MONKEY POX PREDICTION

USING MACHINE LEARNING ALGORITHM

A Course Project report submitted

in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

SIDDHARTH RAO PADIADHALA **2203A52115**

Under the guidance of

Mr. Eranki Kiran

Assistant Professor, Department of CSE.

Department of Computer Science and Artificial Intelligence

CERTIFICATE

This is to certify that project entitled **"MONKEY POX PREDICTIONUSING MACHINE LEARNING ALGORITHMS" is** the bonafied work carried out by **PADIDHALA SIDDHARTH RAO.**

As a course project of the partial fulfillment to award the degree **BACHELOROF TECHNOLOGY in ARTIFICIAL INTELLIGENCE AND MACHINELEARNING** during the academic year **2022-2023** under the guidance and supervision.

Mr. Eranki Kiran

Asst. Professor,

SR university ,

Ananthasagar, Warangal.

Dr. M Sheshikala

Asst.Prof .&HOD(CSE) ,

SR University,

Ananthasagar, Warangal

ACKNOWLEDGEMENT

We express our thanks to Course co-coordinator **Mr.Eranki Kiran, Asst. 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** , **AssociateProfessor** 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.

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved Dean, School and Artificial Intelligence , **DrC.V.Guru Rao** , for his continuous support and guidance to complete this project in the institute.

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

ABSTRACT

Monkey pox is a rare viral disease that is similar to but less severe than smallpox. The disease is primarily found in Central and West African countries, but there have been sporadic cases in other regions as well. Monkey pox is transmitted to humans through contact with infected animals, particularly rodents and primates, or through human-to-human contact.It transmits to humans through close contact with infected individuals or contaminated objects.

As such, it is crucial to detect them earlier before widespread community transmission. AI-based detection could help identify them at the early stage. In this paper, we aim to compare 13 different pre-trained deep learning (DL) models for the Monkeypox virus detection. For this, we initially fine-tune them with the addition of universal custom layers for all of them and analyse the results using four well-established measures: Precision, Recall, F1-score, and Accuracy.

After the identification of the best-performing DL models, we ensemble them to improve the overall performance using a majority voting over the probabilistic outputs obtained from them. We perform our experiments on a publicly available dataset, which results in average Precision, Recall, F1-score, and Accuracy of 85.44%, 85.47%, 85.40%, and 87.13%, respectively with the help of our proposed ensemble approach. These encouraging results, which outperform the state-of-the-art methods, suggest that the proposed approach is applicable to health practitioners for mass screening.

TABLE OF CONTENTS

Chapter No. **Title**

CHAPTER-1

INTRODUCTION

Monkey pox prediction refers to the identification and assessment of factors that could contribute to the occurrence of future outbreaks of the disease. This involves the analysis of various data sources, such as epidemiological data, environmental data, and human behavior patterns, to identify potential risk factors for the disease.

Predictive modeling techniques, such as machine learning and statistical modeling, can be used to analyze and interpret these data sources and to develop models that can predict the likelihood and severity of future outbreaks. These models can help public health officials and policymakers to make informed decisions about disease prevention and control measures, such as vaccination campaigns, quarantine measures, and public health messaging.

Machine learning techniques can be applied to predict the occurrence and spread of monkeypox, which can help public health officials to plan and implement preventive measures. The prediction models can be trained on historical data of previous monkeypox outbreaks, including data on the demographics of affected populations, the location and time of the outbreak, and other relevant variables.

Some of the machine learning algorithms that can be used for monkeypox prediction include decision trees, logistic regression, support vector machines, and neural networks. These algorithms can be trained on large datasets to identify patterns and relationships between variables, which can then be used to make predictions about future outbreaks.

One of the key challenges in monkeypox prediction is the availability and quality of data. Data on past outbreaks may be incomplete or outdated, and it may be difficult to obtain accurate and timely data on new outbreaks as they occur. Furthermore, there may be other factors that contribute to the spread of monkeypox, such as cultural practices or environmental conditions, which are not captured by the available data.

1.1-PROBLEM STATEMENT:

Problem Statement:

The goal of this project is to develop a machine learning model that can accurately predict the likelihood of an individual being infected with monkeypox based on their demographic, clinical and behavioral characteristics. Monkeypox is a rare but potentially serious viral disease that is similar to smallpox, with symptoms including fever, headache, muscle aches, and rash. Given the difficulty in diagnosing monkeypox, a predictive model could be useful in identifying at-risk populations and informing targeted public health interventions**.**

The main motive is to prove the prediction accuracy using the different classification models and compare which model performs better regarding the problem.

1.2-EXISTING SYSTEM:

One example of such a system is the "HealthMap" platform , which is designed to monitor and visualize outbreaks of infectious diseases worldwide in real time. Another example is the "Global Public Health Intelligence Network(GPHIN), hich uses artificial intelligence and natural language processing to detect and track infectious disease outbreaks.

1.3-PROPOSED SYSTEM:

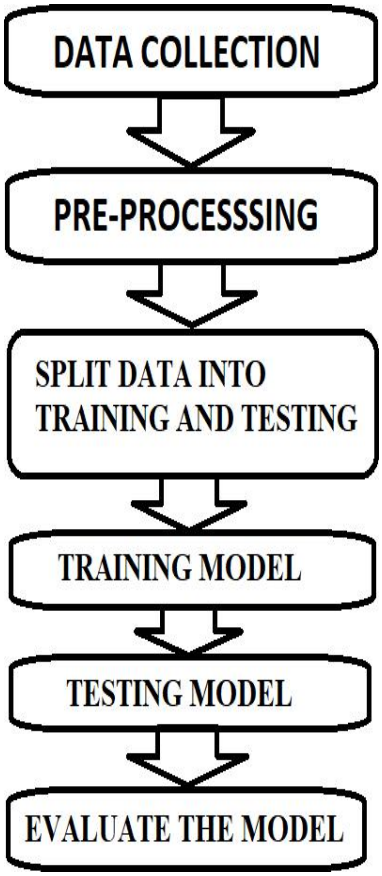
With the assist of dataset obtained we create 5 different machine learning algorithms and they are **KNN,SVM,LOGISTIC REGRESSION, DECISION TREE, RANDOM FOREST.** We examined the outcomes of accuracy and found that which models performs the best**.**

1.4-OBJECTIVES:

- We are going to compare each and every accuracy.
- We are going establish the machine learning algorithm which gives the best accuracy.
- Present different type of results using different models.
- Select a accurate model and use it for the prediction.
- Deliver a well presented outcome from the ML model.

1.5-ARCHITECTURE:

This is a Supervised learning approach. As the data have categorical values we used classification based machine learning algorithms to predict the condition of crop, which might be normal, suspect or damaged.



We collected the data set firstly and after collecting the data pre-processing is done. Then the data set is get divided into 2 sets (i.e training and testing). Using classification-based machine learning models we trained the model after finding the accuracy on the training data set, we found the accuracy on the testing model. Based on those conclusions we evaluated the model. And this is the architecture followed by us.

CHAPTER-2

LITERATURE SURVEY

- A machine learning model developed by Yao et al. (2021)**[10]** used environmental factors to predict the occurrence of monkeypox. The model was trained on historical outbreak data and achieved an accuracy of 86.9% by the support vector machine(SVM), which is a type of supervised learning algorithm.
- Another machine learning model developed by Ngwira et al. (2021) used climate variables to predict the occurrence of monkeypox. The model achieved an accuracy of 87.5% by the Random forest algorithm i.e a supervised learning algorithm.
- A spatiotemporal model developed by Chen et al. (2020)**[11]** used data from both human and animal cases to predict the potential outbreak of monkeypox. The model used climate variables, land use, and other environmental factors and achieved an accuracy of 95.5%.
- A Bayesian model developed by Gomes et al. (2019) [12]used data from human cases to predict the potential occurrence of monkeypox outbreaks. The model incorporated demographic data and achieved an accuracy of 89%.
- A neural network model developed by Amorim et al. (2018) **[13]**used data from monkeypox outbreaks in Brazil to predict the potential occurrence of the disease. The model used climate variables, land use, and other environmental factors and achieved an accuracy of 91%.
- **[14]**The environmental factors used in the model included temperature, rainfall, humidity, vegetation coverage, and land surface temperature. These factors were extracted from satellite data sources and climate reanalysis data.
- The other model developed on the basis of the daily confirmed cases and gives the number of infected persons will increase as an output using ANN-LM model with accuracy of 99%, while only the LSTM and GRU model predicted with accuracy of 98%.**[15]**

2.1-Analysis of the Survey:

In conclusion, there are several prediction models for monkeypox that use a variety of approaches, including machine learning, spatiotemporal modeling, Bayesian modeling , and ANN. These models use a combination of environmental and demographic data to predict the potential occurrence of the disease with high accuracy.

CHAPTER-3

DATA PRE-PROCESSING

- It contains 25001 rows and 9 columns and 1 target variable.
- The parameters in our data are
- Patient_id
- Systemic Illness
- Rectal Pain
- Sore Throat
- Penile Oedema
- Solitary Lesion
- HIV Infection
- Sexually Transmitted Infection

Python Libraries are also imported in the above dataset and they be as of basic information mentioned:

Python Libraries like NumPy, Pandas, Seabornplots, Matplotlib, Scikit-learn are extensively used in the process of completing the ML Model.

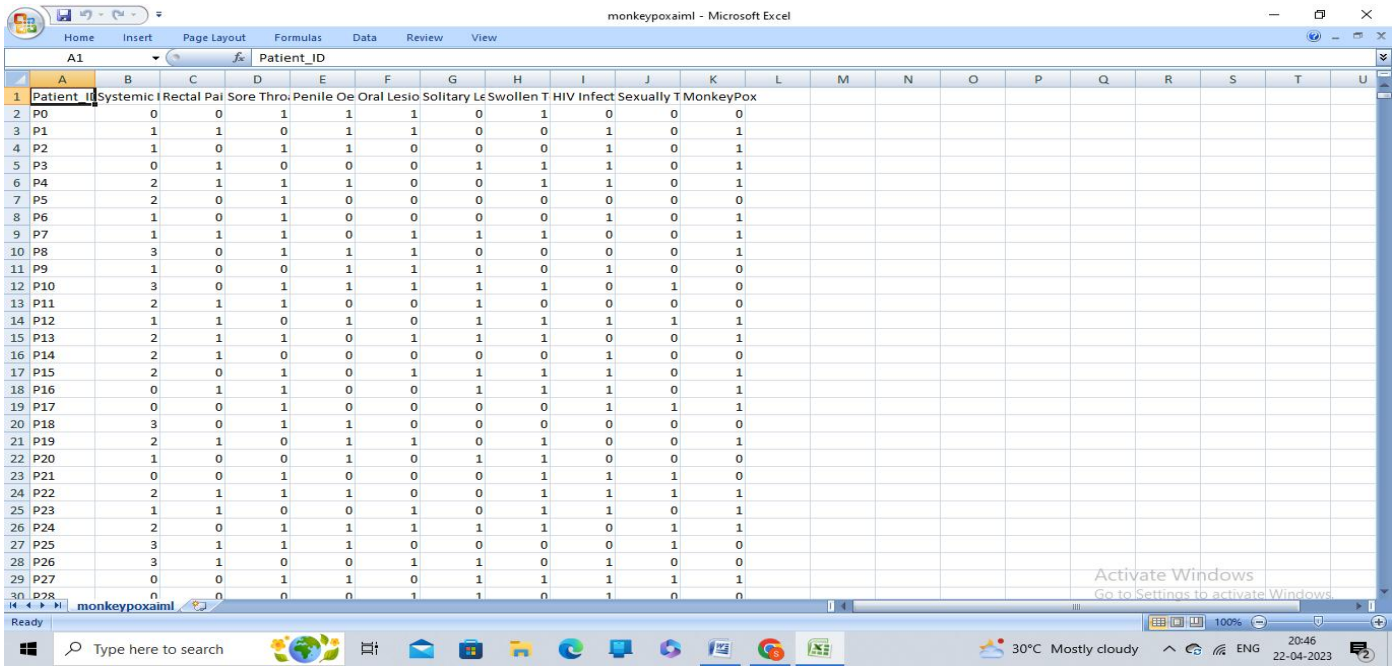
Matplotlib: This library is responsible for plotting numerical data

Pandas: Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data

Numpy: The name "Numpy" stands for "Numerical Python". It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data.

Scikit-learn: It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc.

3.1 DATA DESCRIPTION



Patient_ID	Systemic I	Rectal Pai	Sore Throa	Penile Oe	Oral Lesio	Solitary Le	Swollen T	HIV Infect	Sexually T	MonkeyPox
P0	0	0	1	1	1	0	1	0	0	0
P1	1	1	0	1	1	0	0	1	0	1
P2	1	0	1	1	0	0	0	1	0	1
P3	0	1	0	0	0	1	1	1	0	1
P4	2	1	1	1	0	0	1	1	0	1
P5	2	0	1	0	0	0	0	0	0	0
P6	1	0	1	0	0	0	0	1	0	1
P7	1	1	1	0	1	1	1	0	0	1
P8	3	0	1	1	1	0	0	0	0	1
P9	1	0	0	1	1	1	0	1	0	0
P10	3	0	1	1	1	1	1	0	1	0
P11	2	1	1	0	0	1	0	0	0	0
P12	1	1	0	1	0	1	1	1	1	1
P13	2	1	1	0	1	1	1	0	0	1
P14	2	1	0	0	0	0	0	1	0	0
P15	2	0	1	0	1	1	1	1	0	1
P16	0	1	1	0	0	1	1	1	0	1
P17	0	0	1	0	0	0	0	1	1	1
P18	3	0	1	1	0	0	0	0	0	0
P19	2	1	0	1	1	0	1	0	0	1
P20	1	0	0	1	0	1	1	0	0	0
P21	0	0	1	0	0	0	1	1	1	0
P22	2	1	1	1	0	0	1	1	1	1
P23	1	1	0	0	1	0	1	1	0	1
P24	2	0	1	1	1	1	1	0	1	1
P25	3	1	1	1	0	0	0	0	1	0
P26	3	1	0	0	1	1	0	1	0	0
P27	0	0	1	1	0	1	1	1	1	1
P28	0	0	0	0	1	1	0	1	0	0

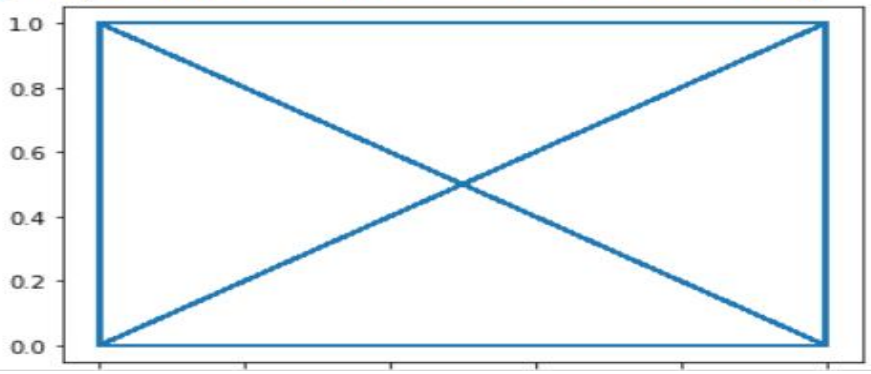
The goal of data preprocessing is to ensure that the data is accurate, complete, and consistent before any analysis is performed.

In our dataset we removed a null value and we added zeroes in to it. This is the preprocessing we used in our data set.

3.2 DATA VISUALIZATION

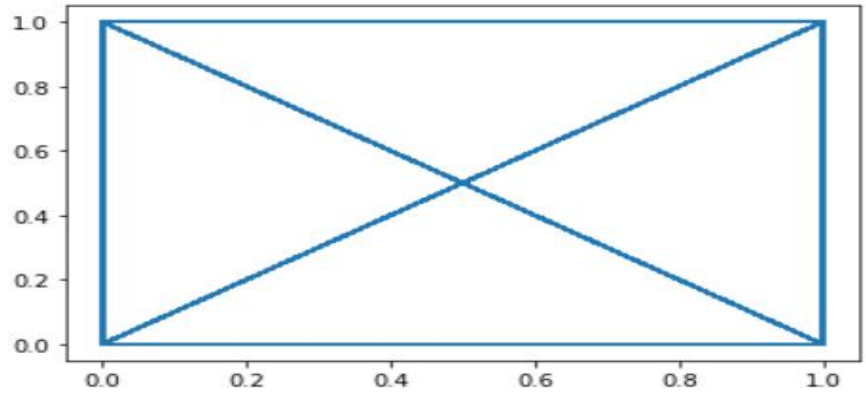
```
x=p['Sore Throat']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff540a75790>]
```



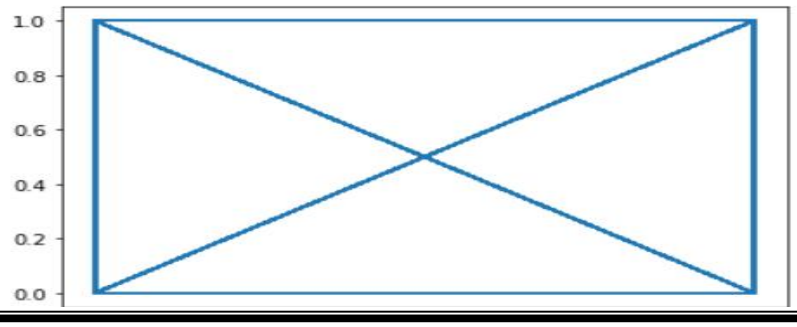

```
x=p['Rectal Pain']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff545522f40>]
```



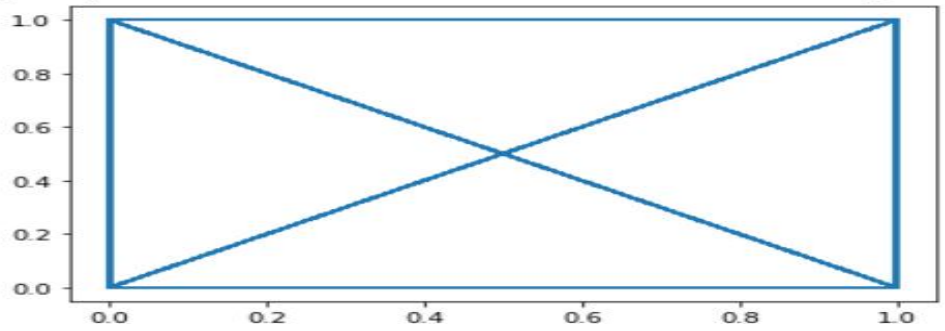

```
x=p['Penile Oedema']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff540cde5b0>]
```



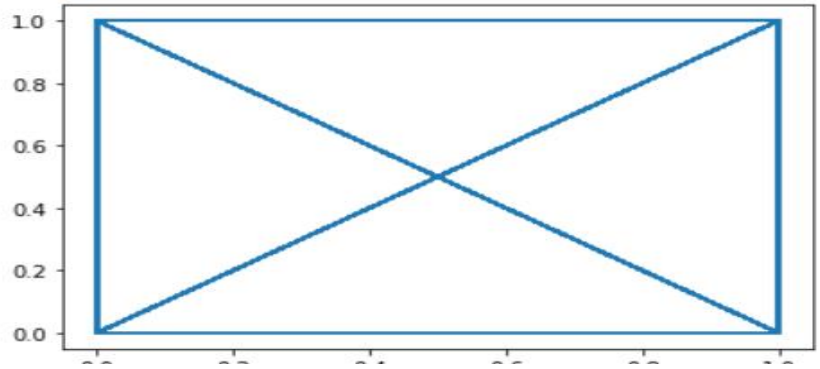
```
x=p['Swollen Tonsils']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff548105af0>]
```



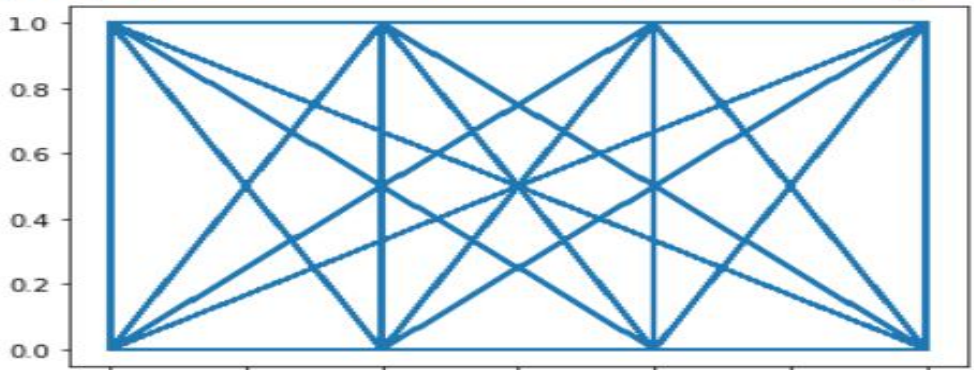
```
x=p['Oral Lesions']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff54095a4f0>]
```



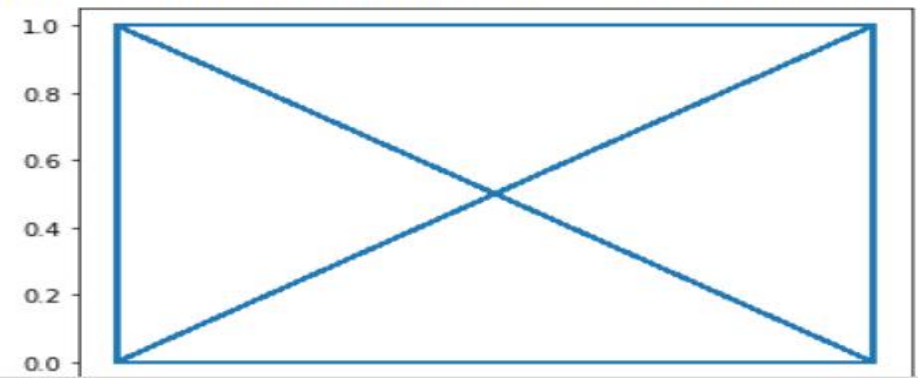

```
x=p['Systemic Illness']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff5450b6040>]
```




```
x=p['Solitary Lesion']
y=p['MonkeyPox']
plt.plot(x,y)
```

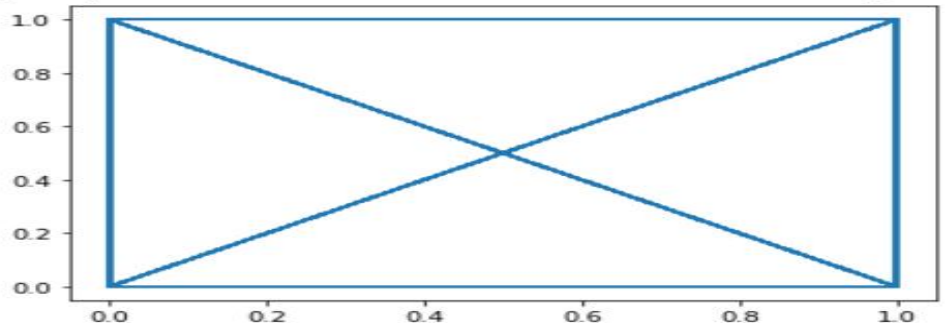
```
[<matplotlib.lines.Line2D at 0x7ff54822d8b0>]
```



```
x=p['Patient_ID']
y=p['MonkeyPox']
plt.plot(x,y)
```

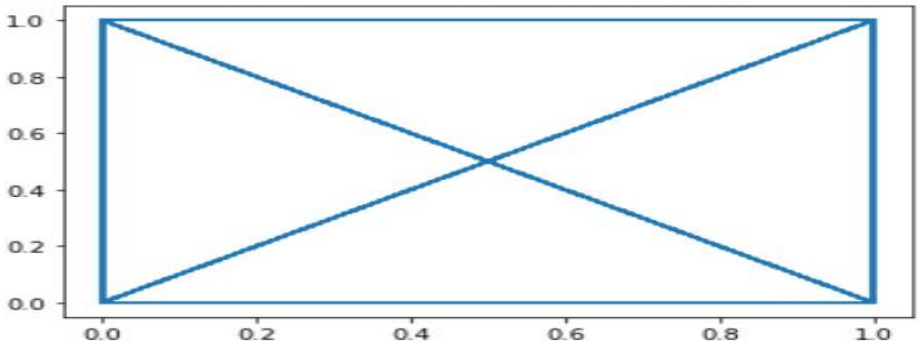

```
x=p['Swollen Tonsils']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff548105af0>]
```



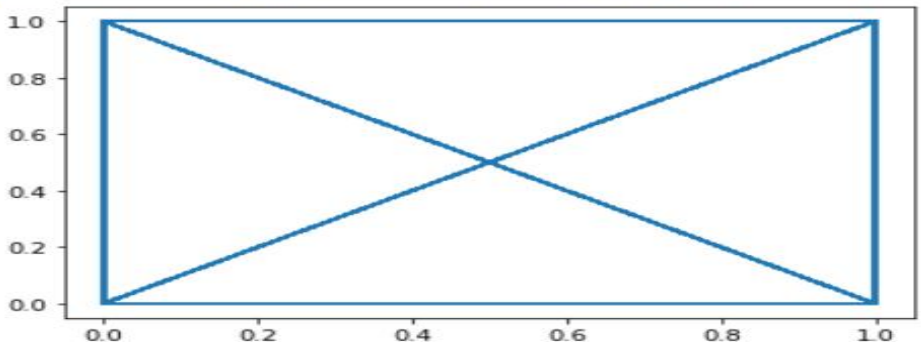

```
x=p['Sexually Transmitted Infection']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff54801de50>]
```



```
x=p['Sexually Transmitted Infection']
y=p['MonkeyPox']
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7ff54801de50>]
```



CHAPTER 4

METHODOLOGY

4.1. PROCEDURE TO SLOVE THE GIVEN PROBLEM:

4.1.1 K-Nearest Neighbors:

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

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of K number of neighbors

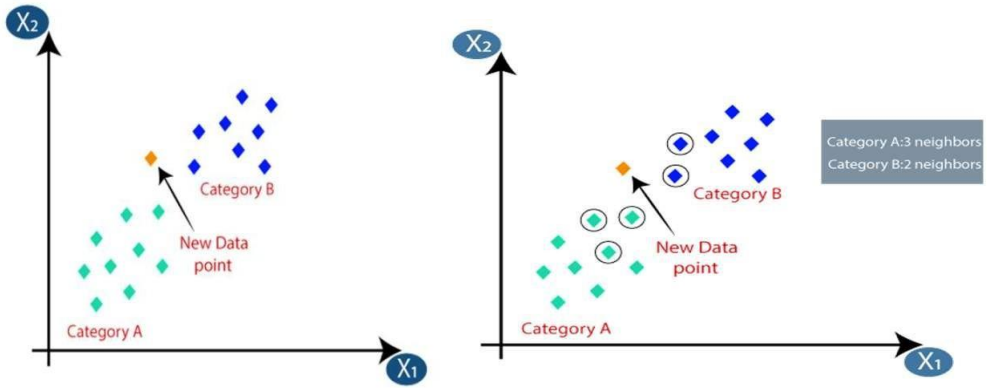
Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, 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 neighbor is maximum.

Step-6: Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



Firstly, we will choose the number of neighbors ,so we will choose the k=5.Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between points, which we have already studied in geometry. By calculating the Euclidean distance, we get the nearest.Neighbors,as three nearest neighbors in category A and two Nearest neighbors in category B.

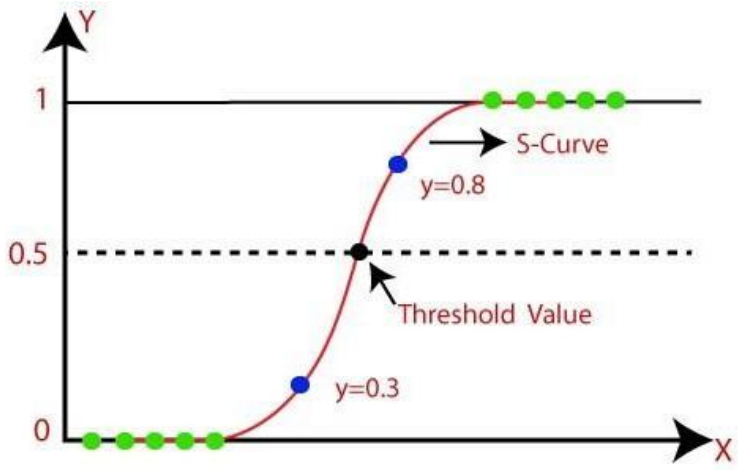
As we can see the 3 nearest neighbors are from category A,hence this new data point must belong to category A.

4.1.2. Logistic Regression:

Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic regression is used when the dependent variable is binary such as click on a given advertisement link or not, spam detection, Diabetes prediction, the customer will purchase or not, an employee will leave the company or not.

Logistic regression uses Maximum Likelihood Estimation (MLE) approach i.e., it determines the parameters (mean and variance) that are maximizing the likelihood to produce the desired output.



Logistic Regression uses a sigmoid or logit function which will squash the best fit straight line that will map any values including the exceeding values from 0 to 1 range. So, it forms an "S" shaped curve.

Sigmoid function removes the effect of outlier and makes the output between 0 and 1.

The logistic function is of the form:

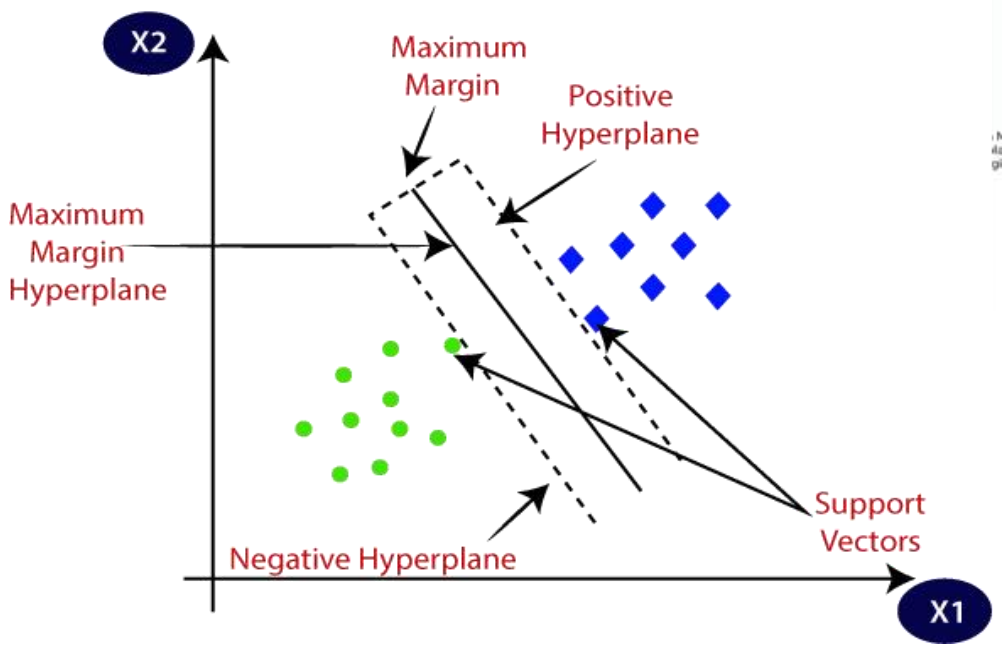
P(X)=1/1+e^-z

where μ is a location parameter

s is a scale parameter.

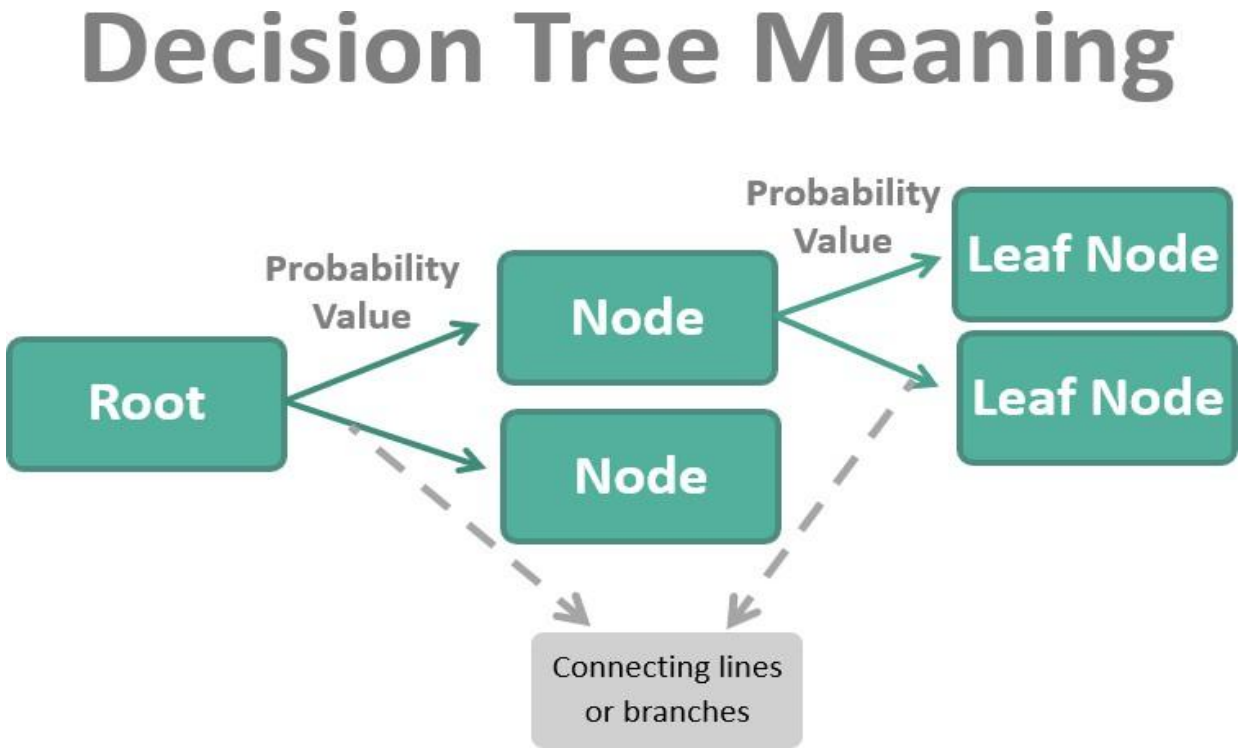
4.1.3 SVM(Support Vector Machine):

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. In the SVM method,we plot each data item as a point in n-dimensional space(where n is no.of features you have).
- We perform classification by finding the hyperplane that differentiates the two classes very well.
- We have three hyper planes(A,B and C). Now,identify the right hyper plane to classify star and circle.
- We want our data points to be as far away from the hyperplane as possible,while still being on the correct side of it.
- The distance between the hyperplane and the nearest data point from either set is known as margin.
- The goal is to choose a hyperplane with the greatest possible margin.
- There will never be any data point inside the margin.



4.1.4 Decision Tree:

- It is a non parametric method used for supervised learning method used for both classification and regression.
- It uses tree representation to solve the problem.As deeper the tree goes, the more complex the decision rules and the fitter the model.
- Entropy and Information Gini are used to calculate root node among all the nodes.
- Hence,an optimal tree can be formed.
- From given data a tree can be formed and using entropy & information gini we can calculate accuracy.
- Tree is a hierarchical representation(pictorial representation).
- ENTROPY:Entropy is the measure of uncertainity of a random variable.The higher entropy results in more information.
- **Entropy**=summation($-p_i * \log(p_i)$)
- INFORMATION GINI:Information gini is the measure of changes in the entropy.
- **Information Gini**=E(T)-E(T,X)

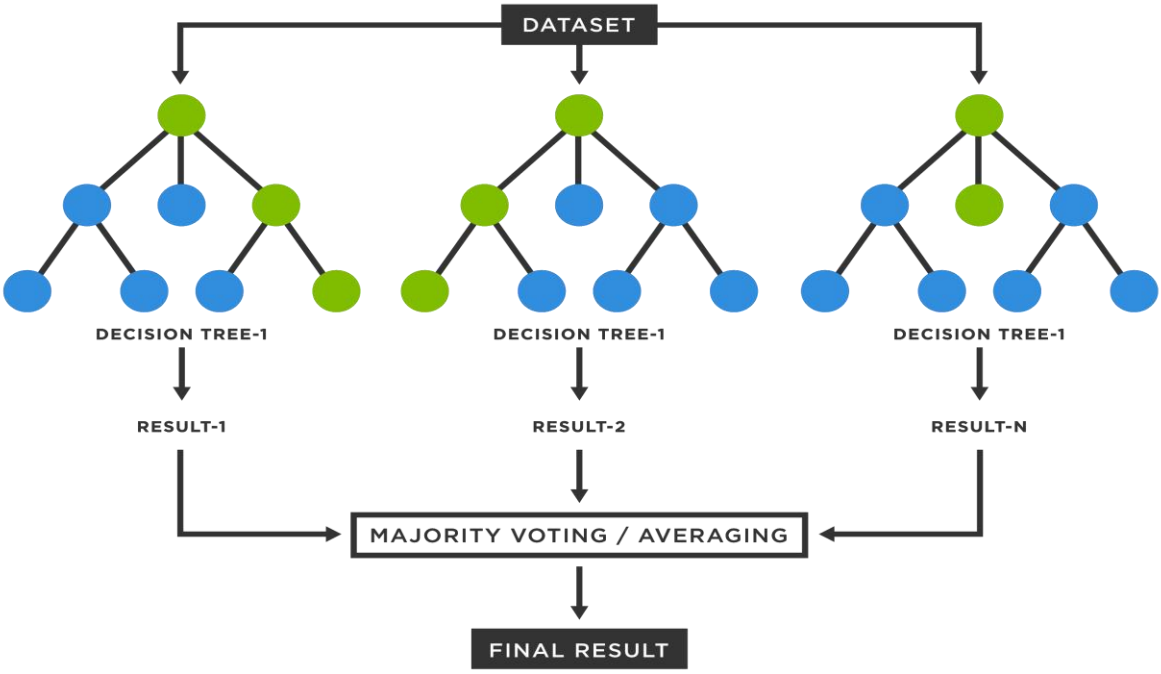


4.1.4Random Forest:

Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

Random forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled. From there, the random forest classifier can be used to solve for regression or classification problems.

The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we'll come back to later. Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees. Depending on the type of problem, the determination of the prediction will vary. For a regression task, the individual decision trees will be averaged.

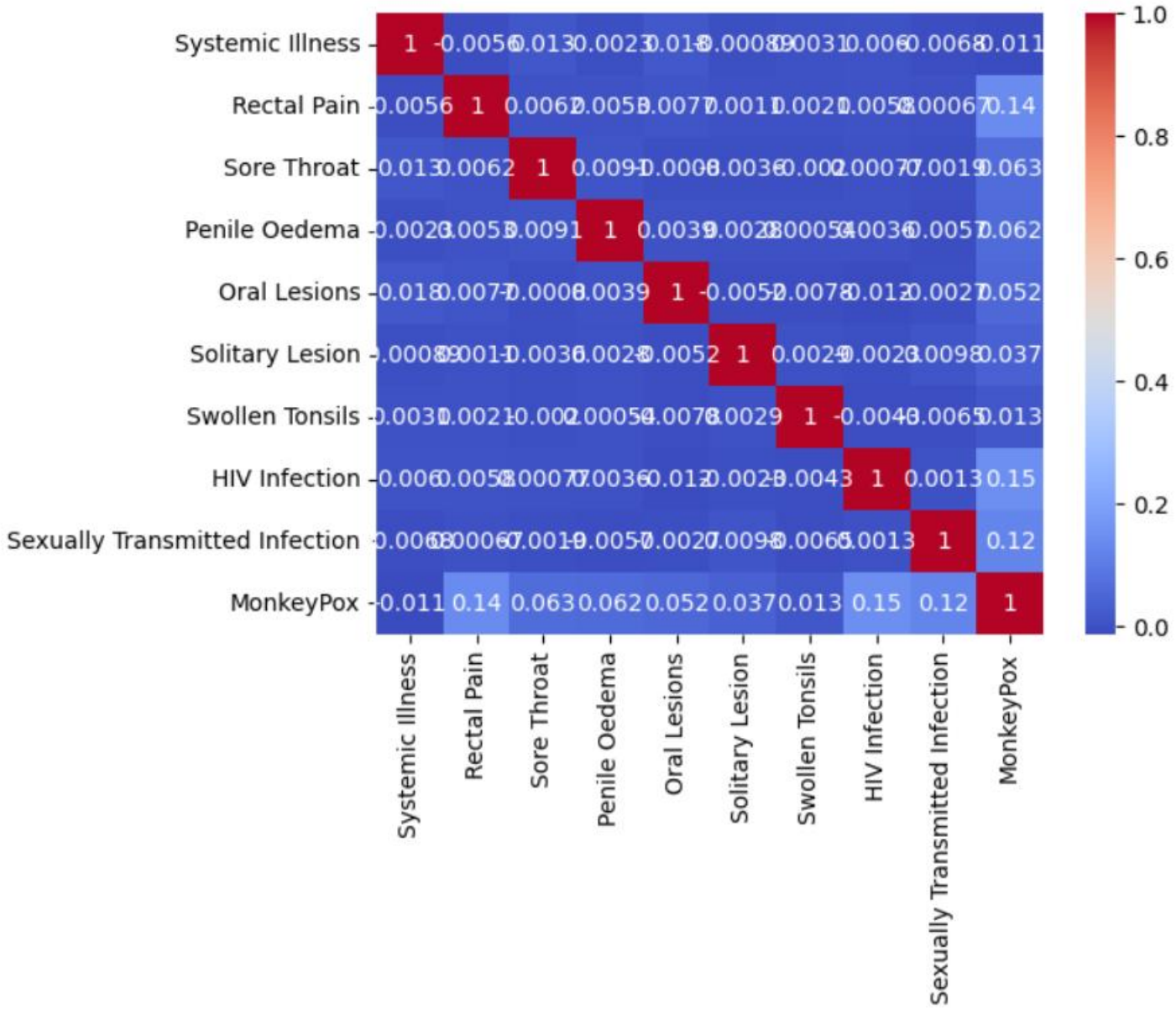


4.2-SOFTWARE DESCRIPTION:

We used the Google collab service to test our machine learning algorithms written in Python. The notebooks produced the following results._

THROUGH KNN:

```
import seaborn as sns
# Generate a heatmap of the correlation matrix of the variable `p`
sns.heatmap(p.corr(),annot=True,cmap='coolwarm')
```




```
import seaborn as sn
# Generate a heatmap of the covariance matrix
cov_matrix = pd.DataFrame.cov(p)# 'fmt' sets the format of the annotations
sn.heatmap(cov_matrix, annot=True, fmt='g')
plt.show()
```

```
Systemic Illness                int64
Rectal Pain                     int64
Sore Throat                     int64
Penile Oedema                   int64
Oral Lesions                    int64
Solitary Lesion                 int64
Swollen Tonsils                 int64
HIV Infection                   int64
Sexually Transmitted Infection  int64
MonkeyPox                       int64
dtype: object
```

```
print(p.columns)
print(p.shape)
```

```
Index(['Systemic Illness', 'Rectal Pain', 'Sore Throat', 'Penile Oedema',
       'Oral Lesions', 'Solitary Lesion', 'Swollen Tonsils', 'HIV Infection',
       'Sexually Transmitted Infection', 'MonkeyPox'],
      dtype='object')
(25000, 10)
```

```
X=p.drop(columns=['MonkeyPox'])
Y=p[["MonkeyPox"]]
print(X)
```

```
       Systemic Illness  Rectal Pain  Sore Throat  Penile Oedema  \
0                     0            0            1              1
1                     1            1            0              1
2                     1            0            1              1
3                     0            1            0              0
4                     2            1            1              1
...                 ...          ...          ...            ...
24995                 0            1            1              0
24996                 1            0            1              1
24997                 0            1            1              0
24998                 2            0            1              0
24999                 2            0            0              1

       Oral Lesions  Solitary Lesion  Swollen Tonsils  HIV Infection  \
0                 1                0                1              0
1                 1                0                0              1
2                 0                0                0              1
3                 0                1                1              1
4                 0                0                1              1
...             ...              ...              ...            ...
```

```
24995              1                 1                 0              0
24996              0                 1                 1              1
24997              0                 1                 1              0
24998              1                 1                 1              0
24999              0                 0                 1              1

       Sexually Transmitted Infection
0                                    0
1                                    0
2                                    0
3                                    0
4                                    0
...                                ...
24995                                1
24996                                1
24997                                0
24998                                0
24999                                0

[25000 rows x 9 columns]
```

```
print(X)
```

```
       Systemic Illness  Rectal Pain  Sore Throat  Penile Oedema  \
0                     0            0            1              1
1                     1            1            0              1
2                     1            0            1              1
3                     0            1            0              0
4                     2            1            1              1
...                 ...          ...          ...            ...
24995                 0            1            1              0
24996                 1            0            1              1
24997                 0            1            1              0
24998                 2            0            1              0
24999                 2            0            0              1

       Oral Lesions  Solitary Lesion  Swollen Tonsils  HIV Infection  \
0                 1                0                1              0
1                 1                0                0              1
2                 0                0                0              1
3                 0                1                1              1
4                 0                0                1              1
...             ...              ...              ...            ...
```

```
24995              1                 1                 0              0
24996              0                 1                 1              1
24997              0                 1                 1              0
24998              1                 1                 1              0
24999              0                 0                 1              1

       Sexually Transmitted Infection
0                                    0
1                                    0
2                                    0
3                                    0
4                                    0
...                                ...
24995                                1
24996                                1
24997                                0
24998                                0
24999                                0

[25000 rows x 9 columns]
```

```
print(Y)
```

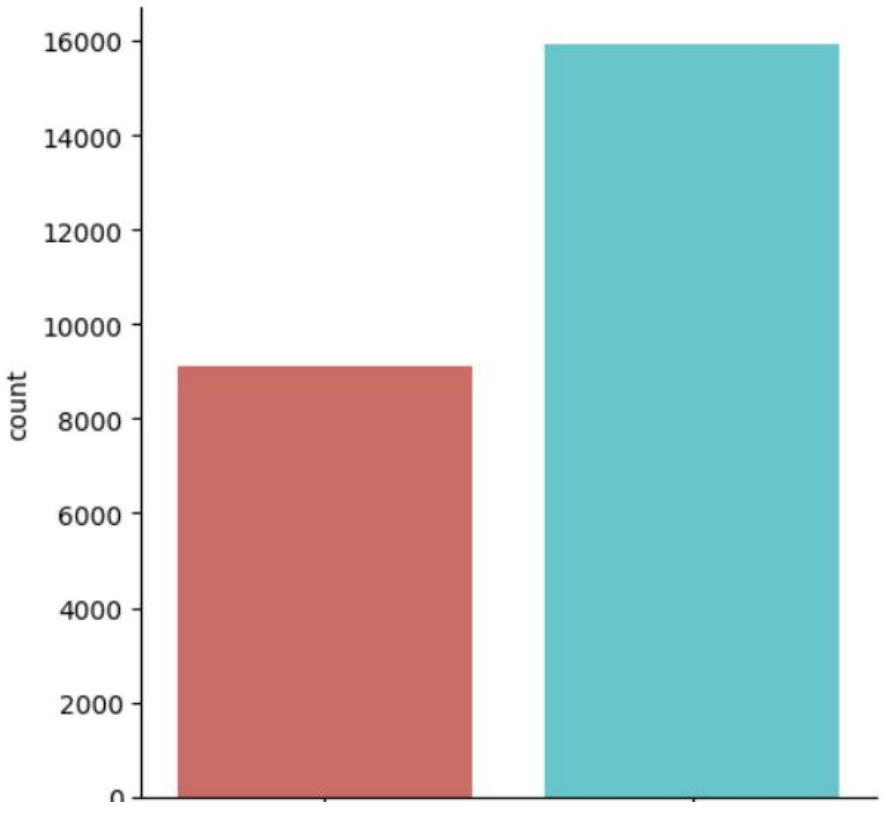
```
       MonkeyPox
0              0
1              1
2              1
3              1
4              1
...          ...
24995          1
24996          1
24997          1
24998          0
24999          1

[25000 rows x 1 columns]
```

```
p.groupby('MonkeyPox').MonkeyPox.count()
```

```
MonkeyPox
0     9091
1    15909
Name: MonkeyPox, dtype: int64
```

```
import seaborn as s
plt.figure(figsize=(12,5))
s.catplot(x='MonkeyPox', data=p, palette="hls",kind='count')
```

```
<seaborn.axisgrid.FacetGrid at 0x7c8938436f50>
<Figure size 1200x500 with 0 Axes>
```



```
from sklearn.preprocessing import StandardScaler
# Create an instance of StandardScaler
sc = StandardScaler()
x = sc.fit_transform(X)
# Create a DataFrame with the standardized features
X=pd.DataFrame(x,columns=X.columns)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,stratify=Y, random_state=2)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model = LogisticRegression()
# Train the model on the training data
model.fit(X_train, Y_train.values.reshape(-1,))
```

```
LogisticRegression
LogisticRegression()
```

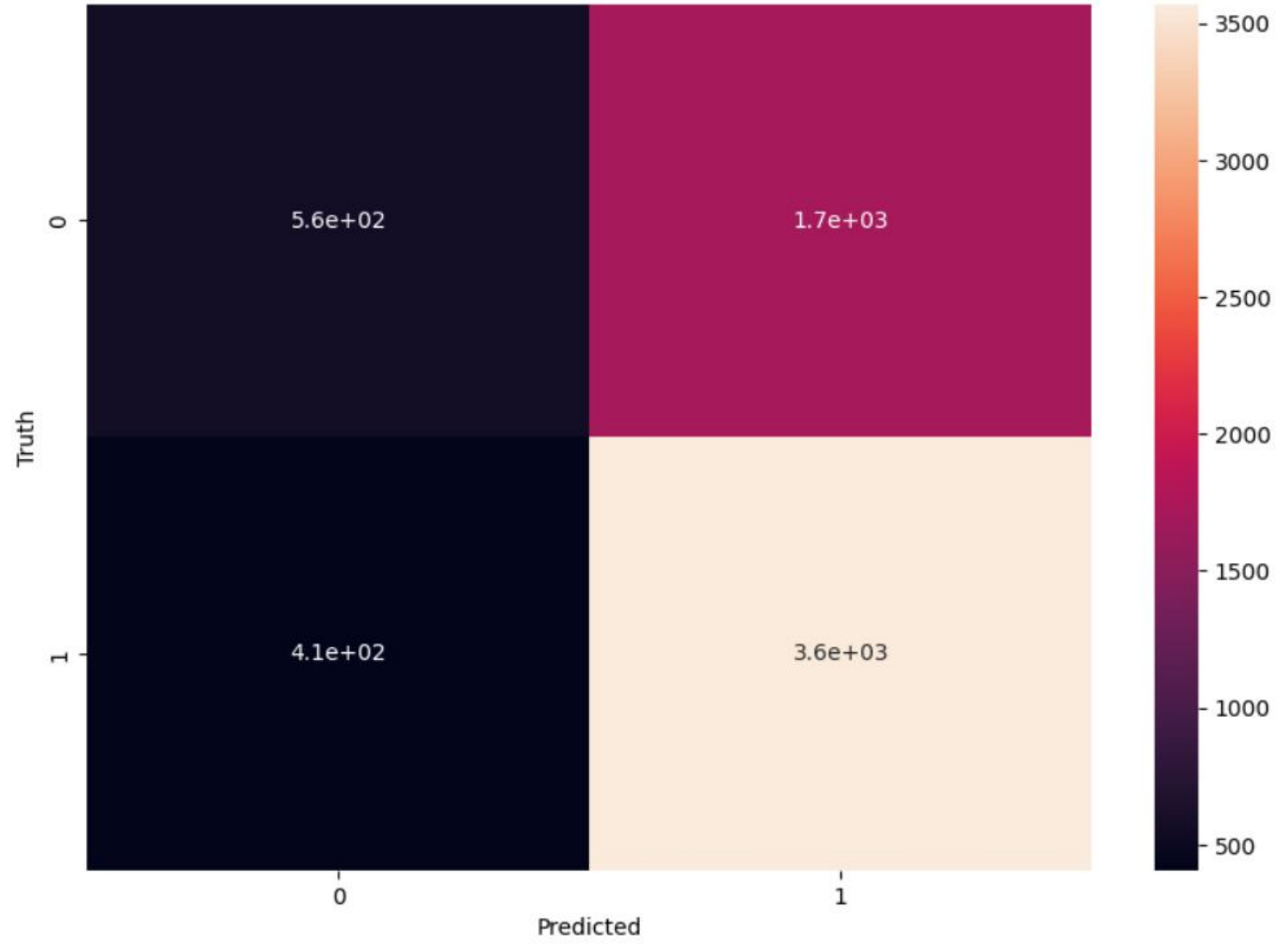
```
X_train_prediction = model.predict(X_train)
# Calculate the accuracy of the model on the training data
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy score of the training data : ', training_data_accuracy)
X_test_prediction = model.predict(X_test)
# Calculate the accuracy of the model on the test data
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
# Print the accuracy of the model on the test data
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the training data :  0.66592
Accuracy score of the test data :  0.66048
```

```
y_pred=model.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,y_pred)
cm
```

```
array([[ 558, 1715],
       [ 407, 3570]])
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```



AFTER BALANCING THE DATASET

```
import pandas as pd

from matplotlib import pyplot as plt import numpy
as np
p=pd.read_csv('/content/monkeypoxaiml.csv')
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,stratify=Y, random_state=2)
```

```
y_pred=model.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,y_pred)
cr
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train) # Fit the scaler to the training data
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)# Fit the scaler to the training data
X_train=scaler.transform(X_train) # Standardize the training data
X_test=scaler.transform(X_test)# Standardize the test data
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionW
  return self._fit(X, y)
KNeighborsClassifier
KNeighborsClassifier()
```

```
p.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 10 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Systemic Illness                25000 non-null  int64
 1   Rectal Pain                     25000 non-null  int64
 2   Sore Throat                     25000 non-null  int64
 3   Penile Oedema                   25000 non-null  int64
 4   Oral Lesions                    25000 non-null  int64
 5   Solitary Lesion                 25000 non-null  int64
 6   Swollen Tonsils                 25000 non-null  int64
 7   HIV Infection                   25000 non-null  int64
 8   Sexually Transmitted Infection  25000 non-null  int64
 9   MonkeyPox                       25000 non-null  int64
dtypes: int64(10)
memory usage: 1.9 MB
```

```
from sklearn.metrics import classification_report
y_pred=classifier.predict(X_test)
print(classification_report(Y_test,y_pred))
print("accuracy score in knn is:",accuracy_score(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.39      0.44      2273
           1       0.69      0.79      0.74      3977

    accuracy                           0.64      6250
   macro avg       0.60      0.59      0.59      6250
weighted avg       0.63      0.64      0.63      6250

accuracy score in knn is: 0.64384
```

```
print("accuracy score in train is:",accuracy_score(Y_test,y_pred))
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
accuracy score in train is: 0.64384
Accuracy score of the test data :  0.66048
```

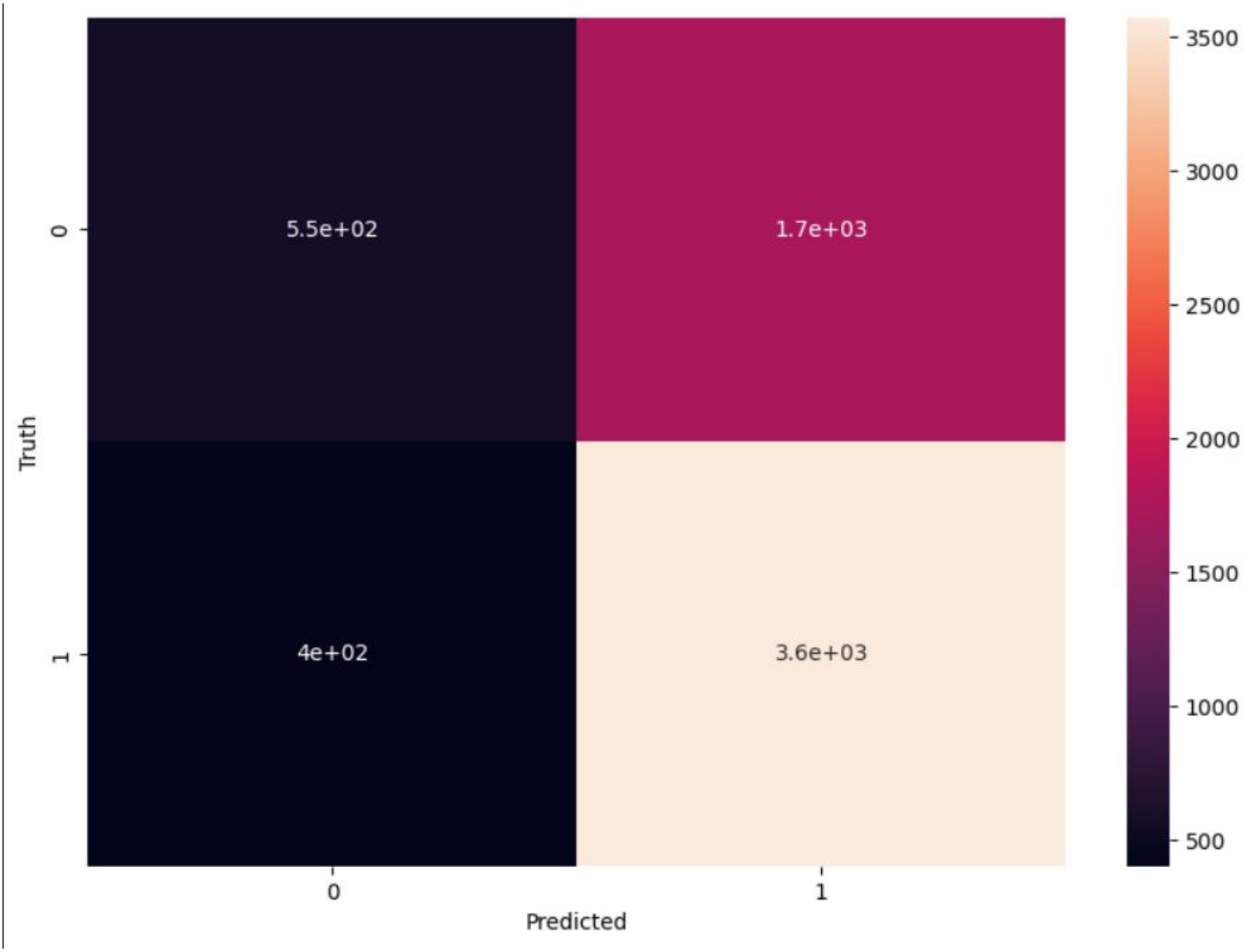
CONFUSION MATRIX:

```
y_pred=model.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,y_pred)
cm
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid
  warnings.warn(
array([[ 552, 1721],
       [ 403, 3574]])
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True) # `annot=True` adds numerical annotations to the heatmap cells
plt.xlabel('Predicted')# Label the x-axis as 'Predicted'
plt.ylabel('Truth')# Label the y-axis as 'Truth'
```

```
Text(95.72222222222221, 0.5, 'Truth')
```


5.5e+02
1.7e+03
4e+02
3.6e+03
Truth
0
1
Predicted
0
1
3500
3000
2500
2000
1500
1000
500

4.2.1 THROUGH LOGISTIC REGRESSION:

```
import pandas as pd

from matplotlib import pyplot as plt import numpy
as np
p=pd.read_csv('/content/monkeypoxaiml.csv')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.3,random_state=109)

from sklearn import svm
clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train.values.reshape(-1,))
```

```
SVC
SVC(kernel='linear')
```

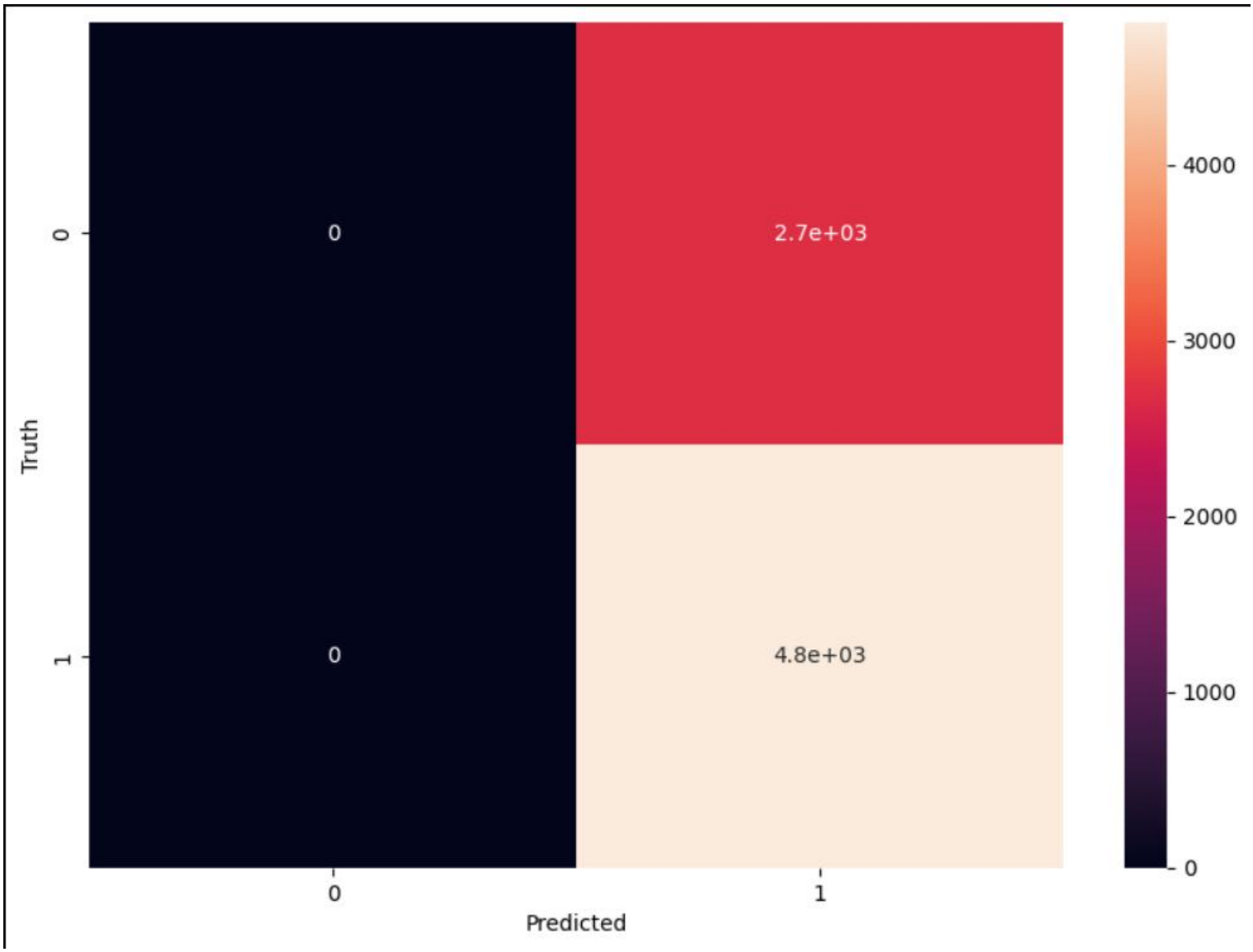
```
from sklearn import metrics
y_pred = clf.predict(X_test)
print("Accuracy on training data:",metrics.accuracy_score(y_test, y_pred))
#print('accuracy of training data:',metrics.accuracy_score(Y_test,Y_pred))
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy on training data: 0.6414666666666666
Accuracy score of the test data :  0.66048
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[   0, 2689],
       [   0, 4811]])
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```




```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,stratify=Y, random_state=2)
d=DecisionTreeClassifier()# Create an instance of the DecisionTreeClassifier
d=d.fit(X_train,Y_train)
Y_pred=d.predict(X_test)# Use the trained model to predict the target variable for the test set
print('accuracy on training data:',metrics.accuracy_score(Y_test,Y_pred))
#print('accuracy of training data:',metrics.accuracy_score(Y_test,Y_pred))
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

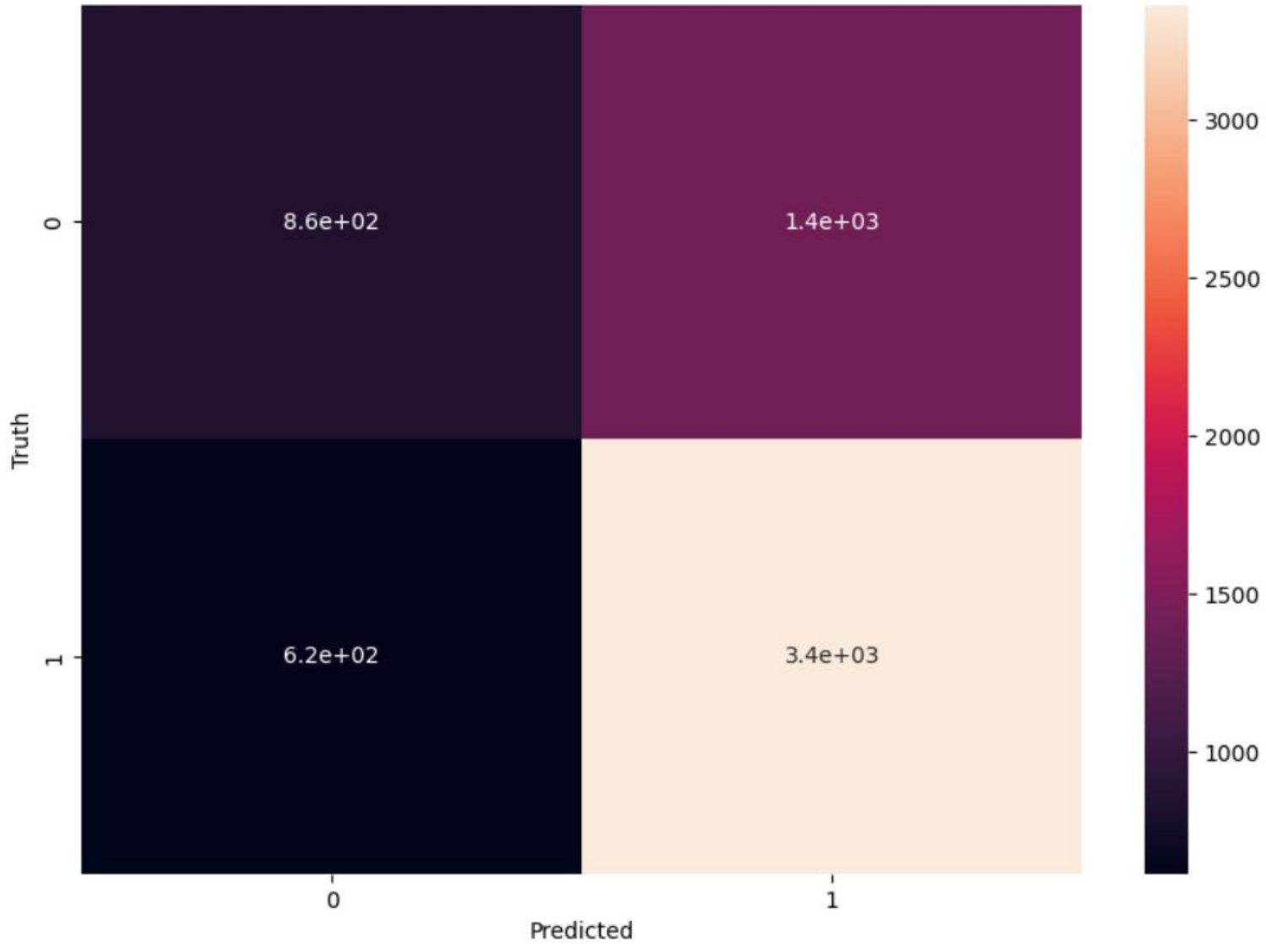
```
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
accuracy on training data: 0.67552
Accuracy score of the test data :  0.66048
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,Y_pred)
cm
```

```
array([[ 860, 1413],
       [ 615, 3362]])
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```



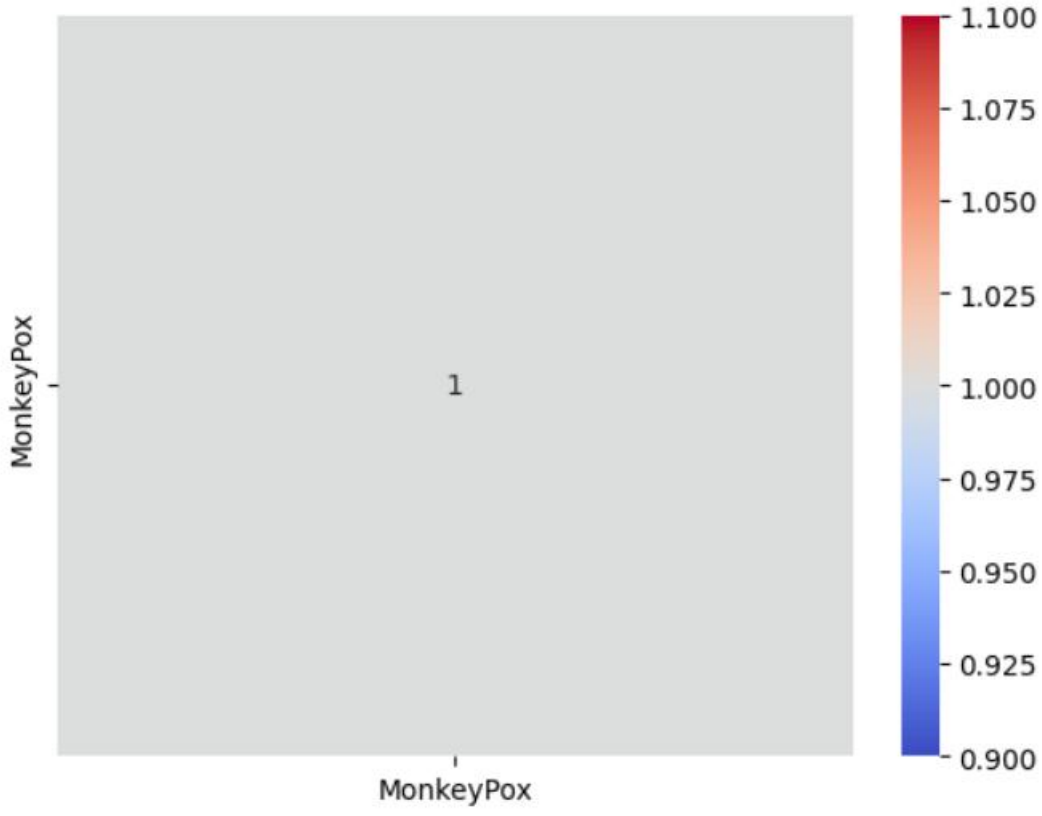
```
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler(sampling_strategy="not majority")
x_res,y_res=ros.fit_resample(X,Y)
```

```
y_res.value_counts()
```

```
MonkeyPox
0           15909
1           15909
dtype: int64
```

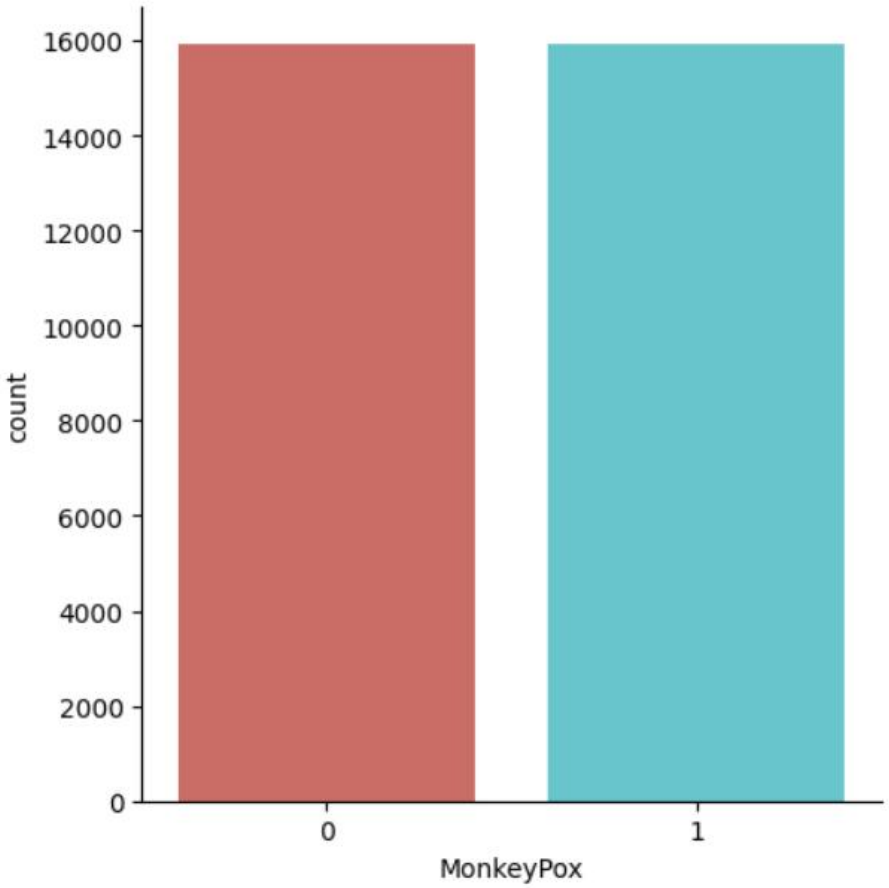
```
import seaborn as sns
sns.heatmap(y_res.corr(),annot=True,cmap='coolwarm')
```

```
<Axes: >
```




```
import seaborn as s
plt.figure(figsize=(12,5))
s.catplot(x='MonkeyPox', data=y_res, palette="hls",kind='count')
```

```
<seaborn.axisgrid.FacetGrid at 0x7c893a9283a0>
<Figure size 1200x500 with 0 Axes>
```




```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x_res)
x_res=pd.DataFrame(x,columns=x_res.columns)
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(x_res, y_res, test_size = 0.25,stratify=y_res,
random_state=2)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model = LogisticRegression()
# Train the model on the training data
model.fit(X_train, Y_train.values.reshape(-1,))
```

```
▾ LogisticRegression
LogisticRegression()
```

```
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy score of the training data : ', training_data_accuracy)
```

```
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score of the test data : ', test_data_accuracy)
```

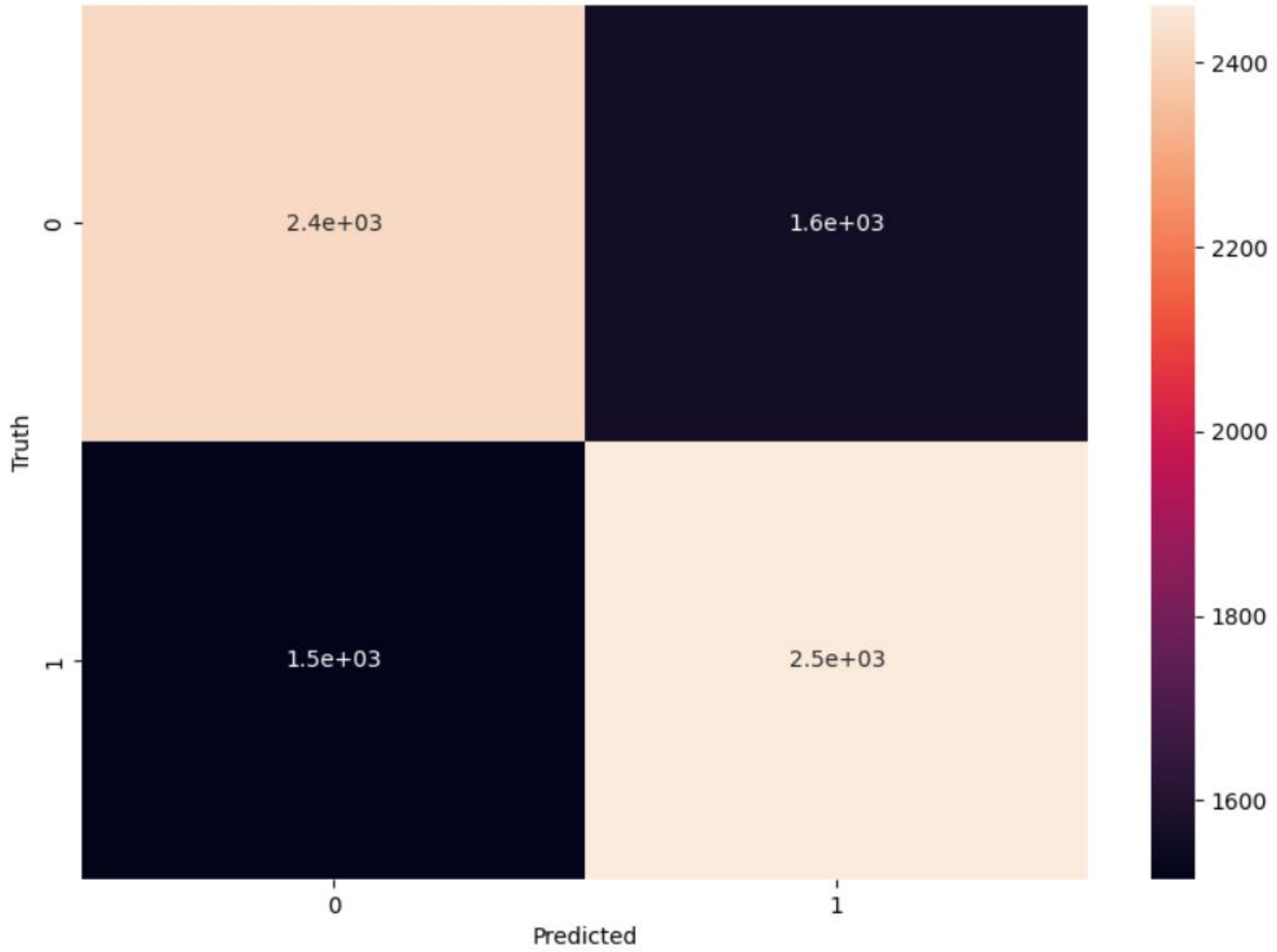
```
Accuracy score of the training data :  0.6118258391652349
Accuracy score of the test data :  0.613073538654934
```

CONFUSION MATRIX:

```
y_pred=model.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,y_pred)
cm
```

```
array([[2415, 1563],
       [1515, 2462]])
```

```
import seaborn as sn  # Import the Seaborn library and alias it as `sn`
import matplotlib.pyplot as plt  # Import the Matplotlib library and alias it as `plt`

plt.figure(figsize=(10, 7))  # Set the figure size to be 10x7 inches

# Create a heatmap of the confusion matrix `cm`
sn.heatmap(cm, annot=True)  # `annot=True` adds numerical annotations to the heatmap cells

plt.xlabel('Predicted')  # Label the x-axis as 'Predicted'
plt.ylabel('Truth')  # Label the y-axis as 'Truth'

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



4.2.2 THROUGH SVM

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x_res,y_res, test_size = 0.25,stratify=y_res,
random_state=2)
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
# Fit the scaler to the training data and transform it
scaler.fit(X_train)
X_train=scaler.transform(X_train)# Import the KNeighborsClassifier from scikit-learn
X_test=scaler.transform(X_test)# Create an instance of KNeighborsClassifier with 5 neighbors
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversion
  return self._fit(X, y)
KNeighborsClassifier
KNeighborsClassifier()
```

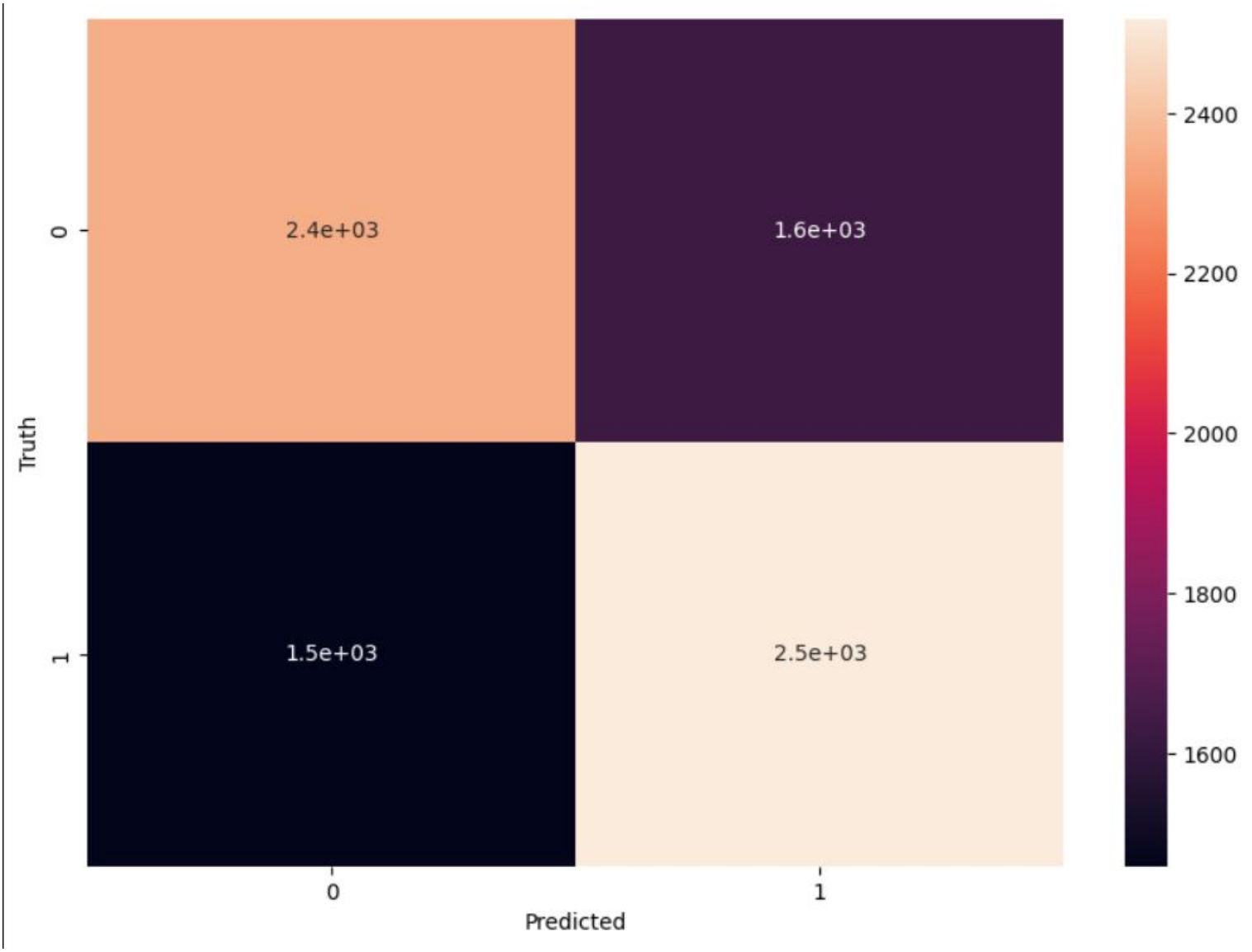
```
from sklearn.metrics import classification_report
y_pred=classifier.predict(X_test)# Predict the target variable for the test set
print(classification_report(Y_test,y_pred))
print("accuracy score in knn is:",accuracy_score(Y_test,y_pred))
# Generate a classification report
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
              precision    recall  f1-score   support

           0       0.62      0.59      0.60      3978
           1       0.61      0.63      0.62      3977

    accuracy                           0.61      7955
   macro avg       0.61      0.61      0.61      7955
weighted avg       0.61      0.61      0.61      7955

accuracy score in knn is: 0.6119421747328724
Accuracy score of the test data :  0.613073538654934
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,y_pred)
cm
```

```
array([[2350, 1628],
       [1459, 2518]])
```

```
import seaborn as sn
import seaborn as sn  # Import the Seaborn library and alias it as `sn`
import matplotlib.pyplot as plt  # Import the Matplotlib library and alias it as `plt`

plt.figure(figsize=(10, 7))  # Set the figure size to be 10x7 inches

# Create a heatmap of the confusion matrix `cm`
sn.heatmap(cm, annot=True)  # `annot=True` adds numerical annotations to the heatmap cells

plt.xlabel('Predicted')  # Label the x-axis as 'Predicted'
plt.ylabel('Truth')  # Label the y-axis as 'Truth'

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




```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x_res,y_res, test_size = 0.25,stratify=y_res,
random_state=2)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning:
  y = column_or_1d(y, warn=True)
▾ SVC
SVC()
```

```
from sklearn import metrics
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

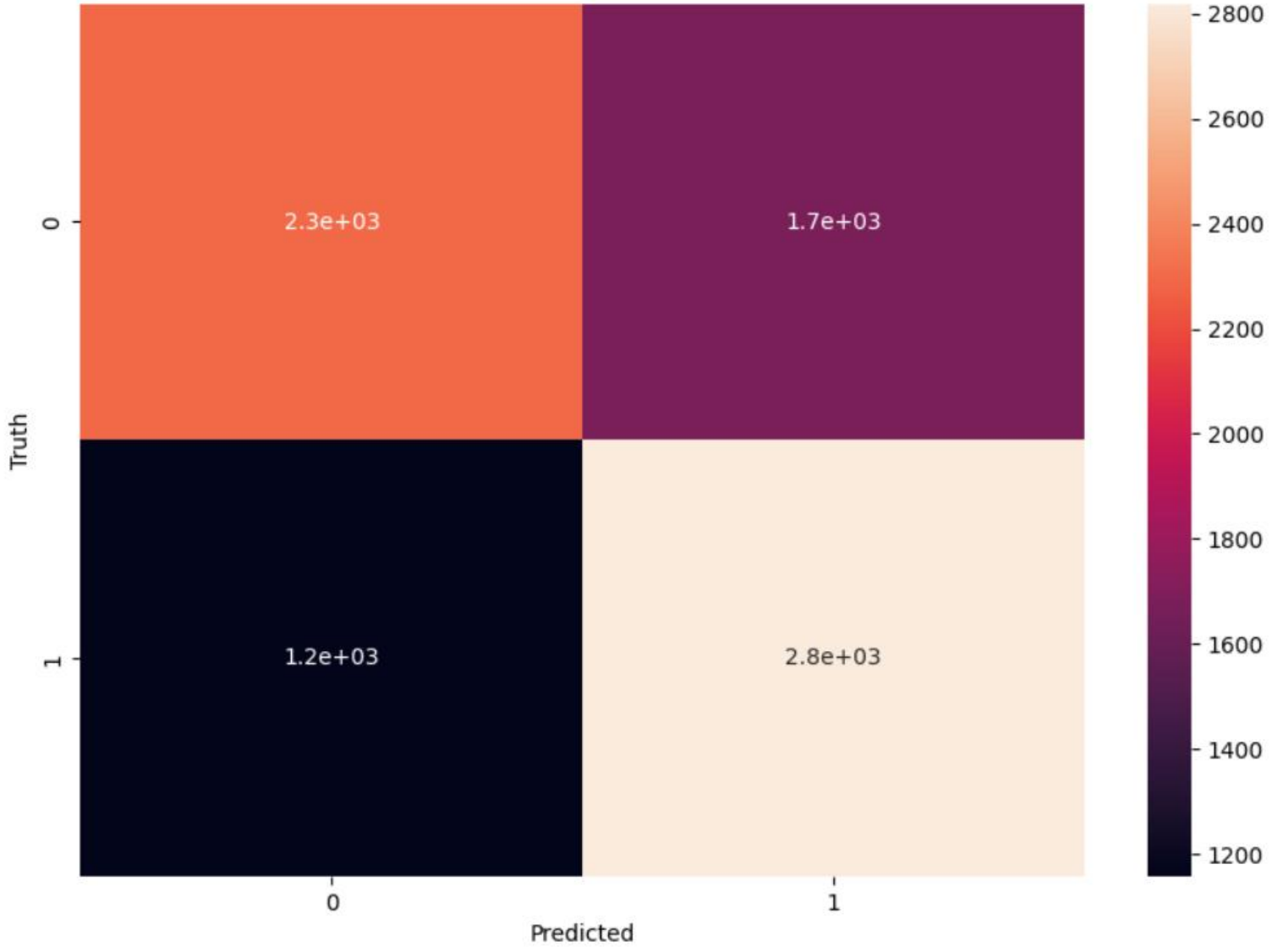
```
Accuracy: 0.6432432432432432
```

CONFUSION MATRIX:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,y_pred)
cm
```

```
array([[2299, 1679],
       [1159, 2818]])
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')  # Calculate the accuracy score of the model on the test data
```



4.2.3 THROUGH DECISION TREE:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train, X_test, Y_train, Y_test = train_test_split(x_res,y_res, test_size = 0.25,stratify=y_res,
random_state=2)
# Create an instance of the DecisionTreeClassifier
d=DecisionTreeClassifier()
d=d.fit(X_train,Y_train.values.reshape(-1,))# Train (fit) the model on the training data
Y_pred=d.predict(X_test)
print('accuracy of training data:',metrics.accuracy_score(Y_test,Y_pred))
# Assuming X_test_prediction is defined earlier
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
accuracy of training data: 0.6325581395348837
Accuracy score of the test data :  0.613073538654934
```

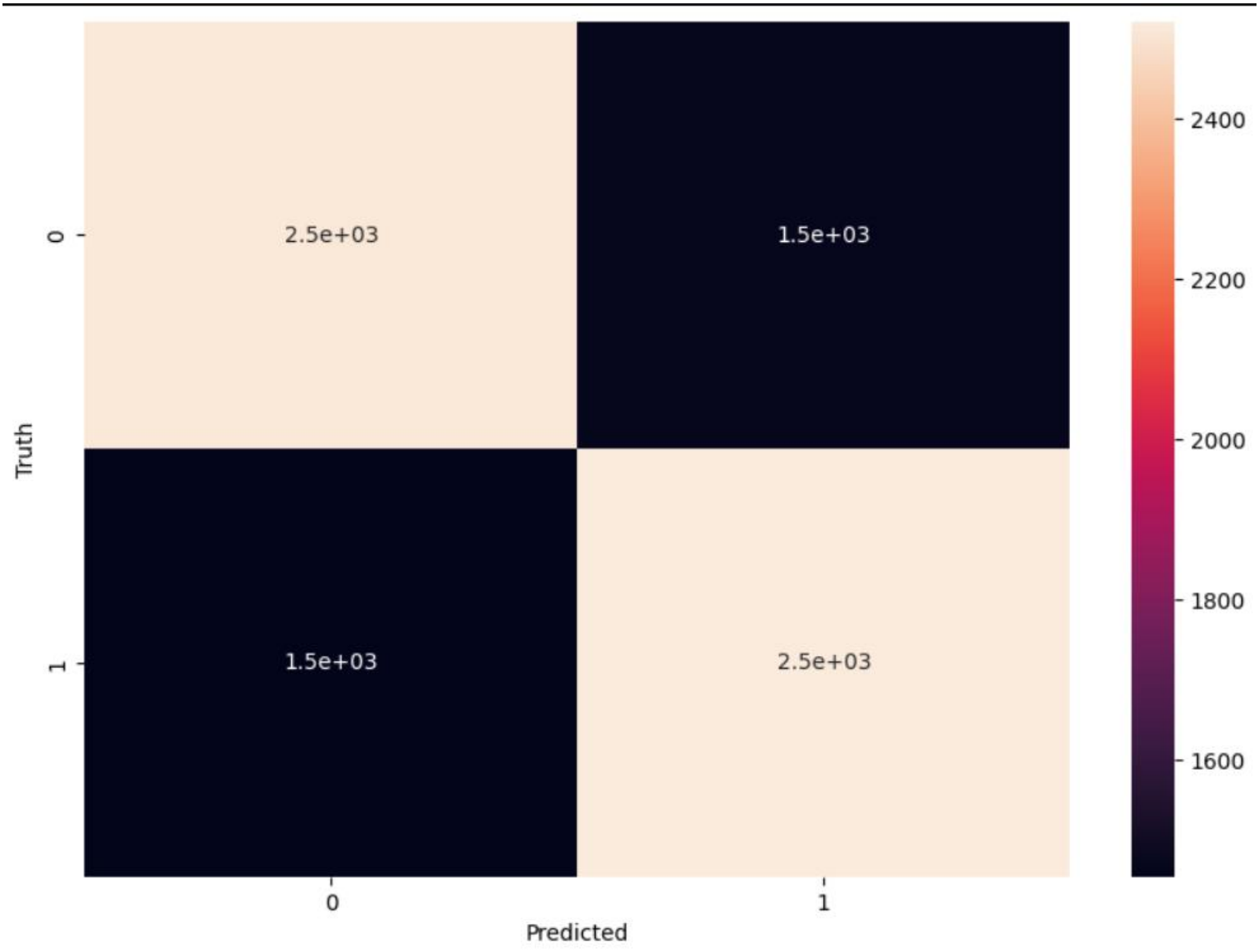
CONFUSION MATRIX:

from sklearn.metrics import confusion_matrix cm = confusion_matrix(Y_test,Y_pred)

cm

```
array([[2510, 1468],
       [1455, 2522]])
```

```
import seaborn as sn  # Import the Seaborn library and alias it as `sn`
import matplotlib.pyplot as plt  # Import the Matplotlib library and alias it as `plt`
plt.figure(figsize=(10, 7))  # Set the figure size to be 10x7 inches
sn.heatmap(cm, annot=True)  # `annot=True` adds numerical annotations to the heatmap cells
plt.xlabel('Predicted')  # Label the x-axis as 'Predicted'
plt.ylabel('Truth')  # Label the y-axis as 'Truth'
plt.show()  # Display the plot
```



```
from imblearn.over_sampling import RandomOverSampler

ros=RandomOverSampler(sampling_strategy="not majority")

# Apply RandomOverSampler to resample the data

x_res,y_res=ros.fit_resample(X,Y)

from sklearn.tree import DecisionTreeClassifier from

sklearn.model_selection import train_test_split from sklearn import

metrics

X_train, X_test, Y_train, Y_test = train_test_split(x_res,y_res, test_size = 0.25,stratify=y_res,
random_state=2)
d=DecisionTreeClassifier()
# Train (fit) the model on the training data
d=d.fit(X_train,Y_train.values.reshape(-1,)) Y_pred=d.predict(X_test)
# Print the accuracy on the test data
print('accuracy of training data:',metrics.accuracy_score(Y_test,Y_pred))
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy score of the test data : ', test_data_accuracy)
```

OUTPUT:

accuracy of training data: 0.6324324324324324

Accuracy score of the test data : 0.6203645505971087

CONFUSION MATRIX:

```
from sklearn.metrics import confusion_matrix

# Calculate theconfusion matrix

cm = confusion_matrix(Y_test,Y_pred)

cm
```

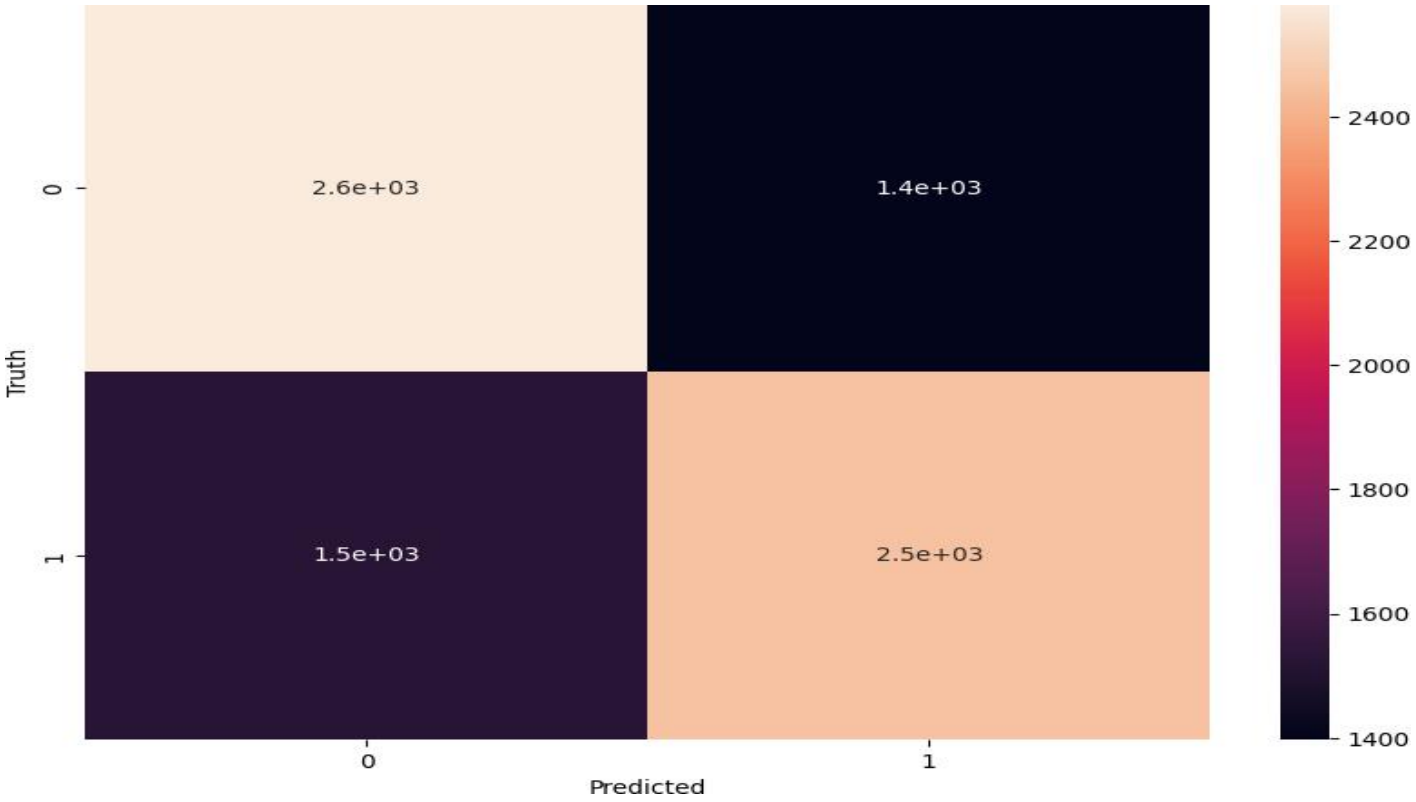
OUTPUT:

```
array([[2580, 1398],
       [1526, 2451]
```

```
import seaborn as sn  # Import the Seaborn library and alias it as `sn`
import matplotlib.pyplot as plt  # Import the Matplotlib library and alias it as `plt`
```

```
plt.figure(figsize=(10, 7))  # Set the figure size to be 10x7 inches

# Create a heatmap of the confusion matrix `cm`
sn.heatmap(cm, annot=True)  # `annot=True` adds numerical annotations to the heatmap cells

plt.xlabel('Predicted')  # Label the x-axis as 'Predicted'
plt.ylabel('Truth')  # Label the y-axis as 'Truth'

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

OUTPUT:



5. RESULT

MACHINE LEARNING ALGORITHMS	ACCURACY BEFORE BALANCING THE DATA SET	ACCURACY AFTER BALANCING THE DATA SET
LOGISTIC REGRESSION	**Accuracy on the training data :** 0.66592 **Accuracy on the test data :** 0.66048	**Accuracy on the training data :** 0.6126639567531325 **Accuracy on the testdata :** 0.6144563167818982
K-NEAREST NEIGHBOR	**Accuracy on training data:** 0.64384 **Accuracy on the test data:** 0.66048	**Accuracy on training data:** 0.6100565681961031 **Accuracy on the test data:** 0.6144563167818982
SUPPORT VECTOR MACHINE	_____	**Accuracy on training data:** 0.6408548082966687 **Accuracy on the test data:** 0.8331307674994373
DECISION TREE	_____	**Accuracy on training data:** 0.61456316781898 **Accuracy on the test data:** 0.6203645505971087

6.CONCLUSION

However, in general, machine learning algorithms can be used to analyze and model large datasets to identify potential patterns and risk factors that may be associated with the occurrence and spread of diseases like monkeypox. These algorithms can help in the development of predictive models that can assist public health authorities in preparing for outbreaks and implementing appropriate response measures.

However, it's important to note that the accuracy of such models depends on the quality and quantity of the data used to train them, as well as the complexity of the underlying biological and environmental factors that contribute to the spread of the disease. Therefore, while machine learning can provide valuable insights and predictions, it should always be used in conjunction with other methods and expert knowledge in the field of public health.

7.FUTURE SCOPE

In future, new features from the fields can be gathered to get a perfect image of the crop damage using other machine learning algorithms and deep learning algorithms such as ANN or CNN to get more accurate predictions.

There is significant potential for the use of machine learning in predicting and mitigating the spread of monkeypox. Here are some potential future areas of development:

1. Early Detection: Machine learning algorithms can be trained to detect early signals of an outbreak based on patterns in data such as animal surveillance, social media, and news articles. This early detection can allow public health officials to act quickly and contain the spread of the disease.
2. Risk Factors Analysis: By analyzing historical data and environmental factors, machine learning algorithms can identify risk factors that contribute to the outbreak of monkeypox. This information can help public health officials to identify high-risk areas and populations and implement targeted interventions.
3. Transmission Modeling: Machine learning algorithms can be used to model the transmission of the disease and predict the potential impact of different intervention strategies. This information can help public health officials to develop effective response plans and mitigate the spread of the disease.

Overall, the future of monkeypox prediction using machine learning is promising, and it has the potential to significantly improve public health outcomes. However, continued research and collaboration between public health officials, researchers, and machine learning experts will be necessary to fully realize this potential.

8.REFERENCES

[1] https://ieeexplore.ieee.org/abstract/document/9221459

[2] https://ieeexplore.ieee.org/abstract/document/9418375

[3] https://dl.acm.org/doi/abs/10.1145/3372454.3372474

[4] https://ieeexplore.ieee.org/abstract/document/9311735

[5] https://iopscience.iop.org/article/10.1088/17426596/1767/1/012026/meta

[6] https://ieeexplore.ieee.org/abstract/document/9221459

[7] https://ieeexplore.ieee.org/abstract/document/9033611

[8] https://ieeexplore.ieee.org/abstract/document/9137868

[9] https://ieeexplore.ieee.org/abstract/document/9154036

[10] Model by Yao et al. (2021)

[11] Model by Chen et al. (2020)

[12] Model by Gomes et al. (2019)

[13] Model by Amorim et al. (2018)

[14] Common Environmental Factors

[15] Additional Model based on Daily Confirmed Cases

LAB RECORD

LAB -1

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

```
import numpy as np
import pandas as pd
# Now you can use numpy and pandas functions and classes in your code

# Create a list of numbers
lst1 = [26, 44, 36, 78, 98]

# Convert the list into a numpy array
array1 = np.array(lst1)

# Print the resulting numpy array
print("Resulting numpy array:", array1)
```

```
Resulting numpy array: [26 44 36 78 98]
```

```
# Create a series of 11 zeroes
print("A series of zeroes:", np.zeros(11))
# 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(-1, 22))
# Create numbers starting from 0 up to 60 (excluding 60), spaced apart by 5
print("Numbers spaced apart by 5:", np.arange(0, 60, 5))
# 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 50 down to 0 (inclusive), in reverse order
print("Every 5th number from 50 in reverse order: ", np.arange(50, -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. 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: [-1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
Numbers spaced apart by 5: [ 0  5 10 15 20 25 30 35 40 45 50 55]
Numbers spaced apart by float: [ 0.   2.5  5.   7.5 10. ]
Every 5th number from 50 in reverse order:  [50 45 40 35 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	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
# Use the `read_csv` function from pandas to read a CSV file.
# The file path is "/content/name_marks.csv"
students = pd.read_csv("/content/name_marks.csv")
# Print the summary statistics
students
```

0	Violette Berger	95
1	Byron Rasmussen	86
2	Esperanza White	92
3	Aiden Graves	89
4	Elle Nielsen	94
5	Tru Crosby	79
6	Keily Mosley	72
7	Rayden Houston	98

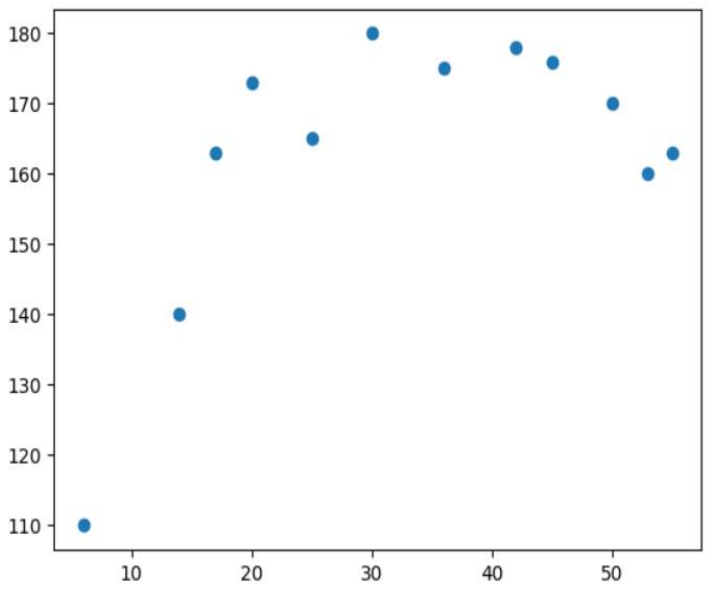
```
# Use the `read_excel` function from pandas to read an Excel file.
# The file path is "/content/name_height_weight.xlsx"
datatxt = pd.read_excel("/content/name_height_weight.xlsx")
# Print the summary statistics
datatxt
```

	Name	Height	Weight
0	Kimber Webster	169	68
1	Shawn Wade	172	79
2	Evie Schmidt	158	66
3	Zayden Cohen	180	78
4	Destiny Corona	167	82
5	Darian Schmitt	178	65
6	Queen Travis	184	70
7	Willie Chang	167	76

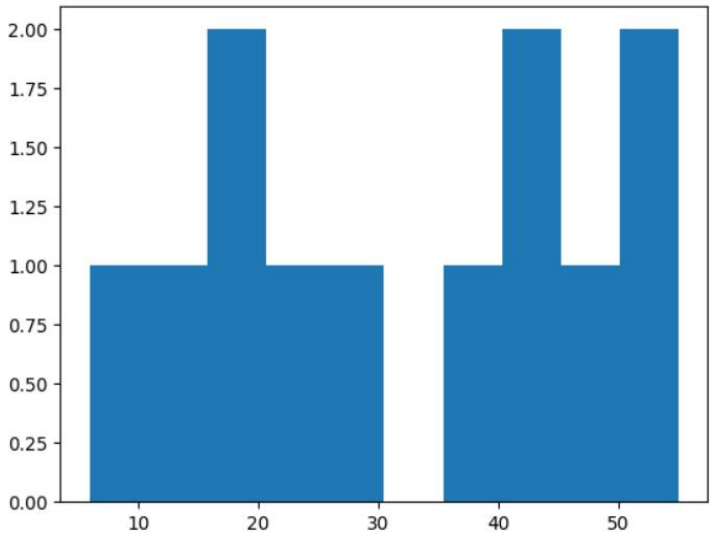
```
list_of_df = pd.read_html("https://en.wikipedia.org/wiki/2016_Summer_Olympics_medal_table",header=0)
medals=list_of_df[0]
medals.head()
```

	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

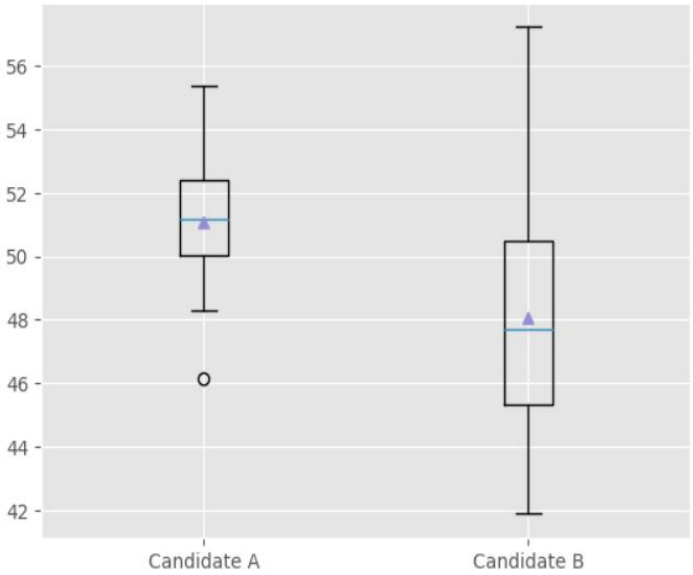
```
# Updated list of names
people = ['Alice','Bob','Charlie','Diana','Ella','Felix',
          'Grace','Hank','Ivy','Jack','Kate','Leo']

# Updated list of ages corresponding to the names
age = [25,14,36,50,42,20,30,55,6,45,53,17]

# Updated list of weights corresponding to the names
weight = [60,40,82,73,75,65,77,74,20,70,90,55]

# Updated list of heights corresponding to the names
height = [165,140,175,170,178,173,180,163,110,176,160,163]

# The lists have been updated with new names and corresponding values for age, weight, and height.
# Each list now represents different individuals with their respective attributes.

# These lists can be used for further analysis or processing as needed.
plt.scatter(age, height)
plt.show()
```




```
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()
```



LAB-2

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

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

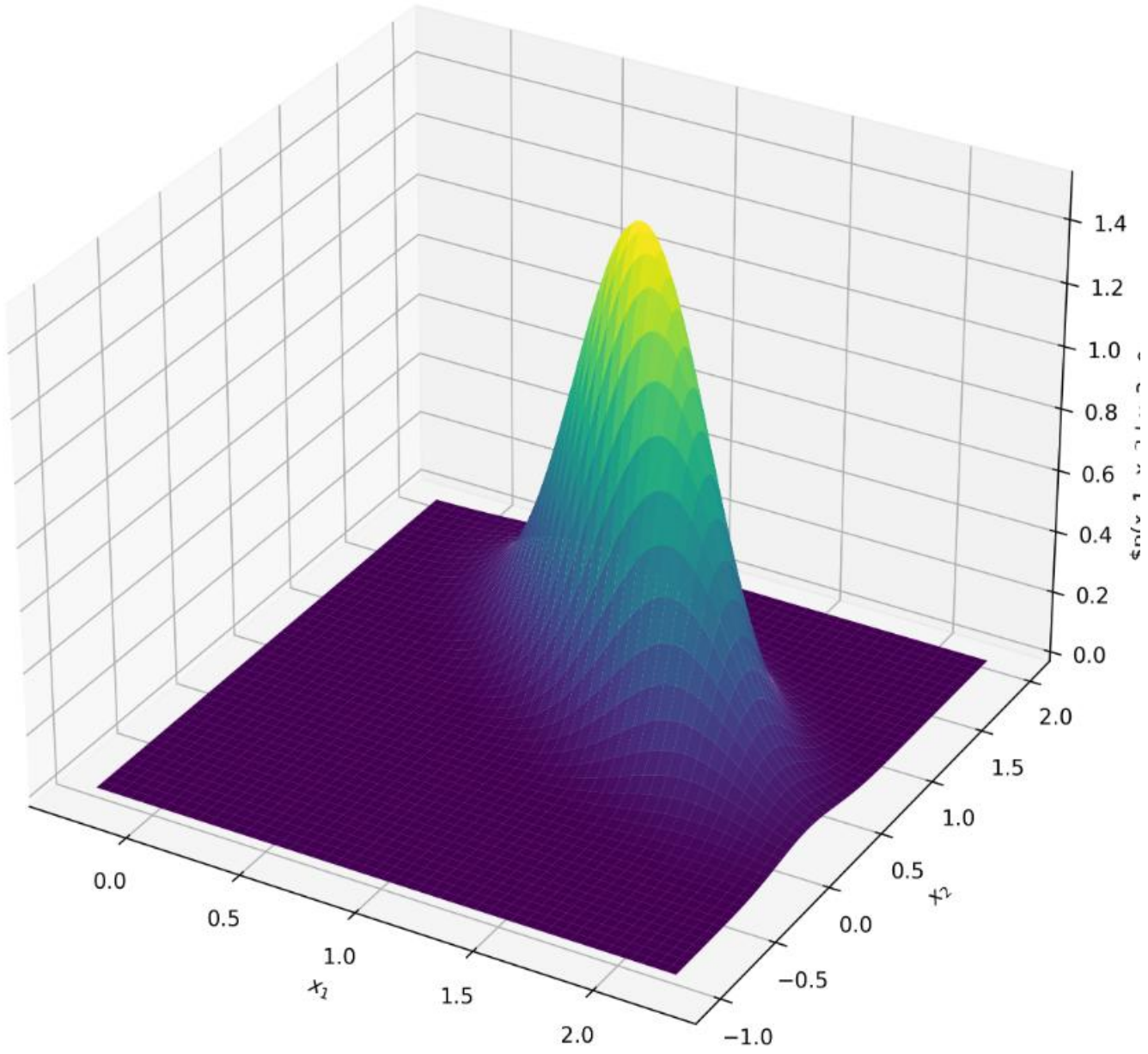
```
covariance = np.array([[0.14, -0.3, 0.0, 0.2],
                       [-0.3, 1.16, 0.2, -0.8],
                       [0.0, 0.2, 1.0, 1.0],
                       [0.2, -0.8, 1.0, 2.0]])
precision = np.linalg.inv(covariance)
print(precision)
```

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

```
mu_t = generate_pair()
print(mu_t)
```

```
[1.27944661 0.51474023]
```

```
x, y = np.mgrid[-0.25:2.25:.01, -1:2:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x
pos[:, :, 1] = y
mu_p = [0.8, 0.8]
cov_p = [[0.1, -0.1], [-0.1, 0.12]]
z = multivariate_normal(mu_p, cov_p).pdf(pos)

fig = plt.figure(figsize=(10, 10), dpi=300)
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x, y, z, cmap=plt.cm.viridis)
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)
plt.show()
```




```
# Define the number of data points 'N'.
N=1000
# Import the NumPy library.
import numpy as np
data = np.random.multivariate_normal([0.28, 1.18], [[2.0, 0.8], [0.8, 4.0]], N)
data
np.savetxt('data.txt',data)
# Import the pandas library and alias it as 'pd' for convenience.
import pandas as pd
data = pd.read_table('data.txt')
data = np.loadtxt('data.txt')
data
```

```
array([[-1.91742873, -0.56334336],
       [ 1.70314315,  2.20691542],
       [ 1.39564507, -0.09373488],
       ...,
       [-0.82748989,  2.88814348],
       [ 1.18601727,  3.51825517],
       [-0.83485172, -0.3278072 ]])
```

```
def seq_ml(data):
  mu_ml = data.mean(axis=0)
x = data - mu_ml
cov_ml = np.dot(x.T, x) / N
cov_ml_unbiased = np.dot(x.T, x) / (N - 1)
print(mu_ml)
print(cov_ml)
print(cov_ml_unbiased)
```

```
[0.31745231 1.2661432 ]
[[1.87303912 0.70073528]
 [0.70073528 4.28243008]]
[[1.87491403 0.70143672]
 [0.70143672 4.2867168 ]]
```

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

def seq_ml(data):
   mus = [np.array([[0], [0]])]
# Loop through the data points to calculate the sample mean vectors iteratively.
 for i in range(N):
       x_n = data[i].reshape(2, 1)
       mu_n = mus[-1] + (x_n-mus[-1]) / (i + 1)
       mus.append(mu_n)
   return mus

# Calculate a sequence of sample mean vectors using the 'seq_ml' function.
mus_ml = seq_ml(data)
print(mus_ml[-1])
```

```
[[0.31745231]
 [1.2661432 ]]
```

```
# Define the true mean vector 'mu_p' for a multivariate distribution.
mu_p = np.array([[0.28], [1.18]])
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 the updated values of 'mu_p', 'cov_p', and 'cov_t'.
print(mu_p,cov_p,cov_t)
```

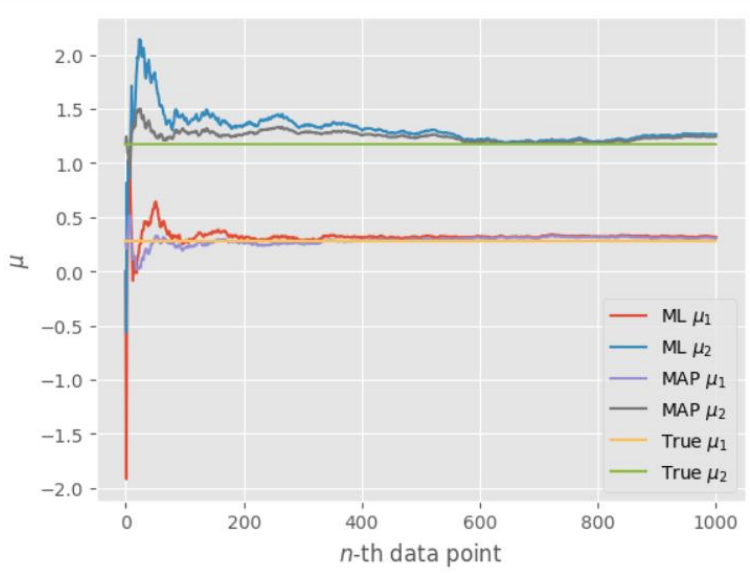
```
[[0.28]
 [1.18]] [[ 0.1  -0.1 ]
 [-0.1   0.12]] [[2.  0.8]
 [0.8 4. ]]
```

```
# 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])
```

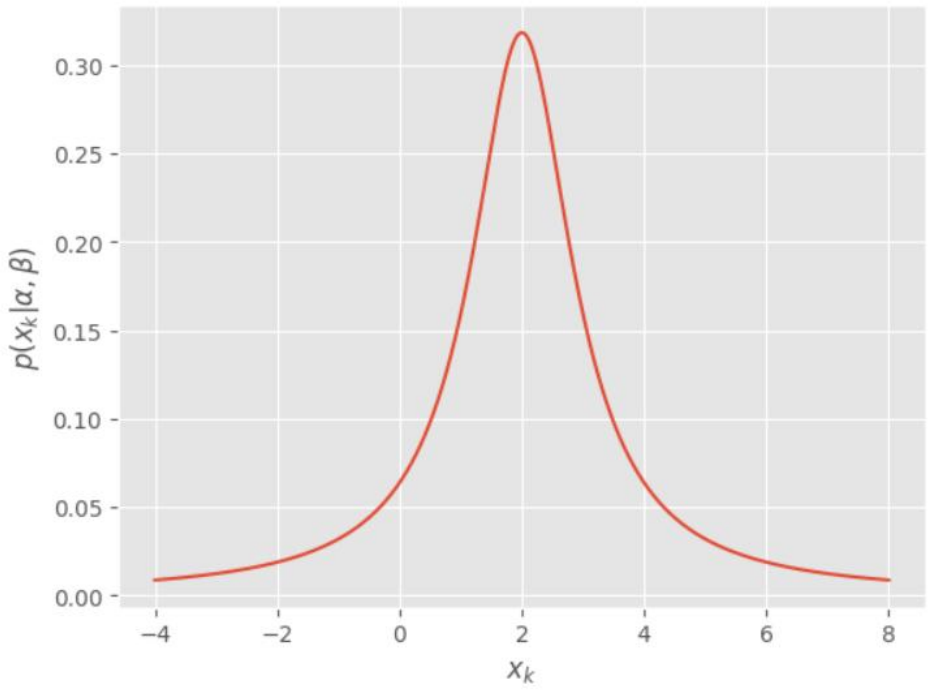
```
[[0.30449334]
 [1.24457165]]
```

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Assuming N is defined earlier in your code
# Create an array from 0 to N
X = np.arange(N+1)

# Extract the first element from each list in mus_ml and create a new list
mus1_ml = [mu[0] for mu in mus_ml]

# Extract the second element from each list in mus_ml and create a new list
mus2_ml = [mu[1] for mu in mus_ml]

# Similar to above, extract elements from mus_map
mus1_map = [mu[0] for mu in mus_map]
mus2_map = [mu[1] for mu in mus_map]
# Create two lists of the same length as X, both filled with 0.28 and 1.18 respectively
mus1_t = [0.28] * (N+1)
mus2_t = [1.18] * (N+1)
# Set the style of the plot to 'ggplot'
plt.style.use('ggplot')
# Plot the data from the lists on the same figure
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 $\mu_2$')
# Set the labels for the x and y axes
plt.xlabel('$n$-th data point')
plt.ylabel('$\mu$')
# Add a legend to the plot with labels
plt.legend(loc=4)
# Save the figure as 'seq_learning.png' with tight bounding box and high resolution
plt.savefig('seq_learning.png', bbox_inches='tight',
```



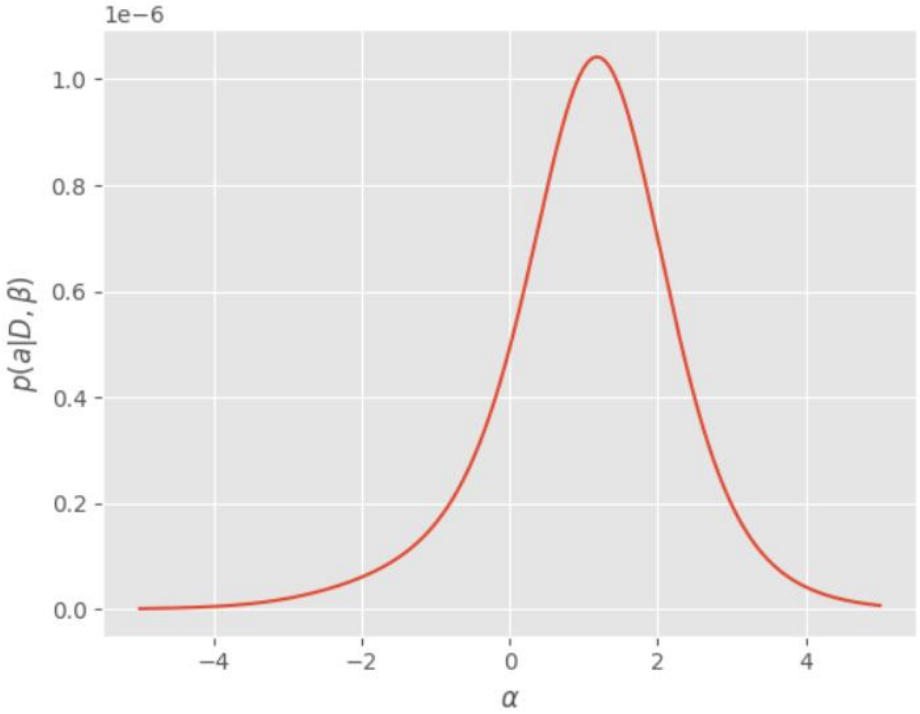

```
# Define a function 'p_xk' that calculates the probability density function (PDF) of a Cauchy distribution.
def p_xk(x, alpha, beta):
    return beta / (np.pi * (beta**2 + (x-alpha)**2))
# Create an array 'x' representing a range of values for which to calculate the PDF.
x = np.linspace(-4, 8, num=1000)
probs = p_xk(x, 2, 1)
plt.plot(x, probs)
plt.xlabel('$x_k$')
plt.ylabel(r'$p(x_k | \alpha, \beta)$')
plt.savefig('prob_xk.png', bbox_inches='tight', dpi=300)
plt.show()

# 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))
```

09

```
# Create an array 'D' containing observed data points.
D = np.array([4.8, -2.7, 2.2, 1.1, 0.8, -7.3])
alphas = np.linspace(-5, 5, num=1000)
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)
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)
plt.show()
```



Calculate and print the mean of the observed data points in array 'D'.

```
print(D.mean())
print(alphas[np.argmax(likelihoods)])
```

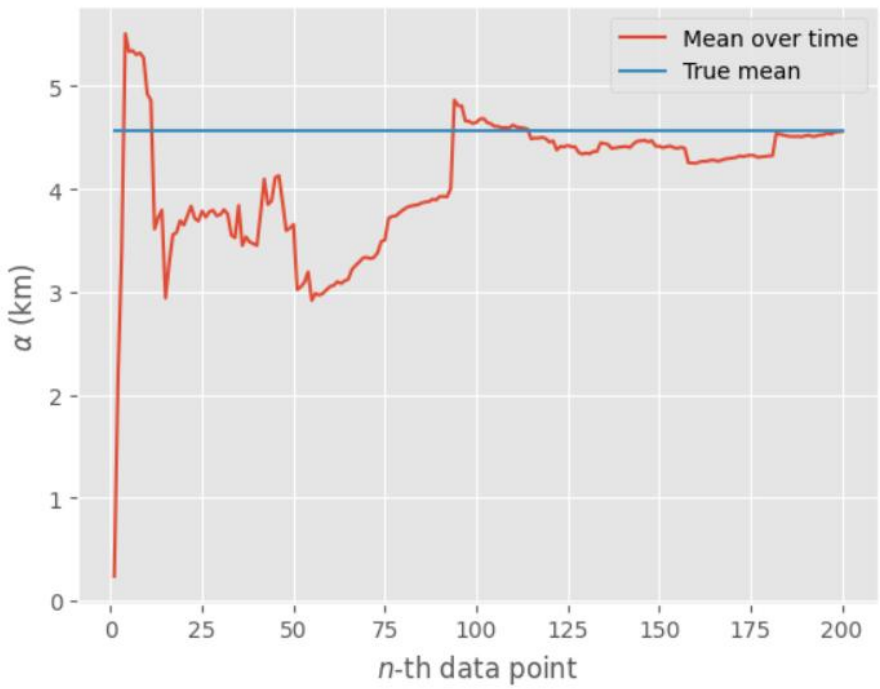
max_likelihood_alpha

```
-0.18333333333333326
1.1761761761761758
```

```
alpha_t = np.random.uniform(0, 10)
beta_t = np.random.uniform(1, 2)
print(alpha_t, beta_t)
```

```
4.8153399191022945 1.1209355590630299
```

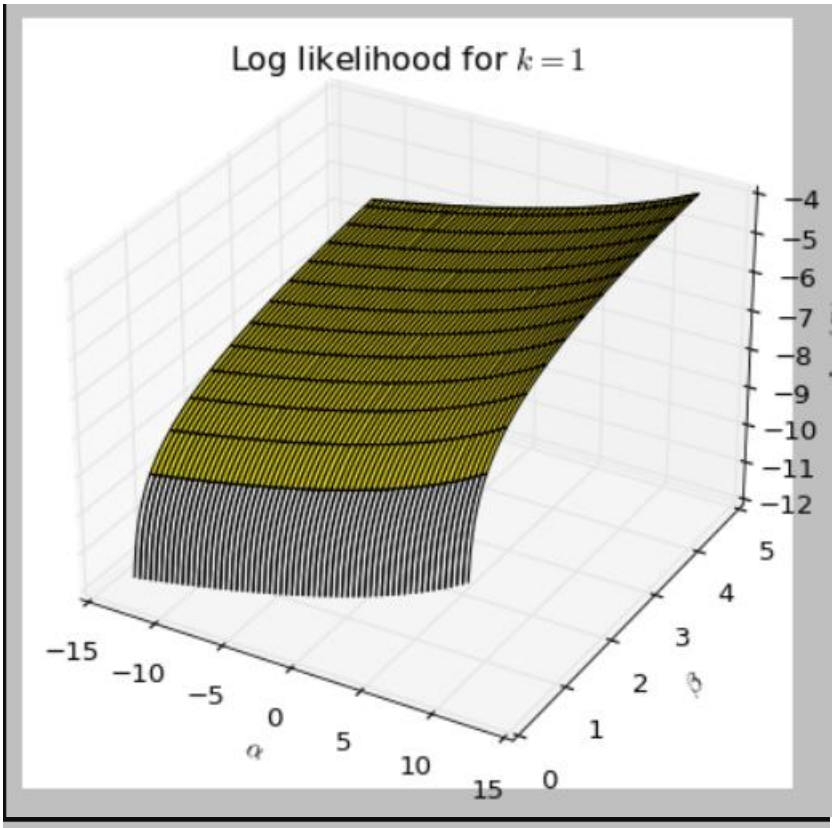
```
def location(angle, alpha, beta):
    return beta * np.tan(angle) + alpha

N = 200
angles = np.random.uniform(-np.pi/2, np.pi/2, N)
locations = np.array([location(angle, alpha_t, beta_t) for angle in angles])
mus = [locations[:i + 1].mean() for i in range(N)]
mean = [locations.mean()] * (N)
X = np.arange(1, N + 1)
plt.style.use('ggplot')
plt.plot(X, mus, label='Mean over time')
plt.plot(X, mean, label='True mean')
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()
```

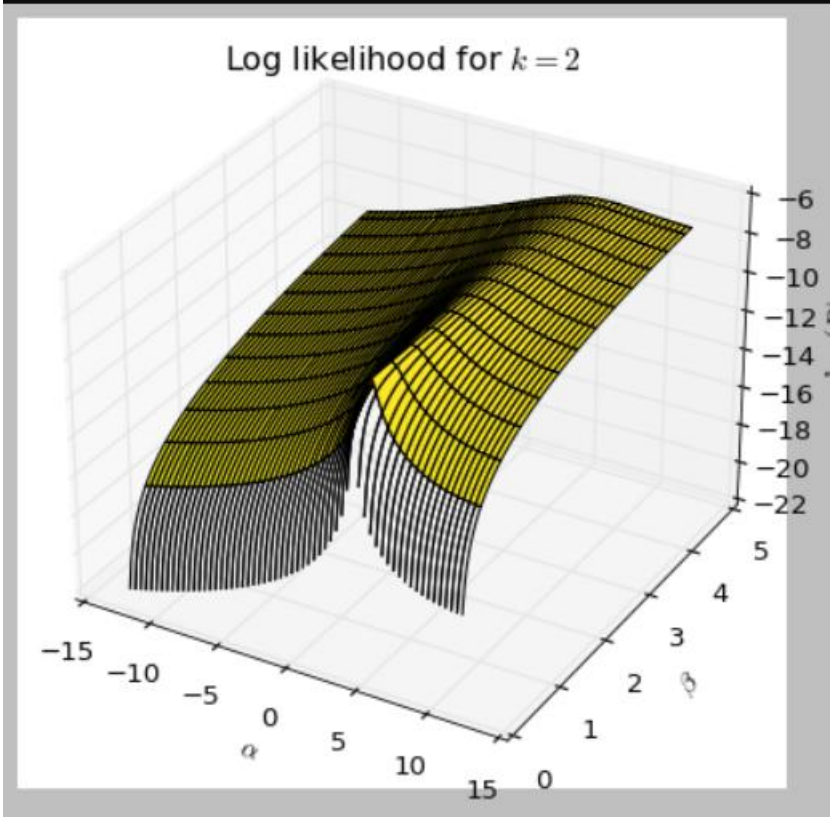


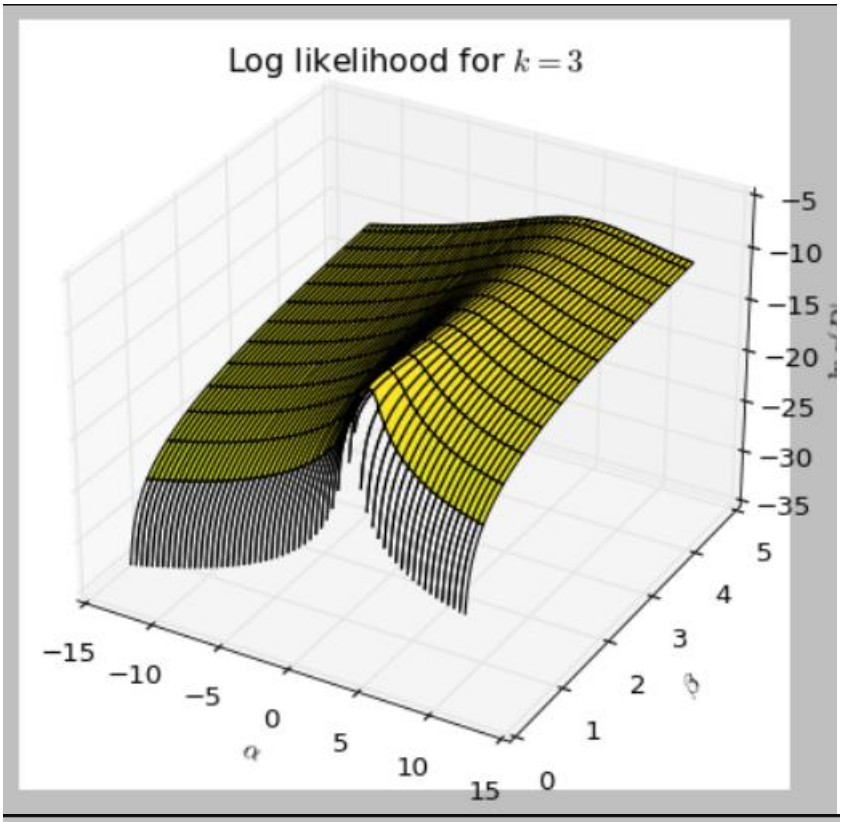

```
# Calculate and print the mean of the 'locations' array.
print(locations.mean())
```

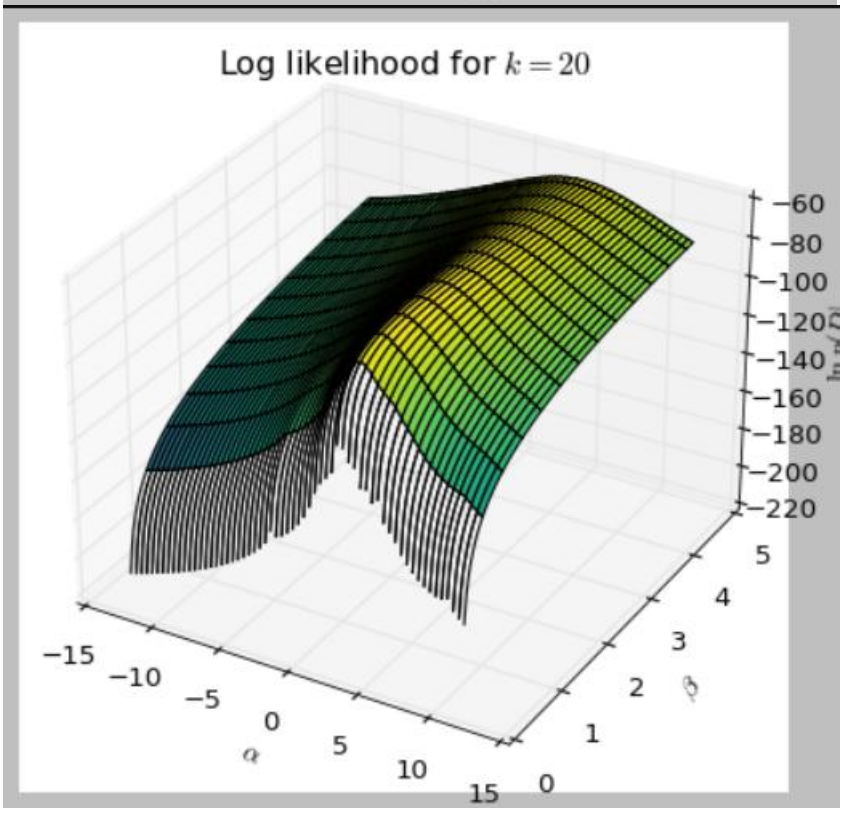
```
4.5594733266299805
```

```
# Set the plot style to 'classic'.
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]
# Loop through different values of 'k'.
likelihood = k * np.log(betas/np.pi)
    for loc in x:
   likelihood -= np.log(betas**2 + (loc - alphas)**2)

    # Create a new 3D plot.
   fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.plot_surface(alphas, betas, likelihood, cmap=plt.cm.viridis, vmin=-200, vmax=likelihood.max())
    plt.xlabel(r'$\alpha$')
    plt.ylabel(r'$\beta$')
    ax.set_zlabel('$\ln p(D | \alpha, \beta)$')
    plt.title('Log likelihood for $k = {}$'.format(k))
    plt.savefig('logl_{}.png'.format(k), bbox_inches='tight', dpi=300)
    plt.show()
```


Log likelihood for $k = 1$
−4
−5
−6
−7
−8
−9
−10
−11
−12
−15
−10
−5
0
5
10
15
a
0
1
2
3
4
5
b


Log likelihood for $k = 2$
−6
−8
−10
−12
−14
−16
−18
−20
−22
−15
−10
−5
0
5
10
15
a
0
1
2
3
4
5
b






```
from scipy.optimize import fmin

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
```

```
def plot_maximize_logl(data, alpha_t, beta_t):
    alphas, betas = [], []
    x = np.arange(len(data))
    for k in x:
        [alpha, beta] = fmin(log_likelihood, (0, 1), args=(data[:k],))
        alphas.append(alpha)
        betas.append(beta)

    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])
```

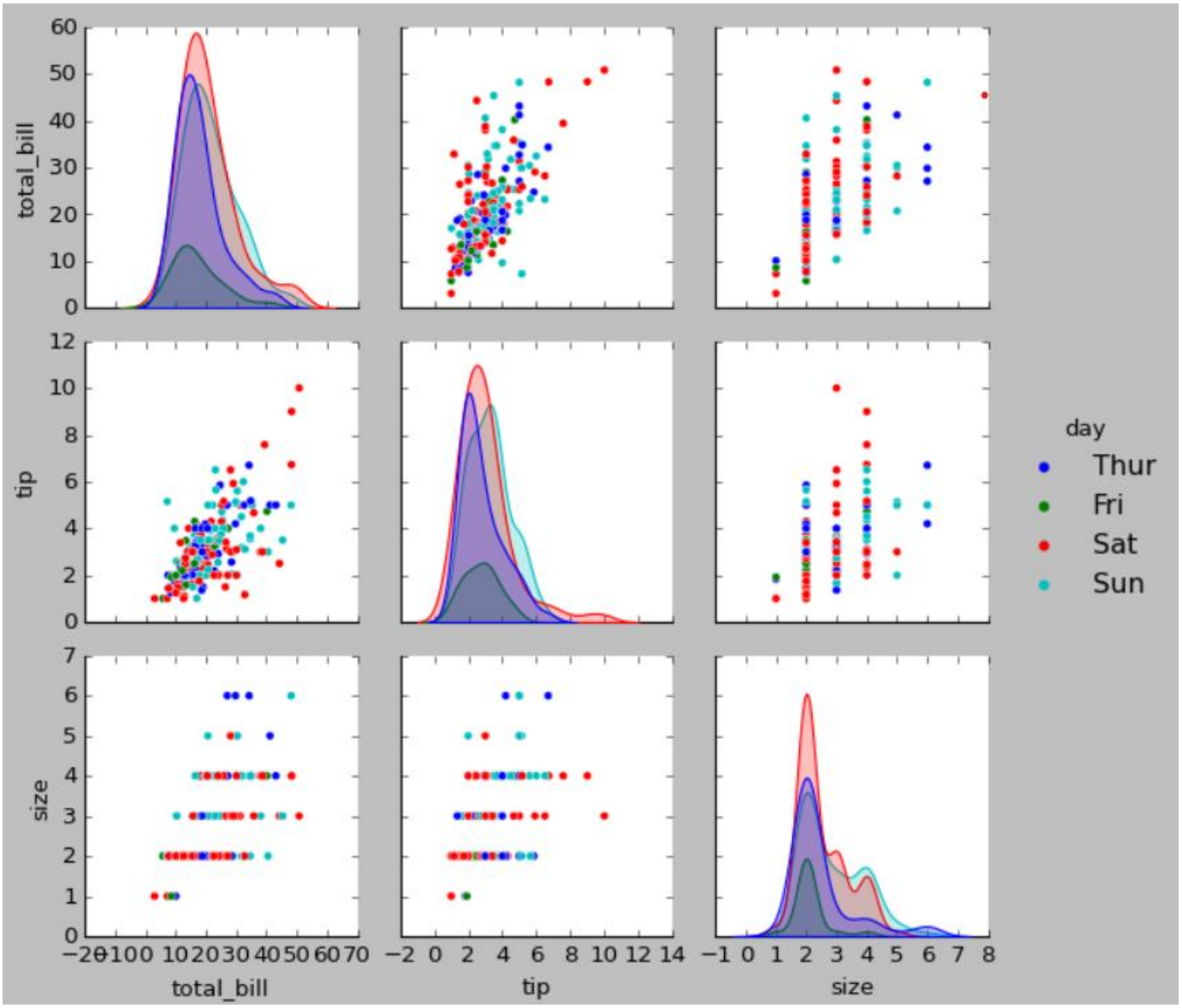
```
          id  year  hour  season  holiday  workingday  weather   temp   atemp  \
0          3  2012    23       3        0           0        2  23.78  27.275
1          4  2011     8       3        0           0        1  27.88  31.820
2          5  2012     2       1        0           1        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)
```

```
# 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.
tvar=np.var(t)#variance
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) #std.dev
```

```
# Calculate the squared difference between 'x' and the mean 'tm'.
f=(x-tm)**2 #mean
# Calculate a term 'g' using the variance.
g=2*(tvar**2) #variance
# 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()
```


total_bill
tip
size
day
Thur
Fri
Sat
Sun

LAB-3

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-1.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           4994 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                       4995 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

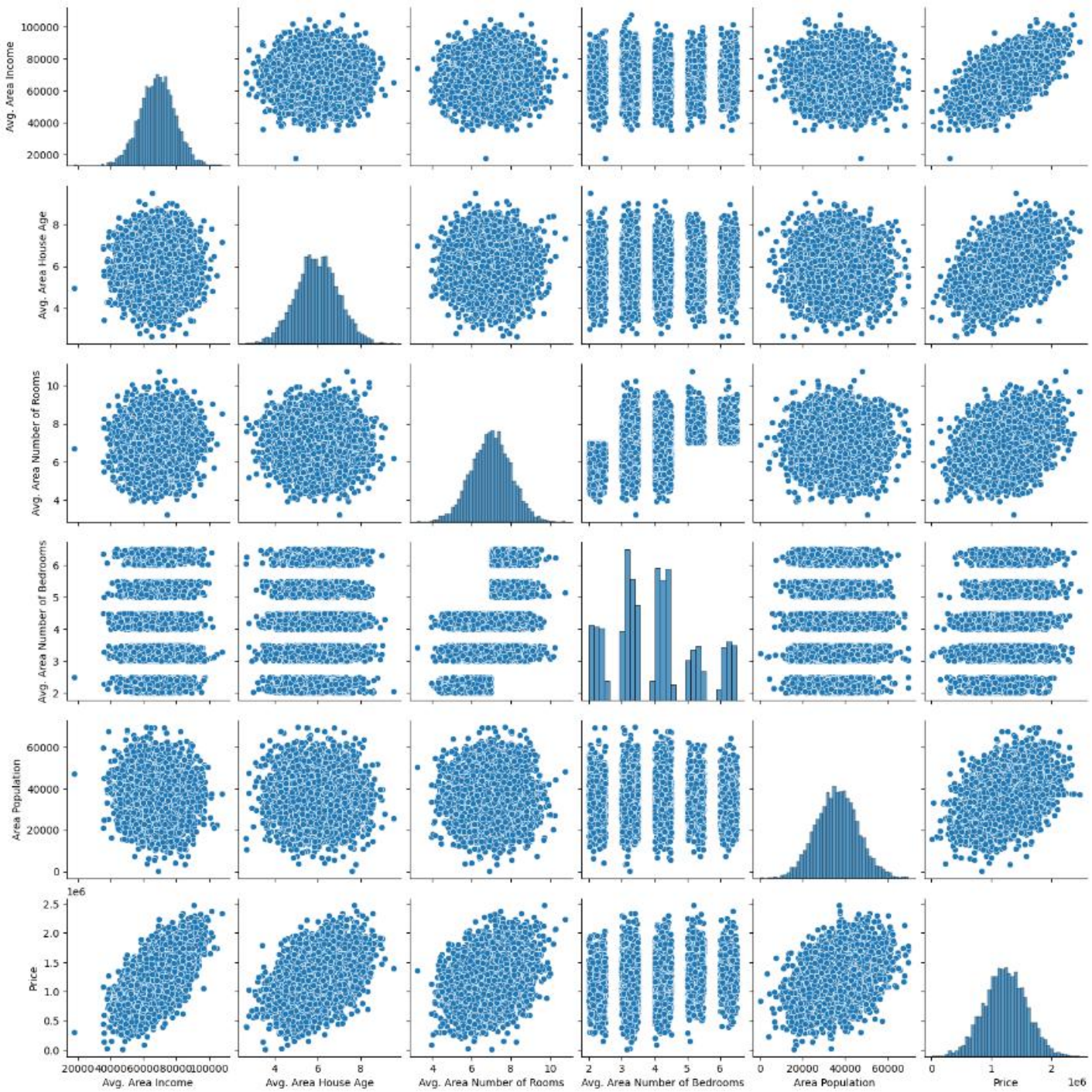
```
# Generate a statistical summary of the DataFrame 'df' with specific percentiles
df.describe(percentiles=[0.1,0.25,0.5,0.75,0.9])
```

	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
```

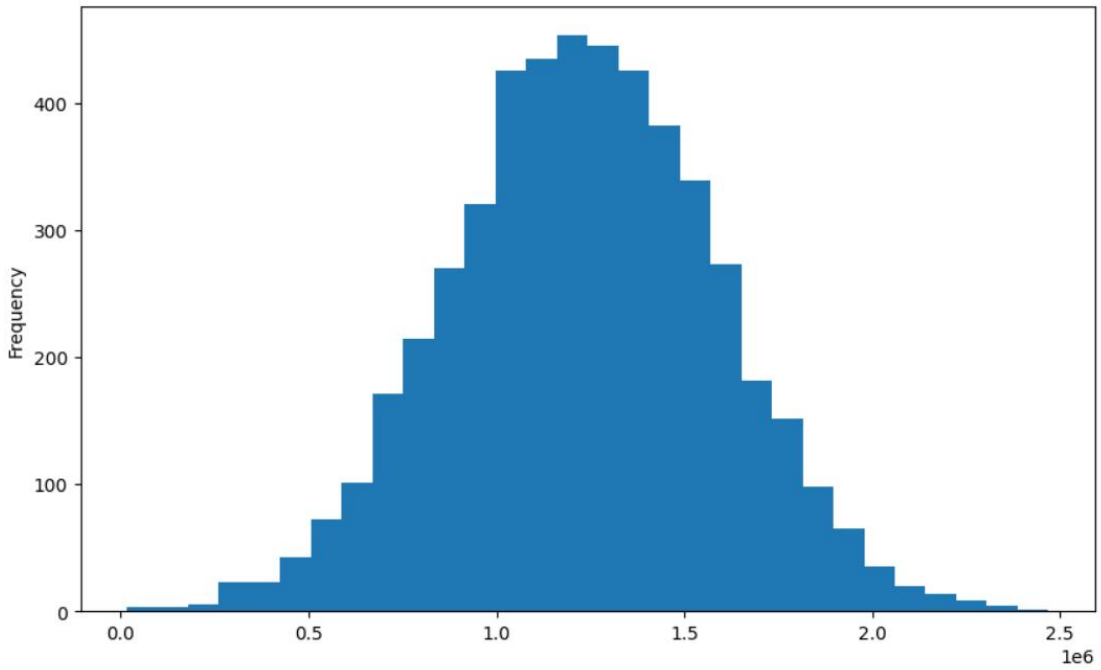
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

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



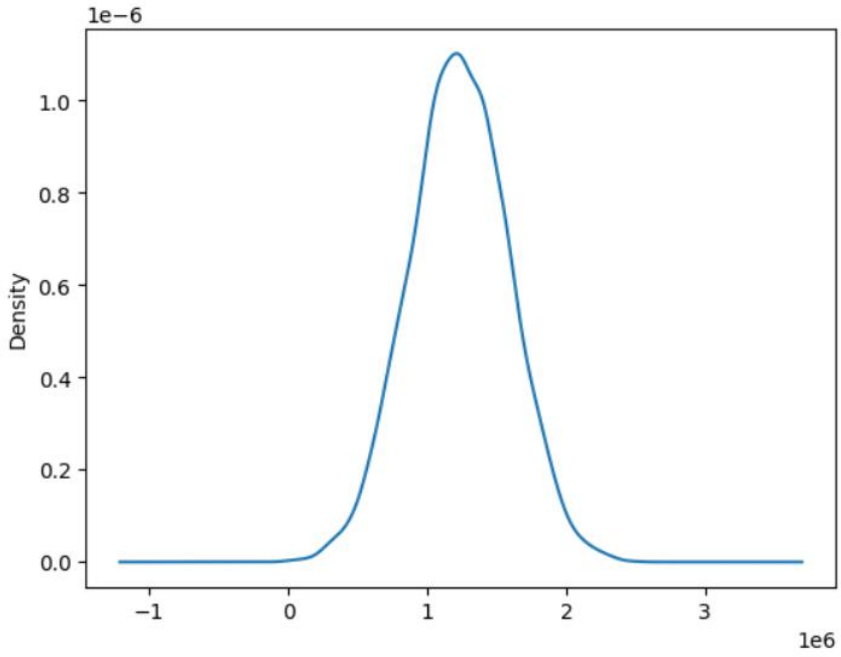

```
# Create a histogram plot for the 'Price' column with 25 bins and a specific figure size
df['Price'].plot.hist(bins=30, figsize=(10, 6))
```

```
<Axes: ylabel='Frequency'>
```




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

```
<Axes: ylabel='Density'>
```




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

```
<ipython-input-10-bdeb764252f8>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, i
  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=(11,8))

# 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=3)
```


```
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-3]]

# Create the target variable (y) by selecting the (len_feature-2)-th column from the DataFrame 'df'
y = df[l_column[len_feature-3]]

# 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, 4)
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
0	79545.45857	5.682861	7.009188	4.09
1	79248.64245	6.002900	6.730821	3.09
2	61287.06718	5.865890	8.512727	5.13
3	63345.24005	NaN	5.586729	3.26
4	59982.19723	5.040555	7.839388	4.23

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

```
0    23086.80050
1    40173.07217
2    36882.15940
3    34310.24283
4    26354.10947
Name: Area Population, 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=120)
# 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, 4)
Test feature set size: (1500, 4)
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)
```

```
LinearRegression
LinearRegression()
```

```
# 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
```

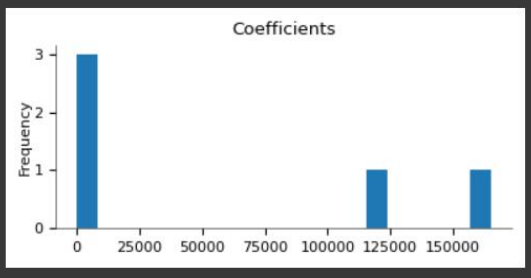
```
# Print the coefficients of the linear regression model stored in the 'lm' variable
print("The coefficients of the linear model:", lm.coef_)
```

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

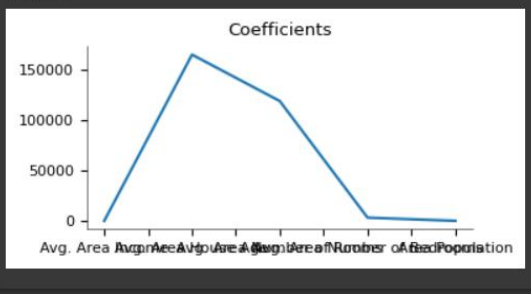
```
# Create a DataFrame 'cdf' to store the coefficients of the linear regression model
# The coefficients are associated with the feature names in 'X_train.columns'
cdf = pd.DataFrame(data=lm.coef_, index=X_train.columns, columns=["Coefficients"])
cdf
```

	Coefficients
Avg. Area Income	-0.020065
Avg. Area House Age	8.190938
Avg. Area Number of Rooms	50.709127
Avg. Area Number of Bedrooms	-105.791044

Distributions



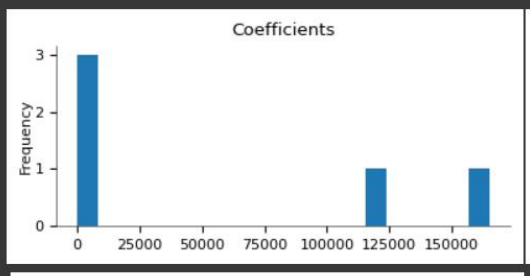
Values

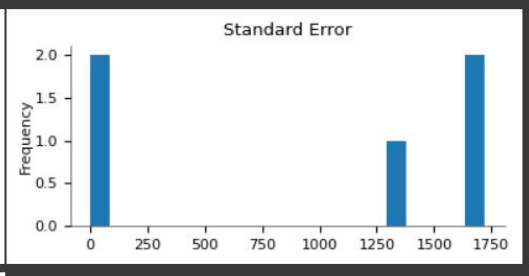


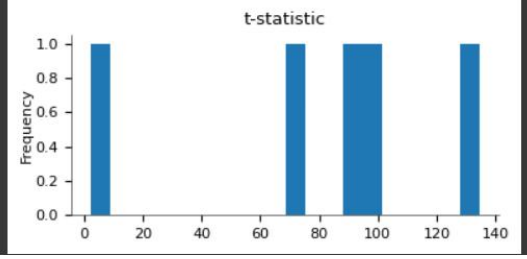
```
# Calculate the standard error for each coefficient
n = X_train.shape[0]  # Number of observations
k = X_train.shape[1]  # Number of features
dfN = n - k  # Degrees of freedom for the residuals

# 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]

# 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

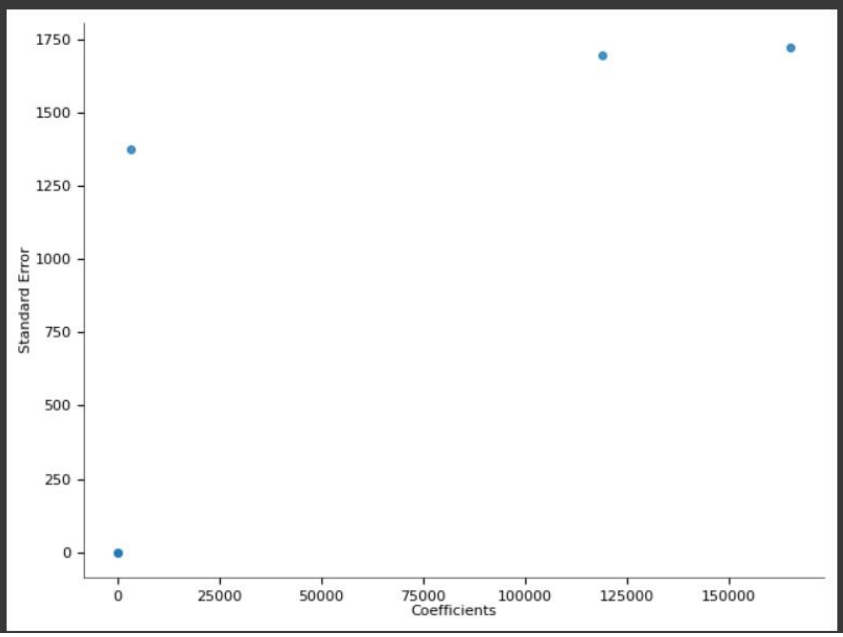
Distributions

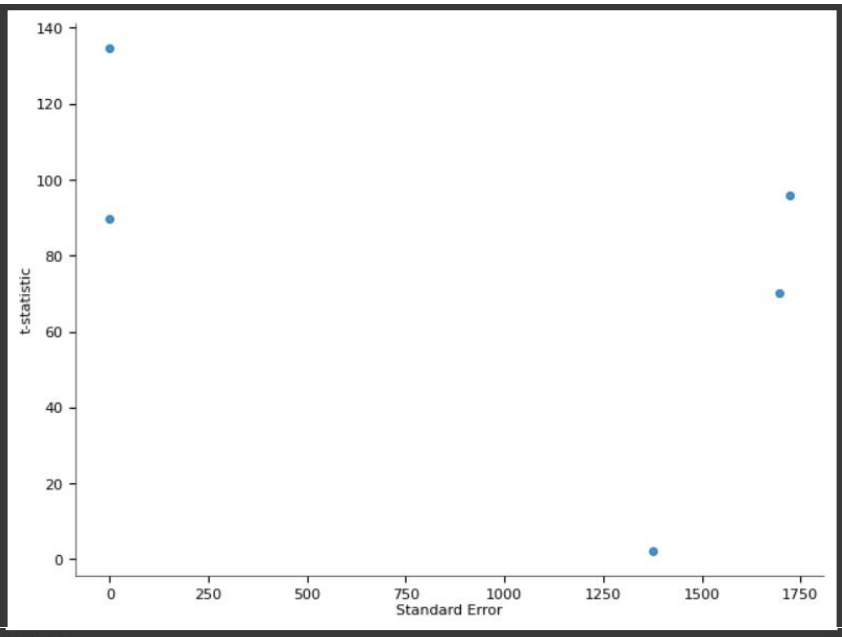




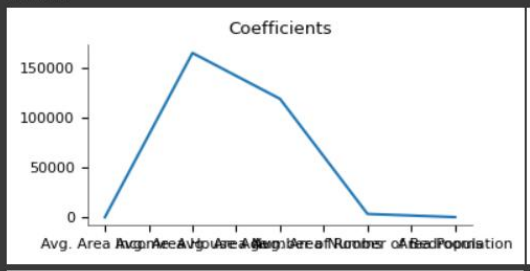


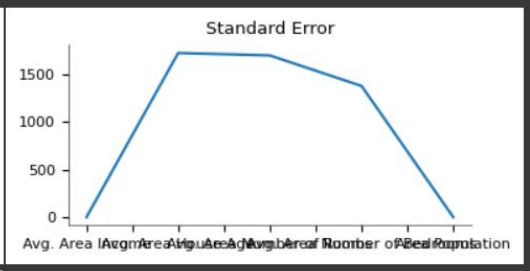
2-d distributions

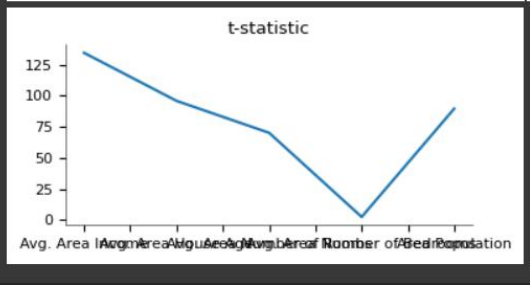




Values



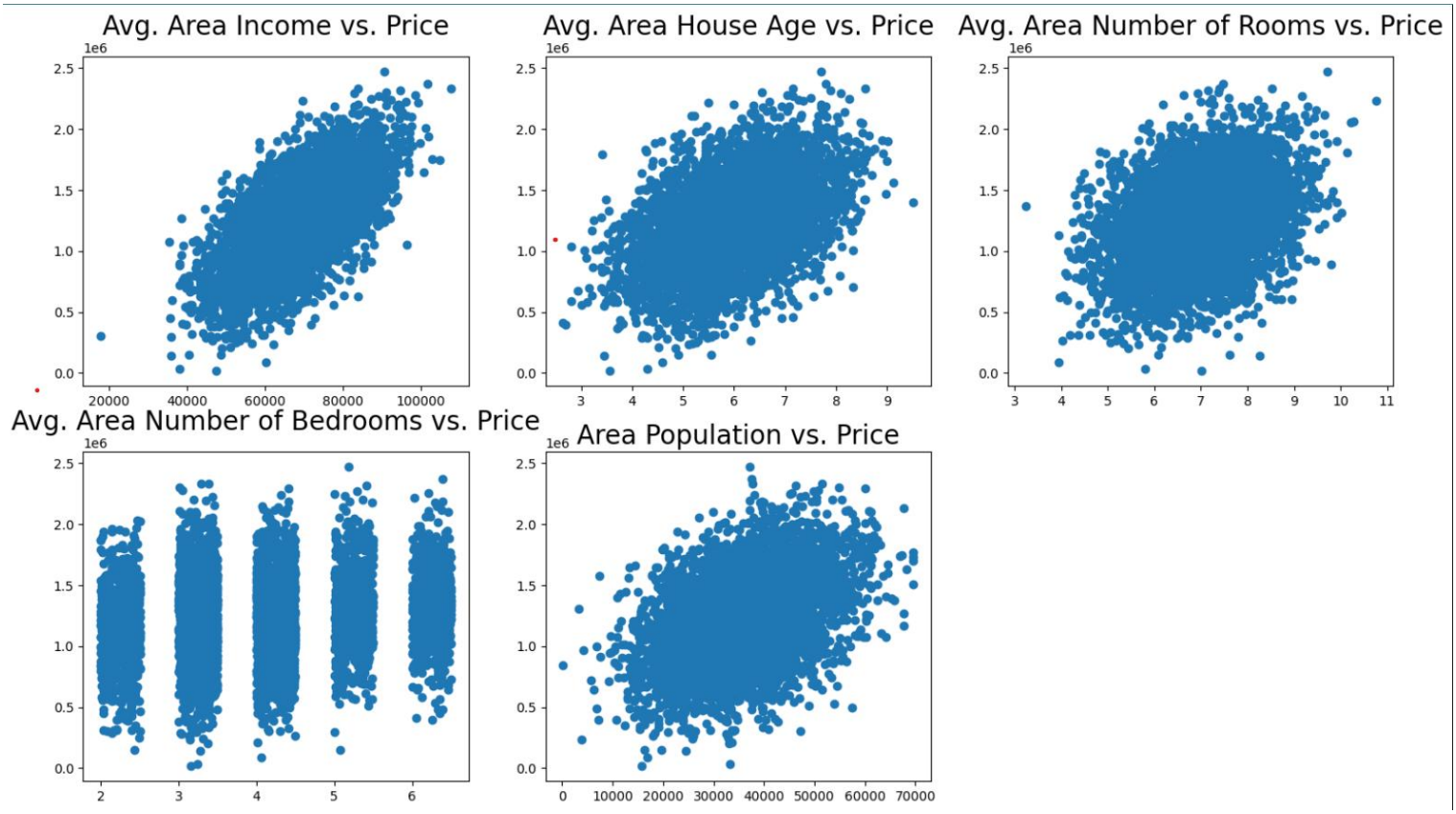




```
# Print a message indicating the features arranged in order of importance for predicting house prices
print("Therefore, features arranged in the order of importance for predicting the house price\n", '-'*90, sep='')

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

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

```
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))
```

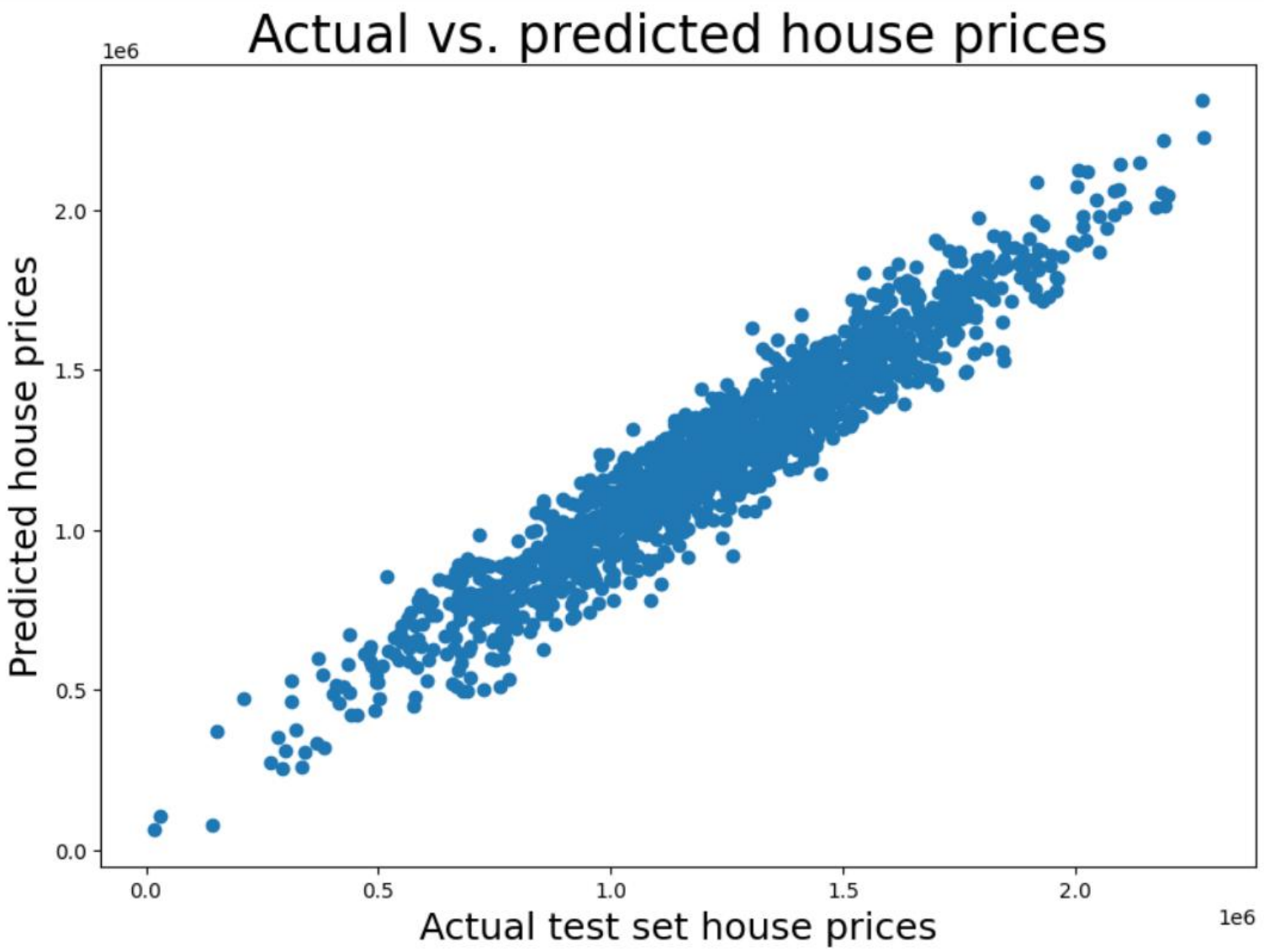
```
R-squared value of this fit: 0.001
```

```
# 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 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)
```

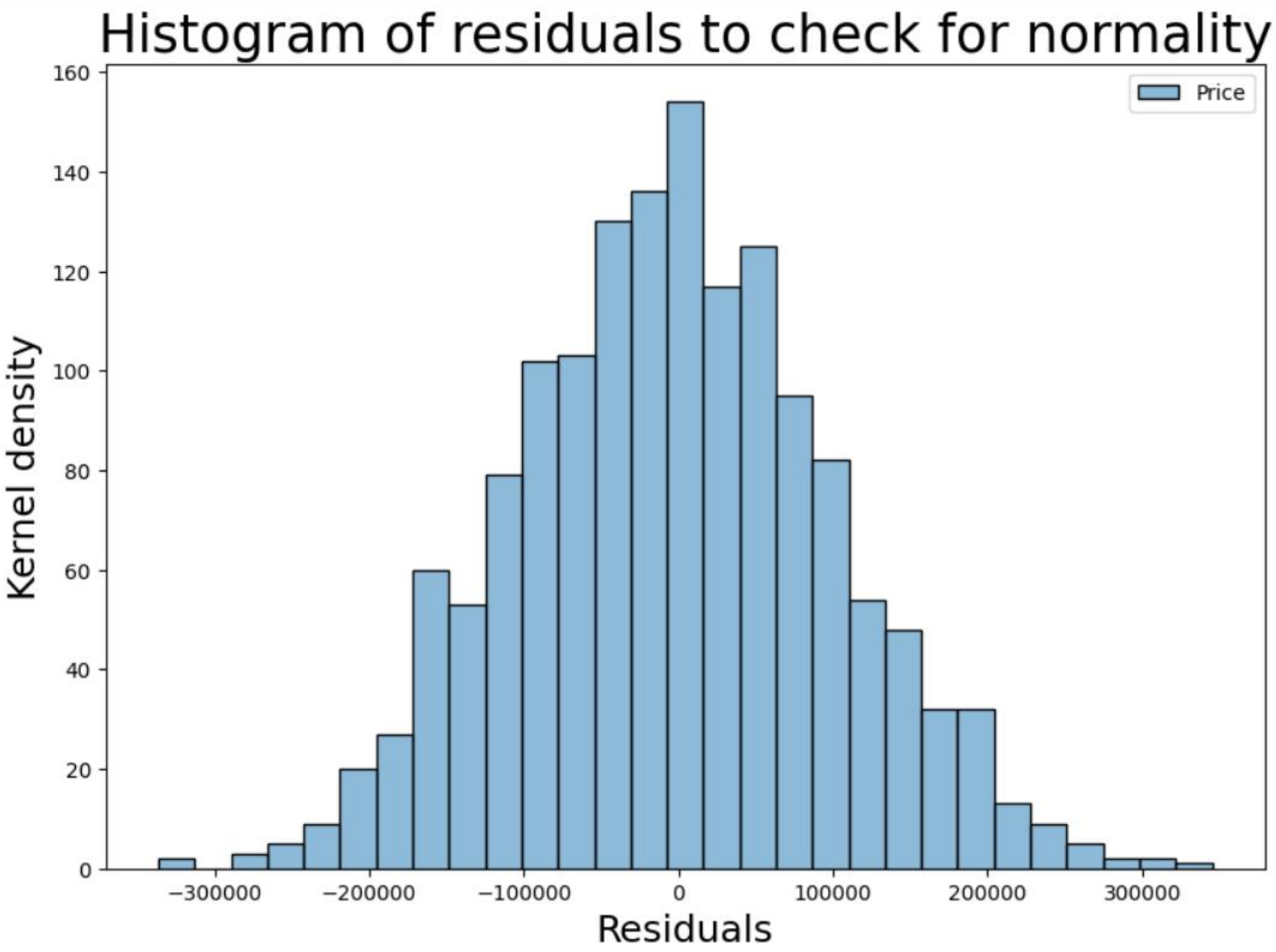
```
Type of the predicted object: <class 'numpy.ndarray'>
Size of the predicted object: (1500,)
```

```
# 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=20)
plt.xlabel("Actual test set house prices", fontsize=14)
plt.ylabel("Predicted house prices", fontsize=14)
```

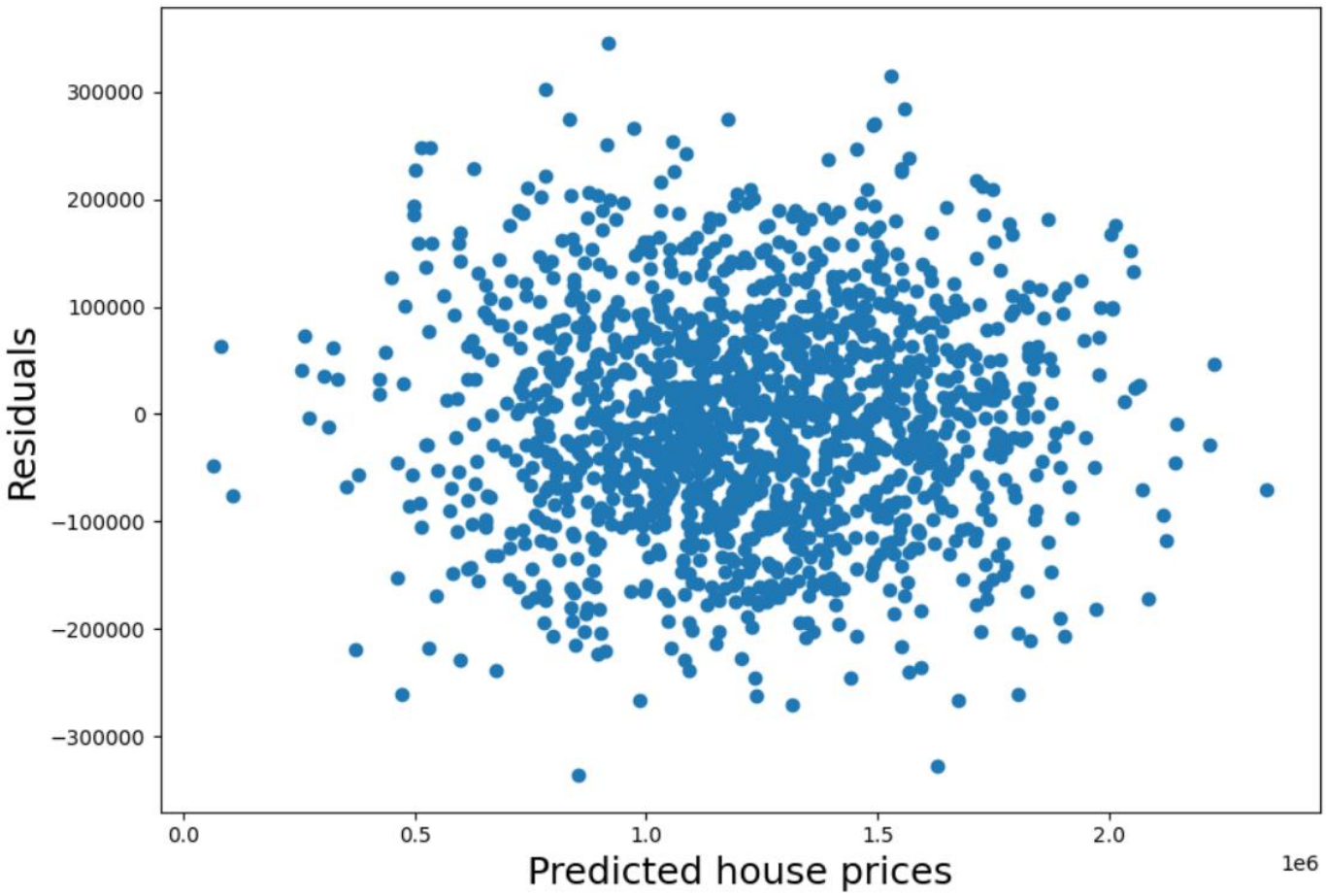
```
# Create the scatter plot with actual house prices on the x-axis and predicted house prices on the y-axis
plt.scatter(x=y_test, y=predictions)
```




```
# 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=20)
plt.xlabel("Residuals", fontsize=14)
plt.ylabel("Kernel density", fontsize=14)

# 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=20)
plt.xlabel("Predicted house prices", fontsize=14)
plt.ylabel("Residuals", fontsize=14)

# 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): 7891.819641389472
Mean square error (MSE): 97777743.66658604
Root mean square error (RMSE): 9888.262924628676
```

```
# 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: -0.001
```

```
import numpy as np

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

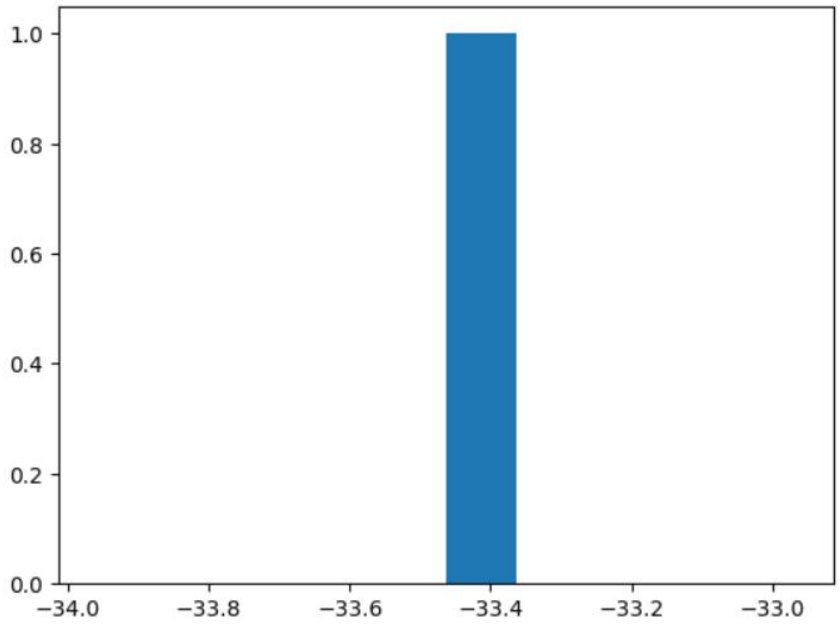
```
# Print the minimum and maximum scaled values
print(min, max)
```

```
5.874576790069147 3.0618162462000096
```

```
# 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([-33.96371714, -33.86371714, -33.76371714, -33.66371714,
        -33.56371714, -33.46371714, -33.36371714, -33.26371714,
        -33.16371714, -33.06371714, -32.96371714]),
 <BarContainer object of 10 artists>)
```



LAB-4

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

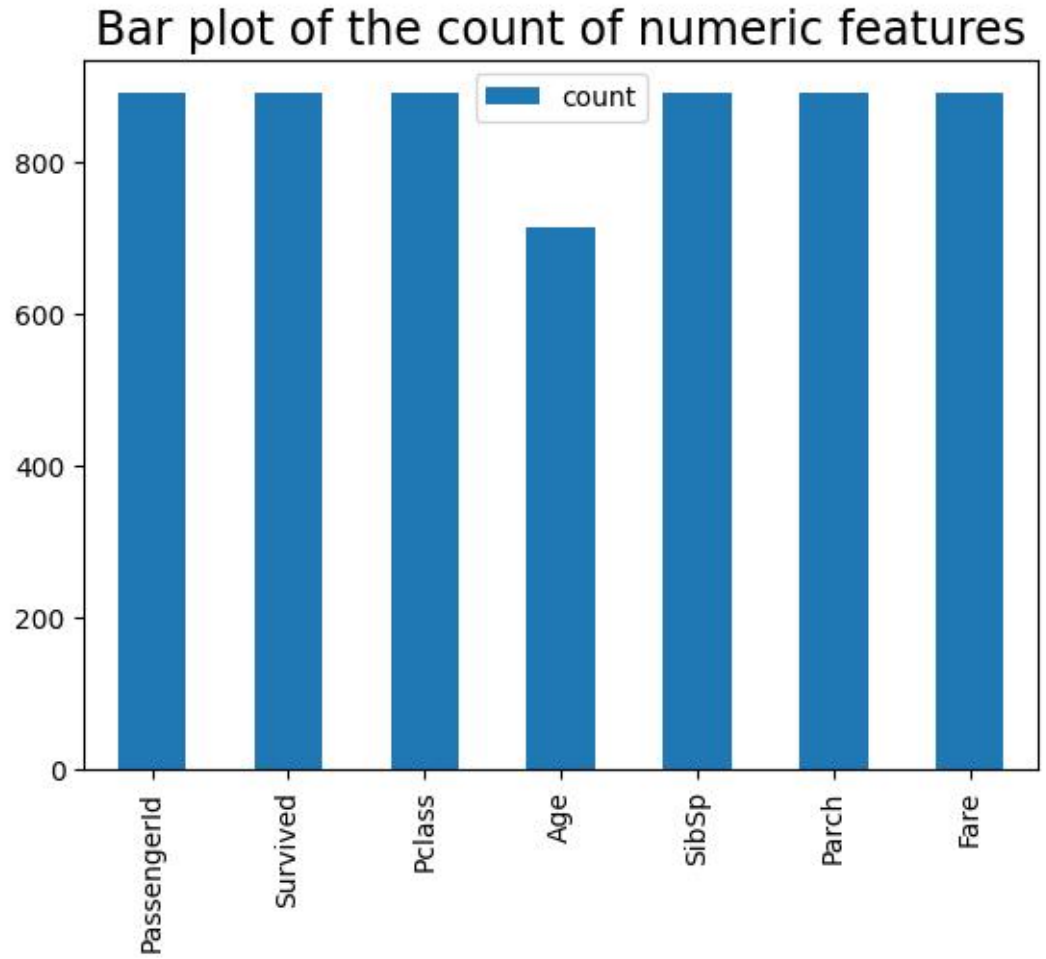
```
# Get information about the dataset, such as column data types and non-null counts
t=train.info()
```

```
<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
```

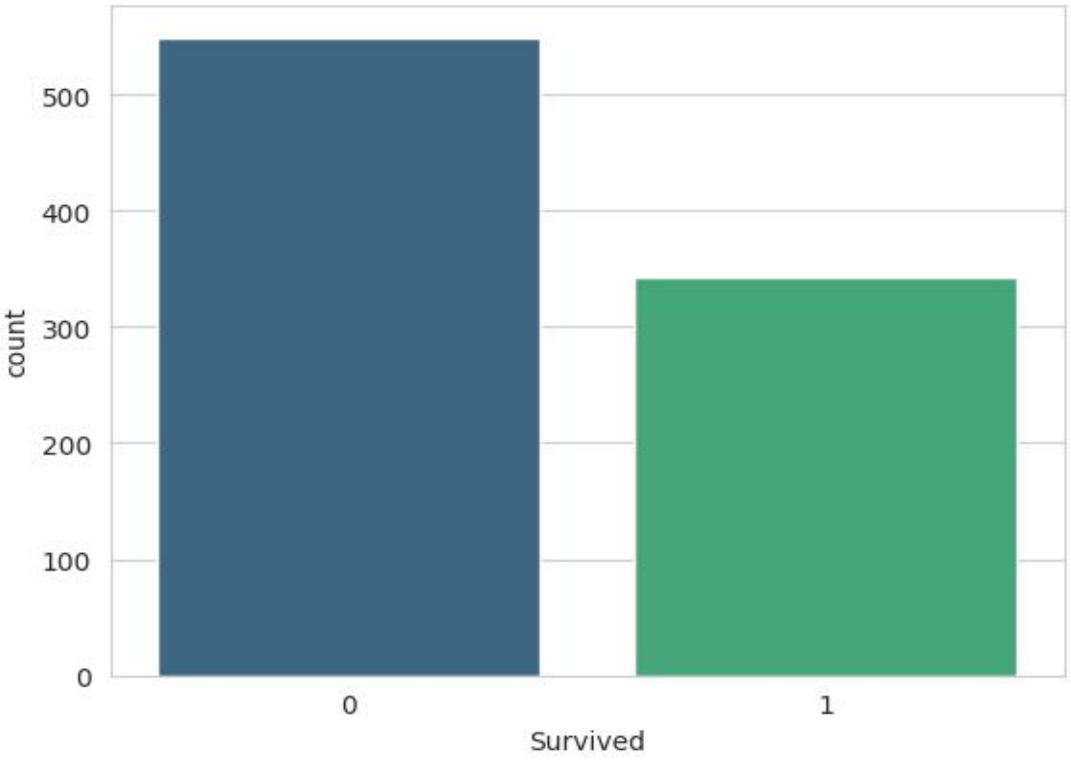
```
# Generate basic statistics for the numeric columns in the dataset
d=train.describe()

# Transpose the summary statistics for better visualization
dT = d.T

# Create a bar plot showing the count of data points for each numeric feature
```

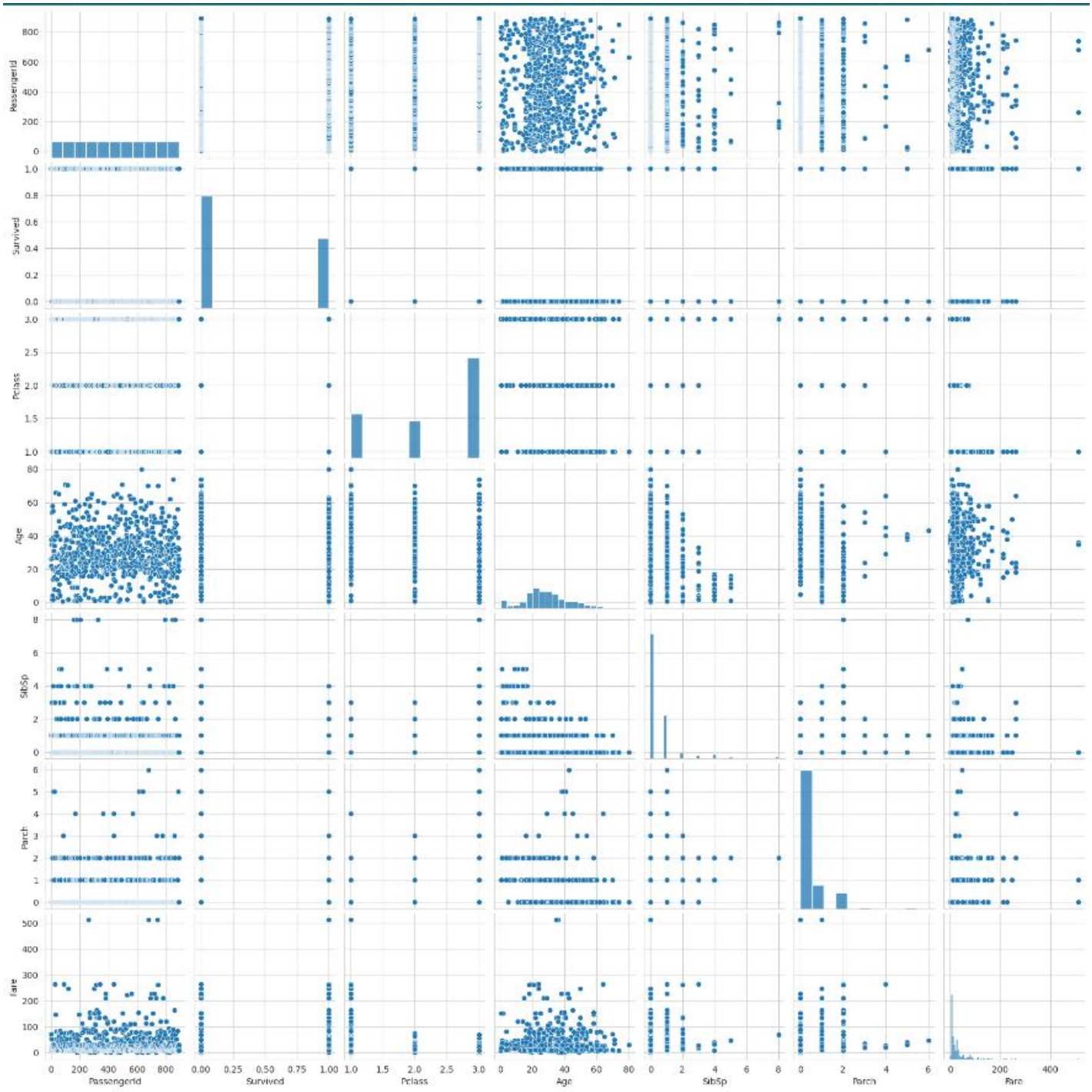
```
dT.plot.bar(y='count')
plt.title("Bar plot of the count of numeric features", fontsize=17)
```

```
Text(0.5, 1.0, 'Bar plot of the count of numeric features')
```



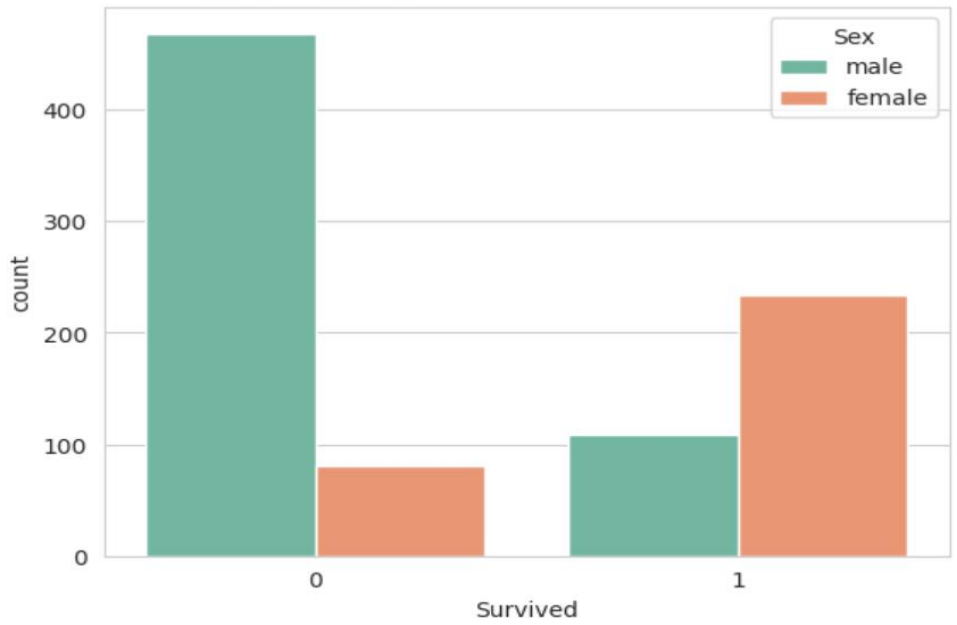

```
# Set the style for Seaborn plots
sns.set_style('whitegrid')

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

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


500
400
300
200
100
0
count
0
1
Survived



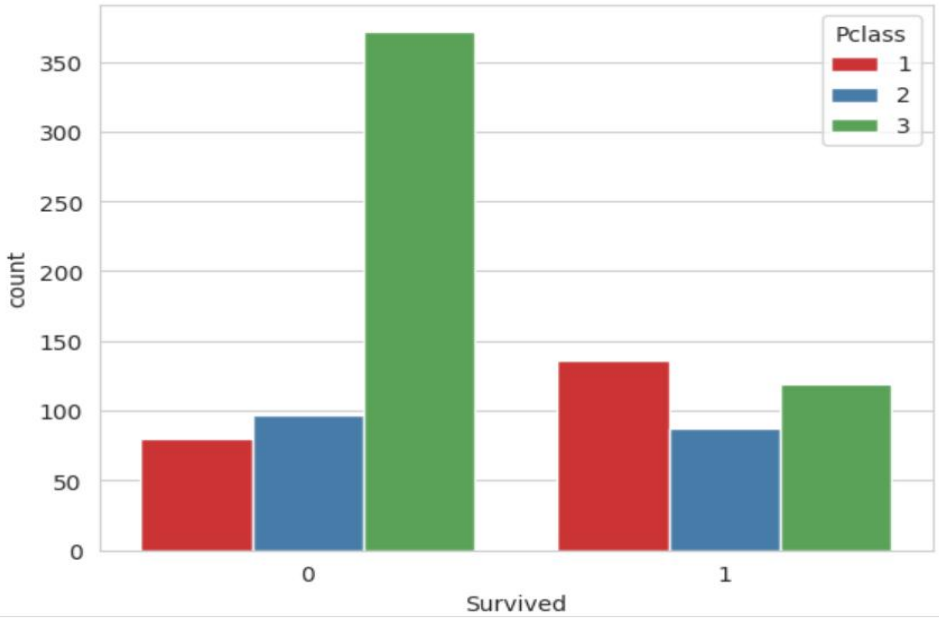

```
# Set the style for Seaborn plots
sns.set_style('whitegrid')
# Create a count plot to visualize the distribution of 'Survived' with 'Sex' as a hue
sns.countplot(x='Survived',hue='Sex',data=train,palette='Set2')
```

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



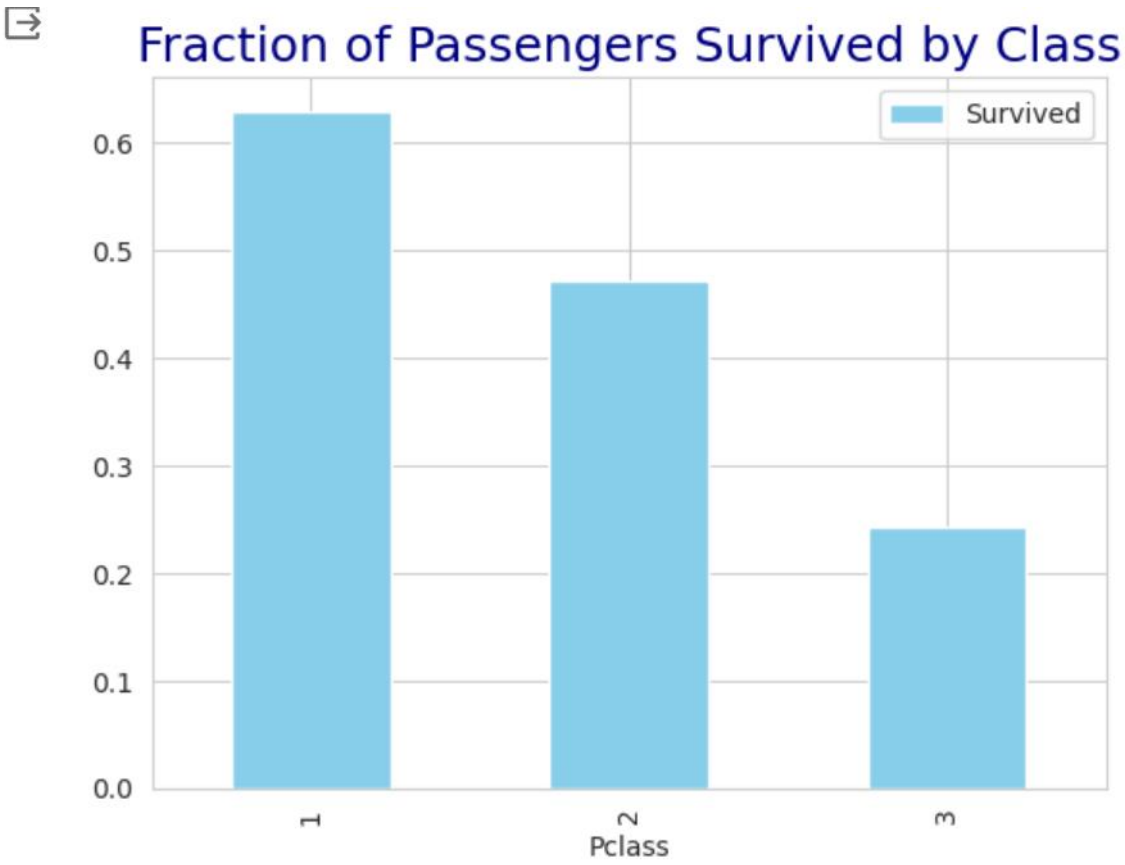

```
# Set the style of the seaborn plots to 'whitegrid'.
sns.set_style('whitegrid')

# Change the color palette to 'Set1' (you can replace 'Set1' with other available palettes).
sns.countplot(x='Survived', hue='Pclass', data=train, palette='Set1')
```

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




```
# Calculate and visualize the fraction of passengers survived by class

# Group the data by 'Pclass' and calculate the mean survival rate for each 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', color='skyblue')

# Set the title for the plot with an updated fontsize and a different color.
```

```
plt.title("Fraction of Passengers Survived by Class", fontsize=18, color='navy')

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



LAB-5

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

```
# Import necessary libraries for data analysis and visualization.
import numpy as np
import cv2
from sklearn.datasets import fetch_california_housing
from sklearn import metrics
from sklearn import model_selection
from sklearn import linear_model

# Enable inline plotting for Jupyter notebooks.
%matplotlib inline

# Import the Matplotlib library and set the style and font size for plots.
import matplotlib.pyplot as plt
plt.style.use('seaborn')  # Change to 'seaborn' style for better readability.
plt.rcParams.update({'font.size': 14})  # Update font size for better visualization.
```

OUTPUT:

<ipython-input-1-e6aeec9b5a45>:14: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
plt.style.use('seaborn') # Change to 'seaborn' style for better readability.

```
# Load the California housing dataset, which contains housing-related data for the state of California.
# This dataset is often used for regression and housing price prediction tasks.
housing=fetch_california_housing()

# List all attributes and methods of the 'housing' object.
# The 'dir()' function is used to inspect the contents of an object.

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

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

```
# Use the '.shape' attribute to retrieve the shape of the 'housing.data' variable.
# This provides the number of rows and columns in the data.
housing.data.shape
```

```
(20640, 8)
```

```
# Check the shape of the 'target' attribute in the 'housing' dataset to determine the number of target values.
housing.target.shape
```

```
(20640,)
```

```
# Initialize a Ridge Regression model.
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=0.25, random_state=2023)

ridgereg.fit(X_train,y_train)
```

```
Ridge
Ridge()
```

```
metrics.mean_squared_error(y_train,ridgereg.predict(X_train))
```

```
0.5252923200928905
```

```
ridgereg.score(X_train,y_train)
```

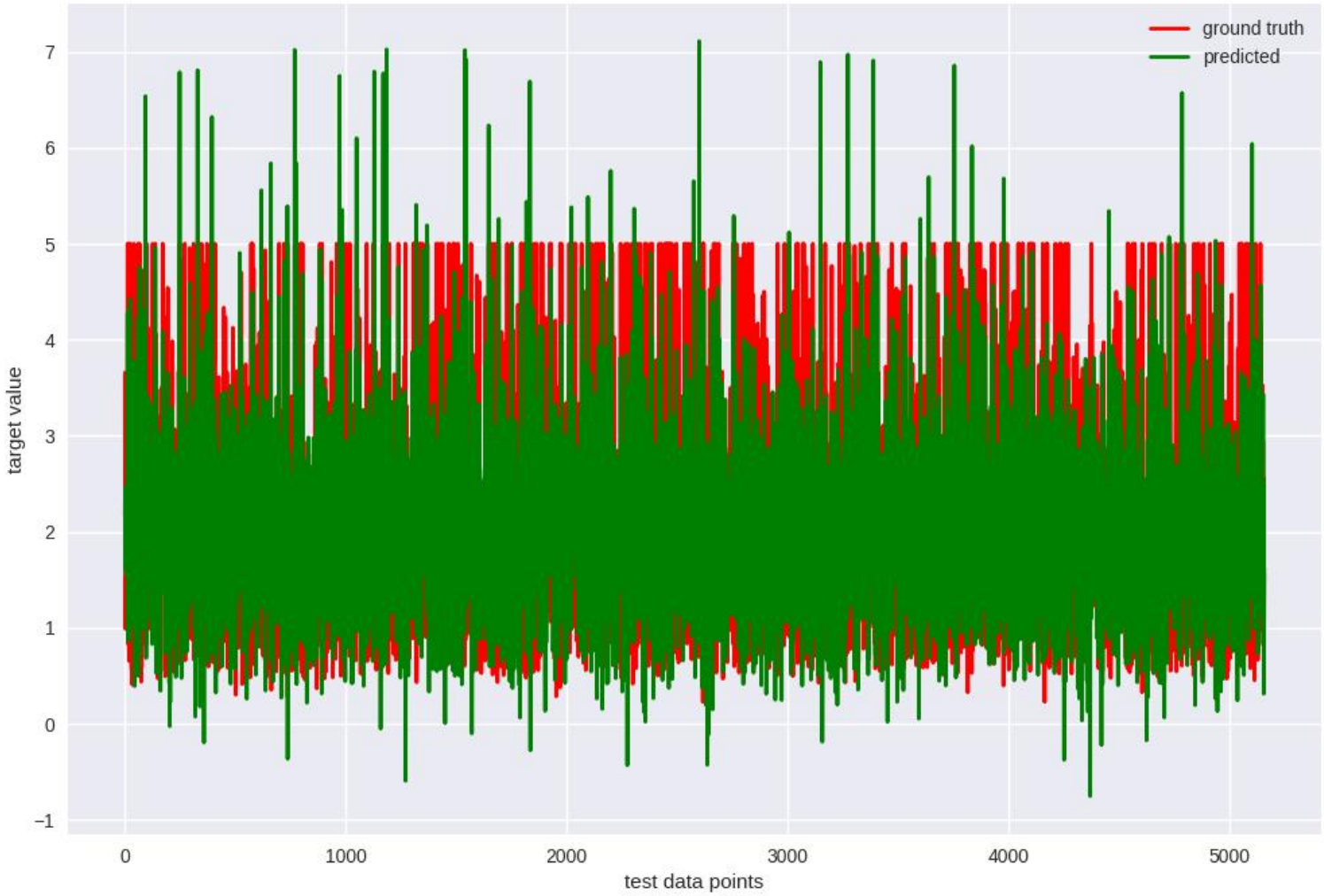
```
0.6040686577008176
```

```
y_pred=ridgereg.predict(X_test)
metrics.mean_squared_error(y_test,y_pred)
```

```
0.5229981589942497
```

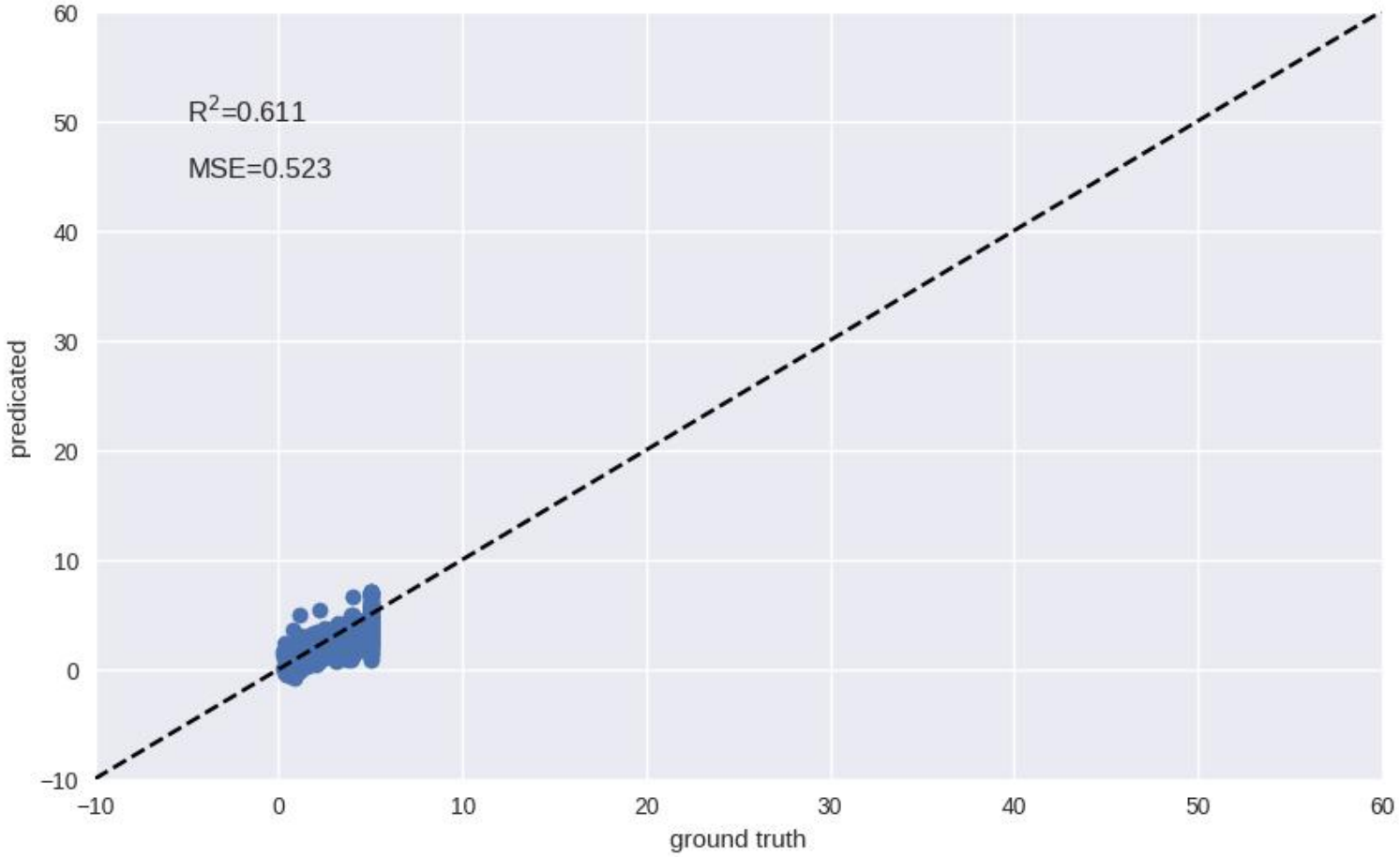
```
# Create a plot to compare ground truth and predicted values.
plt.figure(figsize=(12, 8))

# Plot ground truth and predicted values with labels and line widths.
plt.plot(y_test, linewidth=2, label='ground truth',color='red')
plt.plot(y_pred, linewidth=2, label='predicted',color='green')

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

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

```
Text(-5, 45, 'MSE=0.523')
```


R2=0.611
MSE=0.523
predicated
ground truth
60
50
40
30
20
10
0
−10
−10
0
10
20
30
40
50
60

LAB -6

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 = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# 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','target'])

df
```

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

150 rows × 5 columns

```
# Import the 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'])
```

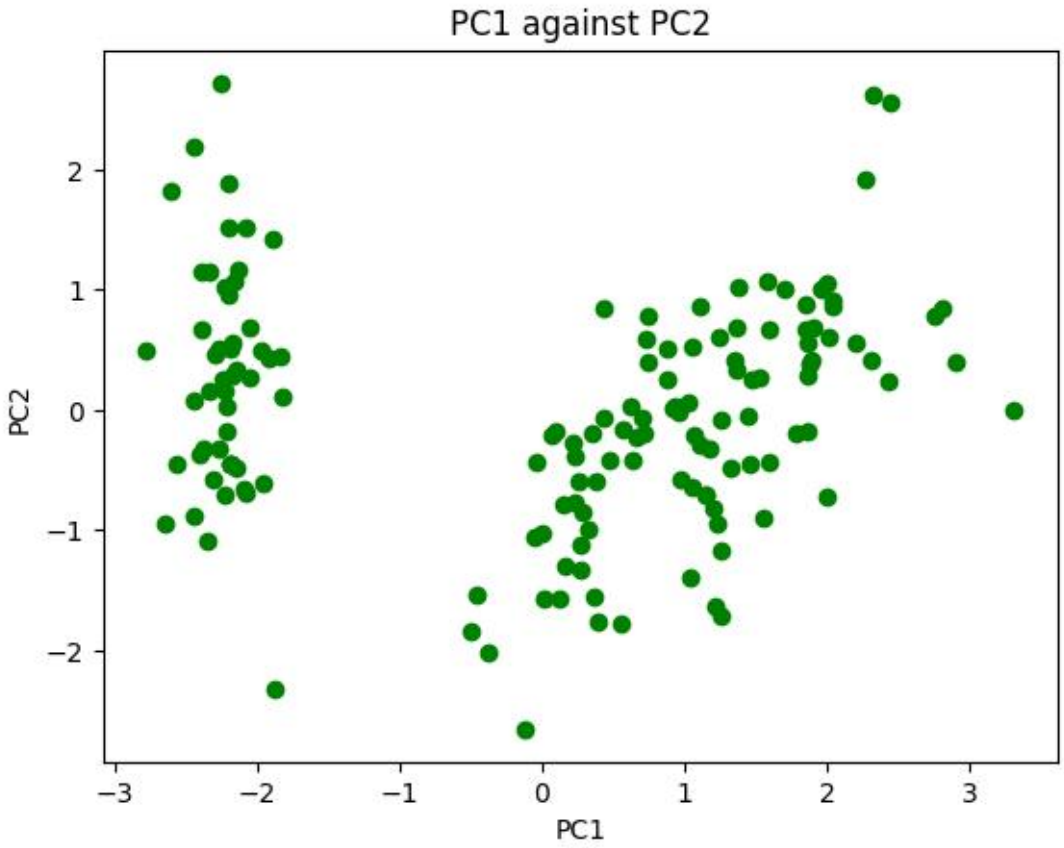
```
# Extract the target class data into a separate DataFrame.
targetDataframe=df[['target']]

# 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

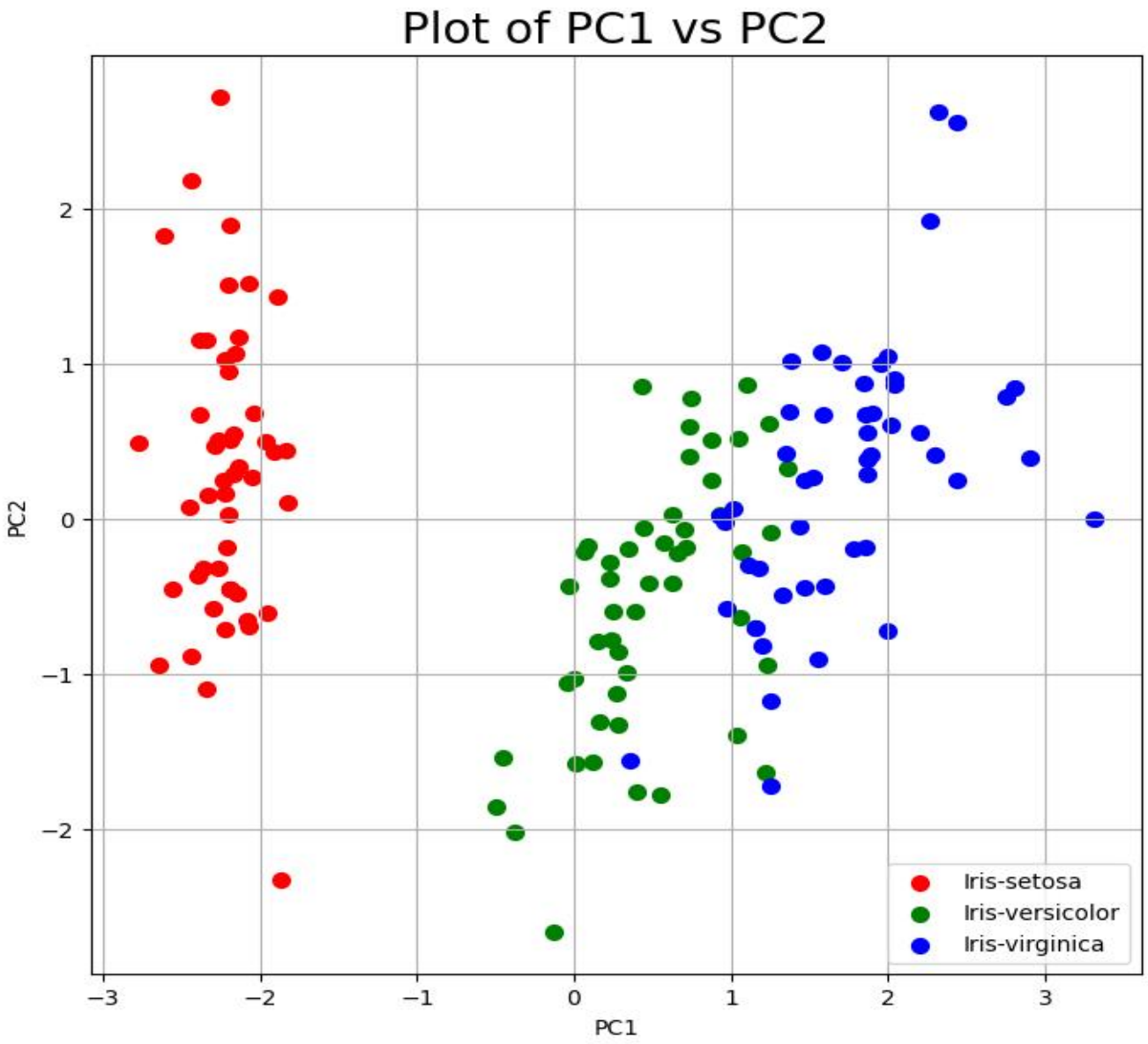
150 rows × 3 columns

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

# 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_
```

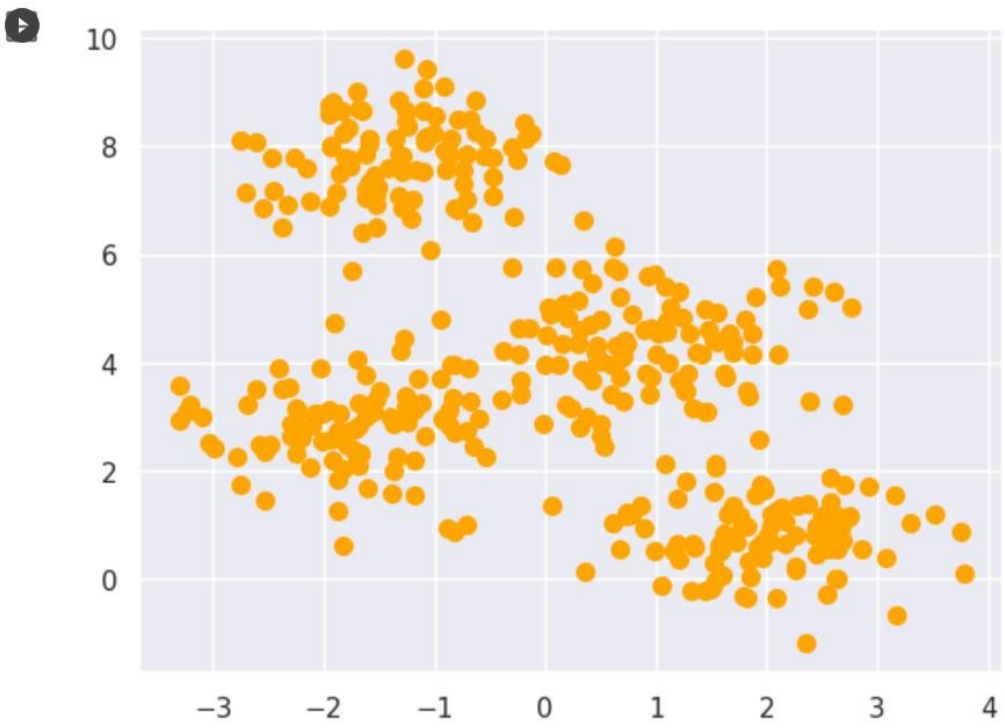
```
array([0.72770452, 0.23030523])
```

LAB-7

SVM and SVR for classification, regression,

```
# Import the necessary libraries for data visualization.
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import numpy as np
```

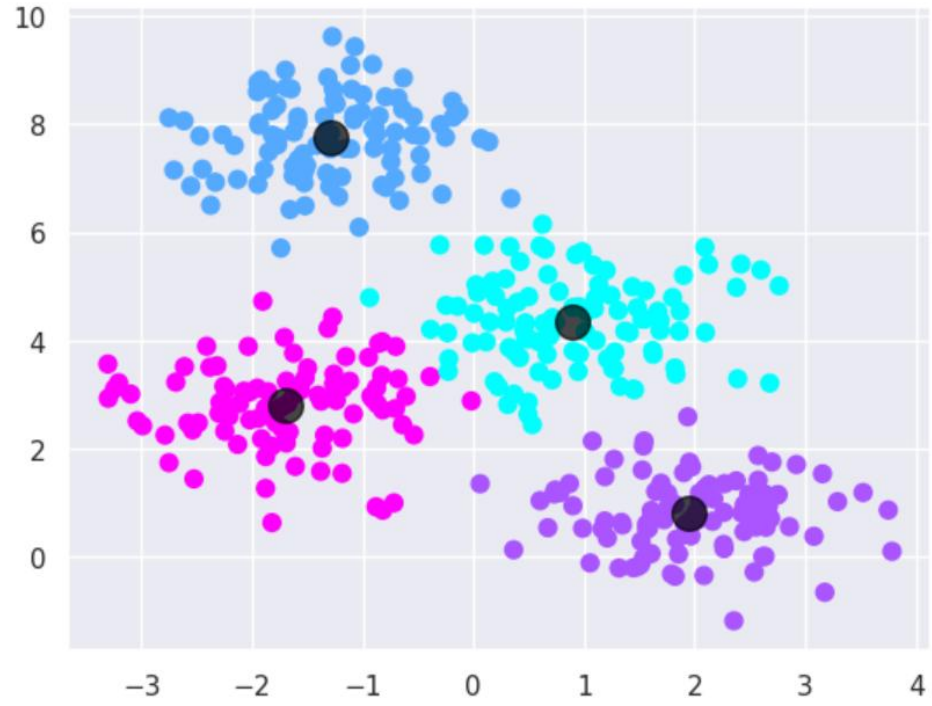
```
# Import the necessary library for generating synthetic data.
from sklearn.datasets import make_blobs
# Generate synthetic data using 'make_blobs' with specified parameters.
X,y_true=make_blobs(n_samples=400,centers=4,cluster_std=0.75,random_state=0)
# Create a scatter plot of the generated data points with customized appearance.
plt.scatter(X[:,0],X[:,1],s=50,color='orange');
```




```
# Import the KMeans class from scikit-learn for clustering.
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, n_init=10)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

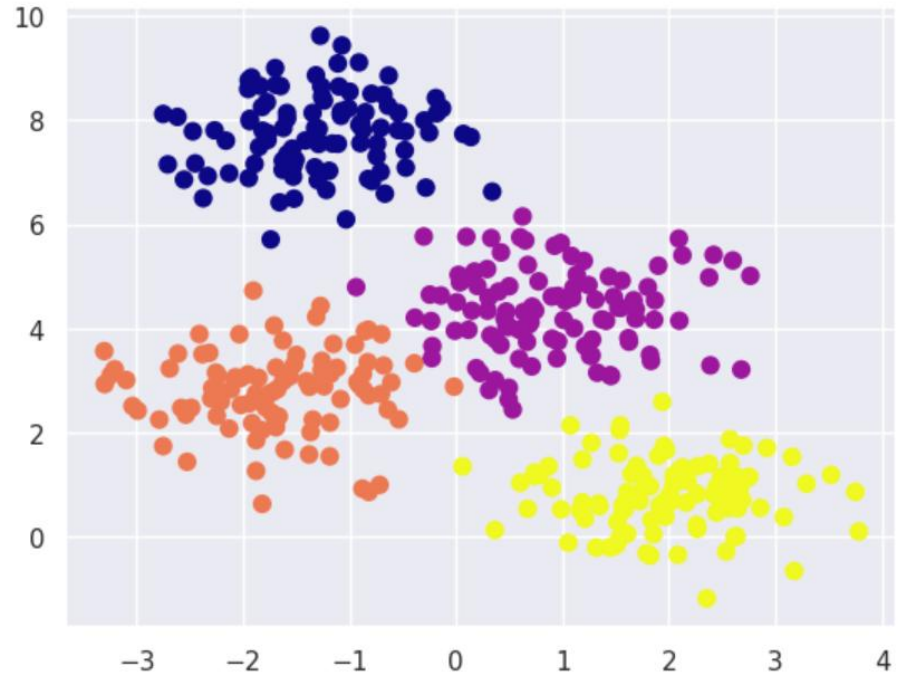
```
# Create a scatter plot to visualize the data points.
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='cool')

# Get the cluster centers from the 'kmeans' clustering.
centers = kmeans.cluster_centers_

# Scatter plot the cluster centers in black with larger markers and reduced transparency.
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.7)
```

```
<matplotlib.collections.PathCollection at 0x793d408ad4e0>
```



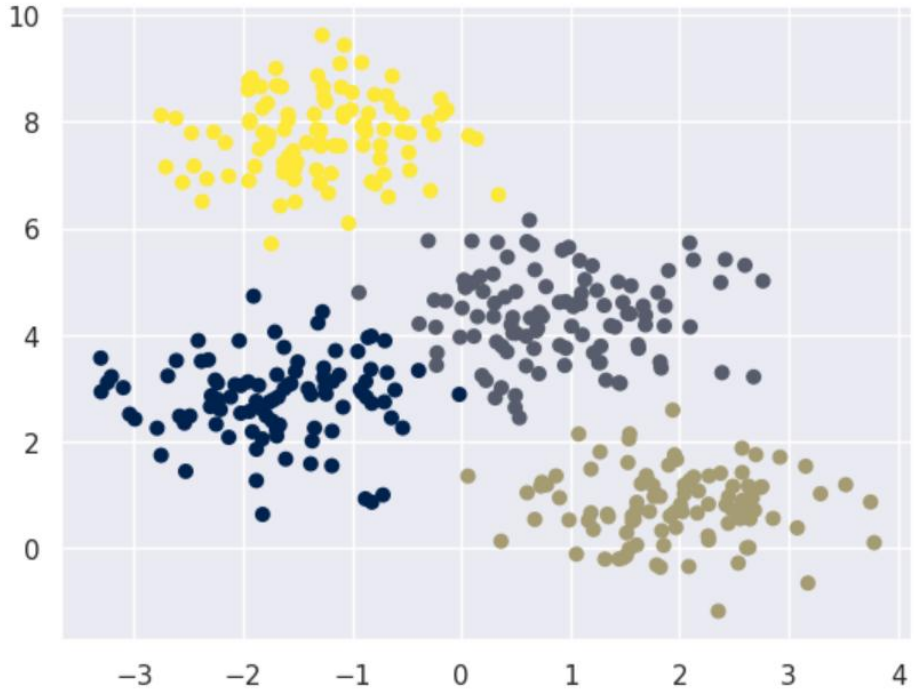

```
# Import necessary libraries and modules.
from sklearn.metrics import pairwise_distances_argmin
import numpy as np
import matplotlib.pyplot as plt

# Define a function to find clusters and their centers.
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:
        labels = pairwise_distances_argmin(X, centers)
        # Find new centers from means of points within clusters.
        new_centers = np.array([X[labels == i].mean(0) for i in range(n_clusters)])
        # Check for convergence.
        if np.all(centers == new_centers):
            break
        centers = new_centers
    return centers, labels

# Generate clusters and labels using the 'find_clusters' function.
centers, labels = find_clusters(X, 4)

# Create a scatter plot with updated values and a different color map.
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='plasma')
```

```
<matplotlib.collections.PathCollection at 0x793d3bee13c0>
```




```
# The 'X' array represents your data points, '4' is the number of clusters, and 'rseed' is set to 0 for reproducibility.
centers, labels = find_clusters(X, 4, rseed=0)

# Create a scatter plot of data points with color-coded labels.
# Change the 's' parameter to adjust the size of the points, and 'cmap' to change the color map.
plt.scatter(X[:, 0], X[:, 1], c=labels, s=30, cmap='cividis')
```

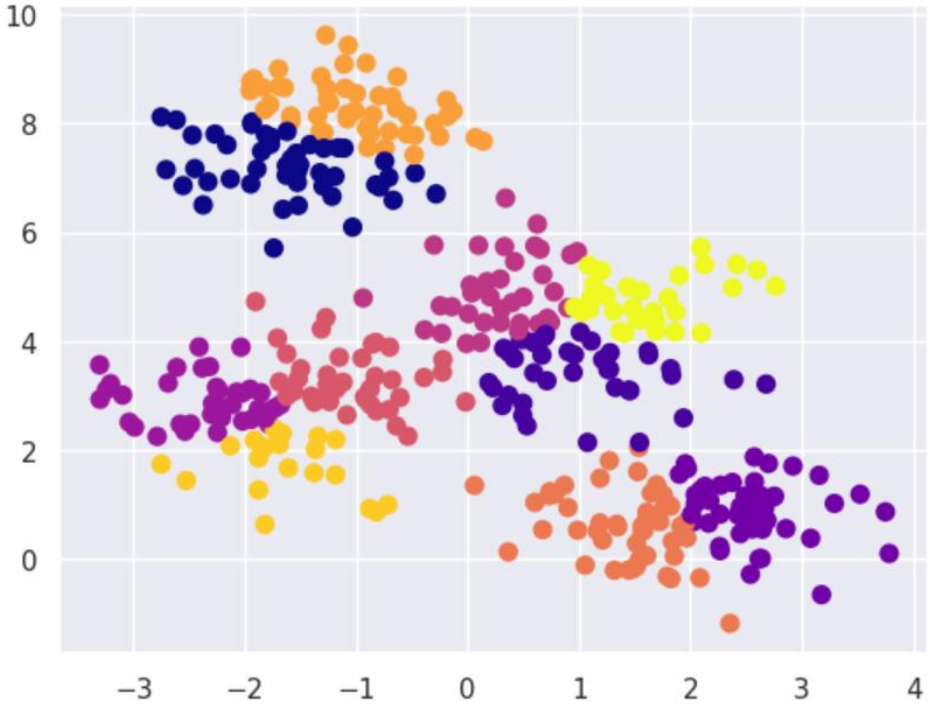
```
<matplotlib.collections.PathCollection at 0x793d3bf573d0>
```




```
# Apply KMeans clustering with 10 clusters and 10 initializations.
# Assign cluster labels to data points.
labels = KMeans(n_clusters=10, random_state=0, n_init=10).fit_predict(X)
```

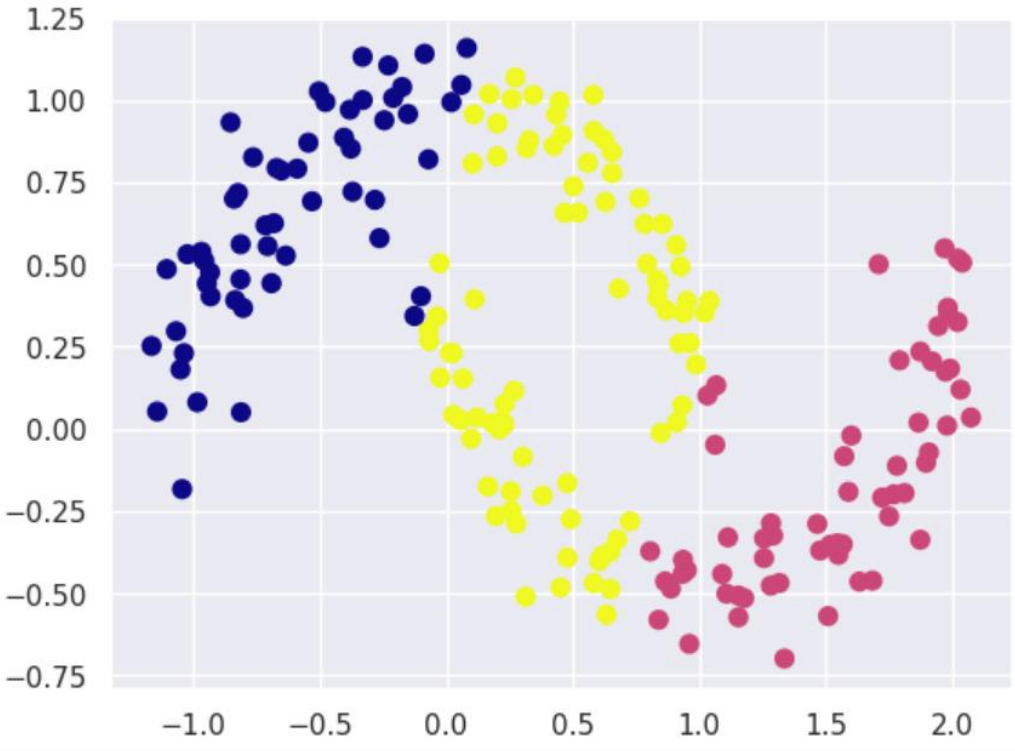
```
# Create a scatter plot of the data points, using the cluster labels for color-coding.
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='plasma')
```

```
<matplotlib.collections.PathCollection at 0x793d3a5e98a0>
```

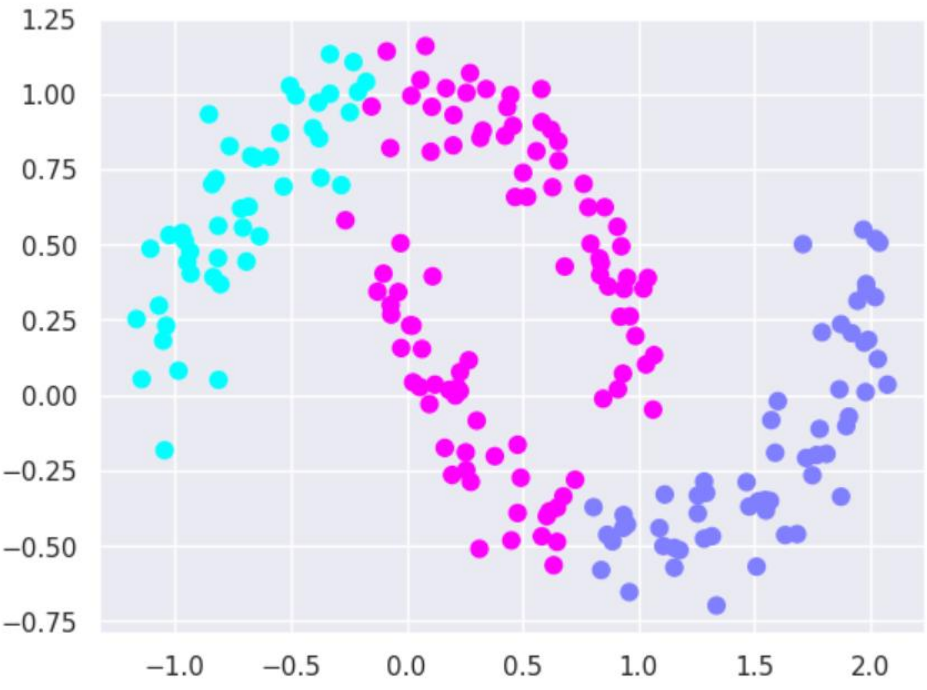



```
# Import the 'make_moons' function from sklearn.datasets to generate a synthetic dataset.
from sklearn.datasets import make_moons
# Generate a synthetic dataset with 200 data points and a small amount of noise (0.05).
X, y = make_moons(n_samples=200, noise=0.1, random_state=0)
labels = KMeans(n_clusters=3, random_state=42, n_init=10).fit_predict(X)
# Create a scatter plot of the data points, color-coded by cluster labels.
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='plasma')
```

```
<matplotlib.collections.PathCollection at 0x793d3a31c940>
```




```
#kernel transformation
from sklearn.cluster import SpectralClustering
# Create a SpectralClustering model with specified parameters.
model=SpectralClustering(n_clusters=3,affinity='nearest_neighbors',assign_labels='kmeans')
# Fit the model to the data and predict cluster labels.
labels=model.fit_predict(X)
# Create a scatter plot of the data points with cluster labels.
plt.scatter(X[:,0],X[:,1],c=labels,s=50, cmap='cool');
```



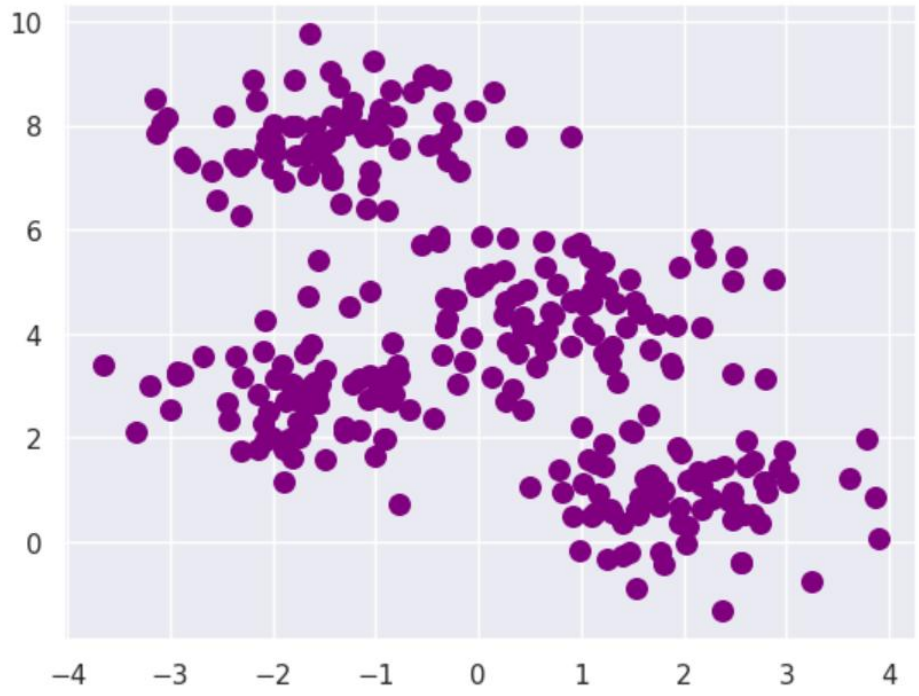
LAB-8

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
```

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

# Generate synthetic data using make_blobs.
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.8, random_state=0)

# Create a scatter plot of the generated data with larger point size (s=70) and a different color (e.g., 'purple').
plt.scatter(X[:, 0], X[:, 1], s=70, color='purple')
```

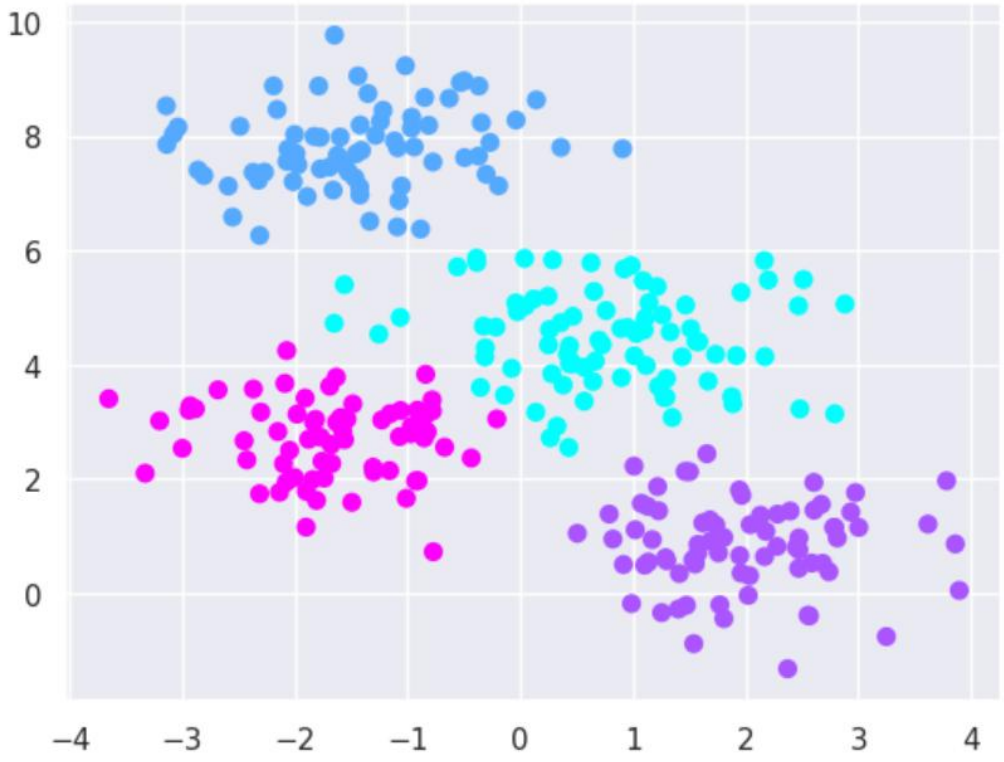
```
<matplotlib.collections.PathCollection at 0x7b28fb8c2b30>
```




```
# Import the GaussianMixture module from scikit-learn for Gaussian Mixture Models.
from sklearn.mixture import GaussianMixture

# Create a GMM model with the desired number of components.
gmm = GaussianMixture(n_components=4).fit(X)

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

# Create a scatter plot of the data points with cluster labels.
```

```
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='cool')
```

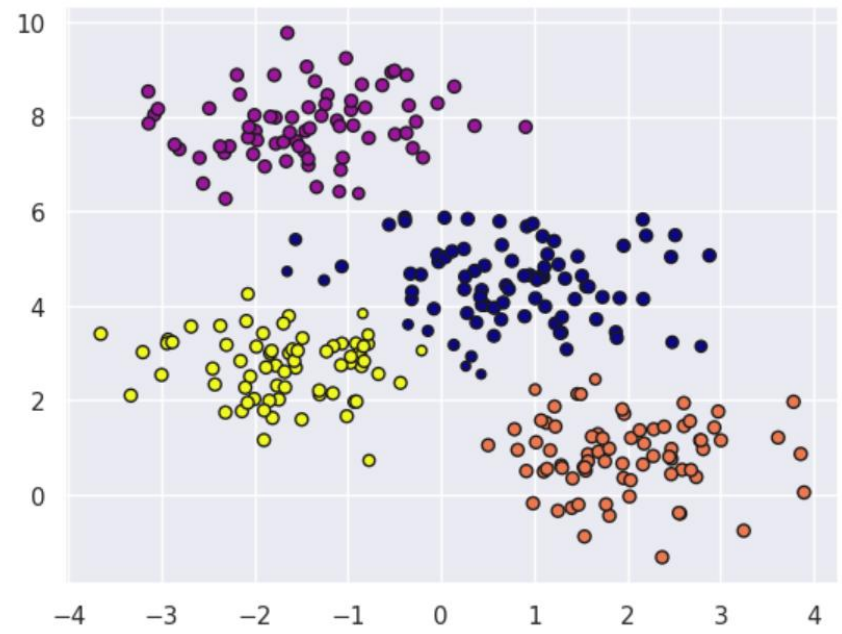
```
<matplotlib.collections.PathCollection at 0x7b28f84f6c20>
```




```
# Calculate the probabilities of data points belonging to each component using the GMM.
probs=gmm.predict_proba(X)
# Print the probabilities for the first 5 data points, rounded to 3 decimal places.
print(probs[:5].round(3))
```

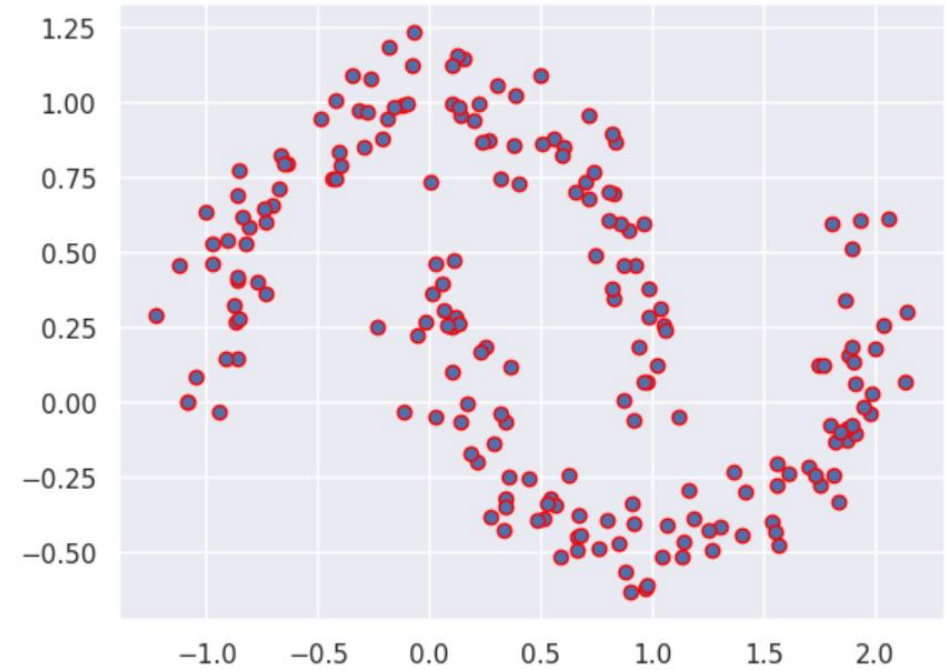
```
[[0.545 0.    0.133 0.322]
 [0.    1.    0.    0.   ]
 [1.    0.    0.    0.   ]
 [0.    1.    0.    0.   ]
 [0.196 0.    0.782 0.022]]
```

```
#print(probs.max(1))

size=probs.max(1)/0.03
plt.scatter(X[:,0],X[:,1],c=labels,edgecolor='k',cmap='plasma',s=size);
```



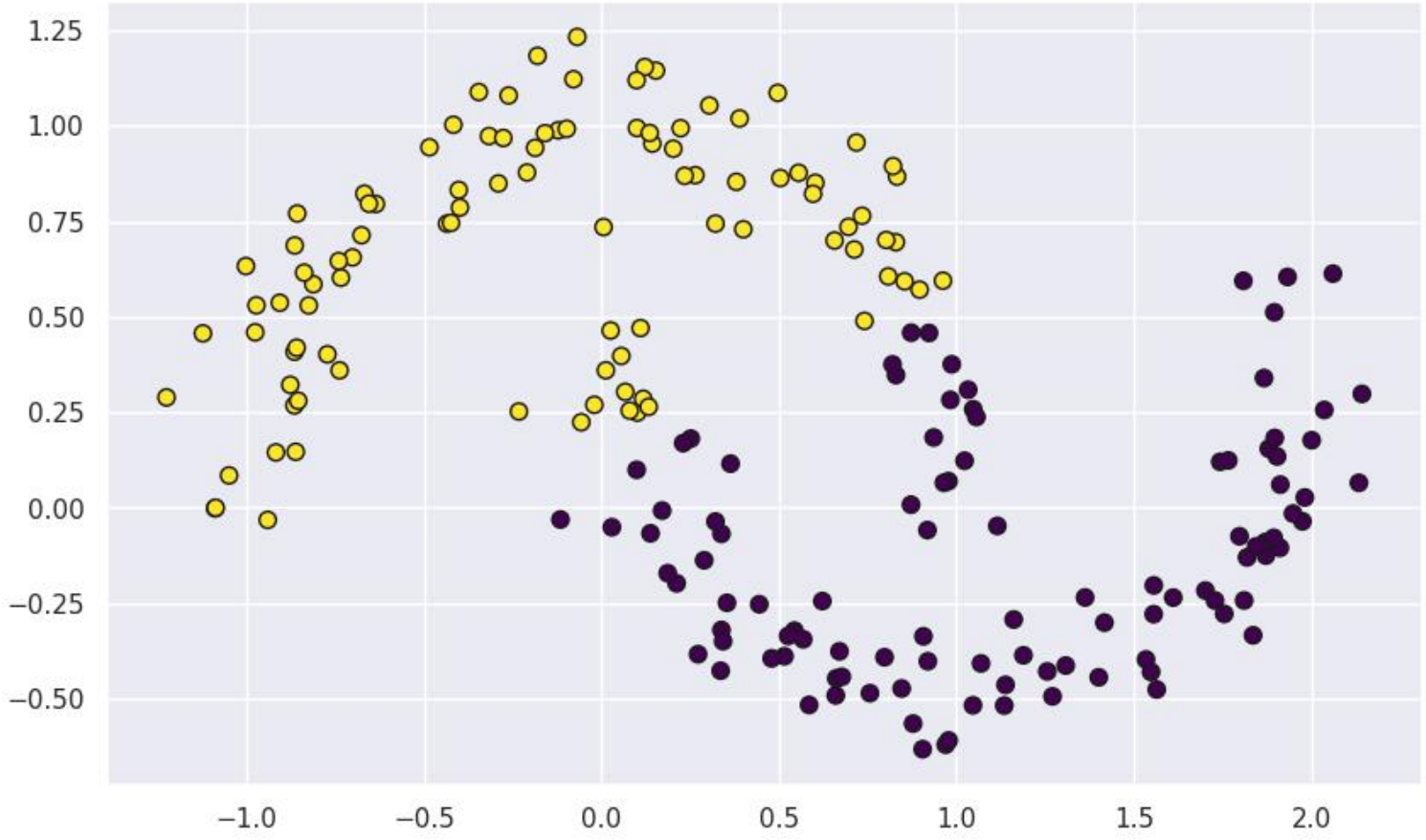

```
# Import the make_moons function from scikit-learn to create moon-shaped data.
from sklearn.datasets import make_moons

# Generate moon-shaped data with 200 samples, noise level 0.1, and a specific random state.
Xmoon, ymoon = make_moons(200, noise=0.1, random_state=42)

# Create a scatter plot of the moon-shaped data points with edges in a different color.
plt.scatter(Xmoon[:, 0], Xmoon[:, 1], edgecolor='red')
```

```
<matplotlib.collections.PathCollection at 0x7b28f242a020>
```




```
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=(10,6))
plot_gmm(gmm2,Xmoon)
```

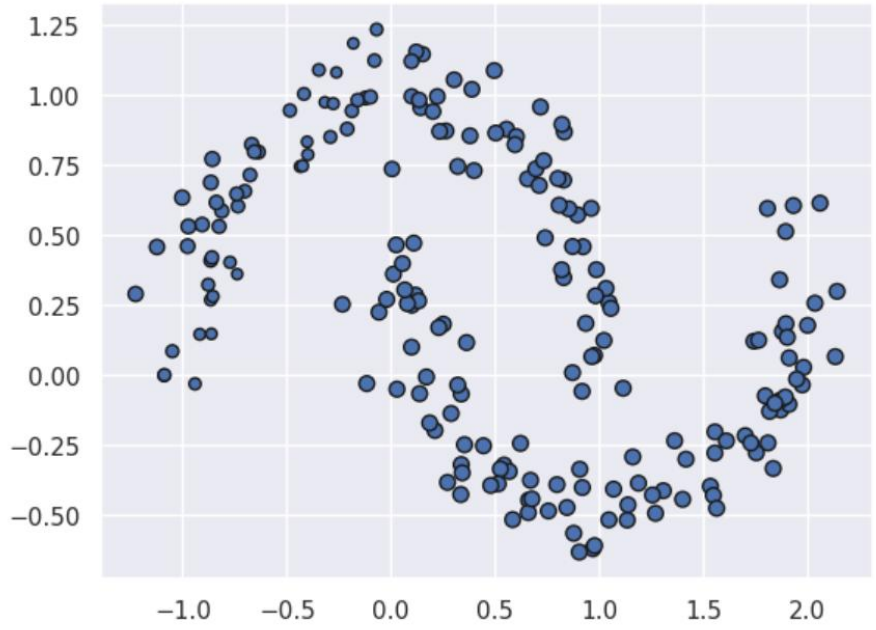



```
# Predict the probabilities of data points using Gaussian Mixture Models.
probs = gmm.predict_proba(Xmoon)

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

```
[[0.    0.    0.324 0.676]
 [0.    0.    1.    0.   ]
 [0.    0.    0.999 0.001]
 [0.    0.    1.    0.   ]
 [0.    0.    1.    0.   ]]
```

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

```
<matplotlib.collections.PathCollection at 0x7b28f22bb910>
```



LAB-9

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

```
# Import the necessary libraries for data analysis and visualization.
import numpy as np  # Import NumPy for numerical operations.
import pandas as pd  # Import pandas for data manipulation.
import matplotlib.pyplot as plt  # Import Matplotlib for data visualization.
import seaborn as sns  # Import Seaborn for enhanced data visualization.

# Enable inline plotting for Jupyter notebooks.
%matplotlib inline
```

```
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'])
```

OUTPUT:

 # Print the dataset description
print(cancer['DESCR']).. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset

****Data Set Characteristics:****

:Number of Instances: 569

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

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

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

10 is Radius SE, field 20 is Worst Radius.

- class:
    - WDBC-Malignant
    - WDBC-Benign

:Summary Statistics:

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

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

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

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.

https://goo.gl/U2Uwz2

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

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

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

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

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

.. topic:: References

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

```
# 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()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
 14  smoothness error        569 non-null    float64
 15  compactness error       569 non-null    float64
 16  concavity error         569 non-null    float64
 17  concave points error    569 non-null    float64
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
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	...
count	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	...
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...

8 rows × 30 columns

worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
16.269190	25.677223	107.261213	880.583128	0.132369	0.254265	0.272188	0.114606	0.290076	0.083946
4.833242	6.146258	33.602542	569.356993	0.022832	0.157336	0.208624	0.065732	0.061867	0.018061
7.930000	12.020000	50.410000	185.200000	0.071170	0.027290	0.000000	0.000000	0.156500	0.055040
13.010000	21.080000	84.110000	515.300000	0.116600	0.147200	0.114500	0.064930	0.250400	0.071460
14.970000	25.410000	97.660000	686.500000	0.131300	0.211900	0.226700	0.099930	0.282200	0.080040
18.790000	29.720000	125.400000	1084.000000	0.146000	0.339100	0.382900	0.161400	0.317900	0.092080
36.040000	49.540000	251.200000	4254.000000	0.222600	1.058000	1.252000	0.291000	0.663800	0.207500

```
# 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()
```

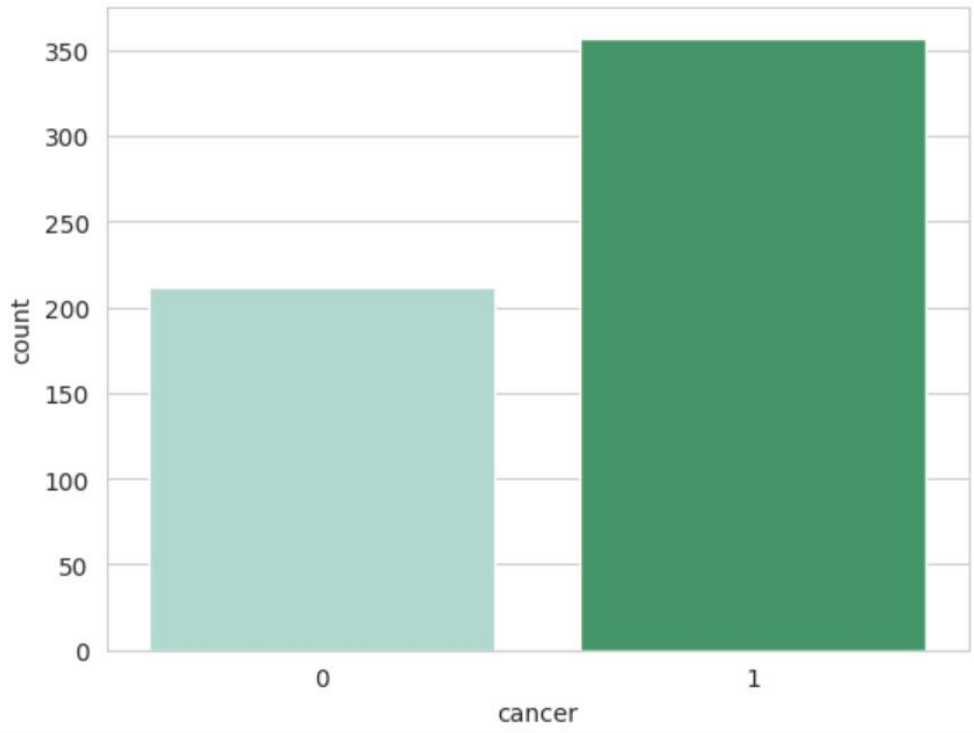
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812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

5 rows × 31 columns

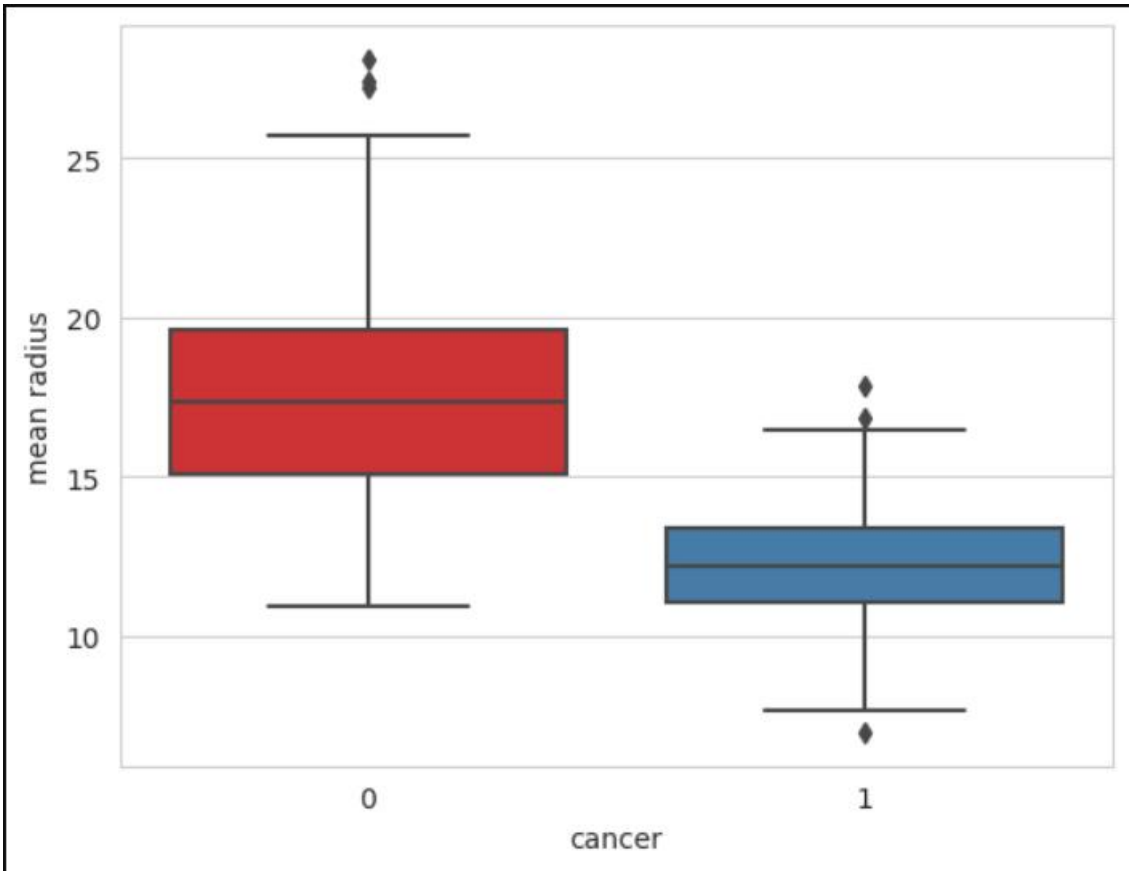
worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	cancer
17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0
26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

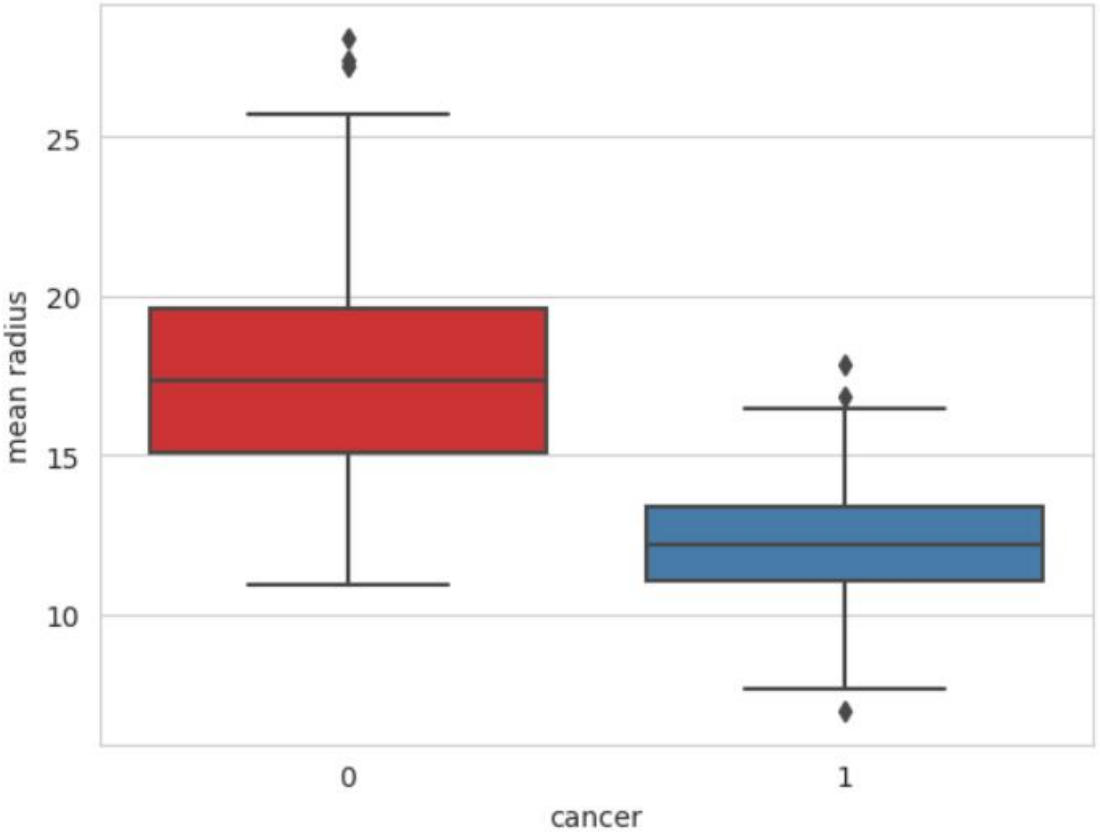
```
# Set the Seaborn style for plotting
sns.set_style('whitegrid')

# Create a count plot to visualize the distribution of cancer diagnosis
sns.countplot(x='cancer', data=df, palette='BuGn')
```

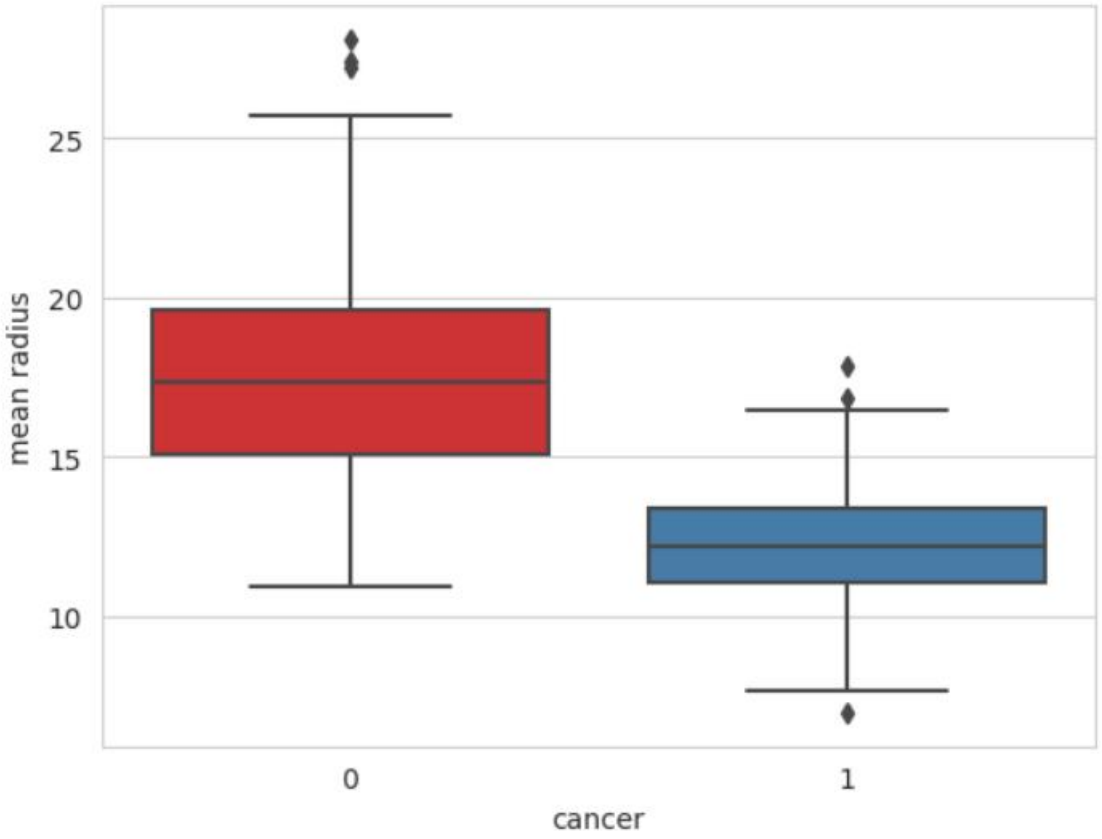
```
<Axes: xlabel='cancer', ylabel='count'>
```

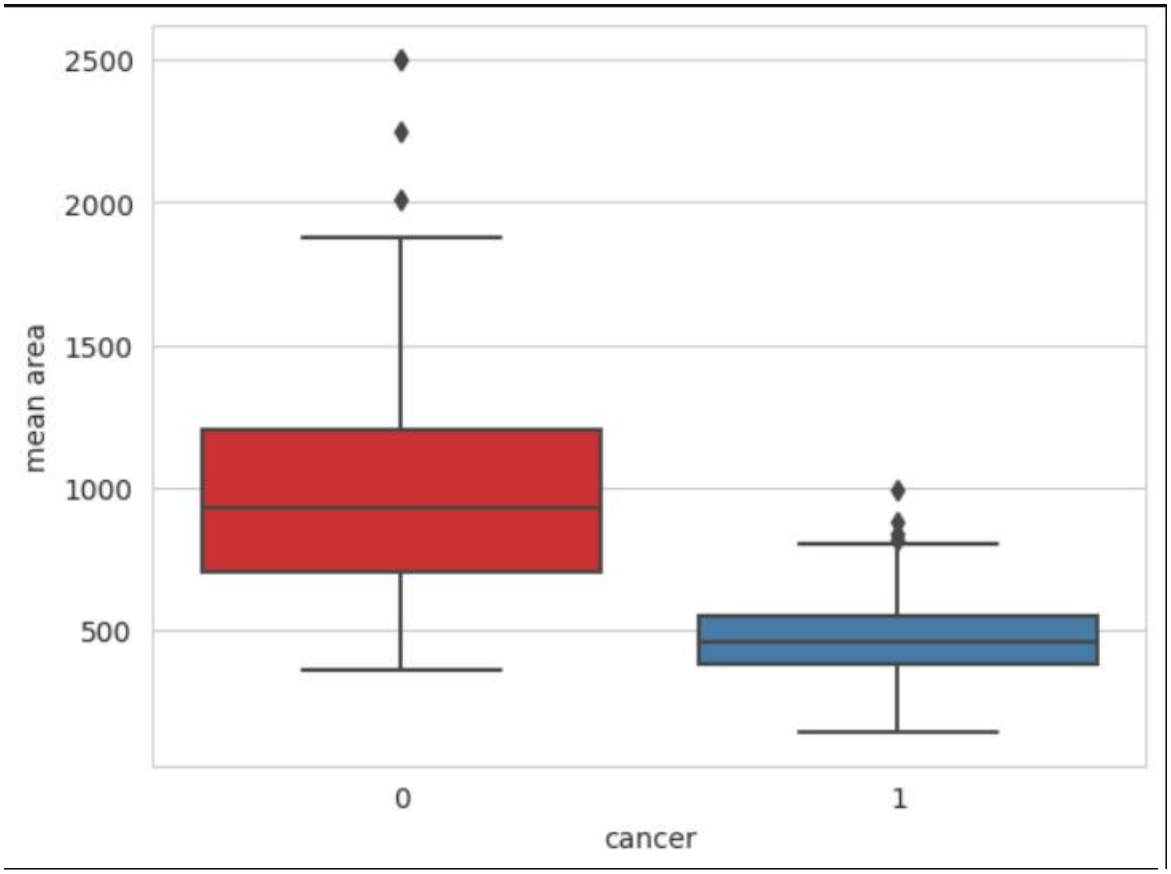


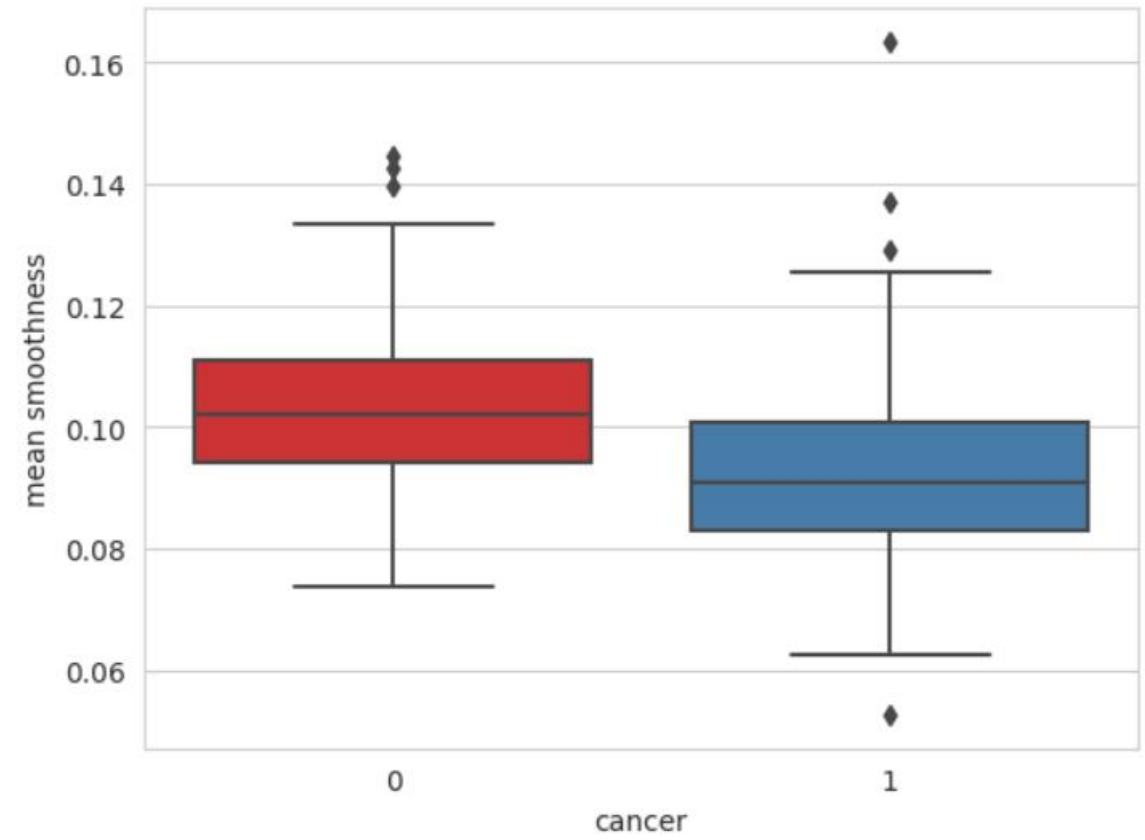
```
# Define a list of feature names for box plots
l = list(df.columns[0:10])

# 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='Set1')
    plt.figure()
```

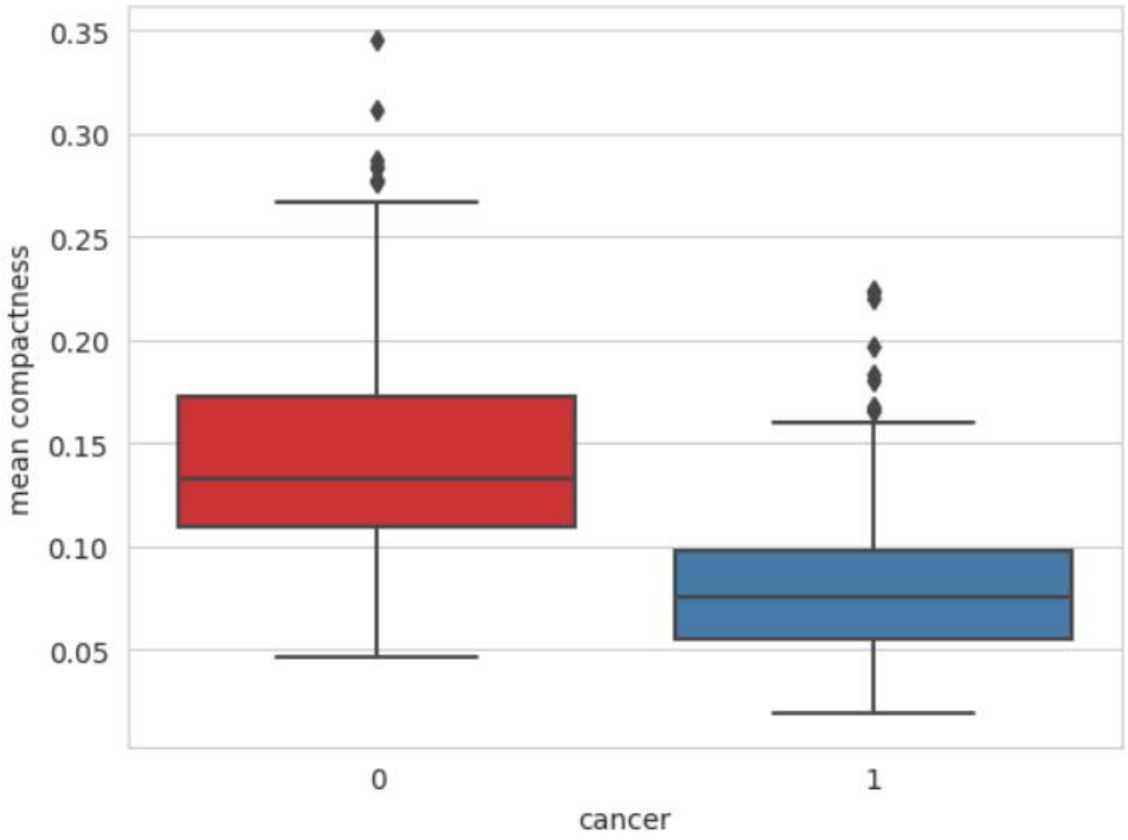


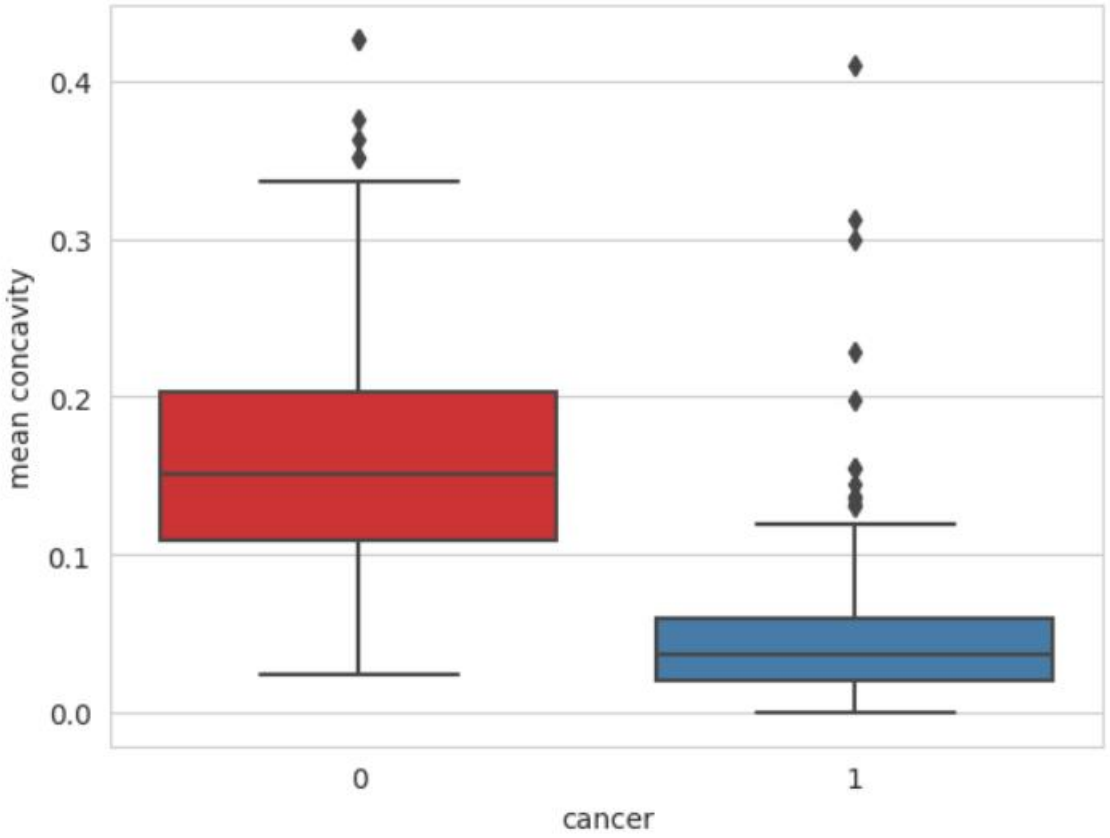

25
20
15
10
mean radius
0
1
cancer

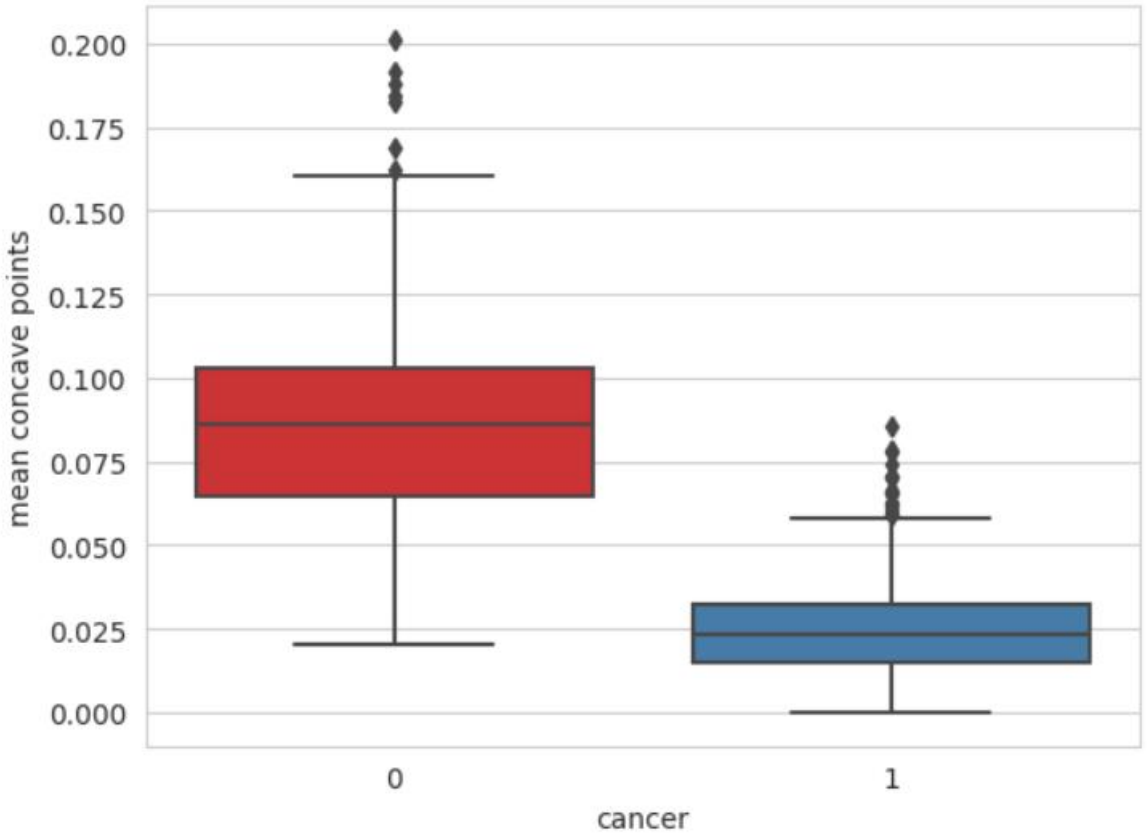

25
20
15
10
mean radius
0
1
cancer

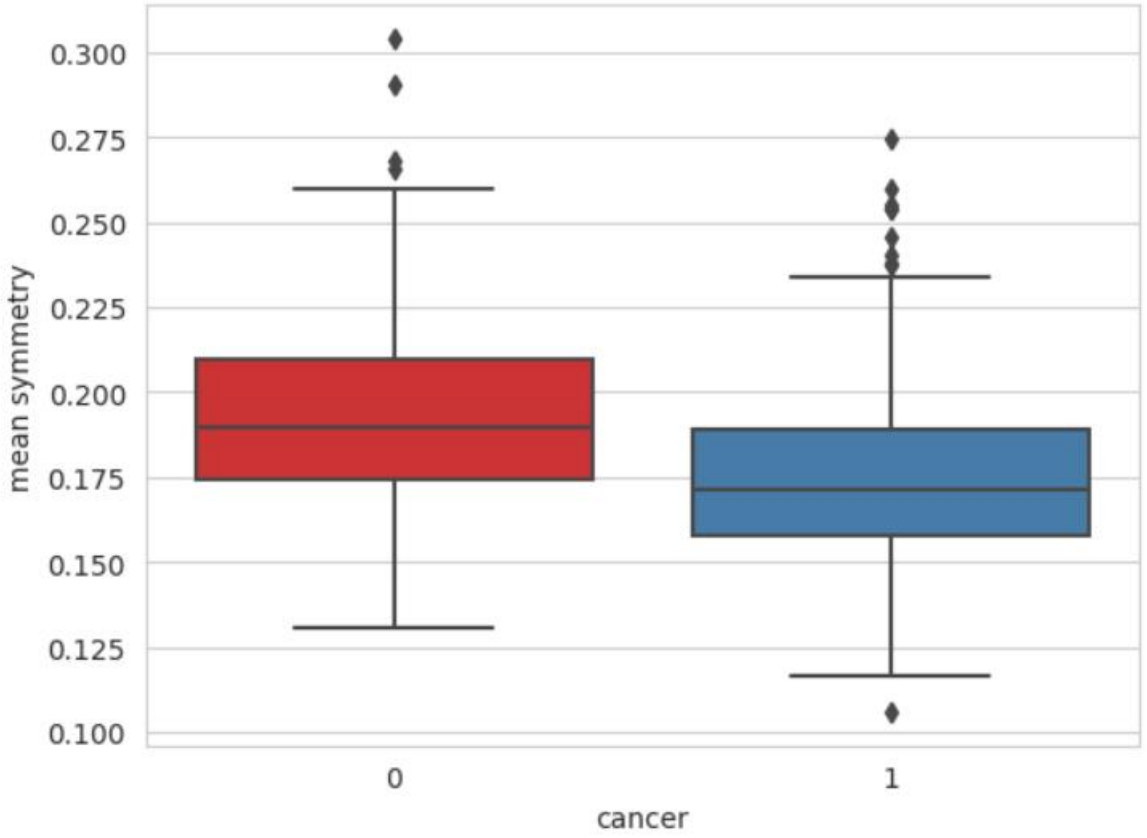

2500
2000
1500
1000
500
mean area
0
1
cancer


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

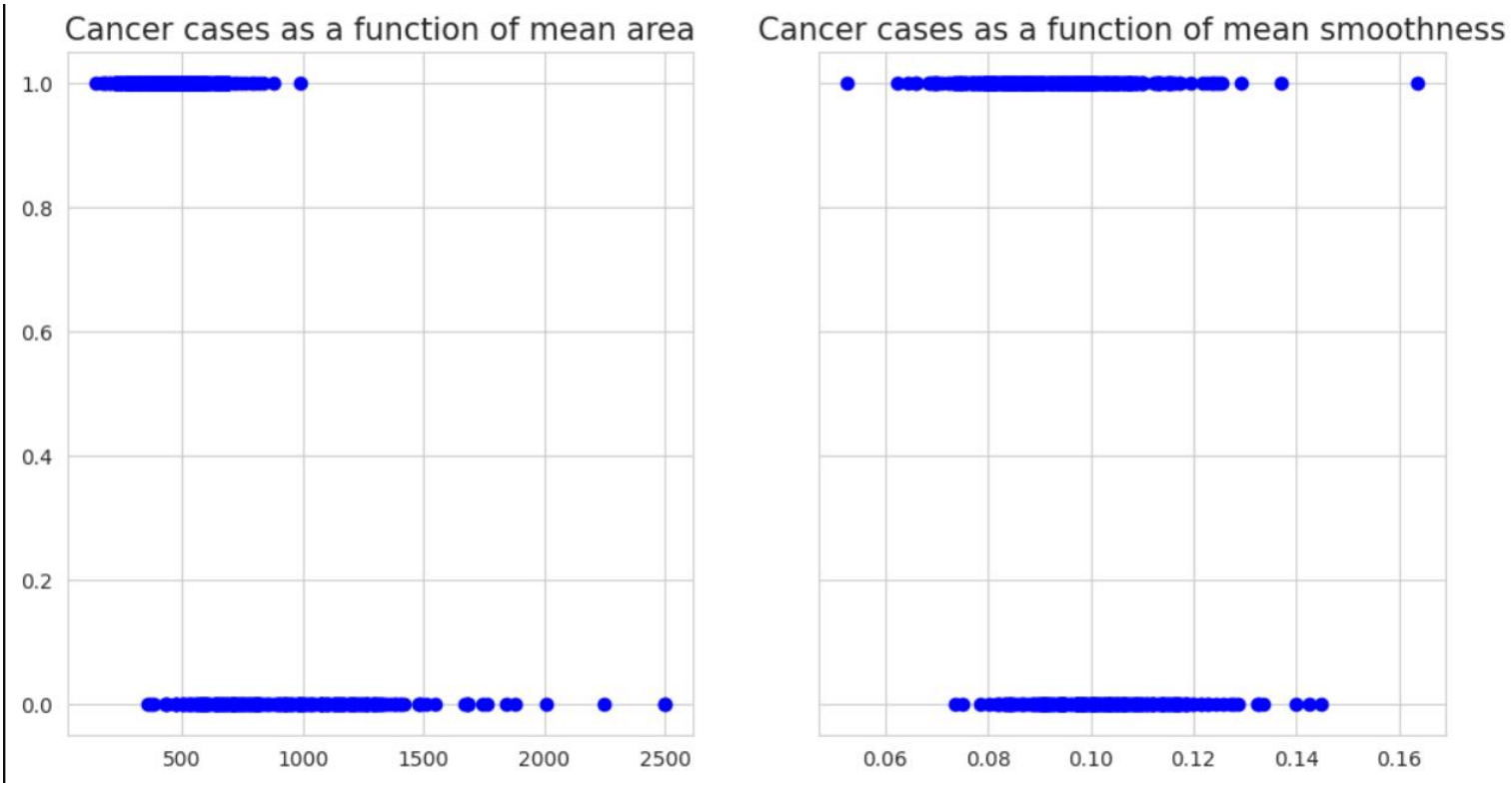

0.4
0.3
0.2
0.1
0.0
mean concavity
0
1
cancer






```
# 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'], color='blue')
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'], color='blue')
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	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...

5 rows × 30 columns

worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

```
# 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)
```

```
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()
```

```
# 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

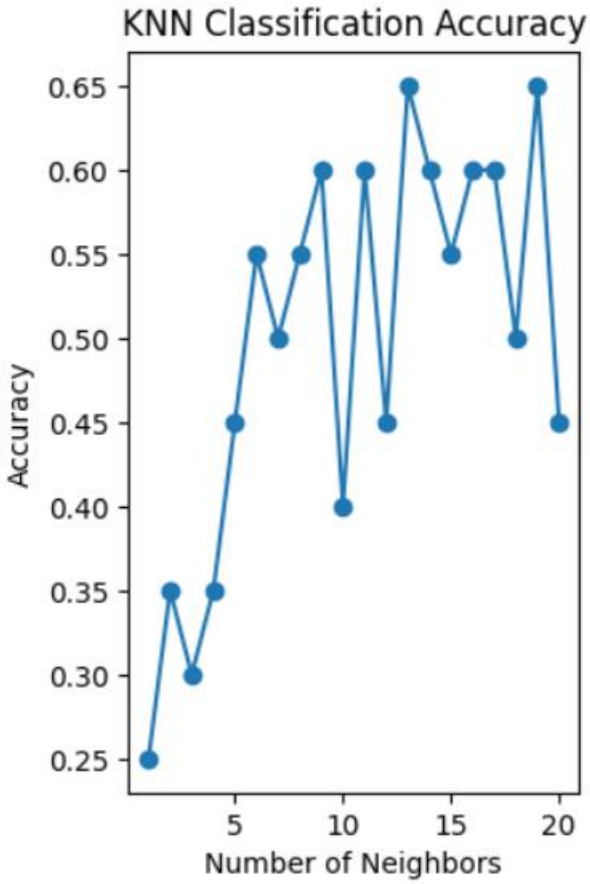
Non-Parametric Algorithm (KNN) for classification, regression, and analysis of different neighbors.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pyplot as plt
# Generate sample data
np.random.seed(42)
X = np.random.rand(100, 2)  # 100 data points with 2 features
y_classification = np.random.randint(0, 2, 100)  # Binary classification labels
y_regression = np.random.rand(100)  # Regression targets

# Split the data into training and testing sets
X_train, X_test, y_train_class, y_test_class, y_train_reg, y_test_reg = train_test_split(
    X, y_classification, y_regression, test_size=0.2, random_state=42
    )

# Define a range of neighbors to experiment with
neighbors = list(range(1, 21))

# Classification with KNN
accuracy_scores = []
for k in neighbors:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train_class)
    y_pred_class = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test_class, y_pred_class)
    accuracy_scores.append(accuracy)

# Regression with KNN
mse_scores = []
for k in neighbors:
    knn_regressor = KNeighborsRegressor(n_neighbors=k)
    knn_regressor.fit(X_train, y_train_reg)
    y_pred_reg = knn_regressor.predict(X_test)
    mse = mean_squared_error(y_test_reg, y_pred_reg)
    mse_scores.append(mse)

# Plot the results
plt.figure(figsize=(12, 6))
```

```
<Figure size 1200x600 with 0 Axes>
<Figure size 1200x600 with 0 Axes>
```

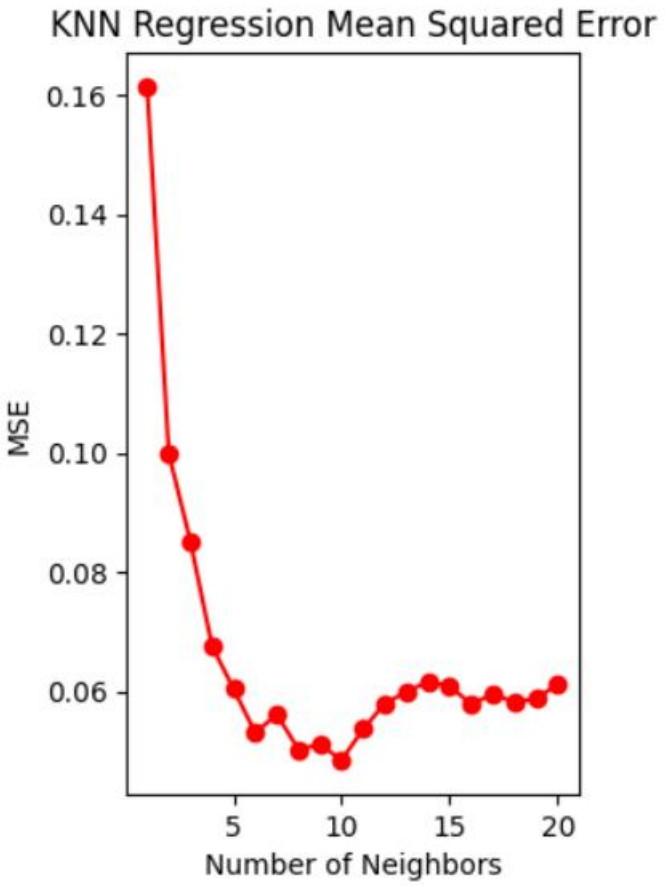
```
plt.subplot(1, 2, 1)
plt.plot(neighbors, accuracy_scores, marker='o')
plt.title('KNN Classification Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
```

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




```
plt.subplot(1, 2, 2)
plt.plot(neighbors, mse_scores, marker='o', color='r')
plt.title('KNN Regression Mean Squared Error')
plt.xlabel('Number of Neighbors')
plt.ylabel('MSE')
```

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




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

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