HEART ATTACK ANALYSIS AND PREDICTION

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

Divya Vishwanath 2203A52130

Under the guidance of

Dr. Kiran Eranki

Assistant Professor, Department of CSE.

Department of Computer Science and Artificial Intelligence

SR UNIVERSITY

Department of Computer Science and Artificial Intelligence

CERTIFICATE

This is to certify that project entitled **"HEART ATTACK ANALYSIS AND PREDICTION"** is the bonafied work carried out by **V. DIVYA** as a Course Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING** during the academic year 2022-2023 under our guidance and Supervision.

Dr. Kiran Eranki
Asst. Professor,
S R University,
Ananthasagar ,Warangal

Dr. M. Sheshikala
Assoc. Prof .& HOD (CSE)
S R University,
Ananthasagar ,Warangal

ACKNOWLEDGEMENT

We express our thanks to Course co-coordinator **Dr. Kiran Eranki Prof.** for guiding us from the beginning through the end of the Course Project. We express our gratitude to Head of the department CS&AI, **Dr. M.Sheshikala, Associate Professor** for encouragement, support and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

ABSTRACT

Heart Attack Prediction using Machine Learning Technique in Big data analytics has started to play an important role in the healthcare practices and research. heart attack prediction will be found primarily on real-time processing, distributed and real-time classification and distribution, storage so; databases can be easily modified by the doctors. If you know all the attributes related to our health we can check easily how much chance to the Heart attack risk, using the system applications. It was recently used to train classification models. After that using extract the features that is condition to be find to be classified by Decision Tree (DT).Compared to existing; algorithms provides better performance. After classification, performance criteria including accuracy, precision, F-measure is to be calculated. If you are concern about the heart attack risks, you might be referred to a heart specialist. Some attributes are Heart Attack risk factors including which is the High blood pressure, high cholesterol and diabetes, increases your risk even more. Hence we are also checking your symptoms of heart attack and take about prevention.

Table of Contents

Chapter No. **Title**

INTRODUCTION

1.1 OVERVIEW

Heart disease is one of the biggest health risks for association today. According to the World Health Organization (WHO), stroke and heart attack are the most common cause of global death (85%). Therefore, the availability of data and data mining techniques, especially machine learning and early detection of Heart Attack, can help patients to anticipate a potential disease response. In the healthcare field, it is becoming more and more common nowadays to source large amounts of data (big data), streaming machines, advanced healthcare services, high throughput instruments, sensor networks, Internet of Things, mobile application applications, data archiving and processing, from many areas. The heart attack prediction is most significant and important duty in medical field which Requires more attention. However there are some techniques for data collection and analysis. Also huge set of medical data is required to correctly predict the heart attack.

1.2. PROBLEM STATEMENT

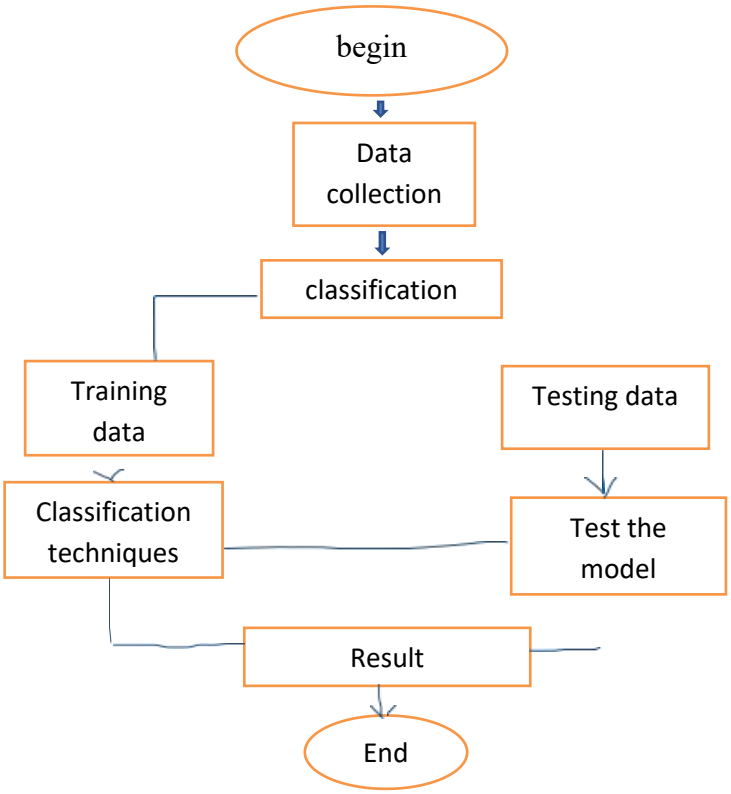
Heart Attack is one of the huge health risks for human's healthy life. big data growth in medical and healthcare association today, early solution and accurate analysis of medical data benefits through patient care and community services. If the quality of medical data some data are incomplete, the accuracy of the analysis decreases. So we need huge medical data to predict the heart attack. The aim of heart attack analysis and prediction is to Design and Implement the Heart Attack analysis and Prediction System using machine learning techniques.

1.3. EXISTING SYSTEM

In this system, the input details are obtained from the patients. Then from the user inputs, and Using machine learning techniques heart attack is analyzed. Now the obtained results are compared with the results of existing models within the same domain and found to be improved. The data of patient is collected from the UCI laboratory and used to discover the patterns with the existed machine learning models. The existing systems of predicting the heart attack are K nearest neighbours, support vector machines (SVM) and naive bayes. The results are compared for performance and accuracy with these machine learning algorithms.

1.4. PROPOSED SYSTEM

The proposed work predicts heart disease by exploring the above mentioned four classification algorithms and does performance analysis. The objective of this study is to effectively predict if the patient suffers from heart disease. The health professional enters the input values from the patient's health report. The data is fed into model which predicts the probability of having heart disease. Figure shows the entire process involved.

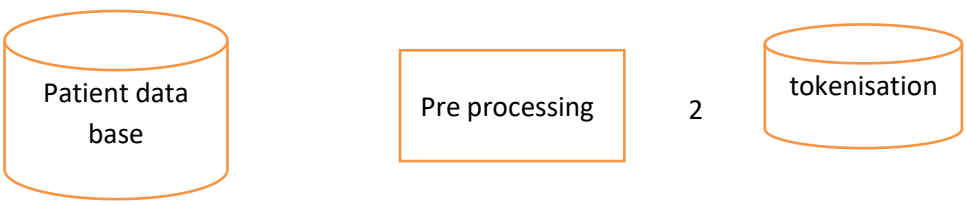


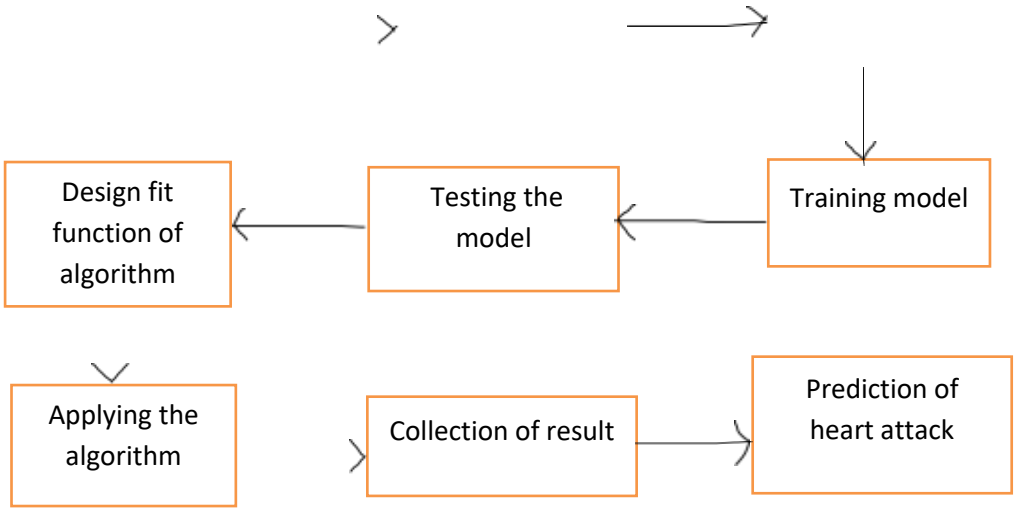
1.5. OBJECTIVES

The main objective of this research is to develop a heart prediction system. The system can discover and extract hidden knowledge associated with diseases from a historical heart data set Heart Disease prediction system aims to exploit data mining techniques on medical data set to assist in the prediction of the heart diseases. The prediction Provides the new approach to concealed patterns in the data. It Helps to avoid human biasness and also Reduce the cost of medical tests and gives accurate result whether the patient gets heart attack or not.

1.7. ARCHITECTURE

The architecture of the proposed system is as displayed in the figure below. The major components of the architecture are as follows: patient database, preprocessing, tokenization, training the model, test the model, design fitness function, application of genetic algorithm, results collection and prediction of heart disease.





2.1.1 LITERATURE SURVEY

There are numerous works has been done related to disease prediction systems using different data mining techniques and machine learning algorithms in medical centres.

K. Polaraju et al, [7] proposed Prediction of Heart Disease using Multiple Regression Model and it proves that Multiple Linear Regression is appropriate for predicting heart disease chance. The work is performed using training data set consists of 3000 instances with 13 different attributes which has mentioned earlier. The data set is divided into two parts that is 70% of the data are used for training and 30% used for testing. Based on the results, it is clear that the classification accuracy of Regression algorithm is better compared to other algorithms.

Marjia et al, [8] developed heart disease prediction using KStar, j48, SMO, and Bayes Net and Multilayer perception using WEKA software. Based on performance from different factor SMO and Bayes Net achieve optimum performance than KStar, Multilayer perception and J48 techniques using k-fold cross validation. The accuracy performances achieved by those algorithms are still not satisfactory.

S. Seema et al,[9] focuses on techniques that can predict chronic disease by mining the data containing in historical health records using Naïve Bayes, Decision tree, Support Vector Machine(SVM) and Artificial Neural Network(ANN). A comparative study is performed on classifiers to measure the better performance on an accurate rate. From this experiment, SVM gives highest accuracy rate, whereas for diabetes Naïve Bayes gives the highest accuracy.

Ashok Kumar Dwivedi et al, [10] recommended different algorithms like Naive Bayes, Classification Tree, KNN, Logistic Regression, SVM and ANN. The Logistic Regression gives better accuracy compared to other algorithms

MeghaShahi et al, [11] suggested Heart Disease Prediction System using Data Mining Techniques. WEKA software used for automatic diagnosis of disease and to give qualities of services in healthcare centres. The paper used various algorithms like SVM, Naïve Bayes, Association rule, KNN, ANN, and Decision Tree. The paper recommended SVM is effective and provides more accuracy as compared with other
data mining algorithms.

Jayami Patel et al, [14] suggested heart disease prediction using data mining and the machine learning algorithm. The goal of this study is to extract hidden patterns by applying data mining techniques. The best algorithm J48 based on UCI data has the highest accuracy rate compared to LMT.

3.DATA PRE-PROCESSING

3.1.1 DATASET DESCRIPTION

Sno	Attributes	Description
1.	Age	Age of the patient
2.	Gender	Gender of the patient (male or female)
3.	cp	Chest pain type
4.	chol	Cholesterol
5.	fbs	Fasting blood sugar
6.	thalach	Maximum heart rate achieved in beats per minute
7.	thal	Displays the thalassemia
8.	ca	Number of major vessels
9.	old peak	ST depression induced by exercise related to rest
10	restecg	Resting electro cardio graphic results
11.	exng	Exercise induced angina
12.	sla	
13.		
14.	Out put	Predicts whether patient gets heart attack or not

3.2 DATA CLEANING

Data quality has become an important issue. This issue becomes more and more important in medicine area, where the need for effective decision making is high. In this context, the need for data cleaning to improve data quality is becoming crucial. Duplicate records elimination is a challenging data cleansing task. In this paper, we present a duplicate records elimination approach to improve the quality of data. We propose a deep learning-based approach for duplicate records detection using a sentence embeddings model. Also, we propose an algorithm for duplicated records correction. Then, we apply the proposed duplicate records elimination approach to analyze the effect of data cleaning on the quality of decisions.

Experiments show that the classification performance improves upon the application of the duplicate records elimination approach on datasets compared to that of datasets with duplicate records.

3.3 DATA AUGUMENTATION

Data augmentation is a set of techniques to artificially increase the amount of data by generating new data p oints from existing data. This includes making small changes to data or using deep learning models to gener ate new data points, Machine learning applications especially in the deep learning domain continue to diversify and increase rapidly. Data-centric approaches to model development such as data augmentation techniques can be a good tool against challenges which the artificial intelligence world faces.

Data augmentation is useful to improve performance and outcomes of machine learning models by forming new and different examples to train datasets. If the dataset in a machine learning model is rich and sufficient, the model performs better and more accurately.

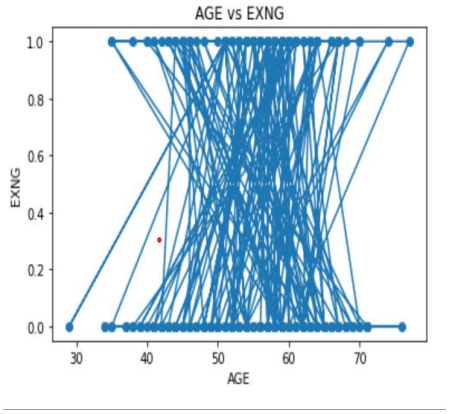
3.4 DATA VISUALISATION

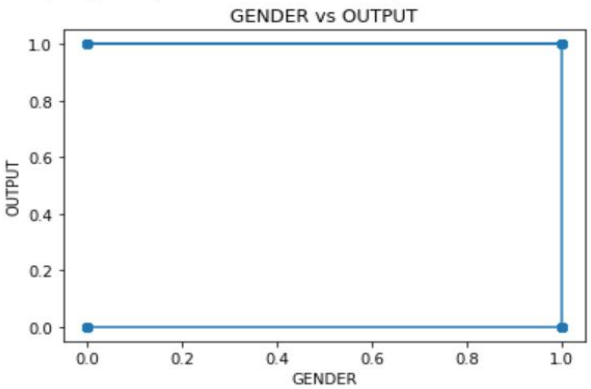
The heart attack analysis and prediction data set contains thirteen feature variables and one target variable output which contains symptoms of patients as features and whether the patient get heart attack or not as target

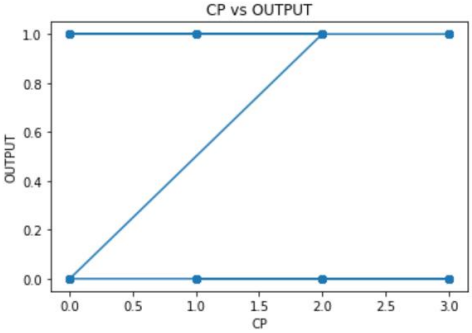
DATASET

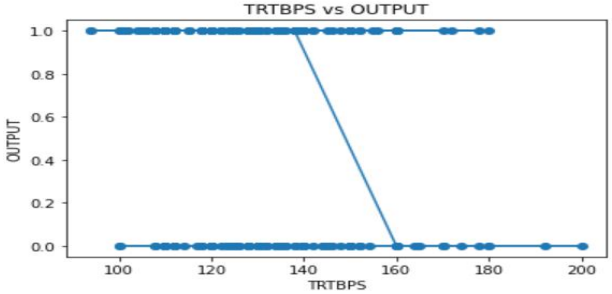
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
2	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
3	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
4	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
5	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
6	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
7	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
8	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
9	44	1	1	120	263	0	1	173	0	0	2	0	3	1
10	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
11	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
12	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
13	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
14	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
15	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
16	58	0	3	150	283	1	0	162	0	1	2	0	2	1
17	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
18	58	0	2	120	340	0	1	172	0	0	2	0	2	1
19	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
20	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
21	69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
22	59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
23	44	1	2	130	233	0	1	179	1	0.4	2	0	2	1

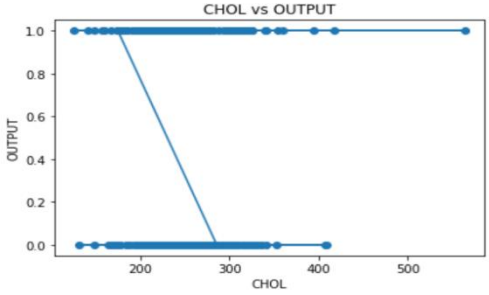
GRAPHS PLOTTED BETWWEN FEATURE AND TARGET VARIABLES:

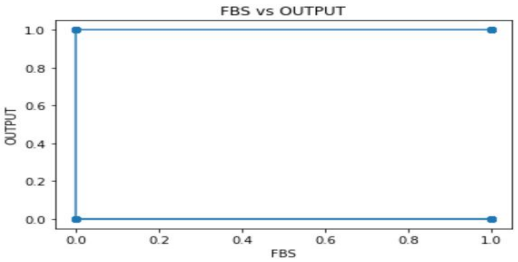

AGE vs EXNG
EXNG
AGE

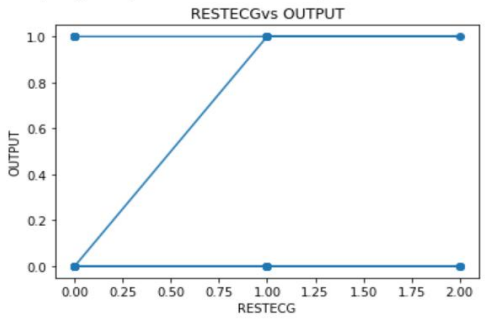

GENDER vs OUTPUT
OUTPUT
GENDER

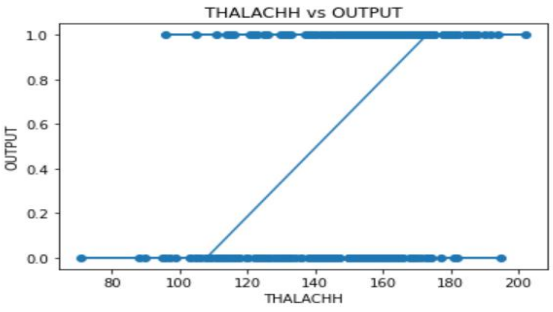

CP vs OUTPUT
OUTPUT
CP

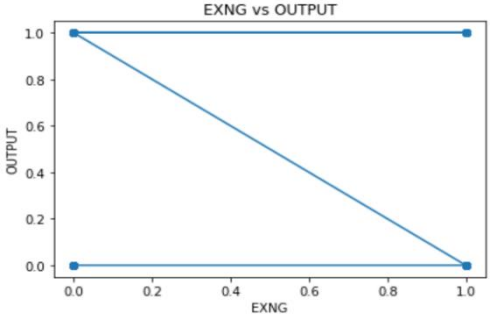

TRTBPS vs OUTPUT
OUTPUT
TRTBPS

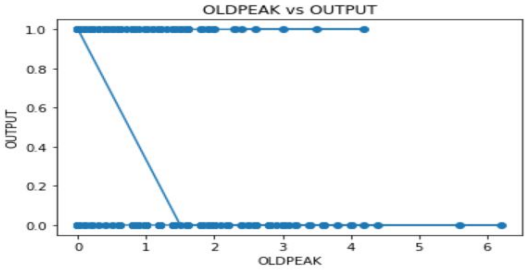

CHOL vs OUTPUT
OUTPUT
CHOL

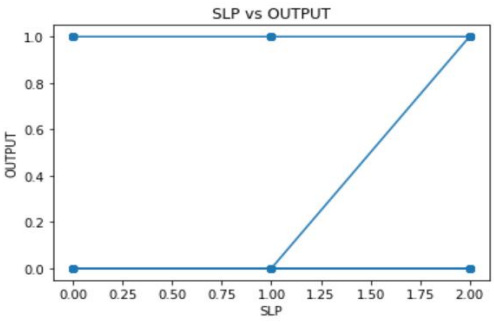

FBS vs OUTPUT
OUTPUT
FBS

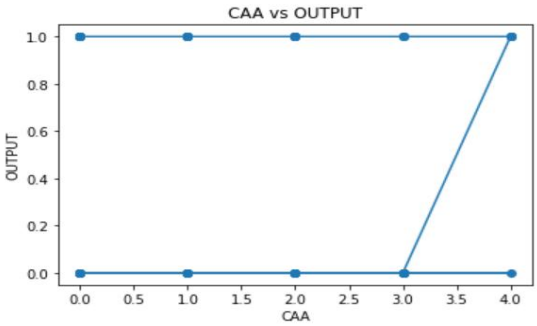

RESTECGvs OUTPUT
OUTPUT
RESTECG

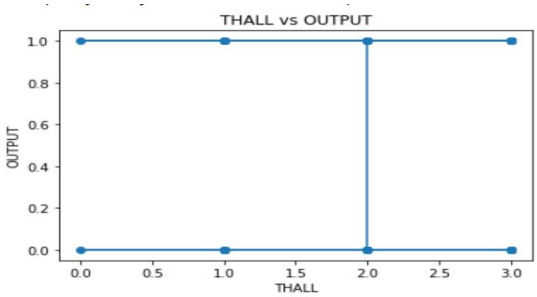

THALACHH vs OUTPUT
OUTPUT
THALACHH


EXNG vs OUTPUT
OUTPUT
EXNG


OLDPEAK vs OUTPUT
OUTPUT
OLDPEAK







4. METHODOLOGY

4.1 PROCEDURE TO SOLVE THE GIVEN PROBLEM

In this project of heart attack analysis and prediction we use two approaches:

- Logistic regression
- KNN (K-Nearest neighbours)

Logistic regression

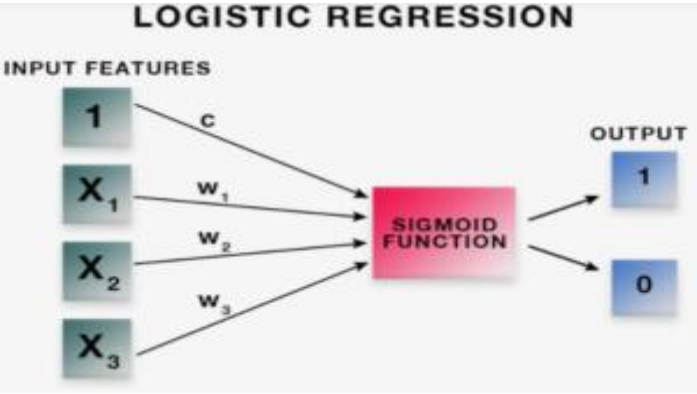
Logistic regression is a supervised machine learning algorithm.

This type of statistical model ,is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure.

Advantages of the Logistic Regression Algorithm:

- Logistic regression performs better when the data is linearly separable
- It does not require too many computational resources as it's highly interpretable
- There is no problem scaling the input features—It does not require tuning
- It is easy to implement and train a model using logistic regression

- It gives a measure of how relevant a predictor (coefficient size) is, and its direction of association (positive or negative)



K-Nearest neighbours

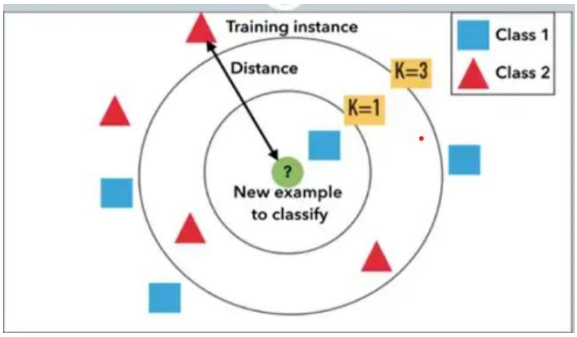
KNN is a non-parametric machine learning algorithm. The KNN algorithm is a supervised learning method. This means that all the data is labelled and the algorithm learns to predict the output from the input data. It performs well even if the training data is large and contains noisy values. The data is divided into training and test sets. The train set is used for model building and training. Ak- value is decided which is often the square root of the number of observations. Now the test data is predicted on the model built. There are different distance measures. For continuous variables, Euclidean distance, Manhattan distance and Minkowski distance measures can be used.

However, the commonly used measure is Euclidean distance. The formula for Euclidean distance is as follows:

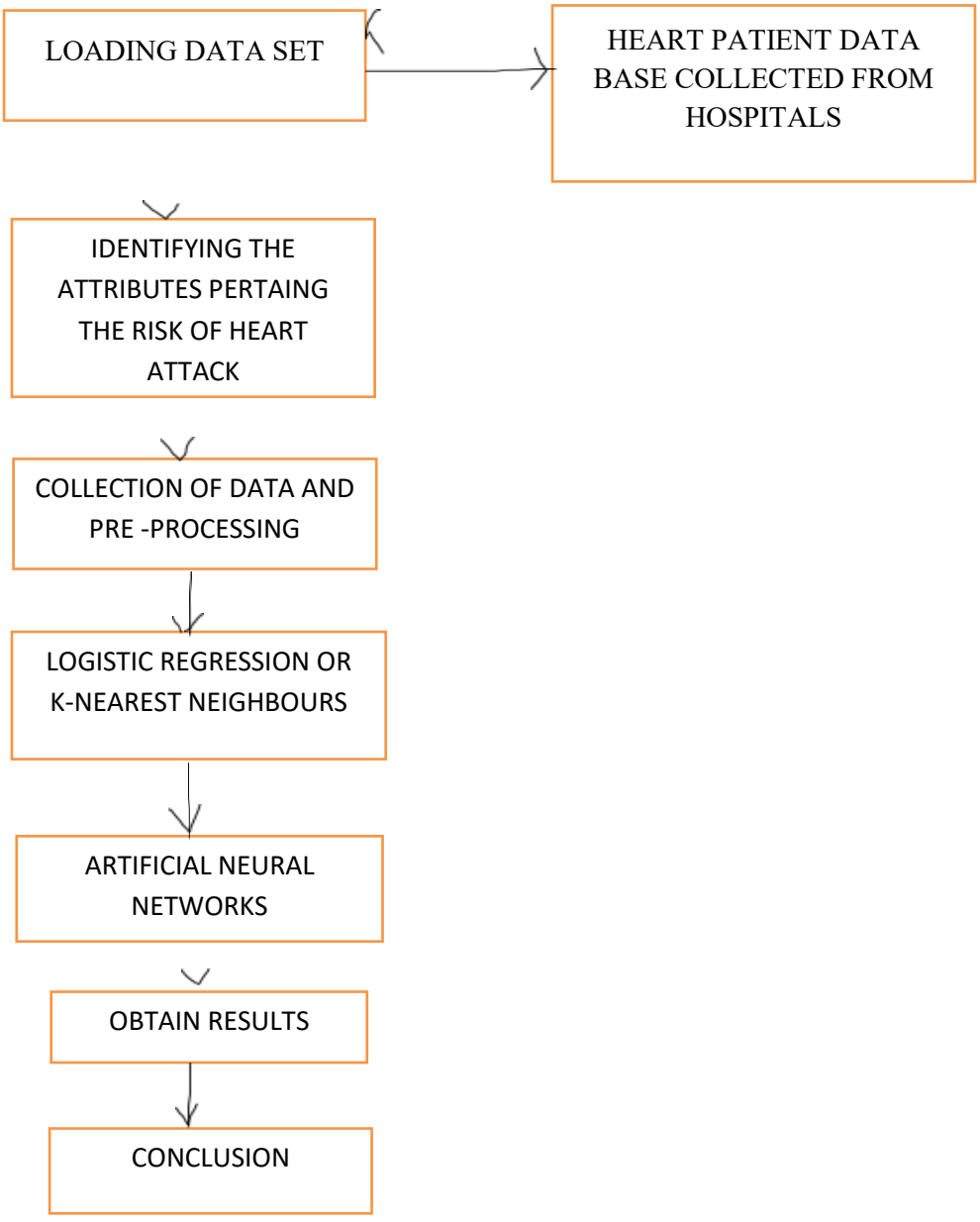
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbours
- **Step-2:** Calculate the Euclidean distance of **K number of neighbours**
- **Step-3:** Take the K nearest neighbours as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbours, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbour is maximum.
- **Step-6:** Our model is ready.



4.2 MODEL ARCHITECTURE



-

4.3 SOFTWARE DESCRIPTION

Software requirements:

Operating system: Windows

Platform: google colab

Programing language: python

5. RESULTS

CODE:

Logistic regression :

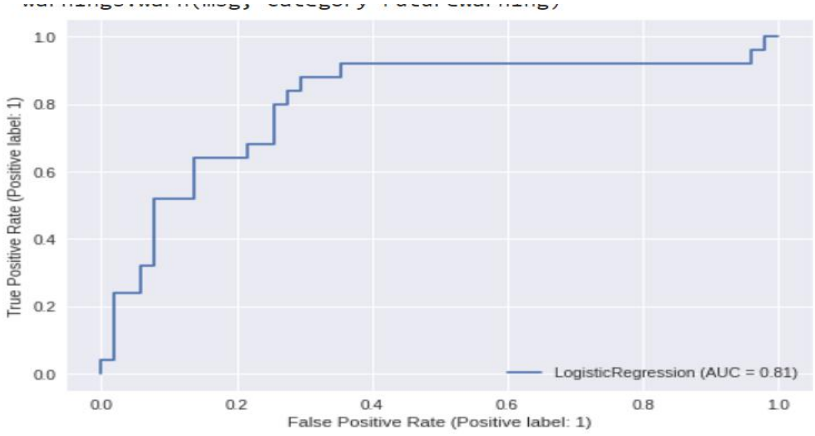
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
data=pd.read_csv('/content/stml data set.csv')
print(data)
x=data.iloc[:,0:8]
y=data.iloc[:,8:9]
from sklearn.preprocessing import StandardScaler
stsc=StandardScaler()
data=stsc.fit(x)
dd=stsc.transform(x)
print(data)
print(dd)
print(x)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random
_state=True)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
lr= LogisticRegression(random_state = 88)
mm=lr.fit(x_train,y_train)
print(mm.score(x_train,y_train))
print(mm.score(x_test,y_test))
yp=mm.predict(x_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(yp,y_test))
from sklearn.metrics import classification_report
print(classification_report(yp,y_test))
from sklearn import metrics
metrics.plot_roc_curve(mm,x_test,y_test)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm=confusion_matrix(yp,y_test)
d=ConfusionMatrixDisplay(cm).plot()
```

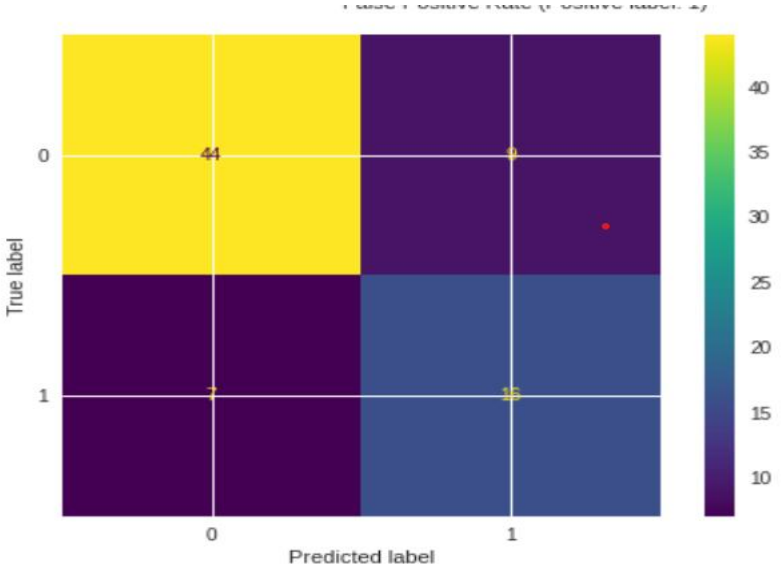
RESULT:

```
     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  \
0     63    1   3     145   233    1        0       150     0      2.3    0
1     37    1   2     130   250    0        1       187     0      3.5    0
2     41    0   1     130   204    0        0       172     0      1.4    2
3     56    1   1     120   236    0        1       178     0      0.8    2
4     57    0   0     120   354    0        1       163     1      0.6    2
..   ...  ...  ..     ...   ...  ...      ...       ...   ...      ...  ...
298   57    0   0     140   241    0        1       123     1      0.2    1
299   45    1   3     110   264    0        1       132     0      1.2    1
300   68    1   0     144   193    1        1       141     0      3.4    1
301   57    1   0     130   131    0        1       115     1      1.2    1
302   57    0   1     130   236    0        0       174     0      0.0    1

     caa  thall  output
0      0      1       1
1      0      2       1
2      0      2       1
3      0      2       1
4      0      2       1
..   ...    ...     ...
298    0      3       0
299    0      3       0
300    2      3       0
301    1      3       0
302    1      2       0

StandardScaler()
[[ 0.9521966   0.68100522  1.97312292 ...  2.394438   -1.00583187
   0.01544279]
 [-1.91531289  0.68100522  1.00257707 ... -0.41763453  0.89896224
   1.63347147]
 [-1.47415758 -1.46841752  0.03203122 ... -0.41763453 -1.00583187
   0.97751389]
 ...
 [ 1.50364073  0.68100522 -0.93851463 ...  2.394438    0.89896224
  -0.37813176]
 [ 0.29046364  0.68100522 -0.93851463 ... -0.41763453  0.89896224
  -1.51512489]
 [ 0.29046364 -1.46841752  0.03203122 ... -0.41763453 -1.00583187
   1.0649749 ]]
     age  sex  cp  trtbps  chol  fbs  restecg  thalachh
0     63    1   3     145   233    1        0       150
1     37    1   2     130   250    0        1       187
2     41    0   1     130   204    0        0       172
3     56    1   1     120   236    0        1       178
4     57    0   0     120   354    0        1       163
..   ...  ...  ..     ...   ...  ...      ...       ...
298   57    0   0     140   241    0        1       123
```

299	45	1	3	110	264	0	1	132
300	68	1	0	144	193	1	1	141
301	57	1	0	130	131	0	1	115
302	57	0	1	130	236	0	0	174

[303 rows x 8 columns]
(227, 8)
(227, 1)
(76, 8)
(76, 1)
0.801762114537445
0.7894736842105263
0.7894736842105263





KNN:

```
import pandas as pd

import matplotlib.pyplot as plt
```

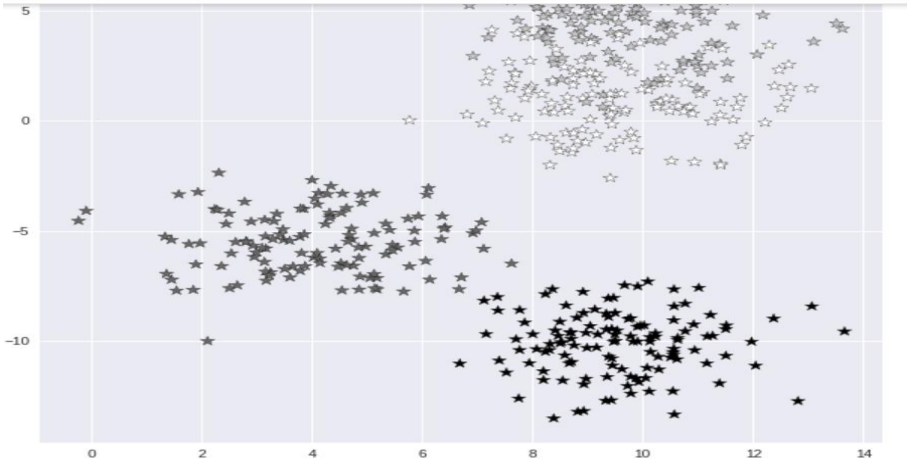
```
from sklearn import numpy as np
.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
data=pd.read_csv('/content/stml data set.csv')
print(data)
X, y = make_blobs(n_samples = 500, n_features = 2, centers = 4,cluster_s
td = 1.5, random_state = 4)
plt.style.use('seaborn')
plt.figure(figsize = (10,10))
plt.scatter(X[:,0], X[:,1], c=y, marker= '*',s=100,edgecolors='black')
plt.show()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
 0)
knn1 = KNeighborsClassifier(n_neighbors = 1)
knn3 = KNeighborsClassifier(n_neighbors=3)
knn1.fit(X_train, y_train)
knn3.fit(X_train, y_train)

y_pred_1 = knn1.predict(X_test)
y_pred_3 = knn3.predict(X_test)
from sklearn.metrics import accuracy_score
print("Accuracy with k=1", accuracy_score(y_test, y_pred_1)*100)
print("Accuracy with k=3", accuracy_score(y_test, y_pred_3)*100)
plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_5, marker= '*', s=100,edg
ecolors='black')
plt.title("Predicted values with k=1", fontsize=20)

plt.subplot(1,2,2)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_1, marker= '*', s=100,edg
ecolors='black')
plt.title("Predicted values with k=3", fontsize=20)
plt.show()
```

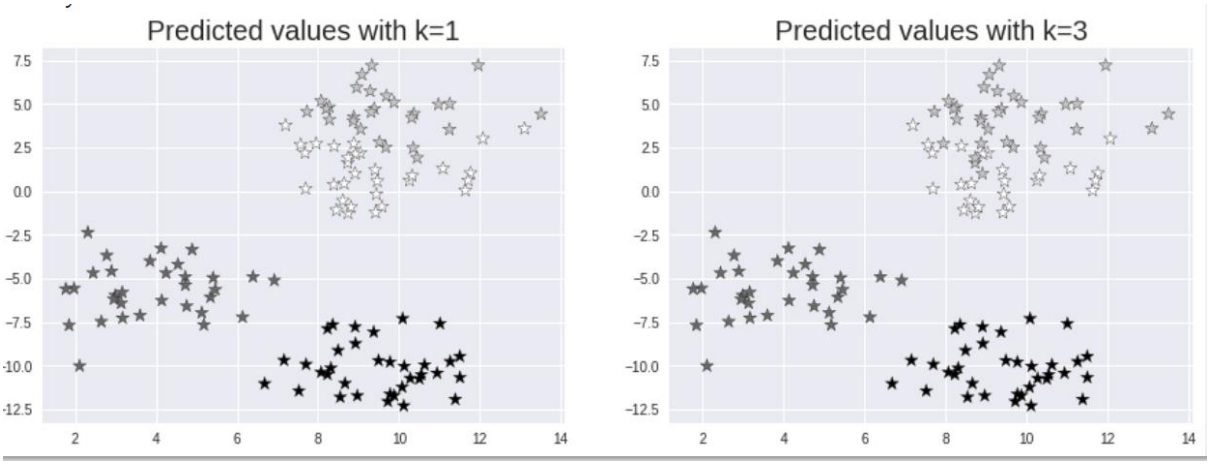
RESULT:

```
    age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak
slp  \
0    63    1   3     145   233    1        0       150     0      2.3
0
1    37    1   2     130   250    0        1       187     0      3.5
0
2    41    0   1     130   204    0        0       172     0      1.4
2
3    56    1   1     120   236    0        1       178     0      0.8
2
4    57    0   0     120   354    0        1       163     1      0.6
2
```

```
..   ...  ...  ..     ...   ...  ...     ...       ...   ...      ...  .
..
298   57    0   0     140   241    0       1       123     1      0.2
1
299   45    1   3     110   264    0       1       132     0      1.2
1
300   68    1   0     144   193    1       1       141     0      3.4
1
301   57    1   0     130   131    0       1       115     1      1.2
1
302   57    0   1     130   236    0       0       174     0      0.0
1
```

```
     caa  thall  output
0      0      1       1
1      0      2       1
2      0      2       1
3      0      2       1
4      0      2       1
..   ...    ...     ...
298    0      3       0
299    0      3       0
300    2      3       0
301    1      3       0
302    1      2       0

[303 rows x 14 columns]
```






```
Accuracy with k=1 90.4
Accuracy with k=3 92.80000000000001
```

6. CONCLUSION AND FUTURE SCOPE

This paper discusses the various machine learning such as logistic regression and k- nearest neighbour which were applied to the data set. It utilizes the data such as blood pressure, cholesterol, diabetes and then tries to predict the patient gets heart attack or not. Family history of heart disease can also be a reason for developing a heart disease as mentioned earlier. So, this data of the patient can also be included for further increasing the accuracy of the model. This work will be useful in identifying the possible patients who may suffer from heart disease in the next 10 years. This may help in taking preventive measures and hence try to avoid the possibility of heart disease for the patient. So when a patient is predicted as positive for heart disease, then the medical data for the patient can be closely analysed by the doctors. An example would be - suppose the patient has diabetes which may be the cause for heart disease in future and then the patient can be given treatment to have diabetes in control which in turn may prevent the heart disease.

7.REFERENCES

[1] Monika Gandhi, Shailendra Narayanan Singh Predictions in heart disease using techniques of data mining (2015)

[2] J Thomas, R Theresa Princy Human heart disease prediction system using data mining techniques (2016)

[3] Sana Bharti, Shailendra Narayan Singh, Amity university, Noida, India Analytical study of heart disease prediction comparing with different algorithms (May 2015)

[4] Purushottam, Kanak Saxena, Richa Sharma Efficient heart disease prediction system using Decision tree (2015)

[5] Sellappan Palaniyappan, Rafiah Awang Intelligent heart disease prediction using data mining techniques (August 2008)

[6] Himanshu Sharma,M A Rizvi Prediction of Heart Disease using Machine Learning Algorithms: A Survey (August 2017)

[7] Animesh Hazra, Subrata Kumar Mandal, Amit Gupta, Arkomita Mukherjee and Asmita Mukherjee Heart Disease Diagnosis and Prediction Using Machine Learning and Data Mining Techniques: A Review (2017)

[8] V.Krishnaiah, G.Narsimha, N.Subhash Chandra Heart Disease Prediction System using Data Mining Techniques and Intelligent Fuzzy Approach: A Review (February 2016)

[9] Ramandeep Kaur, 2Er. Prabhsharn Kaur A Review - Heart Disease Forecasting Pattern using Various Data Mining Techniques (June 2016)

[10] J.Vijayashree and N.Ch. SrimanNarayanaIyengar Heart Disease Prediction System Using Data Mining and Hybrid Intelligent Techniques: A Review (2016)

[11] Benjamin EJ et.al Heart Disease and Stroke Statistics 2018 At-a-Glance (2018)

[12] Abhay Kishore, Ajay Kumar, Karan Singh, Maninder Punia, Yogita Hambir Department of Computer Engineering, Army Institute of Technology, Pune, Maharashtra Professor, Department of Computer Engineering, Army Institute of Technology, Pune, Maharashtra Heart Attack Prediction Using Deep Learning (2018)

[13] M.Nikhil Kumar, K.V.S Koushik, K.Deepak Department of CSE, VR Siddhartha Engineering College, Vijayawada, Andhra Pradesh, India Prediction Heart Diseases using Data mining and machine learning algorithms and tools.(2018)

[14] Amandeep Kaur, Jyoti Arora Dept of CSC Desh Bhagat University, Punjab, India heart disease prediction using data mining techniques: a survey (2018)

[15] Stephen F Weng, Jenna Reps, Joe Kai, Jonathan M Garibaldi and Nadeem Qureshi Can machine learning improve cardio vascular risk prediction using routine clinical data?(2017)

[16] A.Sahaya Arthy, G. Murugeshwari A survey on heart disease prediction using data mining techniques (April 2018)

[17]https://www.kaggle.com/amanajmera1/framing ham-heart-study-dataset

[18] A.Sudha, P.Gayathri, N.Jaisankar Effective analysis and prediction model for stroke disease using classification methods (April 2012)

INDEX

LAB 01

Exposing to various frameworks of StatML - Numpy, Pandas, Matplotlib, Seaborn, Tensorflow, Keras.

```
# Importing numpy and pandas functions and classes in your code
import numpy as np
import pandas as pd
```

```
# Create a list of numbers
lst1=[1,2,3]
# Convert the list into a numpy array
array1=np.array(lst1)
# Print the resulting numpy array
array1
```

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

```
# Create a series of 7 zeroes
print("A series of zeroes:",np.zeros(7))
# Create a series of 17 ones
print("A series of ones:",np.ones(17))
# Create a series of numbers from -1 to 21 (excluding 22)
print("A series of numbers:",np.arange(5,20))
# Create numbers starting from 0 up to 40 (excluding 40), spaced apart by 4
print("Numbers spaced apart by 2:",np.arange(0,40,4))
# Create numbers starting from 0 up to 11 (excluding 11), spaced apart by 2.5
print("Numbers spaced apart by float:",np.arange(0,11,2.5))
# Create every 5th number from 30 down to 0 (inclusive), in reverse order
print("Every 5th number from 30 in reverse order: ",np.arange(30,-1,-5))
```

```
# Create 11 linearly spaced numbers between 9 and 17

print("11 linearly spaced numbers between 9 and 17:",np.linspace(9,17,11))
```

```
A series of zeroes: [0. 0. 0. 0. 0. 0. 0.]
A series of ones: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
A series of numbers: [ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
Numbers spaced apart by 2: [ 0  4  8 12 16 20 24 28 32 36]
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 9 and 17:  [ 9.   9.8 10.6 11.4 12.2 13.  13.8 14.6 15.4 16.2 17. ]
```

```
# Import the pandas library and alias it as pd

import pandas as pd

# Use the `read_csv` function from pandas to read a CSV file.

# The file path is "/content/wine.csv"

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

# Use the `head()` method to display the first few rows (by default, 5 rows) of the DataFrame.

# This provides a quick look at the structure and contents of the data.

df.head()
```

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Colo
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	

```
# Use the `read_csv` function from pandas to read a CSV file.

# The file path is "/content/data (1).csv"

students=pd.read_csv("/content/data (1).csv")

# Print the summary statistics

students
```

	Name	Marks
0	Gregor	93
1	Antony	85
2	Mark	90
3	Andrew	89
4	Botius	94
5	Swper	77
6	Andres	70
7	Wrighter	95

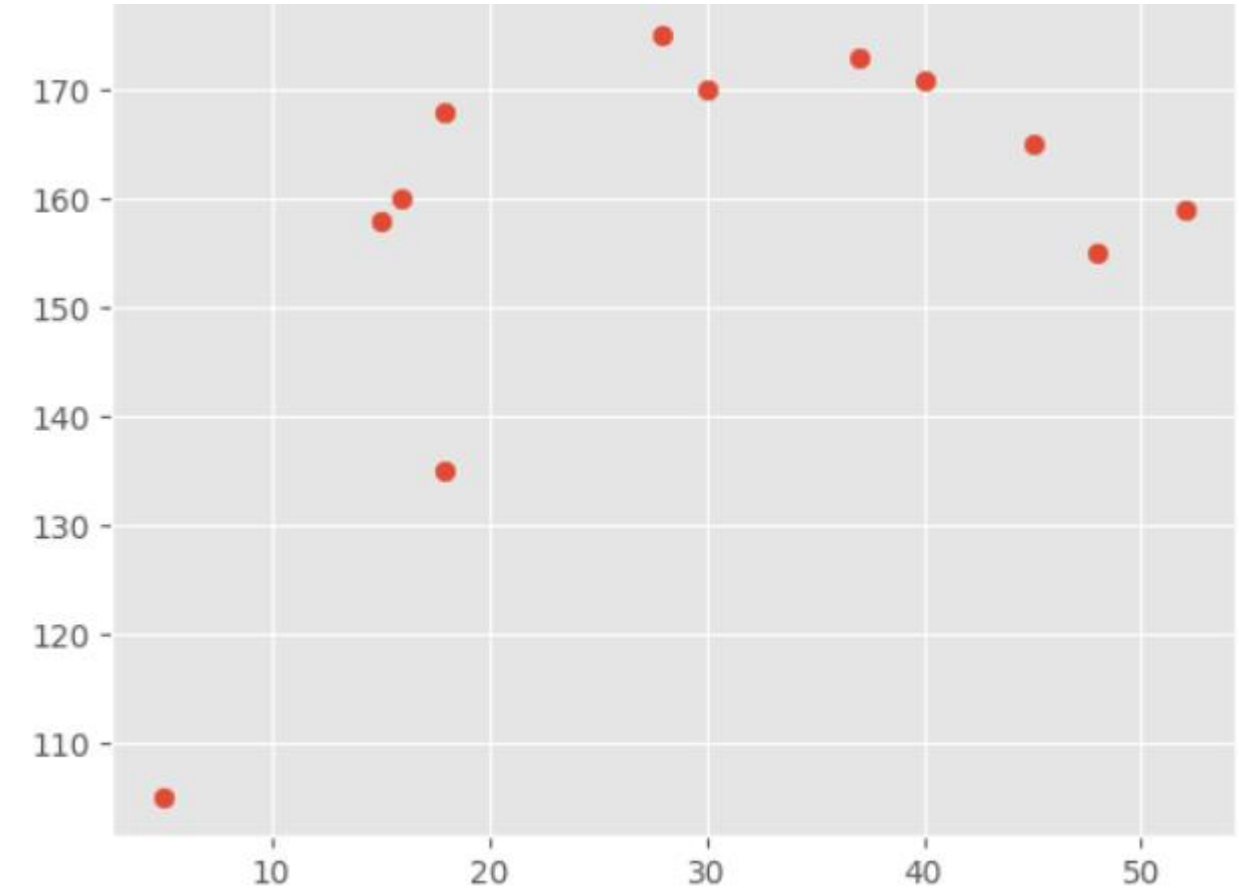
```
#add the fiel path '/content/name_height_weight.xlsx'

datatxt=pd.read_excel("/content/name_height_weight.xlsx")

datatxt
```

	Name	Height	Weight
0	Hoover	169	68
1	McMillan	172	79
2	Journi Lam	158	66
3	Miniox	180	78
4	Conrad	167	82
5	Xavier	178	65
6	Levi	184	70
7	Booker	167	76

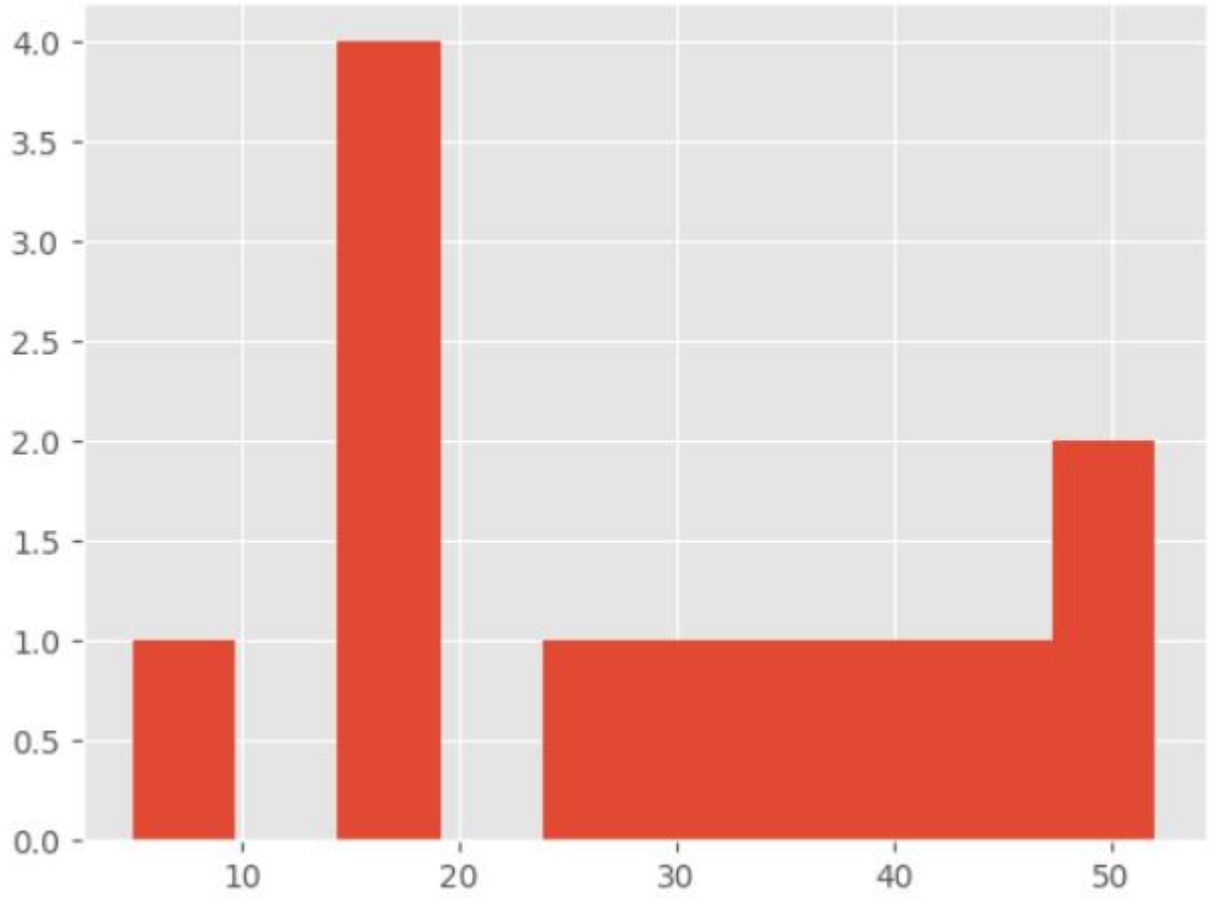
```
list_of_df =
pd.read_html("https://en.wikipedia.org/wiki/2016_Summer_Olympics_medal_table",h
eader=0)

medals=list_of_df[0]

medals.head()
```

	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

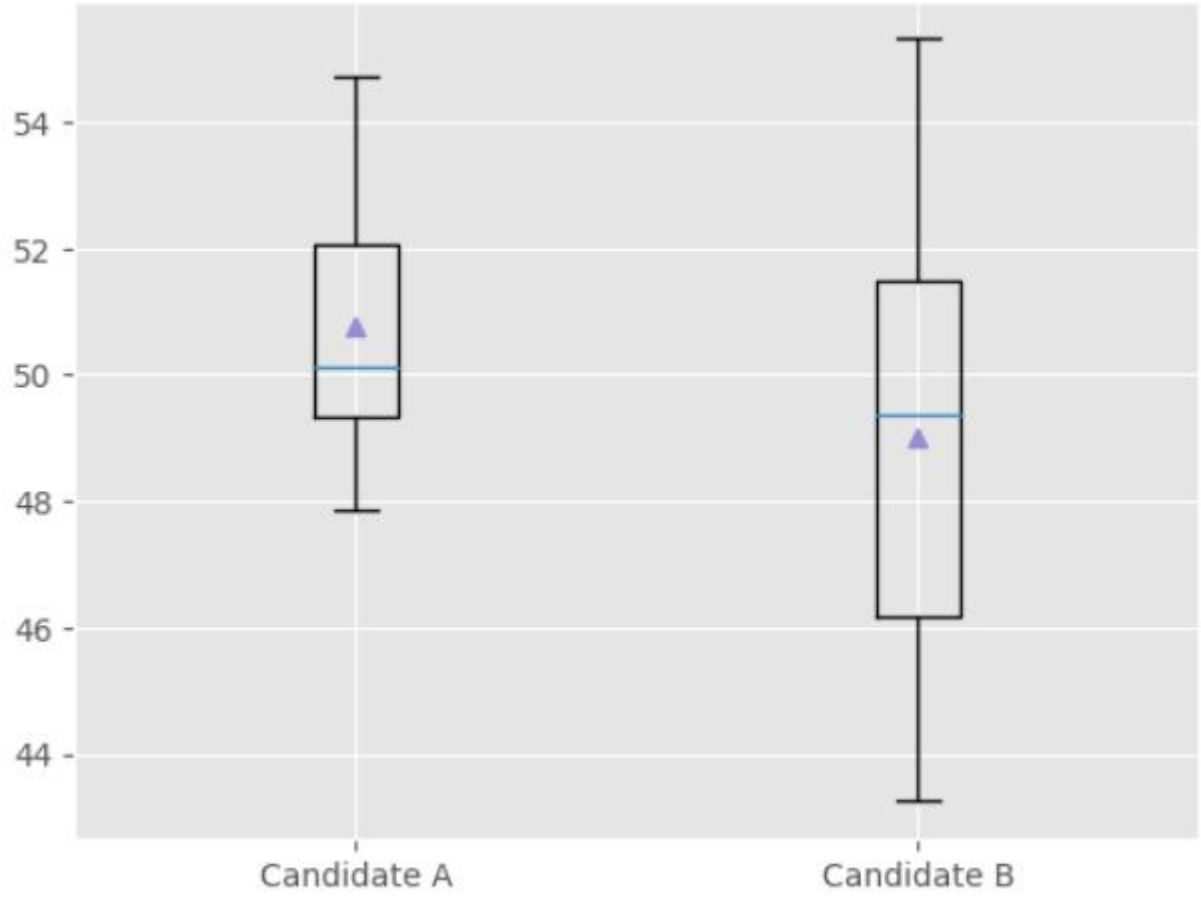
```
import matplotlib.pyplot as plt


people = ['Divya','Brann','Charle','David','paras','Imly'
          'Jagan','Himanshu','Imran','Julio','Katherine','Lily']
age = [16,18,30,45,37,18,28,52,5,40,48,15]
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]


plt.scatter(age,height)
plt.show()
```




```
#plot the age in graph
plt.hist(age)
plt.show()
```




```
days = np.arange(1,31)

candidate_A = 50+days*0.07+2*np.random.randn(30)

candidate_B = 50-days*0.1+3*np.random.randn(30)


plt.style.use('ggplot')

# Note how to convert default numerical x-axis ticks to the list of string by
passing two lists

plt.boxplot(x=[candidate_A,candidate_B],showmeans=True)

plt.grid(True)

plt.xticks([1,2],['Candidate A','Candidate B'])

#plt.yticks(fontsize=15)

plt.show()
```


54
52
50
48
46
44
Candidate A
Candidate B

LAB 02

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

```
# Import the NumPy library and alias it as 'np' for convenience.
import numpy as np
# Import the Matplotlib library and alias it as 'plt' for convenience.
import matplotlib.pyplot as plt
# Import the Axes3D module from the mpl_toolkits.mplot3d library.
from mpl_toolkits.mplot3d import Axes3D
# Import the multivariate_normal function from the scipy.stats library.
from scipy.stats import multivariate_normal
# Enable inline plotting in Jupyter Notebook or IPython with '%matplotlib
inline'.
%matplotlib inline
# Define the covariance matrix. This matrix represents the variances and
covariances
# between variables in a multivariate distribution.
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. The precision matrix is the inverse of the
covariance
# matrix and represents the inverse variances and covariances.
precision = np.linalg.inv(covariance)
# Print the precision matrix, which shows the inverse relationships between
variables.
print(precision)
```

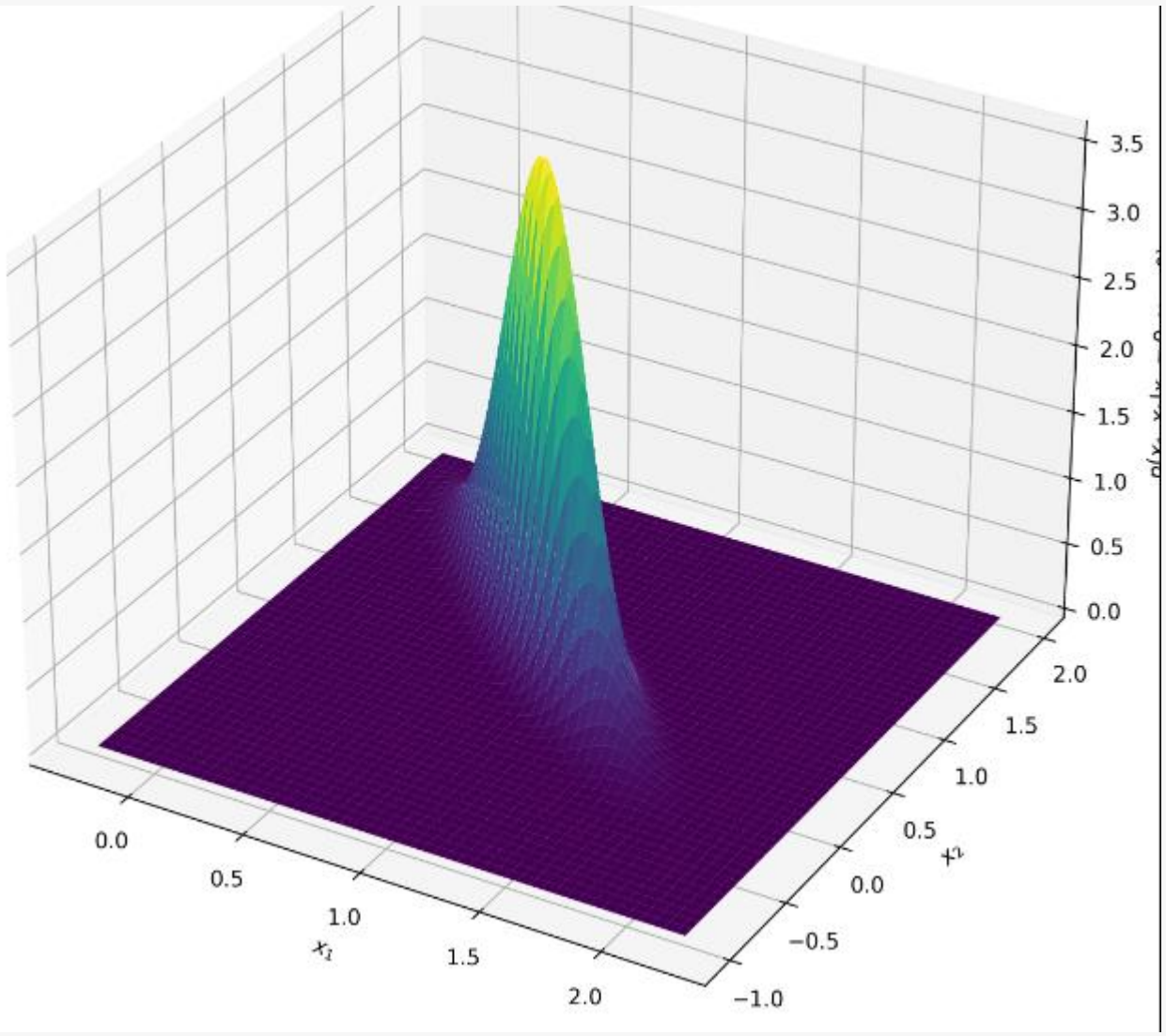
```
[[ 60.   50.  -48.   38. ]
 [ 50.   50.  -50.   40. ]
 [-48.  -50.   52.4 -41.4]
 [ 38.   40.  -41.4  33.4]]
```

```
# Define a function to generate a pair of random values from a multivariate
normal distribution.

def generate_pair():

    return np.random.multivariate_normal([0.8, 0.8], [[0.1, -0.1],[-0.1, 0.12]])

# Call the function 'generate_pair' to generate a pair of random values.

mu_t = generate_pair()

# Print the generated pair of values, which represents a sample from a
multivariate normal distribution.

print(mu_t)
```

```
[0.52117176 1.12871373]
```

```
# Define the grid for x and y values.

x, y = np.mgrid[-0.25:2.25:.01, -1:2:.01]


# Create an array to represent the positions (2D coordinates) on the grid.

pos = np.empty(x.shape + (2,))

pos[:, :, 0] = x

pos[:, :, 1] = y


# Define the mean vector 'mu_p' for the multivariate normal distribution.

mu_p = [0.8, 0.8]


# Define the covariance matrix 'cov_p' for the multivariate normal distribution.

cov_p = [[0.1, -0.1], [-0.1, 0.12]]


# Calculate the probability density function values at each position (x, y).

z = multivariate_normal(mu_p, cov_p).pdf(pos)
```

```
# Create a 3D plot of the probability density.
fig = plt.figure(figsize=(10, 10), dpi=300)
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x, y, z, cmap=plt.cm.viridis)

# Label the axes.
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
ax.set_zlabel('$p(x_1, x_2 | x_3=0, x_4=0)$')
plt.savefig('cond_mvg.png', bbox_inches='tight', dpi=300)
# Display the plot.
plt.show()
```




```
# Define the number of data points 'N'.
```

```
N=1000


import numpy as np

# Generate random data points from a multivariate normal distribution with the specified mean and covariance.

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.

np.savetxt('data.txt',data)
```

```
import pandas as pd

# Read data from a text file ('data.txt') into a pandas DataFrame.

data = pd.read_table('data.txt')

# Alternatively, load data from a text file into a NumPy array.

data = np.loadtxt('data.txt')

# Print or return the loaded data, which could be in the form of a pandas DataFrame or a NumPy array.

data
```

```
array([[-0.65565727,  3.24638345],
       [-1.12242448, -2.5033388 ],
       [-2.37811233, -0.61922491],
       ...,
       [-0.57430947,  2.65492267],
       [-0.76015001, -0.0744587 ],
       [ 2.57530503,  3.26026696]])
```

```
# Calculate the sample mean vector 'mu_ml' by taking the mean along each column of the data.

mu_ml = data.mean(axis=0)

# Center the data by subtracting the sample mean 'mu_ml' from each data point.

x = data - mu_ml

# Calculate the sample covariance matrix 'cov_ml' by taking the dot product of the centered data.

# This is a biased estimate since it divides by 'N'.
```

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

# Calculate the unbiased sample covariance matrix 'cov_ml_unbiased' by taking the dot product of the centered data.

# This estimate divides by 'N - 1' to provide an unbiased estimate.

cov_ml_unbiased = np.dot(x.T, x) / (N - 1)



# Print the sample mean vector, the biased sample covariance matrix, and the unbiased sample covariance matrix.

print(mu_ml)

print(cov_ml)

print(cov_ml_unbiased)
```

```
[0.23934789 1.11985062]
[[1.9897703  0.62799851]
 [0.62799851 4.24193358]]
[[1.99176206 0.62862714]
 [0.62862714 4.24617976]]
```

```
# Define a function 'seq_ml' that calculates the sequence of sample mean vectors.

def seq_ml(data):

    # Initialize a list to store the sample mean vectors, starting with the zero vector.

    mus = [np.array([[0], [0]])]



    # Loop through the data points to calculate the sample mean vectors iteratively.

    for i in range(len(data)):

        # Reshape the current data point to a 2x1 column vector.

        x_n = data[i].reshape(2, 1)

        # Calculate the sample mean 'mu_n' using the sequential mean update formula.

        mu_n = mus[-1] + (x_n - mus[-1]) / (i + 1)

        # Append the calculated 'mu_n' to the list of sample mean vectors.

        mus.append(mu_n)
```

```
    # Return the list of sample mean vectors.
    return mus
```

```
[[0.23934789]
 [1.11985062]]
```

```
# Calculate a sequence of sample mean vectors using the 'seq_ml' function.
mu_p = np.array([[0.28], [1.18]])
# Define the true covariance matrix 'cov_p'. Update the values as needed.
cov_p = np.array([[0.1, -0.1], [-0.1, 0.12]])
# Define the observed covariance matrix 'cov_t' for a dataset. Update the
values as needed.
cov_t = np.array([[2.0, 0.8], [0.8, 4.0]])
print(mu_p,cov_p,cov_t)
```

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

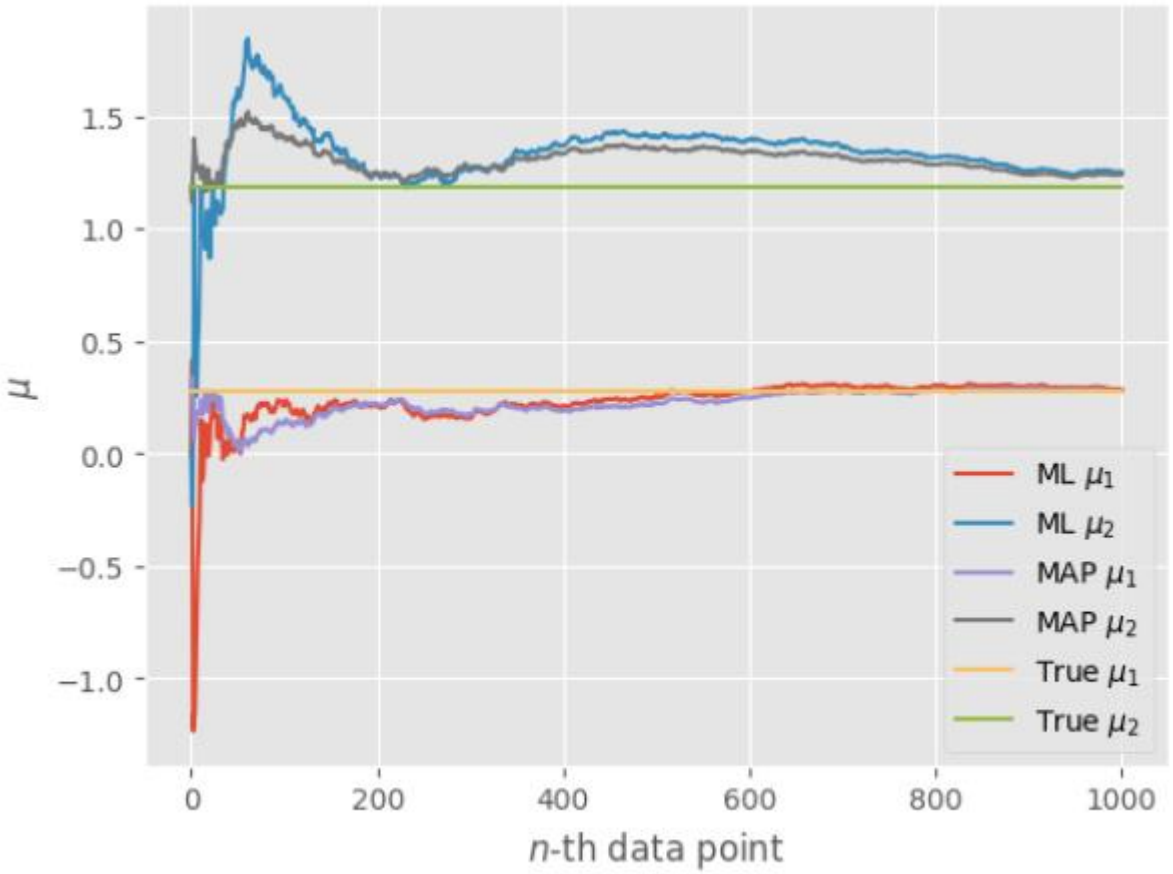
```
# Define a function 'seq_map' that calculates a sequence of posterior mean
vectors and covariance matrices.
```

```
def seq_map(data, mu_p, cov_p, cov_t):
    mus, covs = [mu_p], [cov_p]
    for x in data:
        x_n = x.reshape(2, 1)
        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]))
        mus.append(mu_n)
        covs.append(cov_n)

   return mus, covs
```

```
# Calculate a sequence of posterior mean vectors and covariance matrices using the 'seq_map' function

mus_map, covs_map = seq_map(data, mu_p, cov_p, cov_t)

print(mus_map[-1])
```

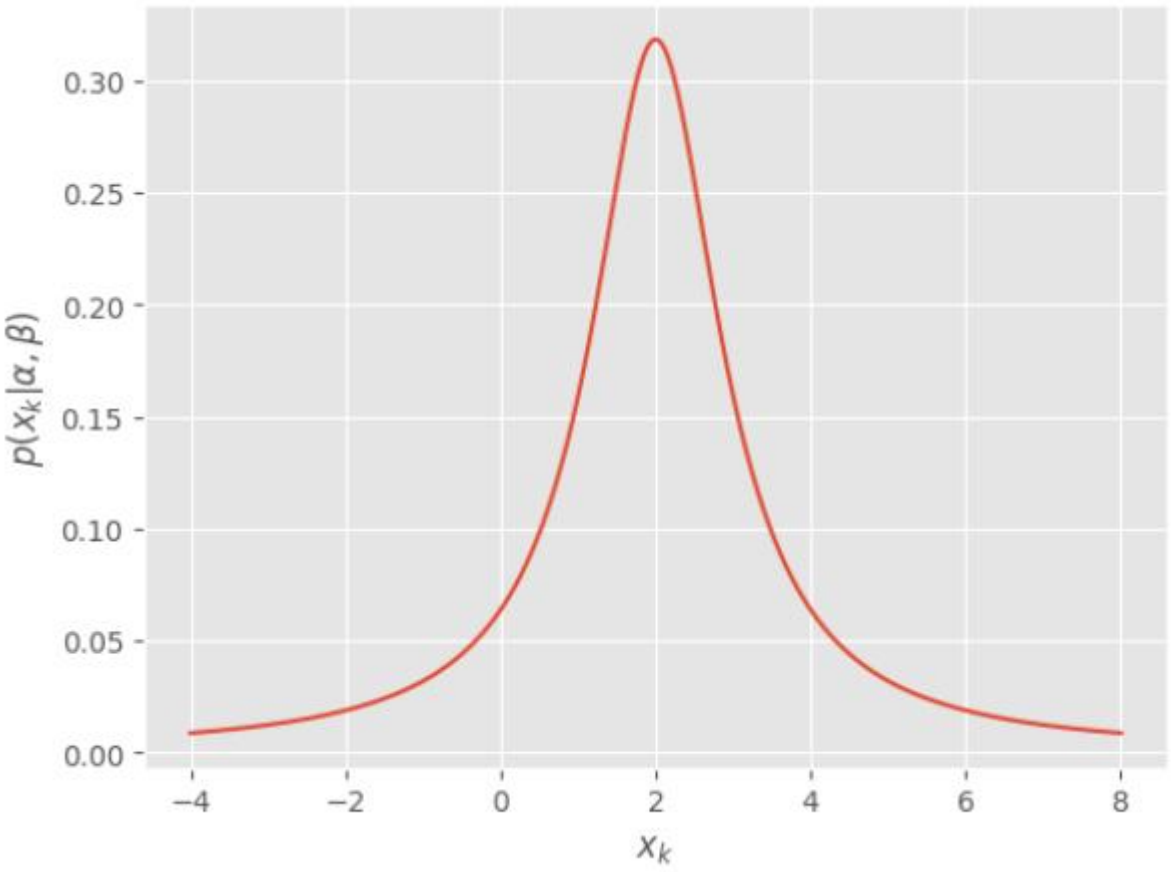
```
[[0.27537996]
 [1.2381292 ]]
```

```
X = np.arange(N+1)

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]

mus1_t = [0.28] * (N+1)

mus2_t = [1.18] * (N+1)

plt.style.use('ggplot')

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 $\mu_2$')

plt.xlabel('$n$-th data point')

plt.ylabel('$\mu$')

plt.legend(loc=4)

plt.savefig('seq_learning.png', bbox_inches='tight', dpi=300)

plt.show()
```



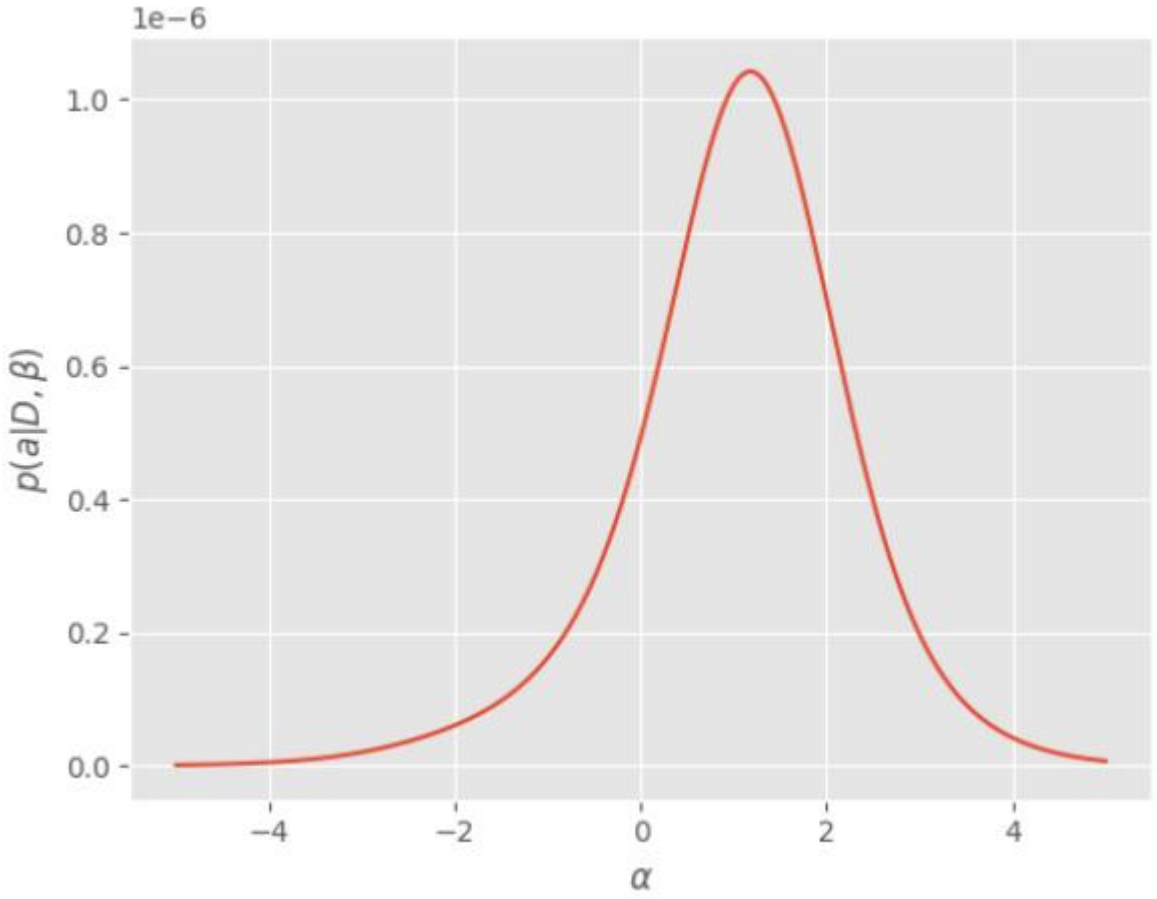

```
# Define a function 'p_xk' that calculates the probability density function
(PDF) of a Cauchy distribution.

def p_xk(x, alpha, beta):

    # Calculate the PDF using the formula for the Cauchy distribution.

   return beta / (np.pi * (beta**2 + (x-alpha)**2))


# Create an array 'x' representing a range of values for which to calculate the
PDF.

x = np.linspace(-4, 8, num=1000)


# Calculate the probability density function (PDF) using the 'p_xk' function
with parameters alpha=2 and beta=1

probs = p_xk(x, 2, 1)


# Create a line plot of the calculated 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)


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




```
# Define a function 'p_a' that calculates the joint probability density
function (PDF) of multiple data points.
def p_a(x, alpha, beta):
    return np.product(beta / (np.pi * beta**2 + (x-alpha)**2))


# Create an array 'D' containing observed data points.
D = np.array([4.8, -2.7, 2.2, 1.1, 0.8, -7.3])
```

```
# Create an array 'alphas' representing a range of values for the location parameter.

alphas = np.linspace(-5, 5, num=1000)

# Set the scale parameter 'beta' for the Cauchy distribution.

beta = 1

# Calculate the likelihood values for different 'alpha' values using the 'p_a' function.

likelihoods = [p_a(D, alpha, beta) for alpha in alphas]

# Create a line plot of the calculated likelihood values.

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)


# Display the plot.

plt.show()
```




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

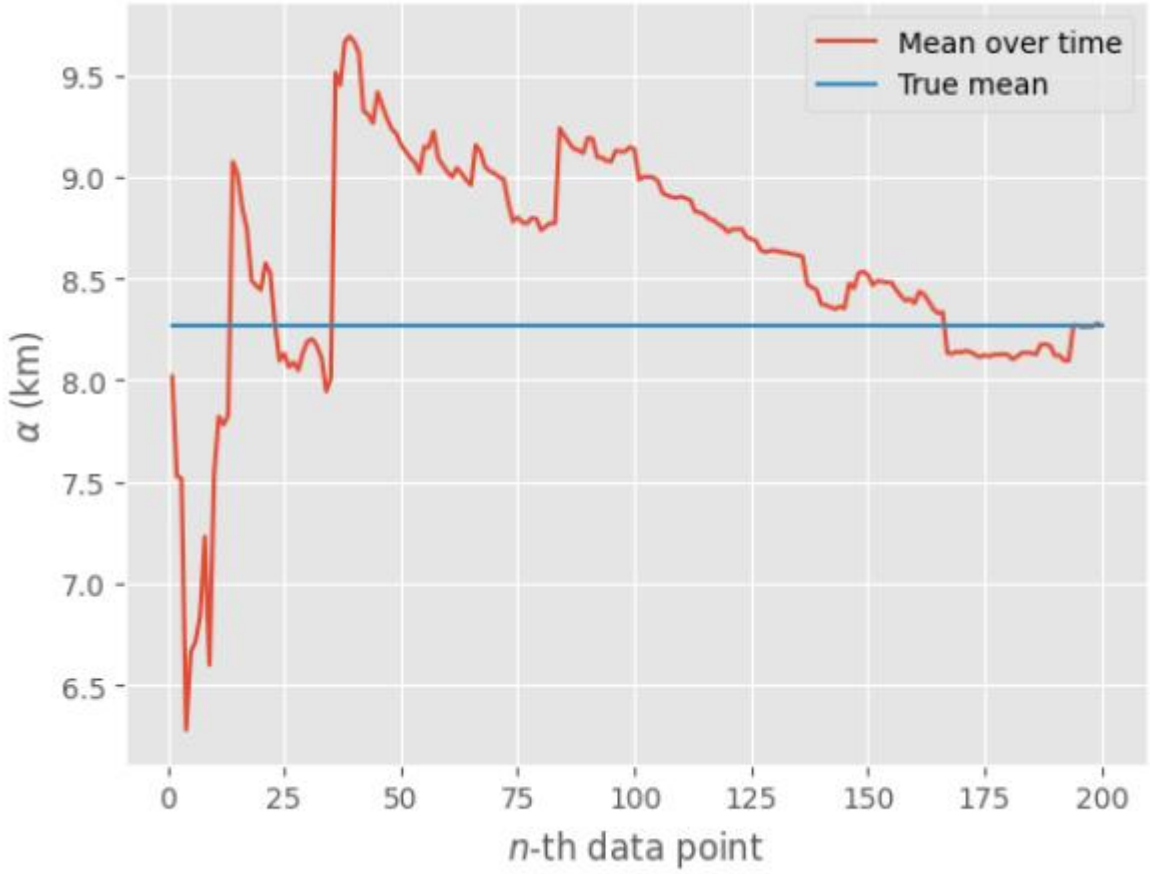
```
-0.18333333333333326
1.1761761761761758
```

```
# Find and print the alpha value corresponding to the maximum likelihood in
'alphas'.
# This identifies the most likely location parameter.
alpha_t = np.random.uniform(0, 10)
beta_t = np.random.uniform(1, 2)
print(alpha_t, beta_t)
```

```
7.7622923552881105 1.1710350426683842
```

```
# Define a function 'location' that calculates the location of a point given an angle, location parameter, and scale parameter.

def location(angle, alpha, beta):

    return beta * np.tan(angle) + alpha


# Define the number of data points 'N'.

N = 200 # Adjusted to a larger value for more data points.


# Generate random angles uniformly distributed between -π/2 and π/2.

angles = np.random.uniform(-np.pi/2, np.pi/2, N)

locations = np.array([location(angle, alpha_t, beta_t) for angle in angles])



# Calculate a list 'mus' containing the mean over time for a series of data points.

mus = [locations[:i + 1].mean() for i in range(N)]

# Create a list 'mean' containing the true mean value repeated for all data points.


mean = [locations.mean()] * (N)

# Create a list 'mean' containing the true mean value repeated for all data points.

X = np.arange(1, N + 1)

# Set the plot style to 'ggplot'.

plt.style.use('ggplot')


# Create line plots for 'mus' and 'mean' over time.

plt.plot(X, mus, label='Mean over time')

plt.plot(X, mean, label='True mean')

# Set labels for the x and y axes and add a legend.

plt.xlabel('$n$-th data point')

plt.ylabel(r'$\alpha$ (km)')
```

```
plt.legend()


plt.savefig('mean_x.png', bbox_inches='tight', dpi=300)
plt.show()
```



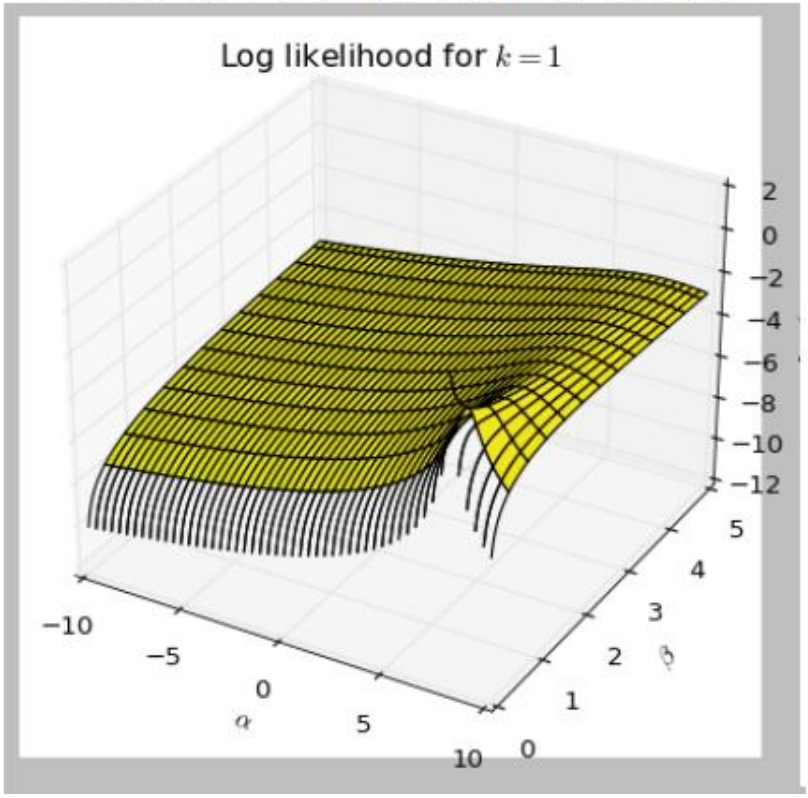

```
# Calculate and print the mean of the 'locations' array.
print(locations.mean())
```

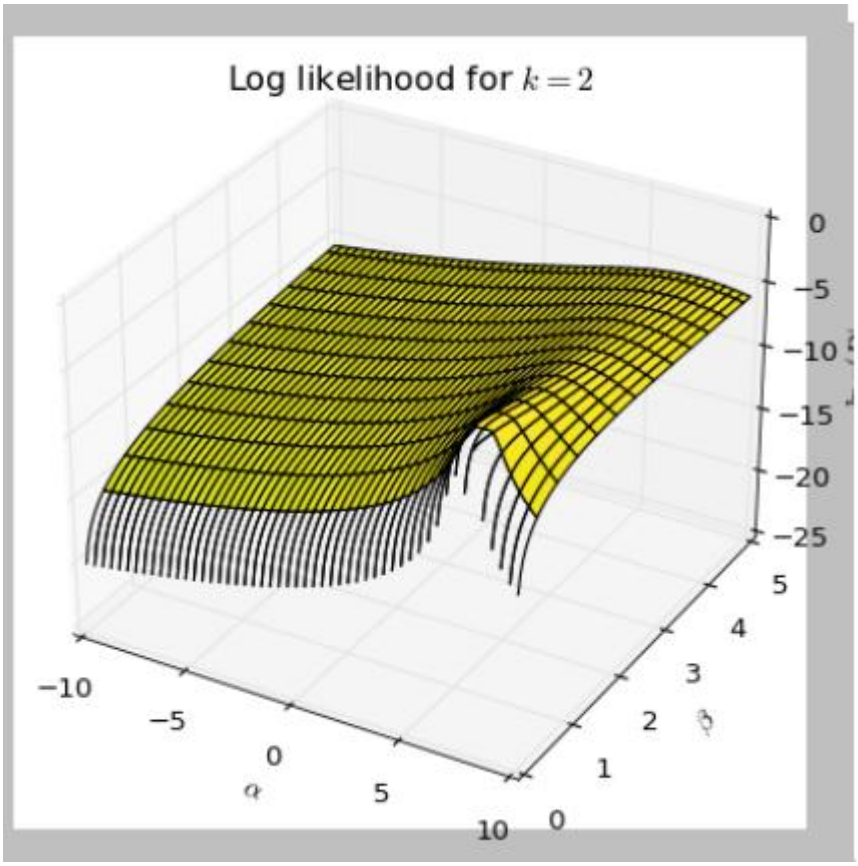
```
8.270979602378144
```

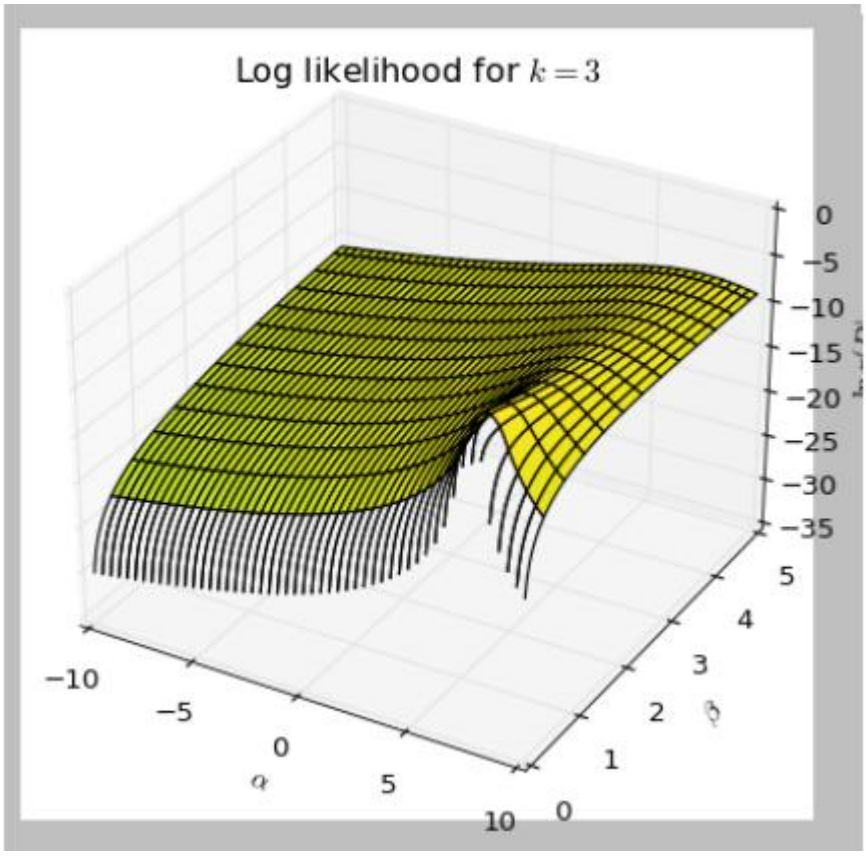
```
plt.style.use('classic')
ks = [1, 2, 3, 20]


# Create a grid of 'alphas' and 'betas' for plotting.
alphas, betas = np.mgrid[-10:10:0.04, 0:5:0.04]
for k in ks:
    x = locations[:k]# Extract the first 'k' elements from the 'locations'
array.
```

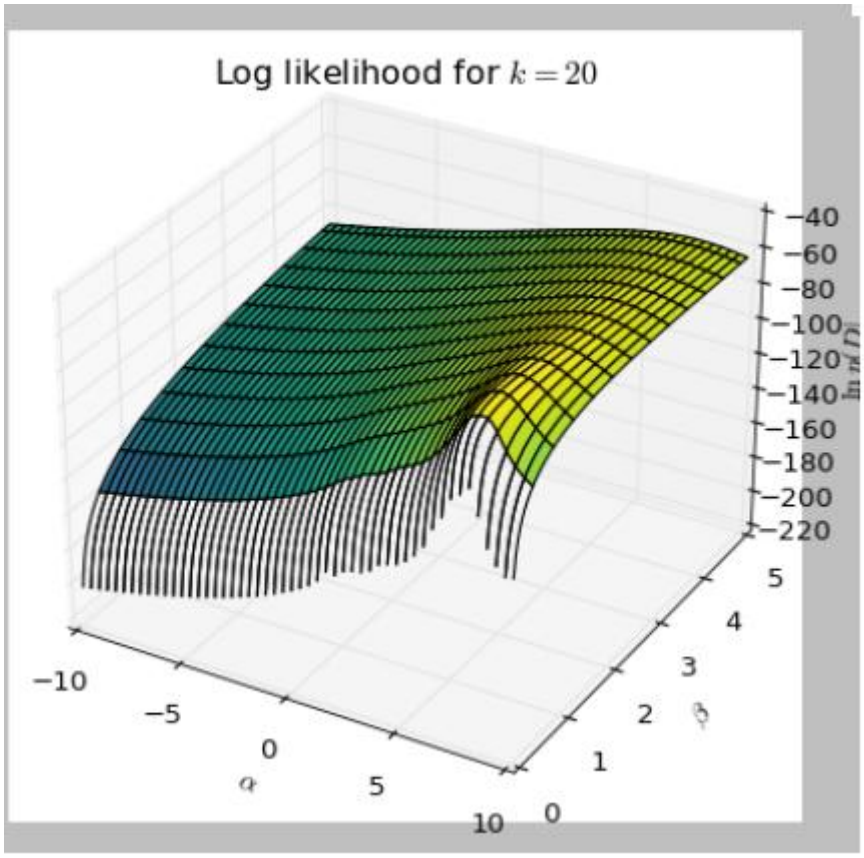
```
    # We only have to calculate the constant once
    likelihood = k * np.log(betas/np.pi)
    for loc in x:
        likelihood -= np.log(betas**2 + (loc - alphas)**2)

    fig = plt.figure()
   ax = fig.add_subplot(projection='3d')
# Create a 3D surface plot of the log-likelihood values.
   ax.plot_surface(alphas, betas, likelihood, cmap=plt.cm.viridis, vmin=-200,
vmax=likelihood.max())
# Set labels for the x and y axes and the z-axis.
    plt.xlabel(r'$\alpha$')
    plt.ylabel(r'$\beta$')
   ax.set_zlabel('$\ln p(D | \alpha, \beta)$')
 # Set the title of the plot based on the current 'k' value.
    plt.title('Log likelihood for $k = {}$'.format(k))
    plt.savefig('logl_{}.png'.format(k), bbox_inches='tight', dpi=300)
    plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot
  txs, tys, tzs = vecw[0]/w, vecw[1]/w, vecw[2]/w
```










```
# Import the necessary module for optimization.

from scipy.optimize import fmin
```

```
# Define a function 'log_likelihood' to calculate the negative log-likelihood
of data given parameters.

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

# Define a function 'plot_maximize_logl' to optimize log-likelihood and create
plots.

def plot_maximize_logl(data, alpha_t, beta_t):

    alphas, betas = [], []

    x = np.arange(len(data))

   for k in x:

# Optimize the log-likelihood for parameters alpha and beta.

        [alpha, beta] = fmin(log_likelihood, (0, 1), args=(data[:k],))

        alphas.append(alpha)

        betas.append(beta)\

 # Set the plot style to 'ggplot'.

    plt.style.use('ggplot')

    plt.plot(x, alphas, label=r'$\alpha$')

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

    plt.plot(x, [alpha_t]*len(data), label=r'$\alpha_t$')

    plt.plot(x, [beta_t]*len(data), label=r'$\beta_t$')

    plt.xlabel('$k$')

    plt.ylabel('location (km)')

    plt.legend()

    plt.savefig('plots/min_logl.png', bbox_inches='tight', dpi=300)

    plt.show()

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

```
import pandas as pd
import numpy as np
# Read data from a CSV file named "train.csv" using ';' as the separator.
a=pd.read_csv("/train.csv",sep=';')
# Print the data read from the CSV file.
print(a)
# Extract the 'hour' column from the DataFrame and store it in variable 'p'.
p=a['hour']
# Calculate the mean value of the 'hour' column.
m = np.mean(p)
# Calculate the standard deviation (sigma) of the 'hour' column.
sd = np.std(p)
# Calculate the variance (sigma square) of the 'hour' column.
var = np.var(p)
# Print the calculated mean, standard deviation, and variance values.
m, sd, var
```

```
          id  year  hour  season  holiday  workingday  weather   temp   atemp  \
0          3  2012    23       3        0           0        2  23.78  27.275
1          4  2011     8       3        0           0        1  27.88  31.820
2          5  2012     2       1        0           1        1  20.50  24.240
3          7  2011    20       3        0           1        3  25.42  28.790
4          8  2011    17       3        0           1        3  26.24  28.790
...      ...   ...   ...     ...      ...         ...      ...    ...     ...
7684   10882  2012    18       1        0           1        1  13.94  15.150
7685   10883  2012     3       1        0           1        1   9.02  11.365
7686   10884  2012    15       2        0           0        1  21.32  25.000
7687   10885  2011    19       4        0           1        1  12.30  14.395
7688   10886  2012    21       3        0           1        1  30.34  34.850

      humidity  windspeed  count
0           73    11.0014    133
1           57     0.0000    132
2           59     0.0000     19
3           83    19.9995     58
4           89     0.0000    285
...        ...        ...    ...
7684        42    22.0028    457
7685        51    11.0014      1
7686        19    27.9993    626
7687        45    15.0013    217
7688        66     7.0015    381

[7689 rows x 12 columns]
(11.56535310183379, 6.915326938018648, 47.82174665968637)
```

```python
# Extract the 'temp' data from an array or DataFrame named 'a'.
t = np.array(a['temp'])

# Calculate the mean value of the 'temp' data.
tm = np.mean(t)

# Calculate the standard deviation (sigma) of the 'temp' data.
tsd = np.std(t)

# Calculate the variance of the 'temp' data.
tvar = np.var(t)

# Define a value 'x'.
```

```
x = 29
```

```
# Calculate the natural logarithm of the square root of 2π.
l = np.log(np.sqrt(2 * 3.14))
```

```
# Calculate the natural logarithm of the standard deviation.
e = np.log(tsd)
# Calculate the squared difference between 'x' and the mean 'tm'.
f = (x - tm) ** 2
# Calculate a term 'g' using the variance.
g = 2 * (tvar ** 2)
# Calculate the fraction 'h' by dividing 'f' by 'g'.
h = f / g
# Calculate the final result by subtracting 'h' from '-l-e'.
i = -l - e
result = i - h
# Print the result.
print(result)
```

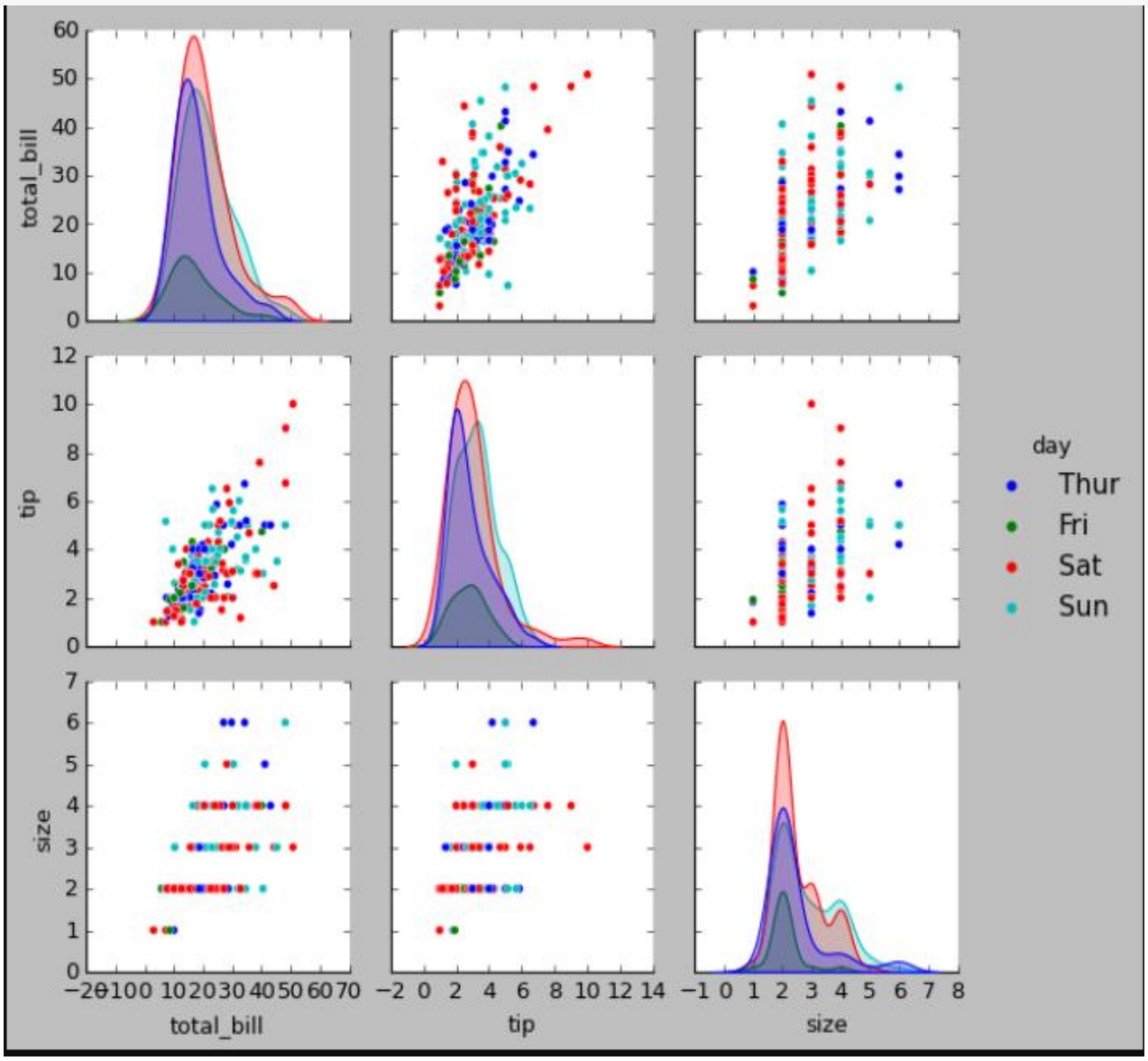
```
-2.9860025235468406
```

```
import seaborn
import matplotlib.pyplot as plt
# Load a sample dataset ('tips' dataset) provided by Seaborn.
df = seaborn.load_dataset('tips')
```

```
# Create a pair plot of the dataset with data points colored by the 'day'
category.
seaborn.pairplot(df, hue='day')
```

```
# Display the pair plot.
```

```
plt.show()
```



LAB 03

Linear Regression Implementation

```
# 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()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	NaN	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

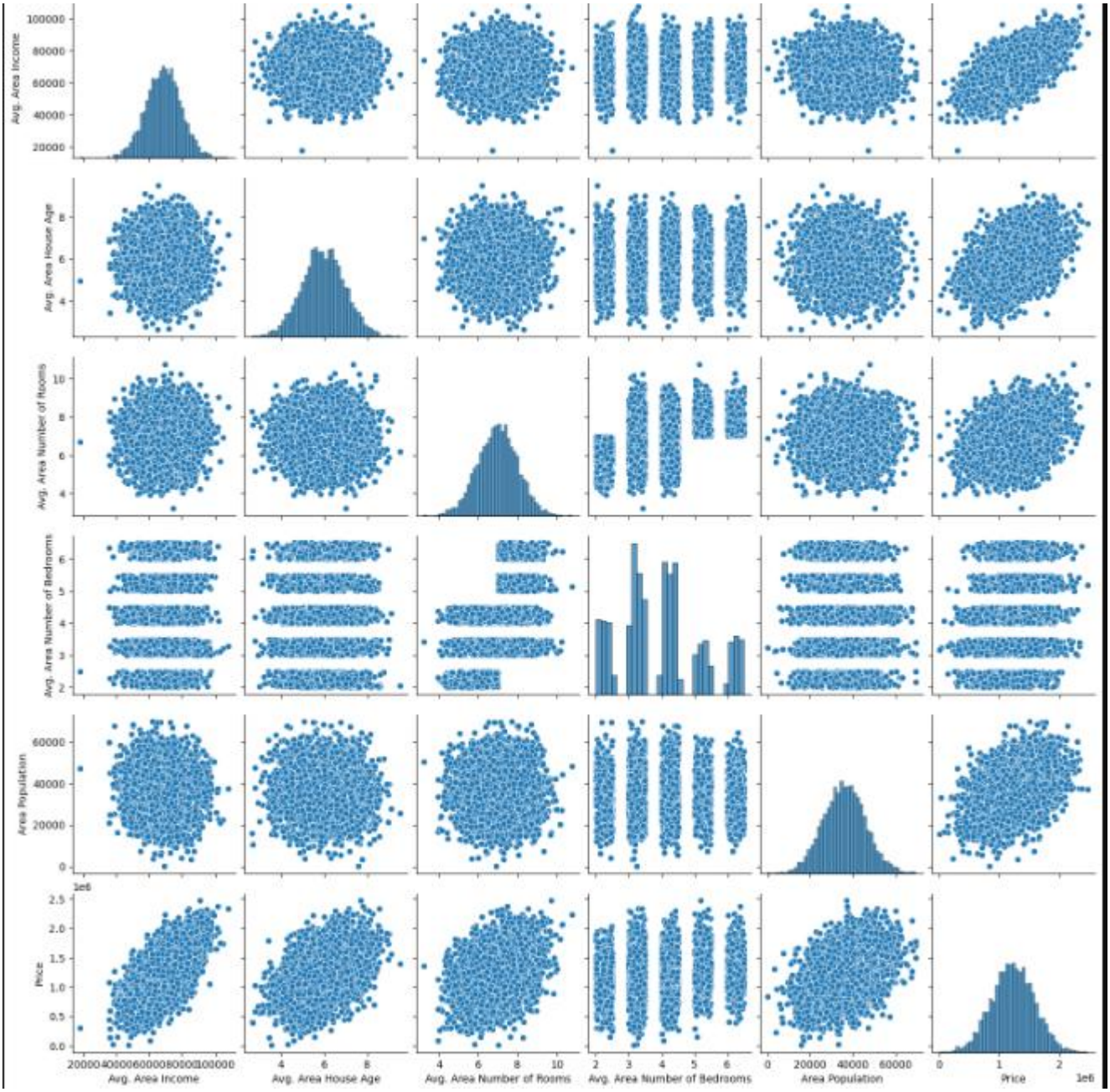
```
# Display comprehensive information about the DataFrame 'df'
df.info(verbose=True)
```

```
<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
```

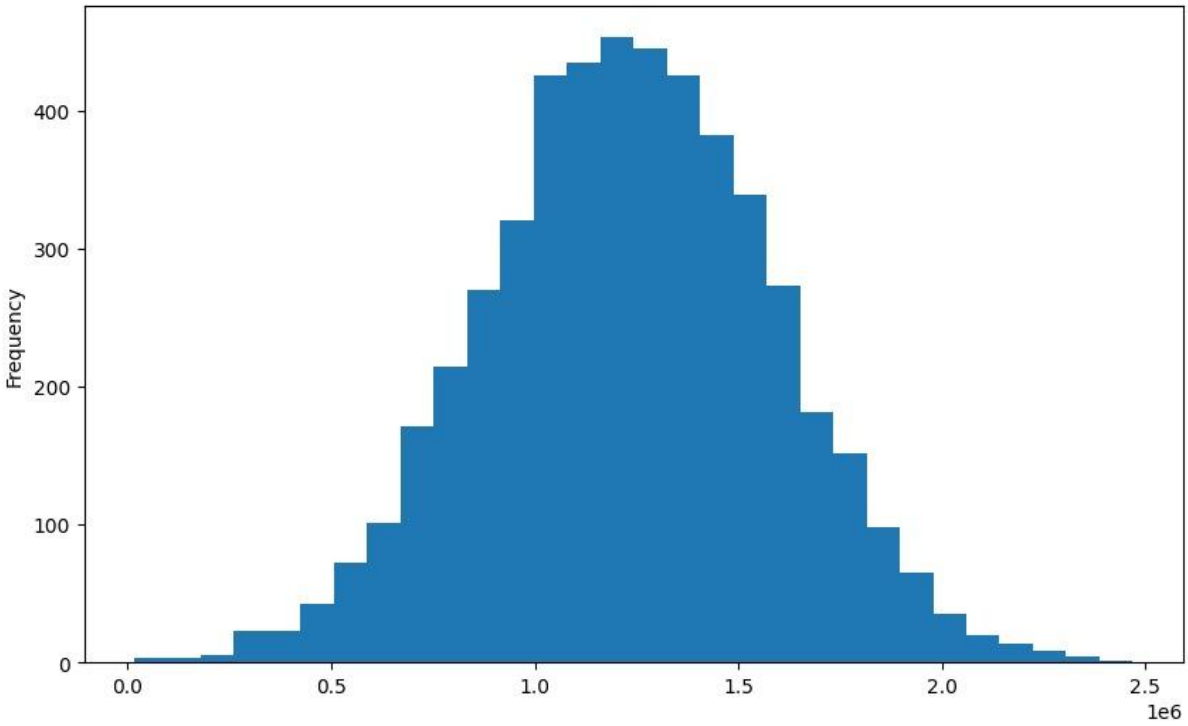
```
# Generate a statistical summary of the DataFrame 'df' with specific percentiles

df.describe(percentiles=[0.1,0.25,0.5,0.75,0.9])
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	4994.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977509	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991637	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
10%	55047.633976	4.698016	5.681951	2.310000	23502.845263	7.720318e+05
25%	61480.562390	5.322274	6.299250	3.140000	29403.928700	9.975771e+05
50%	68804.286405	5.971563	7.002902	4.050000	36199.406690	1.232669e+06
75%	75783.338665	6.650932	7.665871	4.490000	42861.290770	1.471210e+06
90%	82081.188286	7.244251	8.274222	6.100000	48813.618635	1.684621e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

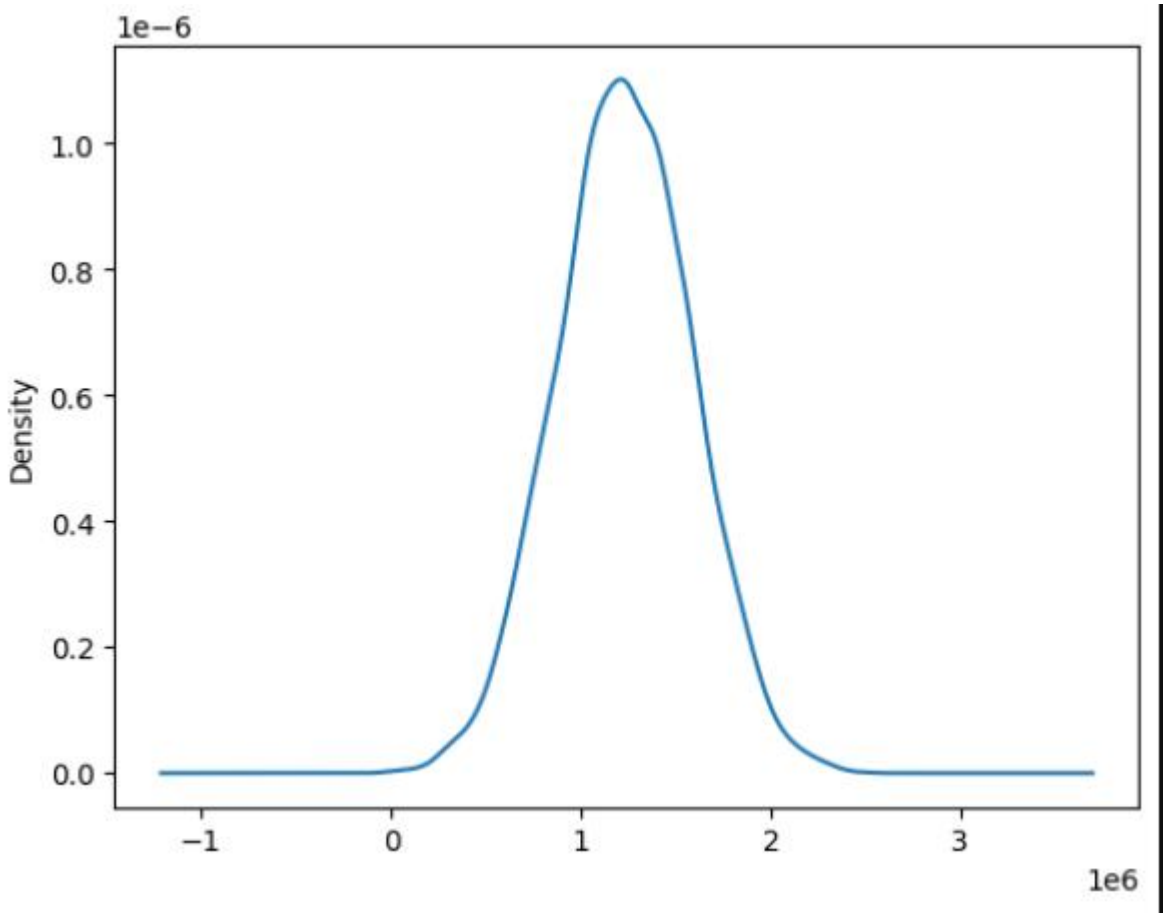
```
#display columns

df.columns
```




```
# 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))
```




```
# Create a density plot for the 'Price' column to visualize its probability density distribution

df['Price'].plot.density()
```




```
# Calculate the correlation matrix for the DataFrame 'df'
df.corr()
```

	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	-0.001444	-0.011032	0.019788	-0.016234	0.639734
Avg. Area House Age	-0.001444	1.000000	-0.009242	0.006497	-0.018711	0.452806
Avg. Area Number of Rooms	-0.011032	-0.009242	1.000000	0.462695	0.002040	0.335664
Avg. Area Number of Bedrooms	0.019788	0.006497	0.462695	1.000000	-0.022168	0.171071
Area Population	-0.016234	-0.018711	0.002040	-0.022168	1.000000	0.408556
Price	0.639734	0.452806	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)
```


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

```
['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)
```

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

```
# Display the first few rows of the feature set 'X'

X.head()
```

	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()
```

```
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)
```

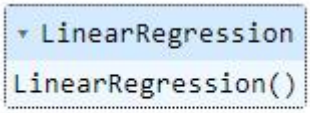
```
# Print the size (dimensions) of the testing variable set 'y_test'

print("Test variable set size:", y_test.shape)
```

```
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)
```




```
# 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_)
```

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

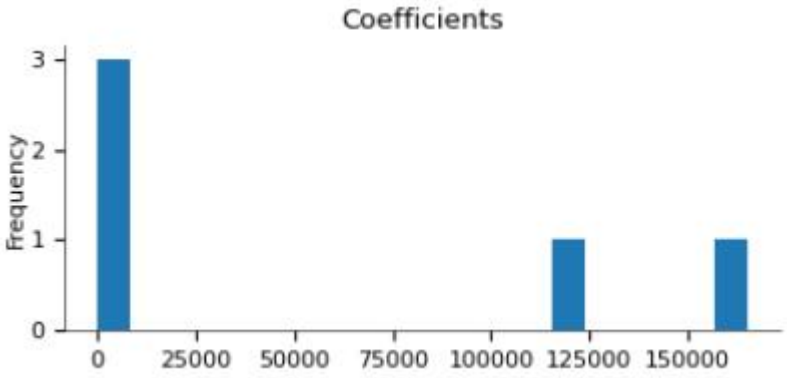
```
# Print the coefficients of the linear regression model stored in the 'lm' variable

print("The coefficients of the linear model:", lm.coef_)
```

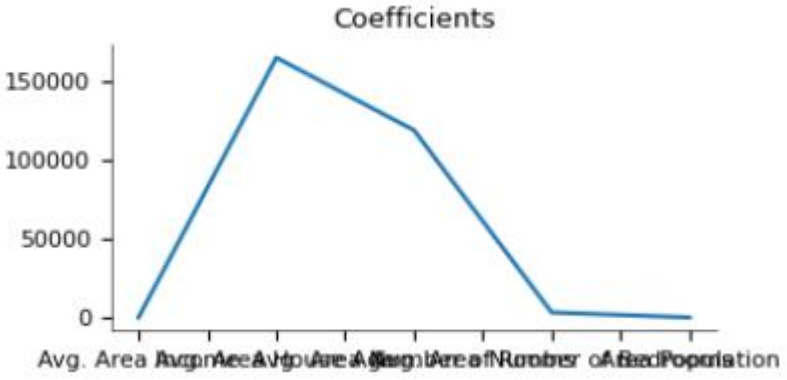
```
The coefficients of the linear model: [2.15976020e+01 1.65201105e+05 1.19061464e+05 3.21258561e+03
 1.52281212e+01]
```

```
# 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
```

	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

Distributions

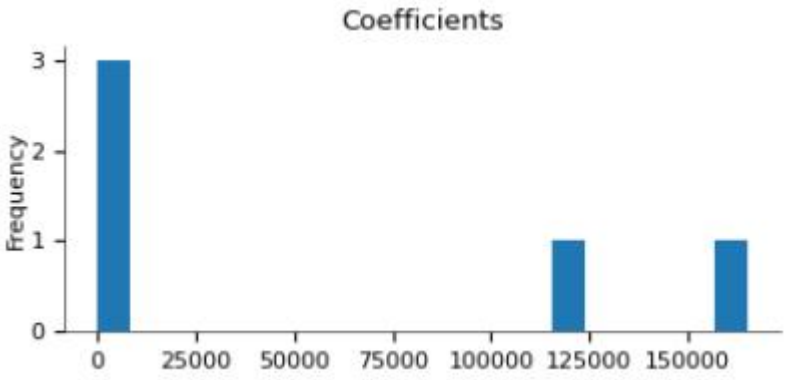


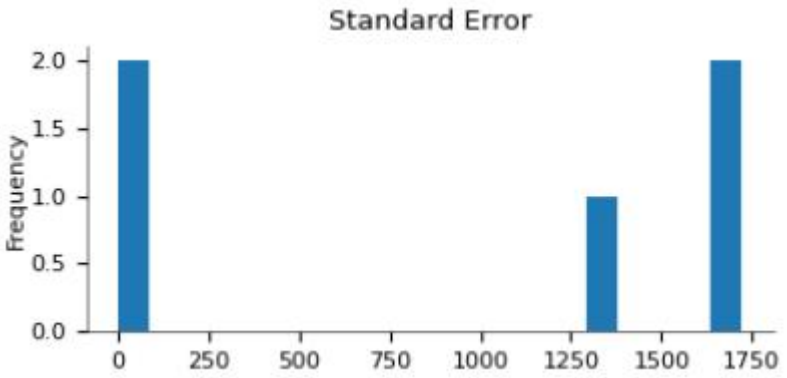


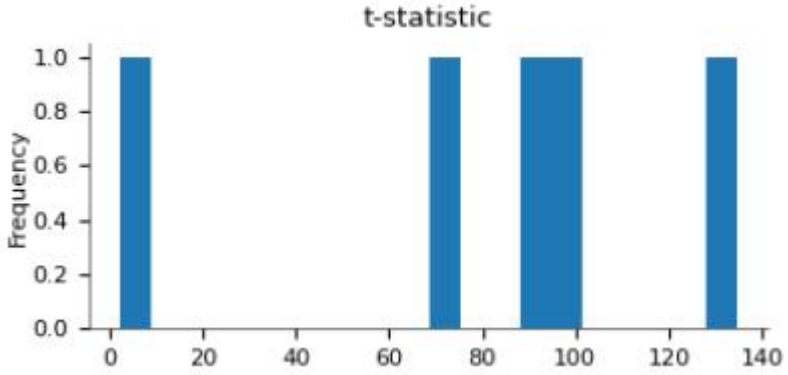

```
# Calculate the standard error for each coefficient

n = X_train.shape[0]  # Number of observations
```

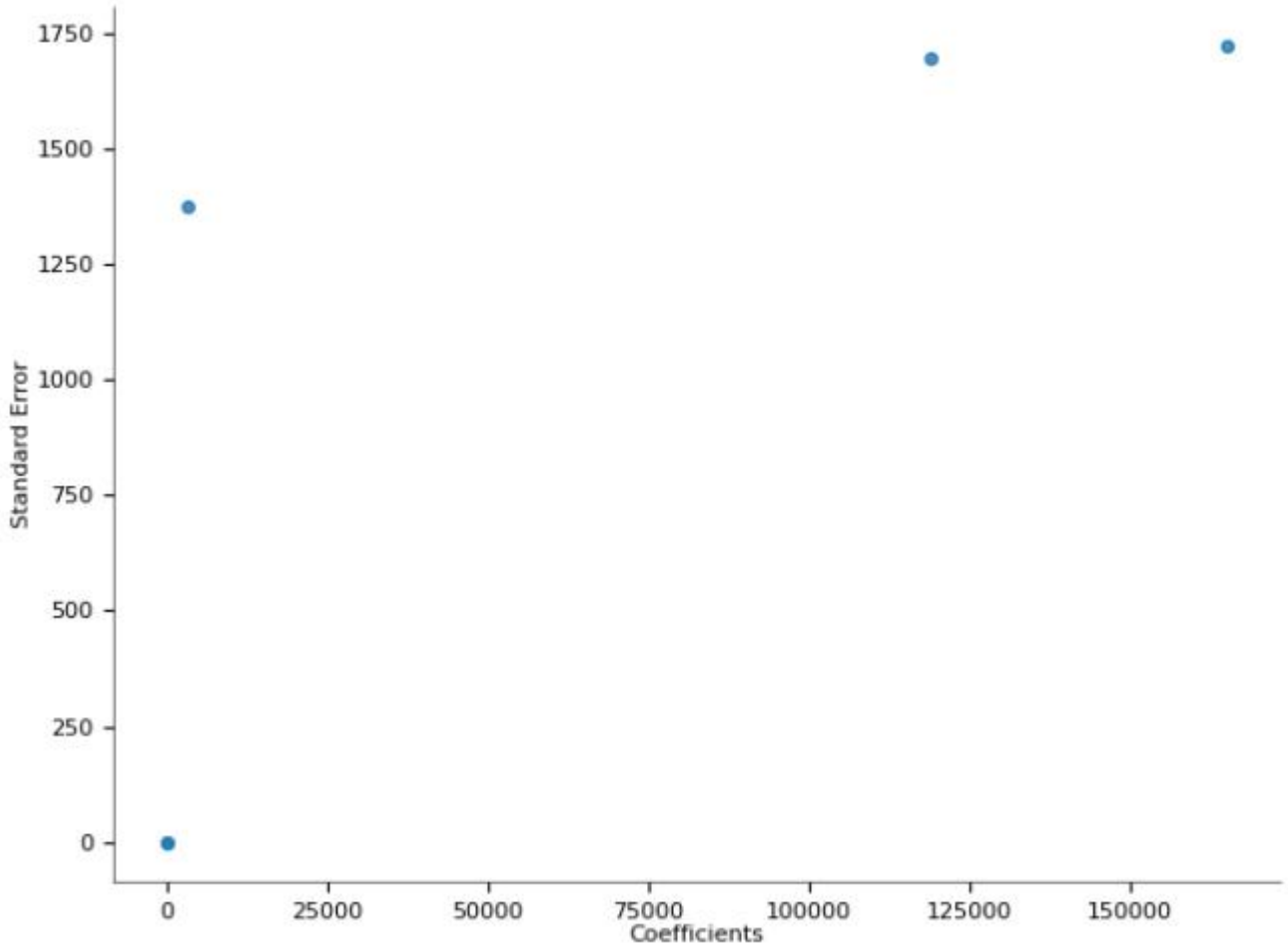
```
k = X_train.shape[1]  # Number of features
dfN = n - k  # Degrees of freedom for the residuals

# 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
```

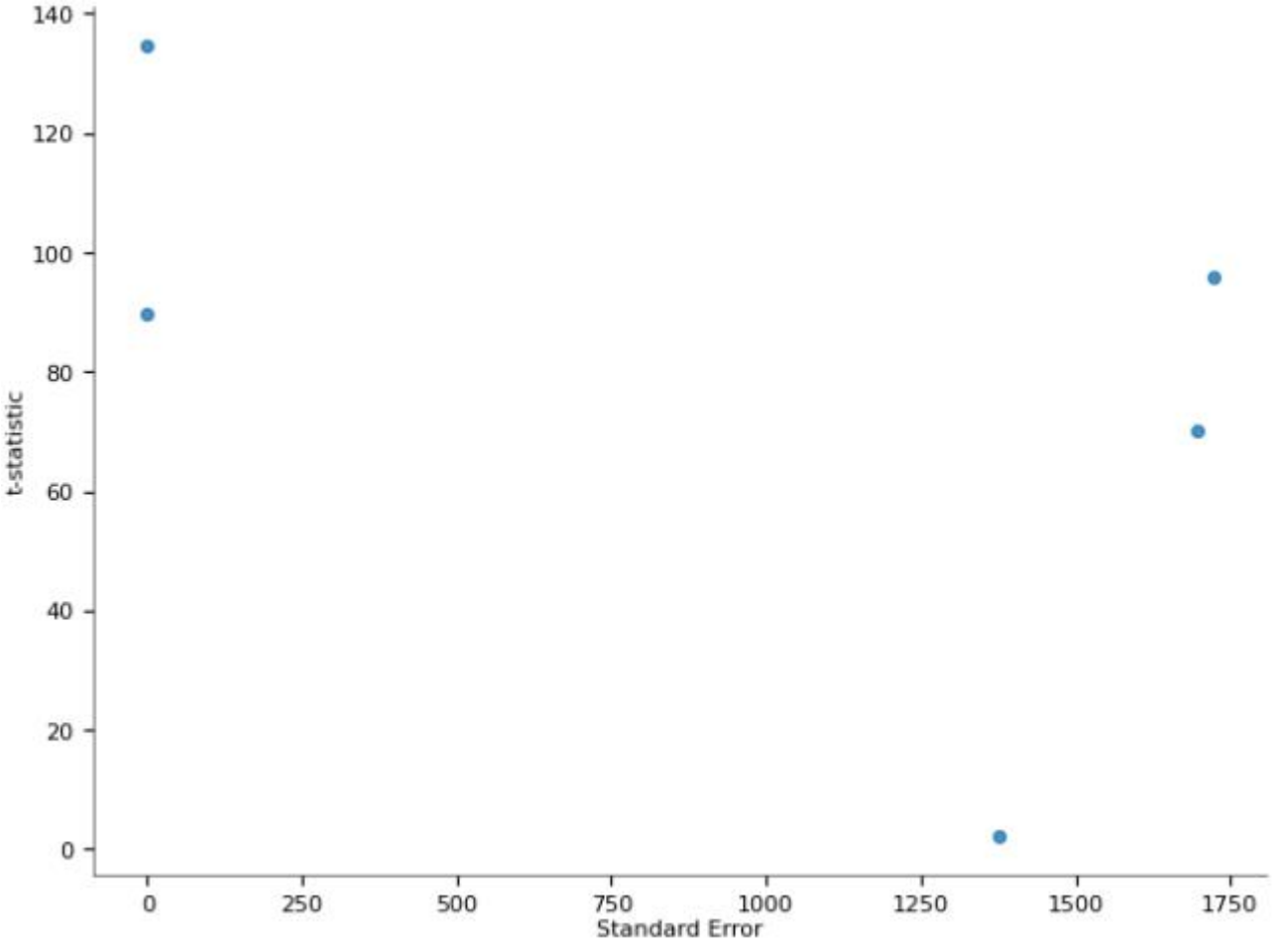
	Coefficients	Standard Error	t-statistic
Avg. Area Income	21.597602	0.160361	134.681505
Avg. Area House Age	165201.104954	1722.412068	95.912649
Avg. Area Number of Rooms	119061.463868	1696.546476	70.178722
Avg. Area Number of Bedrooms	3212.585606	1376.451759	2.333962
Area Population	15.228121	0.169882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

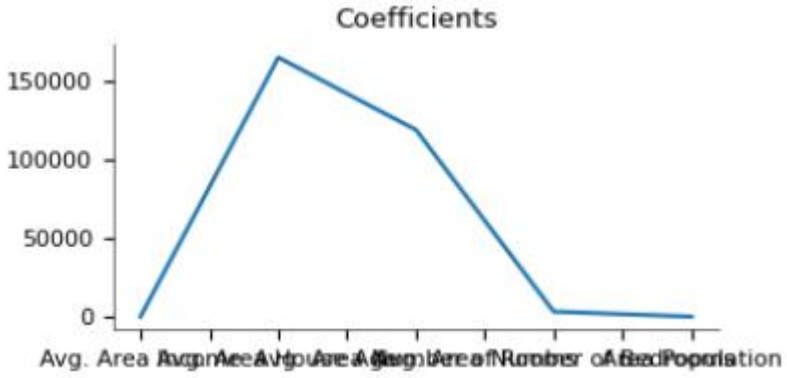


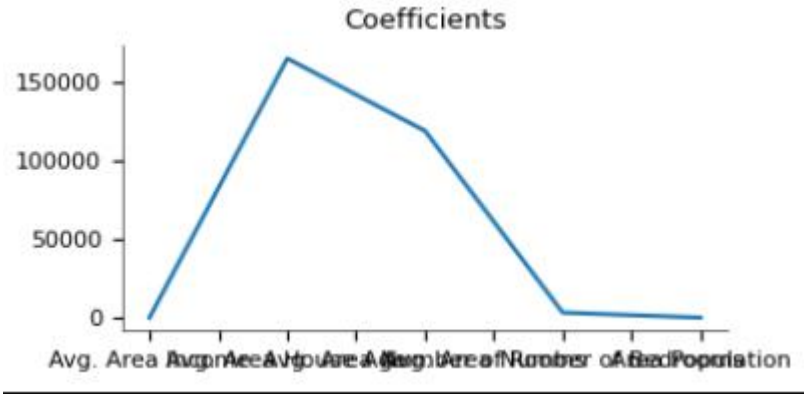



t-statistic
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140


Standard Error
1750
1500
1250
1000
750
500
250
0
0
25000
50000
75000
100000
125000
150000
Coefficients


140
120
100
80
60
40
20
0
t-statistic
0
250
500
750
1000
1250
1500
1750
Standard Error


Coefficients
150000
100000
50000
0
Avg. Area

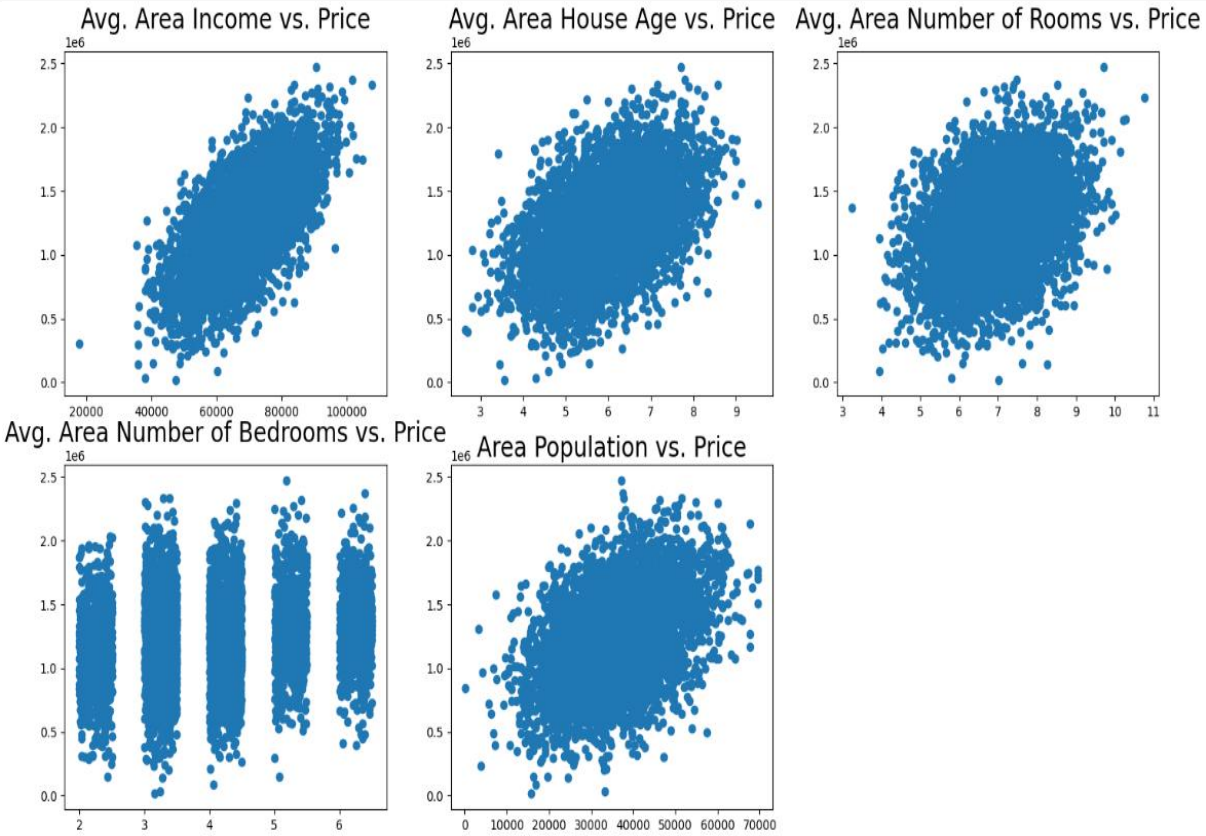



```
# Print a message indicating the features arranged in order of importance for predicting house prices

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))
```

```
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})
```




```
# 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))
```

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

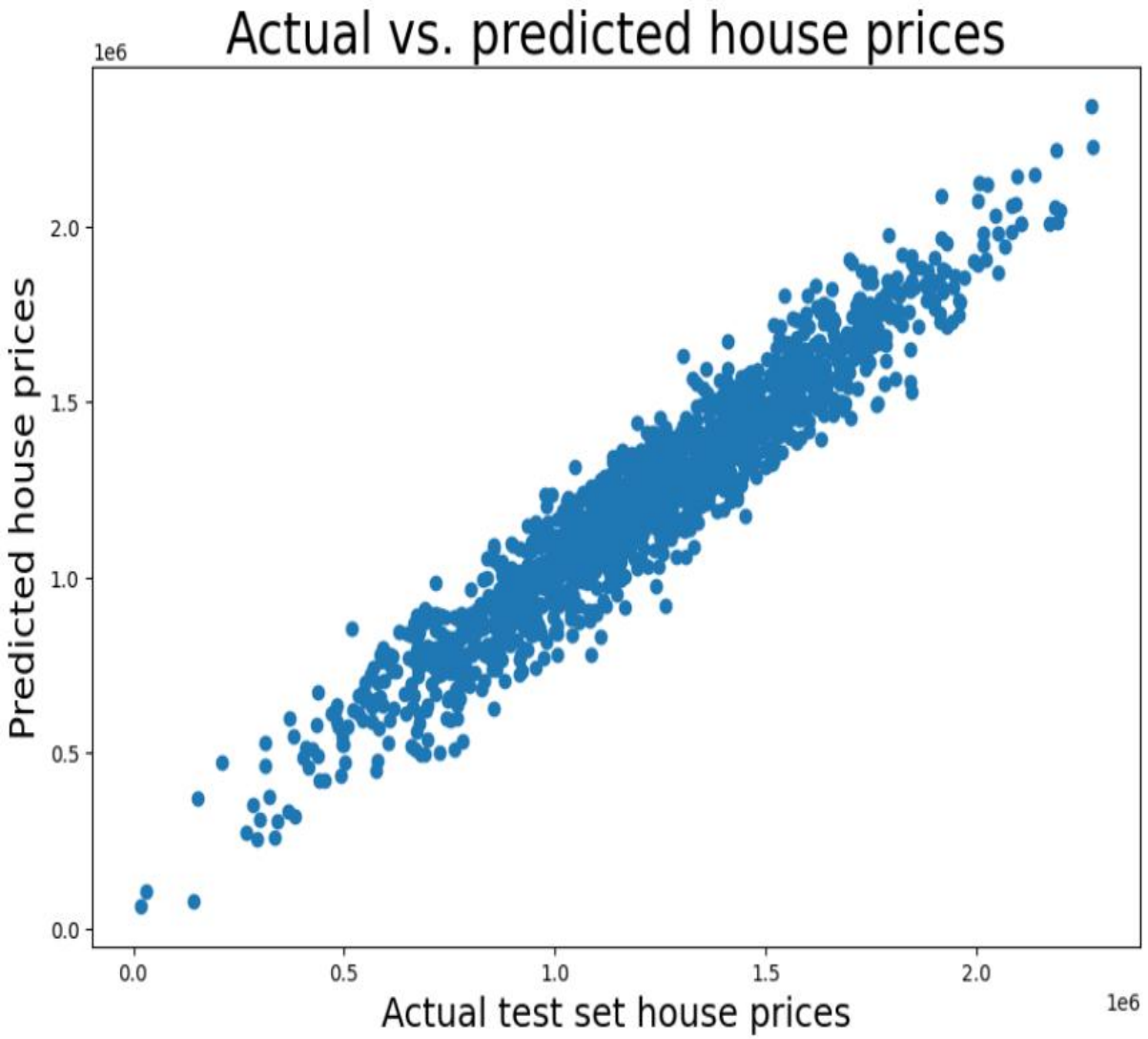
```
# Generate predictions using the linear regression model on the test data

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)
```

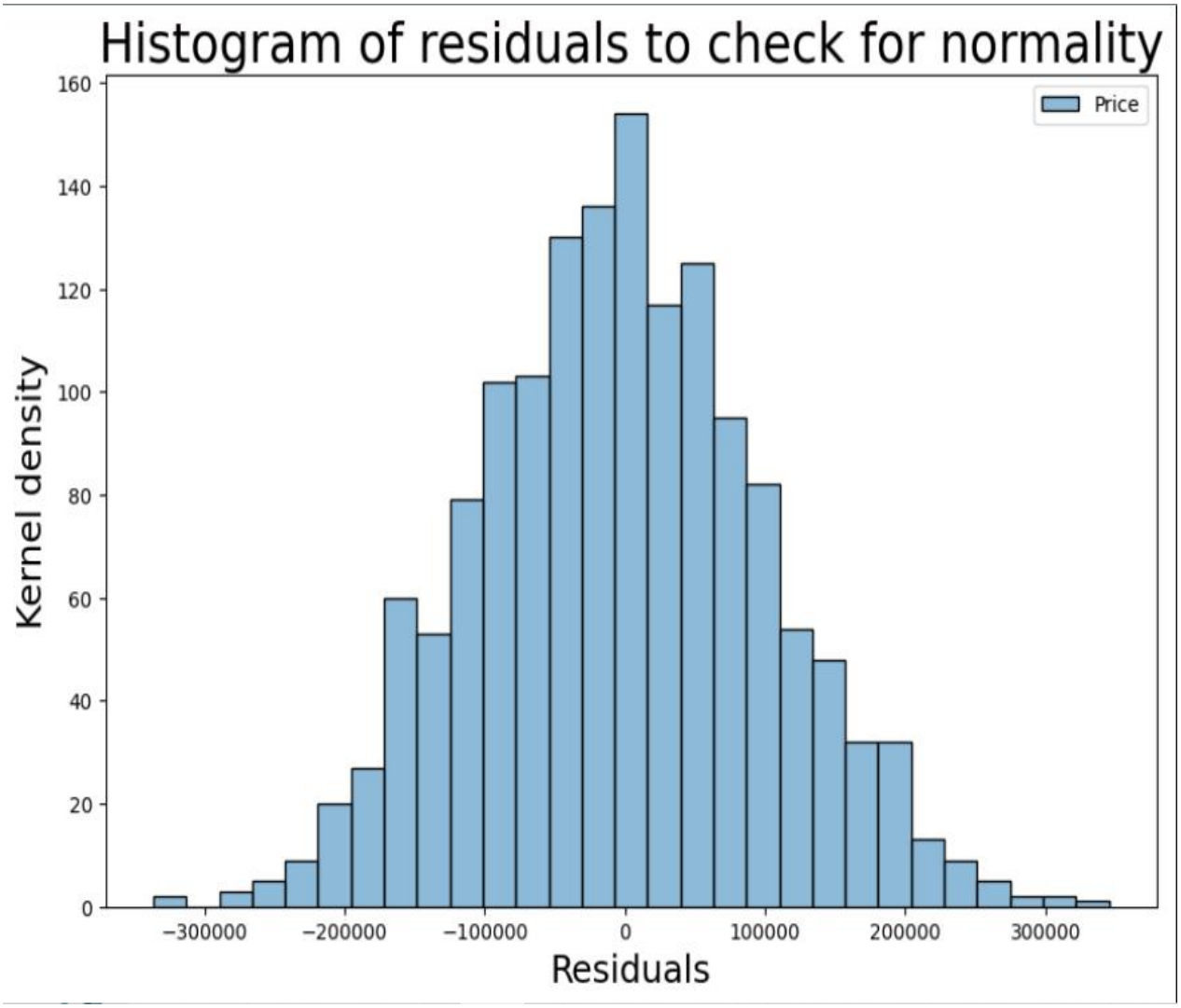
```
Type of the predicted object: <class 'numpy.ndarray'>
Size of the predicted object: (1500,)
```

```
# 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)
```

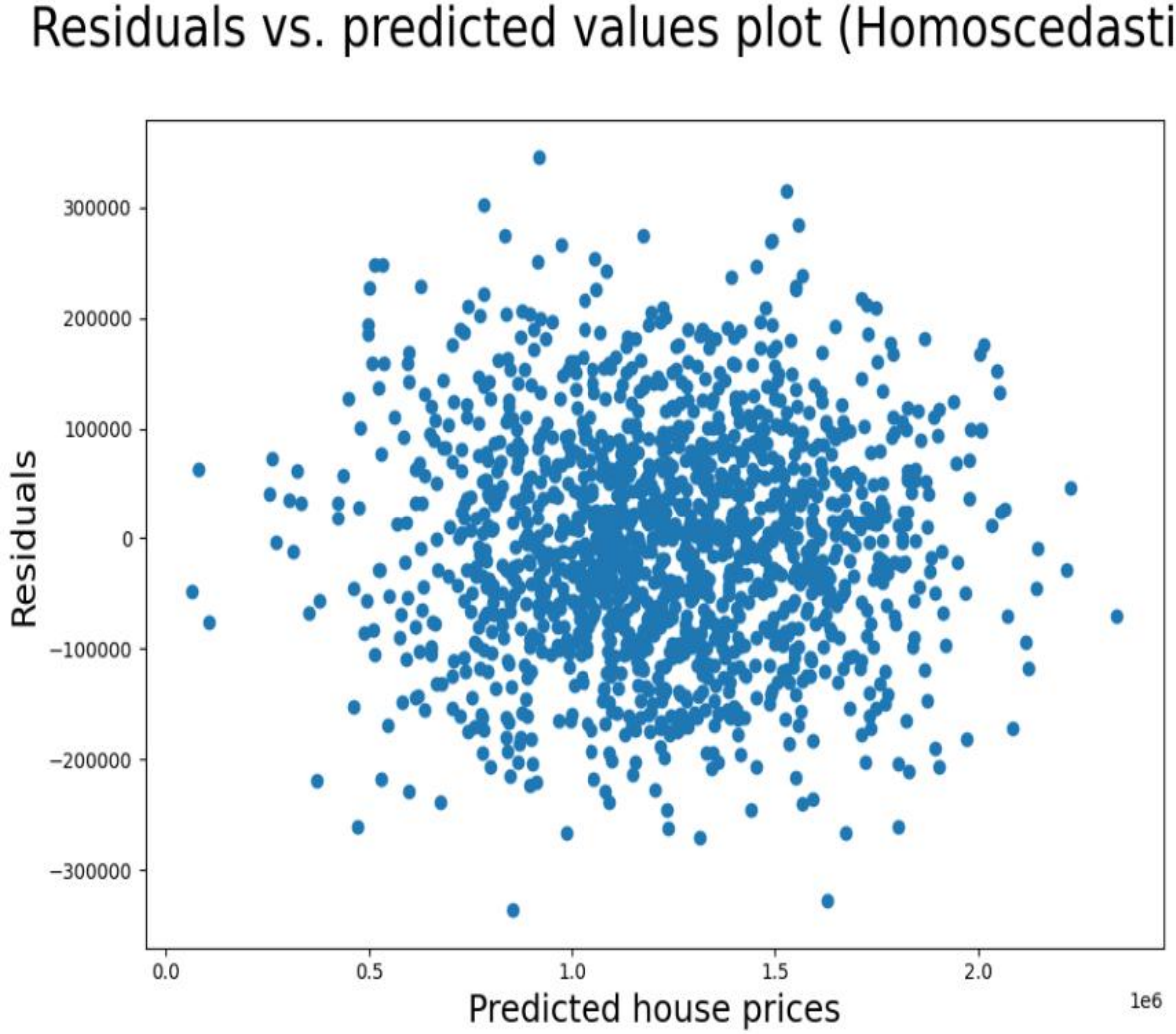



```
# Create a figure for the histogram and kernel density plot
plt.figure(figsize=(10, 7))


# 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])
```




```
# 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)
```




```
# 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)))
```

```
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))
```

```
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)
```

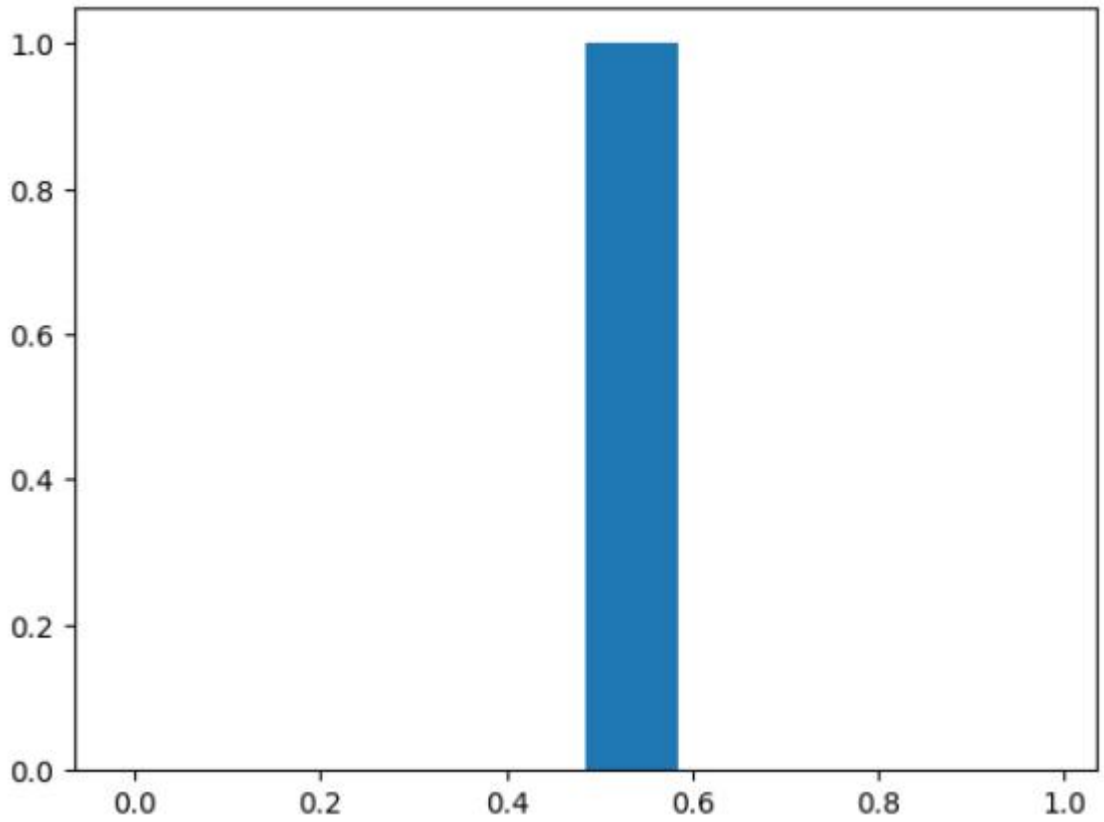
```
# Print the minimum and maximum scaled values
print(min, max)
```

```
 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)
```

```
(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

Linear regression using a pre-defined library. Comparative analysis of both implementations.

```
import nbconvert
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Enable inline plotting of matplotlib figures
%matplotlib inline


# Read the Titanic training dataset from a CSV file
train = pd.read_csv('/content/titanic_train.csv')
```

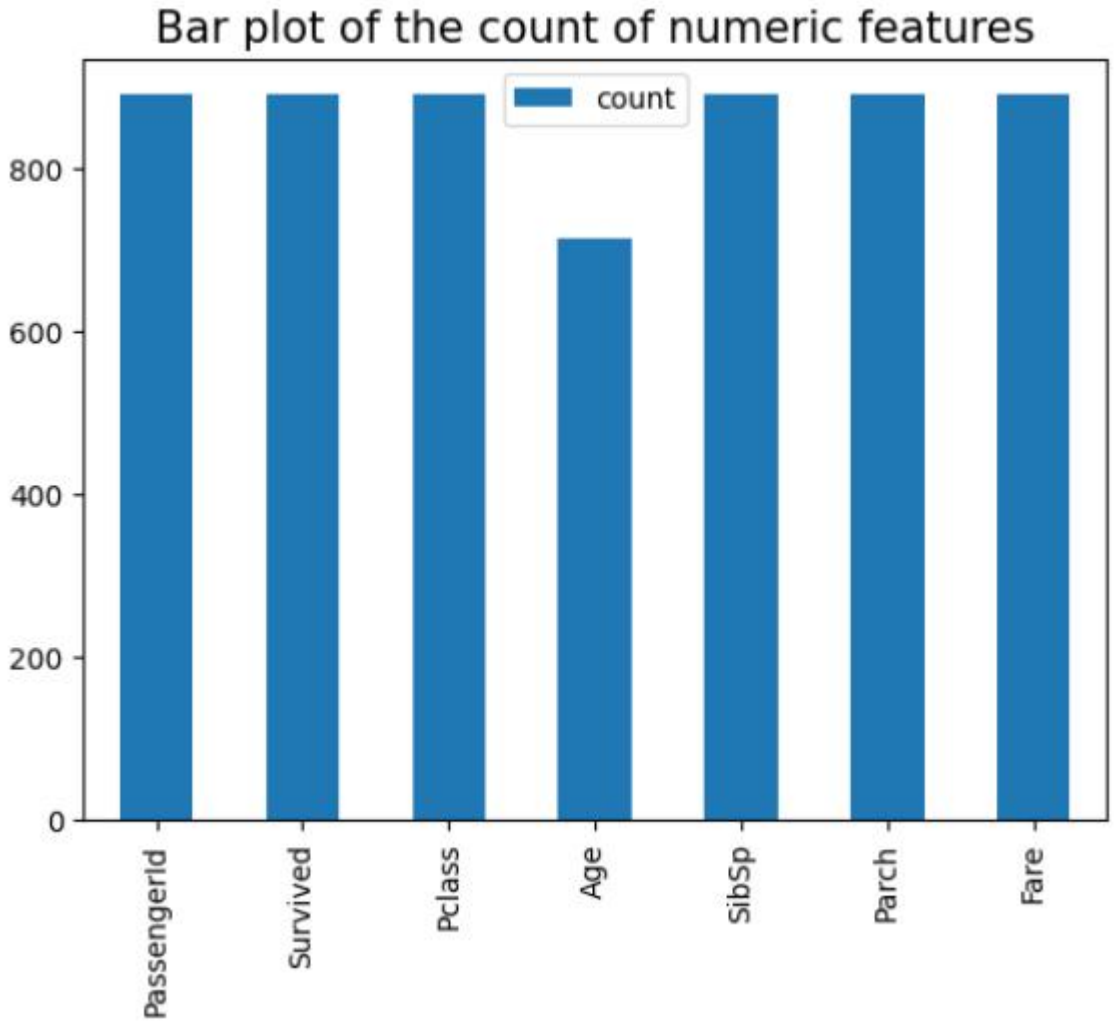
```
# Display the first few rows of the dataset to get a quick overview
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
# Get information about the dataset, such as column data types and non-null counts
t=train.info()
```

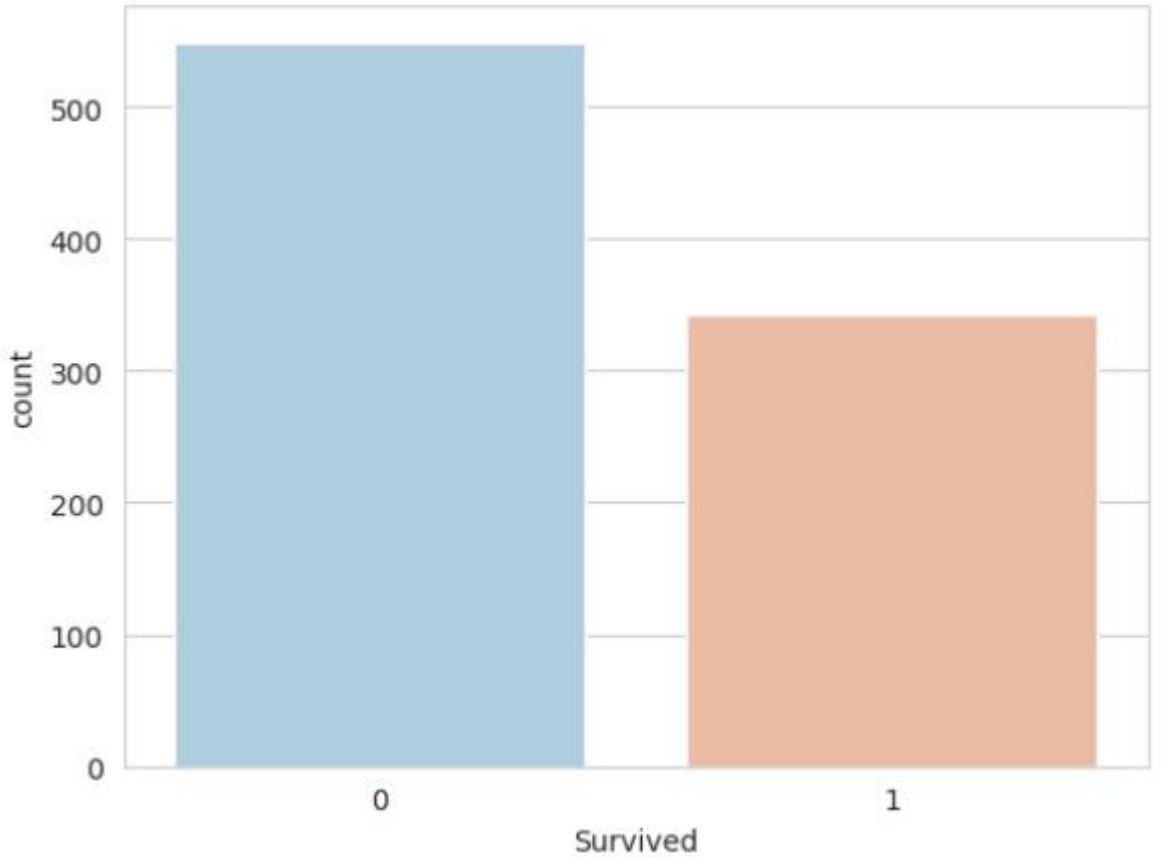
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

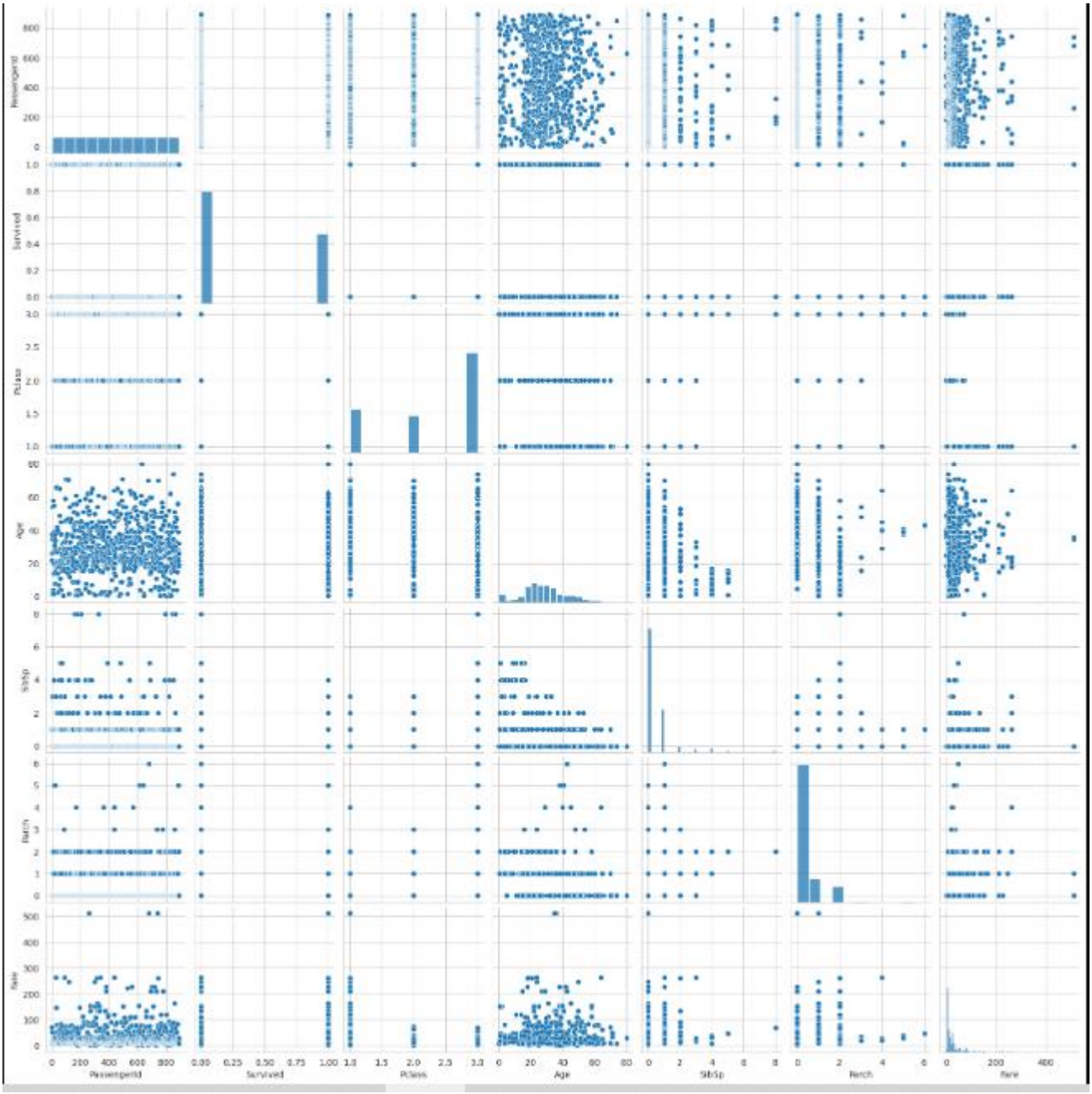
```
# Generate basic statistics for the numeric columns in the dataset
d=train.describe()
# Transpose the summary statistics for better visualization
dT = d.T


# Create a bar plot showing the count of data points for each numeric feature
dT.plot.bar(y='count')
plt.title("Bar plot of the count of numeric features", fontsize=15)
```



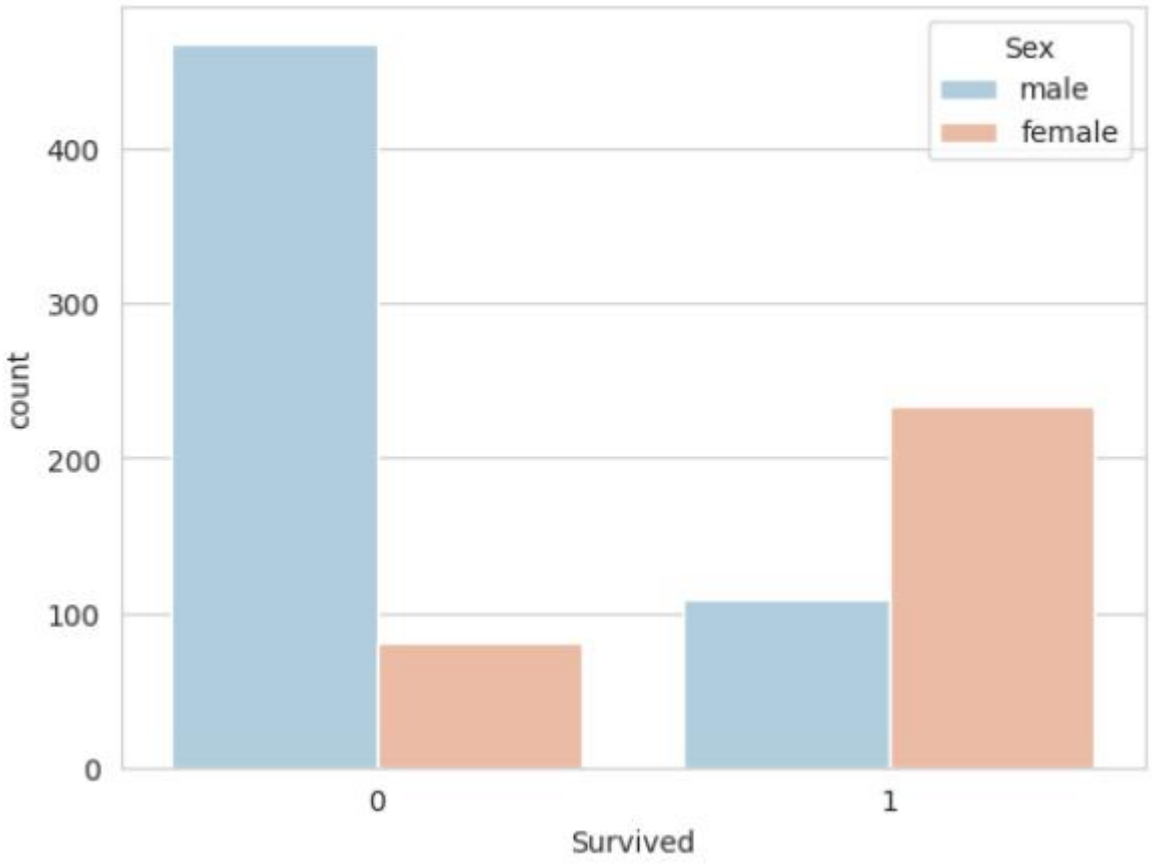

```
# Set the style for Seaborn plots
sns.set_style('whitegrid')

# Create a count plot to visualize the distribution of 'Survived' column
sns.countplot(x='Survived', data=train, palette='RdBu_r')
```

```
# Create a pair plot to explore pairwise relationships between numeric features
sns.pairplot(train)
```

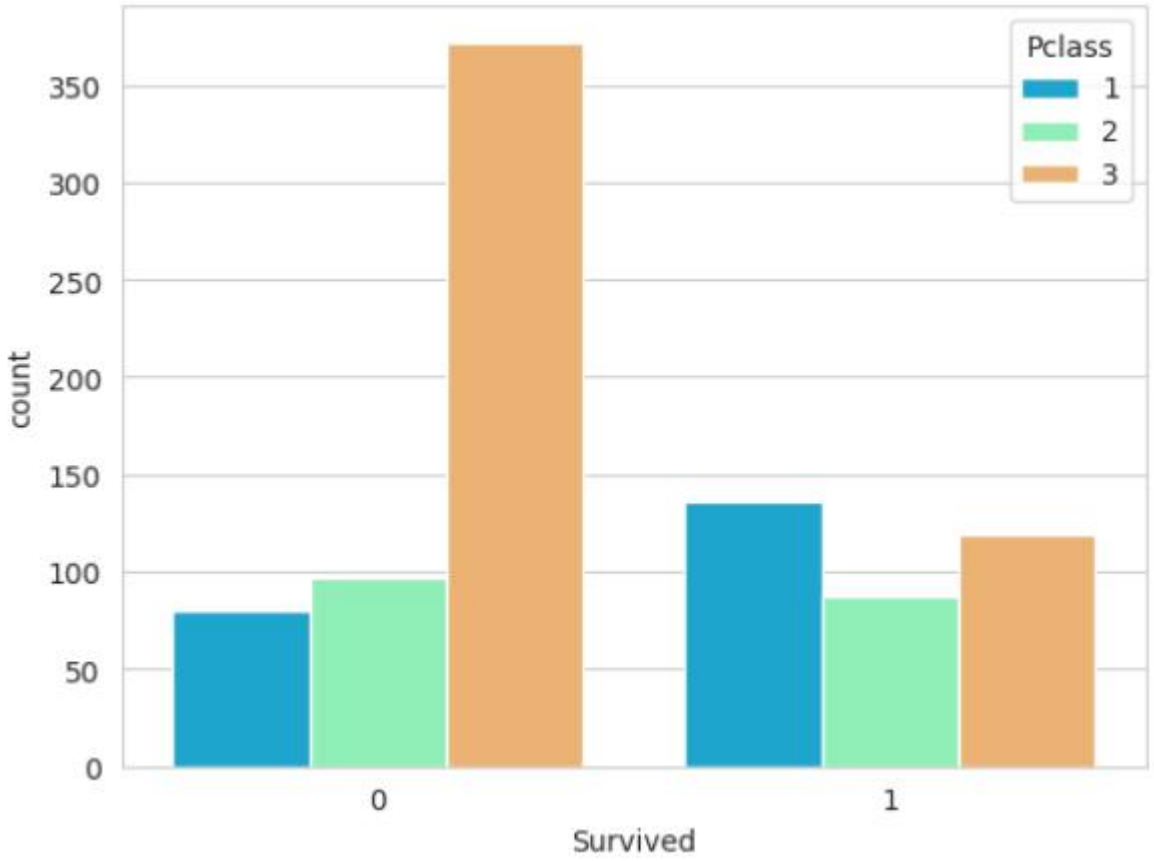

500
400
300
200
100
0
count
0
1
Survived



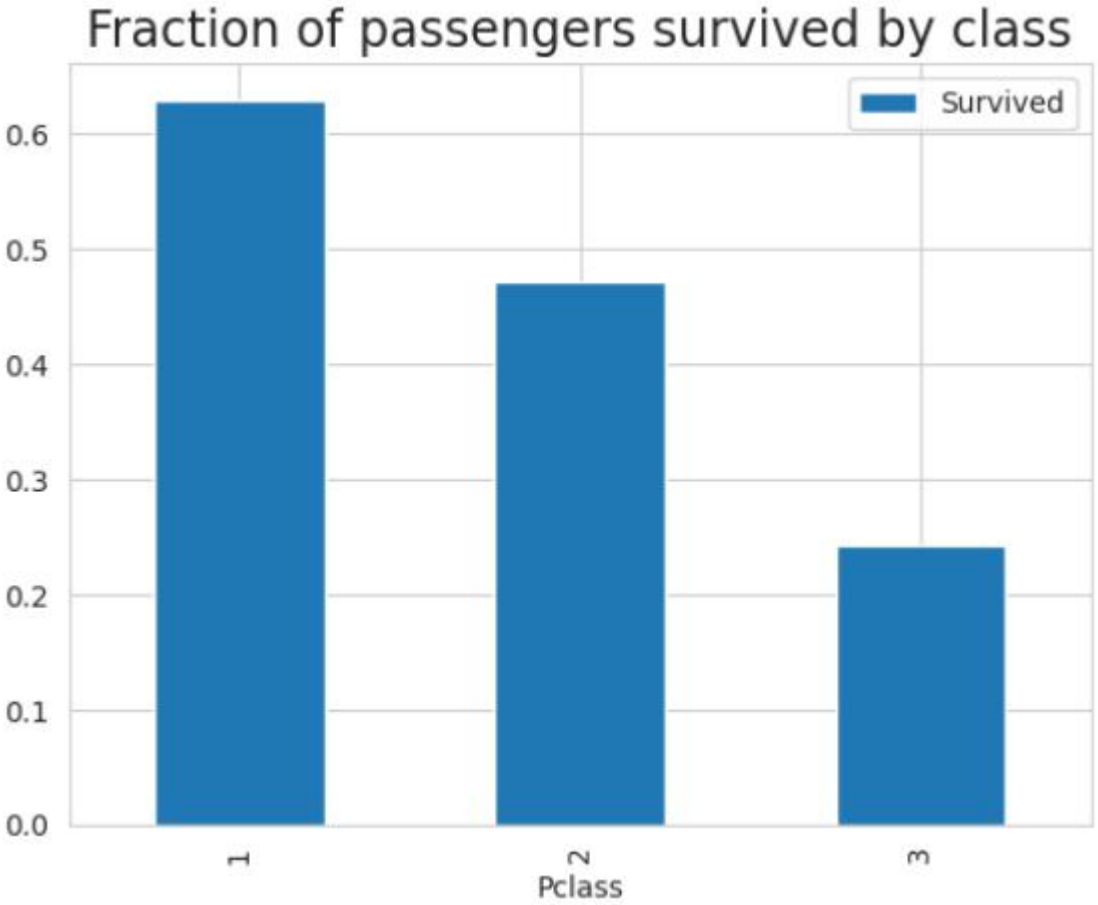

```
# Set the style for Seaborn plots

sns.set_style('whitegrid')

# Create a count plot to visualize the distribution of 'Survived' with 'Sex' as
a hue

sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```




```
# Set the style for Seaborn plots
sns.set_style('whitegrid')
# Create a count plot to visualize the distribution of 'Survived' column
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```




```
# Calculate and visualize the fraction of passengers survived by class
f_class_survived = train.groupby('Pclass')['Survived'].mean()
f_class_survived = pd.DataFrame(f_class_survived)

# Create a bar plot to show the fraction of passengers survived by class
f_class_survived.plot.bar(y='Survived')
plt.title("Fraction of passengers survived by class", fontsize=17)
```


Fraction of passengers survived by class
Survived
0.6
0.5
0.4
0.3
0.2
0.1
0.0
1
2
3
Pclass

LAB 05

Implementation of a Project by taking a data set for any one of the Linear/Logistic/SVM/KNN Models

```
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

%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.rcParams.update({'font.size':16})
```

```
# Import the necessary libraries and modules for the code.
housing = fetch_california_housing()  # Load the California housing dataset.
```

```
dir(housing)#housing.target,DESCR,housing.feature
```

```
['DESCR', 'data', 'feature_names', 'frame', 'target', 'target_names']
```

```
# Check the shape of the data, which represents the features of the dataset.
housing.data.shape
```

```
(20640, 8)
```

```
# Check the shape of the target values, which are the housing prices to be predicted.
```

```
housing.target.shape
```

```
(20640,)
```

```
# Create an instance of the Ridge regression model for later use.
ridgereg = linear_model.Ridge()

# Split the California housing 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)

# Fit a Ridge Regression model to the training data.
ridgereg.fit(X_train, y_train)
```

```
Ridge
Ridge()
```

```
# Calculate the mean squared error for the predictions on the training data.
train_mse = metrics.mean_squared_error(y_train, ridgereg.predict(X_train))
```

```
0.5233577422311327
```

```
# Calculate and print the R-squared (coefficient of determination) for the
training data.
train_r_squared = ridgereg.score(X_train, y_train)
```

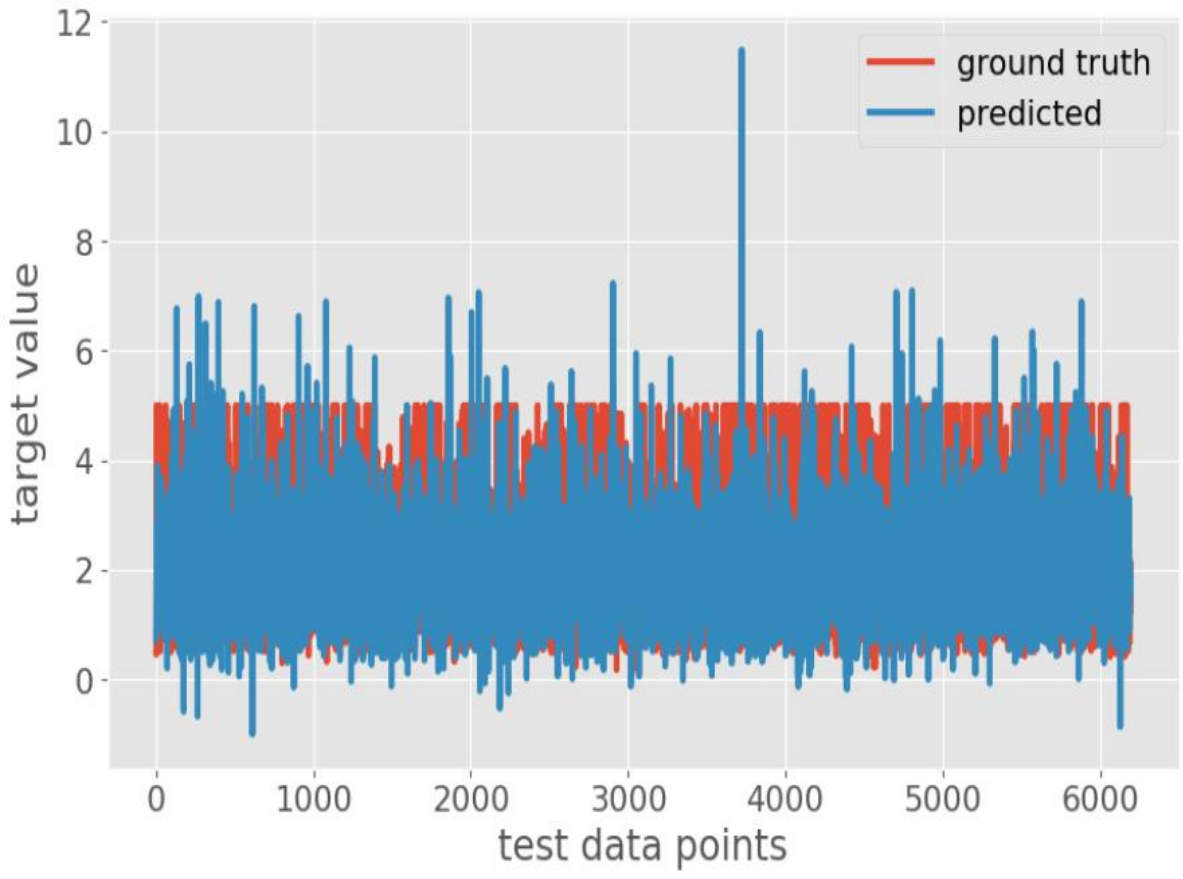
```
0.6093458881478931
```

```
# Make predictions on the test data using the trained Ridge Regression model.
y_pred = ridgereg.predict(X_test)

# Calculate the mean squared error for the predictions on the test data.
test_mse = metrics.mean_squared_error(y_test, y_pred)
```

```
0.5305052690933699
```

```
# Create a plot to compare ground truth and predicted values.
plt.figure(figsize=(10, 6))

# Plot ground truth and predicted values with labels and line widths.
plt.plot(y_test, linewidth=3, label='ground truth')
plt.plot(y_pred, linewidth=3, label='predicted')
```

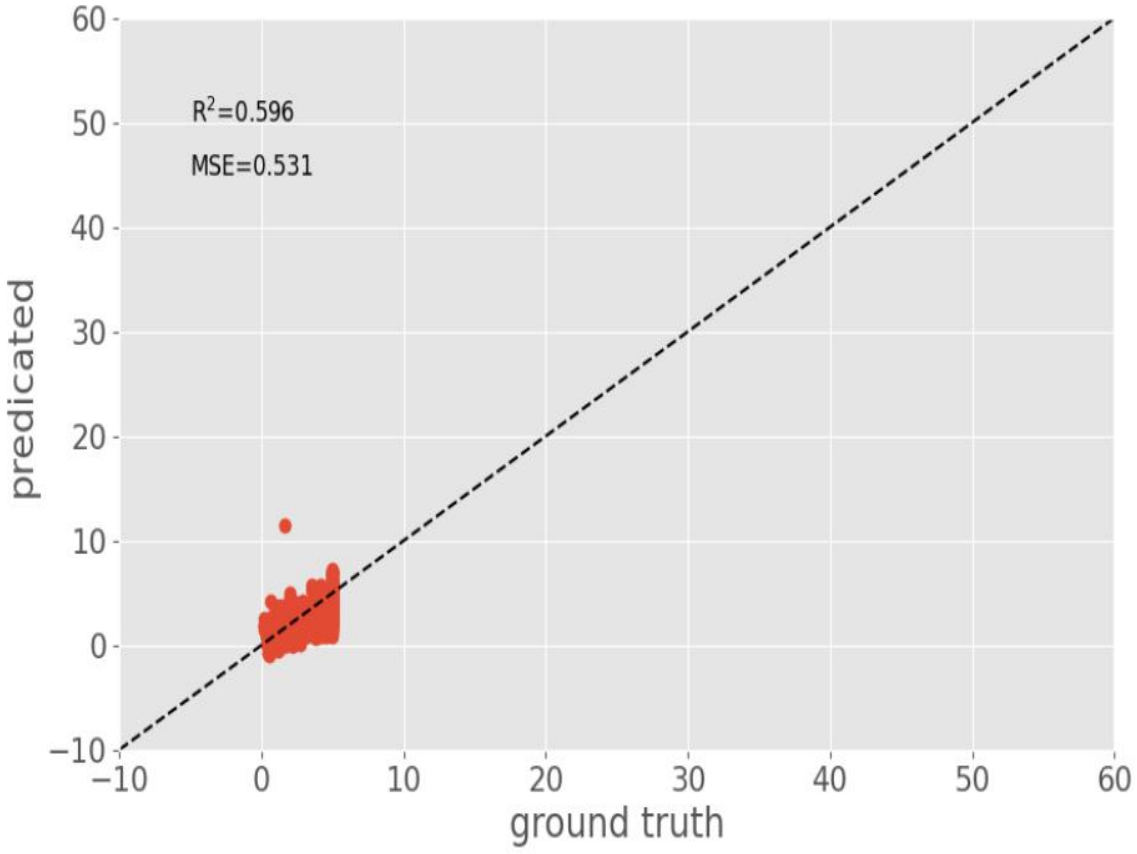
```
# Add a legend, xlabel, and ylabel for visualization.
plt.legend(loc='best')
plt.xlabel('test data points')
plt.ylabel('target value')
```




```
# Create a plot to compare ground truth and predicted values.
plt.figure(figsize=(10,6))
```

```
# Plot ground truth and predicted values with labels and line widths.
plt.plot(y_test,y_pred,'o')
plt.plot([-10,60],[-10,60],'k--')
plt.axis([-10,60,-10,60])
plt.xlabel('ground truth')
plt.ylabel('predicated')
```

```
# Add a legend, xlabel, and ylabel for visualization.
scorestr=r'R$^2$=%.3f' %ridgereg.score(X_test,y_test)
errstr='MSE=%.3f' %metrics.mean_squared_error(y_test,y_pred)
plt.text(-5,50,scorestr,fontsize=12)
plt.text(-5,45,errstr,fontsize=12)
```



LAB 06

Logistic Regression using the pre-defined library. Analysis of different training and testing splits ranges.

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt
```

```
# Define the file path or URL for the Iris dataset in CSV format.
url = "/content/iris.csv"


# Read the dataset into a Pandas DataFrame, specifying custom column names.
df = pd.read_csv(url, names=["sepal length", "sepal width", "petal length",
"petal width", "class"])
```

```
df
```

	sepal length	sepal width	petal length	petal width	class
0	sepal_length	sepal_width	petal_length	petal_width	species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
...	...	...	...	...	...
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

151 rows × 5 columns

```
# Import the StandardScaler for feature scaling.
from sklearn.preprocessing import StandardScaler


# Define feature column names and extract feature and target data from the
DataFrame.
features = ['sepal-length','sepal-width','petal-length','petal-width']
x=df.loc[:,features].values
y=df.loc[:,['target']].values
```

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

```
# Import PCA from scikit-learn and configure it for 2 components.
pca = PCA(n_components=2)


# Apply PCA to the standardized feature data to obtain principal components.
principalComponents = pca.fit_transform(x)
```

```
# Create a DataFrame to hold the principal components with specified column
names.

principalDataframe = pd.DataFrame(data=principalComponents, columns=['PC1',
'PC2'])
```

```
# Extract the target class data into a separate DataFrame.

targetDataframe=df[['target']]


# Combine the principal components DataFrame and the target class DataFrame
horizontally.

newDataframe=pd.concat([principalDataframe,targetDataframe],axis=1)
```

```
# newDataframe contains the combined data of principal components and target
class labels.

newDataframe
```

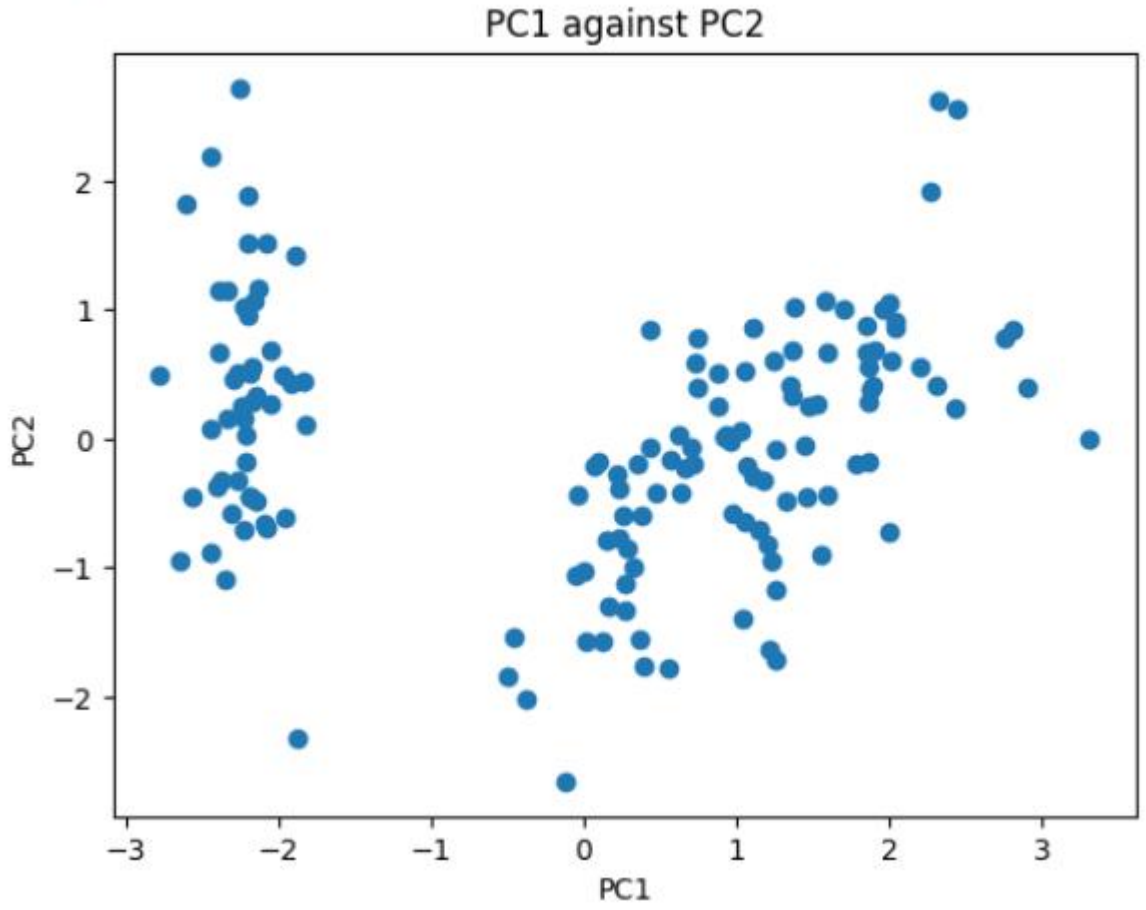
	PC1	PC2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa
...	...	...	...
145	1.870522	0.382822	Iris-virginica
146	1.558492	-0.905314	Iris-virginica
147	1.520845	0.266795	Iris-virginica
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)
```

```
# Set the plot title and axis labels for clear visualization.
plt.title('PC1 against PC2')
plt.xlabel('PC1')
plt.ylabel('PC2')
```

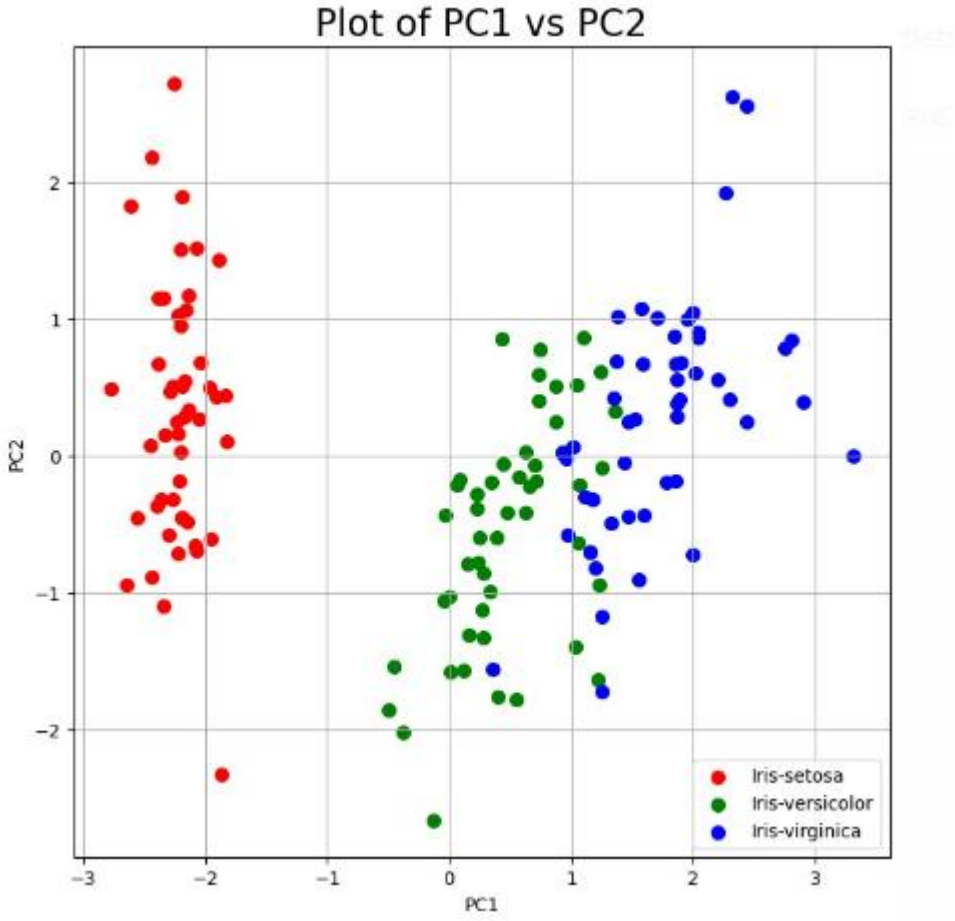
```
Text(0, 0.5, 'PC2')
```




```
# Create a scatter plot with labeled points, legend, and customized appearance.
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_title('Plot of PC1 vs PC2', fontsize=20)
```

```
# Define class labels and colors, then plot the data points with different
colors.

targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

colors = ['r', 'g', 'b']

for target, color in zip(targets, colors):

    # Select and scatter data points based on class label.

    indicesToKeep = newDataframe['class'] == target

    ax.scatter(newDataframe.loc[indicesToKeep, 'PC1'],

               newDataframe.loc[indicesToKeep, 'PC2'],

               c=color, s=50)
```

```
ax.legend(targets)  # Display the legend.

ax.grid()
```




```
pca.explained_variance_ratio_
```

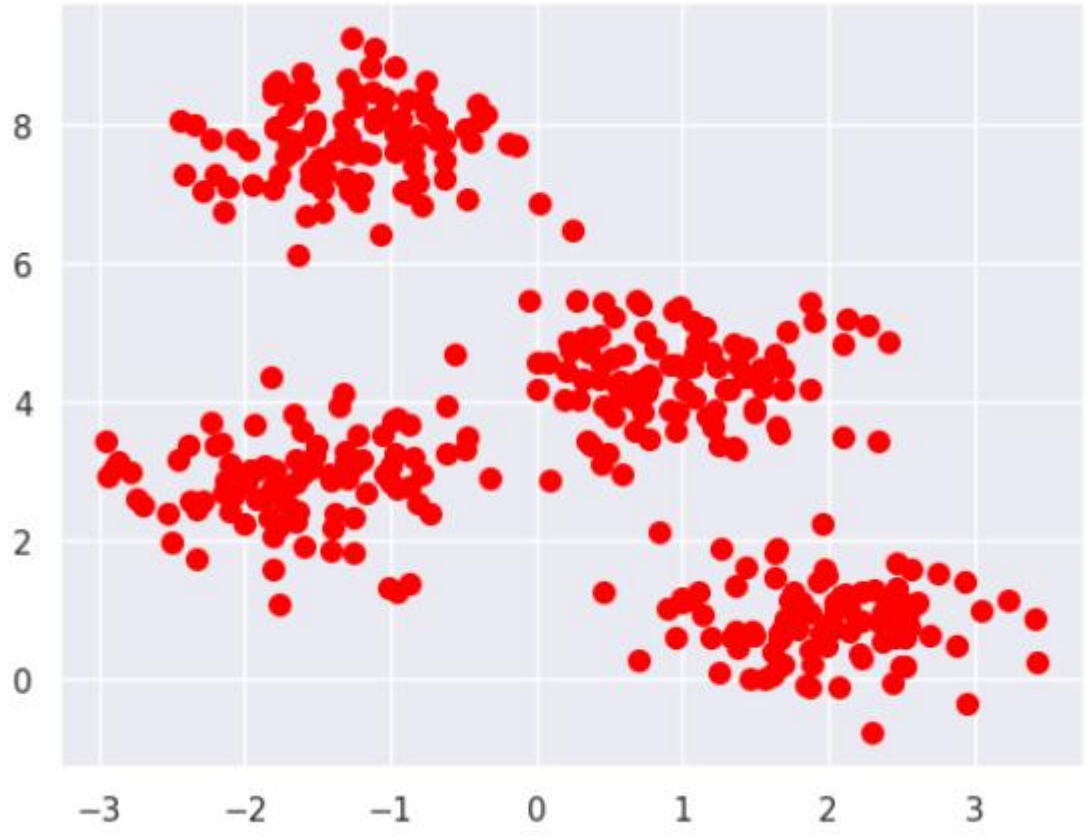
```
array([0.72770452, 0.23030523])
```

LAB 07

SVM and SVR for classification, regression

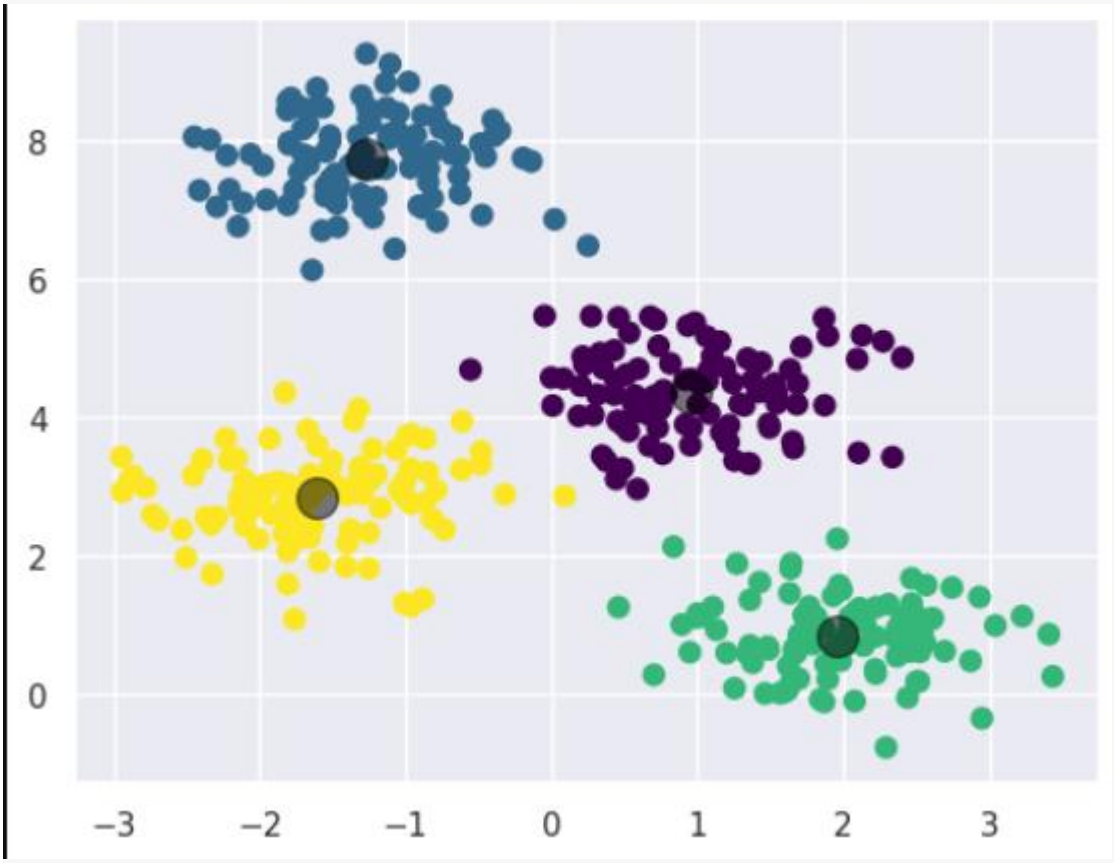
```
import matplotlib.pyplot as plt

import seaborn as sns

sns.set() #plot styling

import numpy as np
```

```
# Generate synthetic data with 4 clusters using make_blobs

from sklearn.datasets import make_blobs

X, y_true = make_blobs(n_samples=400, centers=4, cluster_std=0.60,
random_state=0)


# Scatter plot the data points in the original dataset

plt.scatter(X[:, 0], X[:, 1], s=50, color='red')
```



```
# Import KMeans clustering and fit the data
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=4,n_init=10)
kmeans.fit(X)
y_kmeans=kmeans.predict(X)
```

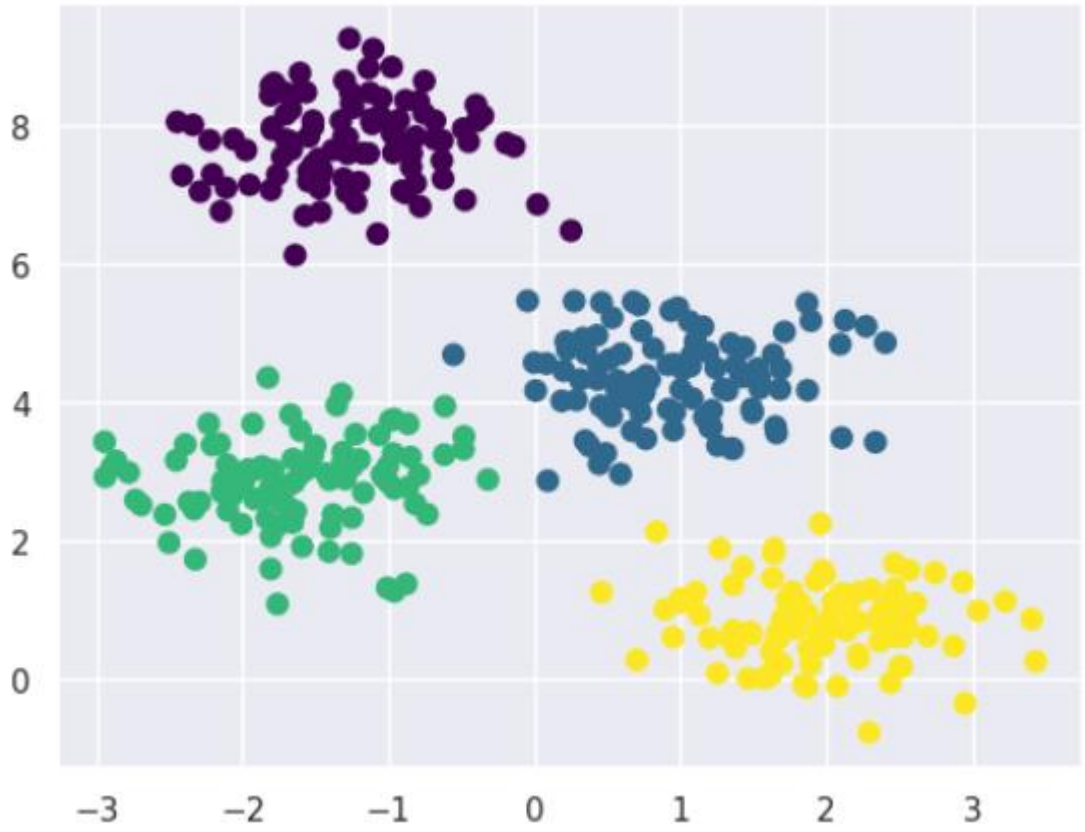
```
# Scatter plot the data points colored by their cluster assignment
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')


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



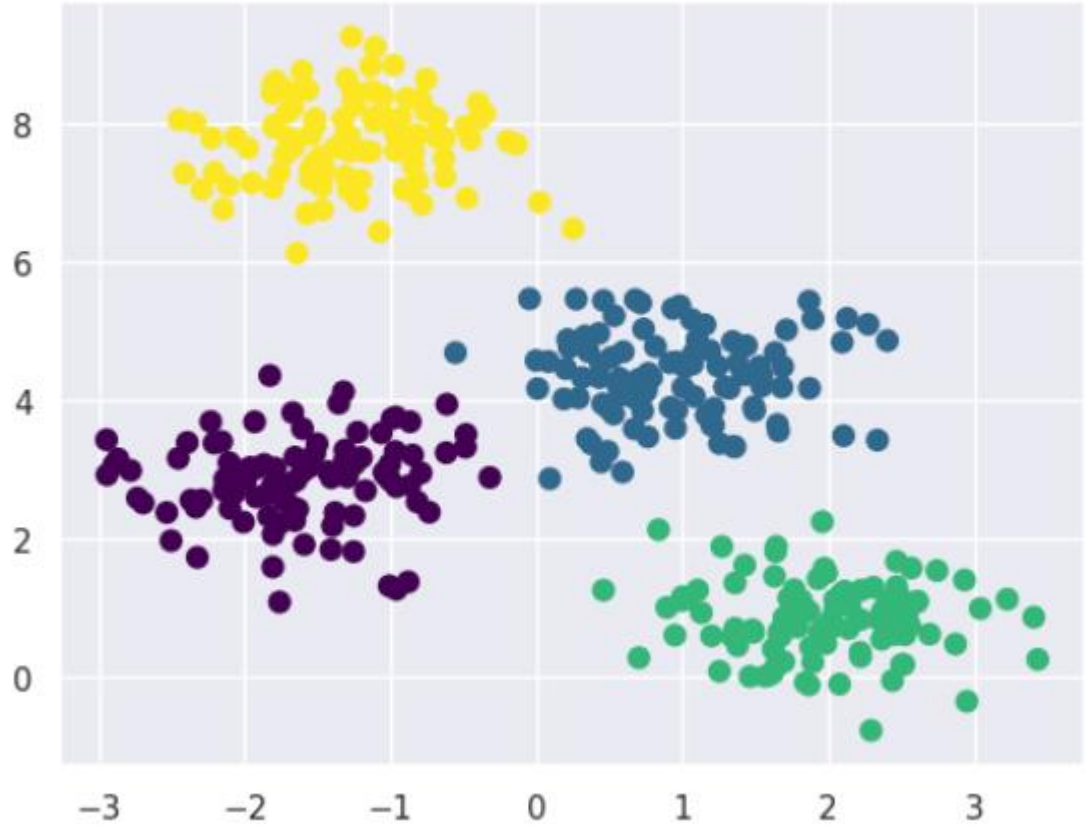

```
# Define a function to find clusters with custom implementation
from sklearn.metrics import pairwise_distances_argmin
def find_clusters(X, n_clusters, rseed=2):
```

```
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]
    while True:
        # 2a. Assign labels based on the 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


# Use the custom find_clusters function to find clusters
centers, labels = find_clusters(X, 4)
```

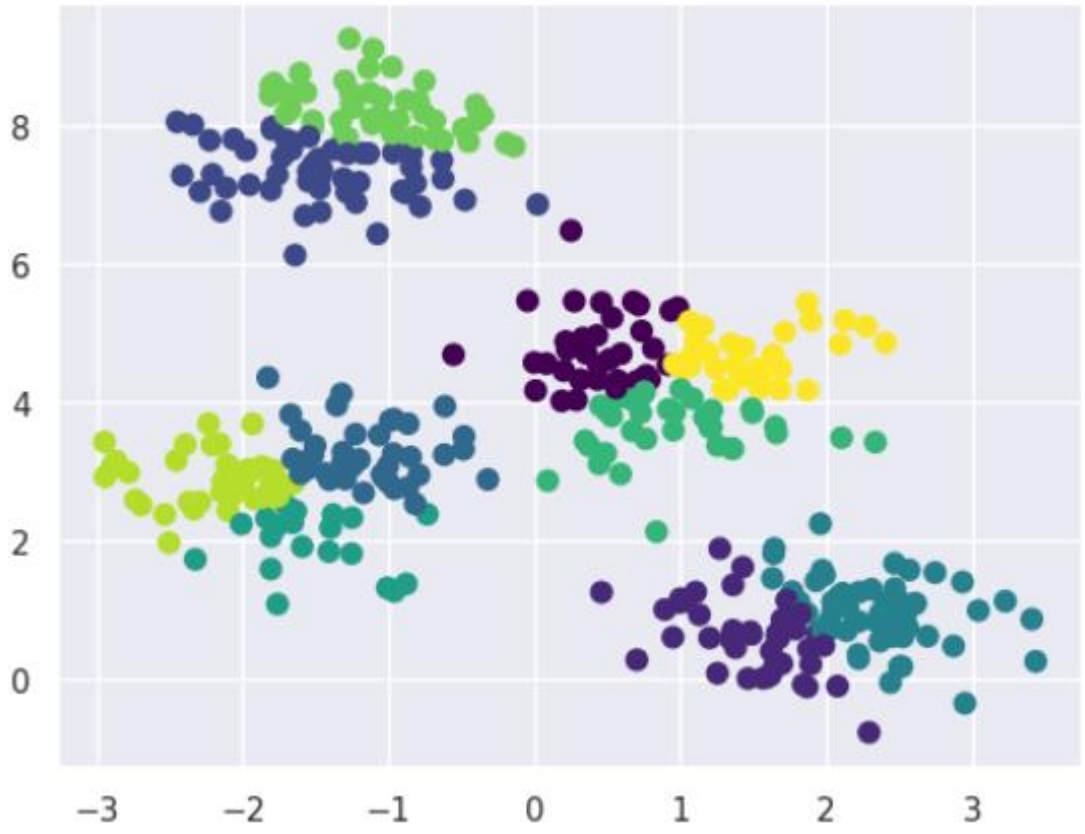
```
# Scatter plot the data points colored by their custom cluster assignment
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
```



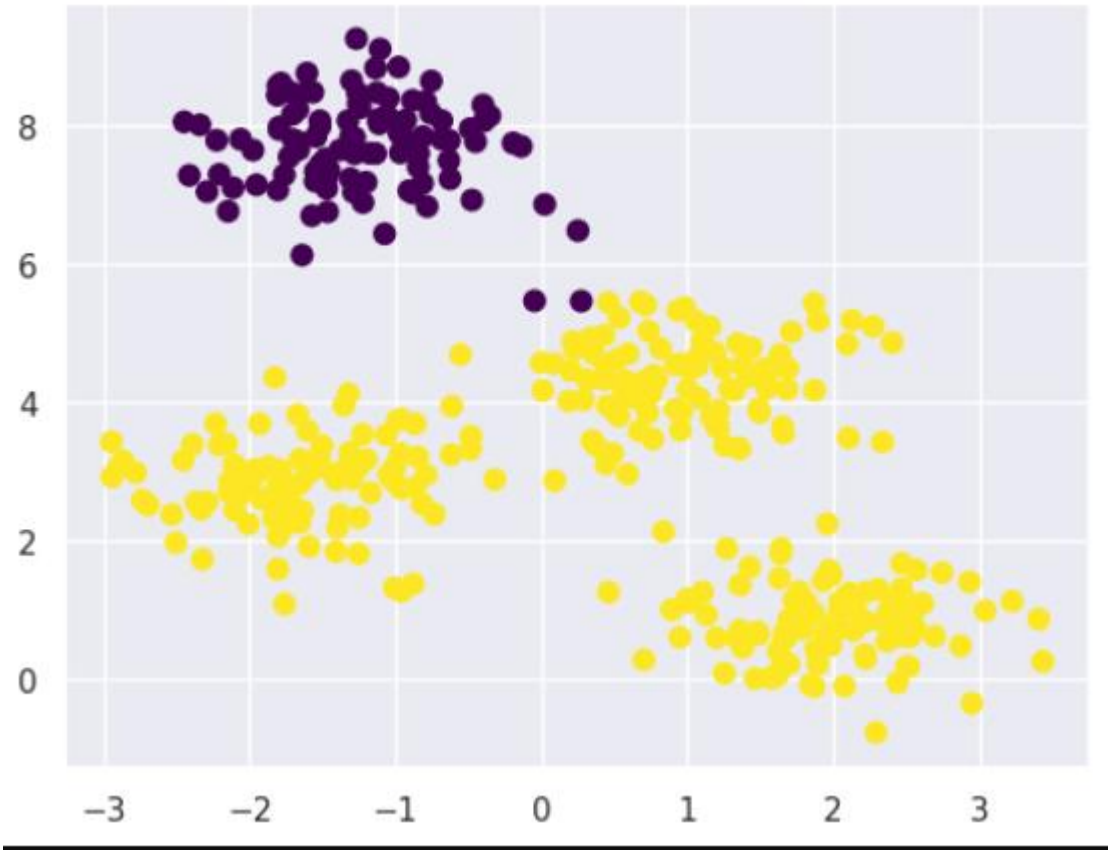

```
# Finding clusters using a custom function with 4 clusters and a specified
random seed.

centers, labels = find_clusters(X, 4, rseed=0)


# Scatter plot of the data points colored by cluster labels.

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



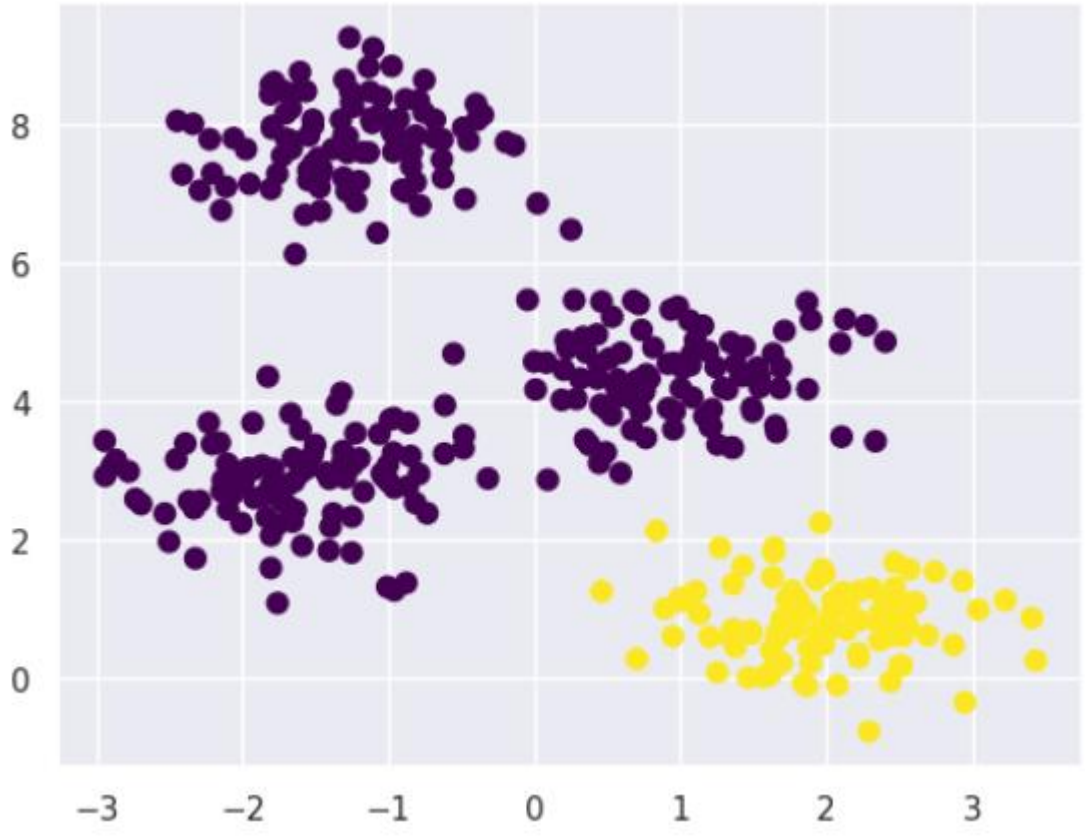

```
# Using scikit-learn's KMeans with 10 clusters and a specified random seed and multiple initialization attempts.

labels = KMeans(10, random_state=0, n_init=10).fit_predict(X)


# Scatter plot of the data points colored by cluster labels.

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




```
from sklearn.datasets import make_moons

# Generating synthetic data with two moon-shaped clusters (200 samples and some
noise) and a random seed.


X, y = make_moons(200, noise=0.05, random_state=0)

# Applying KMeans clustering to the moon-shaped data with 2 clusters and
specified random seed and initialization attempts.

labels = KMeans(2, random_state=0, n_init=10).fit_predict(X)

# Scatter plot of the data points colored by cluster labels.

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



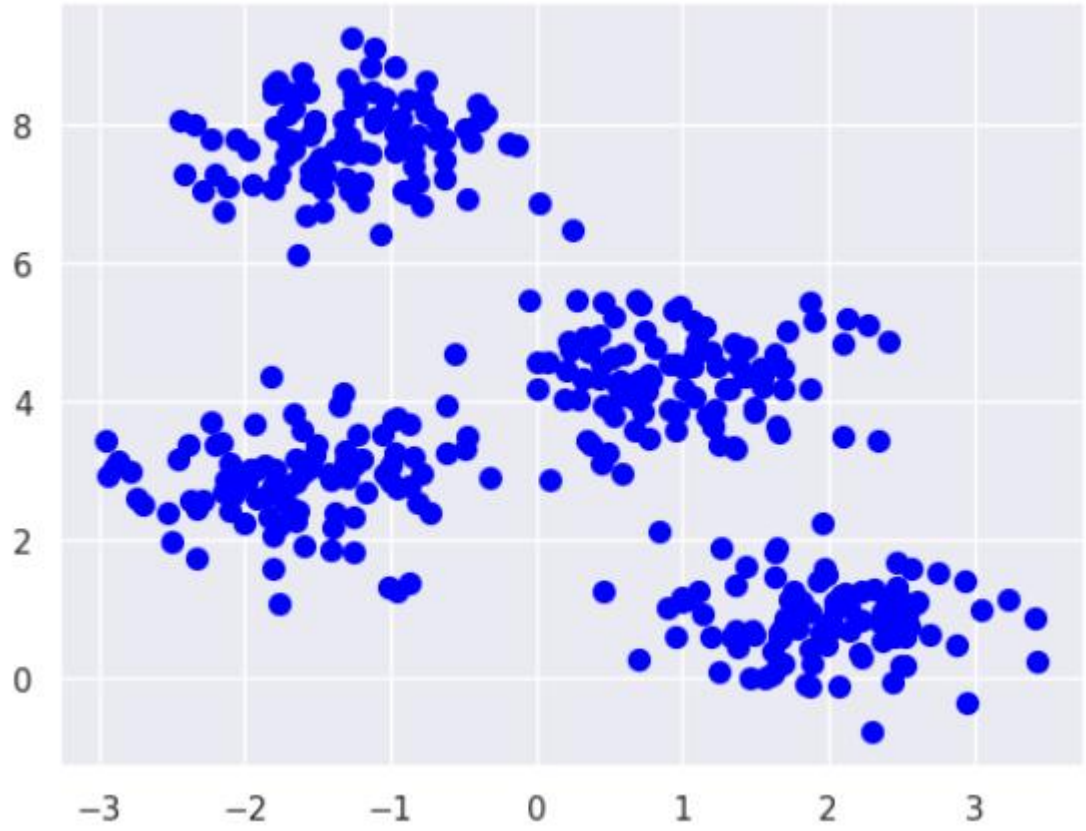

```
#kernel transformation

from sklearn.cluster import SpectralClustering


# Applying Spectral Clustering with 2 clusters using nearest_neighbors affinity
and KMeans for label assignment.

model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',
assign_labels='kmeans')


# Getting cluster labels for the data using Spectral Clustering.

labels = model.fit_predict(X)


# Scatter plot of the data points colored by cluster labels.

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


8
6
4
2
0
−3
−2
−1
0
1
2
3

LAB 08

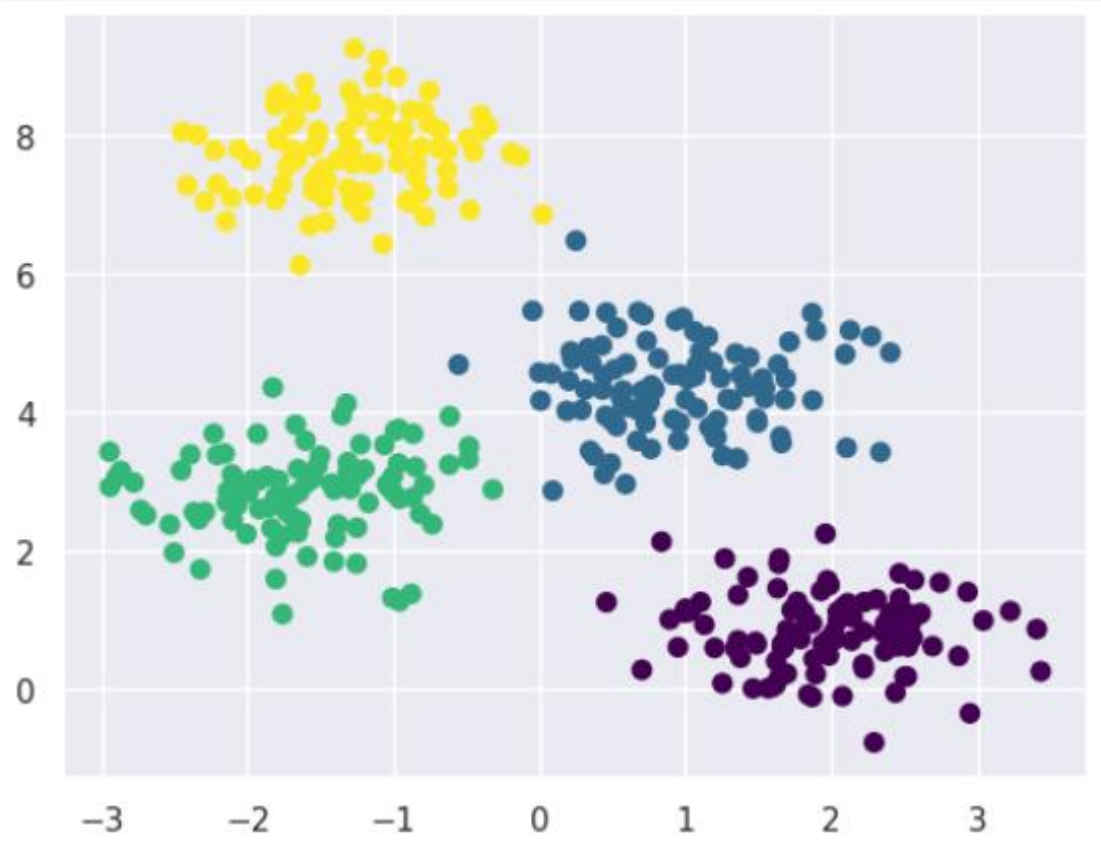
L2 regularization using the predefined library, comparing the results with ordinary regression.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set() #plot styling
import numpy as np
from sklearn.datasets import make_blobs
# Generate synthetic data with 4 clusters
X,y_true=make_blobs(n_samples=400,centers=4,cluster_std=0.60,random_state=0)
plt.scatter(X[:,0],X[:,1],s=50,color='blue');
```




```
from sklearn.mixture import GaussianMixture
# Fit a Gaussian Mixture Model with 4 components to the data
gmm = GaussianMixture(n_components=4).fit(X)
```

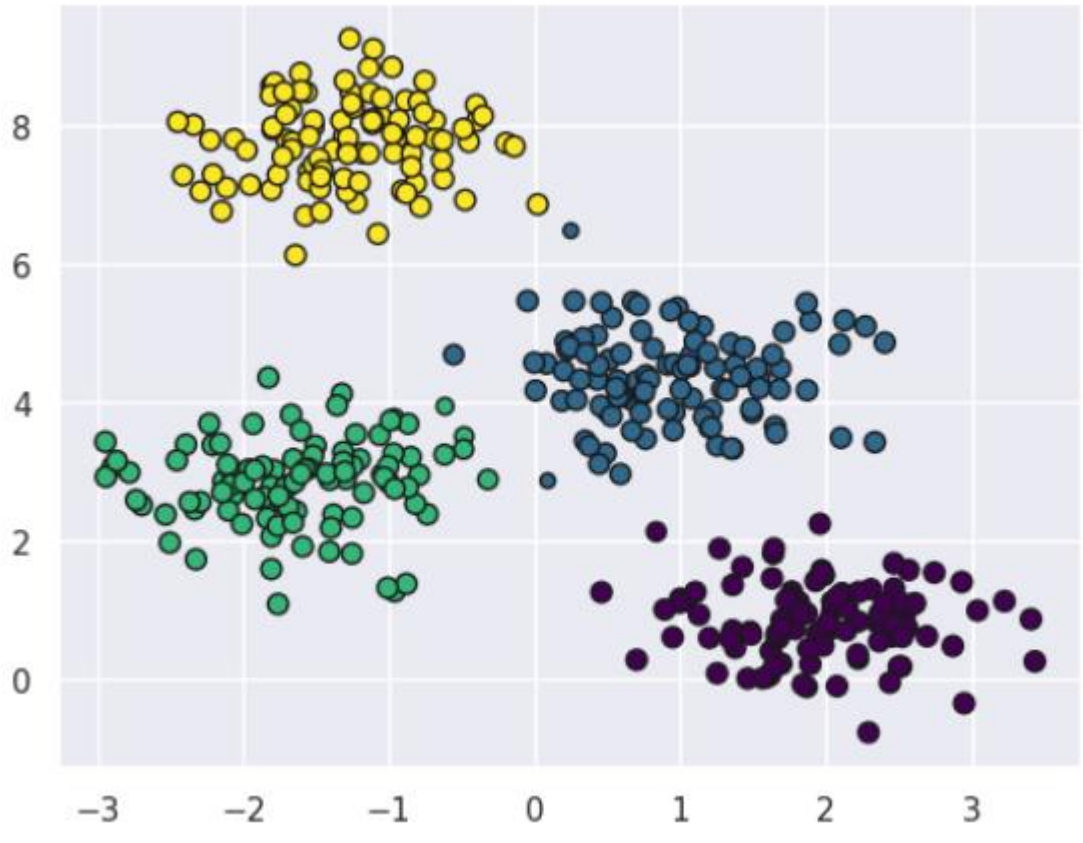
```
# Predict cluster labels using the GMM
labels = gmm.predict(X)plt.scatter(X[:,0],X[:,1],c=labels,s=40,cmap='viridis');
gmm.predict(X)plt.scatter(X[:,0],X[:,1],c=labels,s=40,cmap='viridis');
```



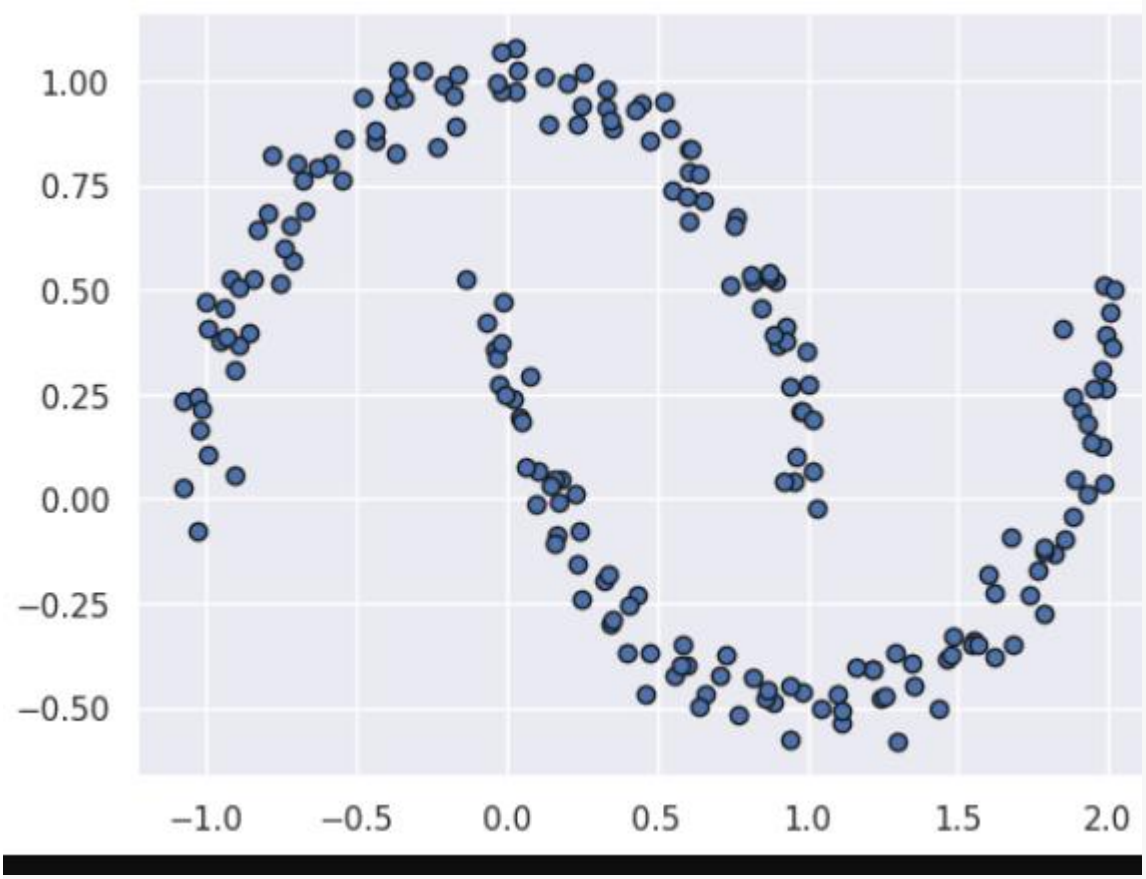

```
# Calculate the probability of each data point belonging to each cluster
probs = gmm.predict_proba(X)
Print(probs[:5].round(3))  # Display the probabilities for the first 5 data
points
```

```
[[0.    0.531 0.    0.469]
 [0.    0.    1.    0.   ]
 [0.    0.    1.    0.   ]
 [0.    1.    0.    0.   ]
 [0.    0.    1.    0.   ]]
```

```
# Determine the size of data points based on the maximum probability
size = probs.max(1) / 0.02
# Scatter plot with varying point sizes based on maximum probability
plt.scatter(X[:, 0], X[:, 1], c=labels, edgecolor='k', cmap='viridis', s=size);
```

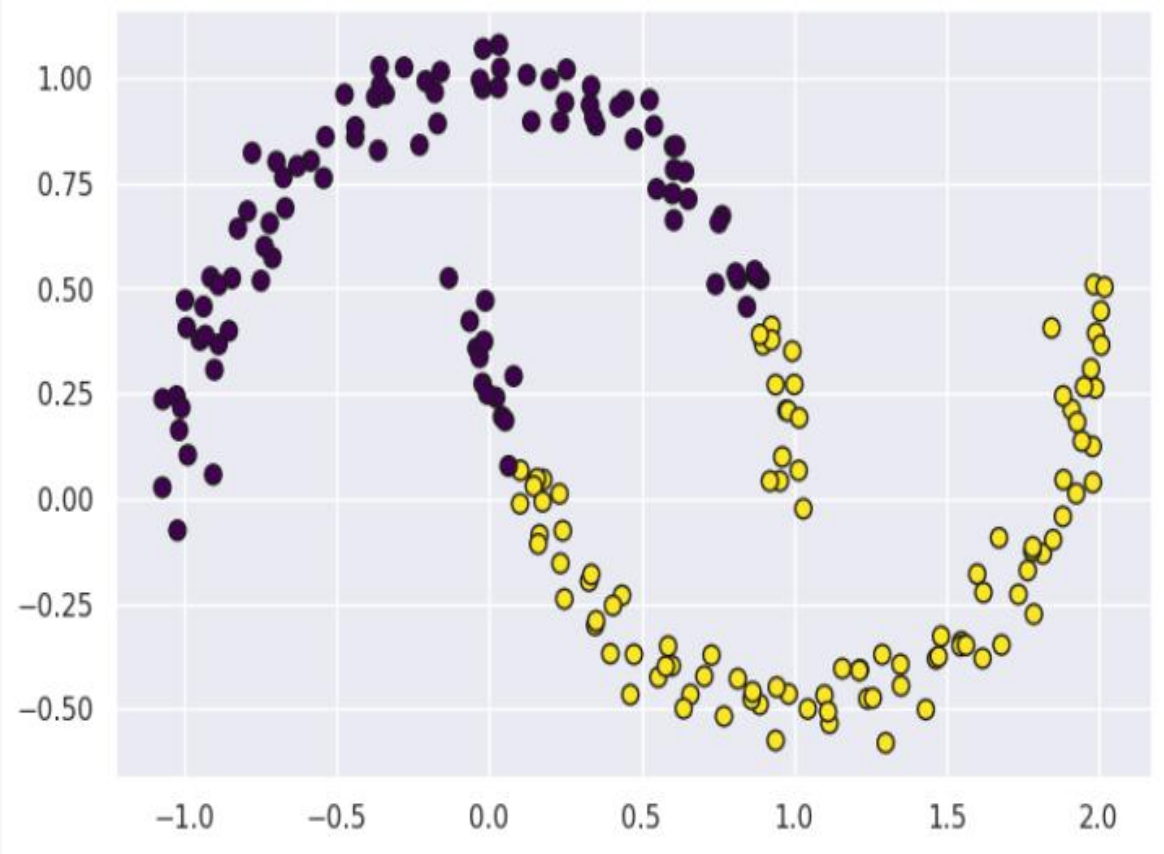



```
# Import necessary libraries
from sklearn.datasets import make_moons
# Generate moon-shaped data with 200 samples and noise
Xmoon, ymoon = make_moons(200, noise=0.05, random_state=0)
# Create a scatter plot of the moon-shaped data
plt.scatter(Xmoon[:, 0], Xmoon[:, 1], edgecolor='k')
```




```
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))
# Define the plot_gmm function to visualize the GMM comp
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=50,
cmap='viridis',zorder=2,edgecolor='k')
  else:
    ax.scatter(X[:, 0], X[:, 1], s=50, 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)
      #Create a GMM model and plot the moon-shaped data with GMM components
      gmm = GaussianMixture(n_components=4,
covariance_type='full',random_state=42)
      plt_gmm(gmm, X_stretched)
      plt.show()
#No.Components determine the gmm structure and its distribution
gmm2= GaussianMixture(n_components=2, covariance_type='full', random_state=0)
plt.figure(figsize=(8,5))
plot_gmm(gmm2,Xmoon)
```

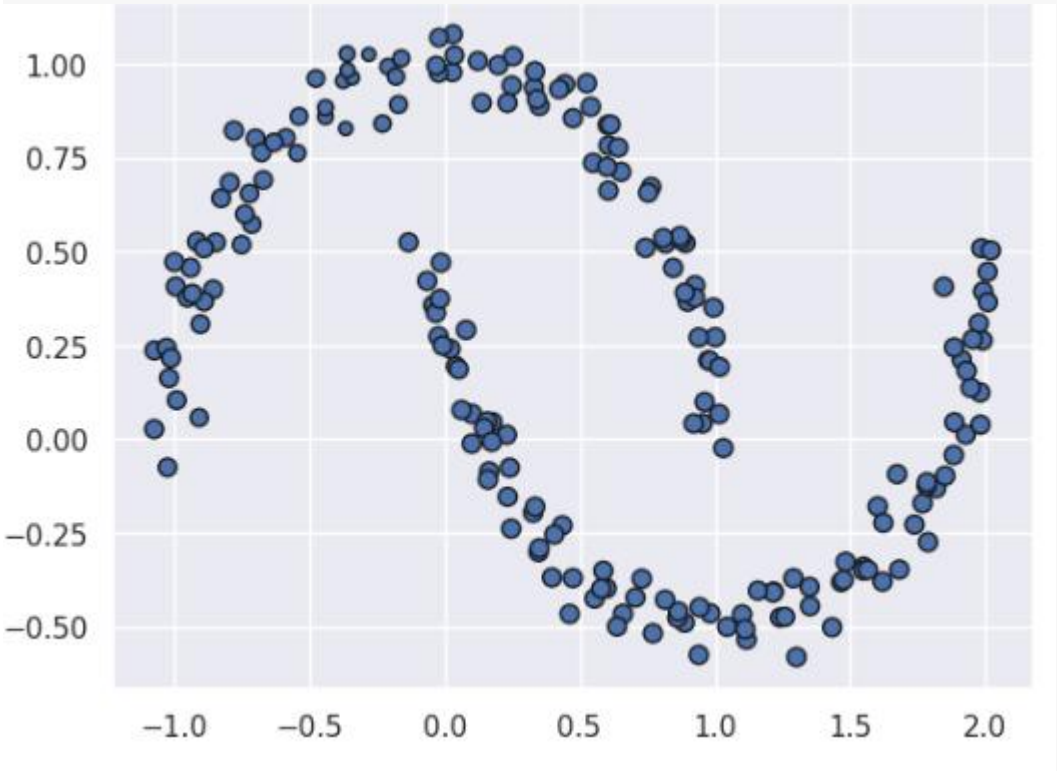



```
probs=gmm.predict_proba(Xmoon)
print(probs[:5].round(3))
```

```
[[1.    0.    0.    0.   ]
 [1.    0.    0.    0.   ]
 [0.998 0.    0.002 0.   ]
 [0.049 0.    0.951 0.   ]
 [1.    0.    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)
```

```
[0.99999919 1.         0.99847553 0.95140652 1.         1.
 0.99999997 1.         0.99999984 1.         0.54552709 0.99999166
 1.         1.         0.97923872 0.99999997 0.99990824 0.99999789
 0.99936379 0.98509956 1.         1.         0.99461071 1.
 0.52273631 0.99998189 1.         1.         0.99996053 0.99400082
 0.9999996  0.82521331 1.         0.99946609 0.99999689 1.
 0.99999889 0.99999998 0.97158503 0.99721432 0.99999987 0.68280808
 0.99998845 0.9998444  1.         0.63362387 0.99999998 0.94185779
 0.99996236 1.         0.96216131 0.99999583 1.         0.90074618
 1.         0.99999683 0.99996261 1.         0.9975625  0.73098645
 0.99999787 0.97964578 1.         0.99999951 1.         0.99163324
 0.80174744 1.         0.97231314 0.98591072 0.99999997 0.99033525
 0.99999971 0.98397376 0.99999999 0.98251749 0.99954222 0.99999985
 0.99997306 0.99999998 1.         0.95434473 0.99999996 1.
 0.99918899 0.68146158 0.99999998 0.55737573 0.99999991 1.
 0.99898395 0.99999994 1.         0.94373029 1.         1.
 1.         0.99999862 0.98667767 0.99975261 1.         1.
 0.95465338 0.99999999 0.96097131 0.99944434 0.96509227 0.83783539
 1.         0.99774562 0.8854553  0.96760871 0.92827439 0.95628794
 0.99999987 1.         0.99998582 1.         0.99999989 0.99999842
 0.99999946 0.90997105 0.99773847 0.94488289 0.99523012 0.97596719
 0.99999998 0.99993738 1.         1.         0.99999996 0.80211427
 0.99313483 0.99998606 1.         1.         1.         0.98230367
 1.         0.9999998  0.99988996 0.99649117 0.99966749 0.62671221
 0.70732431 0.99999999 0.99996972 1.         0.99649564 0.99884653
 0.99999961 1.         0.99999957 0.99922771 0.99999987 0.80886127
 1.         0.99999587 0.99985002 0.99998166 0.9999215  1.
 0.99996244 1.         0.99999909 1.         0.99984946 0.99999997
 1.         1.         0.99996595 0.98078672 0.98660027 0.92229824
 0.99991847 1.         1.         0.91722363 0.93783592 0.99997913
 1.         0.98098823 1.         1.         1.         0.99875887
 0.87195149 0.99999999 0.96493803 0.92100654 0.97967245 0.99961629
 0.99999905 0.99999994 0.99769784 0.85878433 1.         0.9999495
 0.98758841 1.        ]
<matplotlib.collections.PathCollection at 0x7d0fa9cff430>
```



LAB 09

L1 regularization using the predefined library, comparing the results with ordinary regression.

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

from sklearn.datasets import load_breast_cancer

# Load the breast cancer dataset
cancer=load_breast_cancer()

# Display the keys available in the dataset
cancer.keys()
```

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

```
# Print the dataset description
print(cancer['DESCR'])
```

```
: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
```

```
# Access the feature names in the dataset

cancer['feature_names']
```

```
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')
```

```
# Create a DataFrame using the dataset's data and feature names

df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])

# Display basic information about the DataFrame

df.info()
```

```
<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
dtypes: float64(30)
memory usage: 133.5 KB
```

```
# Display summary statistics of the DataFrame
df.describe()
```

	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	worst smoothness	worst compactness
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	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	0.132369	0.254265
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	0.022832	0.157336
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	0.071170	0.027290
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	0.116600	0.147200
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	0.131300	0.211900
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	0.146000	0.339100
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	0.222600	1.058000

8 rows × 30 columns

```
# Check for and sum the number of missing values in the DataFrame
np.sum(pd.isnull(df).sum())
```

```
0
```

```
# Sum the target values (indicating cancer or not)
cancer['target'].sum()
```

```
357
```

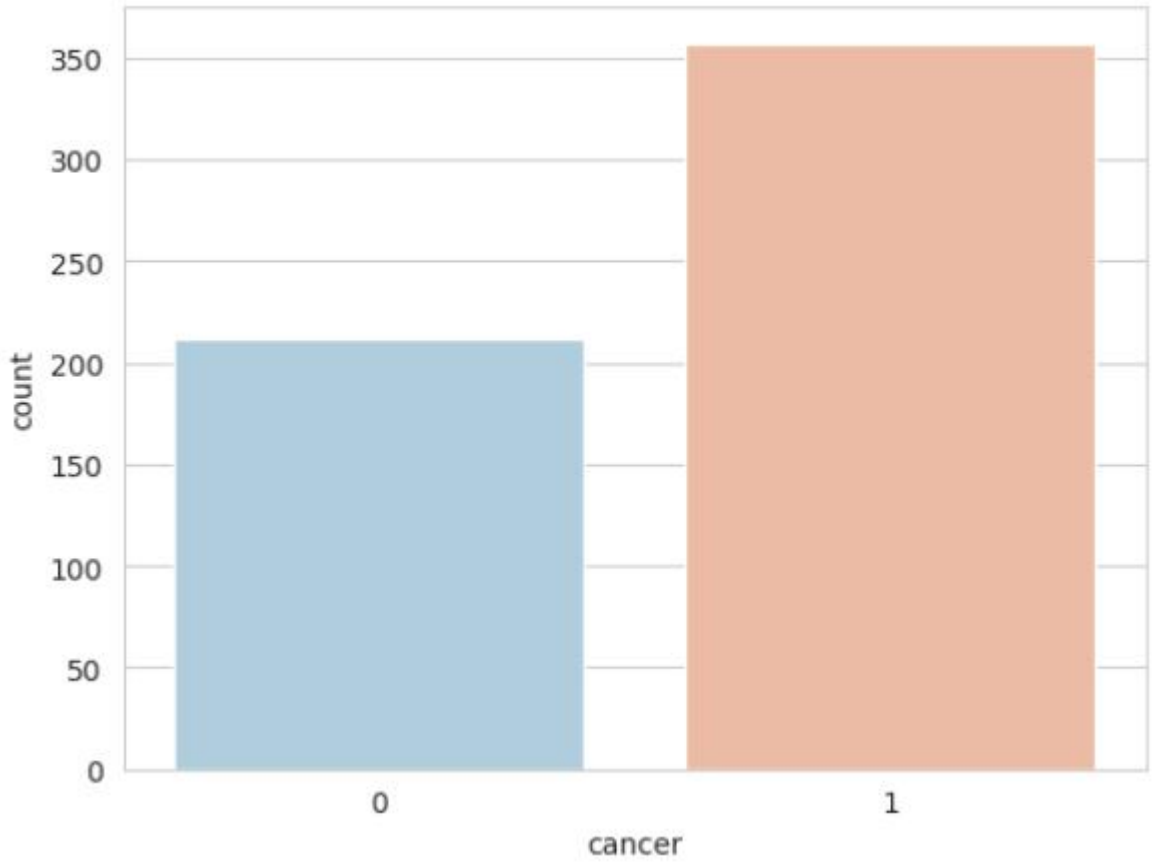
```
# Add the 'cancer' column to the DataFrame using the target values
df['cancer']=pd.DataFrame(cancer['target'])
# Display the first few rows of the DataFrame
df.head()
```

	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	worst smoothness	wor compactne
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	0.1622	0.66
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	0.1238	0.18
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	0.1444	0.42
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	0.2098	0.86
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	0.1374	0.20

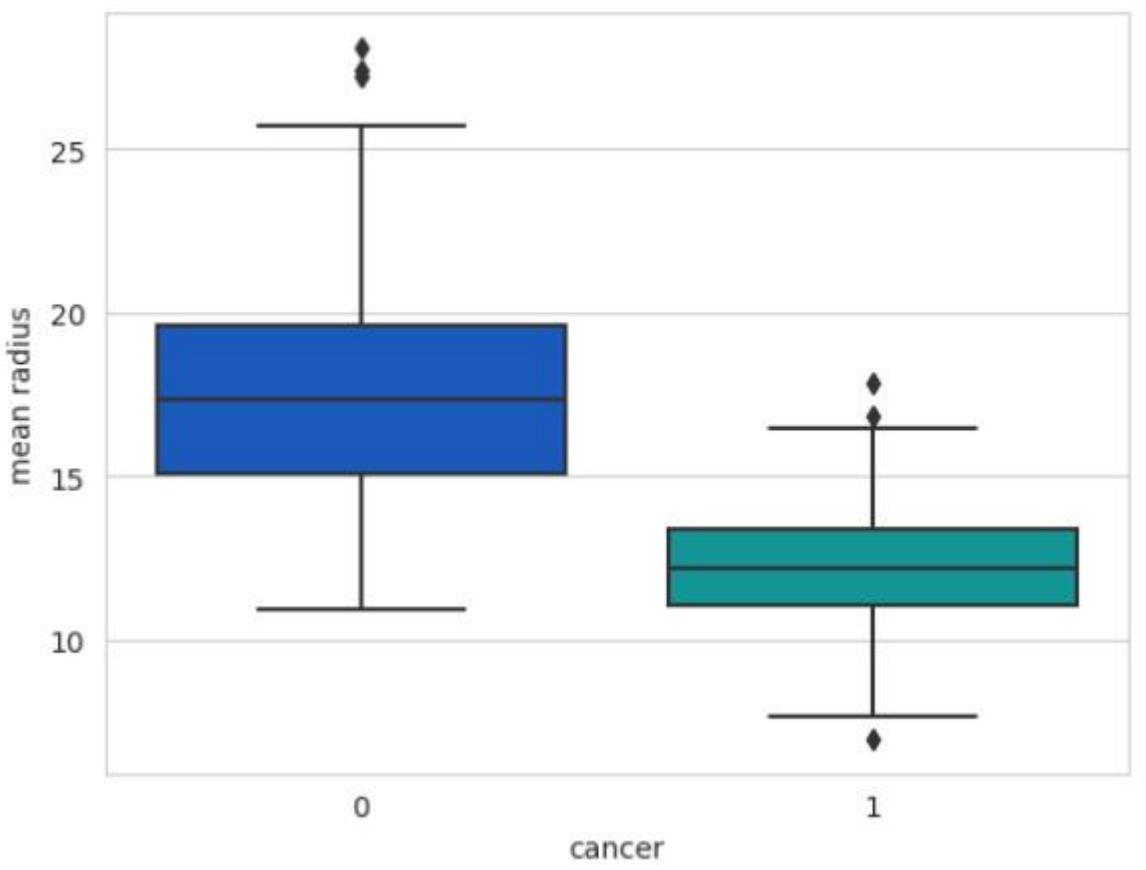
5 rows × 31 columns

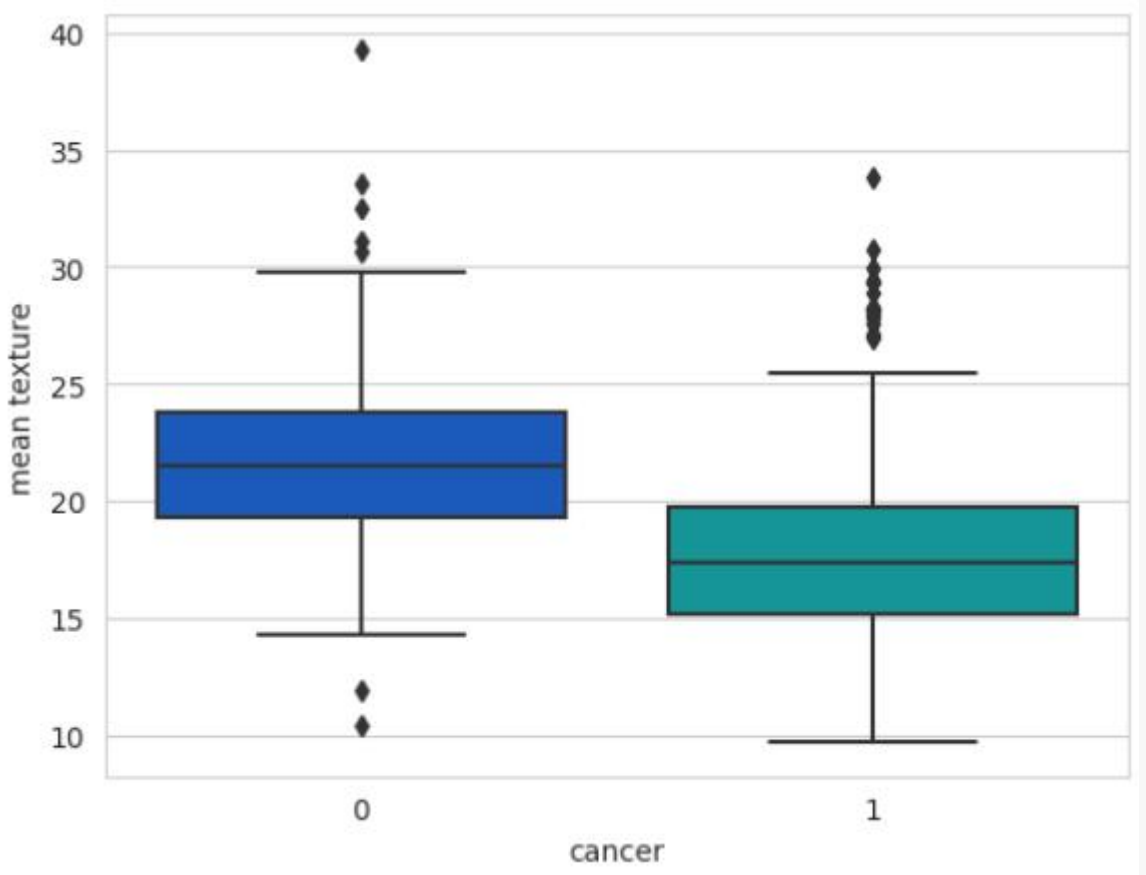
```
# Set the Seaborn style for plotting
sns.set_style('whitegrid')
# Create a count plot to visualize the distribution of cancer diagnosis
sns.countplot(x='cancer', data=df, palette='RdBu_r')
```

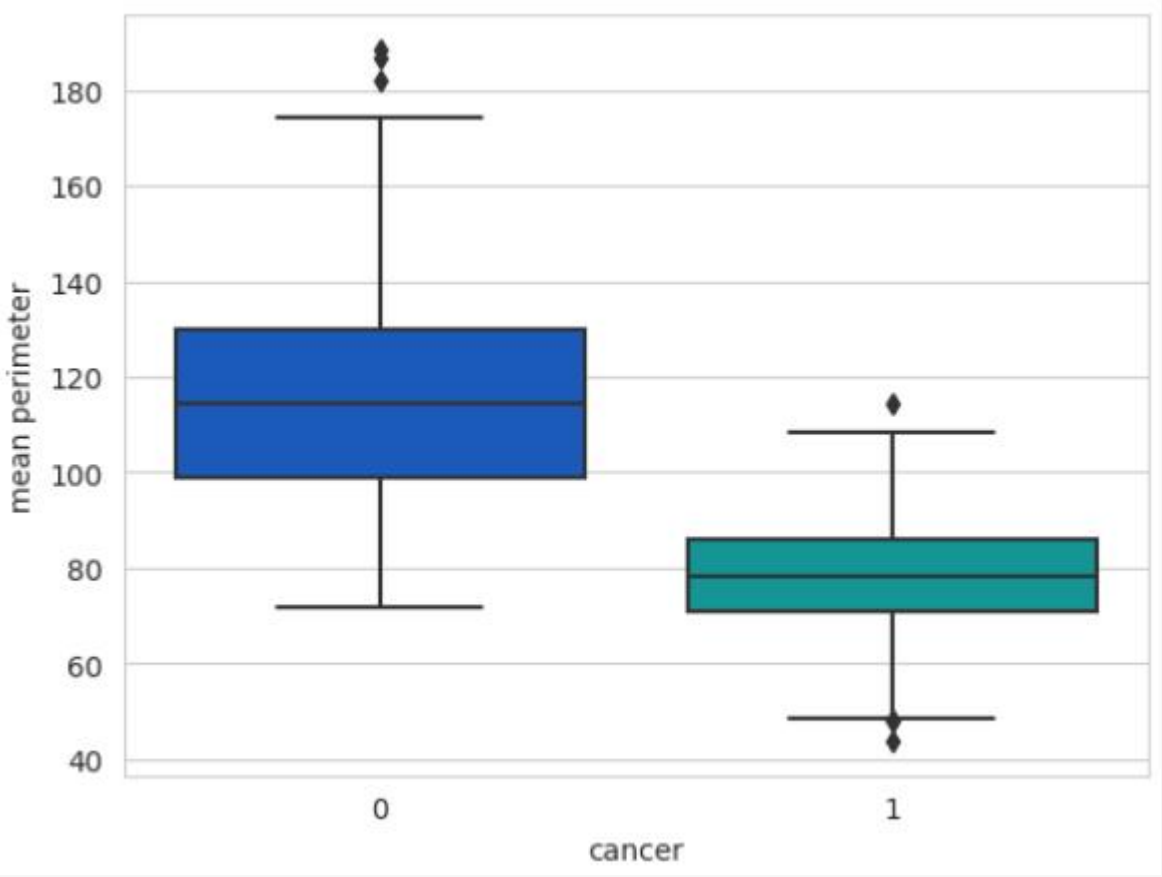
```
<Axes: xlabel='cancer', ylabel='count'>
```

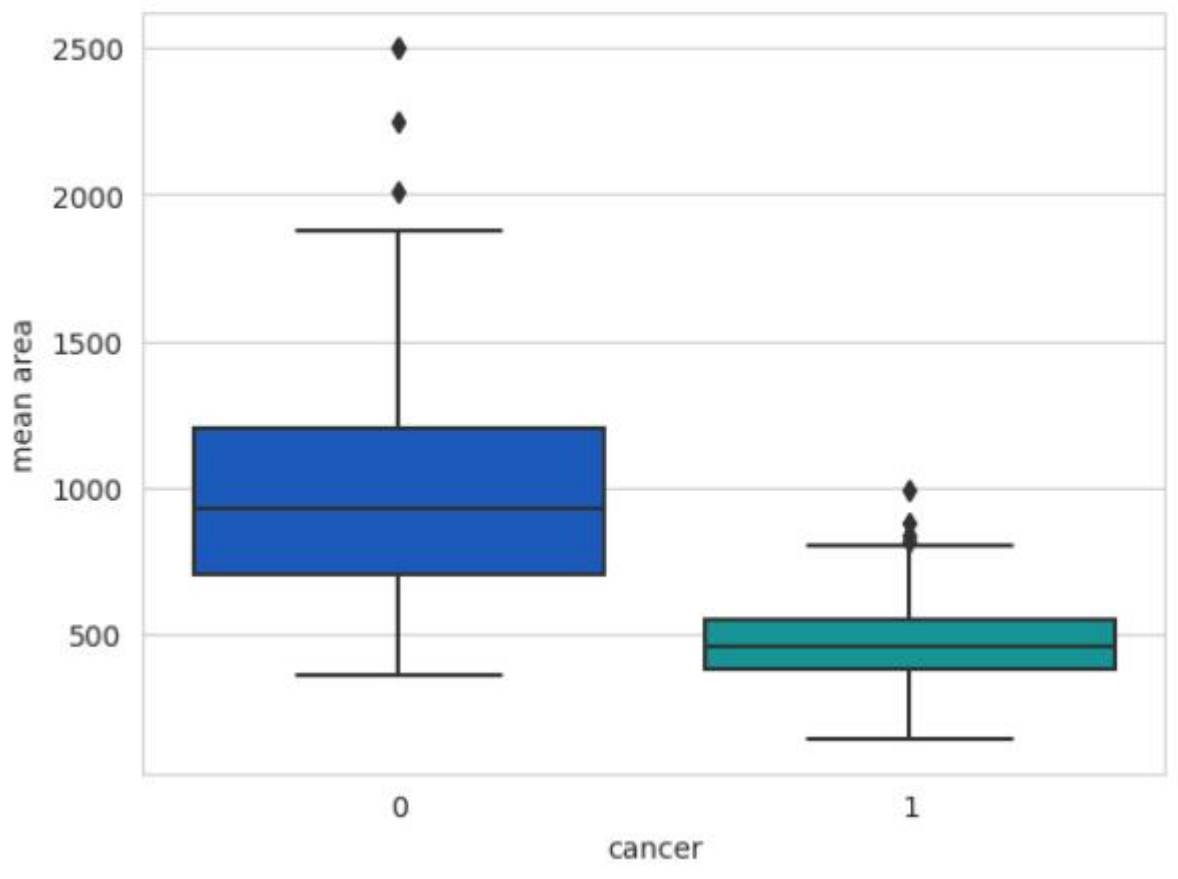


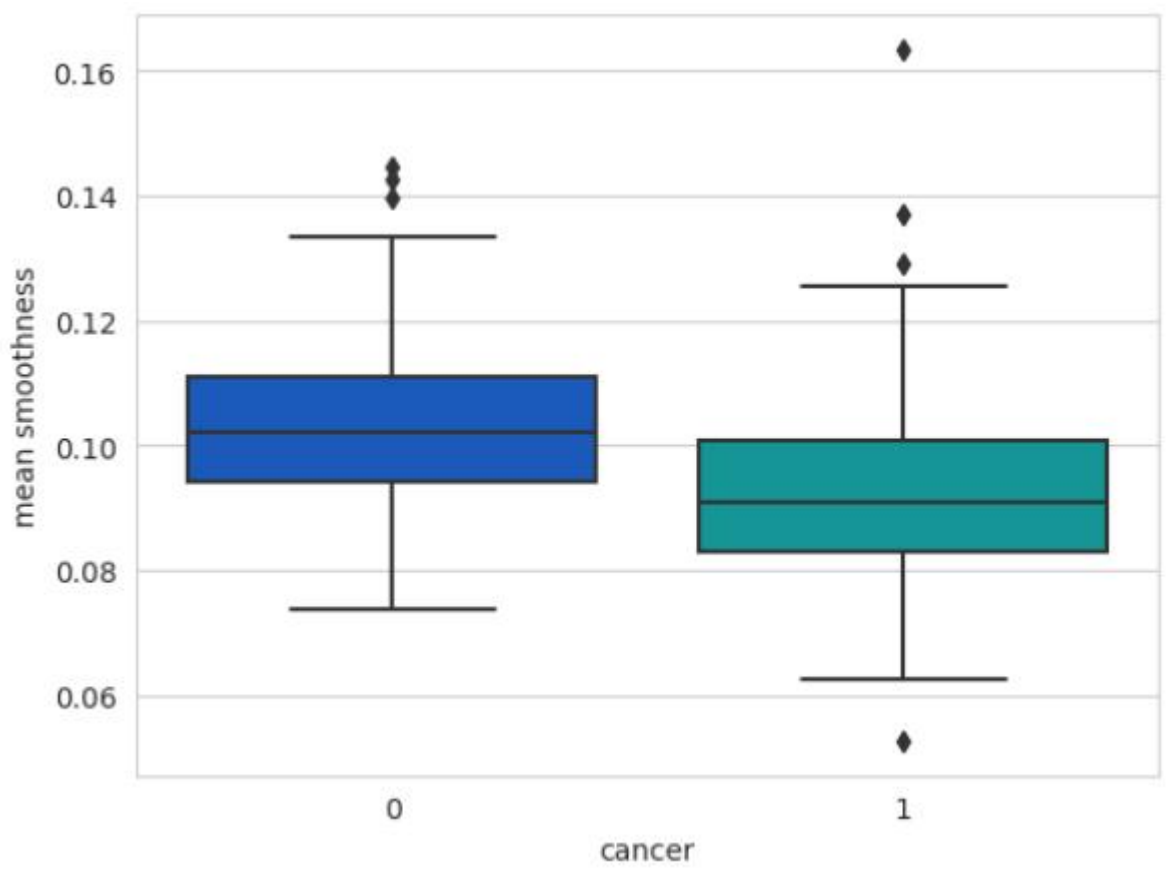

```
# Define a list of feature names for box plots
l = list(df.columns[0:10])
# Create box plots to compare feature distributions for malignant and benign cases
for i in range(len(l) - 1):
    sns.boxplot(x='cancer', y=l[i], data=df, palette='winter')
    plt.figure()
```

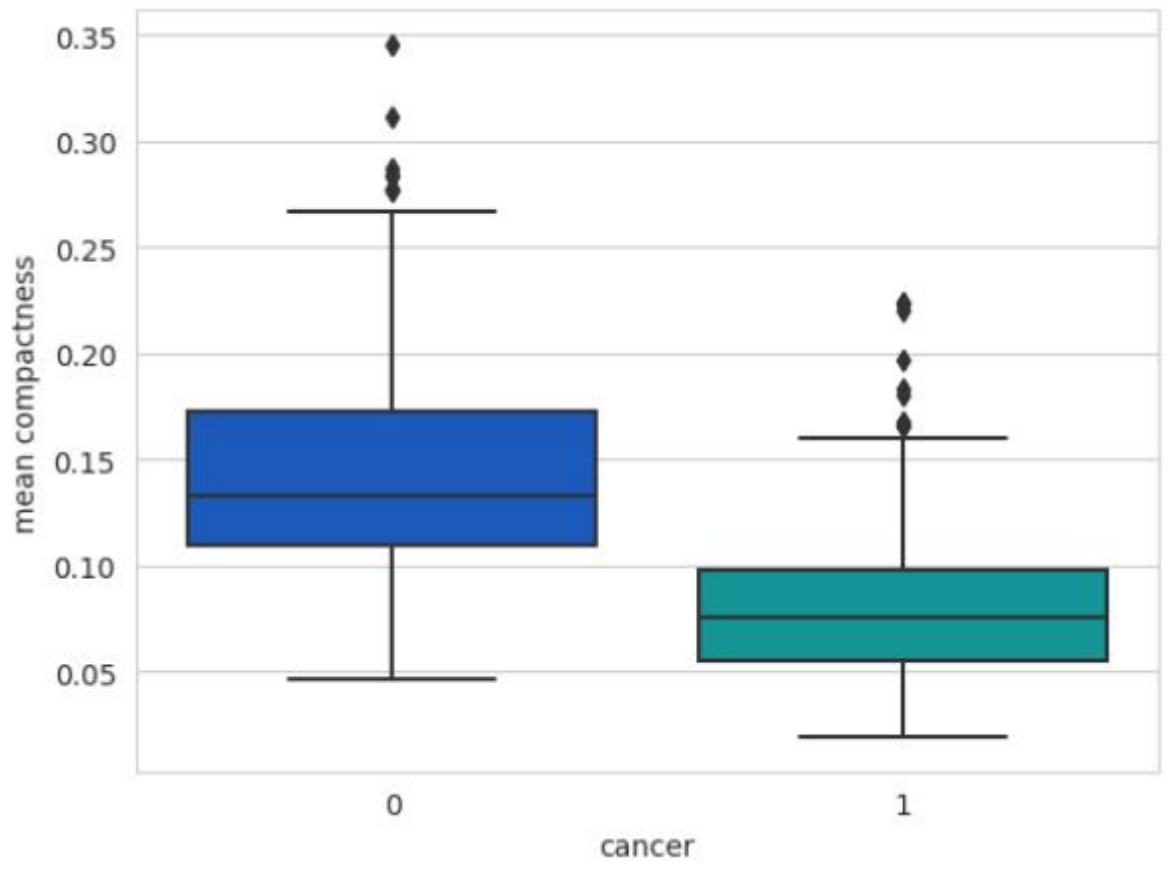

mean radius
25
20
15
10
0
1
cancer

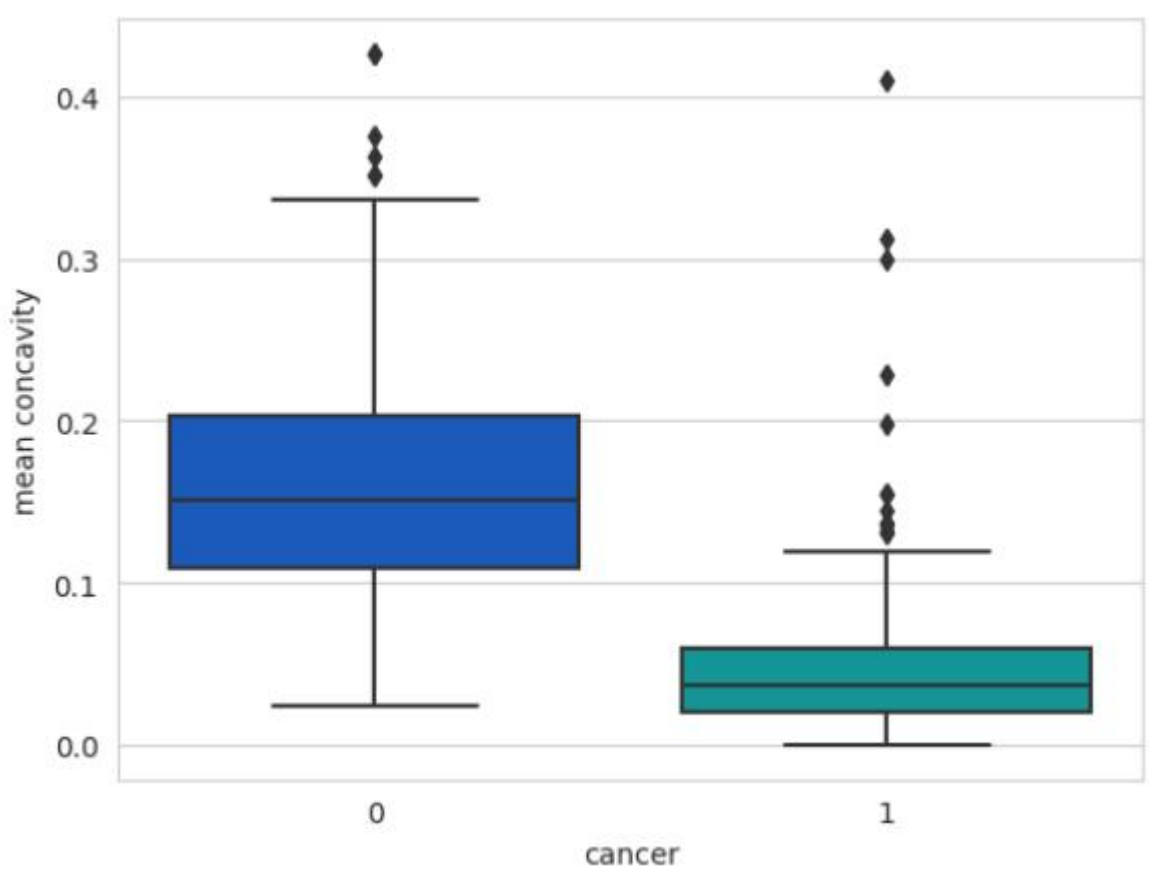

mean texture
40
35
30
25
20
15
10
0
1
cancer

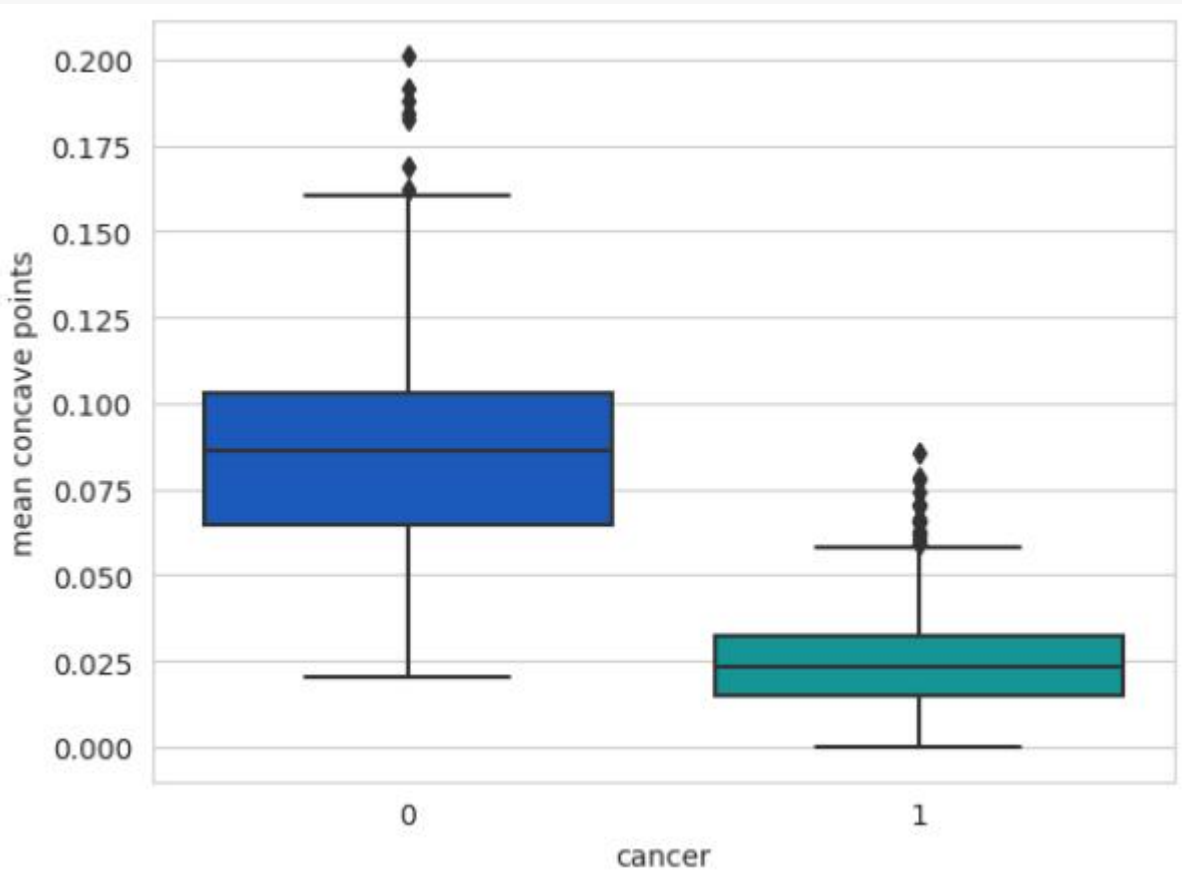

mean perimeter
180
160
140
120
100
80
60
40
0
1
cancer

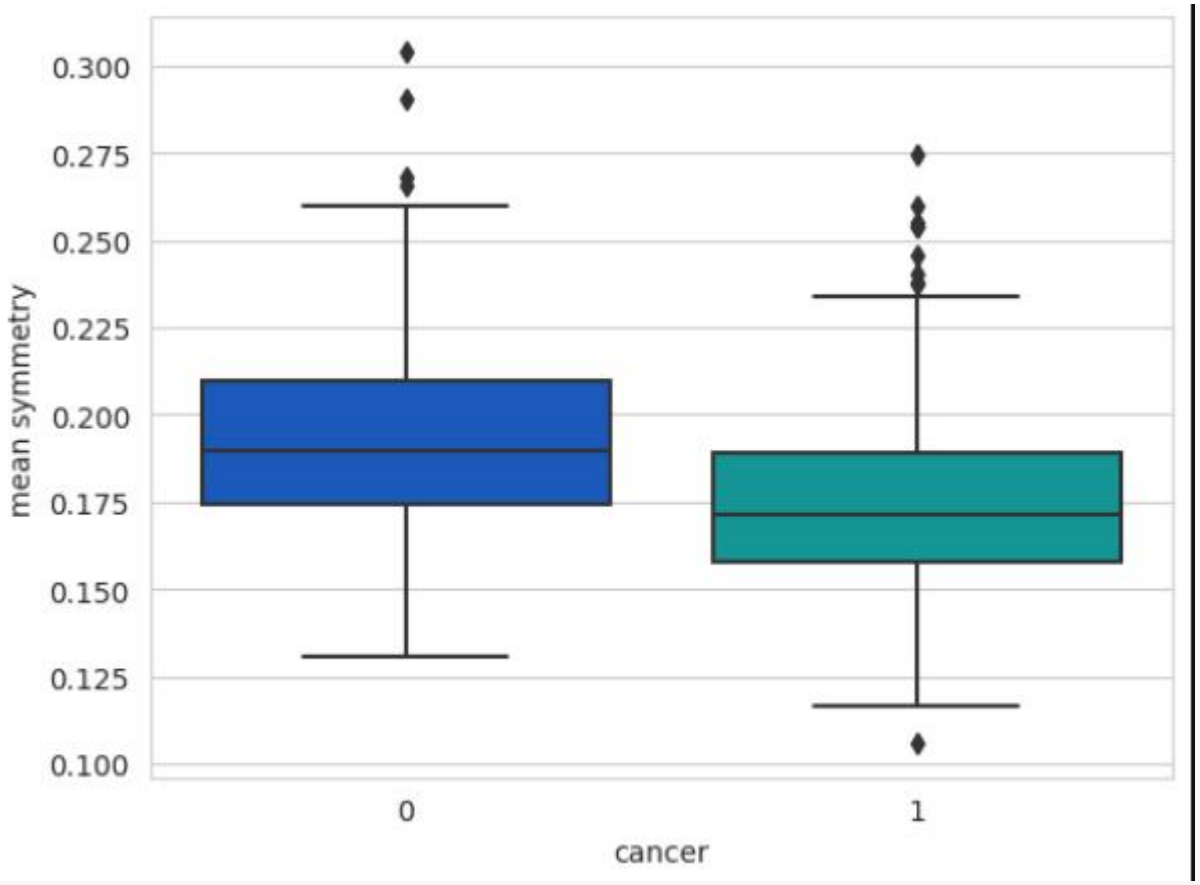

mean area
2500
2000
1500
1000
500
0
1
cancer


0.16
0.14
0.12
0.10
0.08
0.06
mean smoothness
0
1
cancer


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

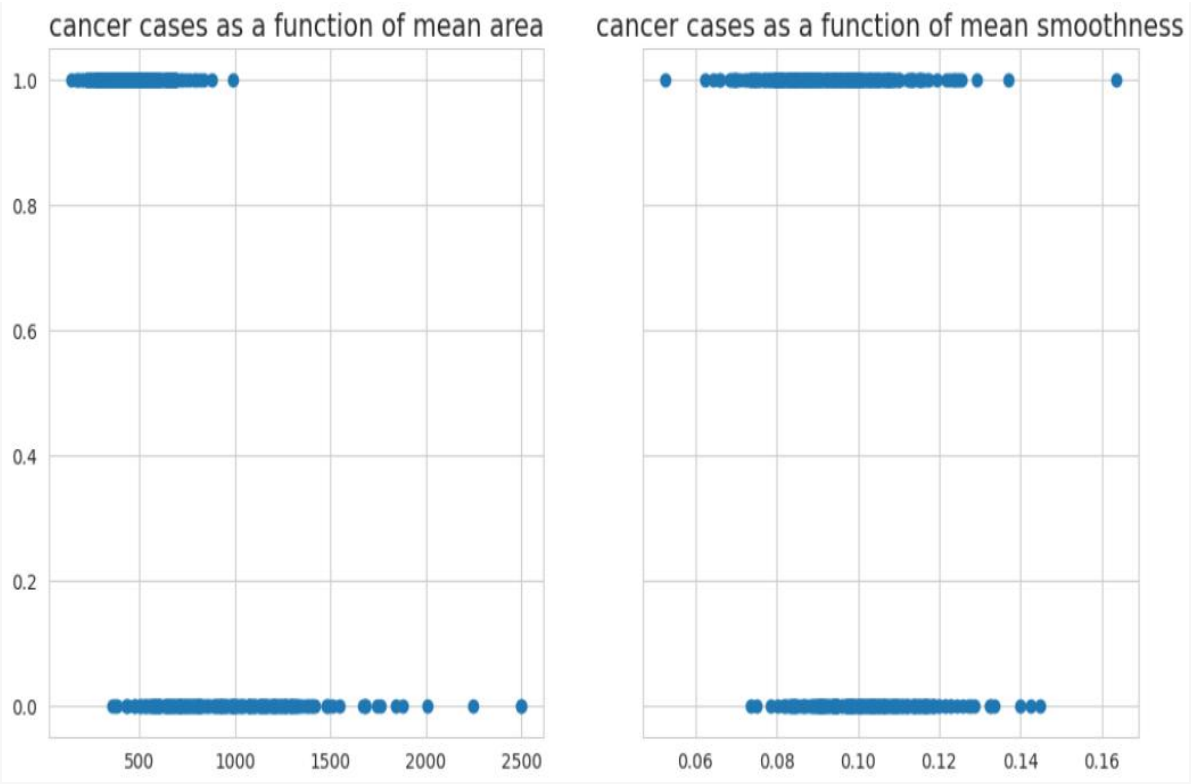

0.4
0.3
0.2
0.1
0.0
mean concavity
0
1
cancer


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




```
# Create a subplot with two side-by-side axes to visualize cancer cases as a function of different features
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(12, 6))


# Scatter plot of 'mean area' against 'cancer' in the first subplot
ax1.scatter(df['mean area'], df['cancer'])
ax1.set_title("Cancer cases as a function of mean area", fontsize=15)
```

```
# Scatter plot of 'mean smoothness' against 'cancer' in the second subplot
ax2.scatter(df['mean smoothness'], df['cancer'])
ax2.set_title("Cancer cases as a function of mean smoothness", fontsize=15)
```




```
# Extract the features (independent variables) by dropping the 'cancer' column
df_feat = df.drop('cancer', axis=1)
# Display the first few rows of the feature dataframe
df_feat.head()
```

	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	worst smoothness
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	184.60	2019.0	0.1622
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	158.80	1956.0	0.1238
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	152.50	1709.0	0.1444
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	98.87	567.7	0.2098
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	152.20	1575.0	0.1374

5 rows × 30 columns

```
# Extract the target variable ('cancer')
df_target = df['cancer']
# Display the first few rows of the target variable
df_target.head()
```

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

```
from sklearn.model_selection import train_test_split


# Split the data into training and testing sets

# X_train: Features for training, y_train: Target variable for training

# X_test: Features for testing, y_test: Target variable for testing

X_train, X_test, y_train, y_test = train_test_split(df_feat, df_target,
test_size=0.30, random_state=101)


# Display the first few rows of the training target variable

y_train.head()
```

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

```
from sklearn.svm import SVC

# Create a Support Vector Machine (SVM) model

model=SVC()

# Fit the SVM model on the training data

model.fit(X_train,y_train)
```

```
▾ SVC
SVC()
```

```
# Make predictions on the test data

predictions = model.predict(X_test)

# Import necessary metrics for model evaluation
```

```
from sklearn.metrics import classification_report,confusion_matrix

# Print the confusion matrix for the model's predictions

print(confusion_matrix(y_test,predictions))
```

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

```
# Print a classification report for the model's predictions

print(classification_report(y_test,predictions))
```

```
              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 parameter grid for hyperparameter tuning using GridSearchCV

param_grid =
{'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.0001],'kernel':['rbf']}

# Import GridSearchCV for hyperparameter tuning

from sklearn.model_selection import GridSearchCV

# Create a GridSearchCV object for the SVM model with the defined parameter
grid

grid=GridSearchCV(SVC(),param_grid,refit=True,verbose=1)

# Fit the GridSearchCV object to the training data to find the best
hyperparameters

grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

```
# Output the best hyperparameters found by the grid search

grid.best_params_
```

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

```
# Output the best estimator (machine learning model) found by the grid search
grid.best_estimator_
```

```
SVC
SVC(C=1, gamma=0.0001)
```

```
# Make predictions on the test data using the best estimator
grid_predictions=grid.predict(X_test)
# Output the confusion matrix to evaluate the model's performance
print(confusion_matrix(y_test,grid_predictions))
```

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

```
# Output a classification report providing various evaluation metrics
print(classification_report(y_test,grid_predictions))
```

	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

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

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pyplot as plt
```

```
# Generate some sample data
np.random.seed(0)
X = np.random.rand(100, 2)  # Feature matrix with 100 data points and 2
features
y_classification = (X[:, 0] + X[:, 1] > 1).astype(int)  # Binary classification
y_regression = X[:, 0] + X[:, 1]  # Regression
```

```
# Split the data into training and testing sets
X_train, X_test, y_train_classification, y_test_classification,
y_train_regression, y_test_regression = train_test_split(
    X, y_classification, y_regression, test_size=0.2, random_state=42
    )
# Analyze different numbers of neighbors (k) for classification
k_values = [1, 3, 5, 7]
classification_scores = []
```

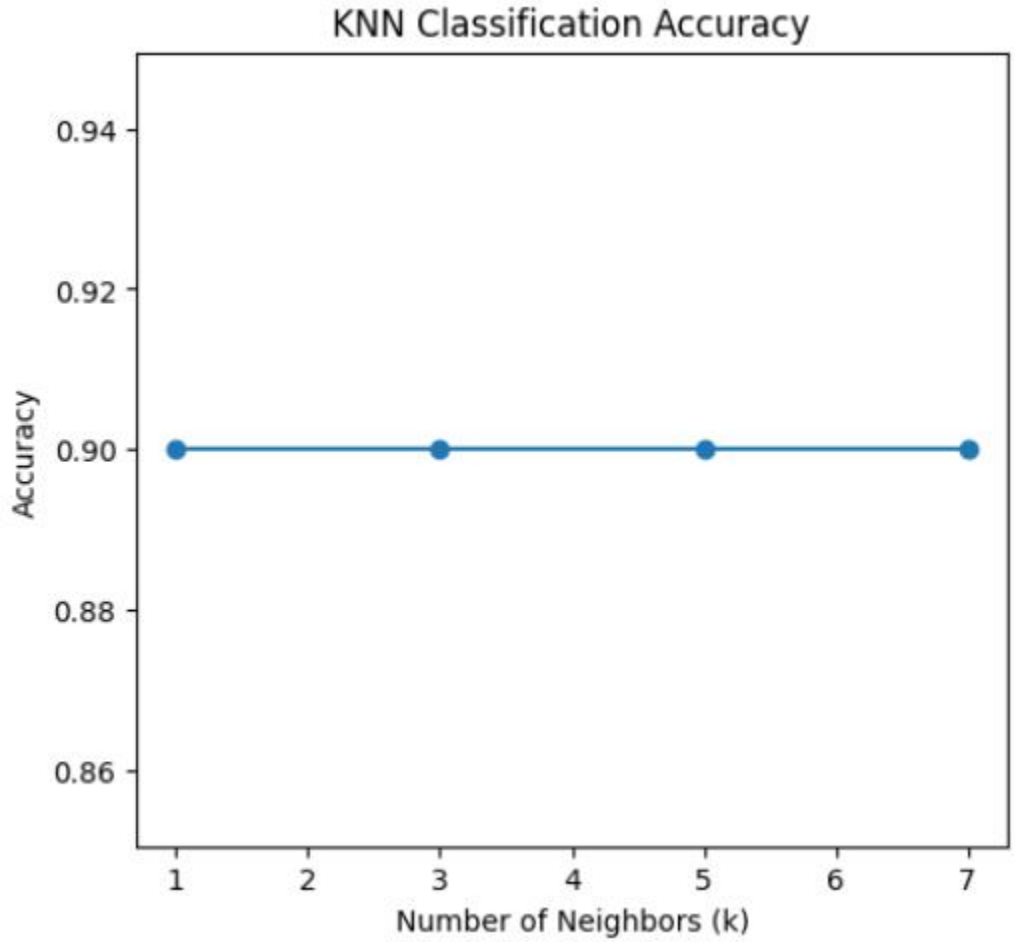
```
for k in k_values:
      clf = KNeighborsClassifier(n_neighbors=k)
      clf.fit(X_train, y_train_classification)
      y_pred = clf.predict(X_test)
      accuracy = accuracy_score(y_test_classification, y_pred)
      classification_scores.append(accuracy)
```

```
# Analyze different numbers of neighbors (k) for regression
regression_scores = []
```

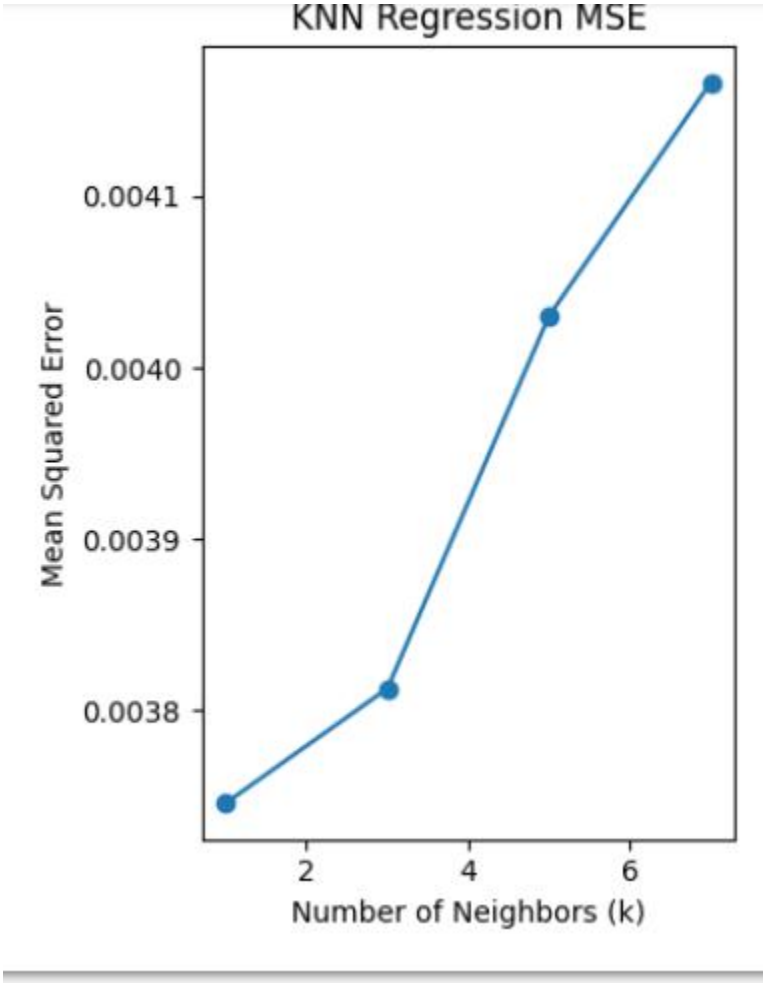
```
for k in k_values:
      reg = KNeighborsRegressor(n_neighbors=k)
```

```
        reg.fit(X_train, y_train_regression)
        y_pred = reg.predict(X_test)
        mse = mean_squared_error(y_test_regression, y_pred)
        regression_scores.append(mse)
```

```
# Plot the results for classification
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_values, classification_scores, marker='o')
plt.title('KNN Classification Accuracy')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
```



```
# Plot the results for regression
plt.subplot(1, 2, 2)
plt.plot(k_values, regression_scores, marker='o')
plt.title('KNN Regression MSE')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Squared Error')
plt.tight_layout()
plt.show()
```


KNN Regression MSE
Mean Squared Error
0.0041
0.0040
0.0039
0.0038
2
4
6
Number of Neighbors (k)