AIR QUALITY PREDICTION

A Course Project report submitted

in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

In

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

By

KOTHA. ABHINAYA SRI **2203A52241**

Under the guidance of

Mr. Eranki Kiran

Assistant Professor, Department of CSE.

Department of Computer Science and Artificial Intelligence

Department of Computer Science and Artificial Intelligence

CERTIFICATE

This is to certify that project entitled **"AIR QUALITY DATA"** is the bonafied work carried out by **K.Abhinaya Sri** as a Course Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING** during the academic year 2022-2023 under our guidance and Supervision.

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

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

ABSTRACT

Examining and protecting air quality has become one of the most essential activities for the government in many industrial and urban areas today. The meteorological and traffic factors, burning of fossil fuels, and industrial parameters play significant roles in air pollution. With this increasing air pollution, We are in need of implementing models which will record information about concentrations of air pollutants(so2,no2,etc).The deposition of this harmful gases in the air is affecting the quality of people's lives, especially in urban areas. Lately, many researchers began to use Big Data Analytics approach as there are environmental sensing networks and sensor data available. In this paper, machine learning techniques are used to predict the concentration of so2 in the environment. Sulphur dioxide irritates the skin and mucous membranes of the eyes, nose, throat, and lungs. Models in time series are employed to predict the so2 readings in nearing years or months. Contains the responses of a gas multisensor device deployed on the field in an Italian city. Hourly responses averages are recorded along with gas concentrations references from a certified analyzer.

Keywords: Machine Learning, Time Series, Prediction, Air Quality, SO2

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, Associate Professor** for encouragement, support and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved Dean, School of Computer Science and Artificial Intelligence, **Dr C. 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.

Table of Contents

CHAPTER-1

INTRODUCTION

1.1 Overview

Poor air quality can negatively affect human and environmental health. In humans, poor air quality can lead to a multitude of problems that include respiratory and cardiovascular diseases. We tend to think first of asthma and respiratory problems, but some particles are so small that they can enter the blood stream through the lungs and cause inflammation leading to issues beyond our breathing. In plants, poor air quality can also cause disease that can result in crop loss. In addition to human and environmental health, many pollutants that we worry about are greenhouse gases and contribute to climate change. Finally, poor air quality can impact quality of life. Consider visibility issues in National Parks and odors near industrial areas of cities; in addition to potential health dangers, these air quality issues can make daily life unpleasant.

Air is one of *the* foundational substances the human body needs to survive and thrive. Clean air keeps us healthy and feels good to breathe in. And so, we've put together this comprehensive resource on air quality. You'll learn everything from how air quality is measured to the harmful health and economic effects of poor air quality to some ways to improve the quality of the air you breathe, and so much more. Air is one of foundational substances the human body needs to survive and thrive. Clean air keeps us healthy and feels good to breathe in. And so, we've put together this comprehensive resource on air quality. You'll learn everything from how air quality is measured to the harmful health and economic effects of poor air quality to some ways to improve the quality of the air you breathe, and so much more.

As mentioned, air quality measures what kind of pollutants and how much of each is in the air you breathe. Poor air quality can lead to or contribute to all sorts of health problems - especially in groups sensitive to air pollutants - such as:

- Respiratory issues
- Cardiovascular dysfunction
- Reduced cognitive function
- Some cancers

Thus, air quality is a crucial consideration for many in these groups, as well as healthy individuals who want to minimize the chances anything in their body suffers harm.

Additionally, poor air quality can harm the economy in several ways, including but not limited to:

- Medical costs associated with poor air quality
- The human cost - deaths due to poor air quality
- Productivity losses
- Pollution remediation costs
- Negative food production impacts

Indoor air quality is especially critical. Americans spend nearly 90% of their time indores on average, according to the EPA. If you spend much more time inside your home, you're breathing in much more interior air.

That said, outdoor air quality remains an important factor in choosing a place to live, especially for those with respiratory issues.

There is much to consider when it comes to choosing a place to live. If you or someone in your family is in one of the aforementioned sensitive groups, air quality may be one of the more important matters to consider if you're moving. It may even be a reason you move from your current location if it's bad enough.

Various factors can impact outdoor air quality, from industry to weather and more. Consequently, each state has a different overall air quality rating. Even within those states, some cities do much better than others in terms of air pollution. Sometimes, a single city or area of the state may be known for poor air quality, whereas the rest of it isn't too bad.

Regardless, let's look at the best five and worst five states for air quality, based on the American Lung Association's (ALA) State of the Air 2021 report.

1.2 PROBLEM STATEMENT

We will make a project for prediction of Air Quality. The problem statement is that predicts what are diseases will cause for a person in future.

1.3 EXISTING SYSTEM

In the existing systems we only have data but there is no correct prediction of humidity so,our model can predict the above.

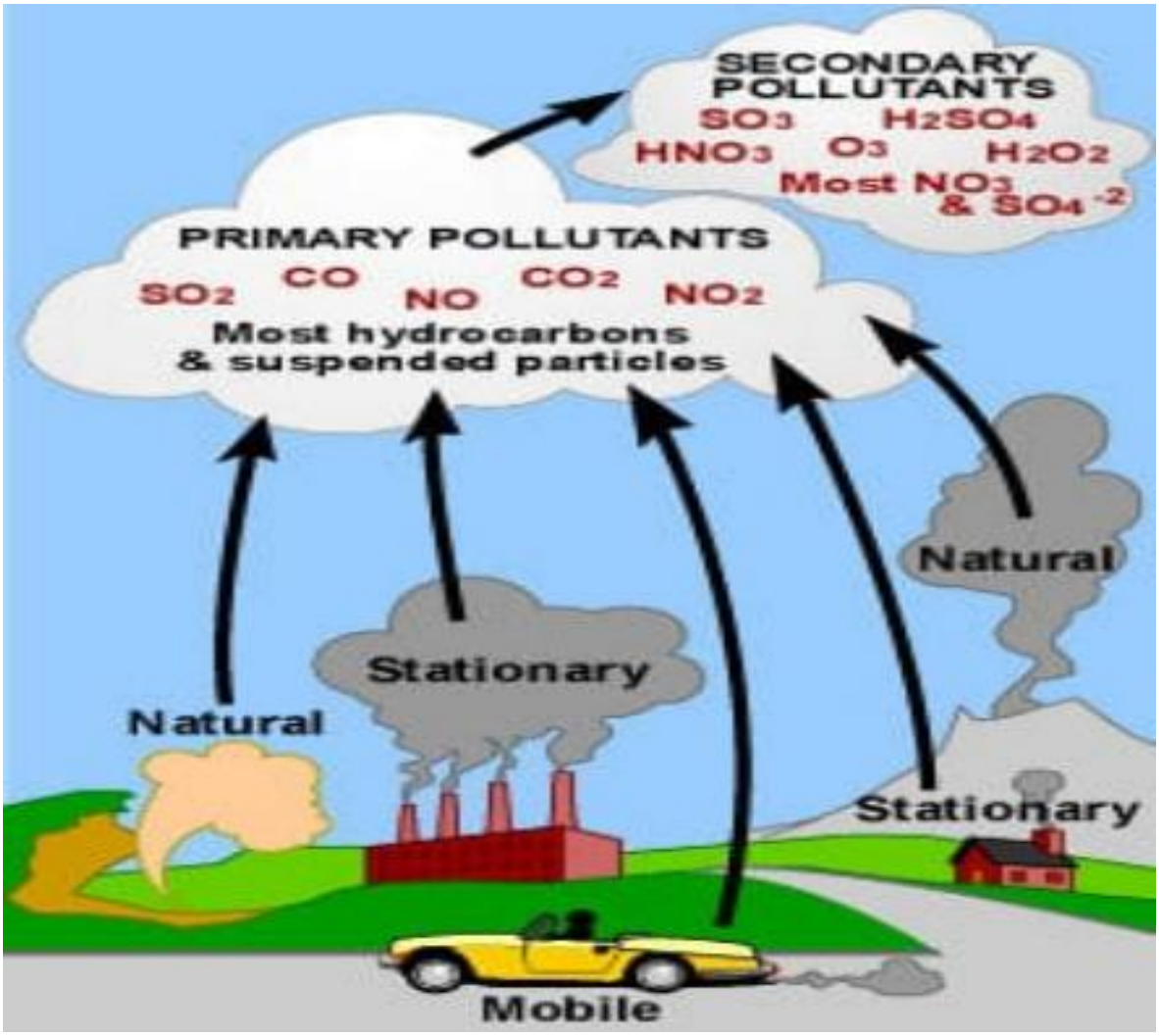
1.4 PROPOSED SYSTEM

I get this data from UCI Machine Learning. Here is about descripstion rows and column also another description.

"The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly

polluted area, at road level,within an Italian city. Data were recorded from March 2004 to February 2005 (one year)representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer. Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value.

1.6 ARCHITECTURE



CHAPTER-2
LITERATURE SURVEY

2.1 Document the survey done by you

With the advancement of IoT infrastructures, big data technologies, and machine learning techniques, real-time air quality monitor and evaluation is desirable for future smart cities. This paper reports our recent literature study, reviews and compares current research work on air quality evaluation based on big data analytics, machine learning models and techniques. Finally, it highlights some observations on future research issues, challenges, and needs.

We have formulated the problem as regularized MTL and employed advanced optimization algorithms for solving different formulations. We have focused on alleviating model complexity by reducing the number of model parameters and on improving the performance by using a structured regularizer. Our results show that the proposed light formulation achieves much better performance than the other two model formulations and that the regularization by enforcing prediction models for two consecutive hours to be close can also boost the performance of predictions. We have also shown that advanced optimization techniques are important for improving the convergence of optimization and that they speed up the training process for big data. For future work, we will further consider the commonalities between nearby meteorology stations and combine them in a MTL framework, which may provide a further boosting for the prediction.

Among the analysed works, 20 of them use open data to perform air quality predictions. These works were carried out from 2014 until now, coinciding with the movement of open data within the cities [70]. Therefore, we can affirm that the open data movement has increased the number of research works in the field of machine learning, especially in the prediction of air quality.

Related to the prediction target, the outcome shows that PM2.5 was the main element, applied in 19 papers, 18 of which utilised data of the cities located in China. Most cases, the authors performed a prediction for the next day. Twenty-seven studies used data hourly collected from the sensors.

In order to predict air quality, pm2_5 is also an important attribute. The values of this must be recorded in future as this particulates are responsible for various health effects including cardiovascular effects such as cardiac arrhythmias and heart attacks, and respiratoryeffects such as asthma attacks and bronchitis. This model further, also makes us aware of the challenges in future and research needs such as pm2.5,AQI,etc.

Most current studies directly predict air pollutants by inputting data for a fixed period of time in the future. The prediction results performed well in the short-term prediction such as 1h-predictions, but in the long-term prediction (up to or above 24 h-predictions), the time interval between forecasting and training will become very long, which results in lowering the temporal correlations between them. Thus the accuracy of the long-term prediction is generally low.

CHAPTER-3

DATA PRE-PROCESSING

2.1 Data Description

- **PM2.5-** PM2.5 has a significant influence on human health. And with the modern society developing, PM2.5 has been becoming a severe problem for people.

- **PM10-**High levels of PM10 can irritate the eyes and throat. Like PM2.5, increased symptoms can occur when exposed to PM10 if you have asthma or other lung diseases.

- **NO2** -Air quality has been the main concern worldwide and Nitrous oxide (NO2) is one of the pollutants that have a significant effect on human health and environment.

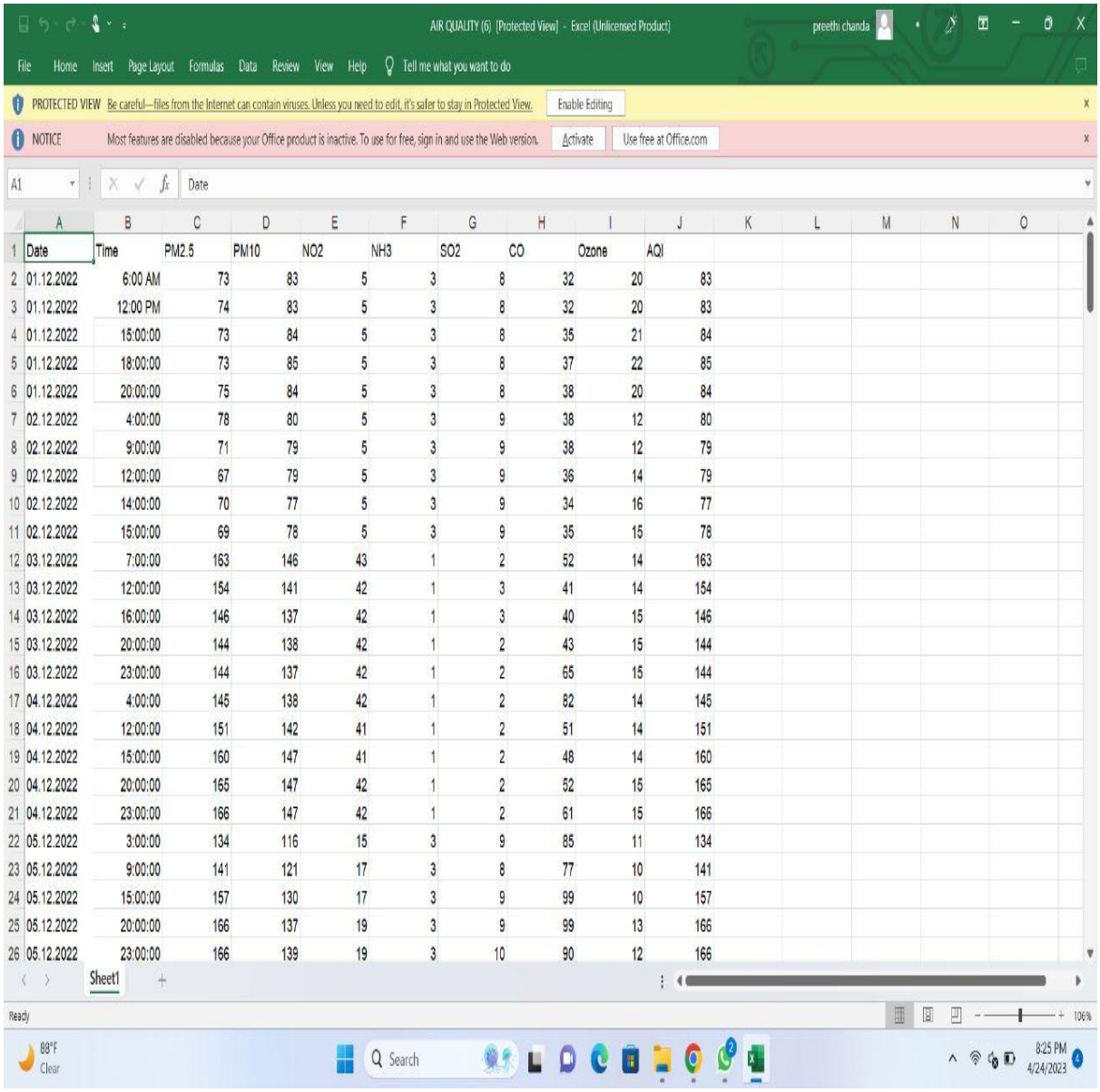
- **NH3**-Ammonia comes from the breakdown and volatilisation of urea. Emissions and deposition vary spatially with "emission hot-spots" associated with high-density intensive farming practices.

- **S02**-Sulfur dioxide (SO2) is a colorless, reactive air pollutant with a strong odor. This gas can be a threat to human health, animal health, and plant life. The main sources of sulfur dioxide emissions are from fossil fuel combustion and natural volcanic activity.

- **CO**-It is a colorless, odorless gas formed by the incomplete reaction of air with fuel.

- **Ozone**-Air quality regulators are concerned about ozone pollution because of its effects on public health and the environment.

DATASET



	A	B	C	D	E	F	G	H	I	J
1	Date	Time	PM2.5	PM10	NO2	NH3	SO2	CO	Ozone	AQI
2	01.12.2022	6:00 AM	73	83	5	3	8	32	20	83
3	01.12.2022	12:00 PM	74	83	5	3	8	32	20	83
4	01.12.2022	15:00:00	73	84	5	3	8	35	21	84
5	01.12.2022	18:00:00	73	85	5	3	8	37	22	85
6	01.12.2022	20:00:00	75	84	5	3	8	38	20	84
7	02.12.2022	4:00:00	78	80	5	3	9	38	12	80
8	02.12.2022	9:00:00	71	79	5	3	9	38	12	79
9	02.12.2022	12:00:00	67	79	5	3	9	36	14	79
10	02.12.2022	14:00:00	70	77	5	3	9	34	16	77
11	02.12.2022	15:00:00	69	78	5	3	9	35	15	78
12	03.12.2022	7:00:00	163	146	43	1	2	52	14	163
13	03.12.2022	12:00:00	154	141	42	1	3	41	14	154
14	03.12.2022	16:00:00	146	137	42	1	3	40	15	146
15	03.12.2022	20:00:00	144	138	42	1	2	43	15	144
16	03.12.2022	23:00:00	144	137	42	1	2	65	15	144
17	04.12.2022	4:00:00	145	138	42	1	2	82	14	145
18	04.12.2022	12:00:00	151	142	41	1	2	51	14	151
19	04.12.2022	15:00:00	160	147	41	1	2	48	14	160
20	04.12.2022	20:00:00	165	147	42	1	2	52	15	165
21	04.12.2022	23:00:00	166	147	42	1	2	61	15	166
22	05.12.2022	3:00:00	134	116	15	3	9	85	11	134
23	05.12.2022	9:00:00	141	121	17	3	8	77	10	141
24	05.12.2022	15:00:00	157	130	17	3	9	99	10	157
25	05.12.2022	20:00:00	166	137	19	3	9	99	13	166
26	05.12.2022	23:00:00	166	139	19	3	10	90	12	166

2.2 Data Cleaning

We have converted all our dataset strings into numerical values.

Contains the responses of a gas multisensory device deployed on the field in an italian city.

Hourly responses averages are recorded along with gas concentrations references from a certified analyzer.

Air pollution is considered to occur whenever harmful or excessive quantities of defined

substances such as gases , particulates , and biological molecules are introduce in to atmosphere.

With the economic and technological development of cities , environmental pollution problems are arising , such as water ,noise ,and ,air pollution . In particular, air pollution has a direct impact on human health through the exposure air of pollutants and particulates ,which has increased interest in air pollution.

The purpose of air quality can deliver substantial health benefits ; reducing air pollution levels means reducing premature deaths and diseases from stroke , heart diseases , lung cancer ,and both chronic and acute respiratory diseases including asthama.

The substances(NO2,CO,SO2) are in compositions of a common index is called air called indes(AQI), indicating how clean or polluted the air is currently or forecasted to become in areas.

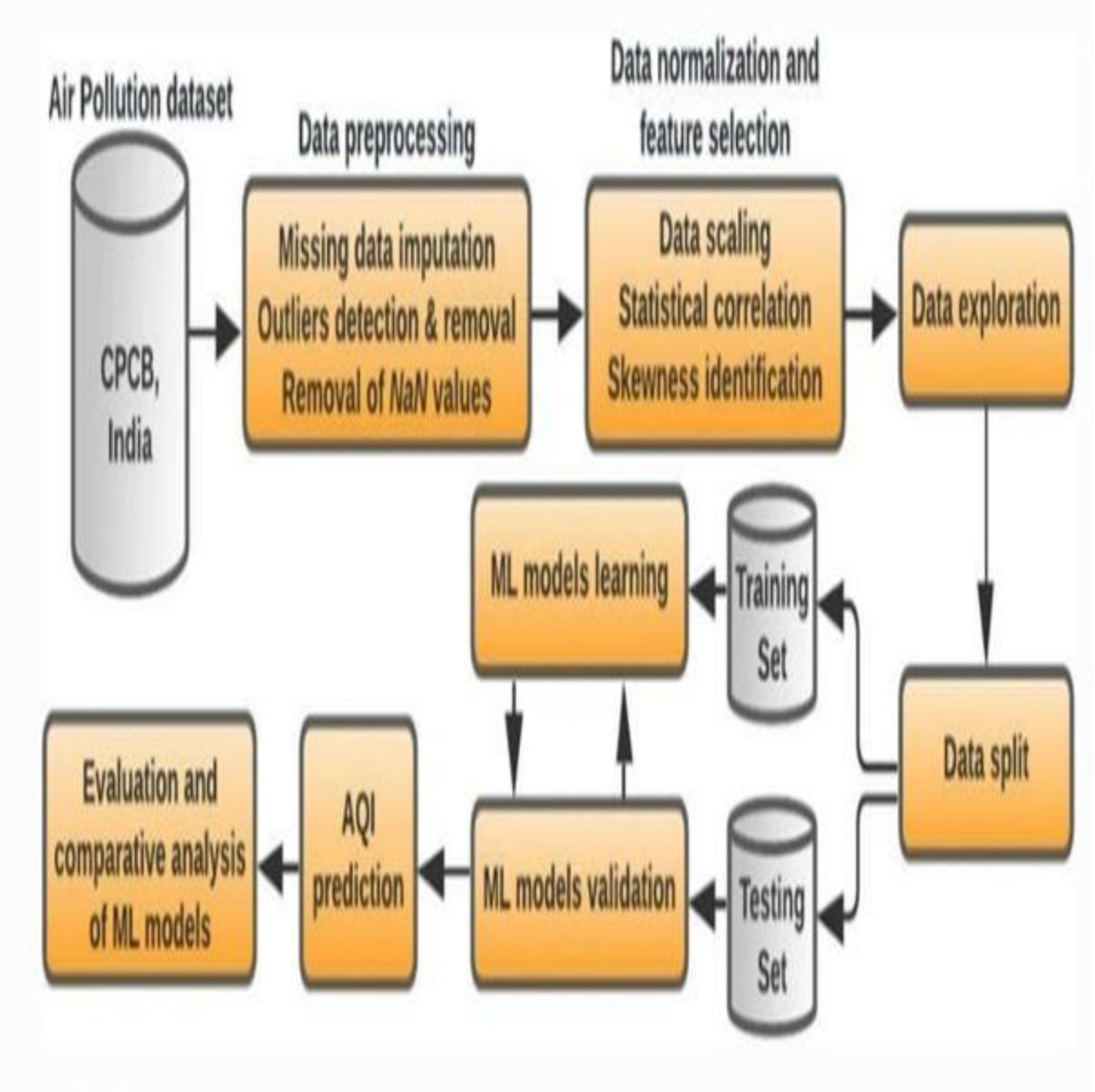
AQI is an index for reporting daily air quality .It tells you how clean or unhealthy.

what associated health effects might be a concern . The AQI focuses on health effects you may experience within a few hours or days after breathing unhealthy air.

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level,within an Italian city.

Data cleaning

10

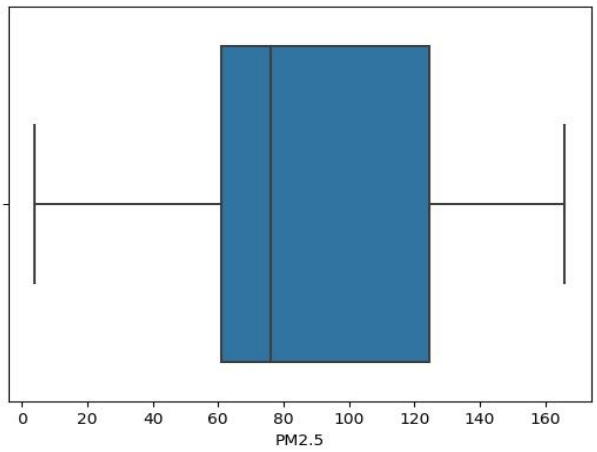


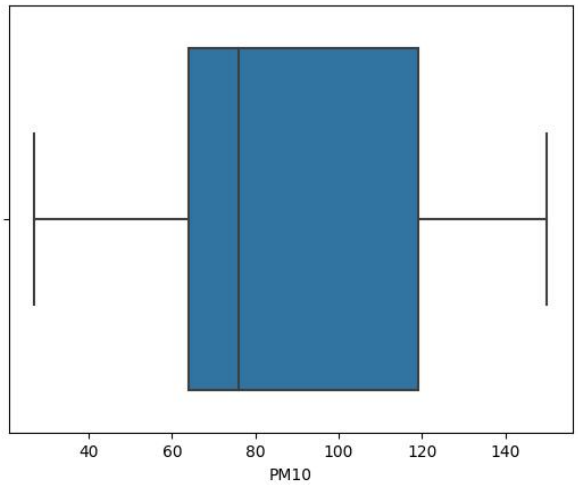
2.3-Data visualisation

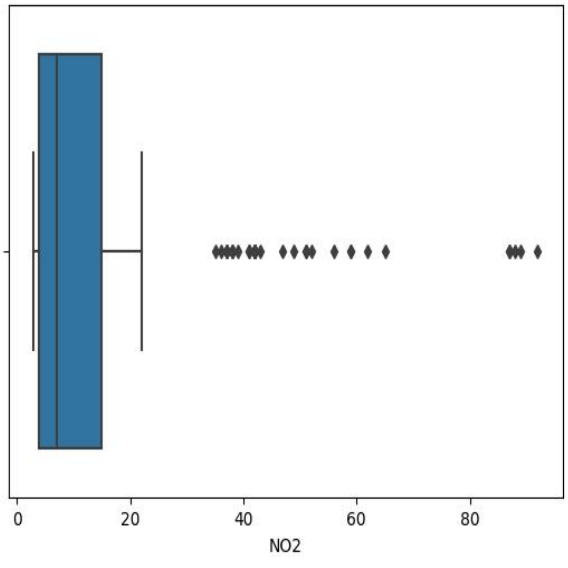
BOX PLOT

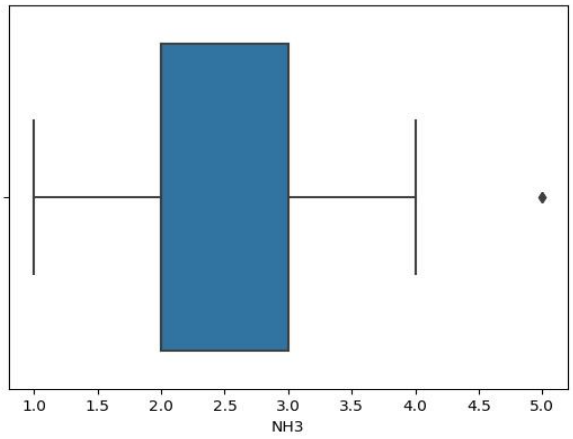
- A box and whisker plot—also called a box plot—displays the five-number summary of a set of data. The five number summary is the minimum, first quartile, median, third quartile, and maximum.
- In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum.
- Box plots are used to show distributions of numeric data values, especially when you want to compare them between multiple groups. They are built to provide highlevel information at a glance, offering general information about a group of data's symmetry, skew, variance, and outliers.

- A boxplot is a way to show a five number summary in a chart. The main part of the chart (the "box") shows where the middle portion of the data is: the interquartile range. At the ends of the box, you" find the first quartile (the 25% mark) and the third quartile (the 75% mark). The far left of the chart (at the end of the left "whisker") is the minimum (the smallest number in the set) and the far right is the maximum (the largest number in the set). Finally, the median is represented by a vertical bar in the center of the box.

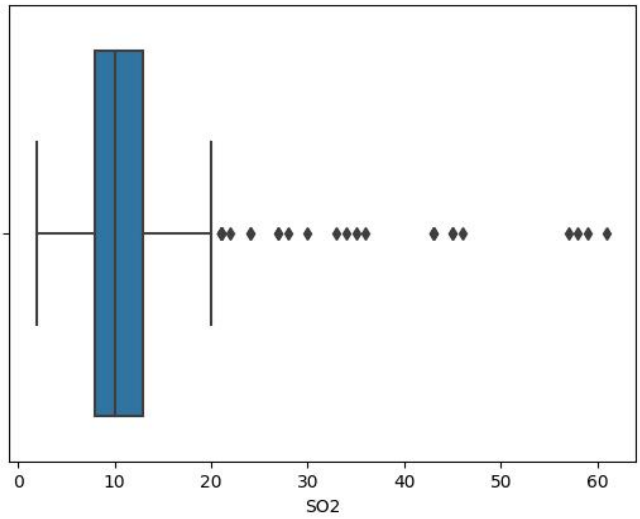


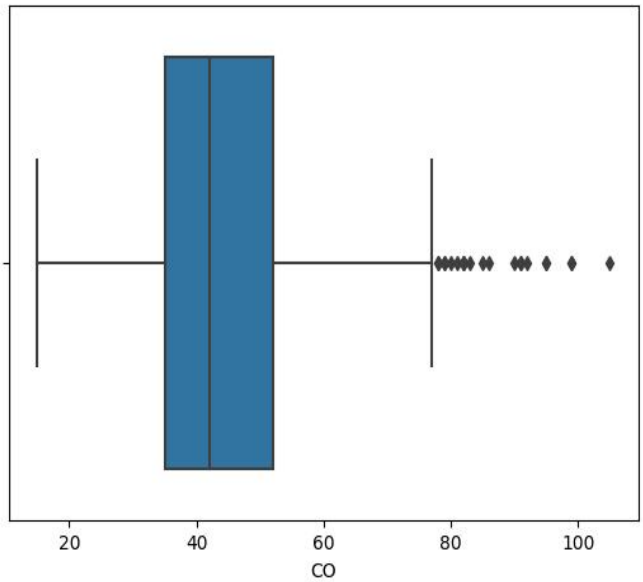




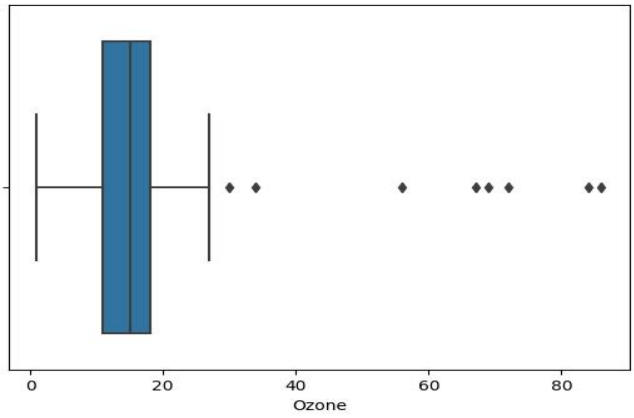

1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
NH3

12


0
10
20
30
40
50
60
SO2

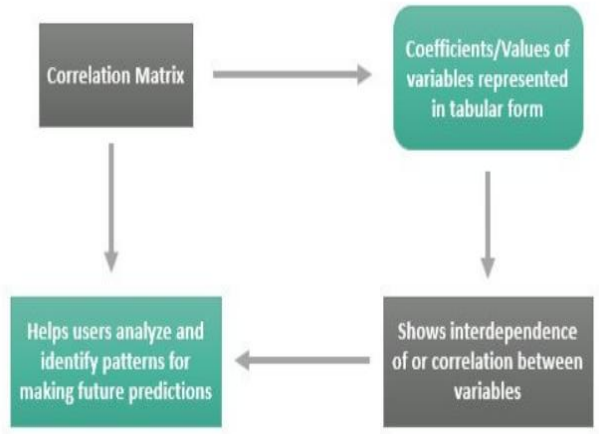

20
40
60
80
100
CO

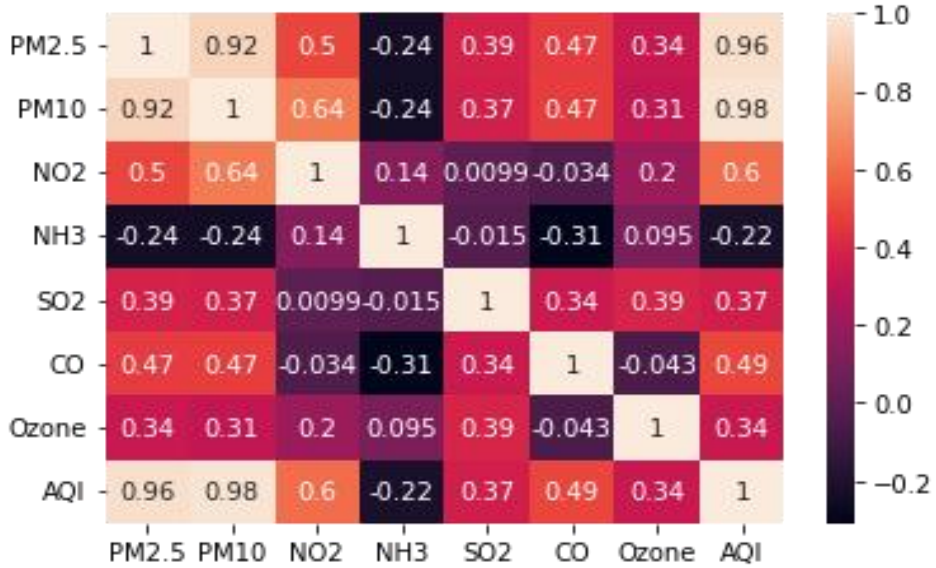
21



CORRELATION MATRIX

Correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It's a common tool for describing simple relationships without making a statemen about cause and effect. A correlation matrix is simply a table showing the correlation coefficients between variables.



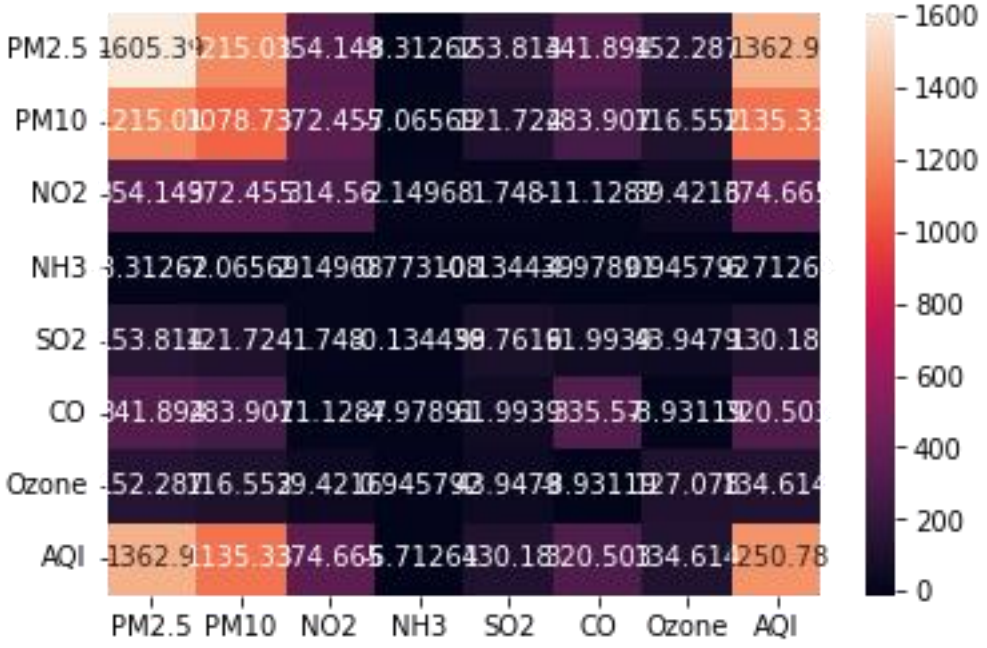


COVARIANCE MATRIX

Covariance matrix is a square matrix (has the same number of rows and columns) that gives the covariance between each pair of elements available in the data. Covariance measures the extent to which to variables move in the same direction.

Covariance matrix is a type of matrix that is used to represent the covariance values between pairs of elements given in a random vector. The covariance matrix can also be referred to as the variance covariance matrix. This is because the variance of each element is represented along the main diagonal of the matrix.

A covariance matrix is always a square matrix. Furthermore, it is positive semi-definite, and symmetric. This matrix is very useful in stochastic modeling and principle component analysis. In this article, we will learn about the variance covariance matrix, its formula, examples, and various important properties associated with it.



CHAPTER-4

METHADOLOGY

1.1Procedure to solve a given problem

- We collected the real time values of the gases which we used in our dataset.
- We choosed some specific area and collected the data of the gases involved in air quality prediction. The main gases which are responsible in damaging the air are CO,NO2 and SO2.
- We calculated and plotted the corelation and covariance matrix.
- Ours is a regression model, in our dataset we have 10 attributes.
- AQI is the y value and all others are x values. We are predicting the y value based on the x values.
- Contains the responses of a gas multisensory device deployed on the field in an italian city. Hourly responses averages are recorded along with gas concentrations references from a certified analyzer.
- Air pollution is considered to occur whenever harmful or excessive quantities of defined substances such as gases particulates , and biological molecules are introduce in to atmosphere.
- The substances(NO2,CO,SO2) are in compositions of a common index is called air called indeces(AQI) indicating how clean or polluted the air is currently or forecasted to become in areas.
- The purpose of air quality can deliver substantial health benefits ; reducing air pollution levels means reducing premature deaths and diseases from stroke , heart diseases , lung cancer ,and both chronic and acute respiratory diseases including asthama.
- With the economic and technological development of cities , environmental pollution problems are arising , such as water ,noise ,and ,air pollution . In particular, air pollution has a direct impact on human health through the exposure air of pollutants and particulates ,which has increased interest in air pollution.

K-NEAREST NEIGHBOUR

- K-nearest neighbor (KNN) is a non-parametric algorithm that can be used for classification and regression tasks. It works by finding the k closest data points in the training set to a given test point and then using the labels or values of those data points to predict the label or value of the test point.
- To use KNN for air quality prediction, you would first need to collect data on the air quality at various locations and times. This data could include measurements of pollutants such as particulate matter (PM), ozone (O3), nitrogen dioxide (NO2), and sulfur dioxide (SO2), as well as temperature, humidity, wind speed, and other relevant variables.
- It is important to note that KNN assumes that similar data points have similar values, which may not always be the case for air quality data. Additionally, KNN can be computationally expensive, especially for large datasets, so it may not be the best choice for real-time air quality prediction. Other machine learning algorithms such as decision trees, random forests, and neural networks may also be useful for air quality prediction.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

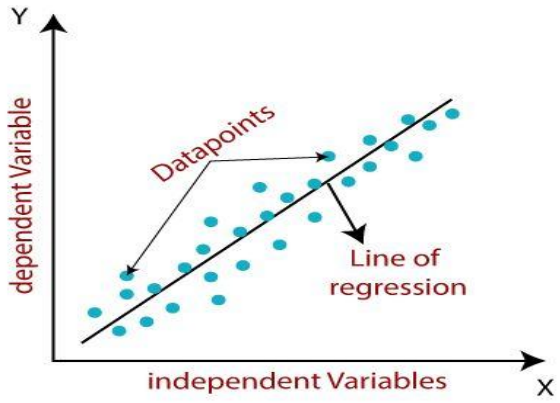
Calculations:

- Mean Square Error:25.623478260869575
- Mean Absolute Error:3.2913043478260877
- Root Mean Square Error:5.061963873919842

LINEAR REGRESSION

- Linear regression is a statistical technique used to analyze the relationship between a dependent variable (Y) and one or more independent variables (X). In the context of air quality prediction, linear regression can be used to model the relationship between air quality measurements (such as particulate matter concentration or ozone levels) and various environmental factors, such as temperature, humidity, and wind speed.
- To use linear regression for air quality prediction, you would first need to collect data on the air quality measurements and environmental factors of interest. This data could be collected using sensors installed at various locations throughout a city or region, or by collecting data from existing monitoring stations.

- It is important to note that linear regression is just one of many statistical techniques that can be used for air quality prediction.

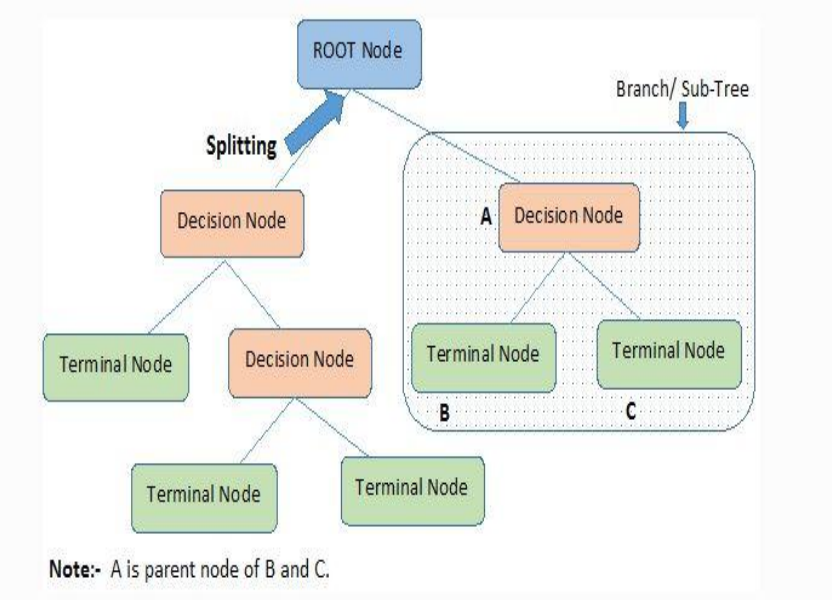


Calculations:

- Mean Square Error: 20.186750203222648
- Mean Absolute Error: 3.6098028860602294
- Root Mean Absolute Error: 4.492966748510682

DECISION TREE

- Decision tree algorithm is a computational method used to build decision trees from data. The goal of the algorithm is to identify the most important variables and their values that are predictive of a particular outcome or class.
- Once the decision tree is built, it can be used to predict the outcome or class of new data based on the values of the variables. This is done by traversing the tree from the root node to a leaf node, where each internal node corresponds to a test on a variable and each leaf node corresponds to a prediction of the outcome or class.



Calculations:

- Mean Square Error:12.021739130434783
- Mean Absolute Error:2.108695652173913
- Root Mean Square Error:3.467237968532703

SUPPORT VECTOR MACHINE

- Support Vector Machine (SVM) is a popular machine learning algorithm that can be used for air quality prediction. SVMs are particularly effective in handling non-linear relationships between variables and have been successfully applied to various environmental prediction problems, including air quality.
- To use SVM for air quality prediction, you would first need to collect data on the air quality measurements and environmental factors of interest. This data could be collected using sensors installed at various locations throughout a city or region or by collecting data from existing monitoring stations.

Calculations:

- **Mean Square Error: 24.142612927396755**
- **Mean Absolute Error:3.6699528174496265**
- **Random Mean square Error:4.9135132977734735**

RANDOM FOREST

- Random Forest is a popular machine learning algorithm used for predictive modeling in a wide range of applications, including air quality prediction. Random Forest is an ensemble learning method that combines multiple decision trees to improve the accuracy and generalization of the model.
- In the context of air quality prediction, Random Forest can be used to model the relationship between various air pollutants, meteorological variables, and other environmental factors, and their impact on air quality. The algorithm works by building multiple decision trees on random subsets of the data and combining their predictions to produce a final prediction.
- Overall, Random Forest is a powerful algorithm for air quality prediction that can take into account multiple variables and their complex interactions.

Calculations:

- MeanSquareError:6.253336956521742
- Mean Absolute Error:1.6089130434782613
- Root Mean Suqare Error2.500667302245891

CODE:

```
# Import the pandas library and alias it as 'pd' for convenience.
import pandas as pd
d=pd.read_excel('/content/AIR QUALITY (6).xlsx')
print(d) # Print the contents of the DataFrame 'd' to the console.
```

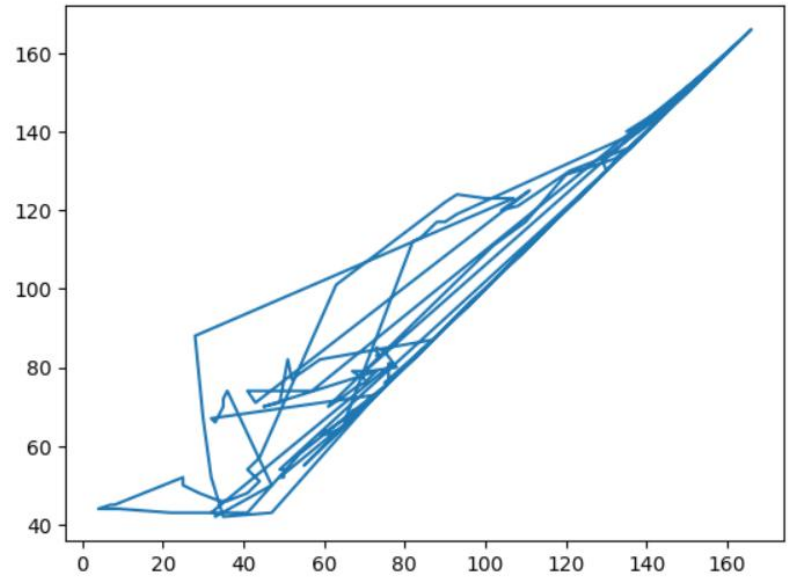
```
            Date      Time  PM2.5  PM10  NO2  NH3  SO2  CO  Ozone  AQI
0     01.12.2022  06:00:00     73    83    5    3    8  32     20   83
1     01.12.2022  12:00:00     74    83    5    3    8  32     20   83
2     01.12.2022  15:00:00     73    84    5    3    8  35     21   84
3     01.12.2022  18:00:00     73    85    5    3    8  37     22   85
4     01.12.2022  20:00:00     75    84    5    3    8  38     20   84
..           ...       ...    ...   ...  ...  ...  ...  ..    ...  ...
222   14.01.2023  14.00.00    129   119    9    2   17  81     18  129
223   14.01.2023  20.00.00    119   114    9    2   16  69     18  119
224   14.01.2023  23.00.00    120   113    9    2   16  72     17  120
225   15.01.2023   9.00.00    136   113    6    2   16  76     16  136
226   15.01.2023  12.00.00    137   113    6    2   16  79     17  137

[227 rows x 10 columns]
```

```
print(d.isnull)
```

```
<bound method DataFrame.isnull of           Date      Time  PM2.5  PM10  NO2  NH3  SO2  CO  Ozone  AQI
0    01.12.2022  06:00:00     73    83    5    3    8  32     20   83
1    01.12.2022  12:00:00     74    83    5    3    8  32     20   83
2    01.12.2022  15:00:00     73    84    5    3    8  35     21   84
3    01.12.2022  18:00:00     73    85    5    3    8  37     22   85
4    01.12.2022  20:00:00     75    84    5    3    8  38     20   84
..          ...       ...    ...   ...  ...  ...  ...  ..    ...  ...
222  14.01.2023  14.00.00    129   119    9    2   17  81     18  129
223  14.01.2023  20.00.00    119   114    9    2   16  69     18  119
224  14.01.2023  23.00.00    120   113    9    2   16  72     17  120
225  15.01.2023   9.00.00    136   113    6    2   16  76     16  136
226  15.01.2023  12.00.00    137   113    6    2   16  79     17  137

[227 rows x 10 columns]>
```

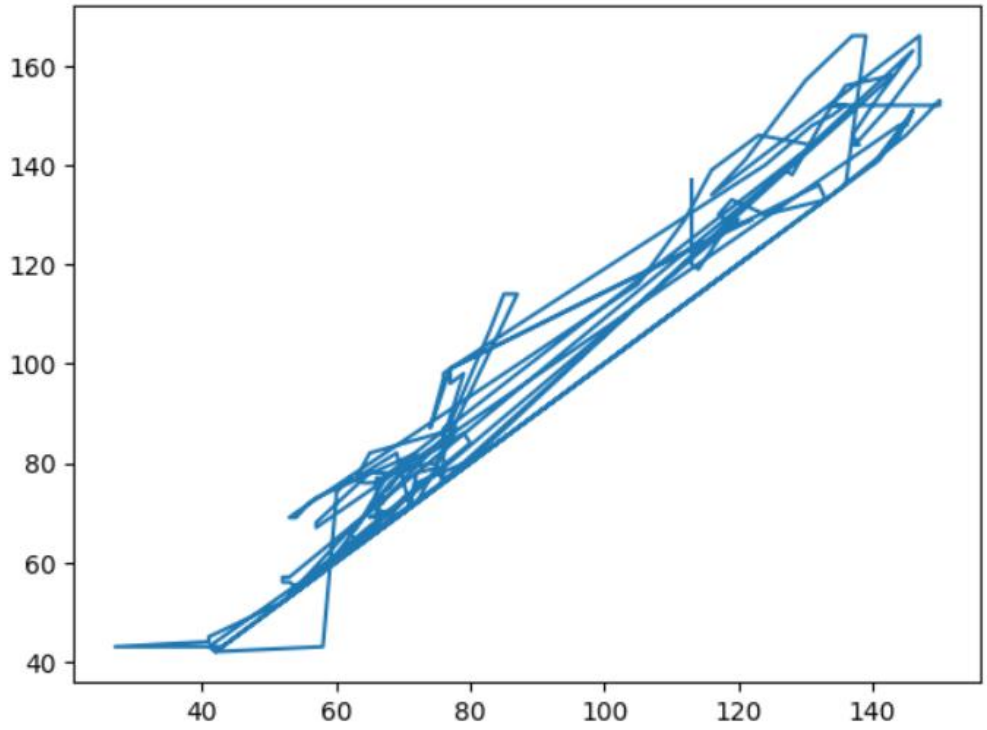
```
# Extract the 'PM2.5' data from the dictionary 'd' and store it in the variable 'x1'.
x1=d['PM2.5']
# Extract the 'AQI' data from the dictionary 'd' and store it in the variable 'y'.
y=d['AQI']
import matplotlib.pyplot as plt
plt.plot(x1,y)
```

```
[<matplotlib.lines.Line2D at 0x7f106e5c63d0>]
```



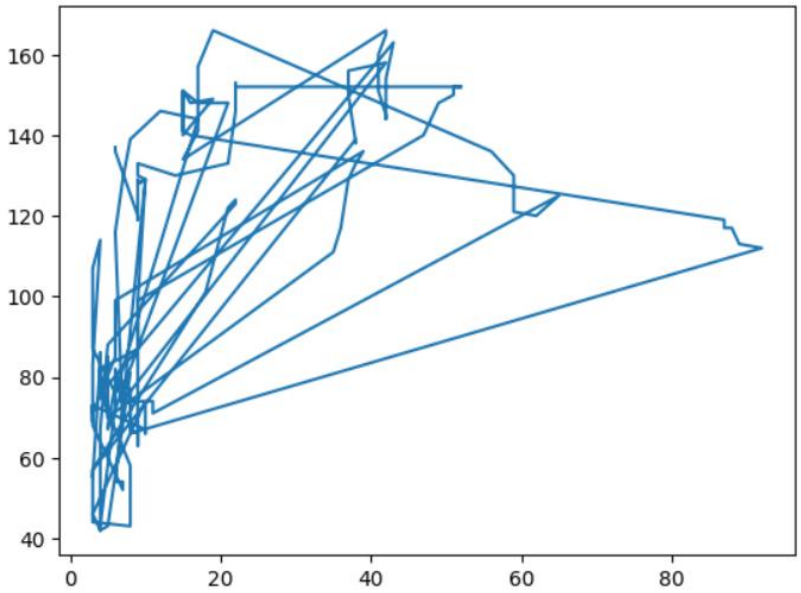

```
x2=d['PM10'] # Extract the 'PM10' data from the dictionary 'd' and store it in the variable 'x2'.
import matplotlib.pyplot as plt
plt.plot(x2,y) # Create a line plot using 'x2' as the x-axis data and 'y' as the y-axis data.
```

```
[<matplotlib.lines.Line2D at 0x7f106e53abb0>]
```




```
x3=d['NO2'] # Extract the 'NO2' data from the dictionary 'd' and assign it to the variable 'x3'.
import matplotlib.pyplot as plt
plt.plot(x3,y) # Create a line plot using the data in 'x3' and 'y'.
```

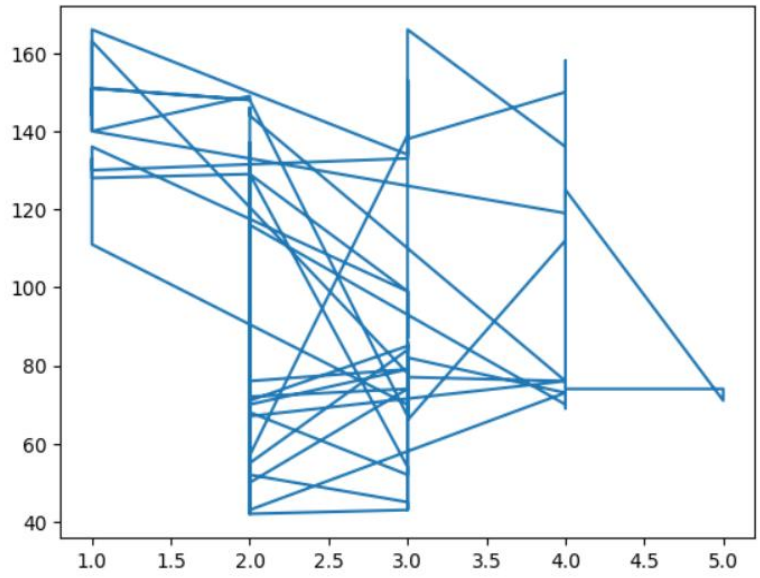
```
[<matplotlib.lines.Line2D at 0x7f10718475b0>]
```




```
x4=d['NH3'] # Extract the 'NH3' data from the dictionary 'd' and assign it to the variable 'x4'.
import matplotlib.pyplot as plt
```

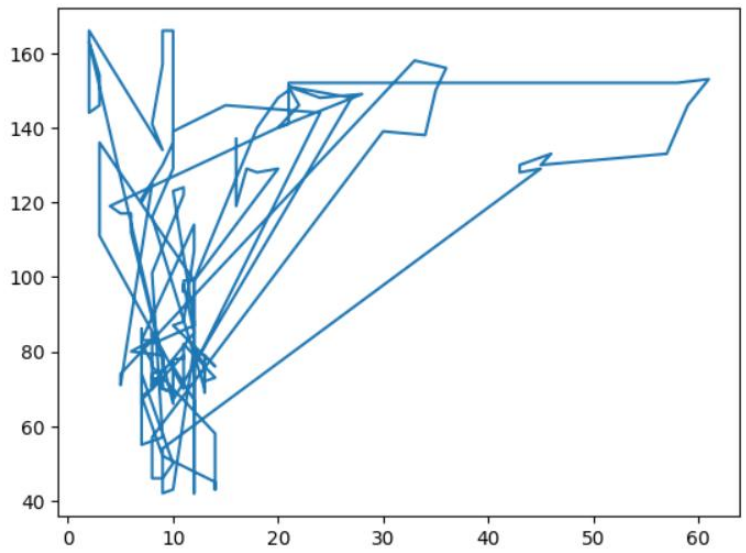
```
plt.plot(x4,y) # Plot the data in 'x4' against the values in 'y'.
```

```
[<matplotlib.lines.Line2D at 0x7f106e499d90>]
```



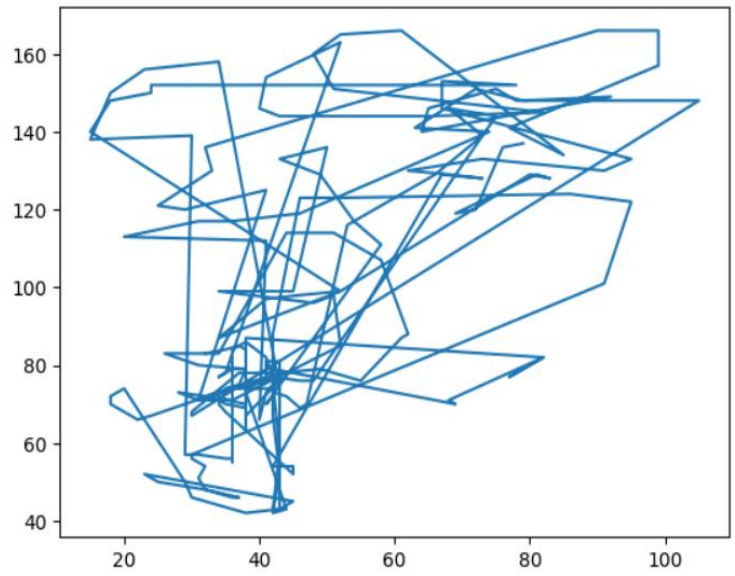

```
x5=d['SO2'] # Extract the 'SO2' data from the dictionary 'd' and assign it to the variable 'x5'
import matplotlib.pyplot as plt
plt.plot(x5,y) # Create a line plot using 'x5' as the x-axis and 'y' as the y-axis
```

```
[<matplotlib.lines.Line2D at 0x7f106e416be0>]
```



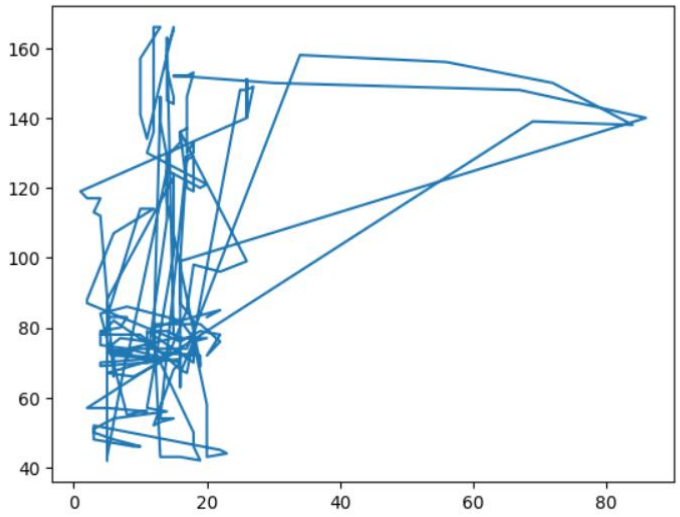

```
x6=d['CO'] # Retrieve the 'CO' data from the dictionary 'd' and store it in the variable 'x6'.
import matplotlib.pyplot as plt
plt.plot(x6,y)
```

```
[<matplotlib.lines.Line2D at 0x7f1072fc5d00>]
```



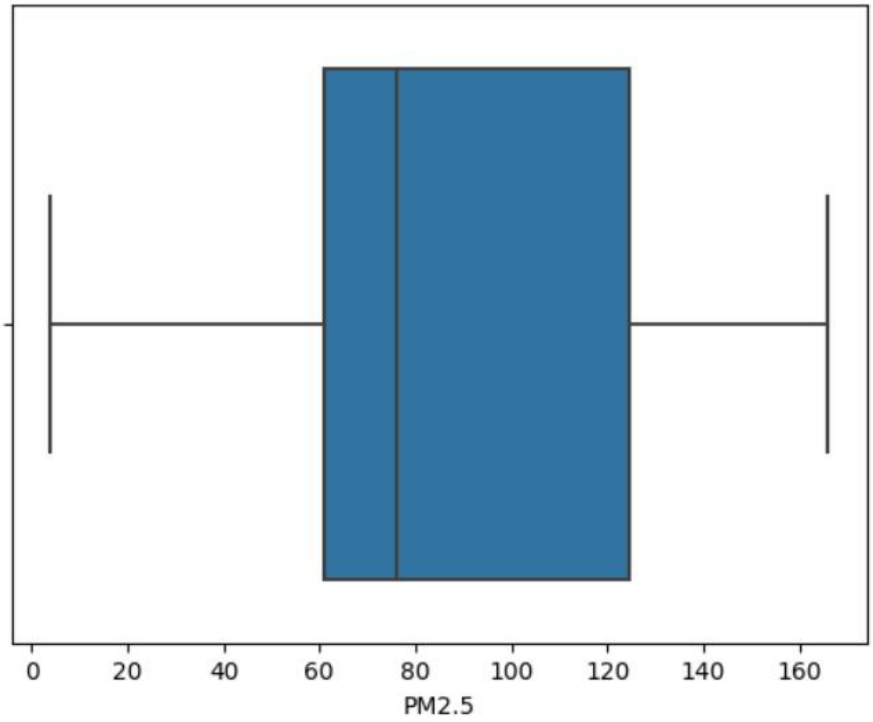

```
# Extract the 'Ozone' data from the dictionary 'd' and store it in the variable 'x7'
x7=d['Ozone']
import matplotlib.pyplot as plt
plt.plot(x7,y)
```

```
[<matplotlib.lines.Line2D at 0x7f1074d362b0>]
```



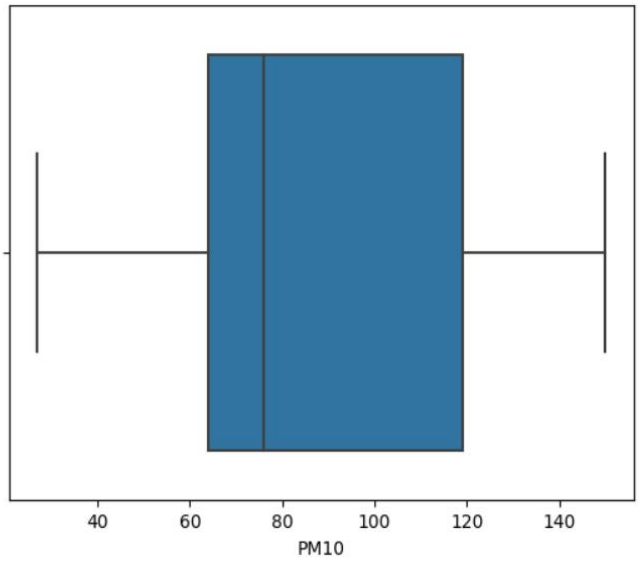

```
import seaborn as sns
# Create a boxplot using Seaborn to visualize the distribution of the 'PM2.5' variable.
sns.boxplot(data=d,x='PM2.5')
```

```
<Axes: xlabel='PM2.5'>
```



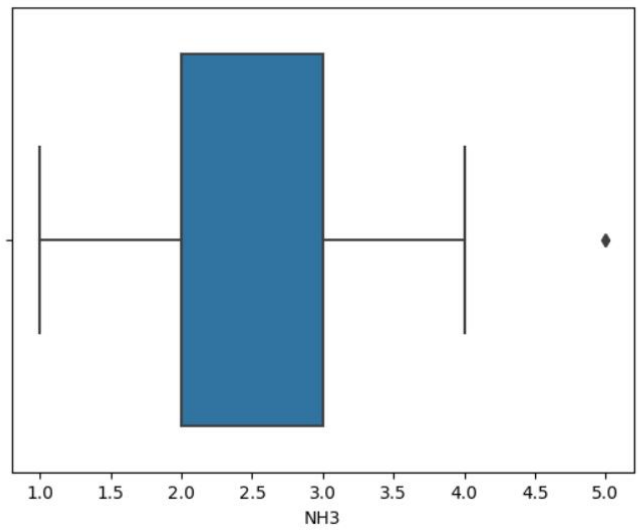

```
import seaborn as sns
# Create a boxplot using Seaborn to visualize the distribution of the 'PM10' variable.
sns.boxplot(data=d,x='PM10')
```

```
<Axes: xlabel='PM10'>
```



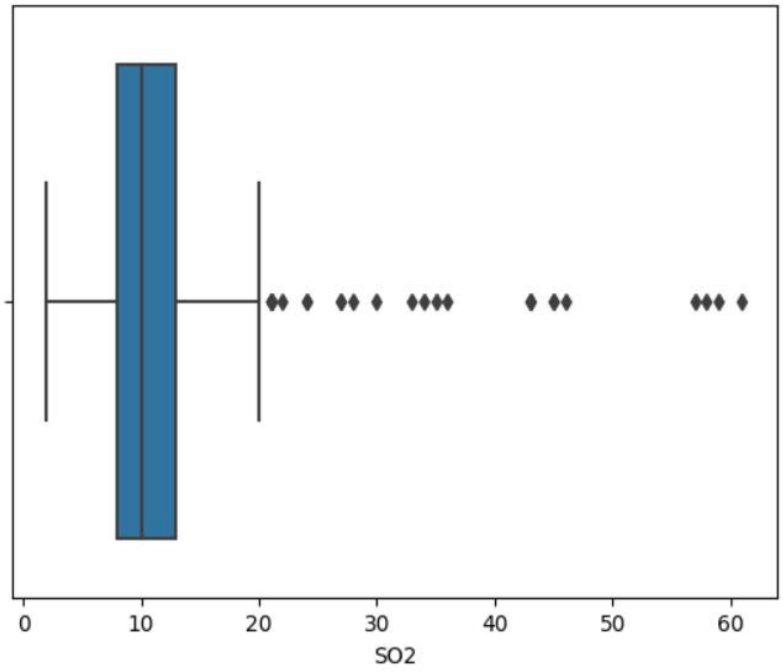

```
import seaborn as sns
# Create a boxplot using Seaborn to visualize the distribution of the 'NH3' variable.
sns.boxplot(data=d,x='NH3')
```

```
<Axes: xlabel='NH3'>
```



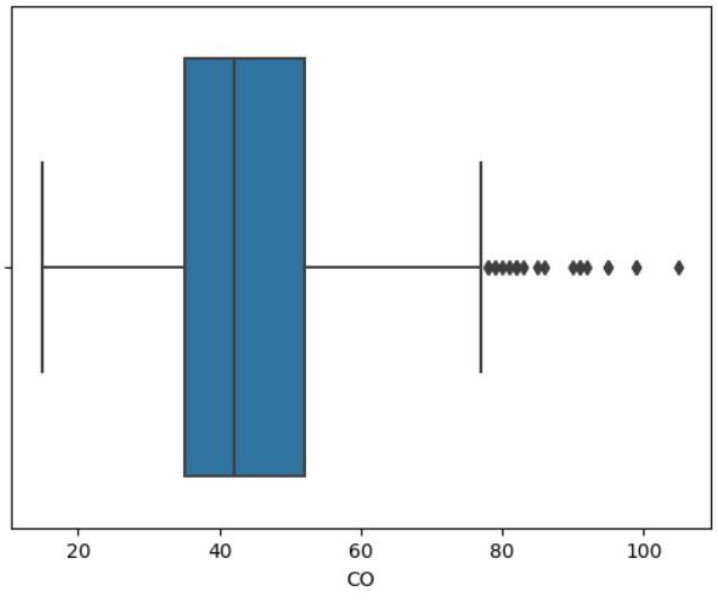

```
import seaborn as sns
# Create a boxplot using Seaborn to visualize the distribution of the 'SO2' variable
sns.boxplot(data=d,x='SO2')
```

```
<Axes: xlabel='SO2'>
```



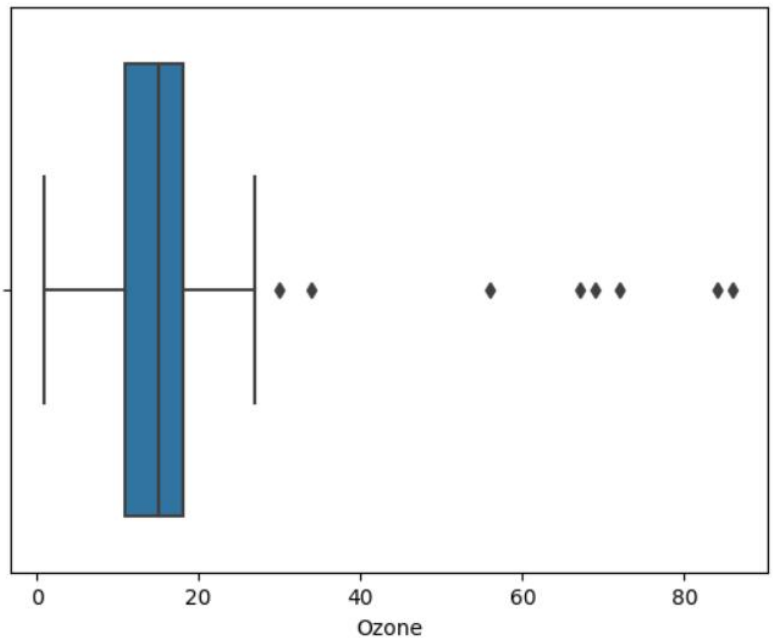

```
import seaborn as sns
# Create a boxplot using Seaborn to visualize the distribution of 'CO' data
sns.boxplot(data=d,x='CO')
```

```
<Axes: xlabel='CO'>
```




```
import seaborn as sns
# Create a boxplot using the 'boxplot' function from seaborn.
sns.boxplot(data=d,x='Ozone')
```

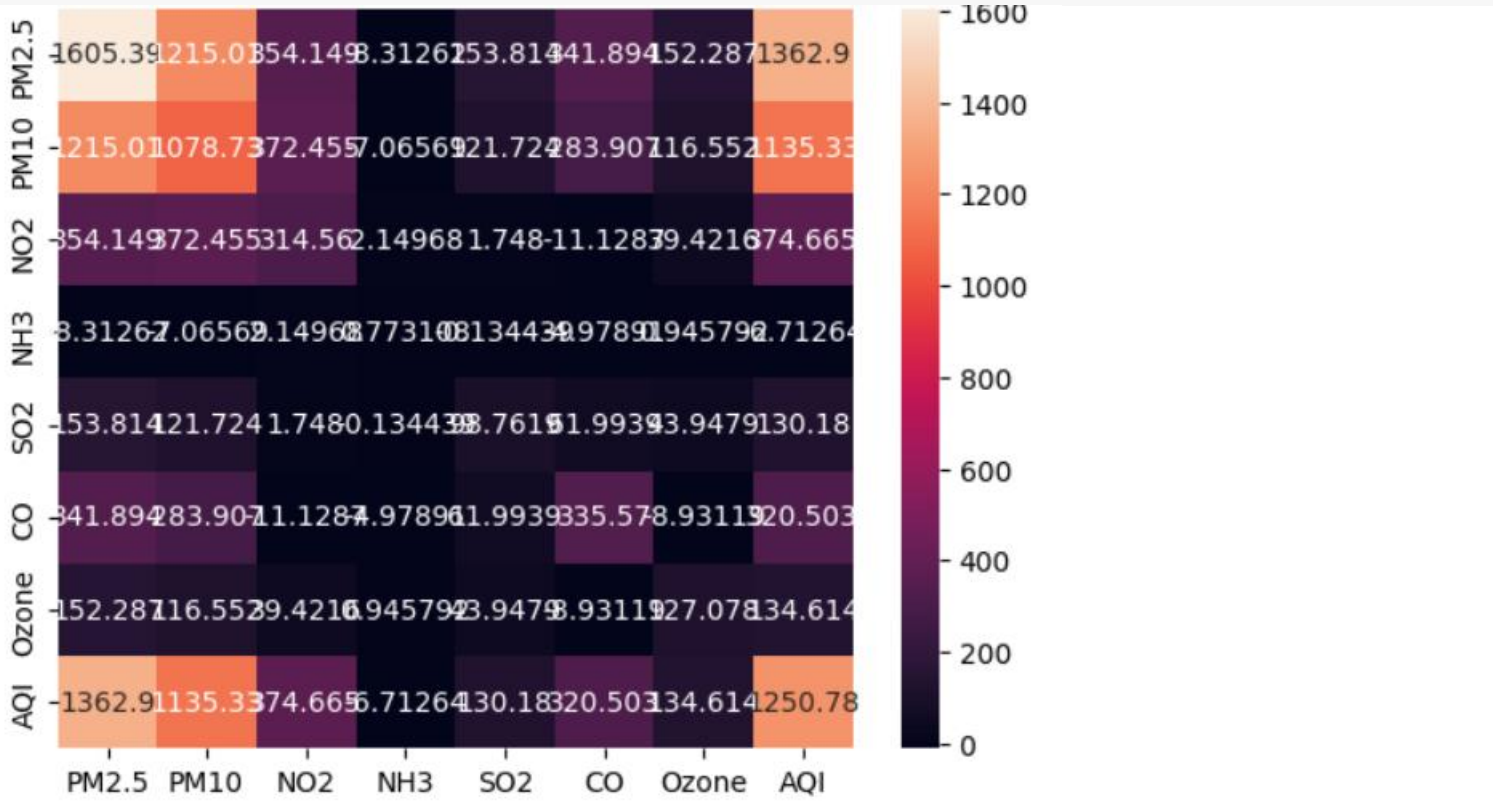
```
<Axes: xlabel='Ozone'>
```



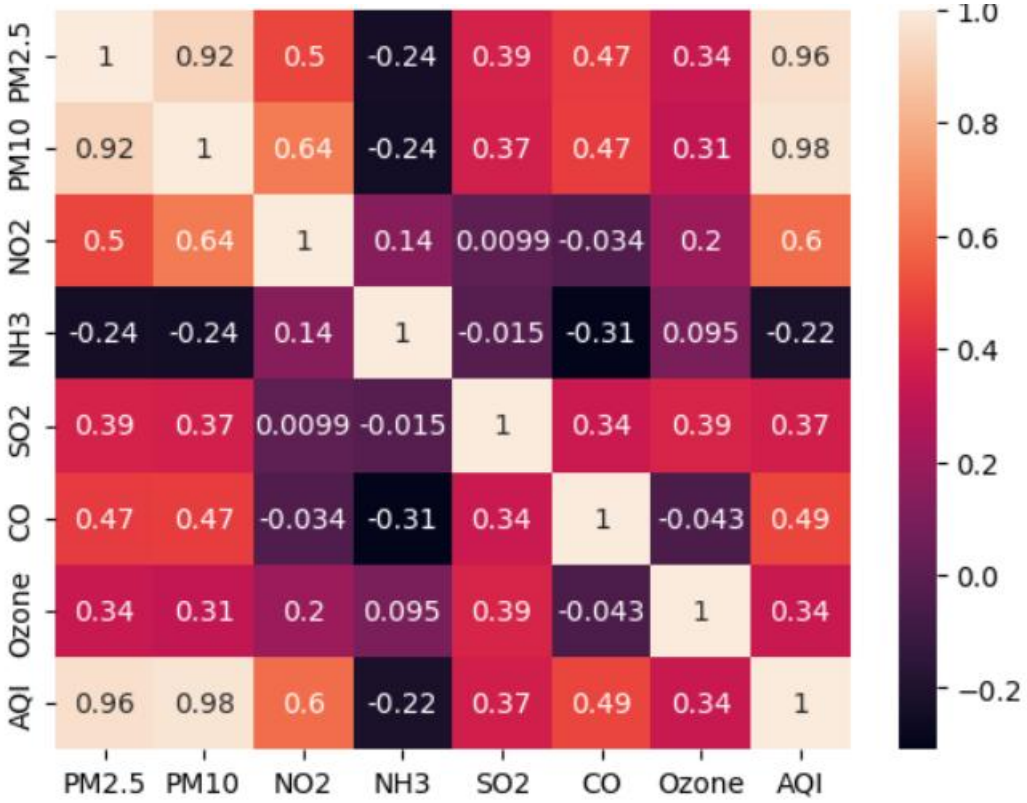

```
corr_matrix=d.corr() # Calculate the correlation matrix for the DataFrame 'd'.
print(corr_matrix) # Display the correlation matrix.
```

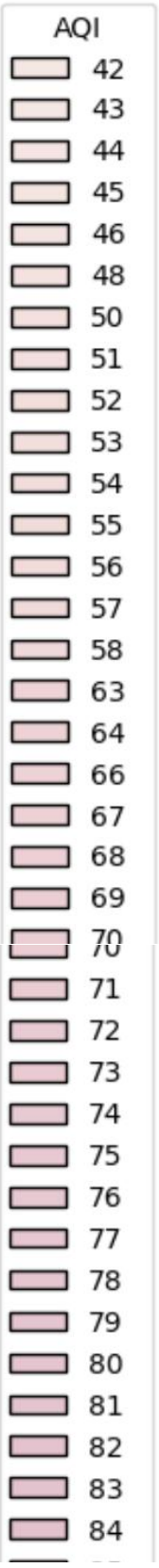
```
          PM2.5      PM10       NO2       NH3       SO2        CO     Ozone  \
PM2.5  1.000000  0.923282  0.498360 -0.235954  0.386288  0.465810  0.337161
PM10   0.923282  1.000000  0.639389 -0.244668  0.372927  0.471876  0.314796
NO2    0.498360  0.639389  1.000000  0.137849  0.009917 -0.034253  0.197173
NH3   -0.235954 -0.244668  0.137849  1.000000 -0.015385 -0.309117  0.095420
SO2    0.386288  0.372927  0.009917 -0.015385  1.000000  0.340536  0.392291
CO     0.465810  0.471876 -0.034253 -0.309117  0.340536  1.000000 -0.043250
Ozone  0.337161  0.314796  0.197173  0.095420  0.392291 -0.043250  1.000000
AQI    0.961794  0.977402  0.597310 -0.215865  0.370389  0.494708  0.337648

            AQI
PM2.5  0.961794
PM10   0.977402
NO2    0.597310
NH3   -0.215865
SO2    0.370389
CO     0.494708
Ozone  0.337648
AQI    1.000000
<ipython-input-253-5c706ce1cb70>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version,
  corr_matrix=d.corr()
```

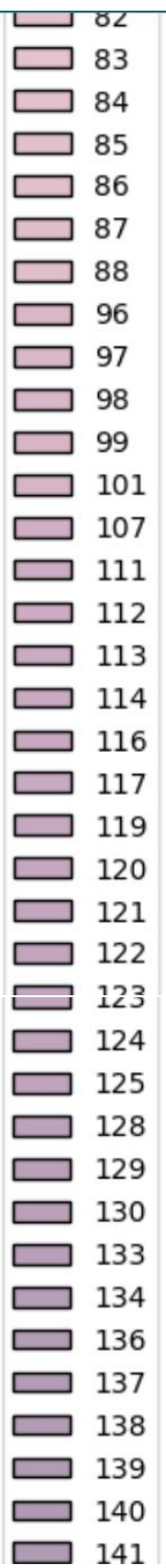
```
cov_matrix=pd.DataFrame.cov(d)
# Create a heatmap to visualize the covariance matrix using seaborn's sns.heatmap() function.
sns.heatmap(cov_matrix,annot=True,fmt='g')
#plot.show
```

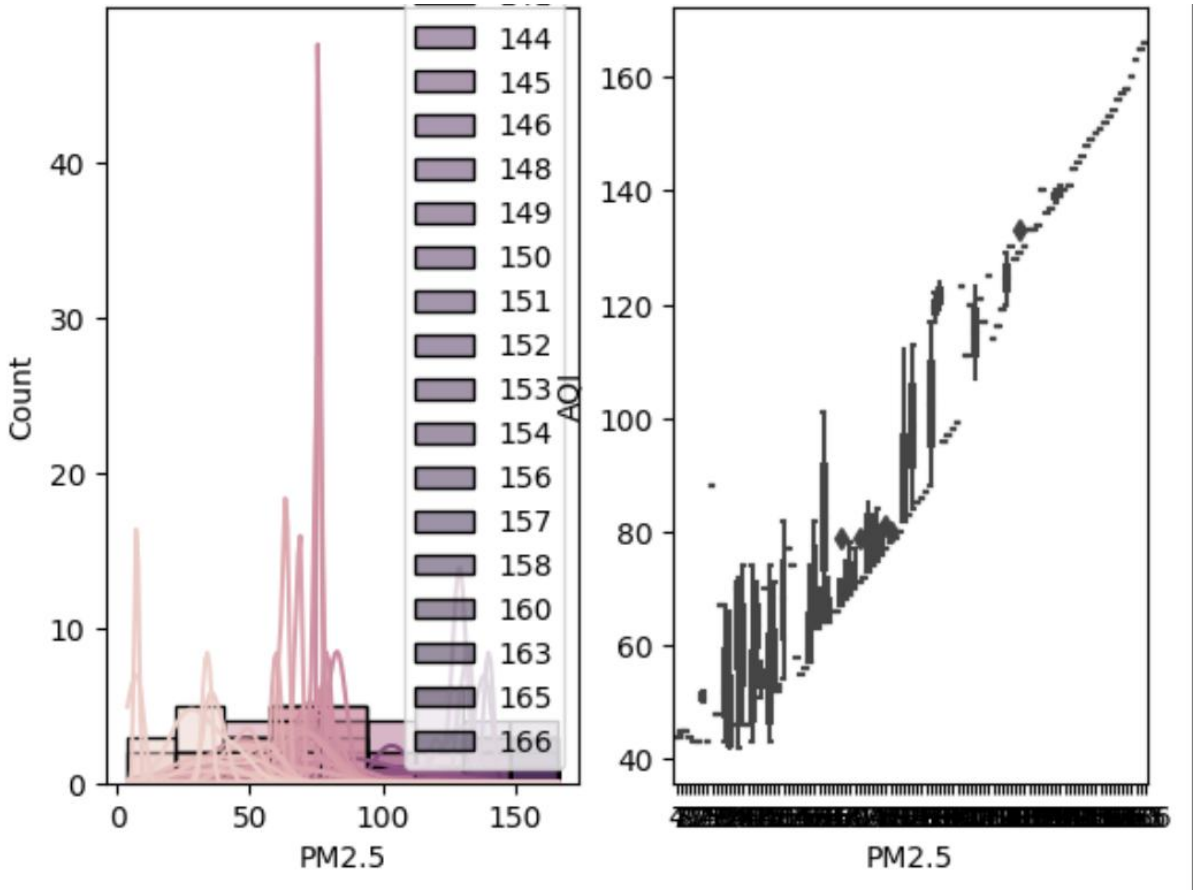



```
sns.heatmap(corr_matrix, annot=True) # Generate a heatmap of the correlation matrix with annotated values
plt .show() # Display the heatmap
```


PM2.5
PM10
NO2
NH3
SO2
CO
Ozone
AQI
1 0.92 0.5 -0.24 0.39 0.47 0.34 0.96
0.92 1 0.64 -0.24 0.37 0.47 0.31 0.98
0.5 0.64 1 0.14 0.0099 -0.034 0.2 0.6
-0.24 -0.24 0.14 1 -0.015 -0.31 0.095 -0.22
0.39 0.37 0.0099 -0.015 1 0.34 0.39 0.37
0.47 0.47 -0.034 -0.31 0.34 1 -0.043 0.49
0.34 0.31 0.2 0.095 0.39 -0.043 1 0.34
0.96 0.98 0.6 -0.22 0.37 0.49 0.34 1
PM2.5 PM10 NO2 NH3 SO2 CO Ozone AQI
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


AQI
42
43
44
45
46
48
50
51
52
53
54
55
56
57
58
63
64
66
67
68
69
70
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84


82
83
84
85
86
87
88
96
97
98
99
101
107
111
112
113
114
116
117
119
120
121
122
123
124
125
128
129
130
133
134
136
137
138
139
140
141




```
x=d.iloc[:,2:9] # Select columns 2 through 8 from the DataFrame 'd' and store them in 'x'
y=d.iloc[:,9:10] # Select column 9 from the DataFrame 'd' and store it in 'y'
print(x,'\n',y)
```

```
     PM2.5  PM10  NO2  NH3  SO2  CO  Ozone
0       73    83    5    3    8  32     20
1       74    83    5    3    8  32     20
2       73    84    5    3    8  35     21
3       73    85    5    3    8  37     22
4       75    84    5    3    8  38     20
..     ...   ...  ...  ...  ...  ..    ...
222    129   119    9    2   17  81     18
223    119   114    9    2   16  69     18
224    120   113    9    2   16  72     17
225    136   113    6    2   16  76     16
226    137   113    6    2   16  79     17

[227 rows x 7 columns]
      AQI
0      83
1      83
2      84
3      85
4      84
..    ...
222   129
223   119
224   120
225   136
226   137

[227 rows x 1 columns]
```

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

```
X_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=2)
print(X_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(181, 7)
(181, 1)
(46, 7)
(46, 1)
```

```
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,mean_absolute_error
reg_all=linear_model.LinearRegression()
reg_all.fit(X_train,y_train)
y_pred=reg_all.predict(x_test)
mse=mean_squared_error(y_test,y_pred)
print("mse:",mse)
mae = mean_absolute_error(y_test,y_pred)
print("mae:",mae)
print("rmse:",np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
mse: 20.186750203222648
mae: 3.6098028860602294
rmse: 4.492966748510682
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score
cls=DecisionTreeRegressor() # Fit the decision tree regressor model to the training data (X_train and y_train)
cc=cls.fit(X_train,y_train)
from sklearn import tree
tree.plot_tree(cc,filled=True) # Plot the decision tree with filled nodes
```

OUTPUT:

[Text(0.5520977608494921, 0.9615384615384616, 'x[1] <= 106.0\nsquared_error = 1150.348\nsamples = 181\nvalue = 91.541'), Text(0.31102839335180055, 0.8846153846153846, 'x[1] <= 59.0\nsquared_error = 254.212\nsamples = 127\nvalue = 71.764'), Text(0.12188365650969529, 0.8076923076923077, 'x[0] <= 62.0\nsquared_error = 71.572\nsamples = 34\nvalue = 51.324'), Text(0.08125577100646353, 0.7307692307692307, 'x[1] <= 49.0\nsquared_error = 29.316\nsamples = 30\nvalue = 48.867'), Text(0.0332409972299169, 0.6538461538461539, 'x[5] <= 37.5\nsquared_error = 2.729\nsamples = 15\nvalue = 44.267'),

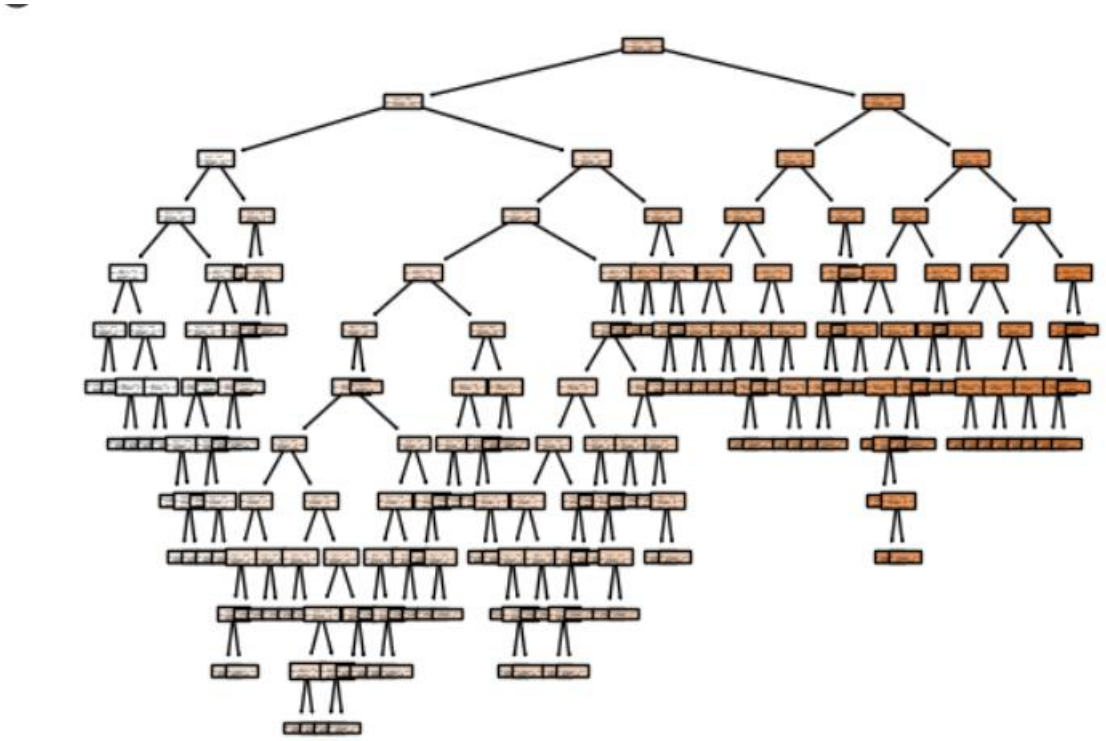
Text(0.014773776546629732, 0.5769230769230769, 'x[1] <= 47.0\nsquared_error = 0.75\nsamples = 4\nvalue = 46.5'), Text(0.007386888273314866, 0.5, 'squared_error = 0.0\nsamples = 3\nvalue = 46.0'), Text(0.0221606648199446, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 48.0'), Text(0.05170821791320406, 0.5769230769230769, 'x[0] <= 18.0\nsquared_error = 0.975\nsamples = 11\nvalue = 43.455'), Text(0.03693444136657433, 0.5, 'x[5] <= 44.5\nsquared_error = 0.24\nsamples = 5\nvalue = 44.4'), Text(0.029547553093259463, 0.4230769230769231, 'squared_error = 0.0\nsamples = 3\nvalue = 44.0'), Text(0.0443213296398892, 0.4230769230769231, 'squared_error = 0.0\nsamples = 2\nvalue = 45.0'), Text(0.0664819944598338, 0.5, 'x[2] <= 4.5\nsquared_error = 0.222\nsamples = 6\nvalue = 42.667'), Text(0.05909510618651893, 0.4230769230769231, 'squared_error = 0.0\nsamples = 2\nvalue = 42.0'), Text(0.07386888273314866, 0.4230769230769231, 'squared_error = 0.0\nsamples = 4\nvalue = 43.0'), Text(0.12927054478301014, 0.6538461538461539, 'x[0] <= 52.5\nsquared_error = 13.582\nsamples = 15\nvalue = 53.467'), Text(0.11080332409972299, 0.5769230769230769, 'x[1] <= 57.5\nsquared_error = 12.926\nsamples = 11\nvalue = 52.273'), Text(0.10341643582640812, 0.5, 'x[1] <= 53.5\nsquared_error = 4.76\nsamples = 10\nvalue = 53.2'), Text(0.0886426592797784, 0.4230769230769231, 'x[1] <= 51.0\nsquared_error = 1.44\nsamples = 5\nvalue = 51.4'), Text(0.08125577100646353, 0.34615384615384615, 'squared_error = 0.0\nsamples = 2\nvalue = 50.0'), Text(0.09602954755309326, 0.34615384615384615, 'x[6] <= 12.5\nsquared_error = 0.222\nsamples = 3\nvalue = 52.333'), Text(0.0886426592797784, 0.2692307692307692, 'squared_error = 0.0\nsamples = 2\nvalue = 52.0'), Text(0.10341643582640812, 0.2692307692307692, 'squared_error = 0.0\nsamples = 1\nvalue = 53.0'), Text(0.11819021237303785, 0.4230769230769231, 'x[1] <= 55.0\nsquared_error = 1.6\nsamples = 5\nvalue = 55.0'), Text(0.11080332409972299, 0.34615384615384615, 'squared_error = 0.0\nsamples = 3\nvalue = 54.0'), Text(0.12557710064635272, 0.34615384615384615, 'x[0] <= 43.5\nsquared_error = 0.25\nsamples = 2\nvalue = 56.5'), Text(0.11819021237303785, 0.2692307692307692, 'squared_error = 0.0\nsamples = 1\nvalue = 56.0'), Text(0.1329639889196676, 0.2692307692307692, 'squared_error = 0.0\nsamples = 1\nvalue = 57.0'), Text(0.11819021237303785, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 43.0'), Text(0.14773776546629733, 0.5769230769230769, 'x[4] <= 11.0\nsquared_error = 0.688\nsamples = 4\nvalue = 56.75'), Text(0.14035087719298245, 0.5, 'x[6] <= 6.0\nsquared_error = 0.222\nsamples = 3\nvalue = 56.333'), Text(0.1329639889196676, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 57.0'), Text(0.14773776546629733, 0.4230769230769231, 'squared_error = 0.0\nsamples = 2\nvalue = 56.0'), Text(0.15512465373961218, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 58.0'), Text(0.16251154201292706, 0.7307692307692307, 'x[6] <= 14.5\nsquared_error = 3.688\nsamples = 4\nvalue = 69.75'), Text(0.15512465373961218, 0.6538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 73.0'), Text(0.1698984302862419, 0.6538461538461539, 'x[3] <= 3.5\nsquared_error = 0.222\nsamples = 3\nvalue = 68.667'), Text(0.16251154201292706, 0.5769230769230769, 'squared_error = 0.0\nsamples = 1\nvalue = 68.0'), Text(0.1772853185595568, 0.5769230769230769, 'squared_error = 0.0\nsamples = 2\nvalue = 69.0'), Text(0.5001731301939059, 0.8076923076923077, 'x[0] <= 92.5\nsquared_error = 112.396\nsamples = 93\nvalue = 79.237'), Text(0.42878578024007385, 0.7307692307692307, 'x[1] <= 74.5\nsquared_error = 48.633\nsamples = 83\nvalue = 76.446'), Text(0.3303324099722992, 0.6538461538461539, 'x[0] <= 74.5\nsquared_error = 28.296\nsamples = 53\nvalue = 72.925'), Text(0.265466297322253, 0.5769230769230769, 'x[5] <= 75.5\nsquared_error = 19.25\nsamples = 36\nvalue = 70.5'), Text(0.2580794090489381, 0.5, 'x[3] <= 2.5\nsquared_error = 12.145\nsamples = 34\nvalue = 69.824'), Text(0.1948291782086796, 0.4230769230769231, 'x[1] <= 64.5\nsquared_error = 9.868\nsamples = 22\nvalue = 68.364'), Text(0.16251154201292706, 0.34615384615384615, 'x[0] <= 65.0\nsquared_error = 2.49\nsamples = 7\nvalue = 64.714'), Text(0.14773776546629733, 0.2692307692307692, 'x[1] <= 63.5\nsquared_error = 0.16\nsamples = 5\nvalue = 63.8'), Text(0.14035087719298245, 0.19230769230769232, 'x[0] <= 62.5\nsquared_error = 0.25\nsamples = 2\nvalue = 63.5'), Text(0.1329639889196676, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 63.0'), Text(0.14773776546629733, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 64.0'), Text(0.15512465373961218, 0.19230769230769232, 'squared_error = 0.0\nsamples = 3\nvalue = 64.0'), Text(0.1772853185595568, 0.2692307692307692, 'x[4] <= 9.5\nsquared_error = 1.0\nsamples = 2\nvalue = 67.0'),

Text(0.1698984302862419, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 68.0'), Text(0.18467220683287167, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 66.0'), Text(0.22714681440443213, 0.34615384615384615, 'x[0] <= 34.0\nsquared_error = 4.196\nsamples = 15\nvalue = 70.067'), Text(0.20683287165281625, 0.2692307692307692, 'x[0] <= 32.5\nsquared_error = 0.222\nsamples = 3\nvalue = 66.667'), Text(0.1994459833795014, 0.19230769230769232, 'squared_error = 0.0\nsamples = 2\nvalue = 67.0'), Text(0.21421975992613113, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 66.0'), Text(0.247460757156048, 0.2692307692307692, 'x[0] <= 71.5\nsquared_error = 1.576\nsamples = 12\nvalue = 70.917'), Text(0.22899353647276086, 0.19230769230769232, 'x[6] <= 12.0\nsquared_error = 0.444\nsamples = 9\nvalue = 70.333'), Text(0.21421975992613113, 0.11538461538461539, 'x[4] <= 9.5\nsquared_error = 0.222\nsamples = 3\nvalue = 69.667'), Text(0.20683287165281625, 0.038461538461538464, 'squared_error = 0.0\nsamples = 2\nvalue = 70.0'), Text(0.22160664819944598, 0.038461538461538464, 'squared_error = 0.0\nsamples = 1\nvalue = 69.0'), Text(0.24376731301939059, 0.11538461538461539, 'x[0] <= 47.0\nsquared_error = 0.222\nsamples = 6\nvalue = 70.667'), Text(0.2363804247460757, 0.038461538461538464, 'squared_error = 0.0\nsamples = 2\nvalue = 70.0'), Text(0.25115420129270544, 0.038461538461538464, 'squared_error = 0.0\nsamples = 4\nvalue = 71.0'), Text(0.2659279778393352, 0.19230769230769232, 'x[0] <= 73.0\nsquared_error = 0.889\nsamples = 3\nvalue = 72.667'), Text(0.2585410895660203, 0.11538461538461539, 'squared_error = 0.0\nsamples = 2\nvalue = 72.0'), Text(0.27331486611265005, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 74.0'), Text(0.32132963988919666, 0.4230769230769231, 'x[3] <= 3.5\nsquared_error = 5.25\nsamples = 12\nvalue = 72.5'), Text(0.3028624192059095, 0.34615384615384615, 'x[1] <= 70.5\nsquared_error = 6.806\nsamples = 6\nvalue = 71.167'), Text(0.2880886426592798, 0.2692307692307692, 'x[5] <= 40.5\nsquared_error = 8.222\nsamples = 3\nvalue = 69.667'), Text(0.2807017543859649, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 66.0'), Text(0.29547553093259465, 0.19230769230769232, 'x[4] <= 10.5\nsquared_error = 2.25\nsamples = 2\nvalue = 71.5'), Text(0.2880886426592798, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 73.0'), Text(0.3028624192059095, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 70.0'), Text(0.31763619575253926, 0.2692307692307692, 'x[2] <= 8.5\nsquared_error = 0.889\nsamples = 3\nvalue = 72.667'), Text(0.31024930747922436, 0.19230769230769232, 'squared_error = 0.0\nsamples = 2\nvalue = 72.0'), Text(0.3250230840258541, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 74.0'), Text(0.3397968605724838, 0.34615384615384615, 'x[4] <= 8.5\nsquared_error = 0.139\nsamples = 6\nvalue = 73.833'), Text(0.33240997229916897, 0.2692307692307692, 'squared_error = 0.0\nsamples = 4\nvalue = 74.0'), Text(0.3471837488457987, 0.2692307692307692, 'x[0] <= 73.5\nsquared_error = 0.25\nsamples = 2\nvalue = 73.5'), Text(0.3397968605724838, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 73.0'), Text(0.3545706371191136, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 74.0'), Text(0.27285318559556787, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 82.0'), Text(0.39519852262234534, 0.5769230769230769, 'x[0] <= 80.5\nsquared_error = 8.644\nsamples = 17\nvalue = 78.059'), Text(0.3767313019390582, 0.5, 'x[0] <= 76.5\nsquared_error = 1.122\nsamples = 14\nvalue = 76.857'), Text(0.3619575253924284, 0.4230769230769231, 'x[2] <= 3.5\nsquared_error = 0.139\nsamples = 6\nvalue = 75.833'), Text(0.3545706371191136, 0.34615384615384615, 'squared_error = 0.0\nsamples = 1\nvalue = 75.0'), Text(0.36934441366574333, 0.34615384615384615, 'squared_error = 0.0\nsamples = 5\nvalue = 76.0'), Text(0.3915050784856879, 0.4230769230769231, 'x[0] <= 77.5\nsquared_error = 0.484\nsamples = 8\nvalue = 77.625'), Text(0.38411819021237303, 0.34615384615384615, 'squared_error = 0.0\nsamples = 4\nvalue = 77.0'), Text(0.3988919667590028, 0.34615384615384615, 'x[3] <= 2.5\nsquared_error = 0.188\nsamples = 4\nvalue = 78.25'), Text(0.3915050784856879, 0.2692307692307692, 'squared_error = 0.0\nsamples = 1\nvalue = 79.0'), Text(0.40627885503231764, 0.2692307692307692, 'squared_error = 0.0\nsamples = 3\nvalue = 78.0'), Text(0.4136657433056325, 0.5, 'x[5] <= 37.5\nsquared_error = 5.556\nsamples = 3\nvalue = 83.667'), Text(0.40627885503231764, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 87.0'), Text(0.42105263157894735, 0.4230769230769231, 'squared_error = 0.0\nsamples = 2\nvalue = 82.0'), Text(0.5272391505078485, 0.6538461538461539, 'x[2] <= 13.5\nsquared_error = 23.956\nsamples = 30\nvalue = 82.667'), Text(0.5198522622345337, 0.5769230769230769, 'x[0] <=

83.5\nsquared_error = 12.792\nsamples = 29\nvalue = 82.034'), Text(0.4856879039704525, 0.5, 'x[1] <= 82.0\nsquared_error = 10.042\nsamples = 23\nvalue = 80.957'), Text(0.4616805170821791, 0.4230769230769231, 'x[0] <= 75.5\nsquared_error = 6.561\nsamples = 17\nvalue = 79.706'), Text(0.4358264081255771, 0.34615384615384615, 'x[1] <= 77.5\nsquared_error = 2.122\nsamples = 7\nvalue = 77.143'), Text(0.42105263157894735, 0.2692307692307692, 'x[4] <= 10.0\nsquared_error = 0.5\nsamples = 4\nvalue = 76.0'), Text(0.4136657433056325, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 77.0'), Text(0.42843951985226225, 0.19230769230769232, 'x[0] <= 69.0\nsquared_error = 0.222\nsamples = 3\nvalue = 75.667'), Text(0.42105263157894735, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 75.0'), Text(0.4358264081255771, 0.11538461538461539, 'squared_error = 0.0\nsamples = 2\nvalue = 76.0'), Text(0.45060018467220686, 0.2692307692307692, 'x[5] <= 35.5\nsquared_error = 0.222\nsamples = 3\nvalue = 78.667'), Text(0.44321329639889195, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 78.0'), Text(0.4579870729455217, 0.19230769230769232, 'squared_error = 0.0\nsamples = 2\nvalue = 79.0'), Text(0.48753462603878117, 0.34615384615384615, 'x[0] <= 81.5\nsquared_error = 1.85\nsamples = 10\nvalue = 81.5'), Text(0.4801477377654663, 0.2692307692307692, 'x[2] <= 7.5\nsquared_error = 0.583\nsamples = 6\nvalue = 80.5'), Text(0.4727608494921514, 0.19230769230769232, 'x[4] <= 9.5\nsquared_error = 0.25\nsamples = 2\nvalue = 79.5'), Text(0.46537396121883656, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 80.0'), Text(0.4801477377654663, 0.11538461538461539, 'squared_error = 0.0\nsamples = 1\nvalue = 79.0'), Text(0.48753462603878117, 0.19230769230769232, 'squared_error = 0.0\nsamples = 4\nvalue = 81.0'), Text(0.494921514312096, 0.2692307692307692, 'squared_error = 0.0\nsamples = 4\nvalue = 83.0'), Text(0.5096952908587258, 0.4230769230769231, 'x[3] <= 2.5\nsquared_error = 2.917\nsamples = 6\nvalue = 84.5'), Text(0.5023084025854109, 0.34615384615384615, 'squared_error = 0.0\nsamples = 1\nvalue = 88.0'), Text(0.5170821791320406, 0.34615384615384615, 'x[5] <= 33.5\nsquared_error = 0.56\nsamples = 5\nvalue = 83.8'), Text(0.5096952908587258, 0.2692307692307692, 'squared_error = 0.0\nsamples = 2\nvalue = 83.0'), Text(0.5244690674053555, 0.2692307692307692, 'x[6] <= 21.5\nsquared_error = 0.222\nsamples = 3\nvalue = 84.333'), Text(0.5170821791320406, 0.19230769230769232, 'squared_error = 0.0\nsamples = 2\nvalue = 84.0'), Text(0.5318559556786704, 0.19230769230769232, 'squared_error = 0.0\nsamples = 1\nvalue = 85.0'), Text(0.554016620498615, 0.5, 'x[0] <= 85.5\nsquared_error = 1.806\nsamples = 6\nvalue = 86.167'), Text(0.5392428439519852, 0.4230769230769231, 'x[5] <= 44.0\nsquared_error = 0.25\nsamples = 2\nvalue = 84.5'), Text(0.5318559556786704, 0.34615384615384615, 'squared_error = 0.0\nsamples = 1\nvalue = 84.0'), Text(0.5466297322253001, 0.34615384615384615, 'squared_error = 0.0\nsamples = 1\nvalue = 85.0'), Text(0.5687903970452447, 0.4230769230769231, 'x[0] <= 86.5\nsquared_error = 0.5\nsamples = 4\nvalue = 87.0'), Text(0.5614035087719298, 0.34615384615384615, 'squared_error = 0.0\nsamples = 1\nvalue = 86.0'), Text(0.5761772853185596, 0.34615384615384615, 'x[5] <= 61.5\nsquared_error = 0.222\nsamples = 3\nvalue = 87.333'), Text(0.5687903970452447, 0.2692307692307692, 'squared_error = 0.0\nsamples = 2\nvalue = 87.0'), Text(0.5835641735918744, 0.2692307692307692, 'squared_error = 0.0\nsamples = 1\nvalue = 88.0'), Text(0.5346260387811634, 0.5769230769230769, 'squared_error = 0.0\nsamples = 1\nvalue = 101.0'), Text(0.5715604801477377, 0.7307692307692307, 'x[2] <= 5.0\nsquared_error = 40.44\nsamples = 10\nvalue = 102.4'), Text(0.556786703601108, 0.6538461538461539, 'x[5] <= 54.5\nsquared_error = 10.889\nsamples = 3\nvalue = 111.667'), Text(0.5493998153277931, 0.5769230769230769, 'squared_error = 0.0\nsamples = 2\nvalue = 114.0'), Text(0.564173591874423, 0.5769230769230769, 'squared_error = 0.0\nsamples = 1\nvalue = 107.0'), Text(0.5863342566943675, 0.6538461538461539, 'x[0] <= 98.5\nsquared_error = 0.531\nsamples = 7\nvalue = 98.429'), Text(0.5789473684210527, 0.5769230769230769, 'x[5] <= 43.5\nsquared_error = 0.222\nsamples = 3\nvalue = 97.667'), Text(0.5715604801477377, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 97.0'), Text(0.5863342566943675, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 98.0'), Text(0.5937211449676824, 0.5769230769230769, 'squared_error = 0.0\nsamples = 4\nvalue = 99.0'), Text(0.7931671283471837, 0.8846153846153846, 'x[0] <= 134.0\nsquared_error = 174.423\nsamples = 54\nvalue = 138.056'), Text(0.7045244690674054, 0.8076923076923077, 'x[0] <= 120.5\nsquared_error = 41.424\nsamples = 22\nvalue = 124.409'), Text(0.6528162511542013, 0.7307692307692307, 'x[4] <= 6.5\nsquared_error = 14.744\nsamples =

11\nvalue = 118.727'), Text(0.6232686980609419, 0.6538461538461539, 'x[1] <= 115.0\nsquared_error = 8.64\nsamples = 5\nvalue = 115.4'), Text(0.6084949215143121, 0.5769230769230769, 'x[2] <= 62.0\nsquared_error = 1.0\nsamples = 2\nvalue = 112.0'), Text(0.6011080332409973, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 111.0'), Text(0.615881809787627, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 113.0'), Text(0.6380424746075716, 0.5769230769230769, 'x[6] <= 2.5\nsquared_error = 0.889\nsamples = 3\nvalue = 117.667'), Text(0.6306555863342567, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 119.0'), Text(0.6454293628808865, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 117.0'), Text(0.6823638042474608, 0.6538461538461539, 'x[1] <= 121.5\nsquared_error = 2.917\nsamples = 6\nvalue = 121.5'), Text(0.667590027700831, 0.5769230769230769, 'x[2] <= 34.0\nsquared_error = 0.667\nsamples = 3\nvalue = 120.0'), Text(0.6602031394275162, 0.5, 'x[5] <= 70.5\nsquared_error = 0.25\nsamples = 2\nvalue = 119.5'), Text(0.6528162511542013, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 119.0'), Text(0.667590027700831, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 120.0'), Text(0.6749769159741459, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 121.0'), Text(0.6971375807940905, 0.5769230769230769, 'x[2] <= 21.5\nsquared_error = 0.667\nsamples = 3\nvalue = 123.0'), Text(0.6897506925207756, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 122.0'), Text(0.7045244690674054, 0.5, 'x[1] <= 123.5\nsquared_error = 0.25\nsamples = 2\nvalue = 123.5'), Text(0.6971375807940905, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 123.0'), Text(0.7119113573407202, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 124.0'), Text(0.7562326869806094, 0.7307692307692307, 'x[0] <= 131.0\nsquared_error = 3.537\nsamples = 11\nvalue = 130.091'), Text(0.7488457987072945, 0.6538461538461539, 'x[5] <= 93.0\nsquared_error = 2.025\nsamples = 9\nvalue = 129.444'), Text(0.7414589104339797, 0.5769230769230769, 'x[2] <= 12.0\nsquared_error = 0.5\nsamples = 8\nvalue = 129.0'), Text(0.7340720221606648, 0.5, 'x[0] <= 128.5\nsquared_error = 0.222\nsamples = 6\nvalue = 128.667'), Text(0.7266851338873499, 0.4230769230769231, 'squared_error = 0.0\nsamples = 2\nvalue = 128.0'), Text(0.7414589104339797, 0.4230769230769231, 'squared_error = 0.0\nsamples = 4\nvalue = 129.0'), Text(0.7488457987072945, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 130.0'), Text(0.7562326869806094, 0.5769230769230769, 'squared_error = 0.0\nsamples = 1\nvalue = 133.0'), Text(0.7636195752539243, 0.6538461538461539, 'squared_error = 0.0\nsamples = 2\nvalue = 133.0'), Text(0.8818097876269622, 0.8076923076923077, 'x[0] <= 147.0\nsquared_error = 49.809\nsamples = 32\nvalue = 147.438'), Text(0.8208679593721145, 0.7307692307692307, 'x[0] <= 142.5\nsquared_error = 9.694\nsamples = 14\nvalue = 140.857'), Text(0.7894736842105263, 0.6538461538461539, 'x[2] <= 7.0\nsquared_error = 2.49\nsamples = 10\nvalue = 139.1'), Text(0.7710064635272391, 0.5769230769230769, 'x[0] <= 136.5\nsquared_error = 0.25\nsamples = 2\nvalue = 136.5'), Text(0.7636195752539243, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 136.0'), Text(0.778393351800554, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 137.0'), Text(0.8079409048938134, 0.5769230769230769, 'x[4] <= 25.5\nsquared_error = 0.938\nsamples = 8\nvalue = 139.75'), Text(0.7931671283471837, 0.5, 'x[1] <= 118.5\nsquared_error = 0.472\nsamples = 6\nvalue = 140.167'), Text(0.7857802400738689, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 139.0'), Text(0.8005540166204986, 0.4230769230769231, 'x[0] <= 138.5\nsquared_error = 0.24\nsamples = 5\nvalue = 140.4'), Text(0.7931671283471837, 0.34615384615384615, 'squared_error = 0.0\nsamples = 2\nvalue = 140.0'), Text(0.8079409048938134, 0.34615384615384615, 'x[2] <= 32.0\nsquared_error = 0.222\nsamples = 3\nvalue = 140.667'), Text(0.8005540166204986, 0.2692307692307692, 'squared_error = 0.0\nsamples = 2\nvalue = 141.0'), Text(0.8153277931671283, 0.2692307692307692, 'squared_error = 0.0\nsamples = 1\nvalue = 140.0'), Text(0.8227146814404432, 0.5, 'x[1] <= 127.5\nsquared_error = 0.25\nsamples = 2\nvalue = 138.5'), Text(0.8153277931671283, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 139.0'), Text(0.830101569713758, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 138.0'), Text(0.8522622345337026, 0.6538461538461539, 'x[4] <= 8.5\nsquared_error = 0.688\nsamples = 4\nvalue = 145.25'), Text(0.8448753462603878, 0.5769230769230769, 'x[0] <= 144.5\nsquared_error = 0.25\nsamples = 2\nvalue = 144.5'), Text(0.8374884579870729, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 144.0'), Text(0.8522622345337026, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 145.0'), Text(0.8596491228070176,

0.5769230769230769, 'squared_error = 0.0\nsamples = 2\nvalue = 146.0'), Text(0.9427516158818098, 0.7307692307692307, 'x[0] <= 155.0\nsquared_error = 21.136\nsamples = 18\nvalue = 152.556'), Text(0.9002770083102493, 0.6538461538461539, 'x[0] <= 150.5\nsquared_error = 3.821\nsamples = 14\nvalue = 150.5'), Text(0.8744228993536473, 0.5769230769230769, 'x[4] <= 27.5\nsquared_error = 0.583\nsamples = 6\nvalue = 148.5'), Text(0.8670360110803325, 0.5, 'squared_error = 0.0\nsamples = 4\nvalue = 148.0'), Text(0.8818097876269622, 0.5, 'x[5] <= 55.0\nsquared_error = 0.25\nsamples = 2\nvalue = 149.5'), Text(0.8744228993536473, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 150.0'), Text(0.889196675900277, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 149.0'), Text(0.9261311172668514, 0.5769230769230769, 'x[0] <= 152.5\nsquared_error = 1.0\nsamples = 8\nvalue = 152.0'), Text(0.9113573407202216, 0.5, 'x[0] <= 151.5\nsquared_error = 0.25\nsamples = 6\nvalue = 151.5'), Text(0.9039704524469068, 0.4230769230769231, 'squared_error = 0.0\nsamples = 3\nvalue = 151.0'), Text(0.9187442289935365, 0.4230769230769231, 'squared_error = 0.0\nsamples = 3\nvalue = 152.0'), Text(0.9409048938134811, 0.5, 'x[4] <= 32.0\nsquared_error = 0.25\nsamples = 2\nvalue = 153.5'), Text(0.9335180055401662, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 154.0'), Text(0.948291782086796, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 153.0'), Text(0.9852262234533703, 0.6538461538461539, 'x[0] <= 163.0\nsquared_error = 15.188\nsamples = 4\nvalue = 159.75'), Text(0.9778393351800554, 0.5769230769230769, 'x[1] <= 141.5\nsquared_error = 2.889\nsamples = 3\nvalue = 157.667'), Text(0.9704524469067405, 0.5, 'x[2] <= 27.0\nsquared_error = 0.25\nsamples = 2\nvalue = 156.5'), Text(0.9630655586334257, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 157.0'), Text(0.9778393351800554, 0.4230769230769231, 'squared_error = 0.0\nsamples = 1\nvalue = 156.0'), Text(0.9852262234533703, 0.5, 'squared_error = 0.0\nsamples = 1\nvalue = 160.0'), Text(0.9926131117266851, 0.5769230769230769, 'squared_error = 0.0\nsamples = 1\nvalue = 166.0')]



```
cl=cc.fit(X_train,y_train.values.ravel()) # Fit a classifier (assuming it's a classifier) to the training data
yp=cl.predict(x_test) # Make predictions on the test data using the fitted classifier
print(yp)
```

```
[150.  81. 130. 113.  50. 166.  63.  68.  52.  43.  99.  72.  52. 160.
  74.  69. 144.  70. 146. 166.  68. 117.  98. 156.  82. 140. 146.  69.
  48. 119. 129.  57. 140.  43.  68. 123. 129.  56.  79. 121. 144. 117.
  71. 166.  72.  81.]
```

```
import numpy as np
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
print(mean_squared_error(y_test,yp ))
print(mean_absolute_error(y_test,yp ))
print(np.sqrt(mean_squared_error(y_test,yp)))
```

```
14.021739130434783
1.9782608695652173
3.74456127342507
```

```
from sklearn import svm
from sklearn.metrics import accuracy_score
ca=svm.SVR(kernel='linear')
ck=ca.fit(X_train,y_train.values.ravel())
yp=ck.predict(x_test)
print(yp)
```

```
[147.62259929  84.0078113  131.52450591 112.78951808  51.26774488
 158.0747105   65.40141734  70.55439019  47.04856186  34.11970615
 113.70027808  74.73567956  57.41185177 158.7616967   67.12824551
  71.59086284 145.55048987  58.88854278 140.72856355 158.46658729
  68.82663613 129.94868069  91.2440461  156.19959543  64.61826953
 138.91962034 147.50442717  68.79136871  48.41329817 118.19504745
 123.07715941  57.60196488 145.24411624  31.64553289  66.56750968
 120.28668391 122.5694353   58.29434561  80.38678232 123.60915493
 145.62892173 130.06674224  70.84972155 160.11652519  72.43439862
  83.99975057]
```

```
import numpy as np
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
print(mean_squared_error(y_test,yp ))
print(mean_absolute_error(y_test,yp ))
print(np.sqrt(mean_squared_error(y_test,yp)))
```

```
24.142612927396755
3.6699528174496265
4.9135132977734735
```

```
[149.4  80.6 134.8 120.   54.  150.6  63.8  70.6  45.   43.  119.2  71.
  51.2 155.   71.   71.4 148.4  70.2 144.6 151.2  64.2 122.8  98.6 154.8
  71.8 148.4 140.   70.6  50.2 120.  129.8  56.  144.2  43.   67.4 120.6
 127.6  55.2  77.8 120.8 150.  124.   70.2 155.   72.4  80.6]
```

	AQI
212	150
171	81
20	134
114	112
66	51
23	166
126	63
93	72
77	52
53	43
89	116
159	73
196	52
10	163
30	71
154	71
13	144
163	72
85	144
24	166
120	67
142	129
147	96
35	158
131	77
144	136
144	150
207	146
90	70
65	48
111	117
202	129
41	57
25	136
54	43
99	67
74	123
203	130
44	55
6	79
28	120
12	146
29	125
193	70
18	165
152	73
5	80

```
import numpy as np
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
print(mean_squared_error(y_test,yp ))
print(mean_absolute_error(y_test,yp ))
print(np.sqrt(mean_squared_error(y_test,yp)))
```

```
25.623478260869575
3.2913043478260877
5.061963873919842
```

```
from sklearn.ensemble import RandomForestRegressor
rfc = RandomForestRegressor() # Create an instance of the RandomForestRegressor class.
```

```
rr=rfc.fit(X_train,y_train.values.ravel()) # Fit the RandomForestRegressor model to the training data.
yp=rr.predict(x_test) # Use the trained model to make predictions on the test data.
print(yp)
```

```
[150.16  80.93 131.92 110.74  51.03 160.37  62.6   68.81  48.07  43.18
 113.38  72.87  53.02 160.67  70.49  70.68 144.71  71.23 144.97 160.35
  66.35 120.21  98.15 157.69  71.8  138.72 146.02  68.83  47.68 116.74
 129.03  55.41 138.79  43.18  66.13 121.71 128.9   56.13  78.04 120.24
 145.39 120.46  69.94 162.13  72.61  81.39]
```

```
import numpy as np
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
print(mean_squared_error(y_test,yp ))
print(mean_absolute_error(y_test,yp ))
print(np.sqrt(mean_squared_error(y_test,yp)))
```

```
6.022893478260879
1.5997826086956537
2.4541584052910843
```

RESULTS:

	Linear regression	**Decision Tree**	**SVM**	**KNN**	**Random Forest**
MEAN SQUARE ERROR	20.186750203222648	12.021739130434783	24.142612927396755	25.623478260869575	6.253336956521742
MEAN ABSOULTE ERROR	3.6098028860602294	2.108695652173913	3.6699528174496265	3.2913043478260877	1.6089130434782613

ROOT MEAN SQUARE ERROR	4.4929667485 10682	3.467237968532703	4.9135132977734735	5.061963873919842	2.500667302245 89

By the above table we say that mean square error is smaller for random forest so, random forest is the best model for air prediction.

CONCLUSION AND FUTURE SCOPE

Overall, air quality prediction is a critical tool for managing and reducing the impact of air pollution on human health and the environment. By using accurate air quality predictions, individuals and organizations can take proactive steps to reduce their exposure to harmful pollutants and contribute to a cleaner, healthier environment.

Air quality provides standards and objective for key air pollutants ,which are designed to protect human health and the environment.

Air pollution can cause both short term and long term effects on health and many people are concerned about pollution in the air that they breathe.

REFERENCES:

https://www.kaggle.com/code/kerneler/starter-uci-air-quality-b68c5049-e

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.]
A series of ones: [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

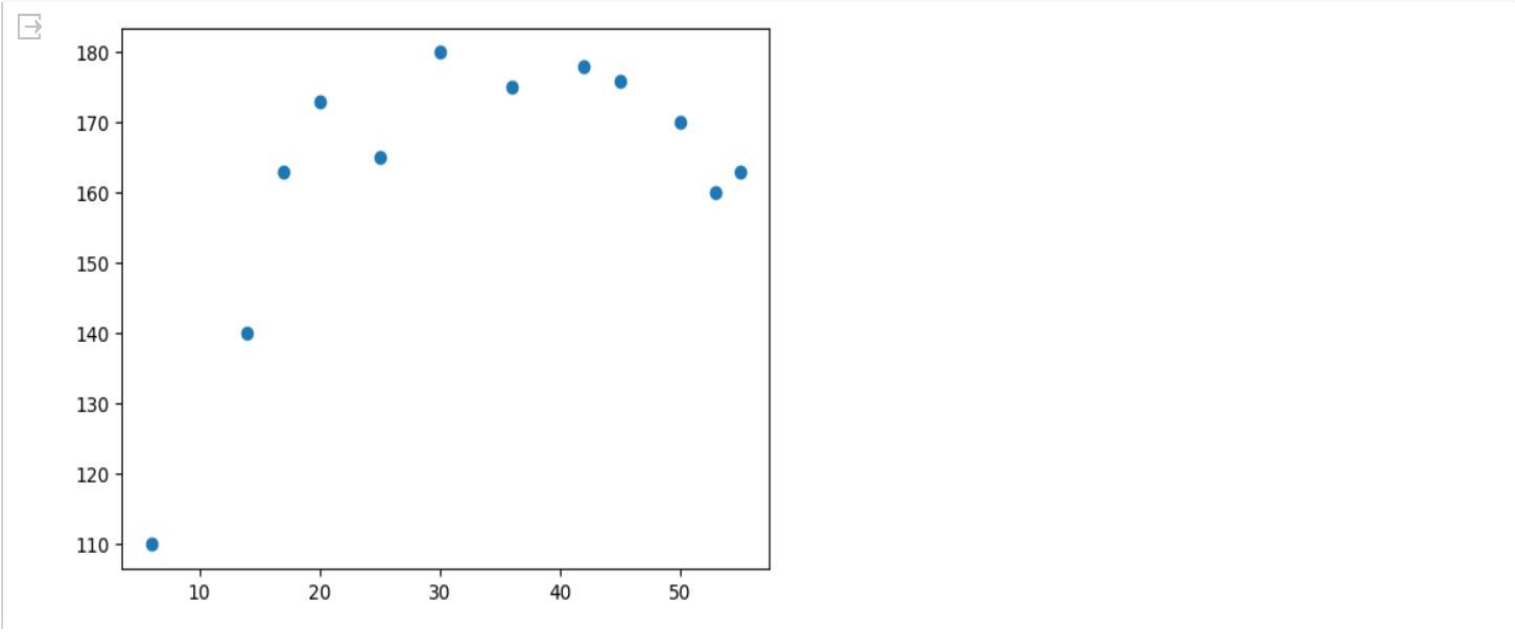
```
# Use the `read_excel` function from pandas to read an Excel file.
# The file path is "/content/name_height_weight.xlsx"
datatxt = pd.read_excel("/content/name_height_weight.xlsx")
# Print the summary statistics
datatxt
```

	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()
```

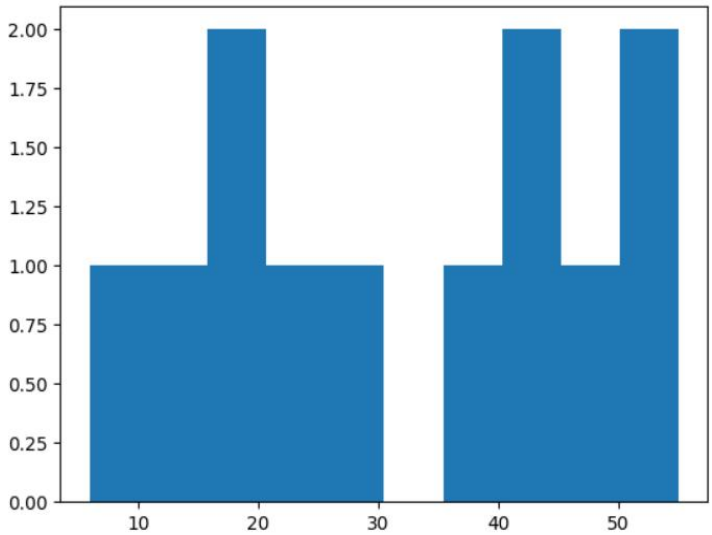
	2016 Summer Olympics medals	2016 Summer Olympics medals.1	Unnamed: 2
0	Location	Rio de Janeiro, Brazil	NaN
1	Highlights	Highlights	NaN
2	Most gold medals	United States (46)	NaN
3	Most total medals	United States (121)	NaN
4	← 2012 · Olympics medal tables · 2020 →	← 2012 · Olympics medal tables · 2020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

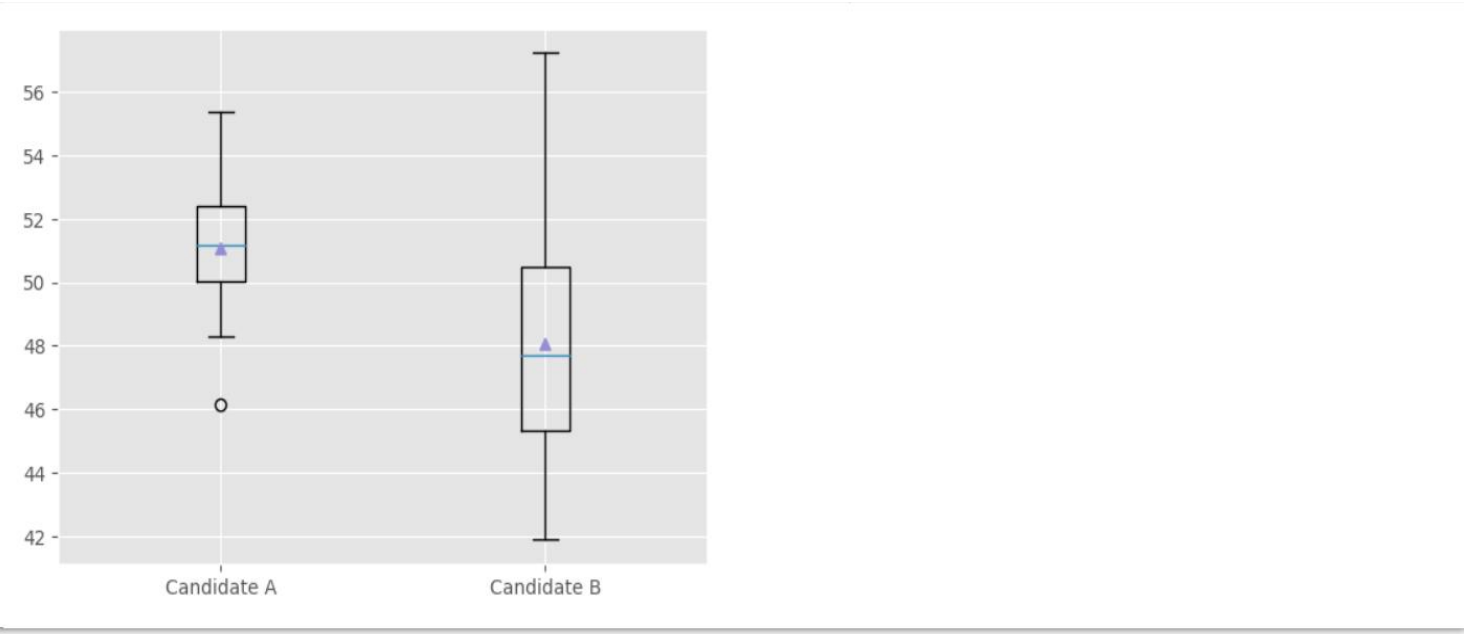



```
plt.hist(age)
```

```
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()
```


56
54
52
50
48
46
44
42
Candidate A
Candidate B

04

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)
```

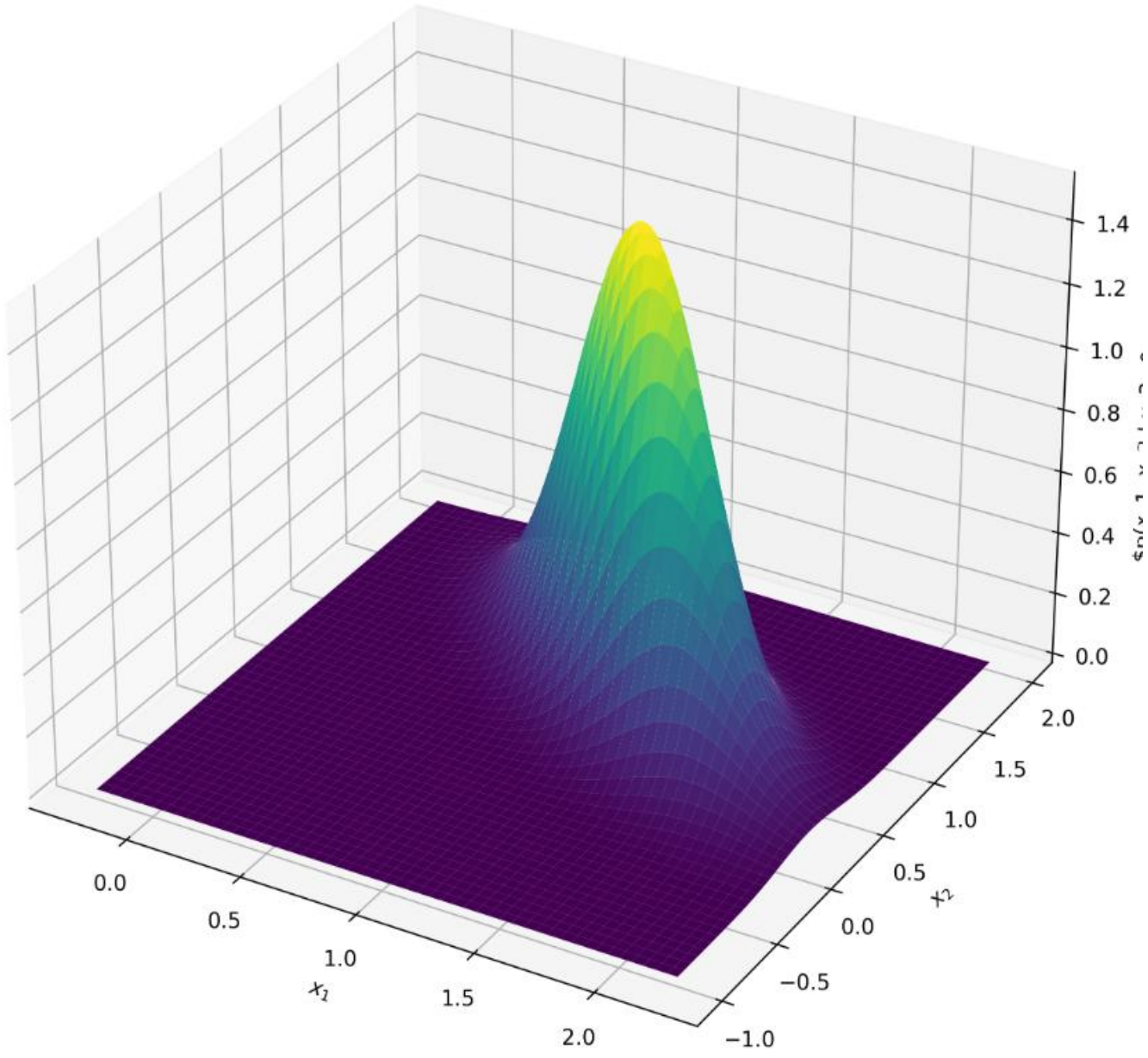
```
[[ 60.   50.  -48.   38. ]
 [ 50.   50.  -50.   40. ]
 [-48.  -50.   52.4 -41.4]
 [ 38.   40.  -41.4  33.4]]
```

```
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])
```

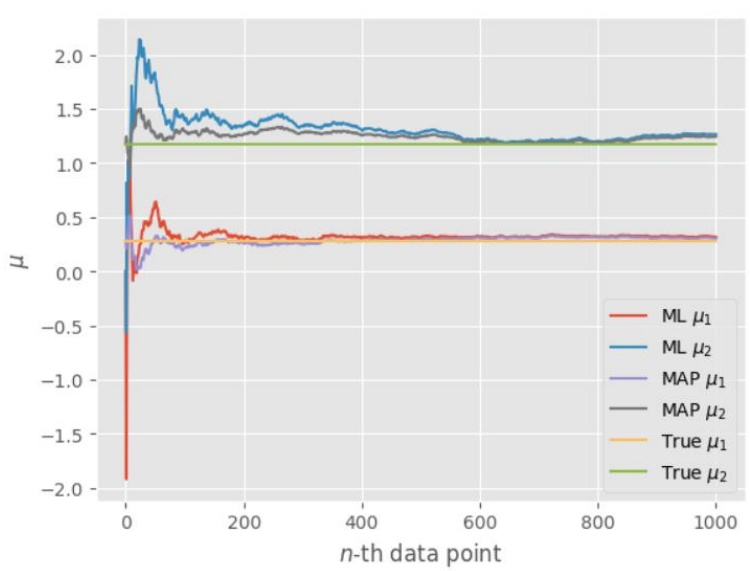
```
[[0.30449334]
 [1.24457165]]
```

```
# 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]
```

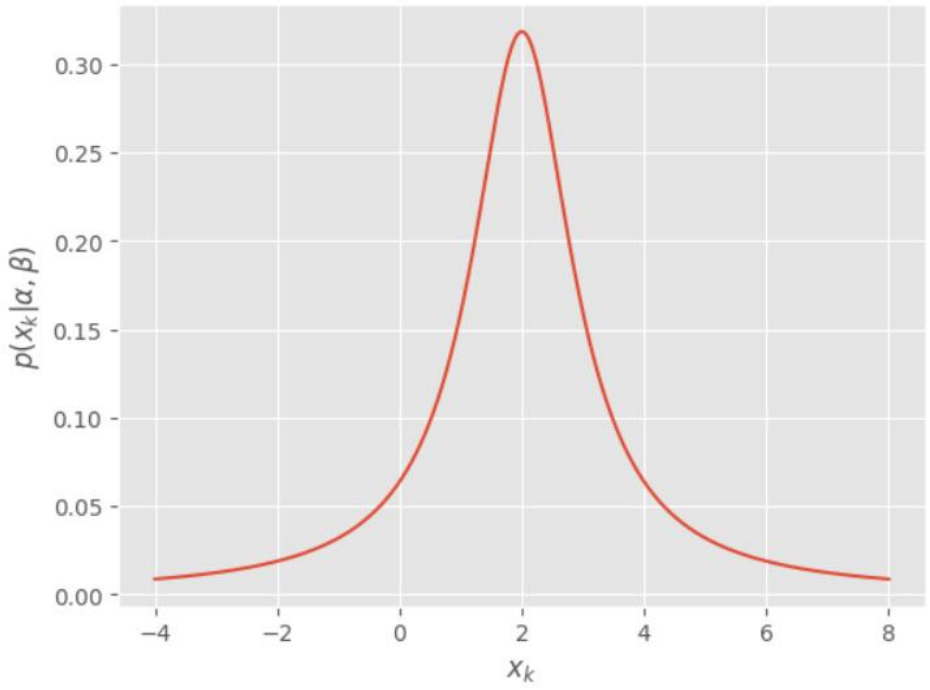
```
# Extract the second element from each list in mus_ml and create a new list
mus2_ml = [mu[1] for mu in mus_ml]


# 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',
```

08




```
# 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()
```

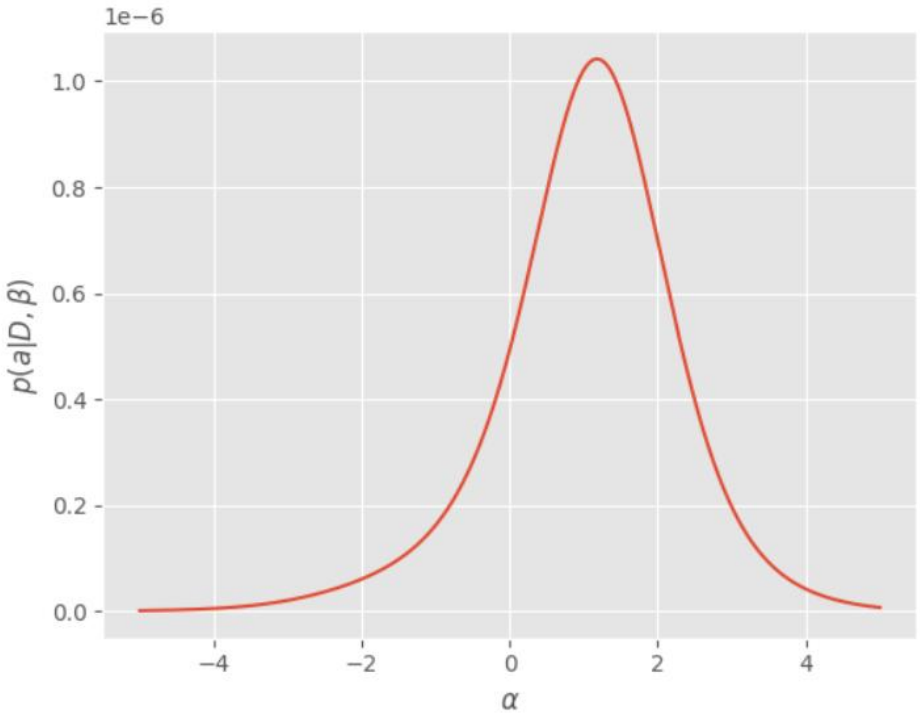
```
# Display the plot.
plt.show()
```




```
# 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

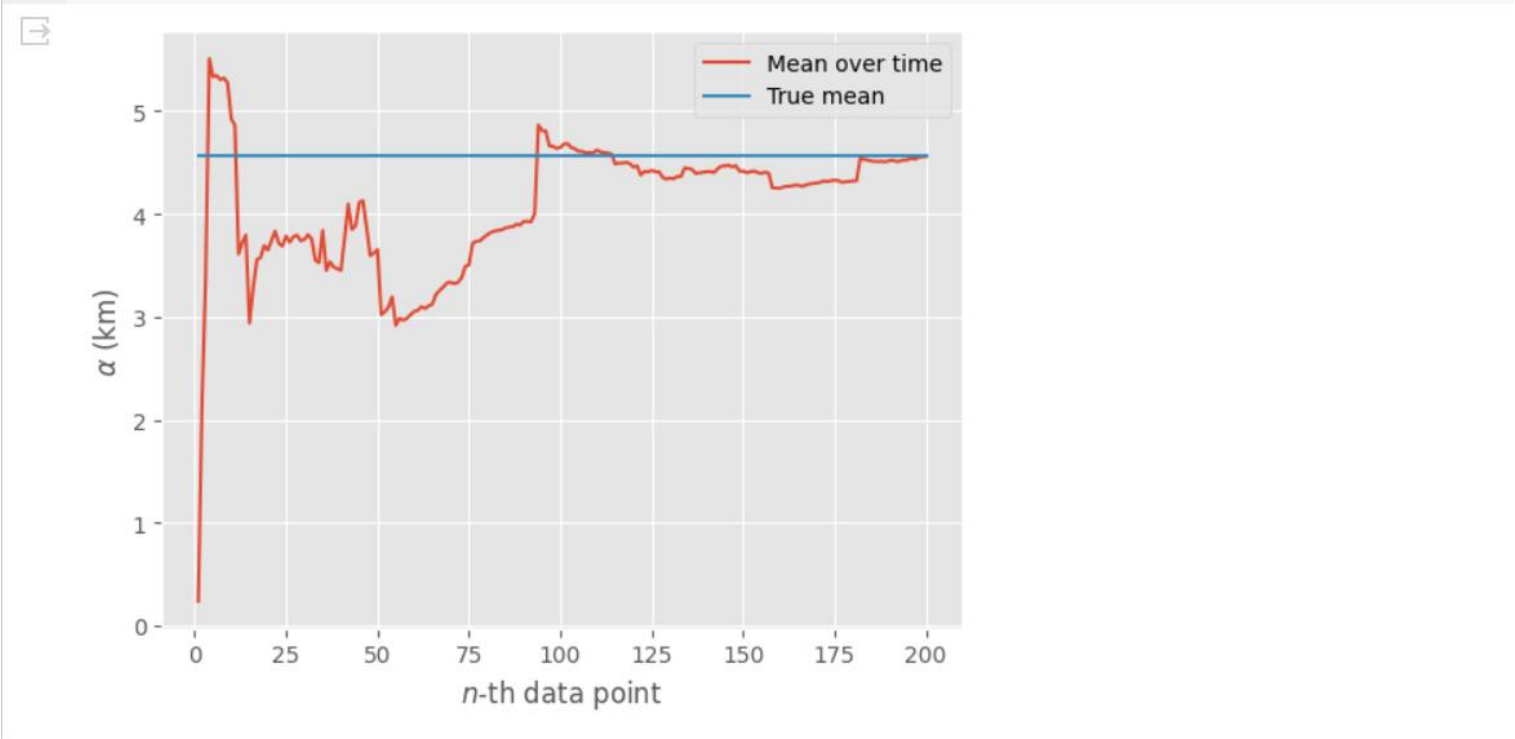
```
-0.18333333333333326
1.1761761761761758
```

```
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()
```

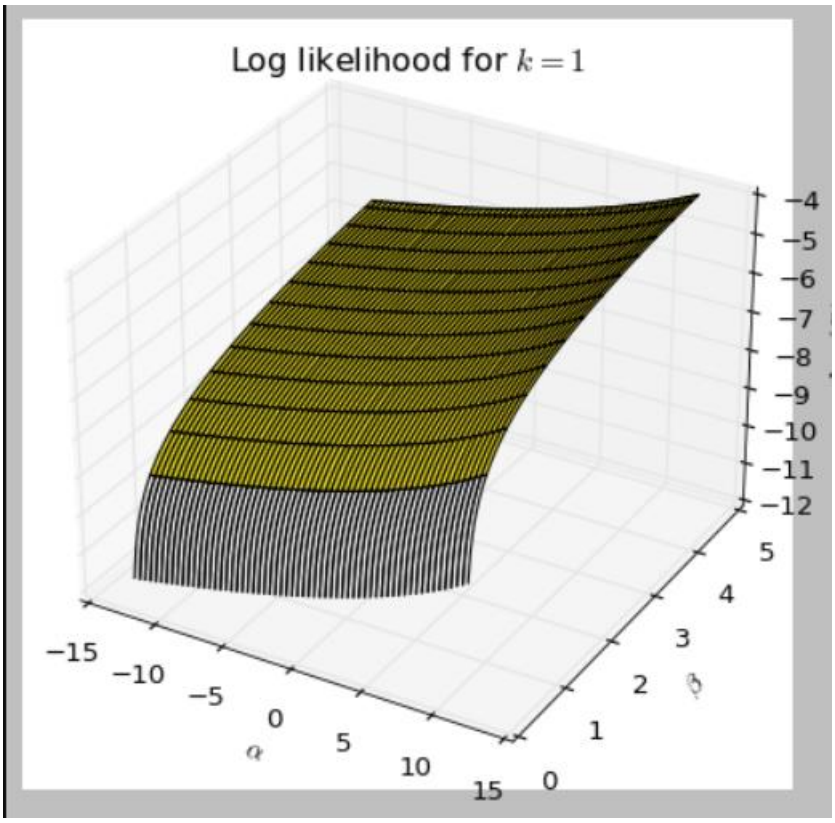


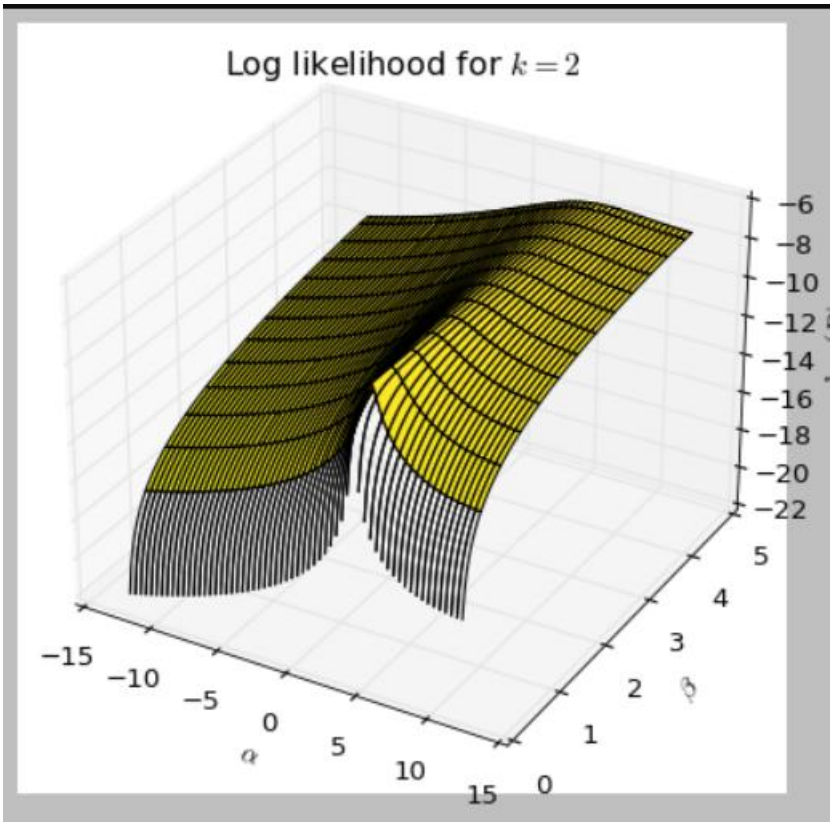

```
# Calculate and print the mean of the 'locations' array.
print(locations.mean())
```

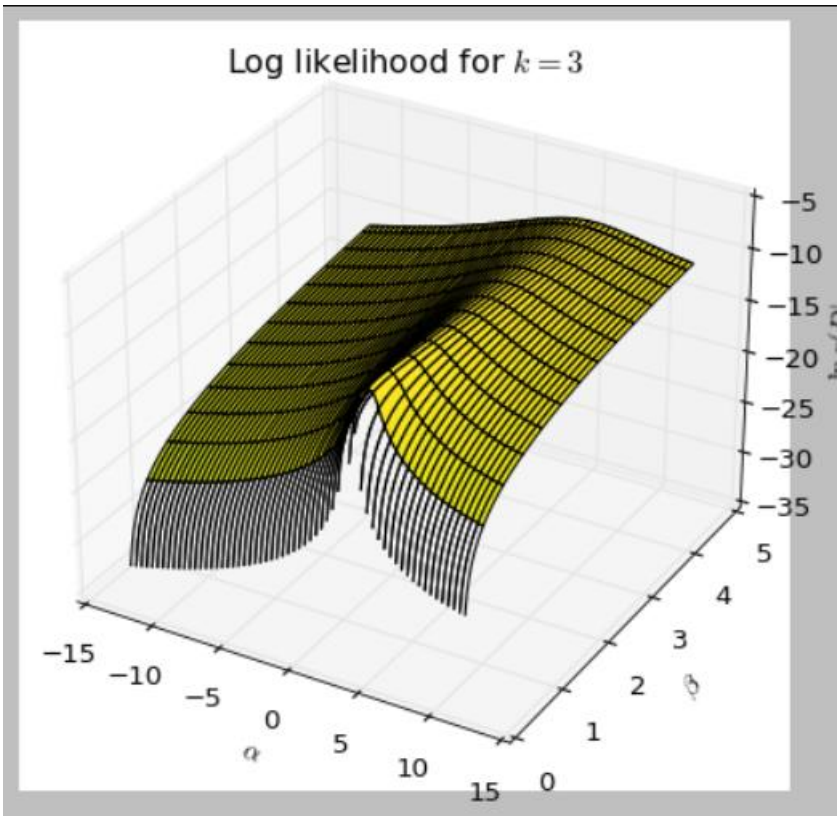
```
4.5594733266299805
```

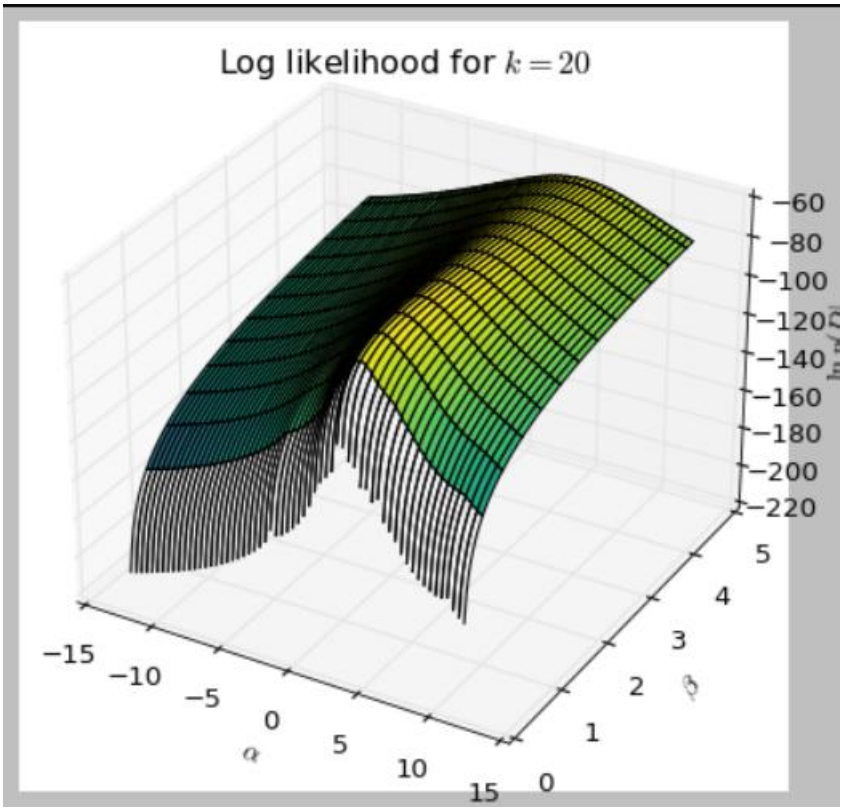
```
# Set the plot style to 'classic'.
plt.style.use('classic')
ks = [1, 2, 3, 20]
# Create a grid of 'alphas' and 'betas' for plotting.
alphas, betas = np.mgrid[-10:10:0.04, 0:5:0.04]
```

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


Log likelihood for $k = 3$
−5
−10
−15
−20
−25
−30
−35
−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])
```

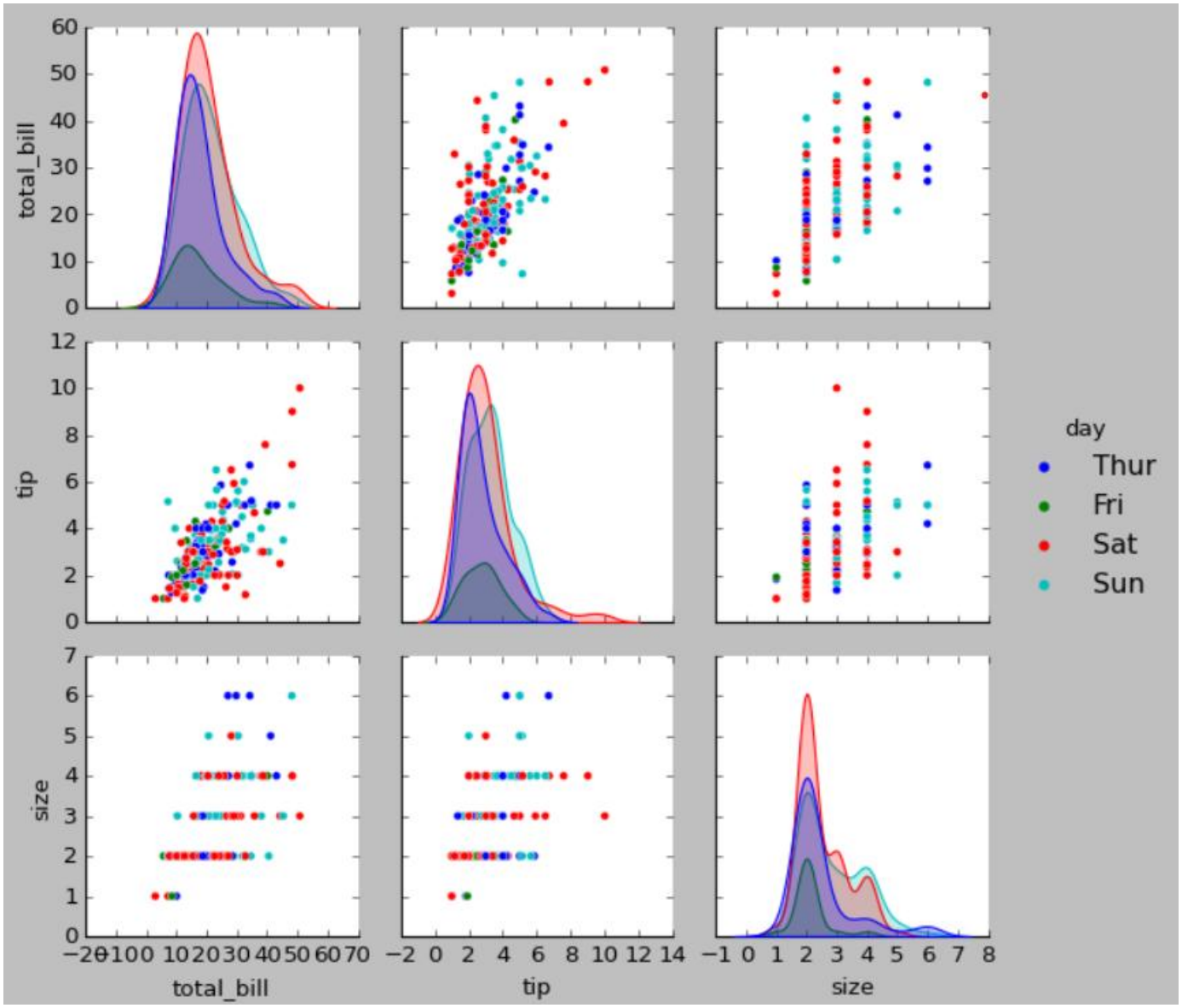
```
         id  year  hour  season  holiday  workingday  weather   temp   atemp  \
0         3  2012    23       3        0           0        2  23.78  27.275
1         4  2011     8       3        0           0        1  27.88  31.820
2         5  2012     2       1        0           1        1 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

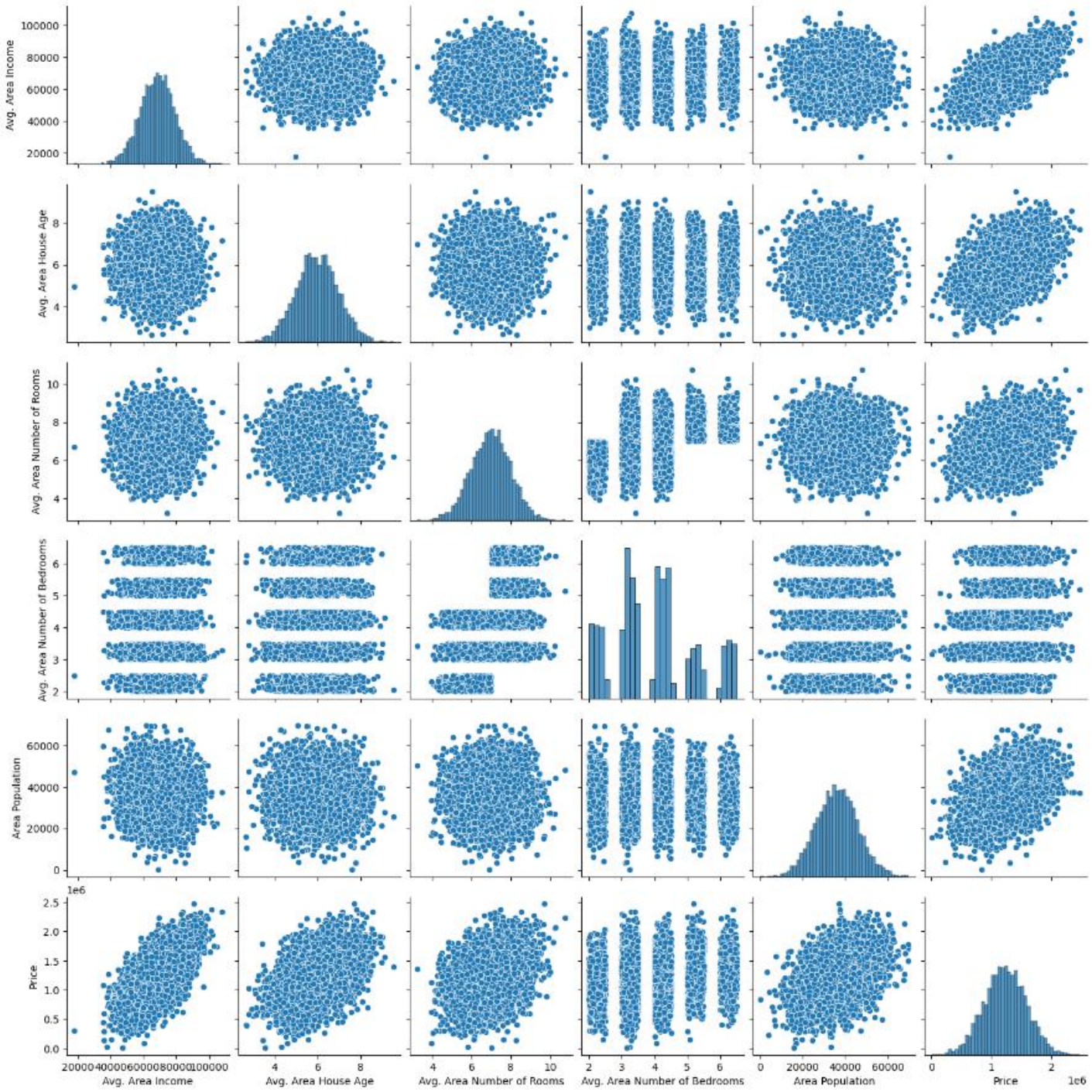
```
# Generate a statistical summary of the DataFrame 'df' with specific percentiles

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')
```

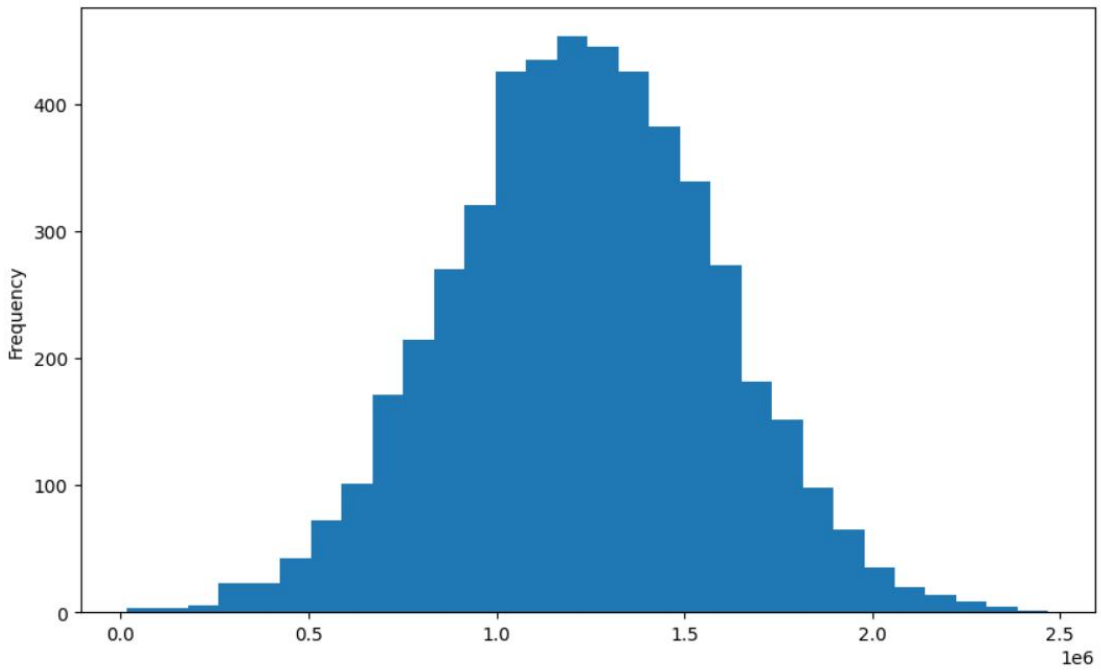
```
# Create a pair plot to visualize relationships between variables in the DataFrame 'df'

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))
```

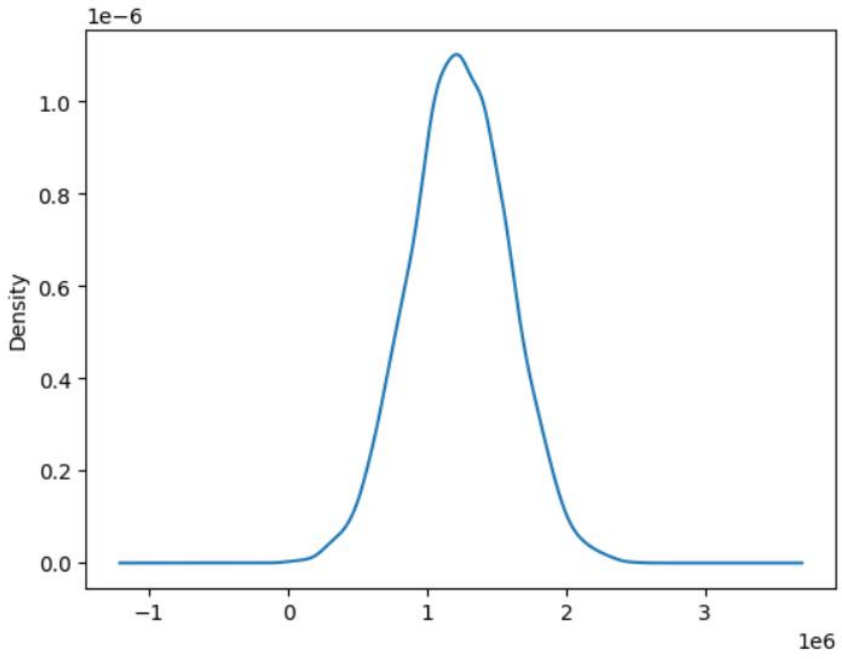
<Axes: ylabel='Frequency'>



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_)
```

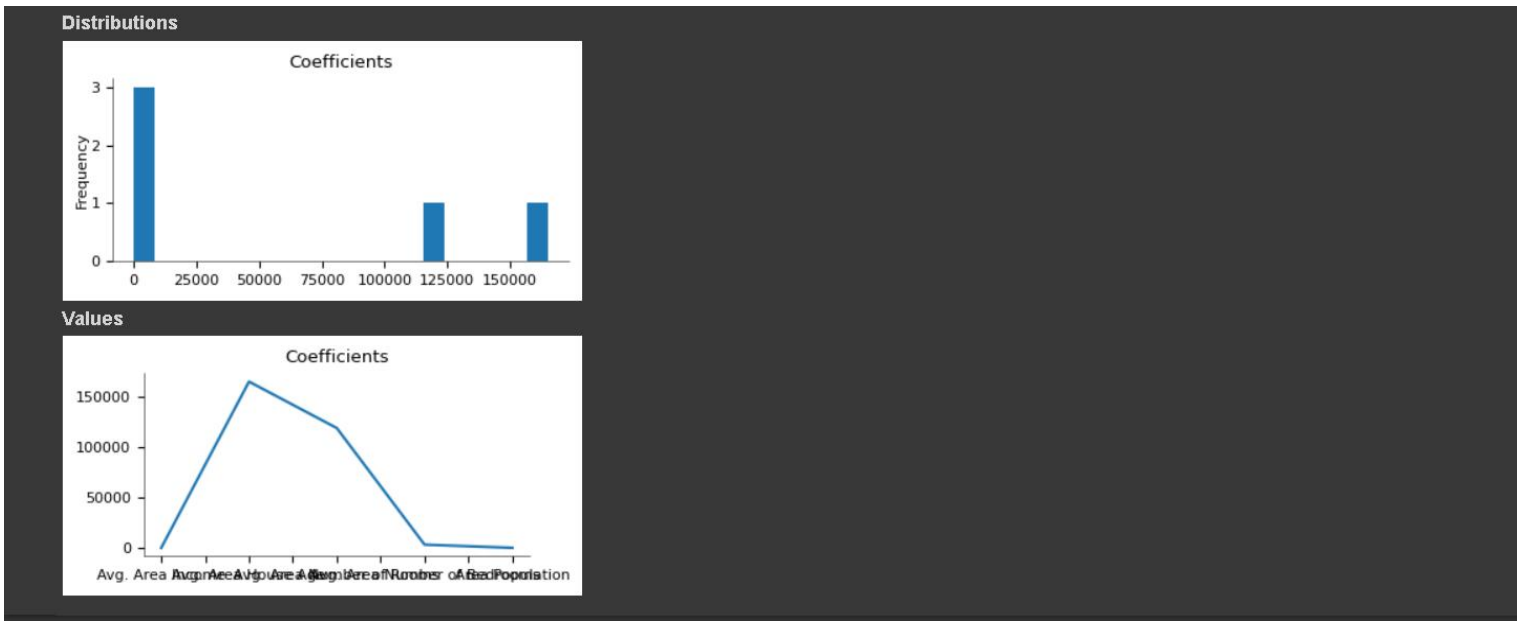
```
The intercept term of the linear model: -2631028.9017454907
```

```
# 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

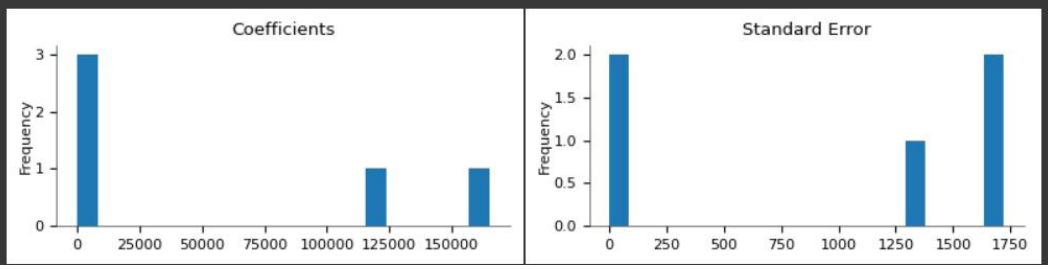


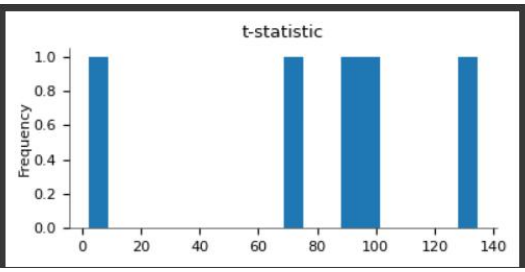

```
# Calculate the standard error for each coefficient
```

```
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
```

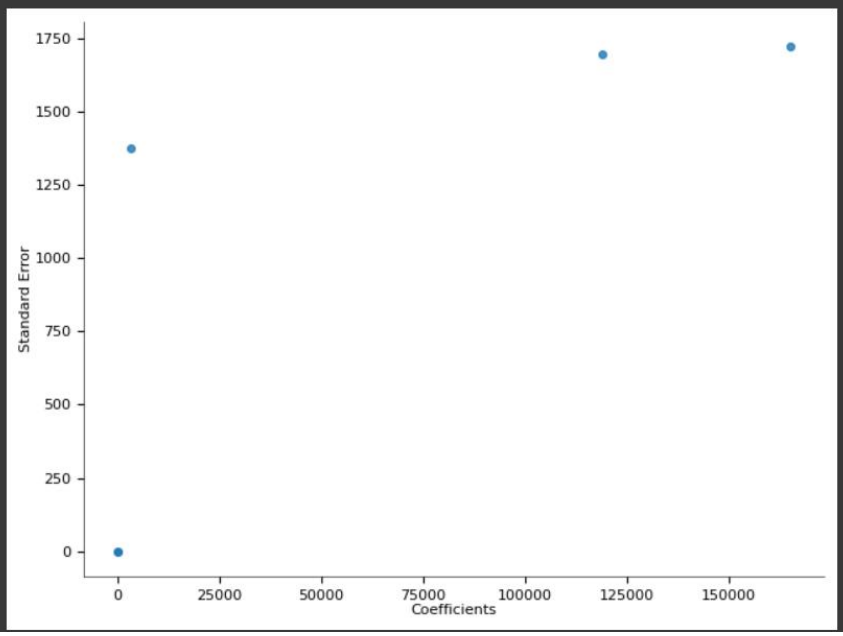
	Coefficients	Standard Error	t-statistic
Avg. Area Income	21.597602	0.160361	134.681505
Avg. Area House Age	165201.104954	1722.412068	95.912649
Avg. Area Number of Rooms	119061.463868	1696.546476	70.178722
Avg. Area Number of Bedrooms	3212.585606	1376.451759	2.333962
Area Population	15.228121	0.169882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

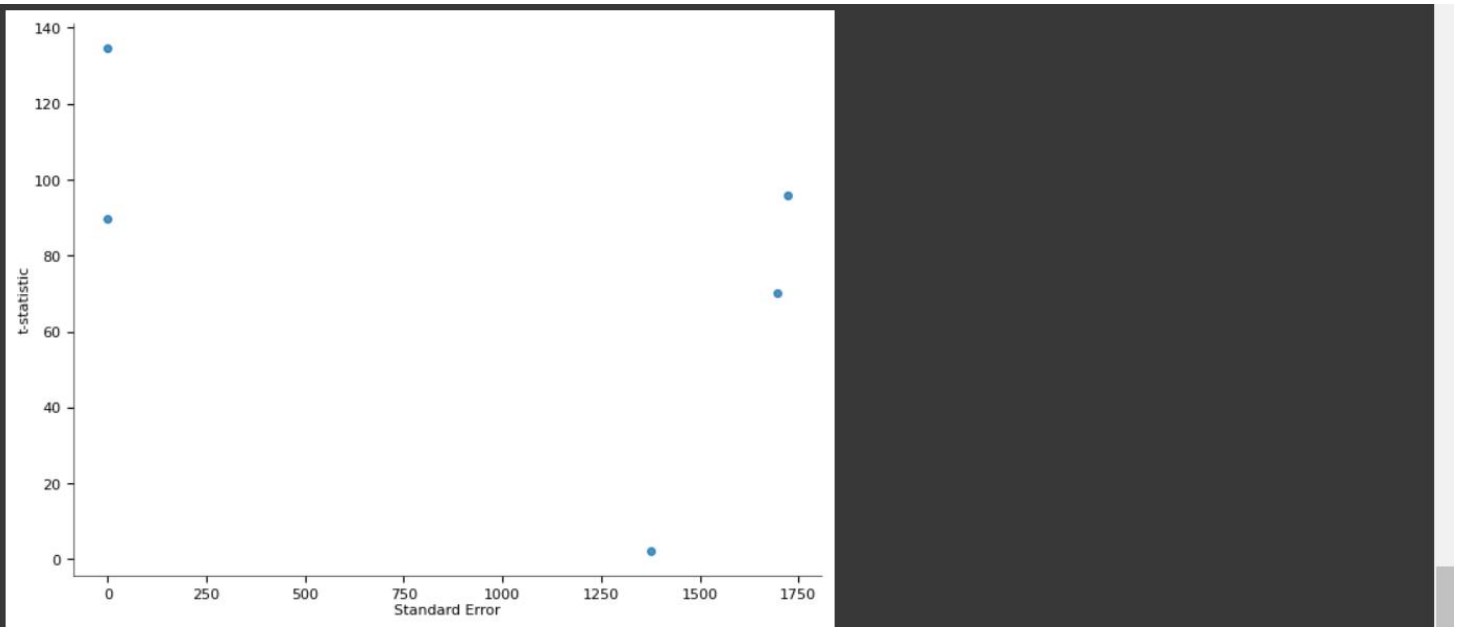
Distributions

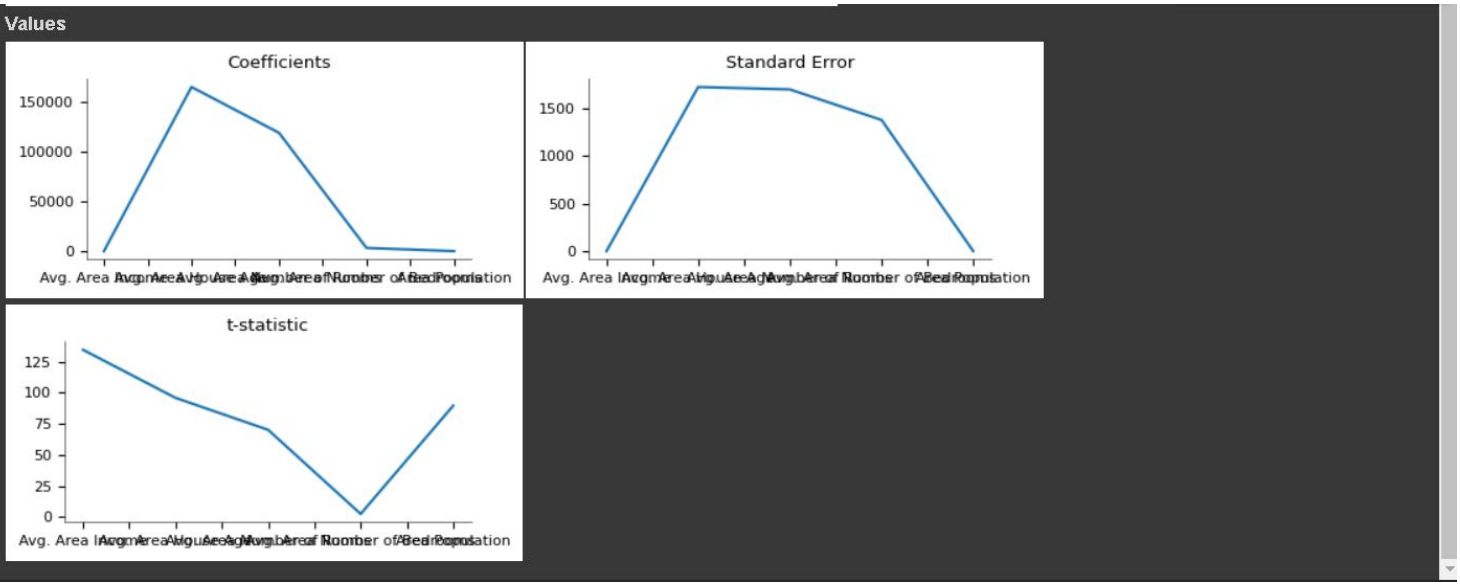




2-d distributions







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

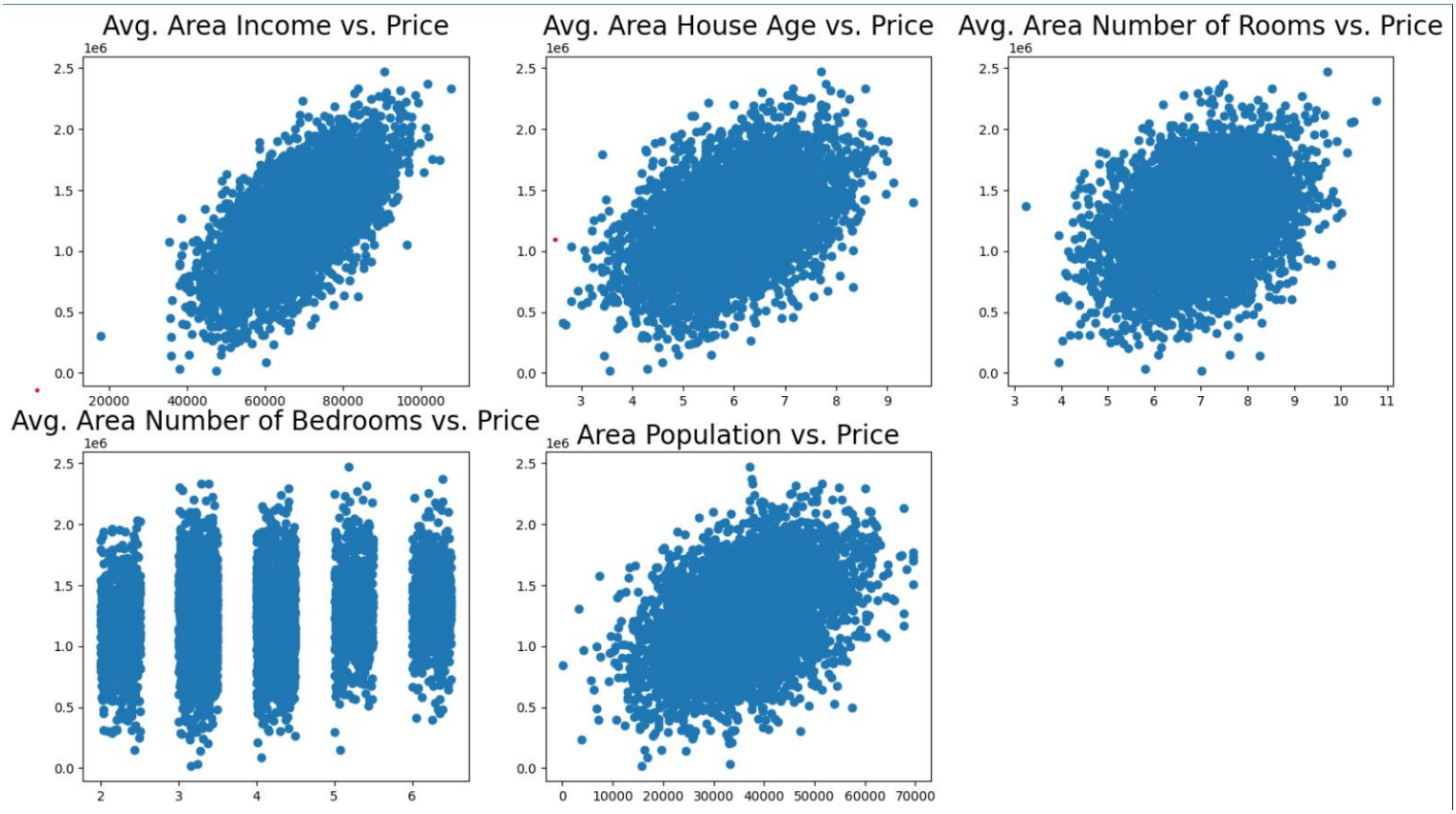
l = list(cdf.sort_values('t-statistic', ascending=False).index)

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
```

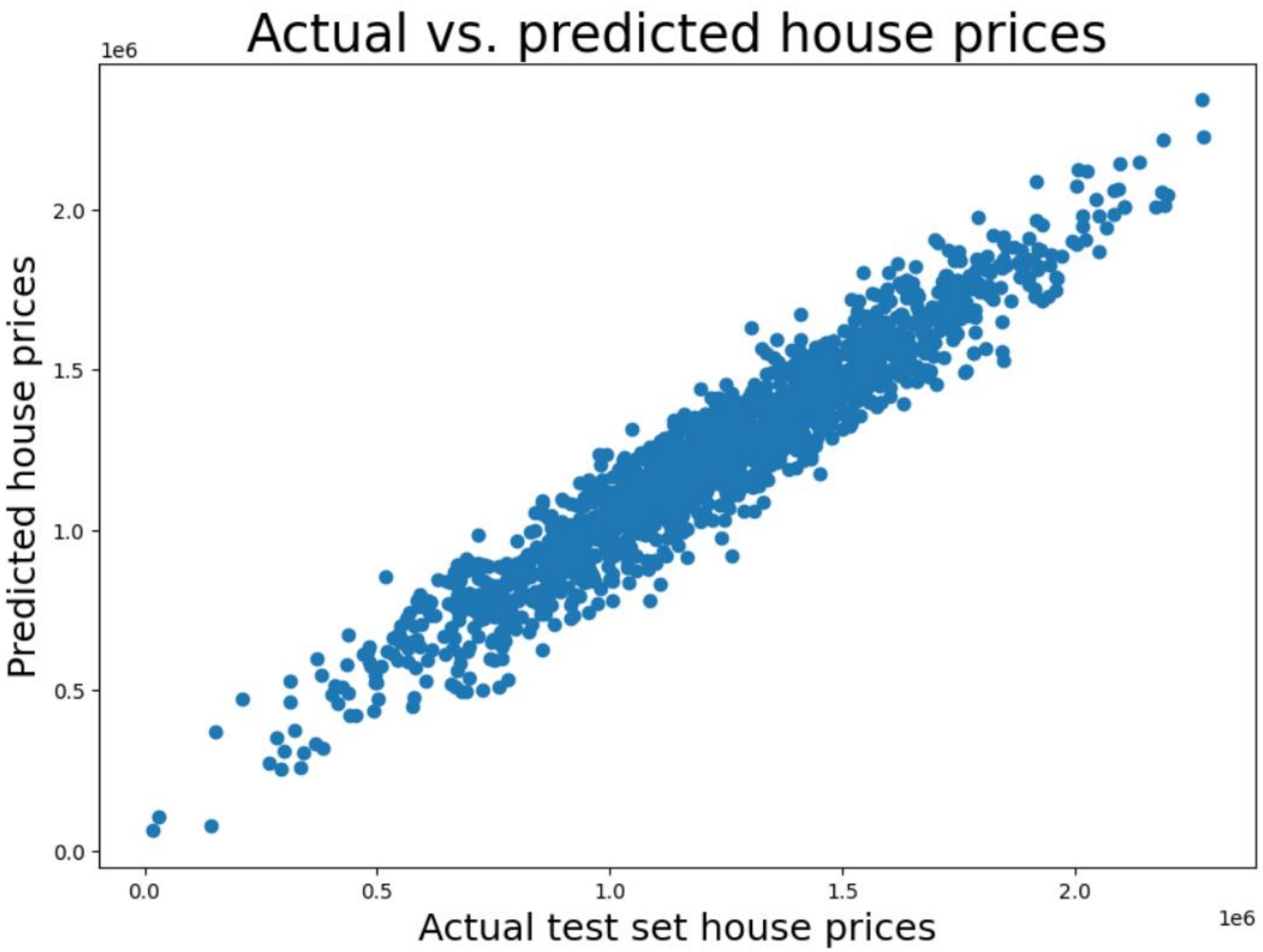
```
# Generate predictions using the linear regression model on the test data
predictions = lm.predict(X_test)
```

```
# 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)
```

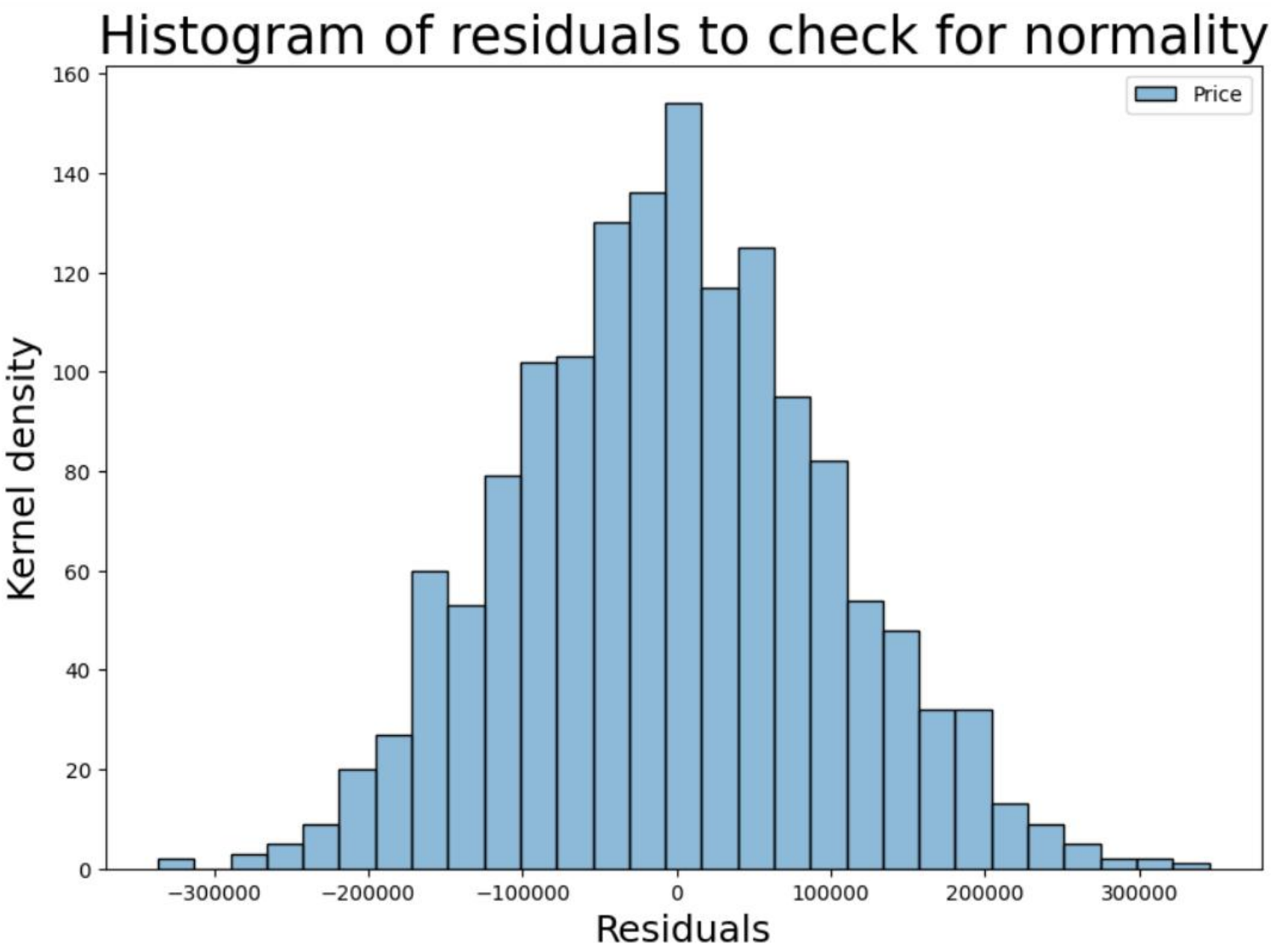
```
Type of the predicted object: <class 'numpy.ndarray'>
Size of the predicted object: (1500,)
```

```
# 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)
```

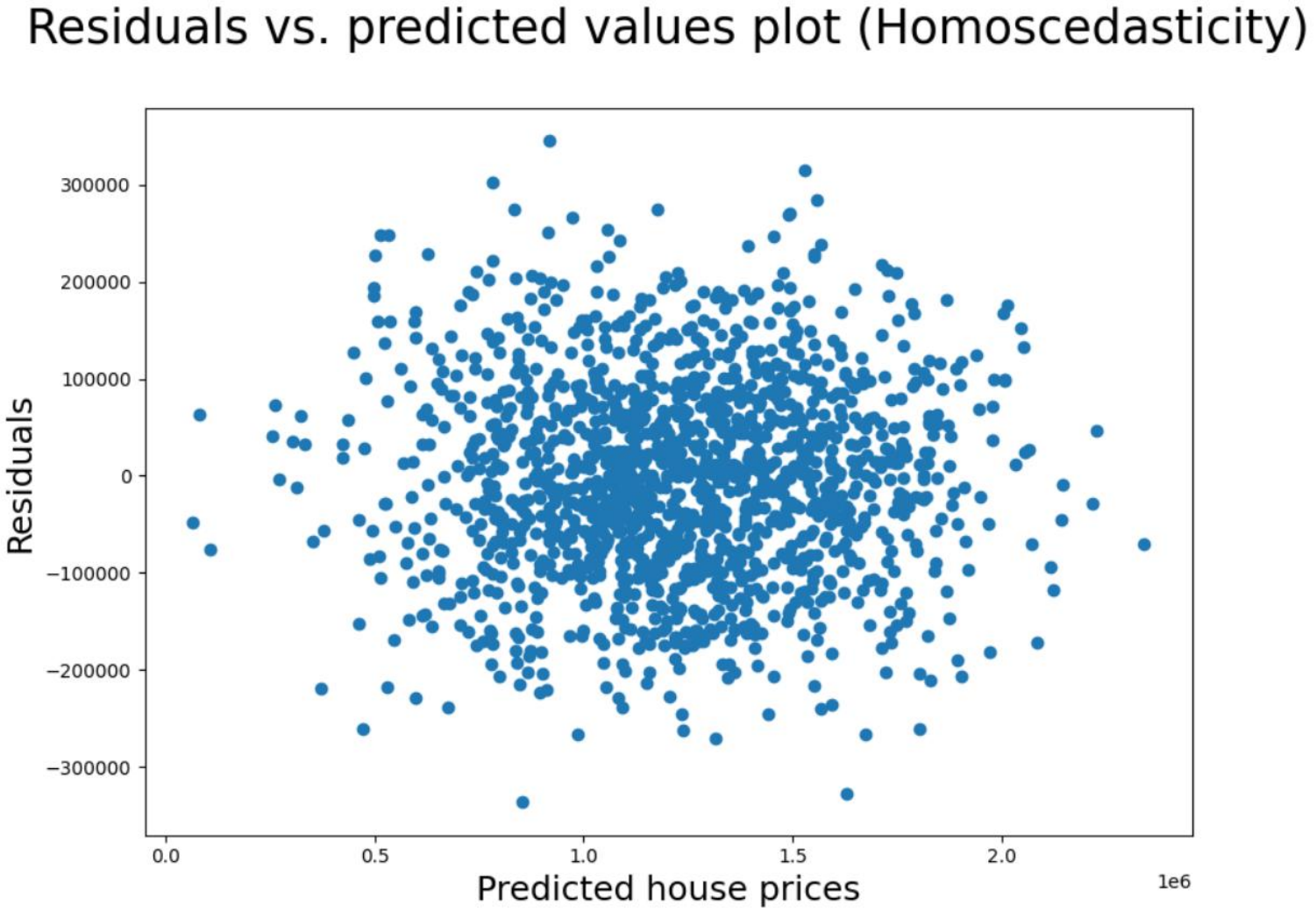



```
# Create a figure for the histogram and kernel density plot
plt.figure(figsize=(10, 7))

# 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)
```

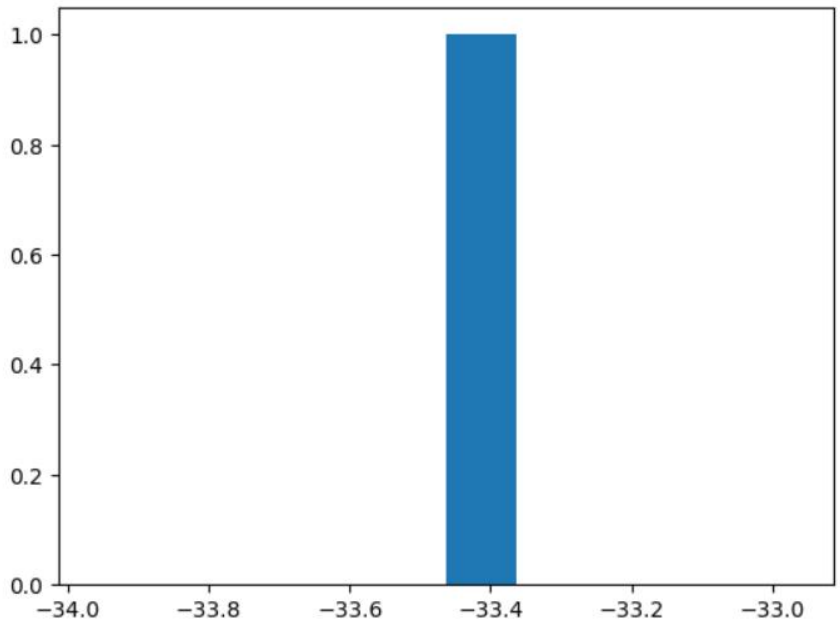
```
# Print the minimum and maximum scaled values
print(min, max)
```

```
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

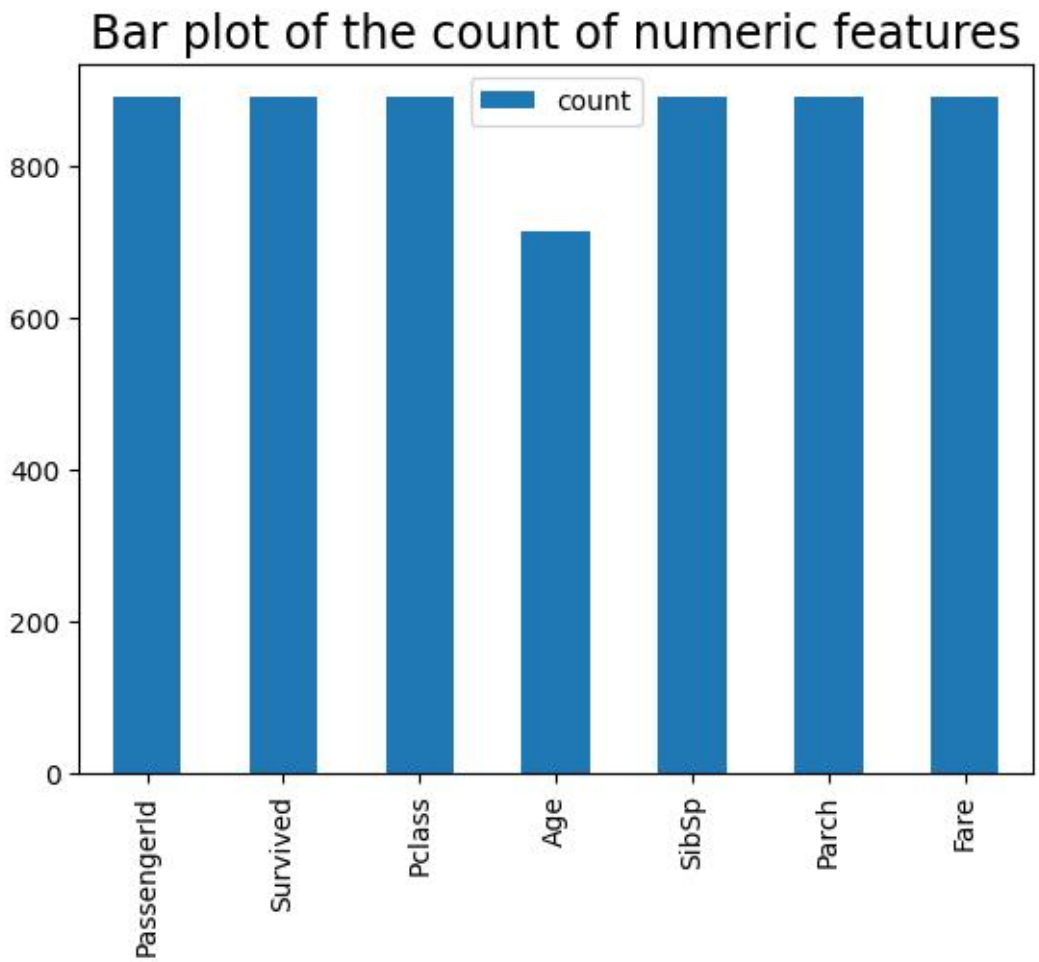
```
# Get information about the dataset, such as column data types and non-null counts
t=train.info()
```

```
<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
```

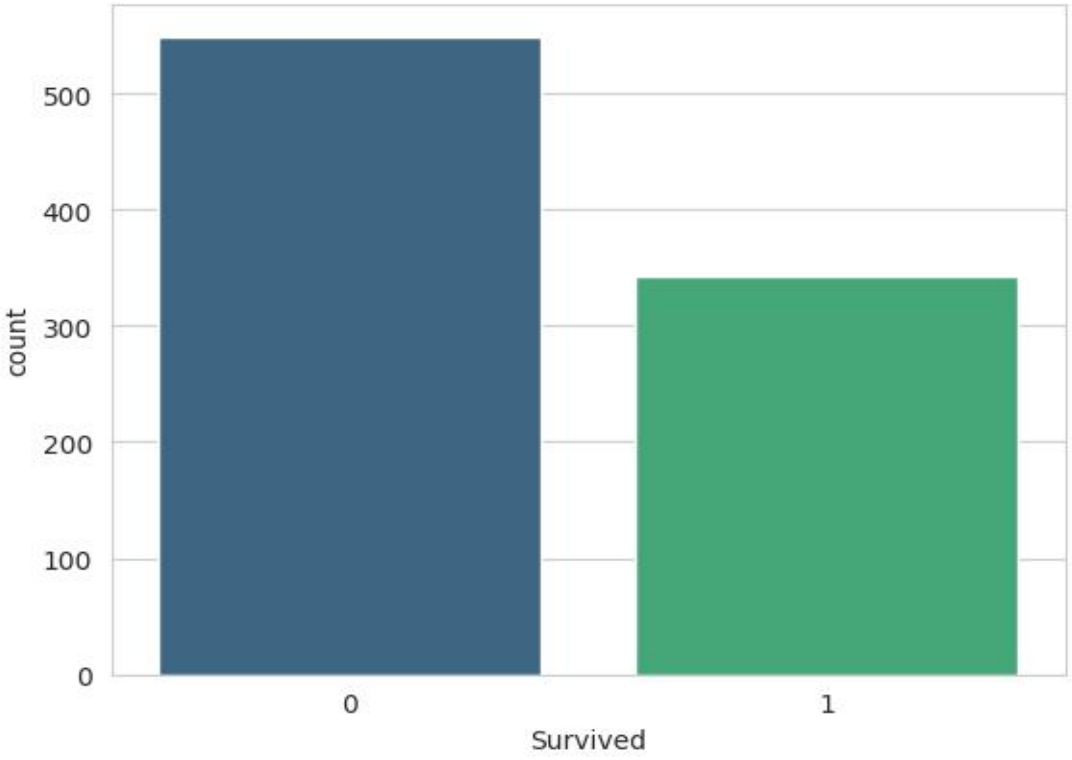
```
# Generate basic statistics for the numeric columns in the dataset
d=train.describe()
```

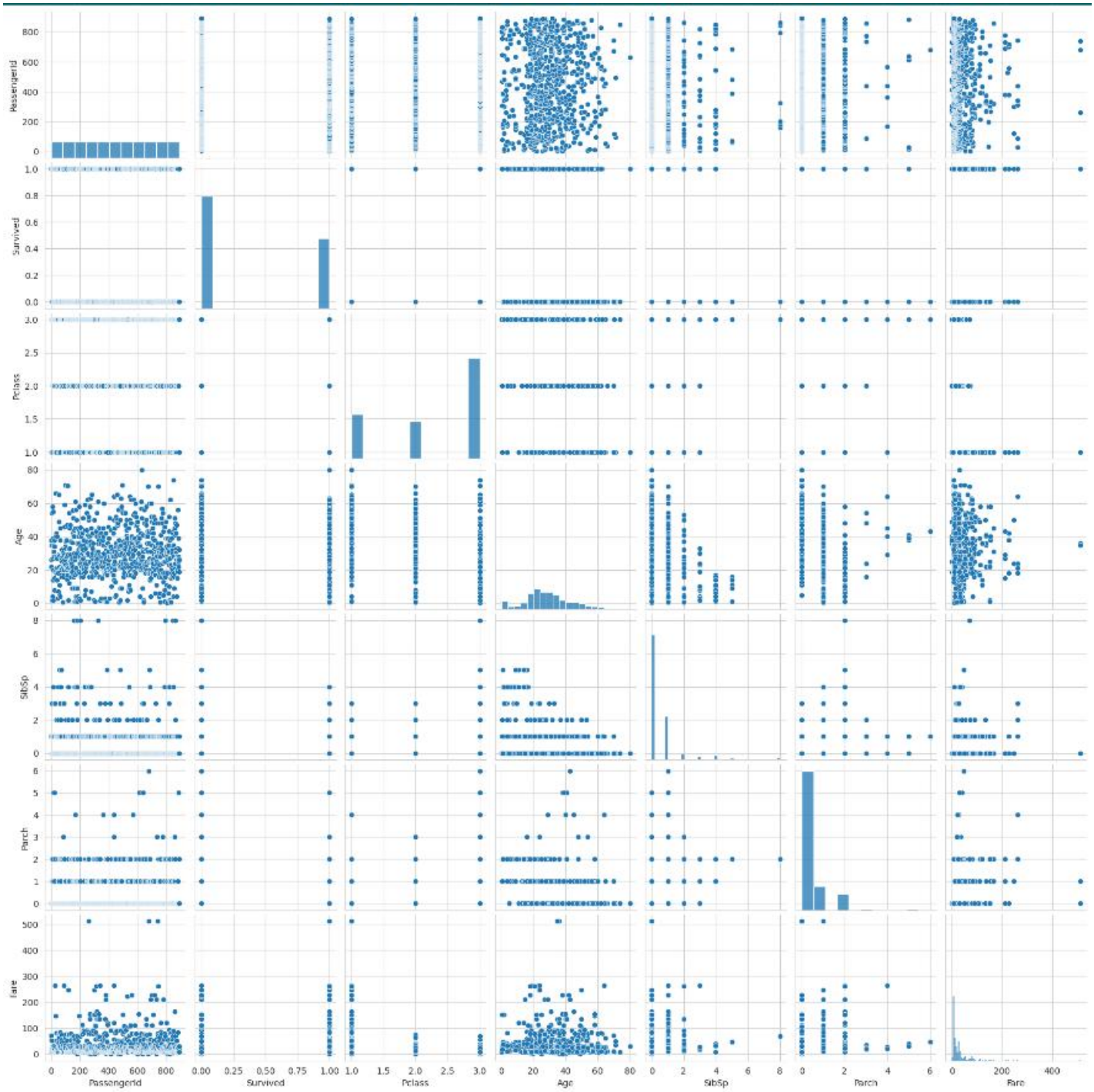
```
# Transpose the summary statistics for better visualization
dT = d.T

# 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')
```



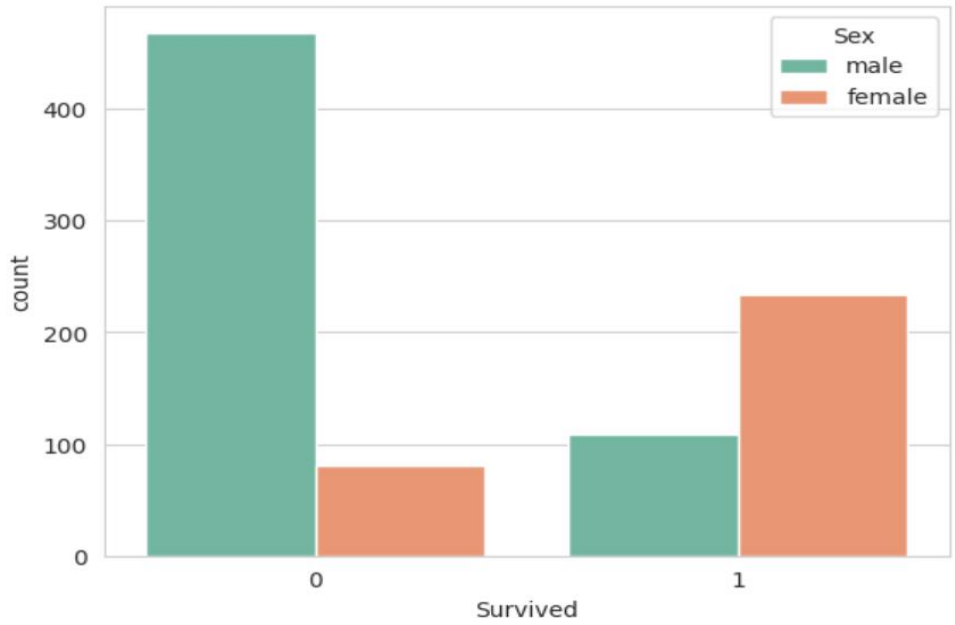

```
# Set the style for Seaborn plots
sns.set_style('whitegrid')

# 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



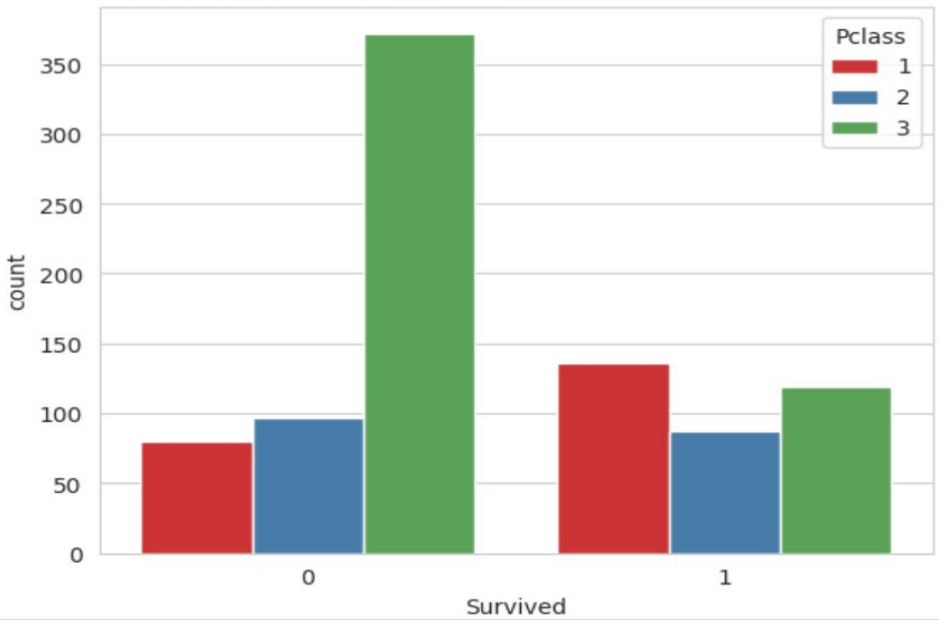

```
# Set the style for Seaborn plots
sns.set_style('whitegrid')
# Create a count plot to visualize the distribution of 'Survived' with 'Sex' as a hue
sns.countplot(x='Survived',hue='Sex',data=train,palette='Set2')
```

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



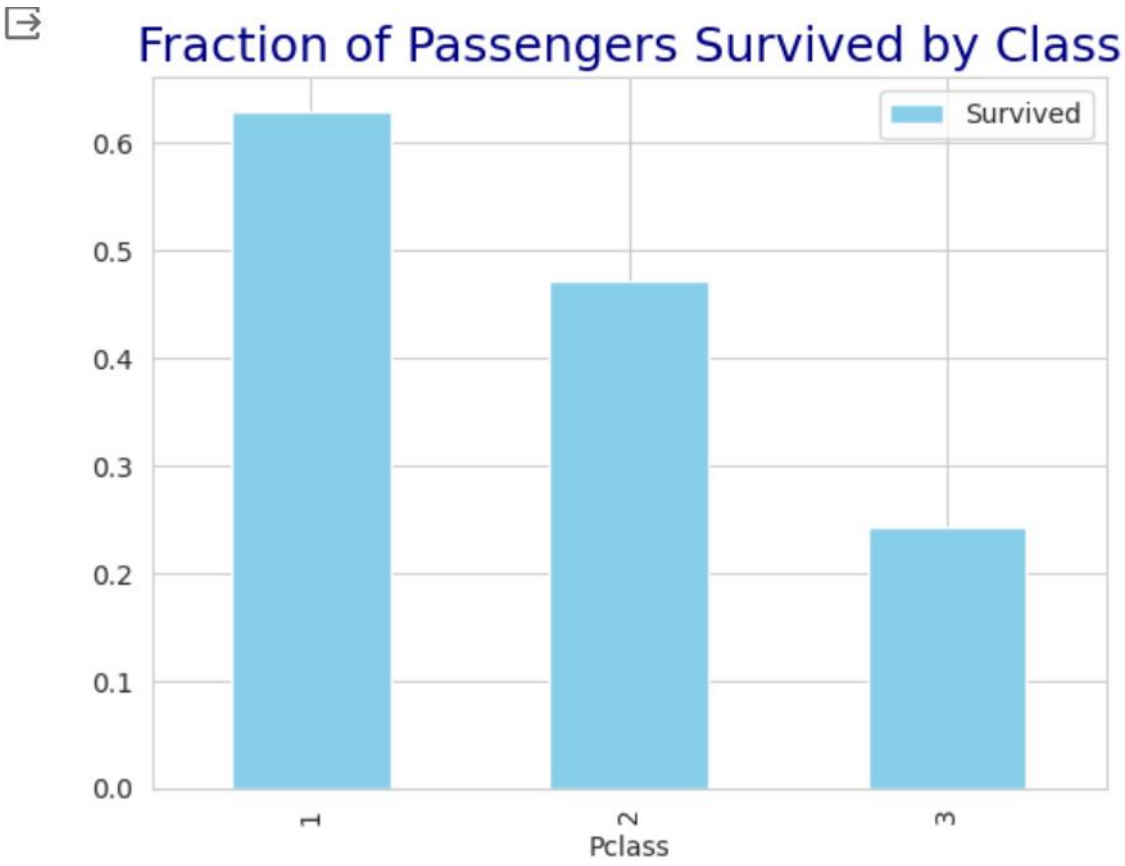

```
# Set the style of the seaborn plots to 'whitegrid'.
sns.set_style('whitegrid')

# 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)
```

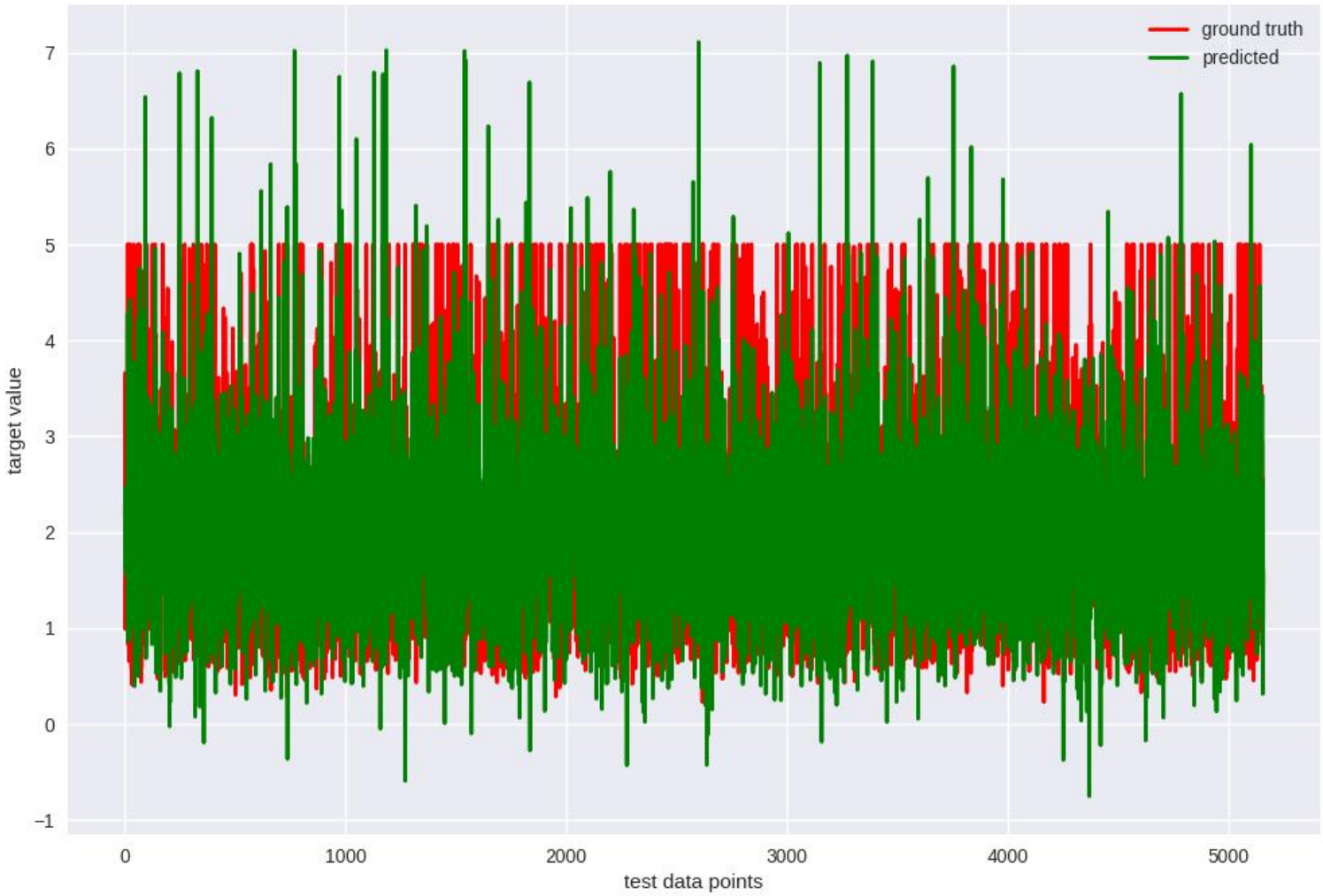
```
0.6040686577008176
```

```
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')
```

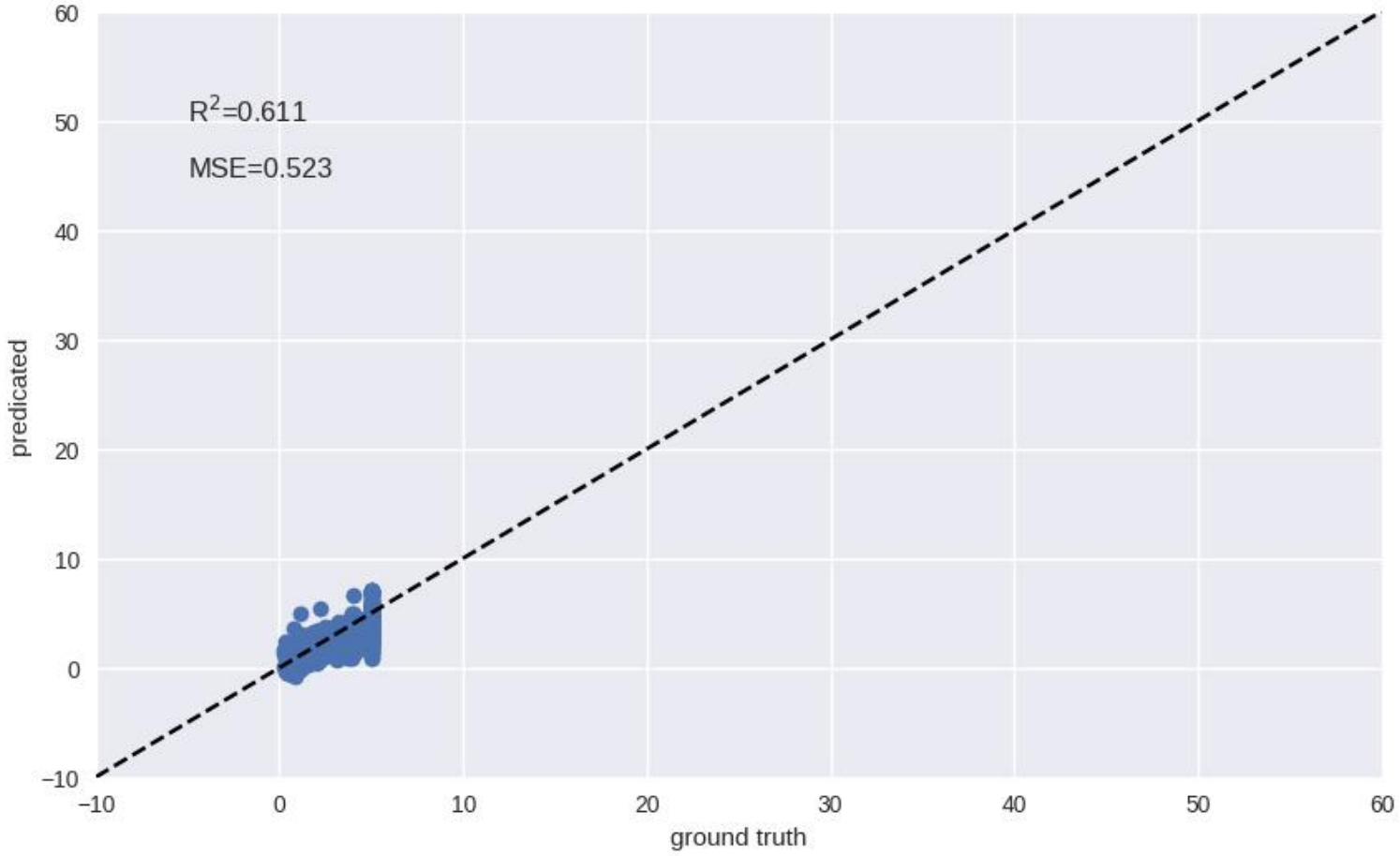
```
Text(0, 0.5, 'target value')
```




```
# 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')
```



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
```

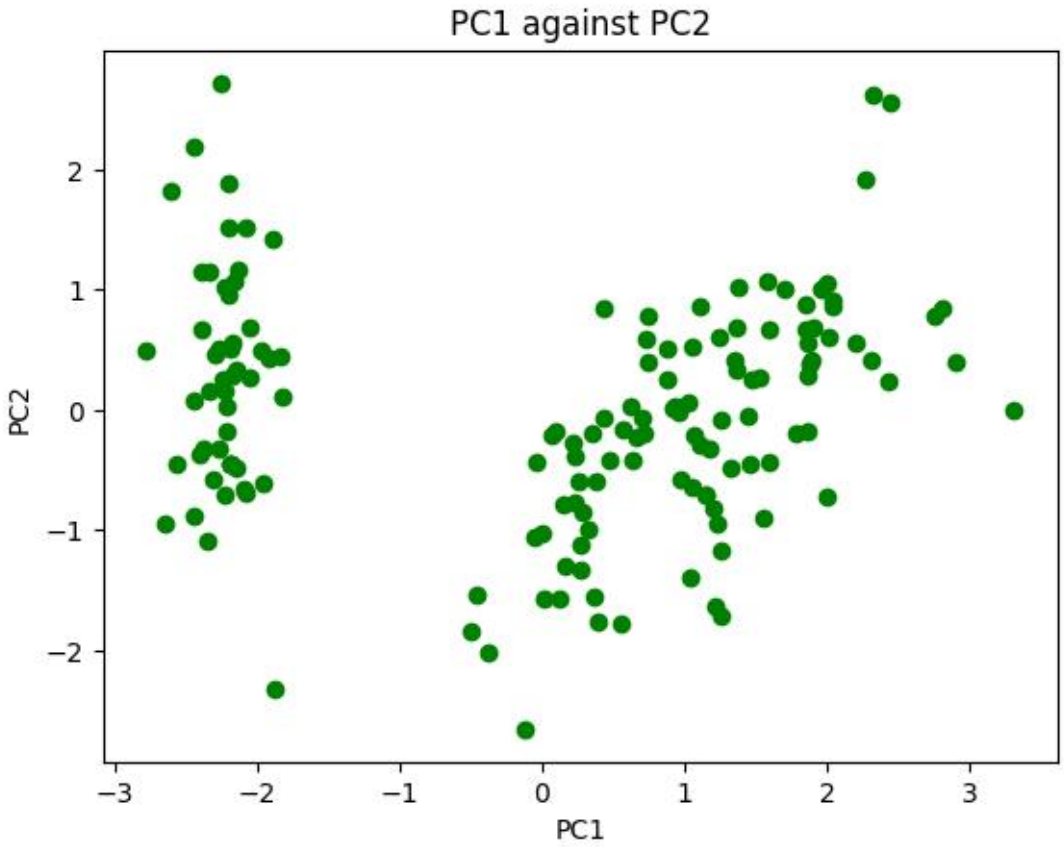
	PC1	PC2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa
...	...	...	...
145	1.870522	0.382822	Iris-virginica
146	1.558492	-0.905314	Iris-virginica
147	1.520845	0.266795	Iris-virginica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

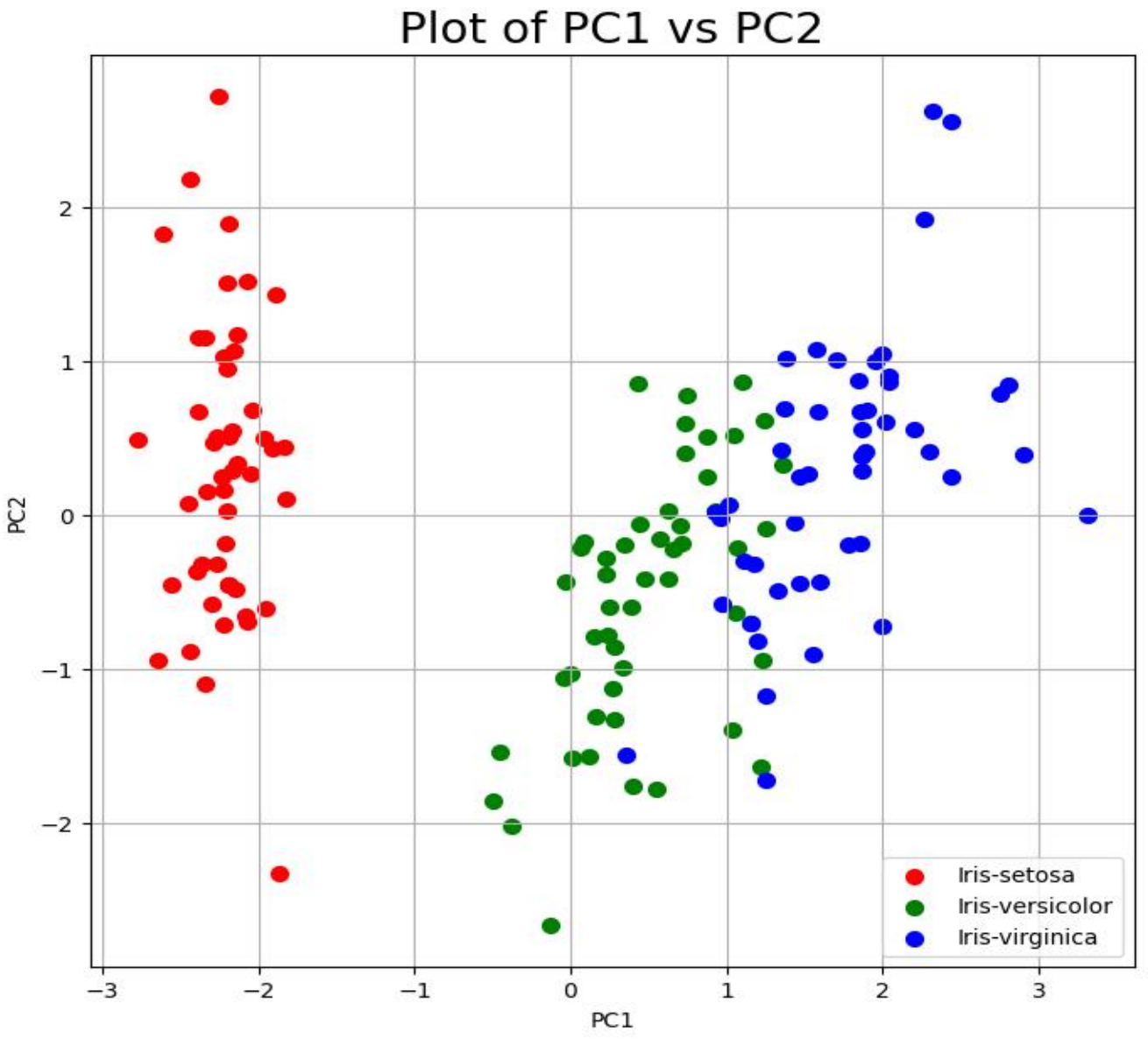
```
plt.scatter(principalDataframe.PC1,principalDataframe.PC2,color='green')


# 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_
```

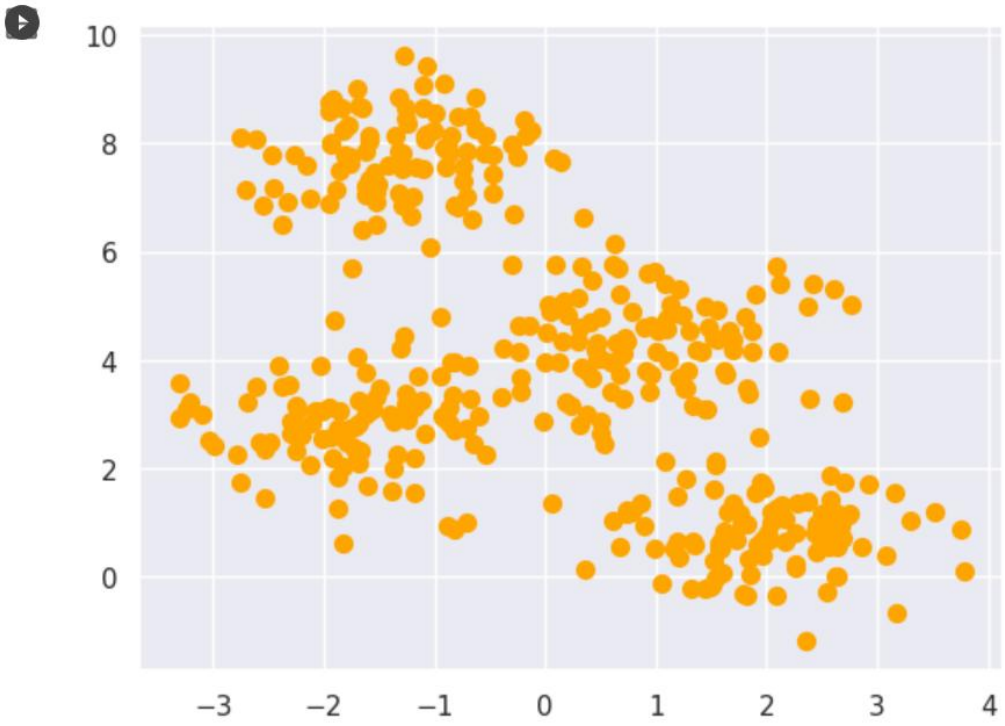
```
array([0.72770452, 0.23030523])
```

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');
```

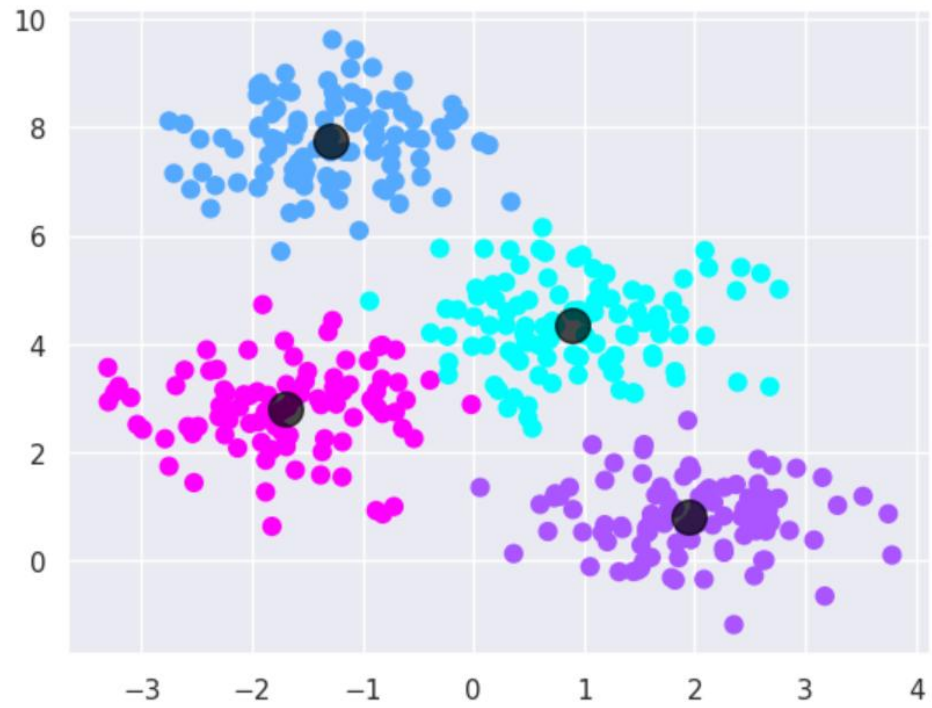



```
# Import the KMeans class from scikit-learn for clustering.
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, n_init=10)
```

```
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)
```

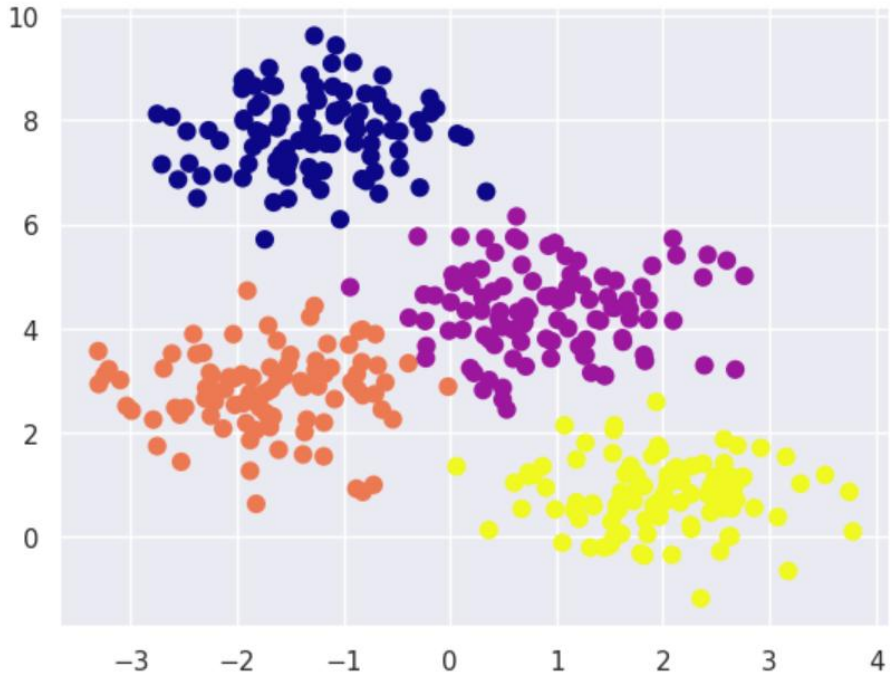
```
<matplotlib.collections.PathCollection at 0x793d408ad4e0>
```




```
# 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')
```

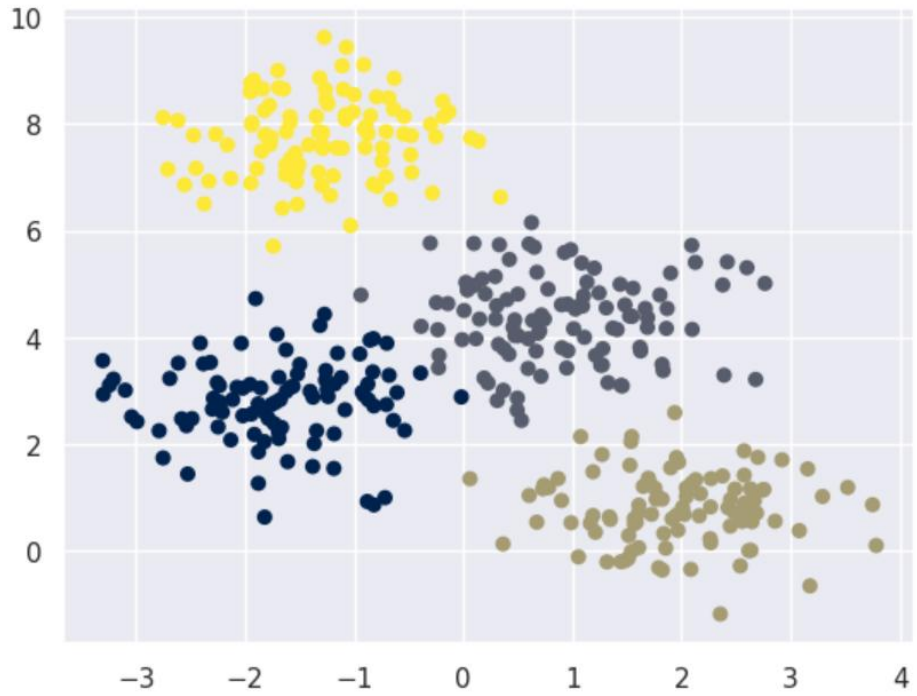
```
<matplotlib.collections.PathCollection at 0x793d3bee13c0>
```




```
# 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)
```

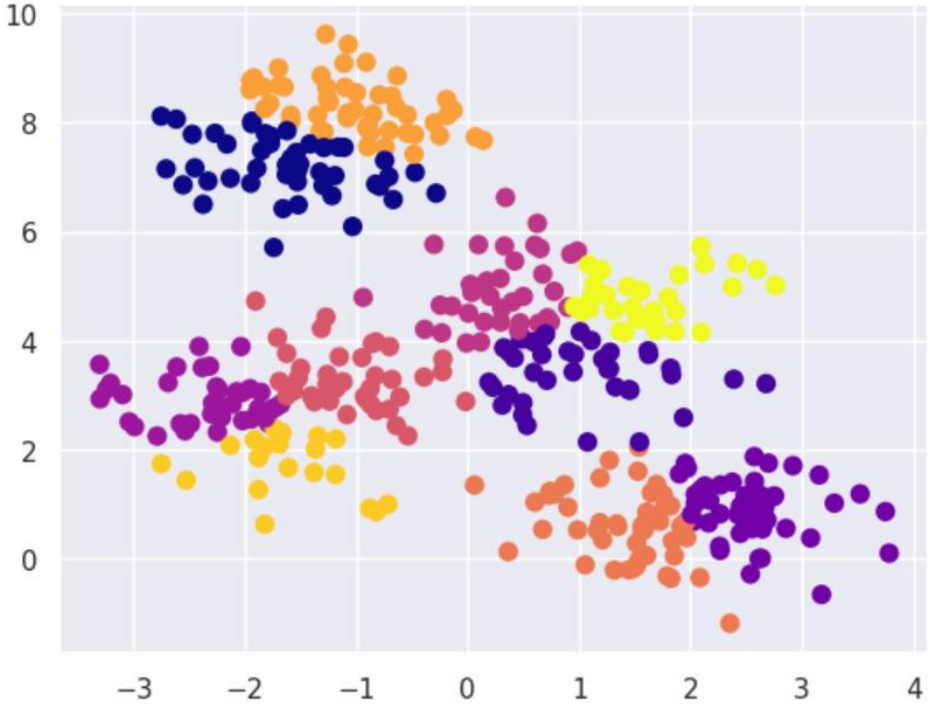
```
# Create a scatter plot of data points with color-coded labels.
# Change the 's' parameter to adjust the size of the points, and 'cmap' to change the color map.
plt.scatter(X[:, 0], X[:, 1], c=labels, s=30, cmap='cividis')
```

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



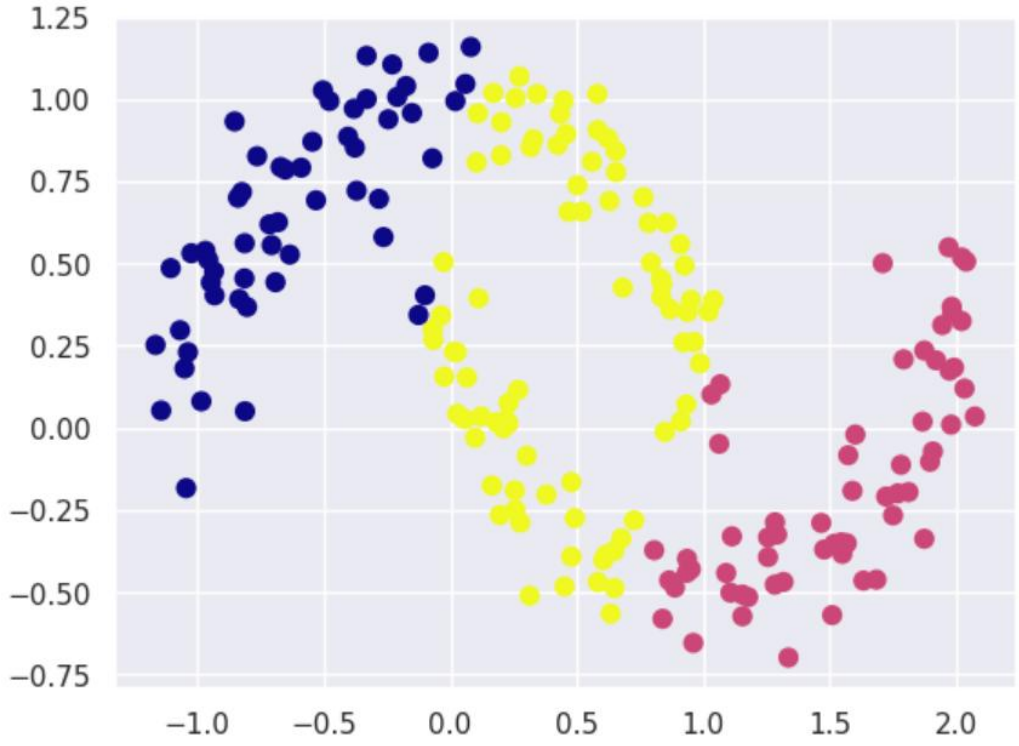

```
# Apply KMeans clustering with 10 clusters and 10 initializations.
# Assign cluster labels to data points.
labels = KMeans(n_clusters=10, random_state=0, n_init=10).fit_predict(X)

# 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>
```

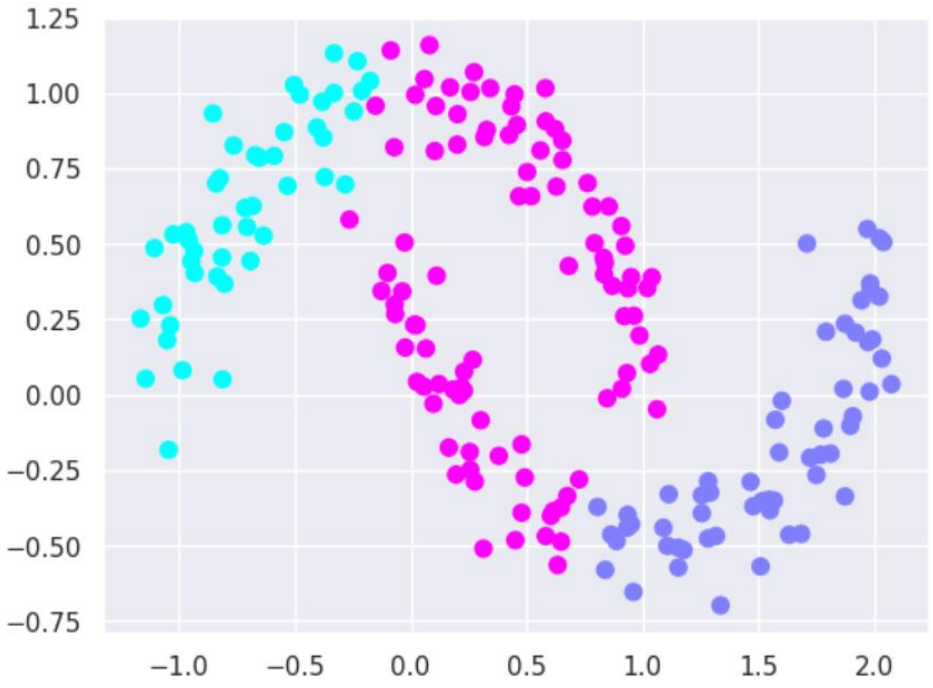



```
# Import the 'make_moons' function from sklearn.datasets to generate a synthetic dataset.
from sklearn.datasets import make_moons
# Generate a synthetic dataset with 200 data points and a small amount of noise (0.05).
X, y = make_moons(n_samples=200, noise=0.1, random_state=0)
labels = KMeans(n_clusters=3, 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');
```


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

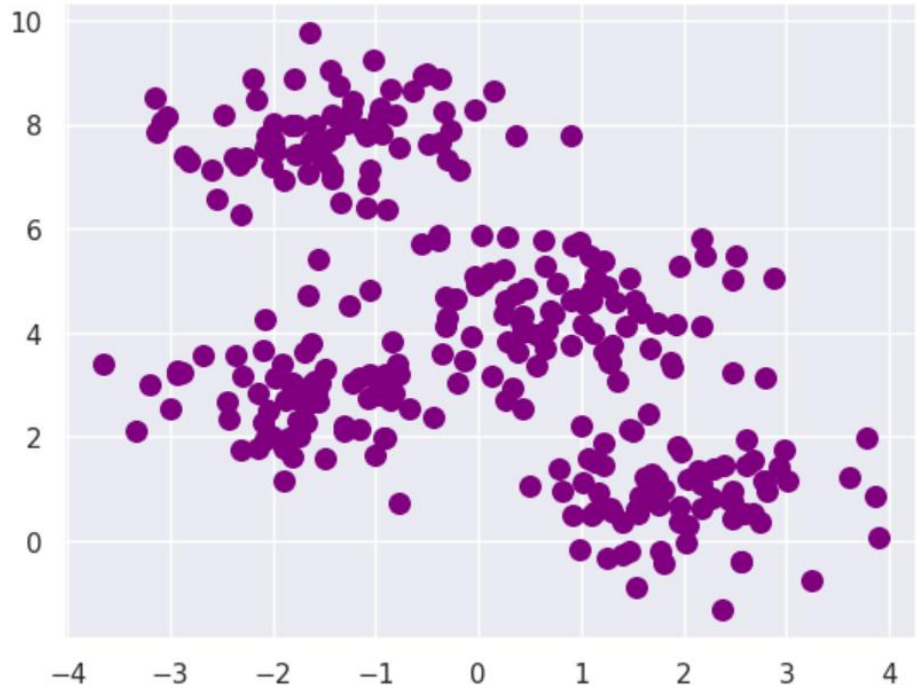
LAB-8

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
```

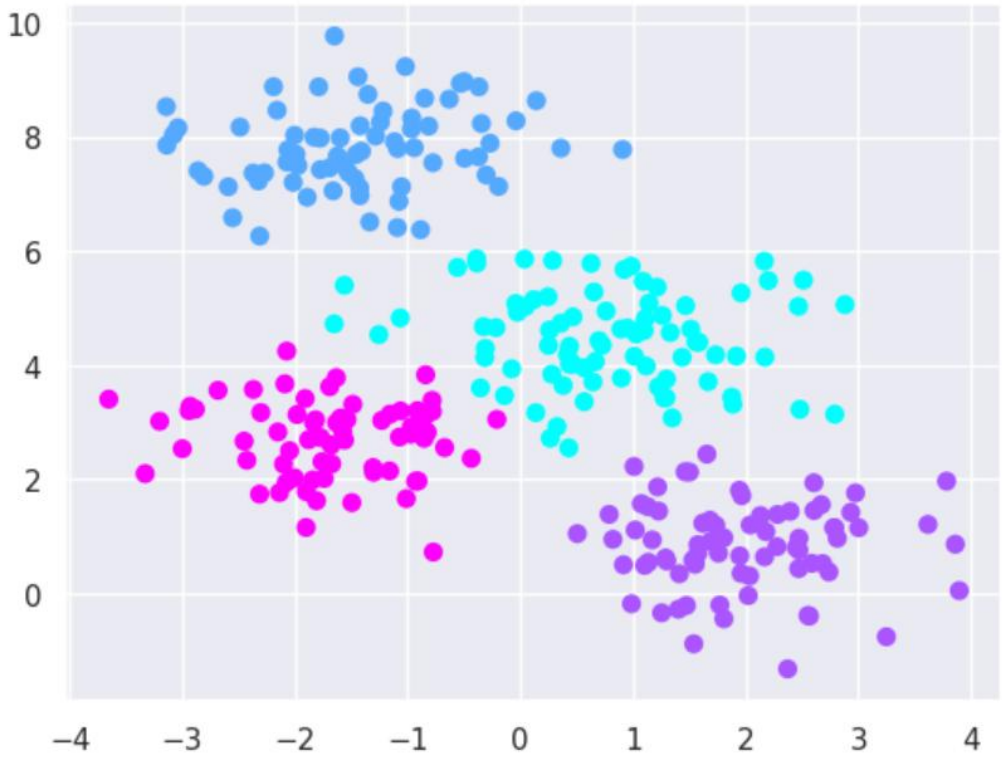
```
# Import the necessary modules.
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# 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')
```

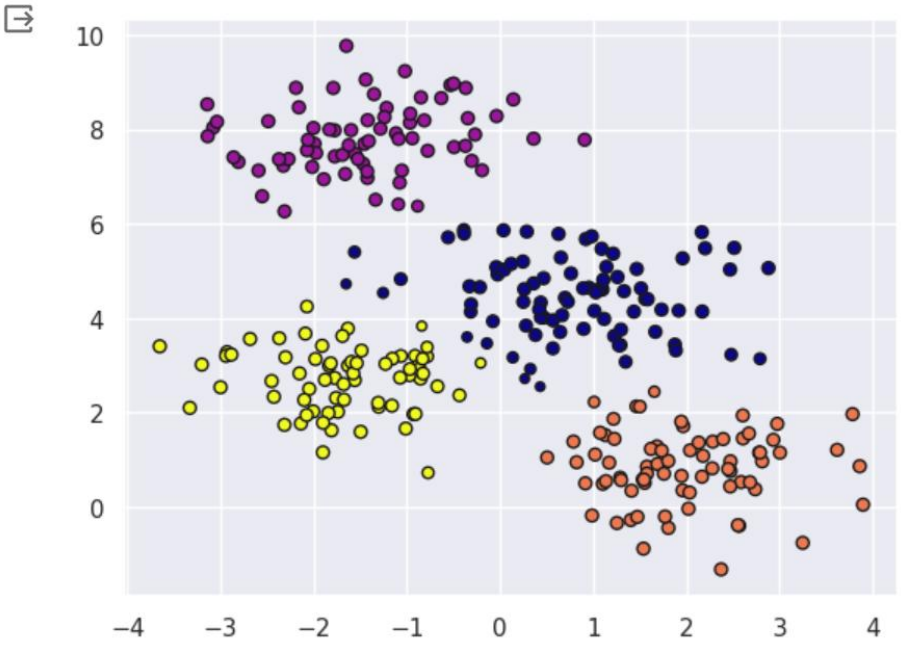
```
<matplotlib.collections.PathCollection at 0x7b28f84f6c20>
```




```
# 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))
```

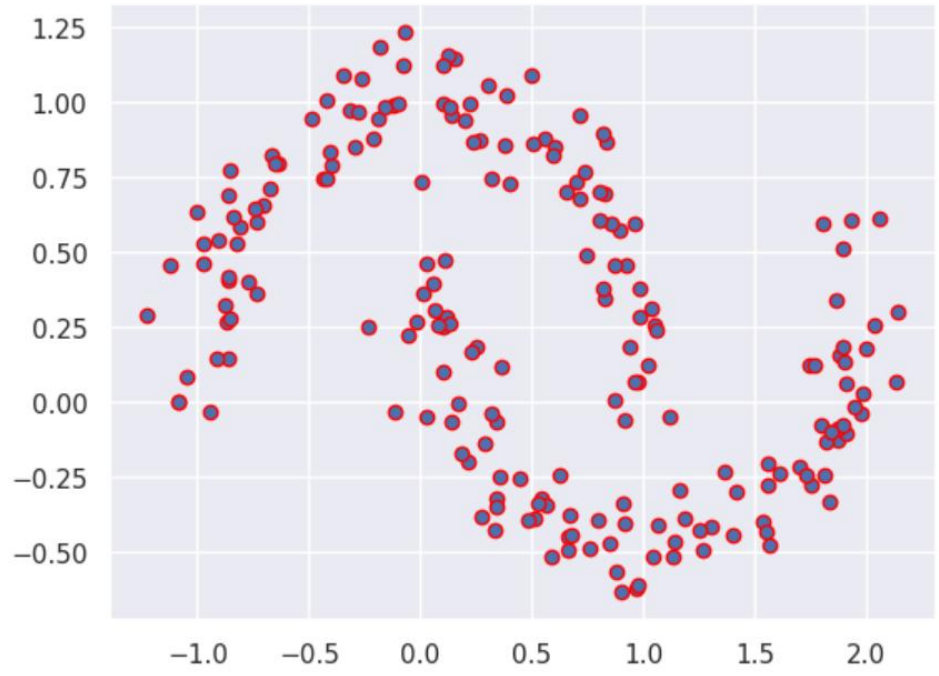
```
[[0.545 0.    0.133 0.322]
 [0.    1.    0.    0.   ]
 [1.    0.    0.    0.   ]
 [0.    1.    0.    0.   ]
 [0.196 0.    0.782 0.022]]
```

```
#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')
```

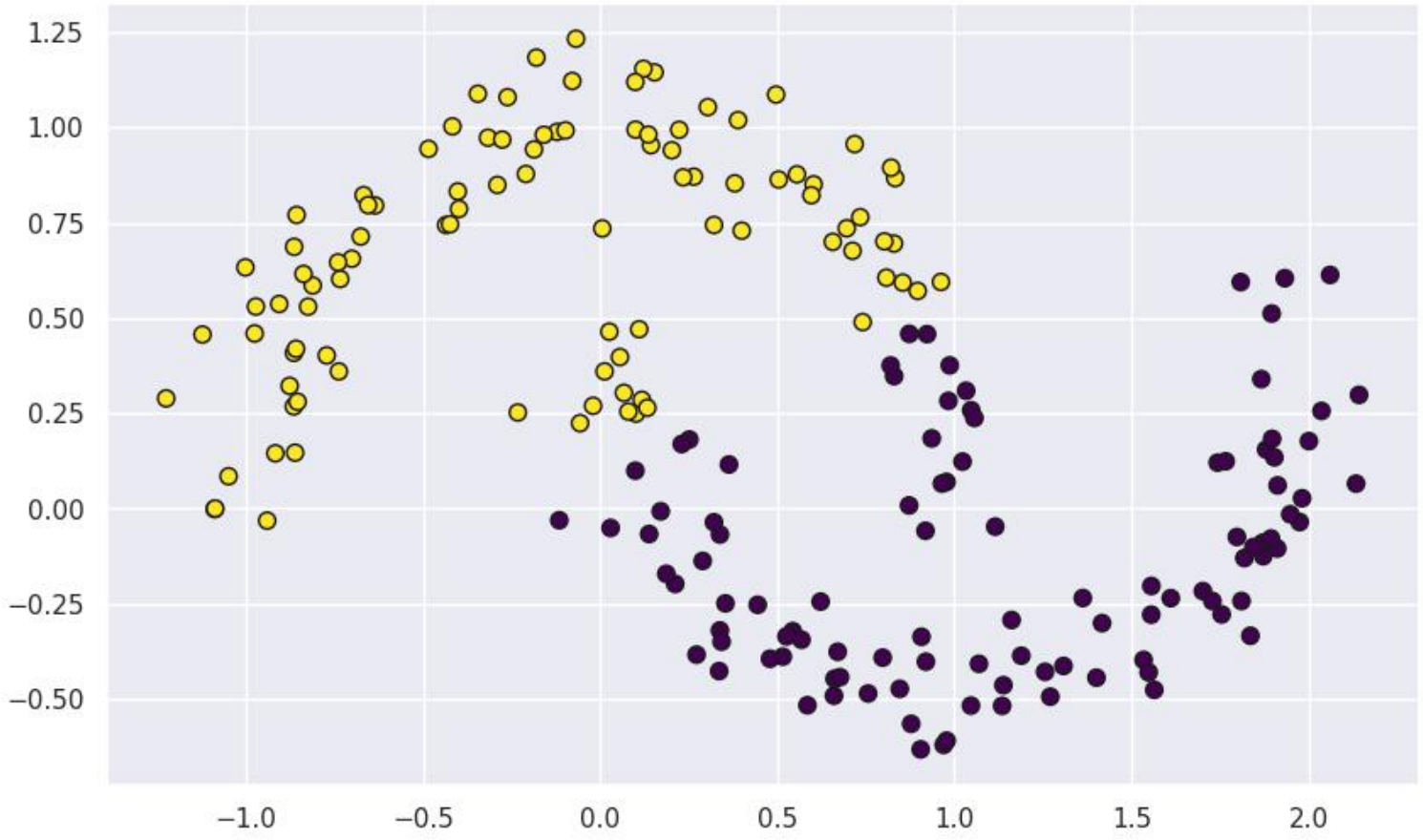
```
<matplotlib.collections.PathCollection at 0x7b28f242a020>
```




```
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)
```



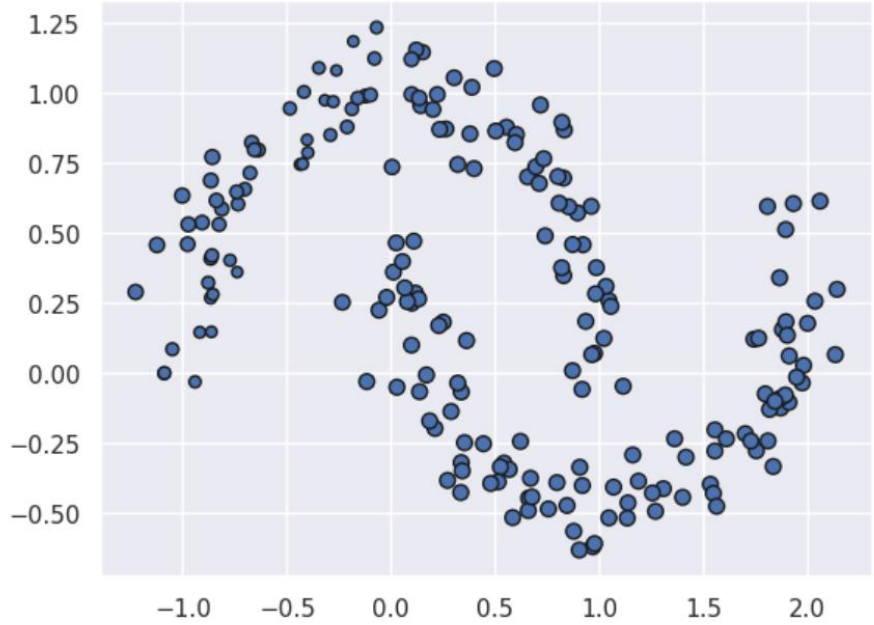

```
# Predict the probabilities of data points using Gaussian Mixture Models.
```

```
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()
```

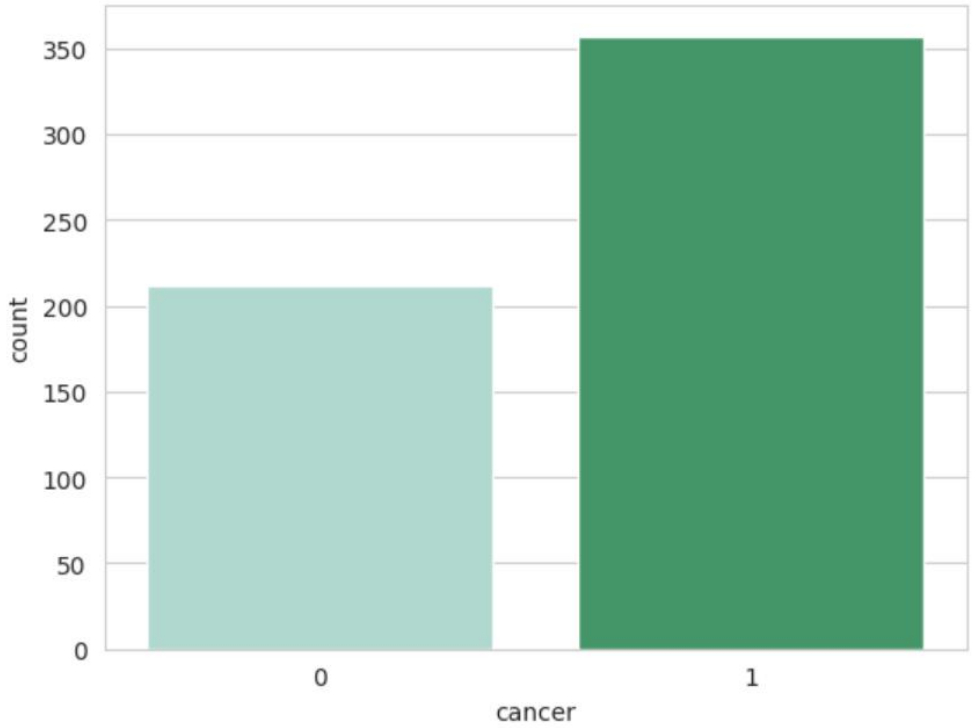
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	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

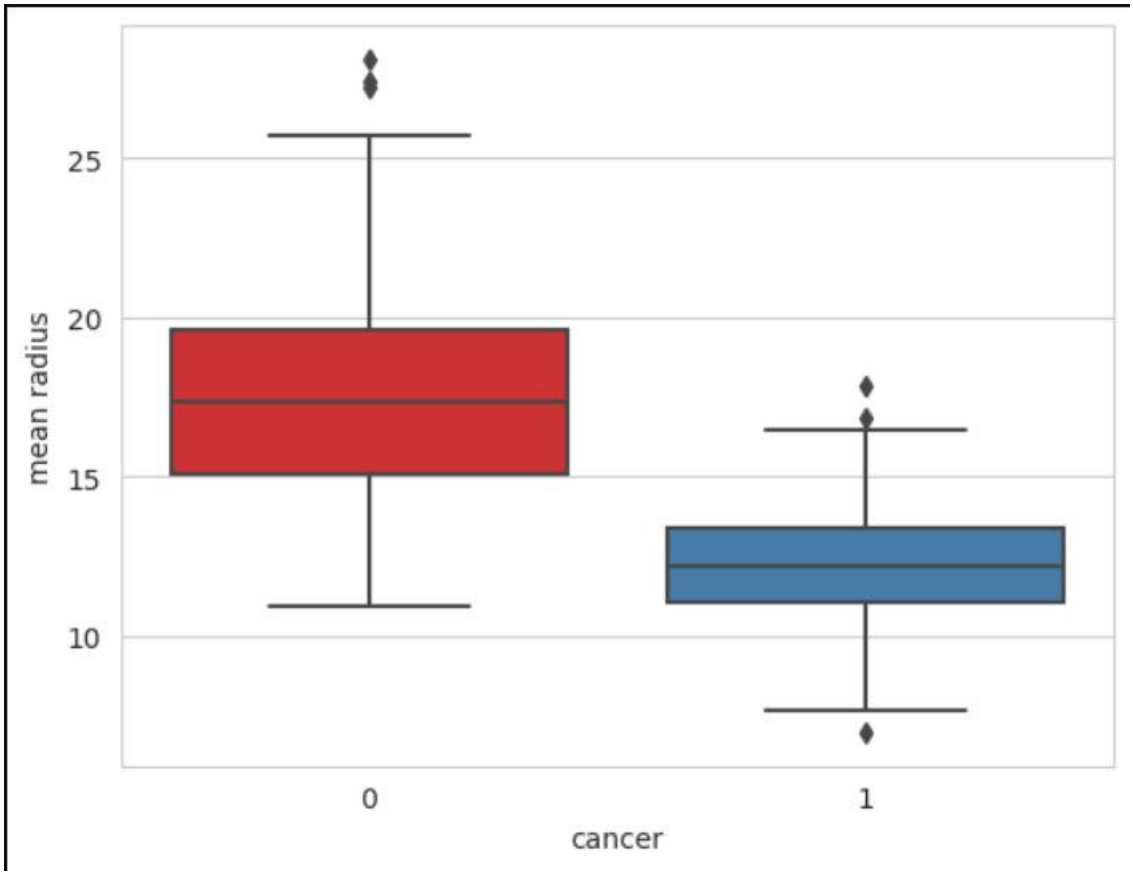
worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	cancer
17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
25.53	152.50	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

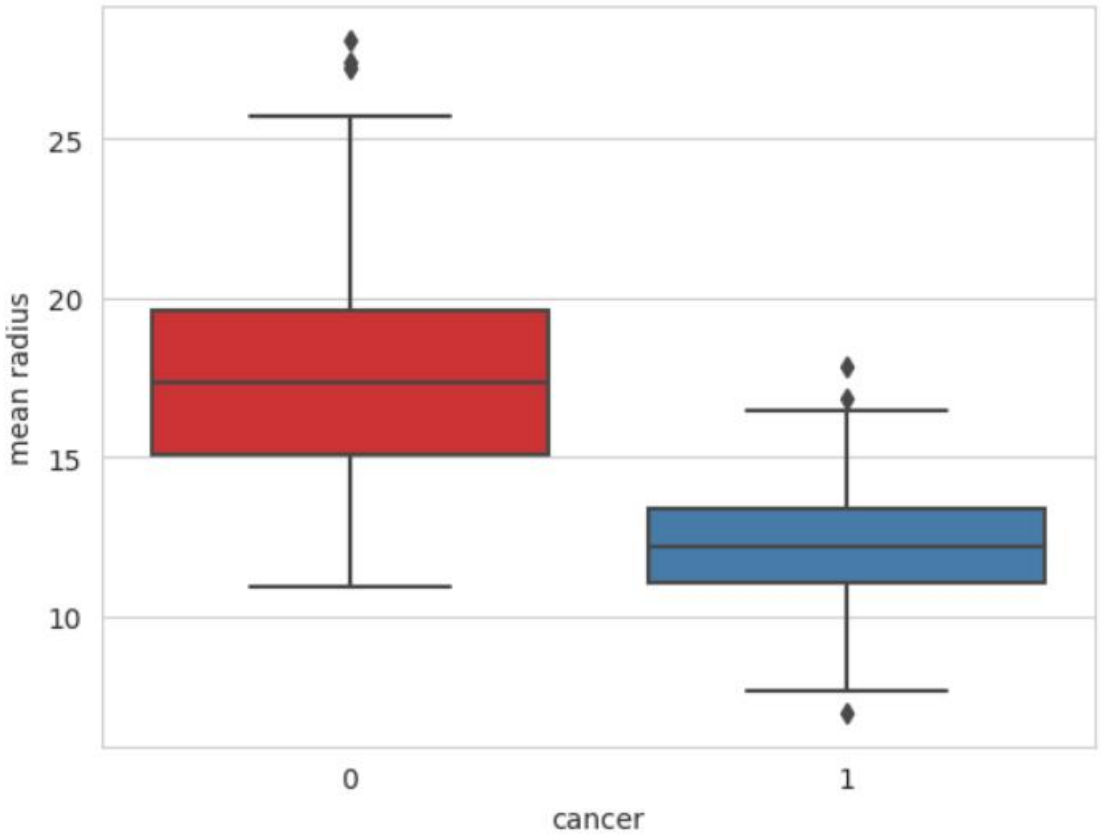
```
# Set the Seaborn style for plotting
sns.set_style('whitegrid')

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

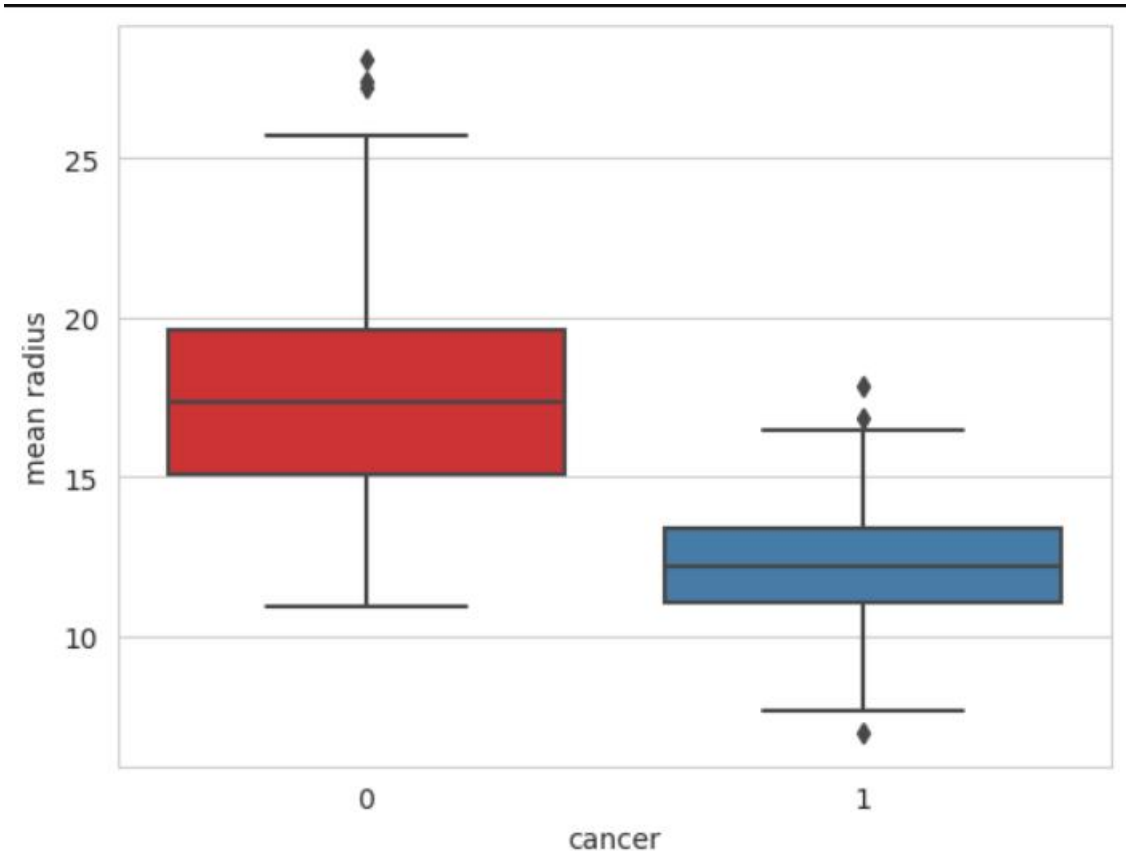
```
<Axes: xlabel='cancer', ylabel='count'>
```

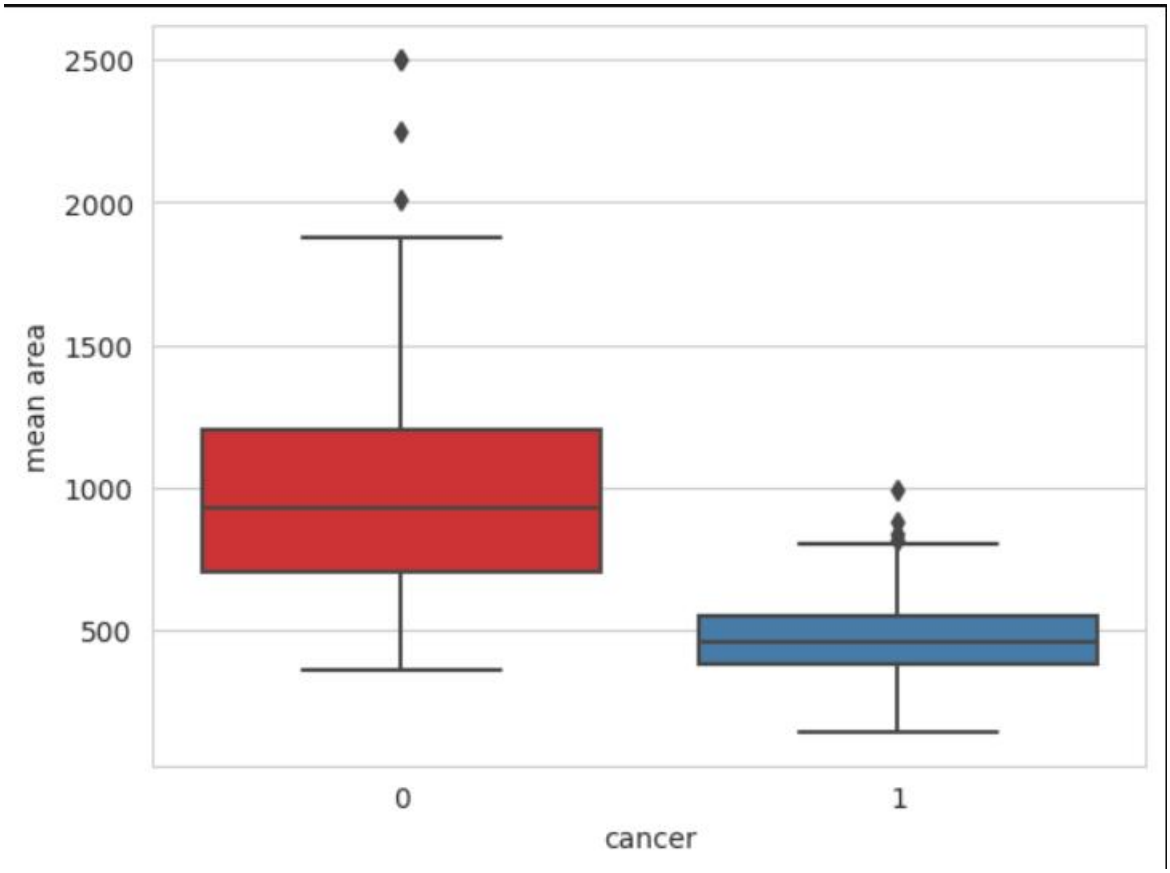


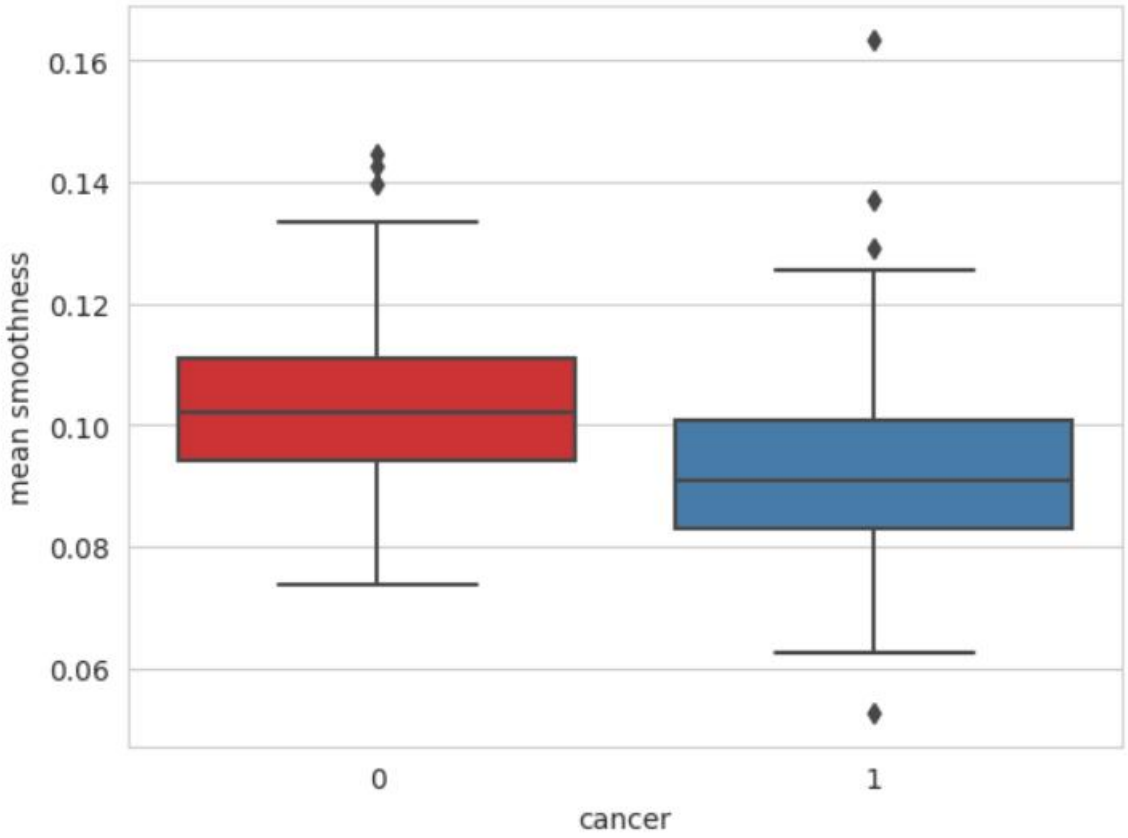

```
# Define a list of feature names for box plots
l = list(df.columns[0:10])


# 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()
```

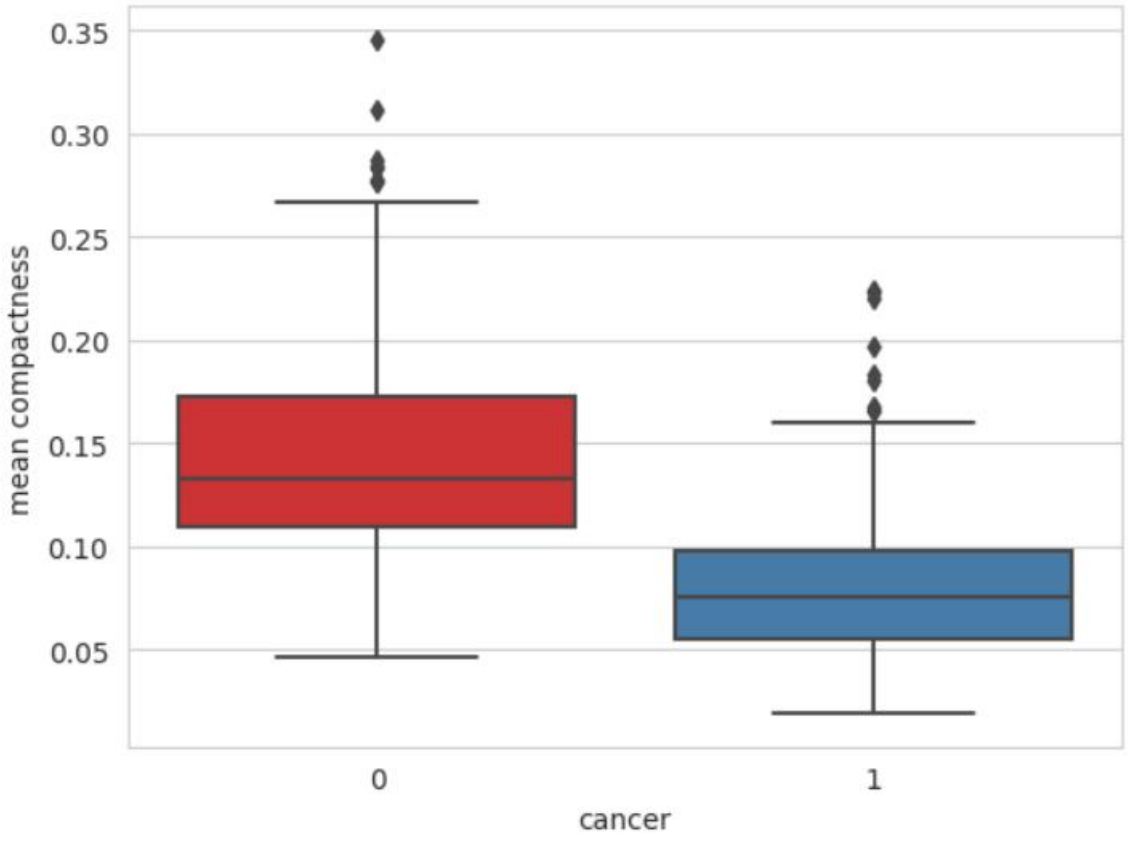

25
20
15
10
mean radius
0
1
cancer

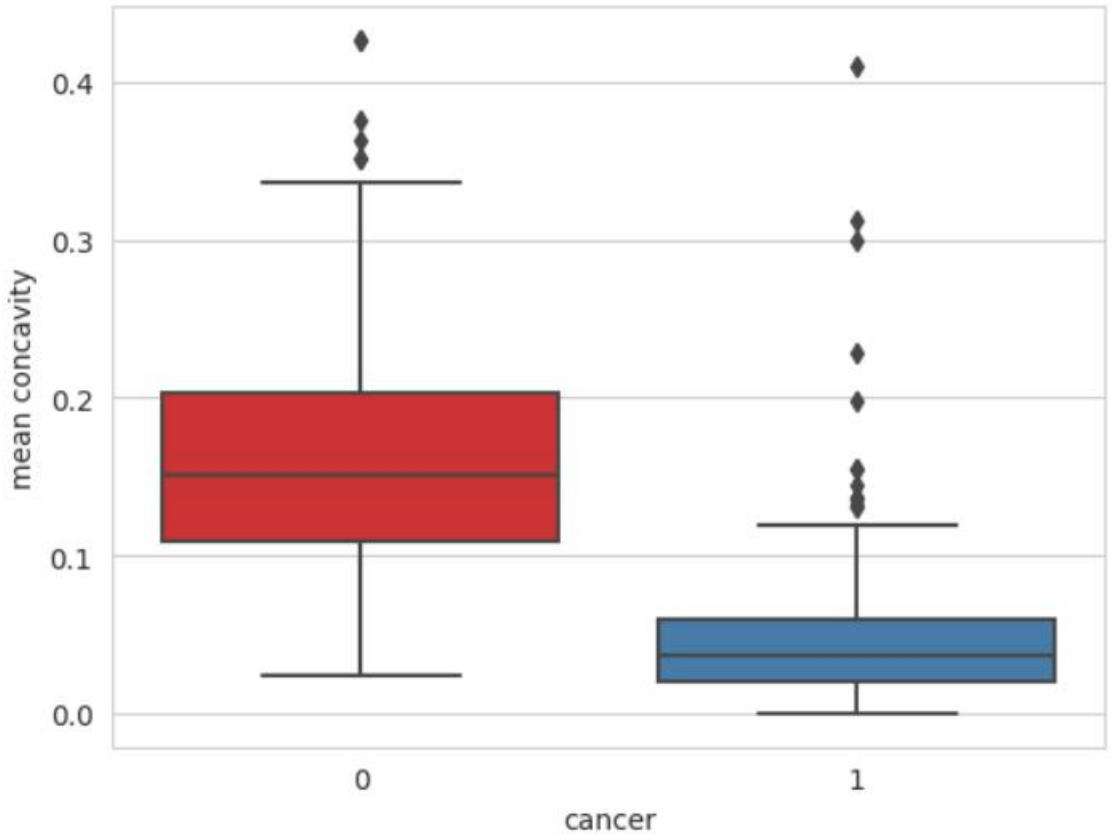

25
20
15
10
mean radius
0
1
cancer

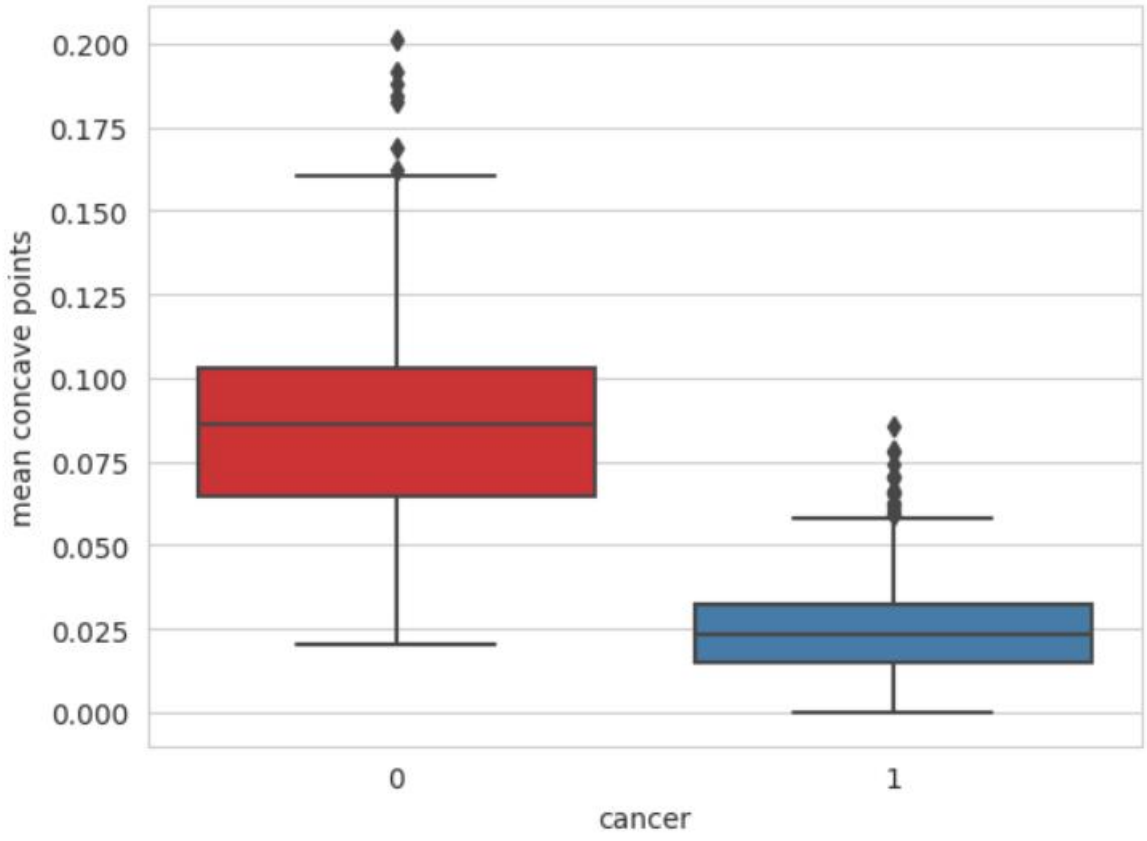

25
20
15
10
mean radius
0
1
cancer

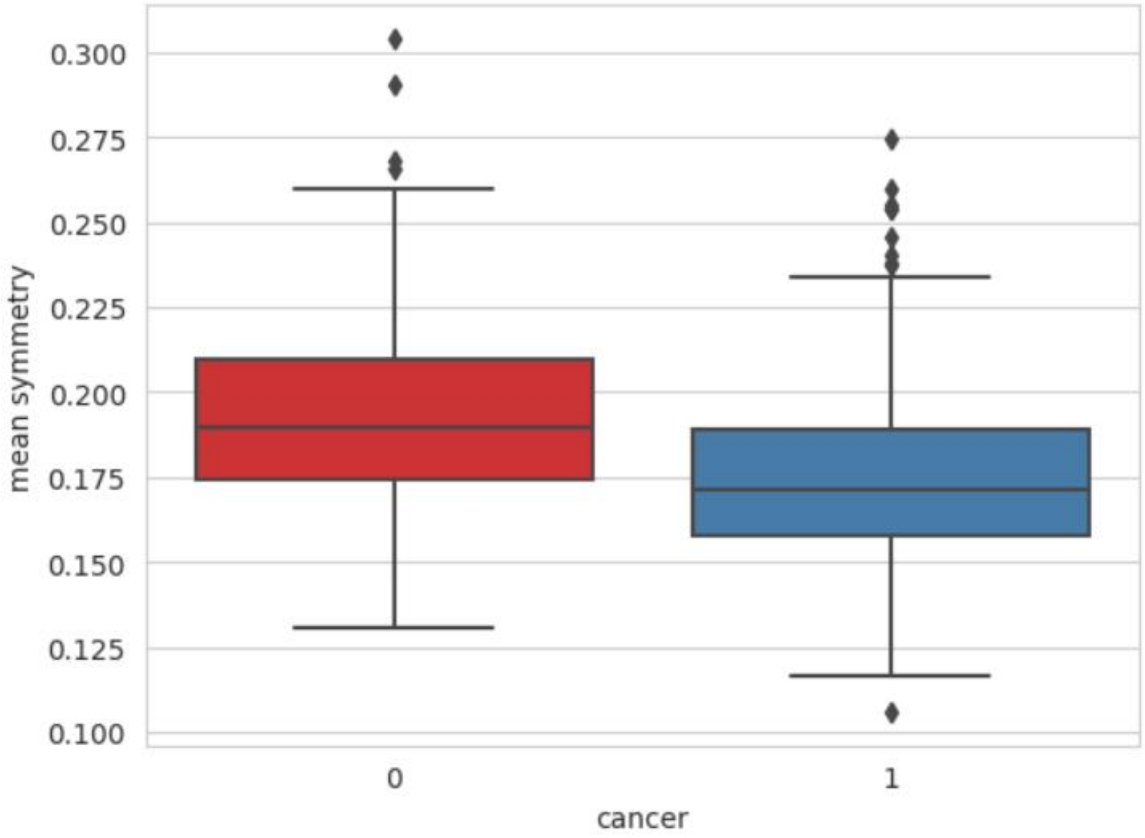

2500
2000
1500
1000
500
mean area
0
1
cancer


0.16
0.14
0.12
0.10
0.08
0.06
mean smoothness
0
1
cancer

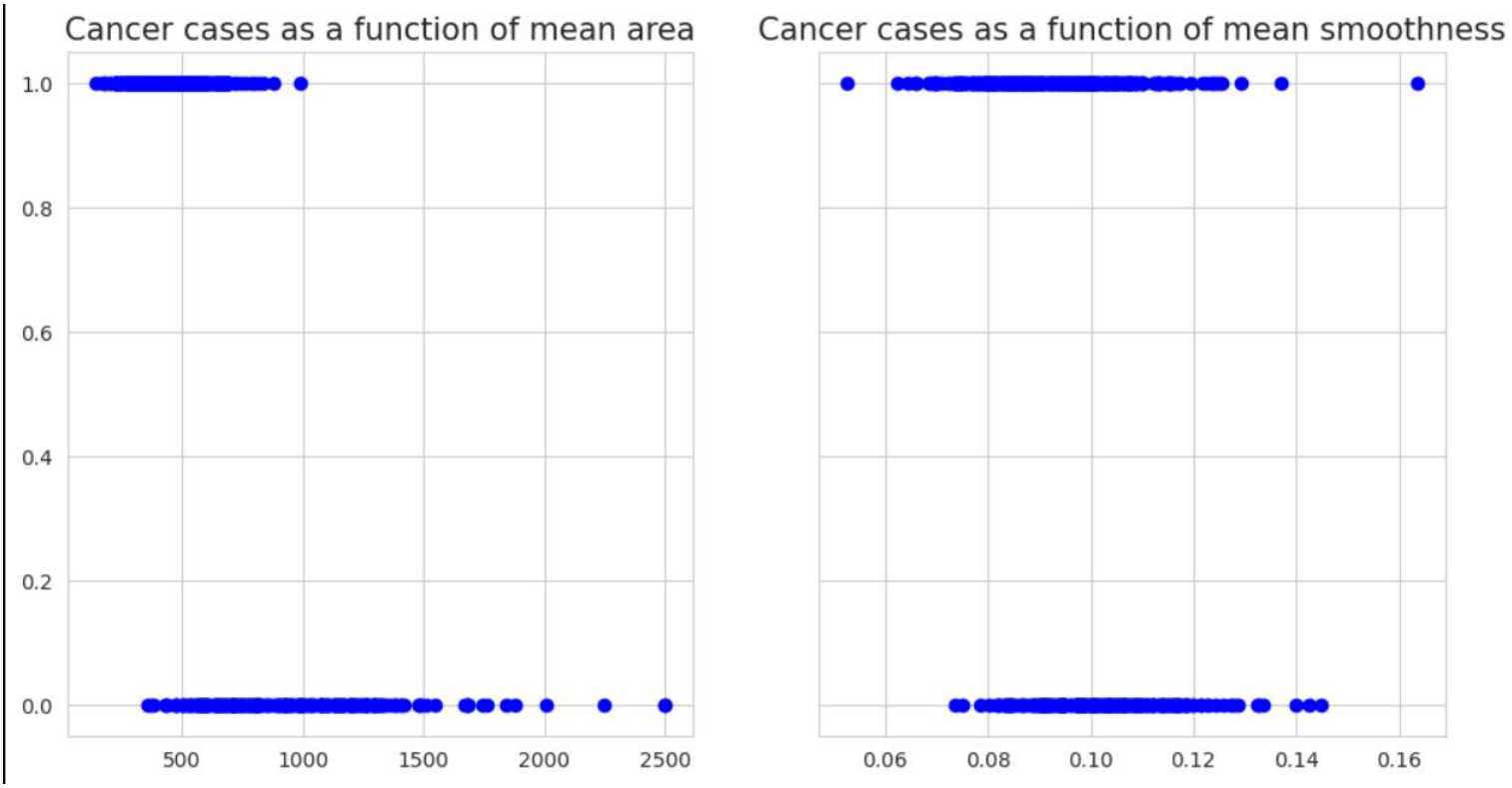

0.35
0.30
0.25
0.20
0.15
0.10
0.05
mean compactness
0
1
cancer


0.4
0.3
0.2
0.1
0.0
mean concavity
0
1
cancer


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




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

# Scatter plot of 'mean area' against 'cancer' in the first subplot
ax1.scatter(df['mean area'], df['cancer'], 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))
```

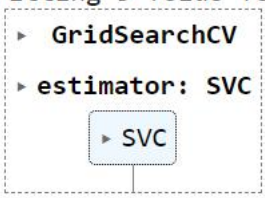
```
[[ 56  10]
 [  3 102]]
```

```
# 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)
```

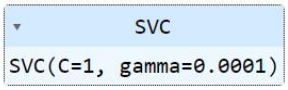
```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```




```
# 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_
```




```
# 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
```

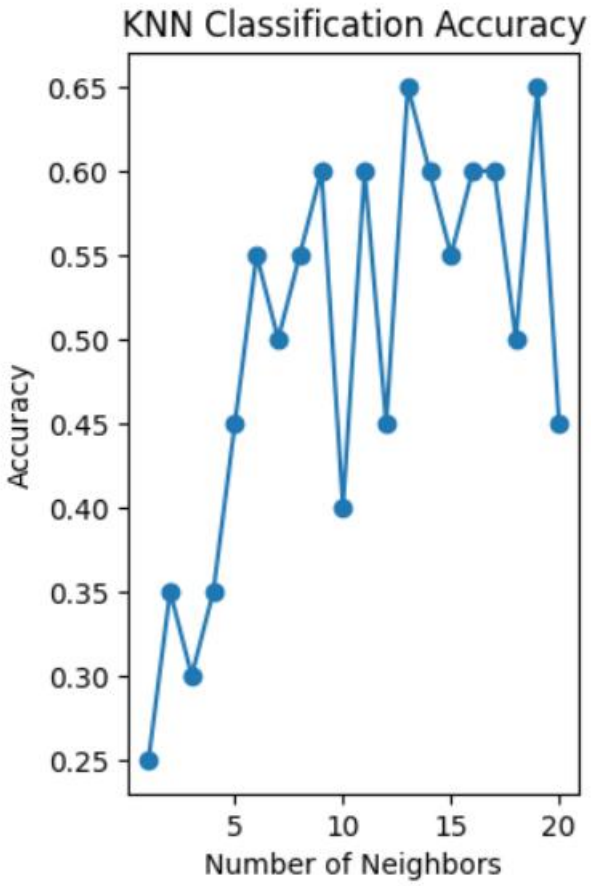
```
# Split the data into training and testing sets
X_train, X_test, y_train_class, y_test_class, y_train_reg, y_test_reg = train_test_split(
    X, y_classification, y_regression, test_size=0.2, random_state=42
```

```
)

# 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>
```

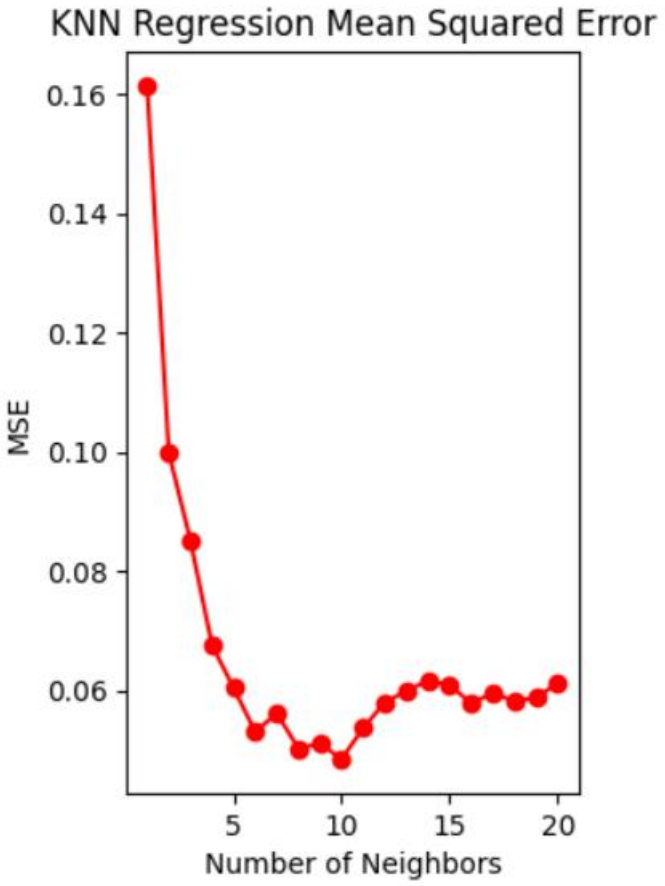
```
plt.subplot(1, 2, 1)
plt.plot(neighbors, accuracy_scores, marker='o')
plt.title('KNN Classification Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
```

```
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>
```