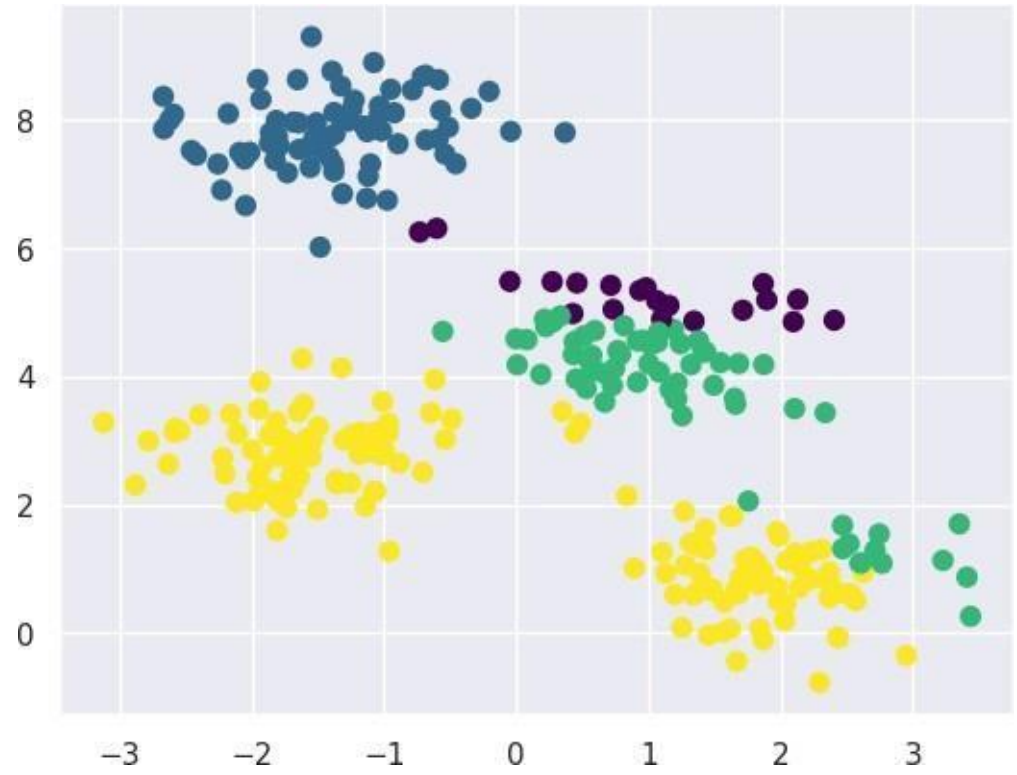
```
# Create a scatter plot of data points with cluster assignments
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

# Get the cluster centers
centers = kmeans.cluster_centers_

# Plot the cluster centers on the scatter plot
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```

OUTPUT:



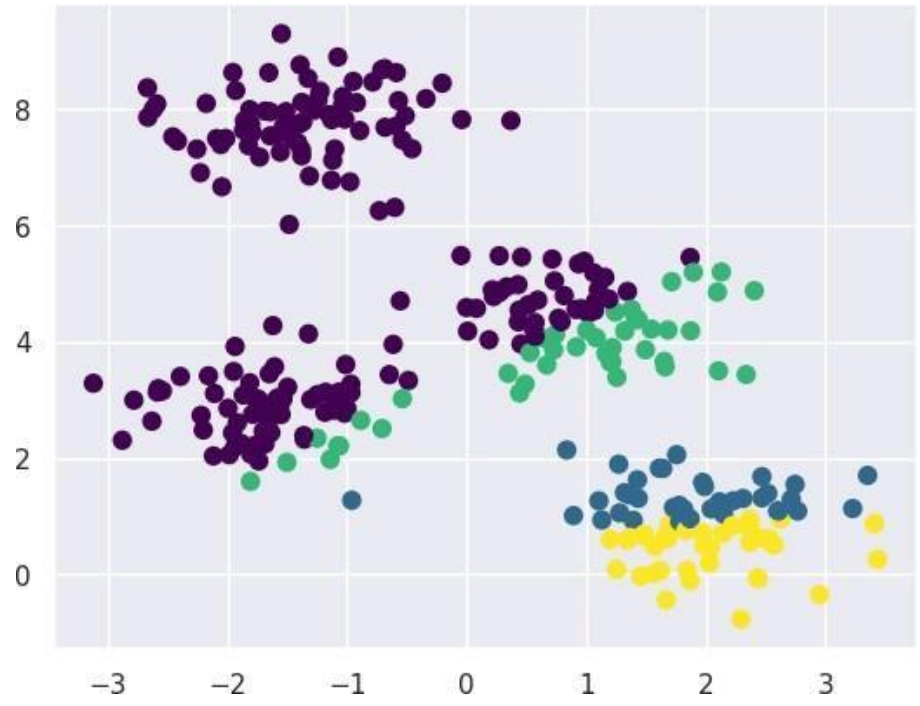

```
from sklearn.metrics import pairwise_distances_argmin
def find_clusters(X, n_clusters, rseed=2):
  # 1. Randomly choose clusters
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]

    while True:
        # 2a. Assign labels based on closest center
        labels = pairwise_distances_argmin(X, centers)

       # 2b. Find new centers from means of points
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])

    # 2c. Check for convergence
        if np.all(centers == new_centers):
           break
        centers = new_centers
        return centers, labels
centers,labels=find_clusters(X,4)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis')
```

OUTPUT:



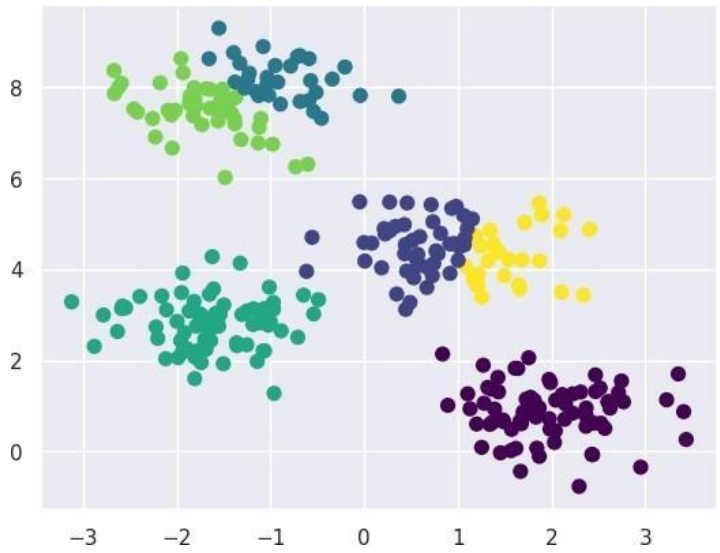

```
centers,labels=find_clusters(X,4,rseed=0)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis')
```

OUTPUT:



```
labels=KMeans(6,random_state=0,n_init=10).fit_predict(X)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis');
```

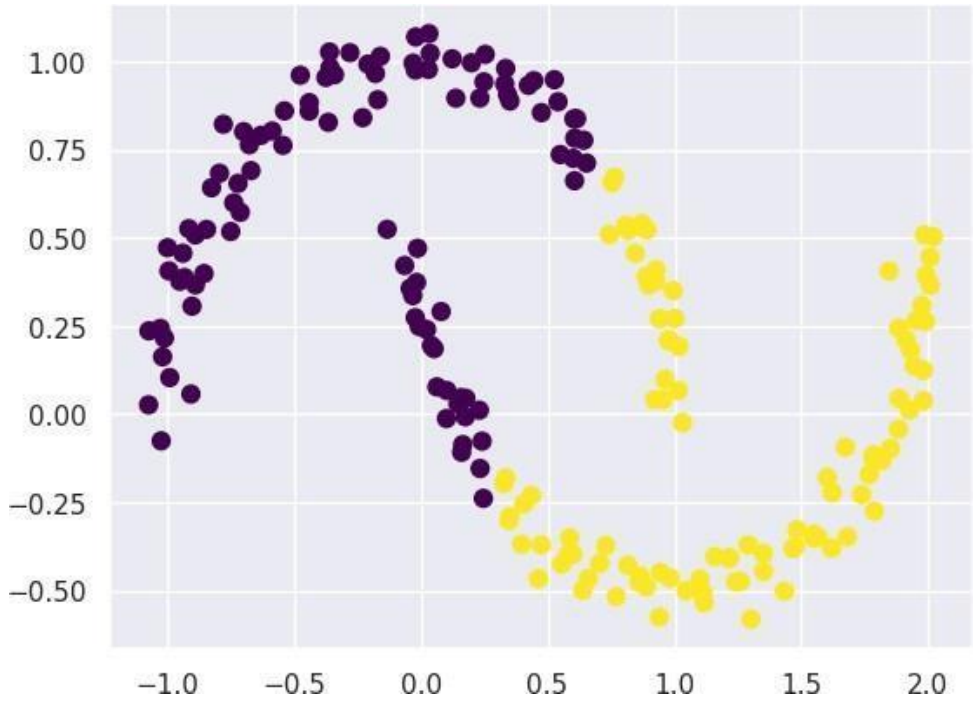
OUTPUT:




```
from sklearn.datasets import make_moons
X,y=make_moons(200,noise=.05,random_state=0)
```

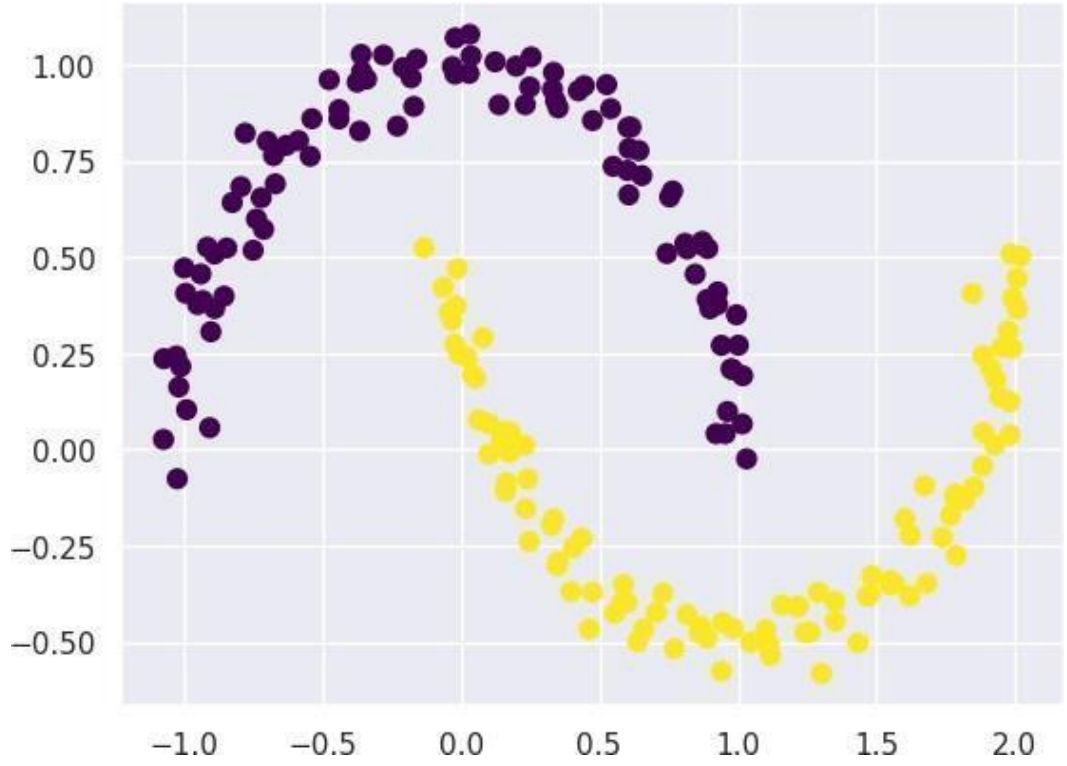
```
labels=KMeans(2,random_state=0,n_init=10).fit_predict(X)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis');
```

OUTPUT:



```
from sklearn.cluster import SpectralClustering
model=SpectralClustering(n_clusters=2,affinity='nearest_neighbors',assi
gn_labels='kmeans')
labels=model.fit_predict(X)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis')
```

OUTPUT:



LAB-08

DESCRIPTION:

Implementation of Gaussian Mixture Model using syntheic data set.

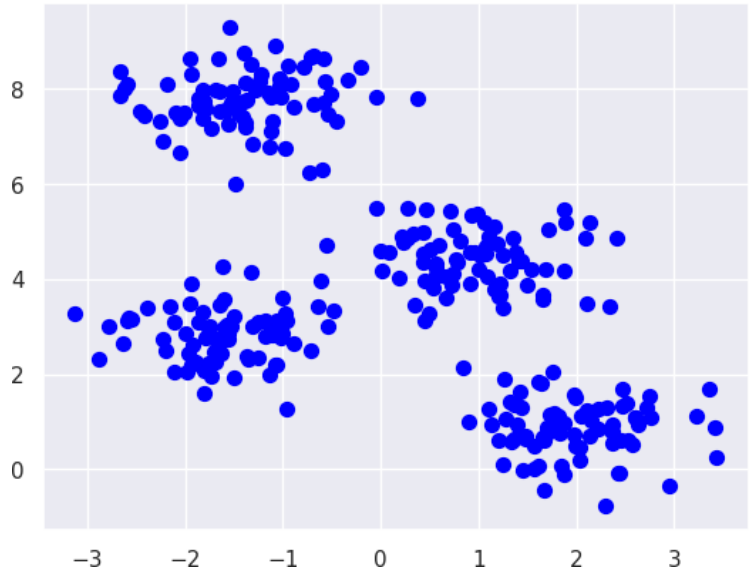
PROBLEM STATEMENT:

Implement L2 regularization using a predefined library (e.g., Scikit-Learn) and compare its impact on regression models against ordinary (non-regularized) regression. Analyze how regularization affects model performance, overfitting, and the ability to handle multicollinearity. Provide a comprehensive evaluation of the results and insights into when and why L2 regularization is beneficial.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()#plot styling
import numpy as np
```

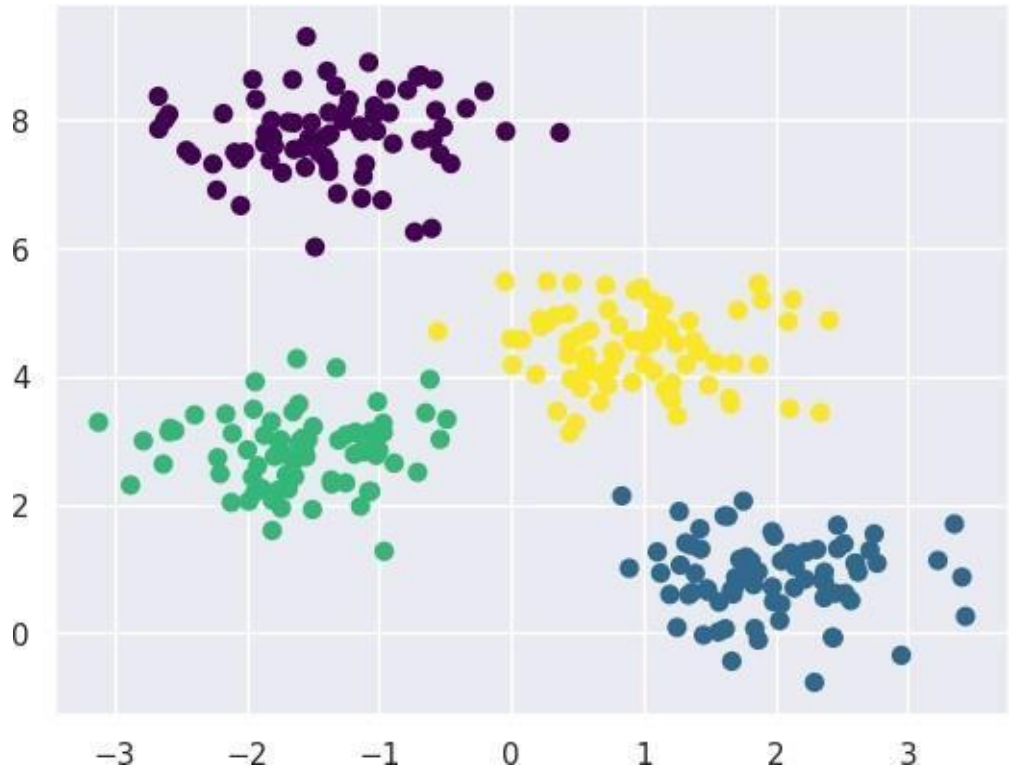
```
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate synthetic data points with 4 clusters
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)

# Scatter plot of the generated data points with blue color
plt.scatter(X[:, 0], X[:, 1], s=50, color='blue')

# Show the plot
plt.show()
```

OUTPUT:




```
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Create a Gaussian Mixture Model with 4 components and fit it to the
data
gmm = GaussianMixture(n_components=4).fit(X)

# Predict the cluster labels for each data point
labels = gmm.predict(X)

# Scatter plot of the data points with different colors representing
the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')

# Show the plot
plt.show()
```

OUTPUT:



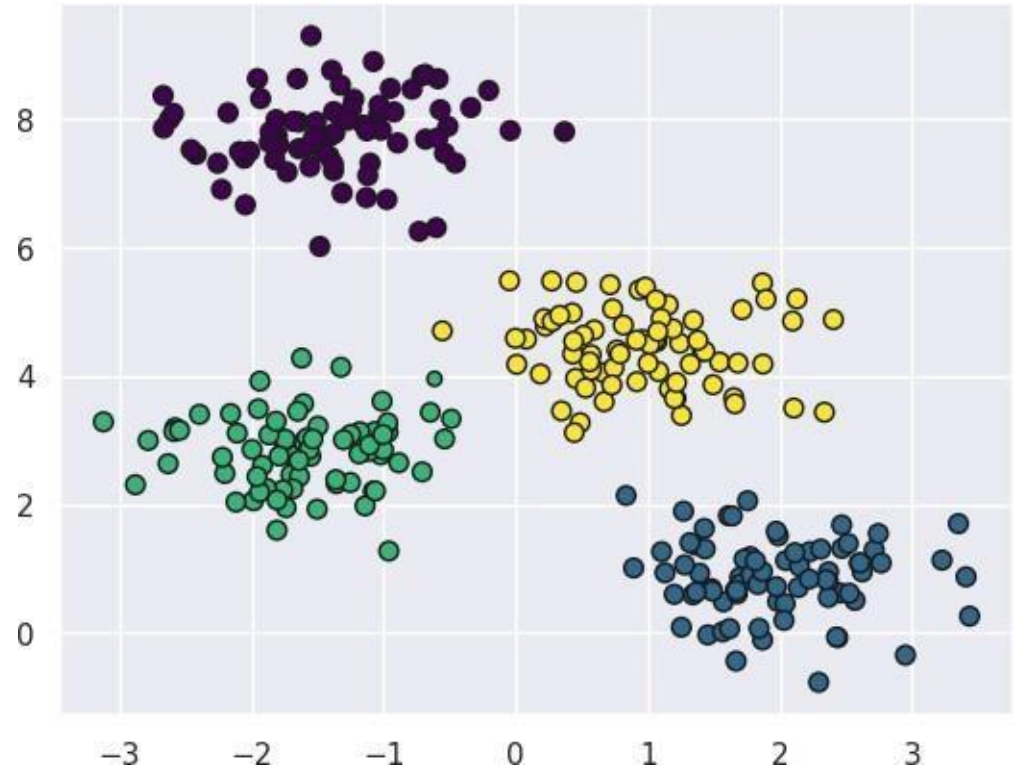

```
# Predict the probabilities of data points belonging to each cluster
probs = gmm.predict_proba(X)

# Print the probabilities for the first 5 data points, rounded to 3
decimal places
print(probs[:5].round(3))
```

OUTPUT:

```
[[0.    0.972 0.002 0.026]
 [1.    0.    0.    0.   ]
 [0.    0.    0.    1.   ]
 [1.    0.    0.    0.   ]
 [0.    0.999 0.    0.001]]
```

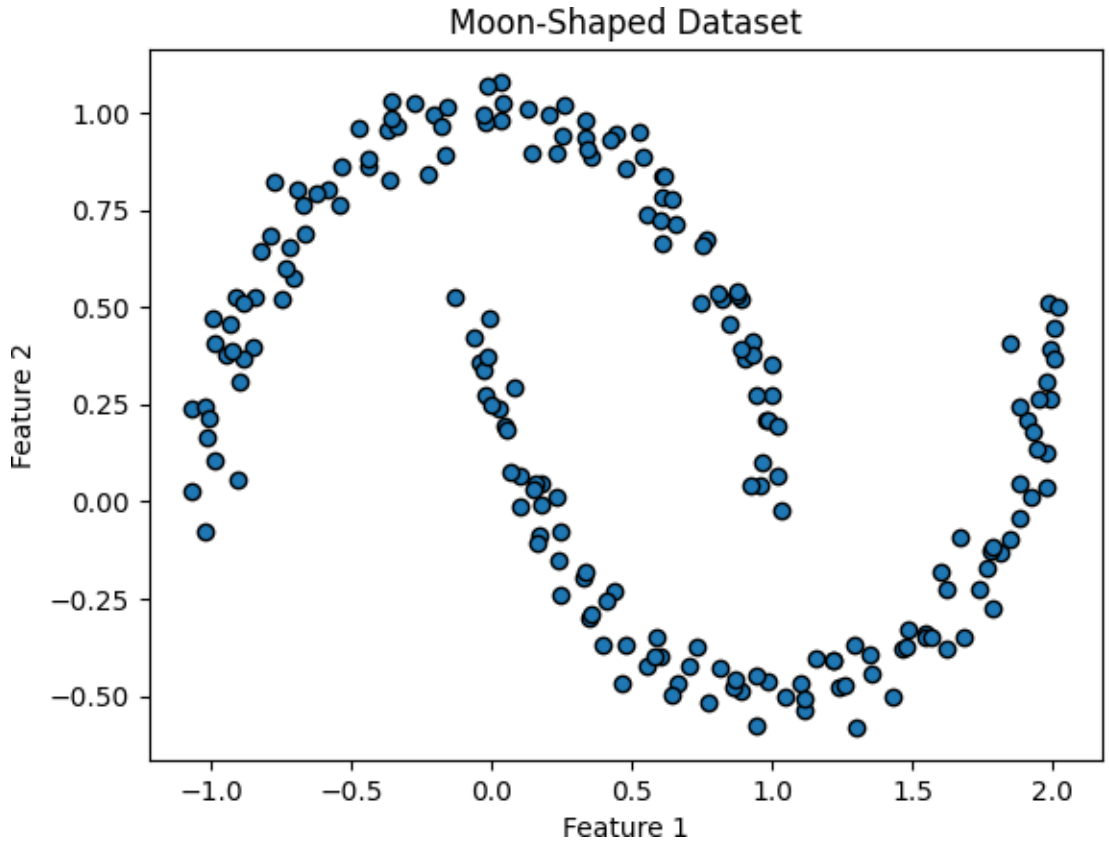
```
#print(probs.max(1))
size=probs.max(1)/0.02 #square emphasizes differences
plt.scatter(X[:,0],X[:,1],c=labels,edgecolor='k',cmap='viridis',s=size)
;
```

OUTPUT:




```
# Import necessary libraries
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

# Generate a moon-shaped dataset with 200 samples and some noise
Xmoon, ymoon = make_moons(200, noise=0.05, random_state=0)

# Create a scatter plot of the moon-shaped dataset
plt.scatter(Xmoon[:, 0], Xmoon[:, 1], edgecolor='k')

# Add labels and show the plot
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Moon-Shaped Dataset')
plt.show()
```

OUTPUT:

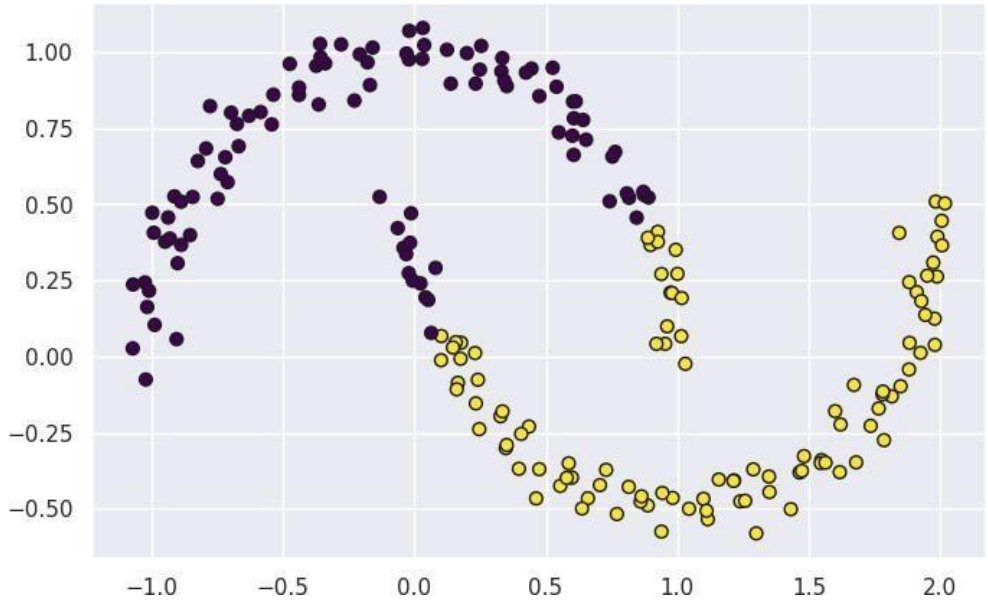



```
from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
 #Draw an ellipse with a given position and covariance"""
 ax = ax or plt.gca()
 # Convert covariance to principal axes
 if covariance.shape == (2, 2):
   U, s, Vt = np.linalg.svd(covariance)
   angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
   width, height = 2 * np.sqrt(s)
 else:
   angle = 0
   width, height = 2 * np.sqrt(covariance)
 # Draw the Ellipse
 for nsig in range(1, 4):
   ax.add_patch(Ellipse(position, nsig * width, nsig * height,angle,
**kwargs))
def plot_gmm(gmm, X, label=True, ax=None):
  ax = ax or plt.gca()
```

```
  labels = gmm.fit(X).predict(X)
  if label:
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=40,
cmap='viridis',zorder=2,edgecolor='k')
  else:
     ax.scatter(X[:, 0], X[:, 1], s=40,
zorder=2,cmap='viridis',edgecolor='k')
     ax.axis('equal')
     w_factor = 0.2 / gmm.weights_.max()
     for pos, covar, w in zip(gmm.means_, gmm.covariances_,
gmm.weights_):
          draw_ellipse(pos, covar, alpha=w * w_factor)
```

```
# Import necessary libraries
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt

# Create a Gaussian Mixture Model (GMM) with 2 components
gmm2 = GaussianMixture(n_components=2, covariance_type='full',
random_state=0)

# Set the figure size for the plot
plt.figure(figsize=(8, 5))

# Call the 'plot_gmm' function to visualize the GMM clustering
plot_gmm(gmm2, Xmoon)
```

OUTPUT:



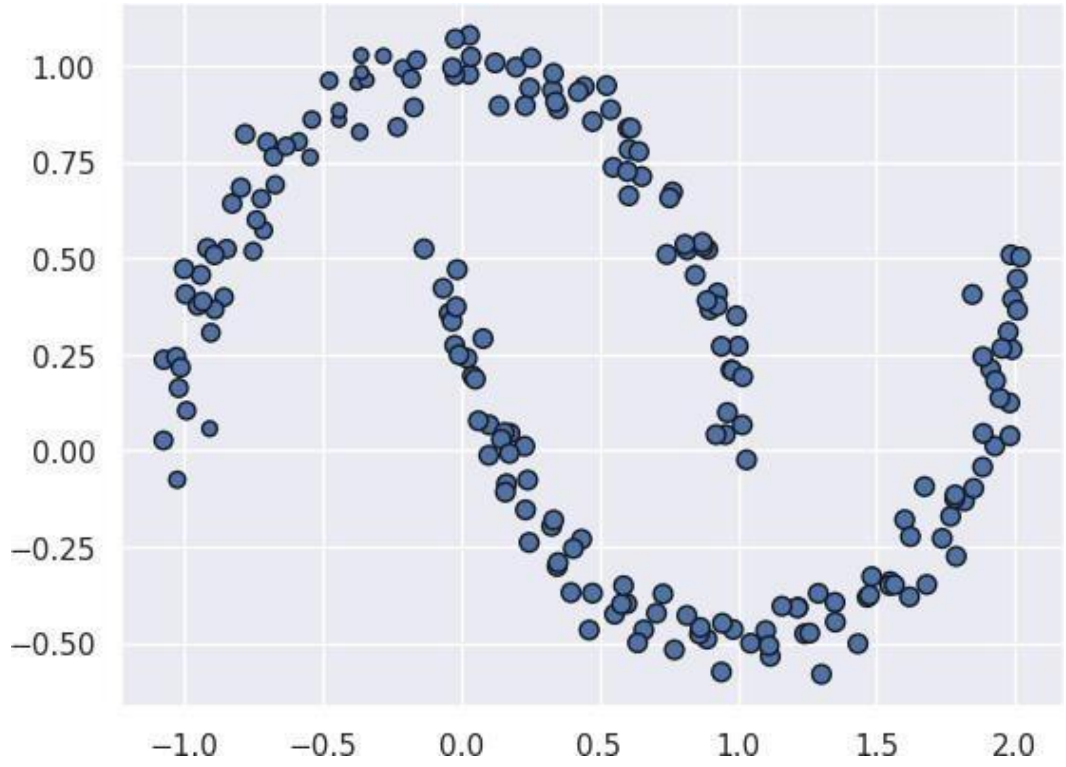
```
# Calculate the probabilities of each data point belonging to each
component
probs=gmm.predict_proba(Xmoon)

# Print the rounded probabilities for the first 5 data points
print(probs[:5].round(3))
```

OUTPUT:

```
[[0.    1.    0.    0.   ]
 [0.    1.    0.    0.   ]
 [0.    1.    0.    0.   ]
 [0.    0.259 0.741 0.   ]
 [0.    1.    0.    0.   ]]
```

```
#print(probs.max(1))
size=probs.max(1)/0.02 #square emphasizes differences
plt.scatter(Xmoon[:,0],Xmoon[:,1],edgecolor='k',s=size);
```

OUTPUT:



LAB-09

DESCRIPTION:

Implementation of Support Vector Machine Classification using breast cancer data set.

PROBLEM STATEMENT:

Implement L1 regularization using a predefined library (e.g., scikit-learn) and compare its results with ordinary regression techniques. Assess the impact of regularization on model performance, feature selection, and the trade-offs between bias and variance in the context of linear regression.

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

```
from sklearn.datasets import load_breast_cancer
```

```
# Load the breast cancer dataset
cancer=load_breast_cancer()
```

```
#list of available keys.
cancer.keys()
```

OUTPUT:

```
dict_keys(['data', 'target', 'frame', 'target_names',
'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
#describe the dataset.
print(cancer['DESCR'])
```

OUTPUT:

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign
```

SUMMARY STATISTICS

```
==================================== ====== ======
                                      Min    Max
==================================== ====== ======
radius (mean):                       6.981  28.11
texture (mean):                      9.71   39.28
perimeter (mean):                    43.79  188.5
area (mean):                         143.5  2501.0
smoothness (mean):                   0.053  0.163
compactness (mean):                  0.019  0.345
concavity (mean):                    0.0    0.427
concave points (mean):               0.0    0.201
symmetry (mean):                     0.106  0.304
fractal dimension (mean):            0.05   0.097
radius (standard error):             0.112  2.873
texture (standard error):            0.36   4.885
perimeter (standard error):          0.757  21.98
area (standard error):               6.802  542.2
smoothness (standard error):         0.002  0.031
compactness (standard error):        0.002  0.135
concavity (standard error):          0.0    0.396
concave points (standard error):     0.0    0.053
symmetry (standard error):           0.008  0.079
fractal dimension (standard error):  0.001  0.03
radius (worst):                      7.93   36.04
texture (worst):                     12.02  49.54
perimeter (worst):                   50.41  251.2
area (worst):                        185.2  4254.0
smoothness (worst):                  0.071  0.223
compactness (worst):                 0.027  1.058
concavity (worst):                   0.0    1.252
concave points (worst):              0.0    0.291
symmetry (worst):                    0.156  0.664
fractal dimension (worst):           0.055  0.208
==================================== ====== ======
```

```
    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:



ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

```
#list of the names of the features (attributes) in the dataset.
cancer['feature_names']
```

OUTPUT:

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

```
#creates a DataFrame df using the dataset's data and feature names.
df=pd.DataFrame(cancer['data'],columns=cancer['feature_names'])

#information about the DataFrame's structure and contents, including
data types and non-null values for each column.
df.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
 14  smoothness error        569 non-null    float64
 15  compactness error       569 non-null    float64
 16  concavity error         569 non-null    float64
 17  concave points error    569 non-null    float64
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
```

```
#summary statistics for the columns in the DataFrame
df.describe()
```

OUTPUT:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190	25.677223	107.261213	880.583128
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.146258	33.602542	569.356993
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.020000	50.410000	185.200000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.080000	84.110000	515.300000
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.410000	97.660000	686.500000
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.720000	125.400000	1084.000000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000	49.540000	251.200000	4254.000000

worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
0.132369	0.254265	0.272188	0.114606	0.290076	0.083946
0.022832	0.157336	0.208624	0.065732	0.061867	0.018061
0.071170	0.027290	0.000000	0.000000	0.156500	0.055040
0.116600	0.147200	0.114500	0.064930	0.250400	0.071460
0.131300	0.211900	0.226700	0.099930	0.282200	0.080040
0.146000	0.339100	0.382900	0.161400	0.317900	0.092080
0.222600	1.058000	1.252000	0.291000	0.663800	0.207500

```
#calculates and returns the total count of missing (null) values in the
DataFrame df.
np.sum(pd.isnull(df).sum())
```

OUTPUT:

0(ZERO)

```
#Print target
cancer['target']
```

OUTPUT:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```
#calculates and returns the sum of the elements in the 'target' array
of the breast cancer dataset
cancer['target'].sum()
```

OUTPUT:

357

```
df['Cancer']=pd.DataFrame(cancer['target'])
df.head()
```

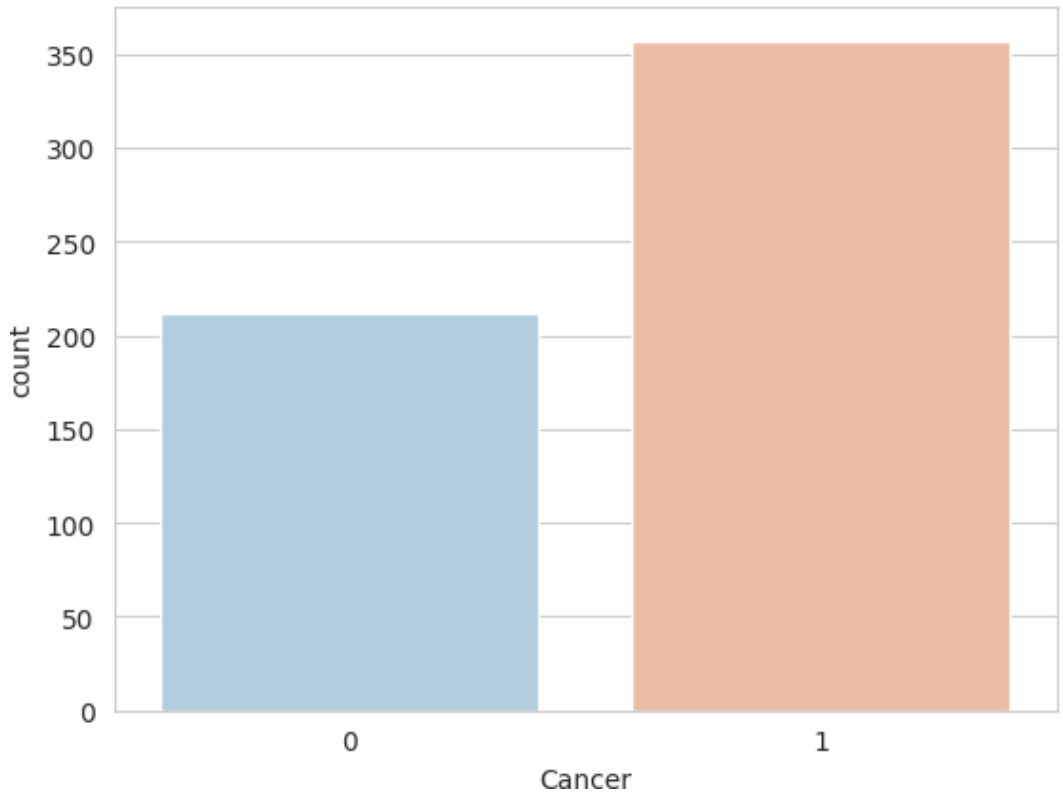
OUTPUT:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0

worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	Cancer
0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0
0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

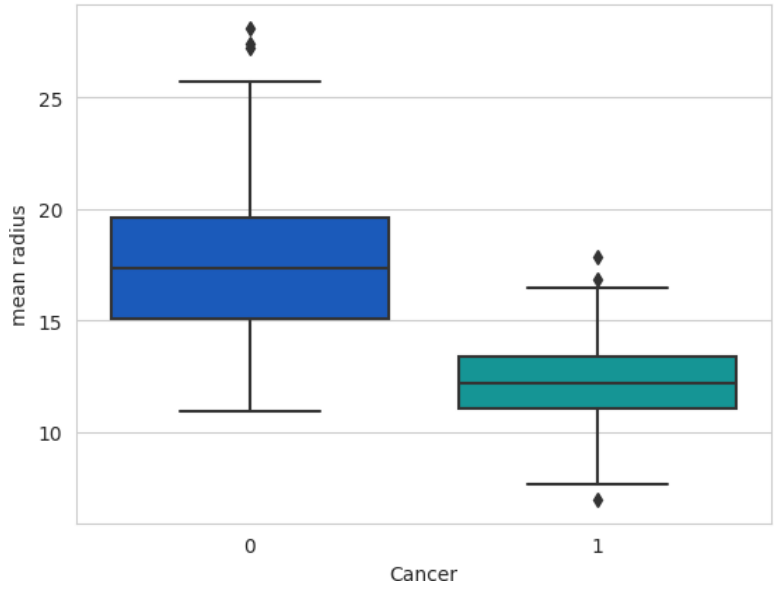
```
#countplot
sns.set_style('whitegrid')
sns.countplot(x='Cancer',data=df,palette='RdBu_r')
```

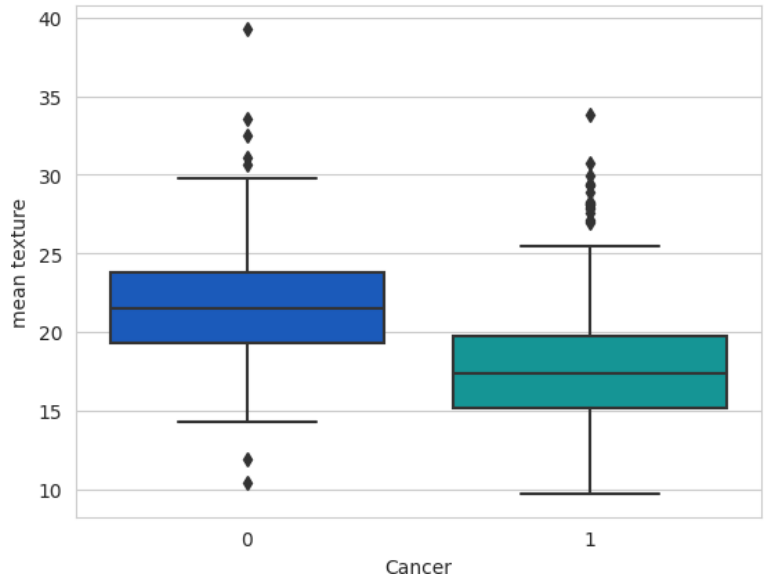
OUTPUT:

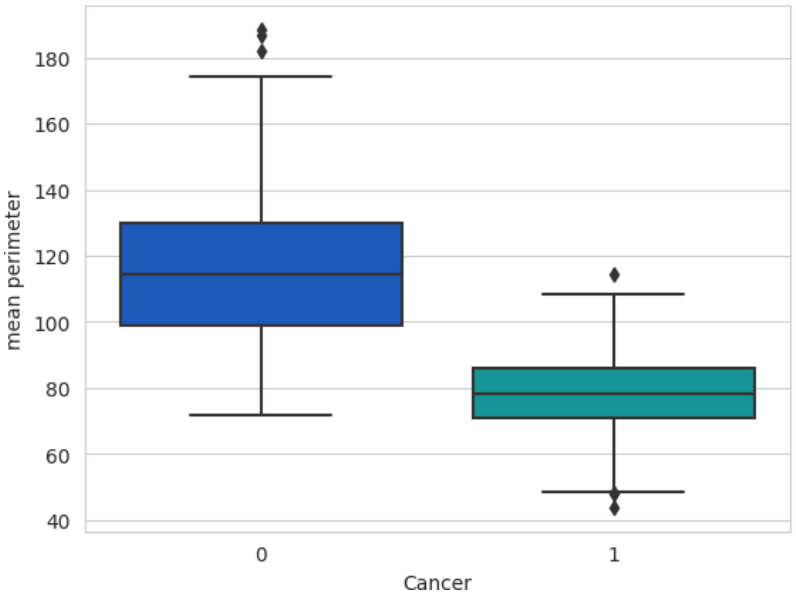


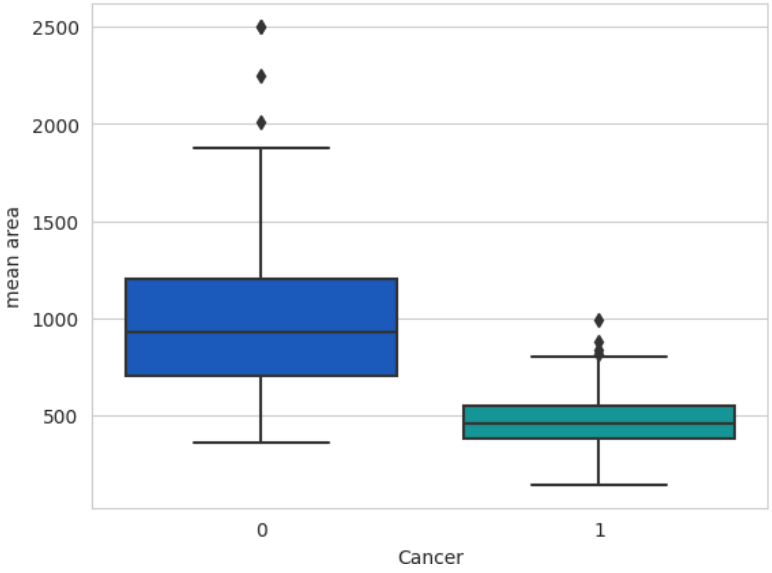
```
# List of column names (features) to create boxplots for
l = list(df.columns[0:10])

# Iterate through the list of features
for i in range(len(l) - 1):
    # Create a boxplot for the current feature
    sns.boxplot(x='Cancer', y=l[i], data=df, palette='winter')

    # Display the current boxplot
    plt.show()
```

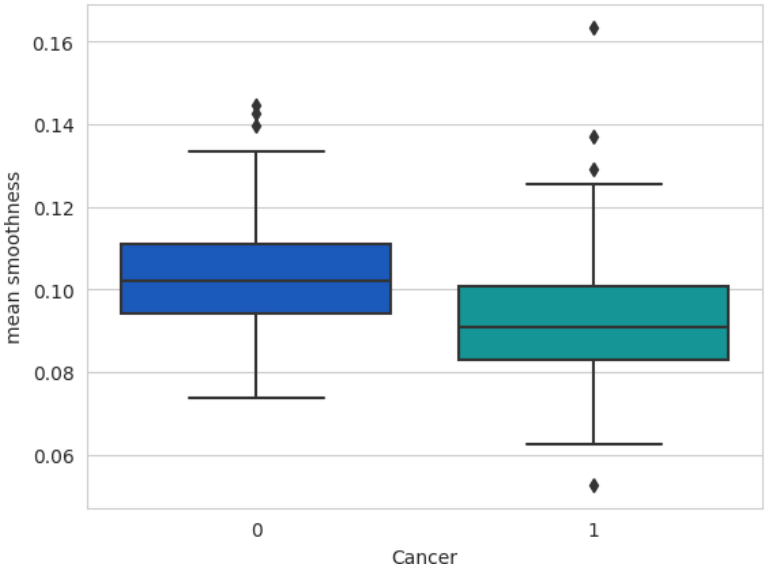
OUTPUT:

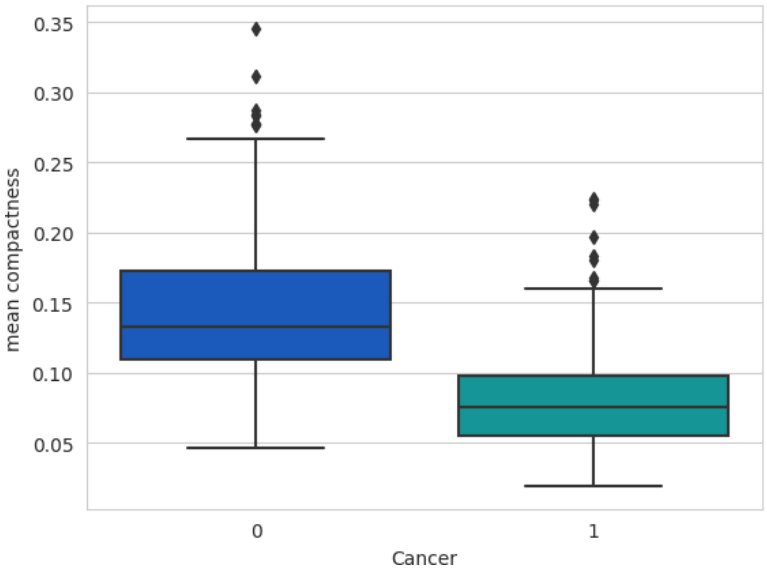


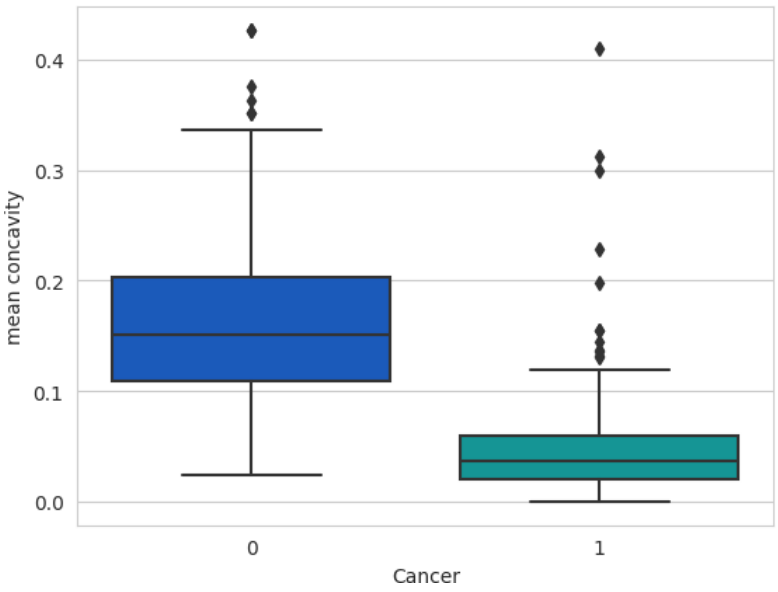


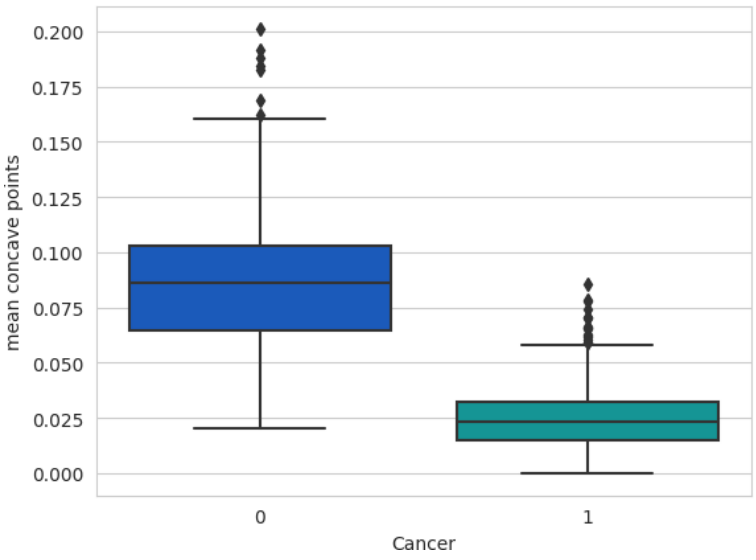

mean perimeter
180
160
140
120
100
80
60
40
0
1
Cancer

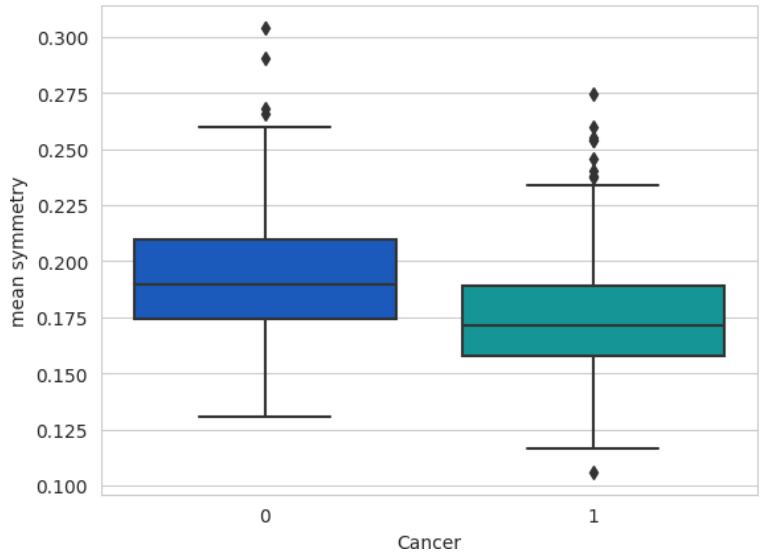

mean area
2500
2000
1500
1000
500
0
1
Cancer


mean smoothness
0.16
0.14
0.12
0.10
0.08
0.06
0
1
Cancer


mean compactness
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
1
Cancer

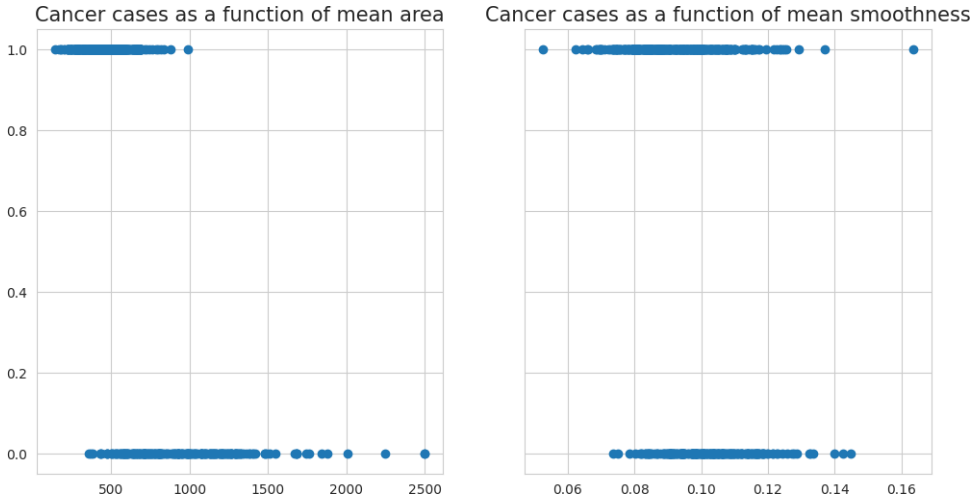

mean concavity
0.4
0.3
0.2
0.1
0.0
0
1
Cancer


mean concave points
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
1
Cancer




```
# Create a figure with two subplots (1 row, 2 columns)
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(12, 6))

# Create a scatter plot for the 'mean area' feature
ax1.scatter(df['mean area'], df['Cancer'])
ax1.set_title("Cancer cases as a function of mean area", fontsize=15)

# Create a scatter plot for the 'mean smoothness' feature
ax2.scatter(df['mean smoothness'], df['Cancer'])
ax2.set_title("Cancer cases as a function of mean smoothness",
fontsize=15)

# Display the subplots
plt.show()
```

OUTPUT:



```
# Create a new DataFrame df_feat by dropping the 'Cancer' column
df_feat = df.drop('Cancer', axis=1)

# Display the first few rows of the new DataFrame
df_feat.head()
```

OUTPUT:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67

5 rows × 30 columns

worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

```
# Create a new DataFrame df_target that contains the 'Cancer' column
df_target = df['Cancer']

# Display the first few rows of the new DataFrame
df_target.head()
```

OUTPUT:

```
0    0
1    0
2    0
3    0
4    0
Name: Cancer, dtype: int64
```

```
from sklearn.model_selection import train_test_split
```

```
# Split the dataset into training and testing sets
# X_train: Training feature set
# X_test: Testing feature set
# y_train: Training target variable
# y_test: Testing target variable
# test_size: Specifies the percentage of the data to be used for
testing (in this case, 30%)
# random_state: A random seed for reproducibility
X_train,X_test,y_train,y_test=train_test_split(df_feat,df_target,test_s
ize=0.30,random state=101)
```

```
#To display the first few rows of the y_train target variable
y_train.head()
```

OUTPUT:

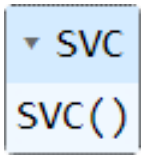
```
178    1
421    1
57     0
514    0
548    1
Name: Cancer, dtype: int64
```

```
from sklearn.svm import SVC
```

```
# Initialize an SVC model
model=SVC()
```

```
# Fit the SVC model to the training data
model.fit(X_train,y_train)
```

OUTPUT:



```
# Use the trained SVC model to make predictions on the test data
predictions=model.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
# Compute the confusion matrix using the true target labels (y_test)
and the predicted labels (predictions)
print(confusion_matrix(y_test,predictions))
```

OUTPUT:

```
[[ 56  10]
 [  3 102]]
```

```
# Generate a classification report using the true target labels
(y_test) and the predicted labels (predictions)
print(classification_report(y_test,predictions))
```

OUTPUT:

	precision	recall	f1-score	support
0	0.95	0.85	0.90	66
1	0.91	0.97	0.94	105
accuracy			0.92	171
macro avg	0.93	0.91	0.92	171
weighted avg	0.93	0.92	0.92	171

```
# Define a grid of hyperparameters for the SVM model
param_grid={'C':[0.1,1,10,100,1000],
            'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel' :['rbf']}
```

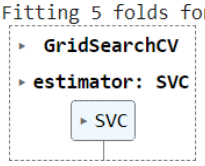
```
from sklearn.model_selection import GridSearchCV
```

```
# Create a GridSearchCV object with the SVM model, parameter grid,
refit=True, and verbose=1
grid=GridSearchCV(SVC(),param_grid,refit=True,verbose=1)
```

```
# Fit the grid search to the training data to find the best
hyperparameters
grid.fit(X_train,y_train)
```

OUTPUT:

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```



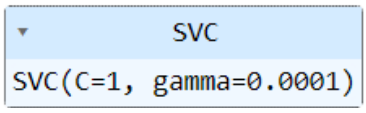

```
# Fit the best parameters on the training data
grid.best_params_
```

OUTPUT:

```
{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
#returns the best estimator
grid.best_estimator_
```

OUTPUT:




```
grid_predictions=grid.predict(X_test)
```

```
#returns a matrix that represents the performance of the classifier.
print(confusion_matrix(y_test,grid_predictions))
```

OUTPUT:

```
[[ 59   7]
 [  4 101]]
```

```
#used to generate a text report that includes various classification
metrics for a classification problem.
print(classification_report(y_test,grid_predictions))
```

OUTPUT:

```
              precision    recall  f1-score   support

           0       0.94      0.89      0.91        66
           1       0.94      0.96      0.95       105

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```

LAB-10

DESCRIPTION:

Non-Parametric Algorithm (KNN) for classification, regression, and analysis of different neighbors.

PROBLEM STATEMENT:

Implement the K-Nearest Neighbors (KNN) algorithm for both classification and regression tasks. Analyze its performance across a range of neighbor values (k) and assess the impact on accuracy and predictive power. Provide insights into the suitability of KNN for various real-world data scenarios.

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, mean_squared_error,
precision_score, recall_score, f1_score
```

```
# Load the Iris dataset for classification and the diabetes dataset for
regression
iris = datasets.load_iris()
diabetes = datasets.load diabetes()
```

```
# Split the data into training and testing sets
X_train_cls, X_test_cls, y_train_cls, y_test_cls =
train_test_split(iris.data, iris.target, test_size=0.3,
random_state=42)
X_train_reg, X_test_reg, y_train_reg, y_test_reg =
train_test_split(diabetes.data, diabetes.target, test_size=0.3,
random_state=42)
```

```
#Define a range of 'k' values to test
k_values = list(range(1, 11))  # From 1 to 10
```

```
# Lists to store results for classification and regression
classification_accuracy = []
classification_precision = []
classification_recall = []
classification_f1_score = []
regression_results = []
```

```
# Perform classification and regression for different 'k' values
for k in k_values:
    # Classification
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train_cls, y_train_cls)
    y_pred_cls = knn_classifier.predict(X_test_cls)

    # Calculate classification metrics
    accuracy = accuracy_score(y_test_cls, y_pred_cls)
    precision = precision_score(y_test_cls, y_pred_cls,
average='weighted')
    recall = recall_score(y_test_cls, y_pred_cls, average='weighted')
    f1 = f1_score(y_test_cls, y_pred_cls, average='weighted')

    classification_accuracy.append(accuracy)
    classification_precision.append(precision)
    classification_recall.append(recall)
    classification_f1_score.append(f1)

    # Regression
    knn_regressor = KNeighborsRegressor(n_neighbors=k)
    knn_regressor.fit(X_train_reg, y_train_reg)
    y_pred_reg = knn_regressor.predict(X_test_reg)
    mse = mean_squared_error(y_test_reg, y_pred_reg)
    regression_results.append(mse)
```

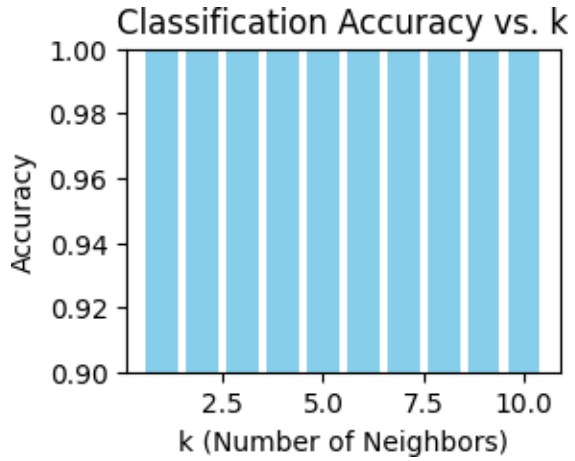
```
#Create a figure with multiple subplots
plt.figure(figsize=(18, 12))
```

OUTPUT:

```
<Figure size 1800x1200 with 0 Axes>
<Figure size 1800x1200 with 0 Axes>
```

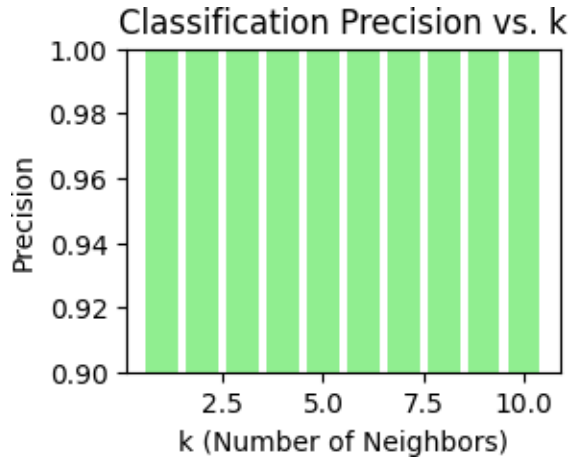
```
# Classification Metrics
plt.subplot(2, 2, 1)
plt.bar(k_values, classification_accuracy, color='skyblue')
plt.title('Classification Accuracy vs. k')
plt.xlabel('k (Number of Neighbors)')
plt.ylabel('Accuracy')
plt.ylim(0.9, 1.0)
```

OUTPUT:



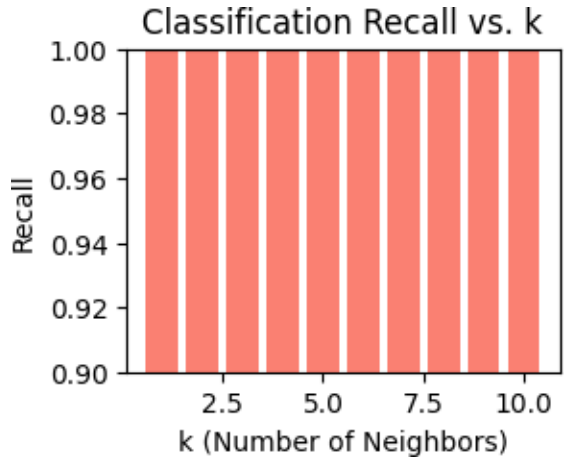

```
plt.subplot(2, 2, 2)
plt.bar(k_values, classification_precision, color='lightgreen')
plt.title('Classification Precision vs. k')
plt.xlabel('k (Number of Neighbors)')
plt.ylabel('Precision')
plt.ylim(0.9, 1.0)
```

OUTPUT:



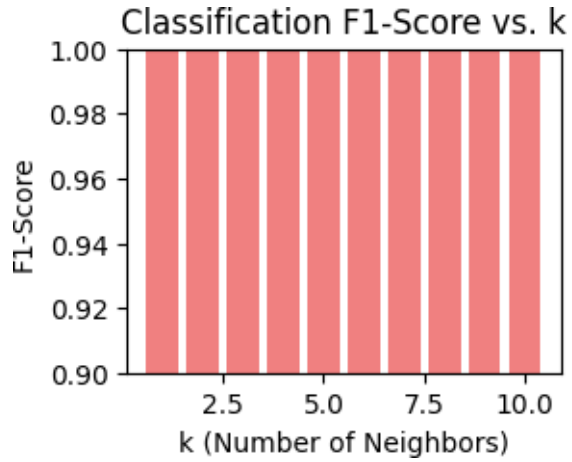
```
plt.subplot(2, 2, 3)
plt.bar(k_values, classification_recall, color='salmon')
plt.title('Classification Recall vs. k')
plt.xlabel('k (Number of Neighbors)')
plt.ylabel('Recall')
plt.ylim(0.9, 1.0)
```

OUTPUT:




```
plt.subplot(2, 2, 4)
plt.bar(k_values, classification_f1_score, color='lightcoral')
plt.title('Classification F1-Score vs. k')
plt.xlabel('k (Number of Neighbors)')
plt.ylabel('F1-Score')
plt.ylim(0.9, 1.0)
```

OUTPUT:



```
plt.tight_layout()
```

OUTPUT:

```
<Figure size 640x480 with 0 Axes>
```

```
# Regression Histogram
plt.figure(figsize=(12, 6))
plt.bar(k_values, regression_results, color='lightcoral')
plt.title('Regression Mean Squared Error vs. k')
plt.xlabel('k (Number of Neighbors)')
plt.ylabel('MSE')
```

OUTPUT: