CRICKET BATSMAN STRIKE RATE 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

M.SRINITHA 2203A52201

Under the guidance of

Dr. KIRAN ERANKI

Associate Professor CSE.

Department of Computer Science and Artificial Intelligence

SR UNIVERSITY

Department of Computer Science and Artificial Intelligence

CERTIFICATE

This is to certify that project entitled **"CRICKET BASTMAN STRIKE RATE PREDICTION**" is the bona fide work carried out by **M.SRINITHA** as a Course Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING** during the academic year 2022-2023 under our guidance and Supervision.

Dr. Kiran Eranki
Associate. Prof. (CSE)
S R University,
Ananthasagar ,Warangal

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

ACKNOWLEDGEMENT

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

I 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, I express my thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

ABSTRACT

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

TABLE OF CONTENTS

INTRODUCTION

1.1 OVERVIEW

Cricket is being played in many countries all around the world. There are lots of domestic and international tournaments being held in many countries which play cricket. Cricket is a game played between two team comprising of 11 players in each team. The result is either a win, loss or tie. However, sometime due to bad weather conditions the game is also washed out as cricket is a game which cannot be played in a rain. Moreover, the game is also extremely unpredictable because at every stage of the game the momentum shifts to one of the team between the two. A lot of times the results gets decided on the last ball of the match where the game gets really close. Considering all these unpredictable scenarios of this unpredictable game, there is a huge interest among the spectators to do some predictions either at the start of the game or during the game.

1.2. PROBLEM STATEMENT

In this to design a system that can be provide the score and winning prediction in cricket match, the system can analyze multiple parameters like winning toss, batting side, DL approach, home ground advantage, player wise performance etc. while declaring a time for particular championship it is very important to select the best team so that the chances of the team winning the championship become easy. This problem had to be solved to generate the best players from both the team for the best battle. To solve this Problem i have collected historical data of all some team like (India, Australia, New Zealand, South Africa etc.), and using prediction algorithm like Naïve Bayes algorithm. I was predicting the best starting players for both the team can be used in fantasy league for winning the maximum points.

1.3. EXISTING SYSTEM

Cricket winning can be predicted like all other games. We need to find the best attributes or factors that influences the match outcome. The result of a cricket match depends upon more of in-game and more of pre-game attributes. Pre-game attributes like pitch, Team strength, weather, venue etc. and in-game attributes like run rate, total run, strike rate, wickets in hand etc. influences a match result predominantly. Below are the attributes that decides outcome of the cricket match.

1.4. PROPOSED SYSTEM

There has been a lot of related study to this problem in various different sports. The paper I have used for references are all related work that had been done on this problem. The paper by Trawinski described the prediction of results using a fuzzy classification system. This paper was predicting the results for basketball games. I had used the attribute selection technique mentioned in this paper for my project. The attribute selection technique proposed in this paper was done using WEKA so it was good reference point for me too. The wrapper method algorithm and the ranker method algorithm implemented in this paper was also used in my project.

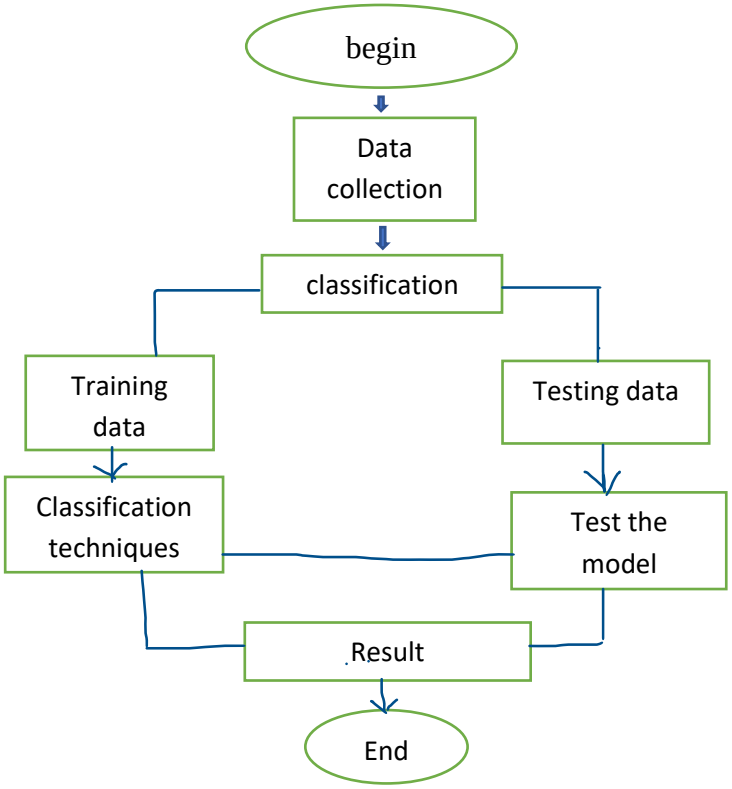


Figure 1.4: Proposed System

1.5. OBJECTIVES

Cricket is played by two teams of 11, with one side taking a turn to bat a ball and score runs, while the other team will bowl and field the ball to restrict the opposition from scoring. The main objective in cricket is to score as many runs as possible against the opponent. Before the match begins, the captain of both teams will toss a coin, with the winner of the toss being able to decide which team bats and fields first. Each cricket match consists of periods known as innings, and the number of innings that each team has will be determined before the match, usually one or two.

During an inning, one team bats the ball while the other attempts to field. Both teams take turns alternating between batting and fielding.

1.6.ARCHITECTURE

The architecture of the proposed system is as displayed in the figure below. The major components of the architecture are as follows:, raw data, feature extraction, training data, testing data, training data result, predicted match outcome.

Figure 1.6:Architecture

2.LITERATURE SURVEY

2.1.1. Document survey

An extensive online search produced very few articles related to players' performance prediction in the game of cricket. A very small number of researchers have studied the performance of cricket players. Muthuswamy and Lam.

The reference [1] predicted the performance of Indian bowlers against seven international teams against which the Indian cricket team plays most frequently. They used back International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.8, No.2, March 2018 20 propagation

network and radial basis network function to predict how many runs a bowler is likely to concede and how many wickets a bowler is likely to take in a given ODI match.[1]

The reference [2] Wickramasinghe predicted the performance of batsmen in a test series using a hierarchical linear model. [2]

The reference [3] Barr and Kantor defined a criterion for comparing and selecting batsmen in limited overs cricket. They defined a new measure P(out) i.e. probability of getting out and used a two dimensional graphical representation with Strike Rate on one axis and P(out) on another. Then they define a selection criterion based on P(out), strike rate and batting average of the batsmen.[3]

The reference [4] Iyer and Sharda used neural networks to predict the performance of players where they classify batsmen and bowlers separately in three categories – performer, moderate and failure. Based on the number of times a player has received different ratings, they recommend if the player should be included in the team to play World Cup 2007. [4]

The reference [5] Jhanwar and Paudi predict the outcome of a cricket match by comparing the strengths of the two teams. For this, they measured the performances of individual players of each team.[5]

3.DATA PRE-PROCESSING

3.1.1 DATASET DESCRIPTION

Sno	Attributes	Description
1.	Team	Name of the team
2.	Player	In which team the player is playing
3.	Filter	On which team the player have scored the runs and played matches
4.	Matches	How many matches the player is played
5.	Runs	Runs that the player have scored
6.	Highest	Among all the matches played from that bastman and scored highest runs
7.	Average	All the matches played by the batsman can take average
8.	Strike rate	The number of the batter divided by the number of balls faced
9.	Output	Target variable

Figure 3.1.1: Dataset Description

3.2 DATA CLEANING

The data obtained from the many websites was already cleaned. So, I did not have to do any sort of cleaning on the data. I had to tackle the missing values data and the data for the matches which were washed out due to rain. Those matches data were present in the .json file. after combining the two data into a single data set, I had to manually fill in the data about the 7 missing teams from the https://espncricinfo.com/ website. Moreover there were 10 matches which were washed out so those instance were filled with null values and they were discarded from the data set. So, the final dataset had 5219 instances with 21 attributes.

3.3 DATA VISUALISATION

An important factor of data visualization project is to visualize the data in a visually appealing format, for the users to draw insights from the data. Graphically represented data are easy to be interpreted. The graphs contain all the details of the features that were extracted from the huge datasets. I have done this using am Charts, A Python library for data visualization. I have further used different packages to get the exact analysis and visualization for teams and their players.

The processing of these data for each individual happens in the backend program which is written with the help of flask. Pandas, an open source for data analysis is used as the data processing tool and to provide input output for both csv files.

3.4 DATASET

Team	Player	Filter	Matches	Runs	Highest	Average	Strike Rate
India	Rohit Shar	vs Pakistar	8	70	30	14	127.27
India	Rohit Shar	vs Afghanis	2	75	74	75	156.25
India	Rohit Shar	vs Sri Lank	18	339	118	21.18	138.93
India	Rohit Shar	vs Banglad	11	452	89	41.09	144.4
India	Rohit Shar	year 2022	13	290	64	24.16	140.09
India	KL Rahul	vs Pakistar	1	3	3	3	37.5
India	KL Rahul	vs Afghanis	1	69	69	69	143.75
India	KL Rahul	vs Sri Lank	8	295	89	42.14	141.14
India	KL Rahul	vs Banglad	5	99	52	33	128.57
India	Virat Kohli	vs Pakistar	7	311	78	77.75	118.25
India	Virat Kohli	vs Afghanis	2	50	50	50	128.2
India	Virat Kohli	vs Sri Lank	7	339	82	84.75	140.6
India	Virat Kohli	vs Banglad	4	129	57	64.5	113.15
India	Virat Kohli	year 2022	4	81	52	20.25	128.57
India	Suryakuma	vs Pakistar	1	11	11	11	137.5
India	Suryakuma	vs Sri Lank	1	50	50	50	147.05
India	Suryakuma	year 2022	12	428	117	38.9	189.38
India	Rishabh Pa	vs Pakistar	1	39	39	39	130
India	Rishabh Pa	vs Afghanis	1	27	27		207.6
India	Rishabh Pa	vs Sri Lank	3	24	23	24	100
India	Rishabh Pa	vs Banglad	4	40	27	13.33	93.02
India	Rishabh Pa	year 2022	13	260	52	26	135.51
India	Deepak Ho	vs Sri Lank	3	21	21	21	131.25
India	Deepak Ho	year 2022	9	274	104	54.8	161.17
India	Dinesh Kar	vs Pakistar	1	11	11	11	122.2
India	Dinesh Kar	vs Sri Lank	7	92	39	92	158.6
India	Dinesh Kar	vs Banglad	3	33	29		275
India	Hardik Pan	vs Pakistar	3	11	11	5.5	110
India	Hardik Pan	vs Afghanis	1	35	35		269.23
India	Hardik Pan	vs Sri Lank	8	55	27	9.16	134.14

Figure 3.4: DATASET

4. METHODOLOGY

4.1 PROCEDURE TO SOLVE THE GIVEN PROBLEM

In this project of strike rate analysis and prediction i use three approaches:

1.Ridge regression

2.Lasso regression

3.Linear regression

1.Ridge Regression

Ridge regression is a model tuning method that is used to analyze any data that suffers from multi collinearity. This method performs L2 regularization. When the issue of multi collinearity occurs,

least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values. Ridge regression is a technique which is used for analyzing multiple regression where the data suffers from multi collinearity.

The problem which arises due to multi collinearity is that the basic linear regression model (least square estimates) becomes unbiased and the variance becomes so large that the predicted values are far from the true value. The advantage of using the ridge regression is to avoid over fitting. It works in the same way as the linear regression but it just adds an extra term (α) which helps in the reduction of overfitting. The goal of any machine learning model is to generalize the pattern which it needs to be predicted, i.e. The model should work best on both training as well as test data. Over fitting occurs when the trained model performs well on the training data and performs poorly on the testing dataset. Fig 4.2: Ridge Regression Formula and Examples of Ridge Regression: The ridge regression formula is of the form: Examples where ridge regression may be used

- Ridge regression can be used for the analysis of prostate-specific antigen.
- It method used for the analysis of multi collinearity in multiple regression data

2.Lasso Regression

Lasso regression is a type of linear regression that uses shrinkage. The word "LASSO" stands for Least Absolute Shrinkage and Selection Operator. It is a statistical formula for the regularization of data models and feature selection. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of muti collinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination. Fig 4.3: Lasso Regression Formula and Examples of LASSO Regression: The formula which is used to calculate Lasso Regression is Where λ is the lasso penalty factor. Examples where multiple linear regression may be used

- For analyzing the prostate-specific antigen and the clinical measures among the patients who were about to have their prostates removed
- For predicting the effect on sales and spending a certain amount of money on advertising

3. Linear regression

Linear regression also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable.Thegoalofmultiplelinearregressionistomodelthelinearrelationshipbetweenthe explanatory (independent) variables and response (dependent) variables. In essence, multiple regression is the extension of ordinary least-squares (OLS) regression because it involves more than one explanatory variable.

A multiple linear regression analysis is carried out to predict the values of a dependent variable, *Y*, given a set of p explanatory variables (x1, x2, …., x*p)*. In these notes, the necessary theory for multiple linear regression is presented and examples of regression analysis with census data are given to illustrate this theory. This course on multiple linear regression analysis is therefore intended to give a practical outline to the technique. Complicated or tedious algebra will be avoided where possible, and references will be given to more theoretical texts on this technique. Important issues that arise when carrying out a multiple linear regression analysis are discussed in detail including model building, the underlying assumptions, and interpretation of results. However, before we consider multiple linear regression analysis we begin with a brief review of simple linear regression.

Formula and Calculation of Linear Regression

$$yi=\beta 0+\beta 1xi1+\beta 2xi2+...+\beta xi+\epsilon$$

where , for $i=n$ observations:
yi=dependent variable
xi=explanatory variables
$\beta 0$=y-intercept (constant term)
βp=slope coefficients for each explanatory variable
ϵ=the model's error term (also known as the residuals)
Examples where multiple linear regression may be used include:

- Tryingtopredictanindividual'sincomegivenseveralsocio-economiccharacteristics.
- Trying to predict the overall examination performance of pupils in 'A' levels, given the values of a set of exam scores at age 16.

- Trying to estimate systolic c systolic blood pressure, given a variety of socio-economic and behavioural characteristics (occupation, drinking smoking, age etc).

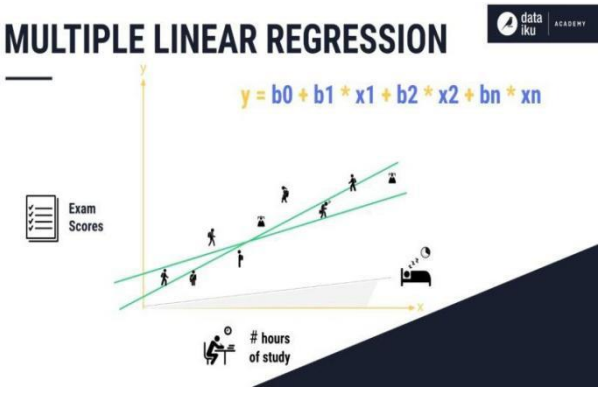


Figure: Multiple Linear Regression

4.2.OVERVIEW TECHNOLOGY

In my program i used python 3 programming language. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing anddynamic binding, make it very attractive for Rapid Application Development, as wellas for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reducesthe cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge forall major platforms, and can be freely distributed.

4.3.SOFTWARE DESCRIPTION

Software requirements:

Operating system: Windows

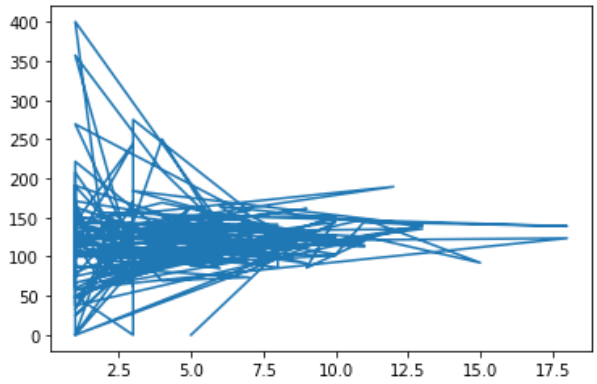
Platform: google co-lab

Programing language: python

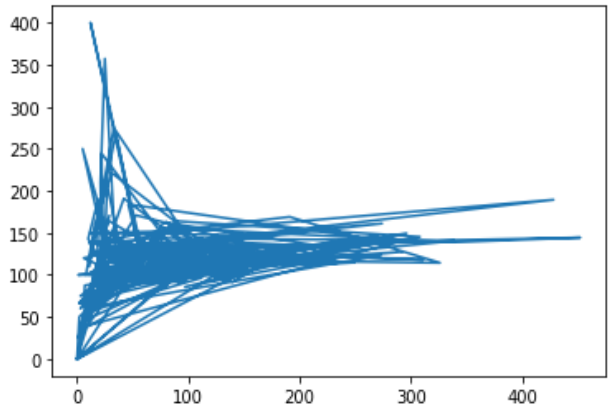
5. RESULTS

DESCRIPTION:

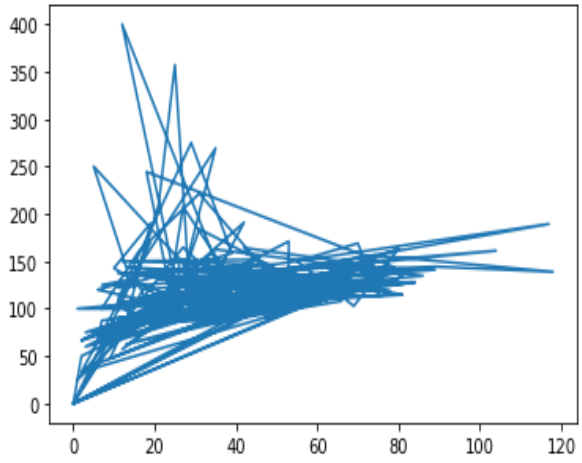
```
 import pandas as pd
 dd=pd.read_csv('all_batsmen_data.csv')
print(dd)
from matplotlib import pyplot as pp
x1=dd['Matches']
y=dd['Strike Rate']
a=pp.plot(x1,y)
print(a)
```



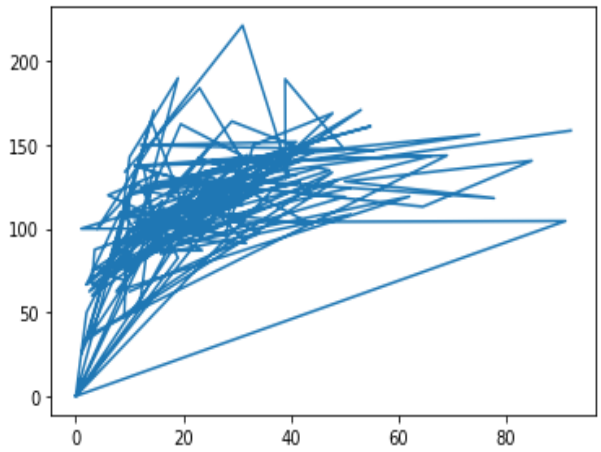

```
x2=dd['Runs']
y=dd['Strike Rate']
a=pp.plot(x2,y)
print(a)
```




```
x3=dd['Highest']

y=dd['Strike Rate']

a=pp.plot(x3,y)

print(a)
```




```
x4=dd['Average']

y=dd['Strike Rate']

a=pp.plot(x4,y)

print(a)
```



CODE:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn. model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import Lasso, Ridge, LinearRegression,
LinearRegression

# Load the dataset from a CSV file
df = pd.read_csv("all_batsmen_data.csv")
# Check for missing values and replace them with 0
print(df. isnull())
df = df.fillna(0)
# Extract the features (independent variables, x) and the target
(dependent variable, y)
y = df.iloc[:, 7:8]
x = df.iloc[:, 4:7]
print(x, '\n', y)
```

OUTPUT :


```
# Split the data into training and testing sets (you need two train-test split)
from sklearn. model_selection import train_test_split
X_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15,random_state=2)
X_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.10,random_state=99)
print(X_train.shape)
print(y_train.shape)
print(y_test.shape)
```

OUTPUT:

(180,3)

(180,1)

(20,3)

(20,1)

```
# Define and train a Lasso regression model
model_lasso=Lasso(alpha=0.01)
model_lasso.fit(x,y)
# Make predictions with the Lasso model on the test data
pred_train_lasso=model_lasso.predict(x_test)
# Calculate and print the mean squared error for Lasso
print(mean_squared_error(y_test,pred_train_lasso))
```

OUTPUT:

```
514.2973692193113
```

```
# Define and train a Ridge regression model
rr=Ridge(alpha=0.01)
```

```
rr.fit(X_train,y_train)
# Make predictions with the Ridge model on the training data
pred_train_rr=rr.predict(X_train)
# Calculate and print the root mean squared error and R-squared for
Ridge
print(np.sqrt(mean_squared_error(y_train,pred_train_rr)))
print(r2_score(y_train,pred_train_rr))
```

OUTPUT:

48.08997951012002
0.09014663197709338

```
from sklearn.linear_model import LinearRegression
# Define and train a Linear Regression model
model=LinearRegression()
model.fit(x,y)
model=LinearRegression().fit(x,y)
# Calculate and print the coefficient of determination (R-squared) for
Linear Regression
r_sq = model.score(x,y)
print("coefficient of determination:",r_sq)
# Print the intercept and coefficients for Linear Regression
print("intercept: ",model.intercept_)
print("coefficients:",model.coef_)
print(X_train)
print(y_train)
```

OUTPUT:

```
# Train a Linear Regression model on the training data
reg_all=linear_model.LinearRegression()
reg_all.fit(X_train,y_train)
# Make predictions with the Linear Regression model on the test data
y_pred=reg_all.predict(x_test)
#Rsquare=reg_all.score(y_pred,y_test)
#print("Rsquare: %f" %(Rsquare))
#print(y_test)
# Calculate and print metrics (mean squared error, mean absolute error,
and root mean squared error) for Linear Regression
print("Intercept:",reg_all.intercept_)
mse=mean_squared_error(y_test,y_pred)
#print("mse: %f" %(mse))
print(mse)
mae=mean_absolute_error(y_test,y_pred)
#print("mae: %f" %(mae))
print(mae)
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
#print("rmse: %f" %(rmse))
print(rmse)
```

OUTPUT :

```
Intercept: [99.94467493]
549.7744911257489
16.36867644792618
23.44727044083701
```

5. CONCLUSION AND FUTURE SCOPE

Selection of the right players for each match plays a significant role in a team's victory. An accurate prediction of how many runs a batsman is likely to score and how many wickets a bowler is likely to take in a match will help the team management select best players for each match. In this paper, we modeled batting and bowling datasets based on players' stats and characteristics. Some other features that affect players' performance such as weather or the nature of the wicket could not be included in this study due to unavailability of data. Four multiclass classification algorithms were used and compared. Random Forest turned out to be the most accurate classifier for both the datasets with an accuracy of 90.74% for predicting runs scored by a batsman and 92.25% for predicting wickets taken by a bowler. Results of SVM were surprising as it achieved an accuracy of just 51.45% for predicting runs and 68.78%

for predicting wickets. Similar studies can be carried out for other formats of the game i.e. test cricket and T20 matches.

6.REFERENCES

[1] A. L. Samuel, “Some studies in machine learning using the game of checkers. recent progress,” in Computer Games I, pp. 366–400, Springer, 1988.

[2] A. Bandula siri, “Predicting the winner in one day international cricket,” Journal of Mathematical Sciences & Mathematics Education, vol. 3, no. 1, pp. 6–17, 2008.

[3] Indian Premier League Official Website

[4] P. Langley, W. Iba, K. Thompson, et al., “An analysis of bayesian classifiers,” in Aaai, vol. 90, pp. 223–228, 1992.

[5] S. Kampakis and W. Thomas, “Using machine learning to predict the outcome of English county twenty over cricket matches,” arXiv preprint arXiv:1511.05837, 2015.

[6] L. Pass field and J. G. Hopper, “A mine of information: can sports analytics provide wisdom from your data?,” International journal of sports physiology and performance, vol. 12, no. 7, pp. 851–855, 2017.

[7] T. H. Davenport, “What businesses can learn from sports analytics,” MIT Sloan Man-agreement Review, vol. 55, no. 4, p. 10, 2014.

INDEX

Lab-1/2203A52201/sec-AB:

Lab01: Exposing to various frameworks of StatML – NumPy, Pandas, Matplotlib, Seaborn, Tensorflow, Keras.

CODES:

```
import numpy as np
import pandas as pd
lst=[1,2,3]
array1=np.array(lst)
array1
```

OUTPUT:

array([1, 2, 3])

```
print("A series of zeroes:",np.zeros(7))
print("A series of ones:",np.ones(9))
print("A series of numbers:",np.arange(5,16))
print("Numbers spaced apart by 2:",np.arange(0,11,2))
print("Numbers spaced apart by float:",np.arange(0,11,2.5))
print("Every 5th number from 30 in reverse order: ",np.arange(30,-1,-
5))
print("11 linearly spaced numbers between 1and5: ",np.linspace(1,5,11))
```

OUTPUT:

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

```
import pandas as pd
df=pd.read_csv("wine.csv")
df.head()
```

OUTPUT:

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

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

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

OUTPUT:

	name	locn	marks
0	a	gfs	5
1	b	gsdf	3

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

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

OUTPUT:

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

```
# Read tables from the specified Wikipedia page and store them in a
list of DataFrames
```

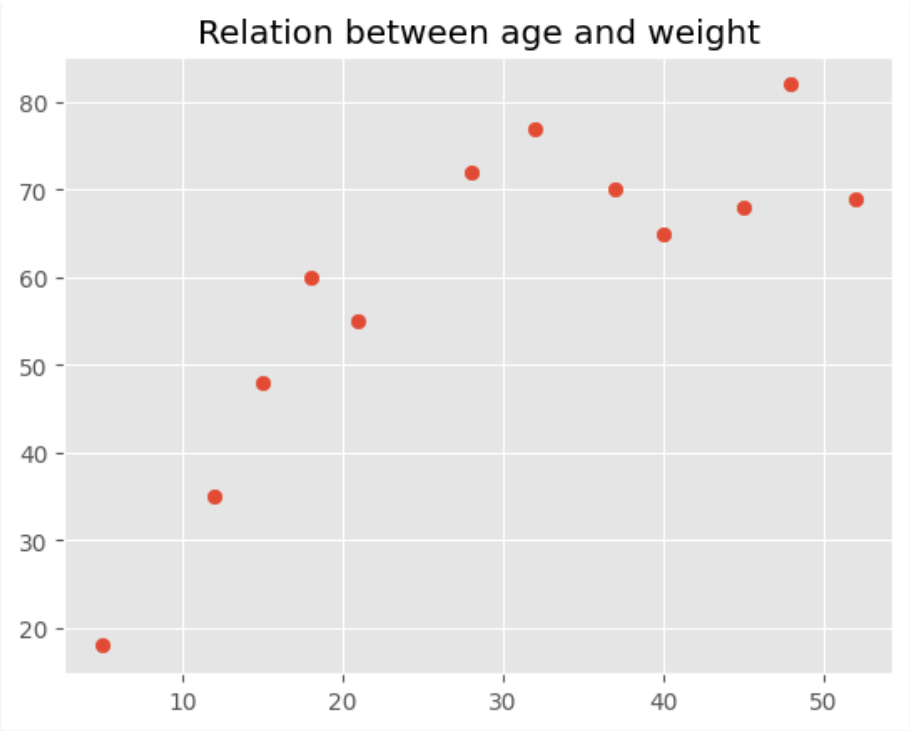
```
list_of_df=pd.read_html("https://en.wikipedia.org/wiki/2016_Summer_Olympics_medal_table",header=0)
# Assuming the desired table is the first one in the list, store it in a DataFrame 'medals'
medals=list_of_df[0]
# Display the DataFrame 'medals' to view the table
medals
```

OUTPUT:

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

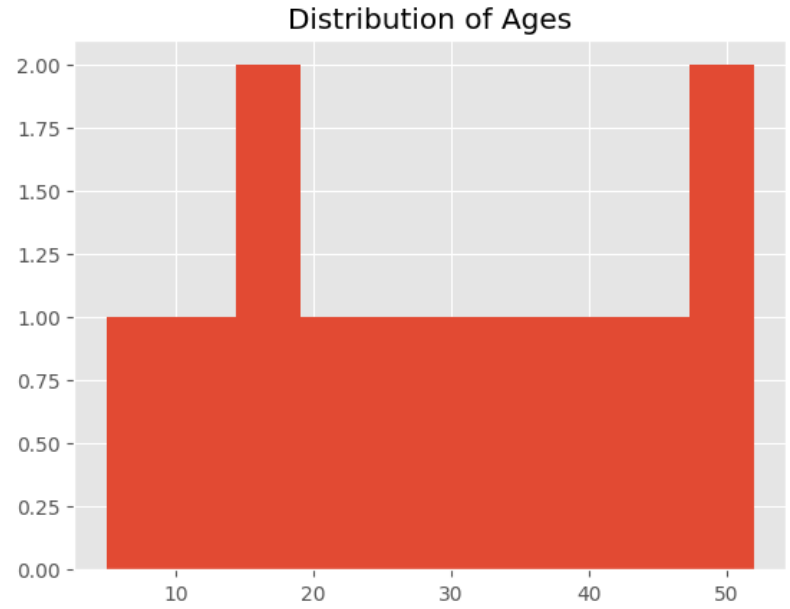
```
import matplotlib.pyplot as plt

# List of people's names
people =
['Ann','Brandon','Chen','David','Emily','Farook','Gagan','Hamish','Imran','Julio','Katherine','Lily']
# List of people's ages
age = [21,12,32,45,37,18,28,52,5,40,48,15]
# List of people's weights (in kilograms)
weight = [55,35,77,68,70,60,72,69,18,65,82,48]
# List of people's heights (in centimeters)
height = [160,135,170,165,173,168,175,159,105,171,155,158]


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

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

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

OUTPUT:




```
# Import the Matplotlib library for data visualization
import matplotlib.pyplot as plt
# Create a histogram of ages
plt.hist(age)
# Display the histogram
plt.show()
```

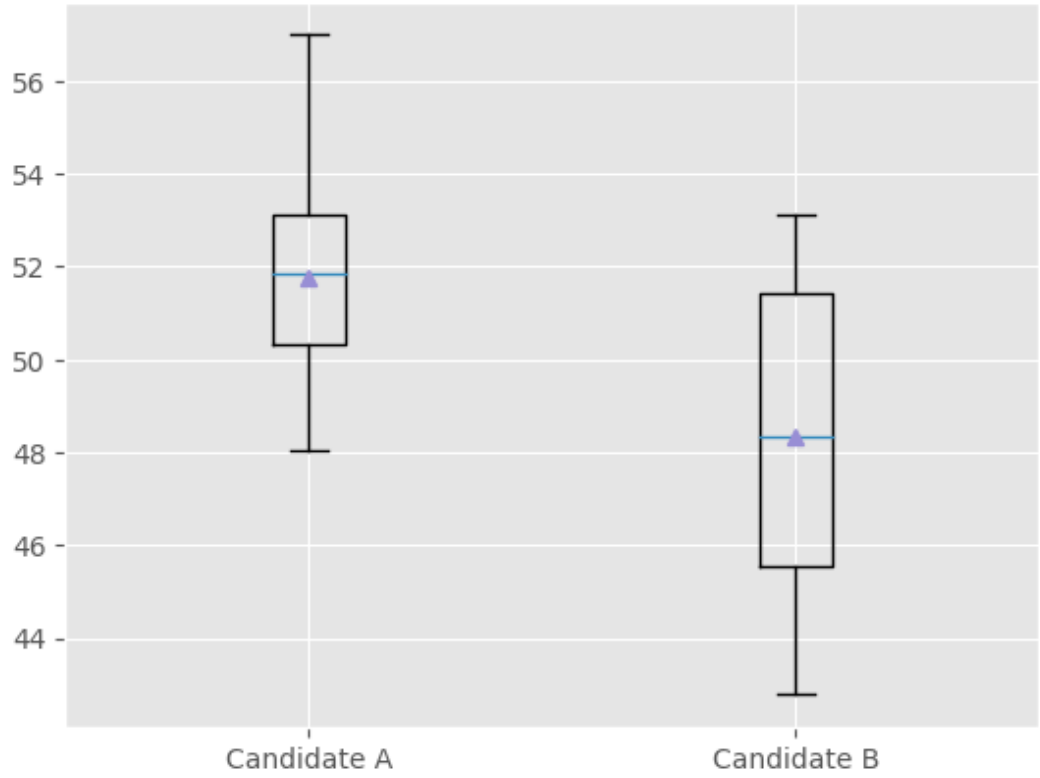
OUTPUT:




```
# Import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
# Generate an array of days from 1 to 30
```

```
days = np.arange(1, 31)
# Simulate data for Candidate A and Candidate B
candidate_A = 50 + days * 0.07 + 2 * np.random.randn(30)
candidate_B = 50 - days * 0.1 + 3 * np.random.randn(30)
# Set the plot style to 'ggplot'
plt.style.use('ggplot')
# Create a boxplot for Candidate A and Candidate B
# Show the means in the plot
plt.boxplot(x=[candidate_A, candidate_B], showmeans=True)
# Add a grid to the plot
plt.grid(True)
# Set the x-axis ticks and labels
plt.xticks([1, 2], ['Candidate A', 'Candidate B'])
# Display the plot
plt.show()
```

OUTPUT:



Lab – 2 /2203A52201/sec-AB:

Lab02: Reading data and identifying variables, finding maximum likelihood estimation, density estimation

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Required for 3D graph of
random variables
from scipy.stats import multivariate_normal  # Gaussian random variable
%matplotlib inline  # Enable inline plotting in Jupyter Notebook or
Jupyter Lab


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

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

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

OUTPUT:

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

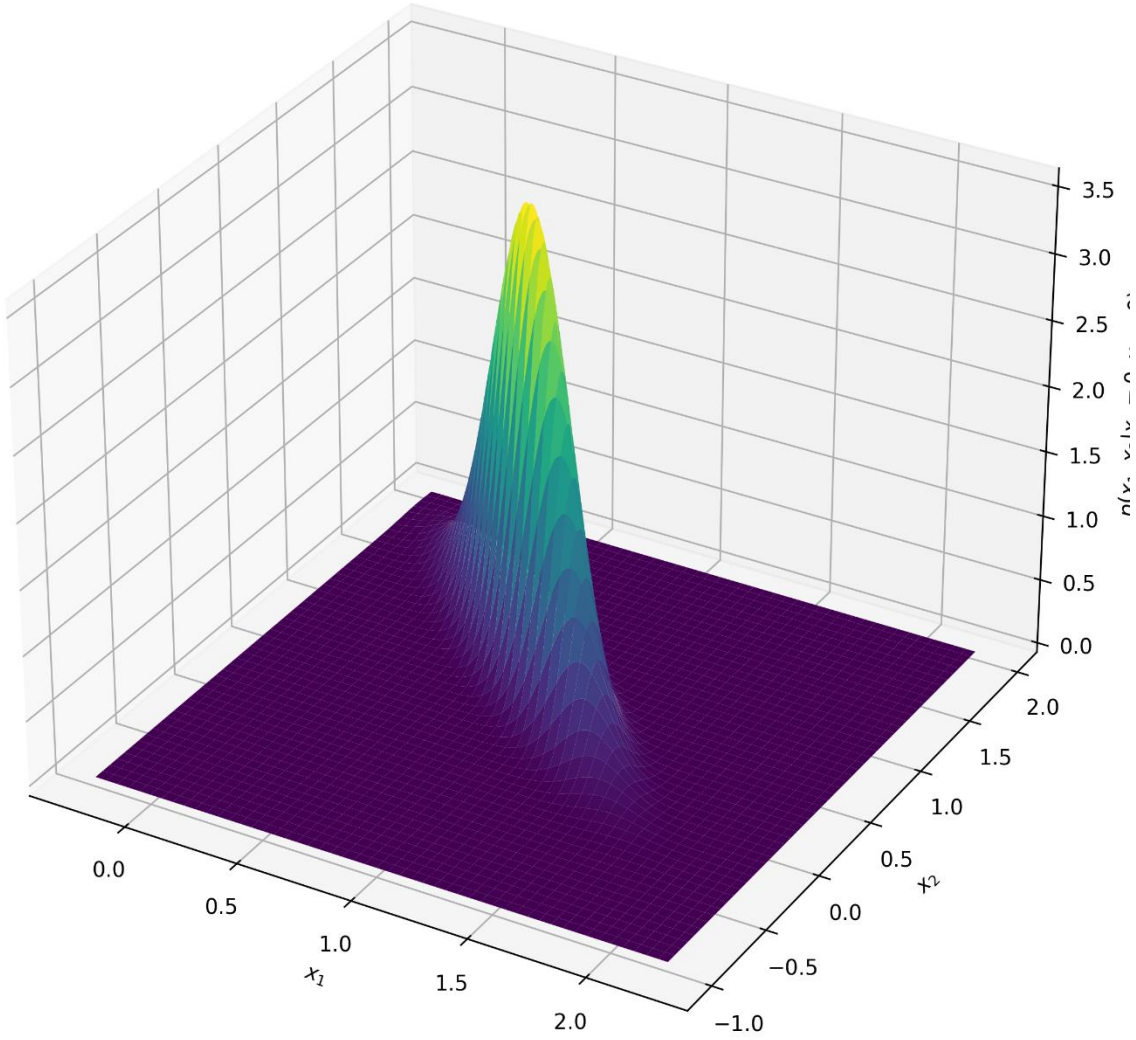
```
# Define a function to generate a random pair of numbers from a
bivariate normal distribution.
# The mean vector is [0.8, 0.8], and the covariance matrix is [[0.1, -
0.1], [-0.1, 0.12]].
def generate_pair():
    return np.random.multivariate_normal([0.8, 0.8], [[0.1, -0.1],[-0.1,
     0.12]])

# Call the 'generate_pair()' function to generate a random pair of
numbers from a bivariate normal distribution.
mu_t = generate_pair()
#Print the generated pair of numbers
print(mu_t)
```

OUTPUT:

```
[0.79116048 0.83708266]
```

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

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

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

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

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

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

#Set labels for the axes and z-axis
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
ax.set_zlabel('$p(x_1, x_2 | x_3=0, x_4=0)$')

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

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

OUTPUT:




```
# Define the number of data points to generate
N=1000
import numpy as np
# Generate random data points from a bivariate normal distribution
data = np.random.multivariate_normal([0.28, 1.18], [[2.0, 0.8], [0.8,
4.0]], N)
data
# Save the generated data to a text file named 'data.txt
np.savetxt('data.txt',data)


import pandas as pd

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

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

# Display the data loaded with NumPy
data
```

OUTPUT:

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

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

# Center the data by subtracting the sample mean from each data point
x = data - mu_ml

# Calculate the maximum likelihood (ML) estimate of the covariance
matrix using N as the denominator
cov_ml = np.dot(x.T, x) / N

# Calculate the ML estimate of the covariance matrix using (N - 1) as
the denominator, which is unbiased
cov_ml_unbiased = np.dot(x.T, x) / (N - 1)

# Print the sample mean, the ML estimate of the covariance matrix, and
the unbiased ML estimate
print(mu_ml)
print(cov_ml)
print(cov_ml_unbiased)
```

OUTPUT:

```
[0.19127533 1.12357084]
[[1.84967948 0.69930346]
 [0.69930346 4.25087478]]
[[1.85153101 0.70000347]
 [0.70000347 4.25512991]]
```

```
# Define a function called seq_ml that calculates sequential maximum
likelihood (ML) estimates for a series of data points.
def seq_ml(data):
    # Initialize a list to store sequential estimates of the mean.
    mus = [np.array([[0], [0]])]
    # Iterate through the data points (N represents the number of data
points).
    for i in range(N):
        # Extract the current data point and reshape it into a 2x1
column vector.
        x_n = data[i].reshape(2, 1)
        # Calculate the sequential ML estimate for the mean, mu_n,
using the previous estimate and the current data point.
        mu_n = mus[-1] + (x_n - mus[-1]) / (i + 1)
        # Append the current estimate to the list of sequential
estimates.
        mus.append(mu_n)
    # Return the list of sequential estimates for the mean.
    return mus


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

# Print the final mean estimate
print(mus_ml[-1])
```

OUTPUT:

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

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

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

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

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

OUTPUT:

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

```
import numpy as np

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

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

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

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

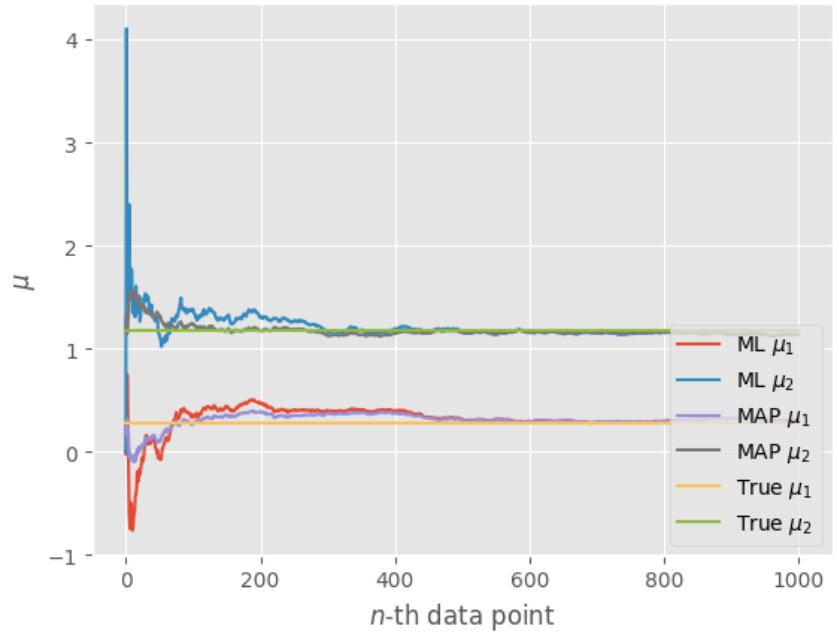
```
# Calculate sequences of mean and covariance estimates using the
'seq_map' function
mus_map, covs_map = seq_map(data, mu_p, cov_p, cov_t)

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

OUTPUT:

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

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

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

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

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

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

# Plot the mean estimates and true mean values
plt.plot(X, mus1_ml, label='ML $\mu_1$')
plt.plot(X, mus2_ml, label='ML $\mu_2$')
plt.plot(X, mus1_map, label='MAP $\mu_1$')
plt.plot(X, mus2_map, label='MAP $\mu_2$')
plt.plot(X, mus1_t, label='True $\mu_1$')
plt.plot(X, mus2_t, label='True $\mu_2$')

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

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


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

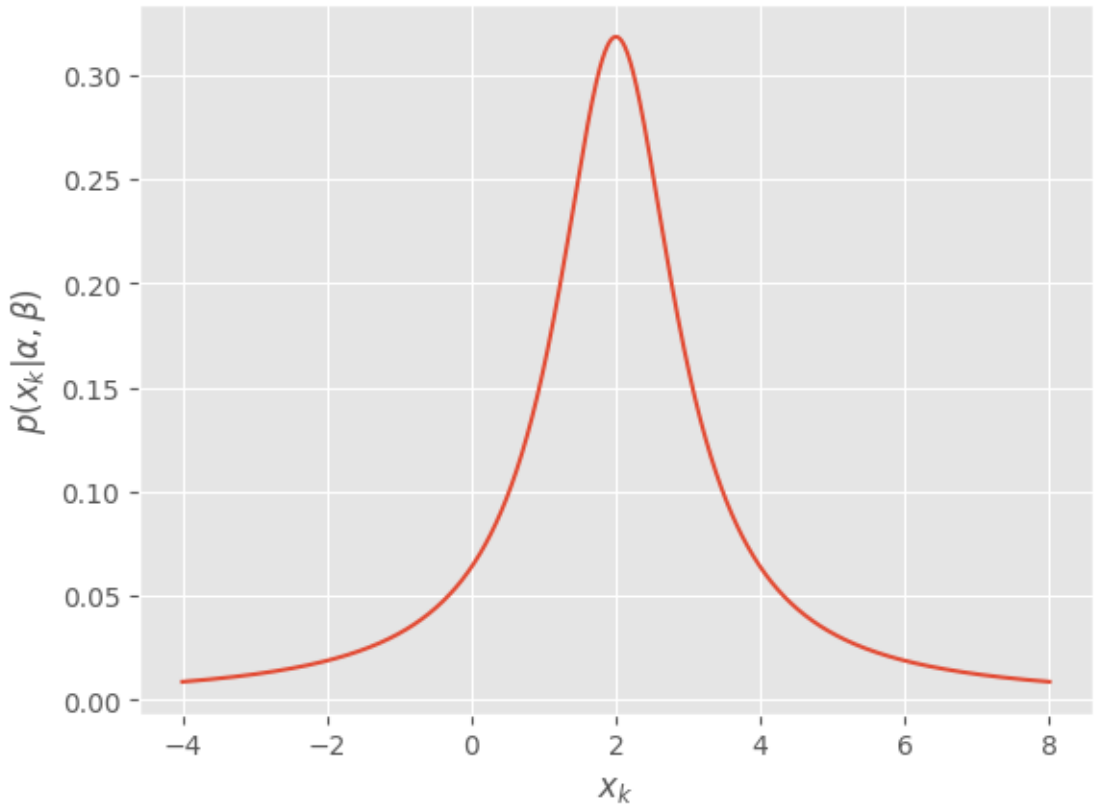
OUTPUT:




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

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

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

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

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

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

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

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

OUTPUT:

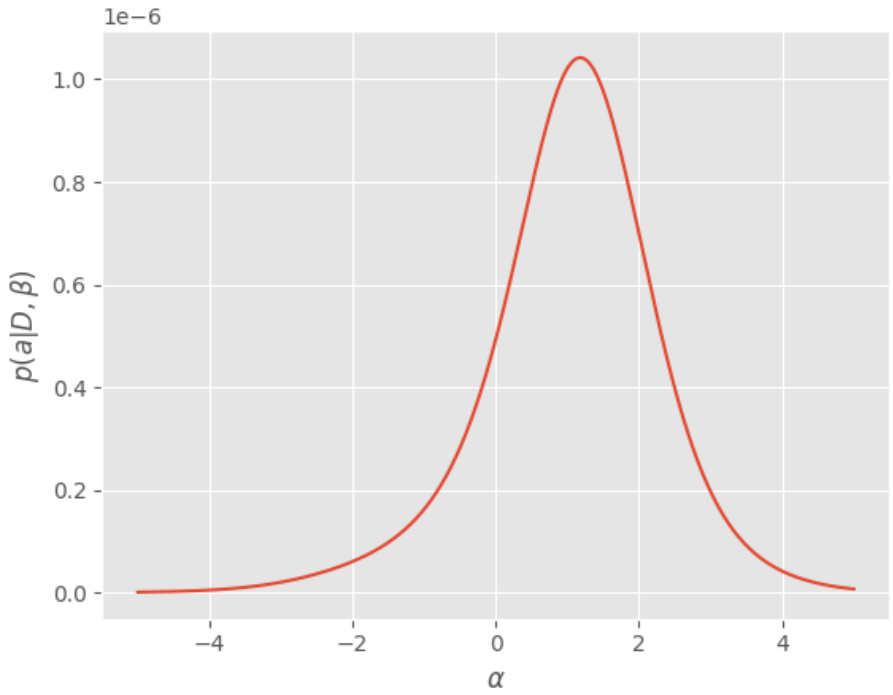



```
# Define a function to calculate the joint probability of observing
values x given alpha and beta
def p_a(x, alpha, beta):
    return np.product(beta / (np.pi * beta**2 + (x-alpha)**2))
```

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

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

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

# Define the fixed beta parameter
beta = 1

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

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

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

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

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

OUTPUT:




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

OUTPUT:

```
-0.18333333333333326
1.1761761761761758
```

```
import numpy as np
```

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

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

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

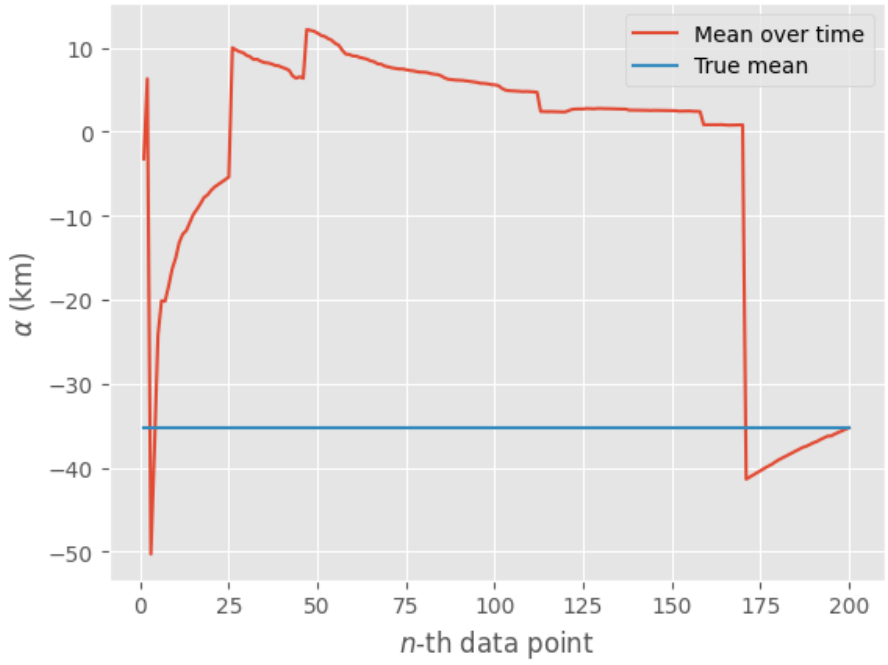
OUTPUT:

```
1.343863693601337 1.8988905573522503
```

```
import numpy as np

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

# Number of data points
N = 200

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

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

import numpy as np
import matplotlib.pyplot as plt

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

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

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

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

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

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

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

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

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

OUTPUT:



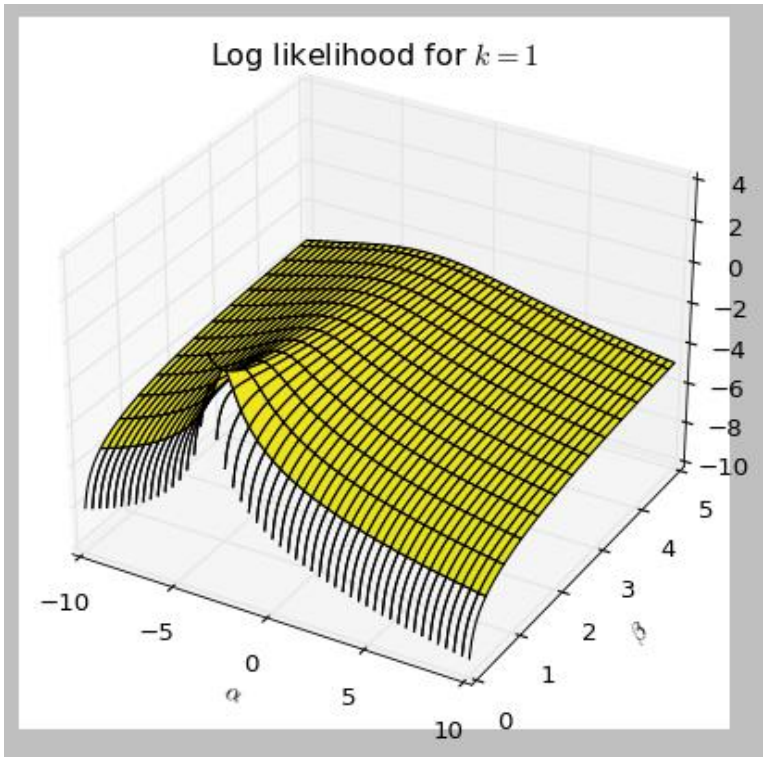

```
print(locations.mean())
```

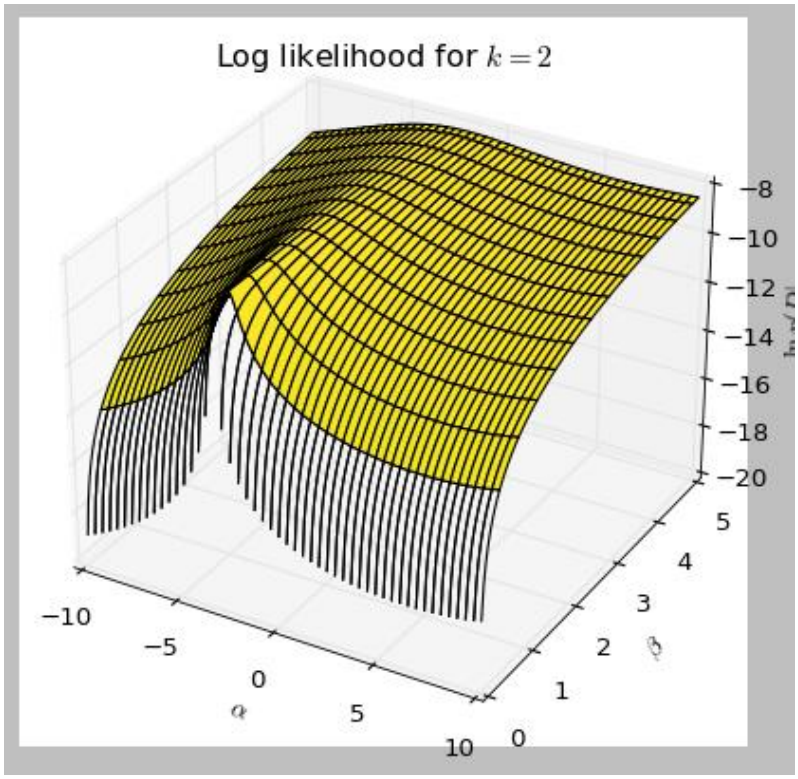
OUTPUT:

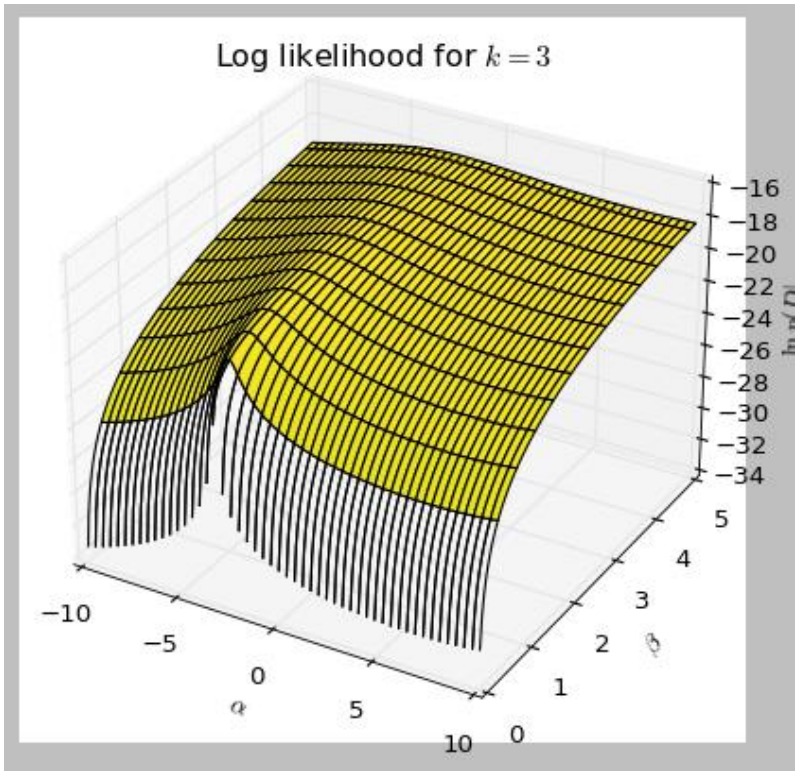
```
3.292333775274883
```

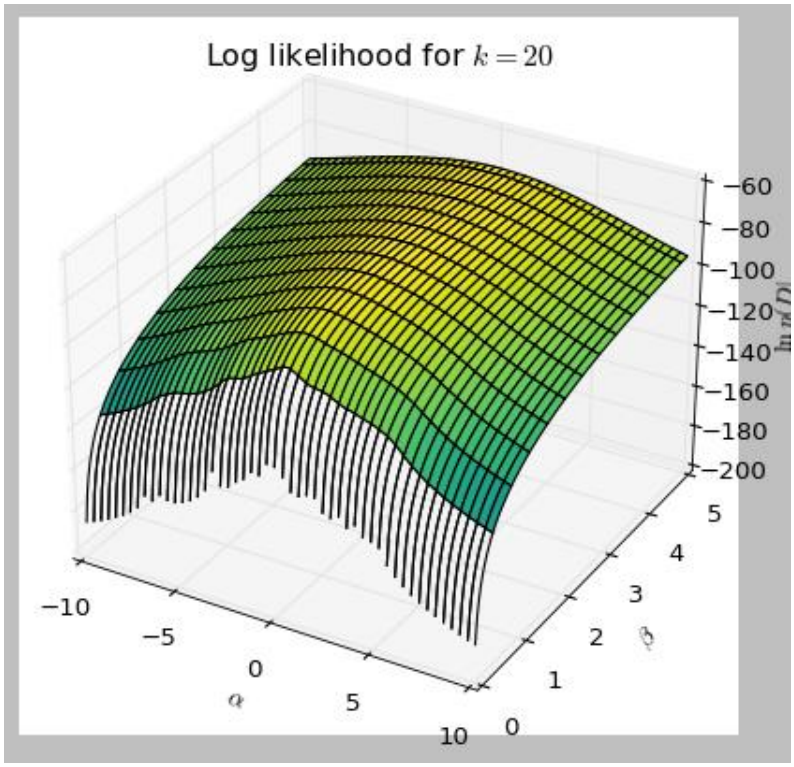
```
import numpy as np
import matplotlib.pyplot as plt

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

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

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

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

    # Calculate the likelihood for each combination of 'alpha' and
'beta'
    likelihood = k * np.log(betas / np.pi)

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

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

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

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


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

OUTPUT:


Log likelihood for $k = 1$
4
2
0
−2
−4
−6
−8
−10
−10
−5
0
5
10
a
0
1
2
3
4
5
b


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


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


Log likelihood for $k = 20$
−60
−80
−100
−120
−140
−160
−180
−200
−10
−5
0
5
10
a
0
1
2
3
4
5
b

```
from scipy.optimize import fmin

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

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

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

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

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

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

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

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

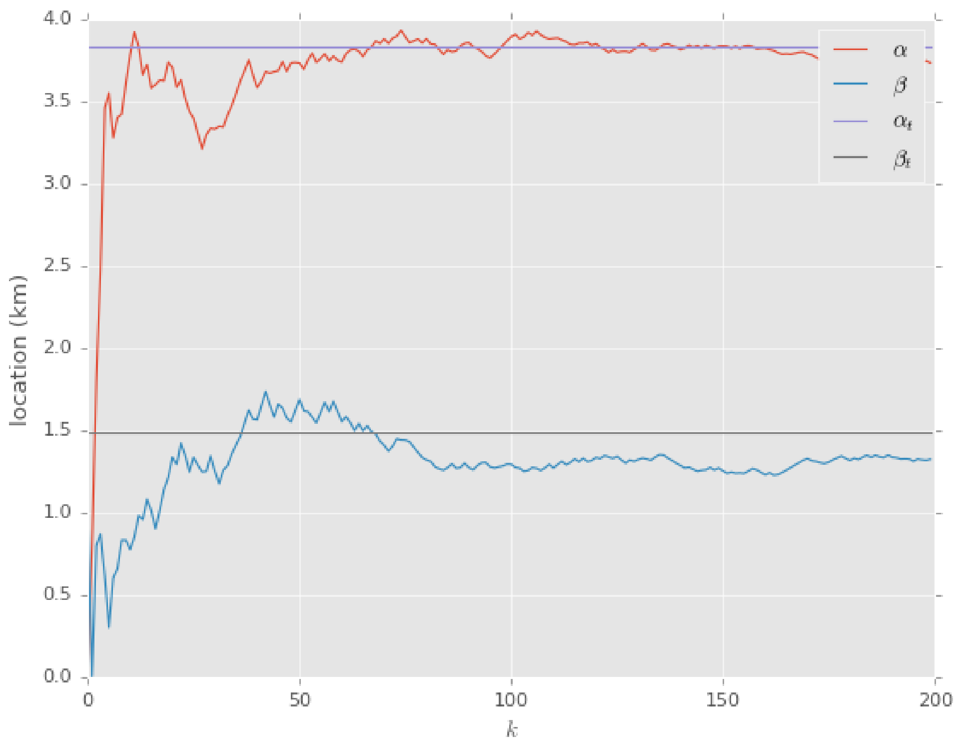
```
print(alphas[-1], betas[-1])
```

OUTPUT:

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

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

OUTPUT:



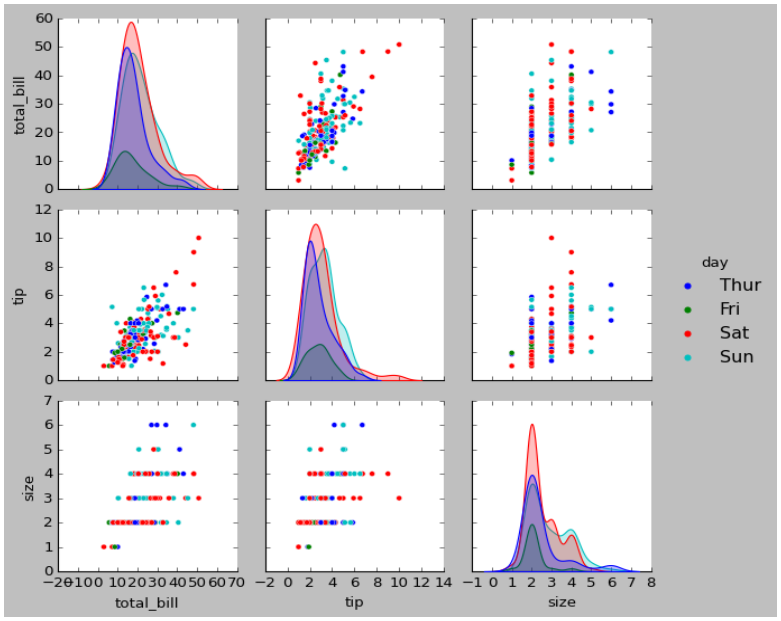
```
import pandas as pd
import numpy as np

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

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

OUTPUT:

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

[891 rows x 1 columns]
```

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

OUTPUT:


total_bill
tip
size
day
Thur
Fri
Sat
Sun

Lab-3/2203A52201/sec-AB:

Lab 03: Implementation of Linear Regression Model Using US Housing Data

```
#Import Necessary Libraries for data analysis and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Read a CSV file named "USA_Housing.csv" and store its contents in a
DataFrame named 'df'
df = pd.read_csv("USA_Housing.csv")
# Display the first few rows of the DataFrame to inspect the data
df.head()
```

OUTPUT:

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

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

OUTPUT:

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

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

OUTPUT:

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

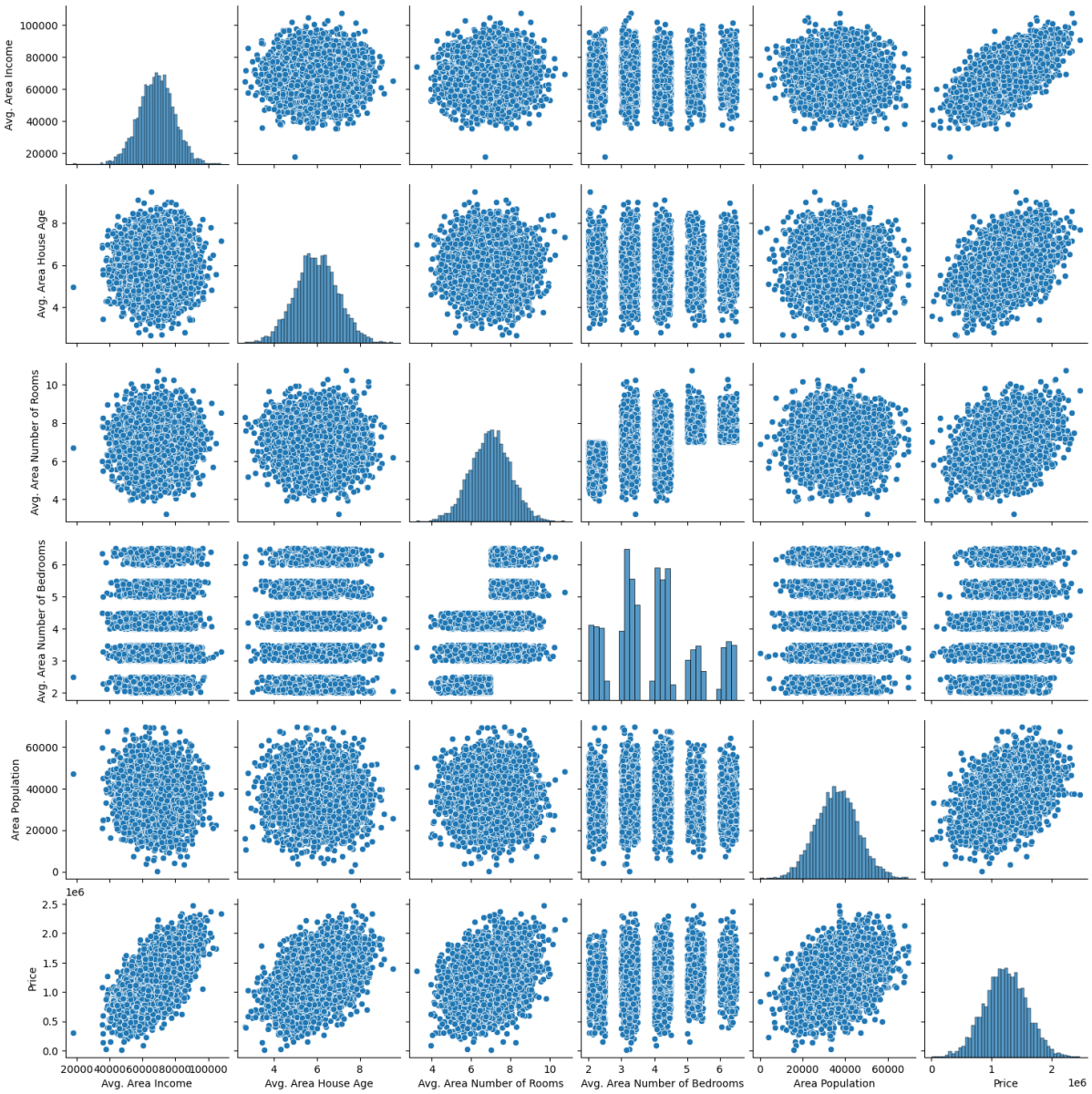
```
#display columns
df.columns
```

OUTPUT:

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

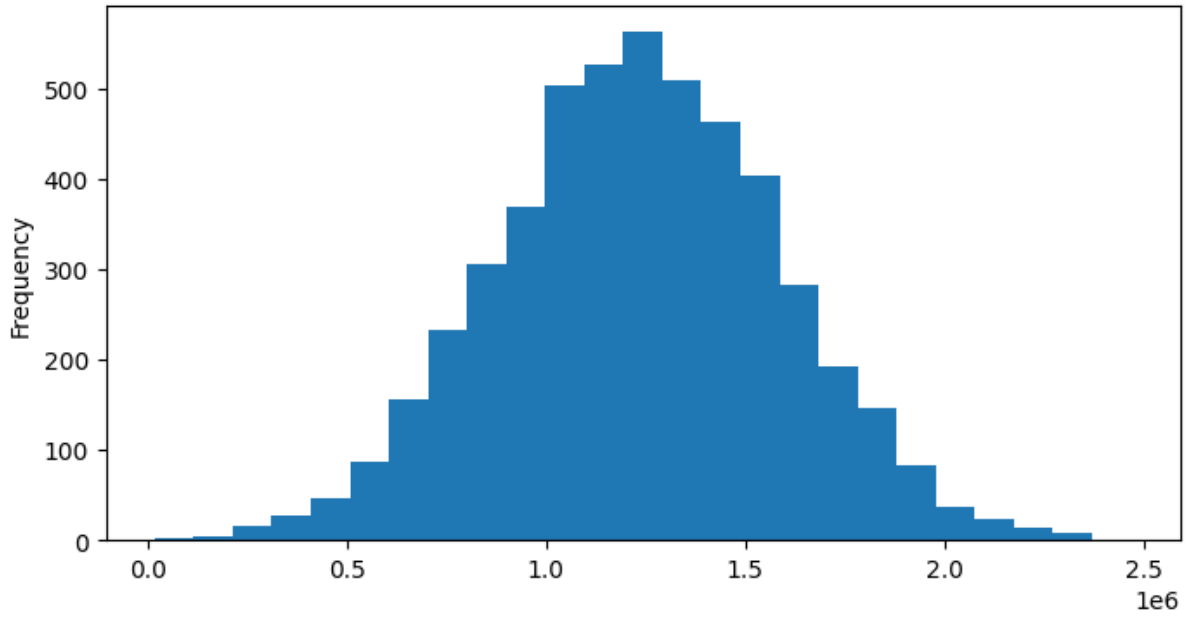
```
# Create a pair plot to visualize relationships between variables in
the DataFrame 'df'
sns.pairplot(df)
```

OUTPUT:



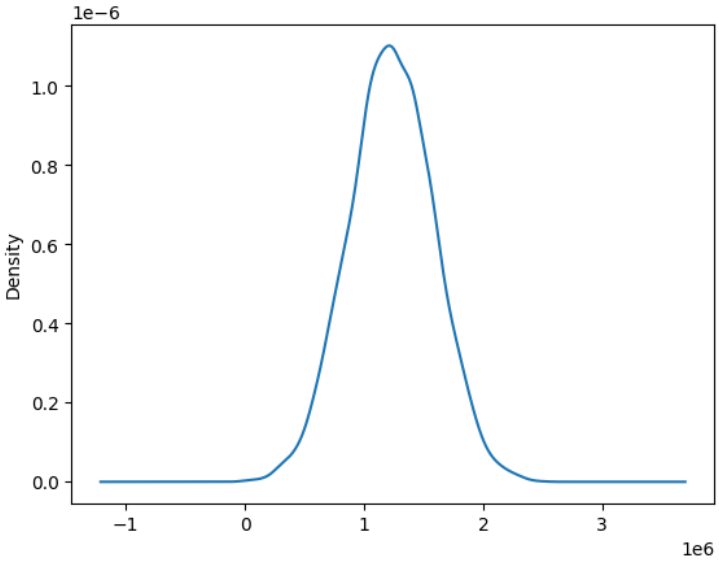

```
# Create a histogram plot for the 'Price' column with 25 bins and a specific figure size
df['Price'].plot.hist(bins=25,figsize=(8,4))
```

OUTPUT:




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

OUTPUT:



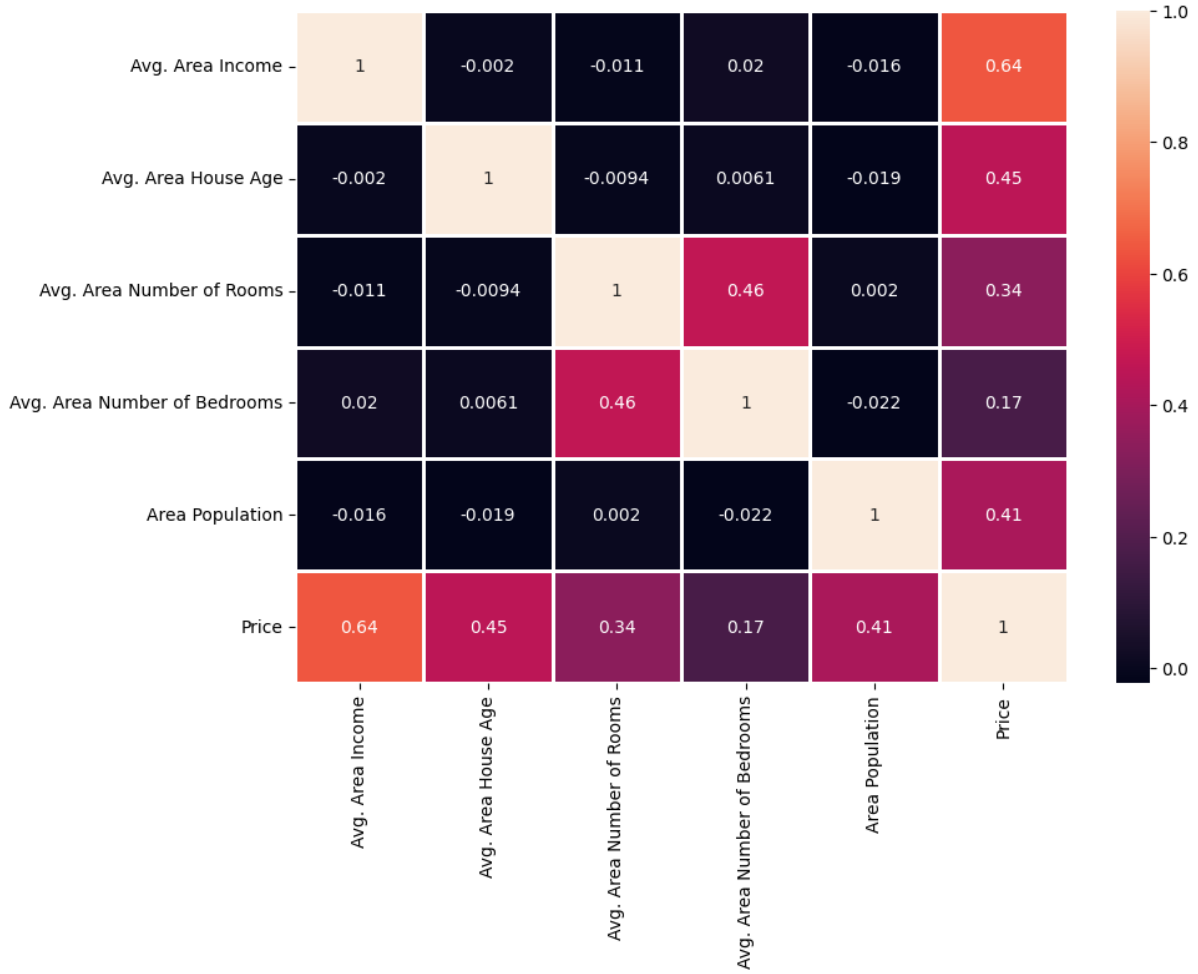

```
# Calculate the correlation between numerical columns in the DataFrame
'df'
df.corr()
```

OUTPUT:

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

```
# Set the figure size for the heatmap to make it 10x7 inches
plt.figure(figsize=(10, 7))

# Create a heatmap of the correlation matrix for the DataFrame 'df'
# - 'df.corr()' calculates the correlation matrix
# - 'annot=True' adds numerical annotations to each cell
# - 'linewidths=2' sets the width of the lines between cells in the
heatmap
sns.heatmap(df.corr(), annot=True, linewidths=2)
```

OUTPUT:



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

OUTPUT:

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

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

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

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

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

OUTPUT:

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

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

```
X.head()
```

OUTPUT:

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

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

OUTPUT:

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

```
# Import the 'train_test_split' function from Scikit-Learn for
splitting datasets
from sklearn.model_selection import train_test_split
```

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

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

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

# Print the size (dimensions) of the training variable set 'y_train'
print("Training variable set size:", y_train.shape)

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

OUTPUT:

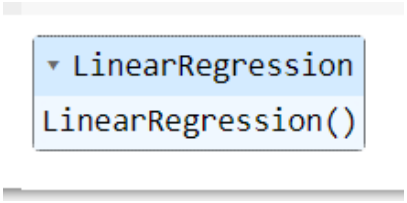
```
Training feature set size: (3500, 5)
Test feature set size: (1500, 5)
Training variable set size: (3500,)
Test variable set size: (1500,)
```

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

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

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

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

OUTPUT:




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

OUTPUT:

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

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

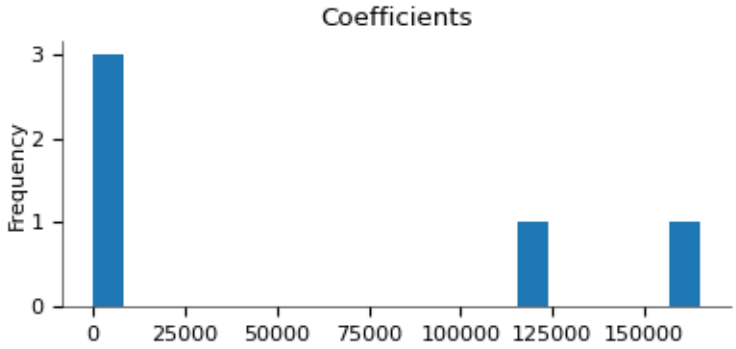
OUTPUT:

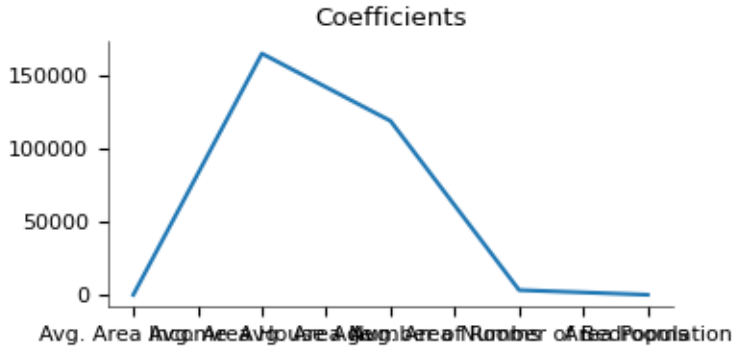
```
The coefficients of the linear model: [2.15976020e+01
1.65201105e+05 1.19061464e+05 3.21258561e+03
 1.52281212e+01]
```

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

OUTPUT:

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





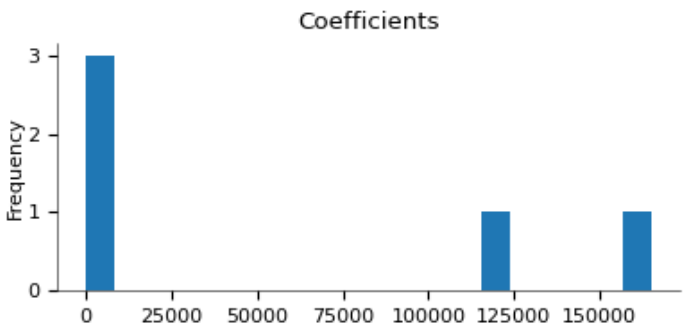

```
# Calculate the standard error for each coefficient
n = X_train.shape[0]  # Number of observations
k = X_train.shape[1]  # Number of features
```

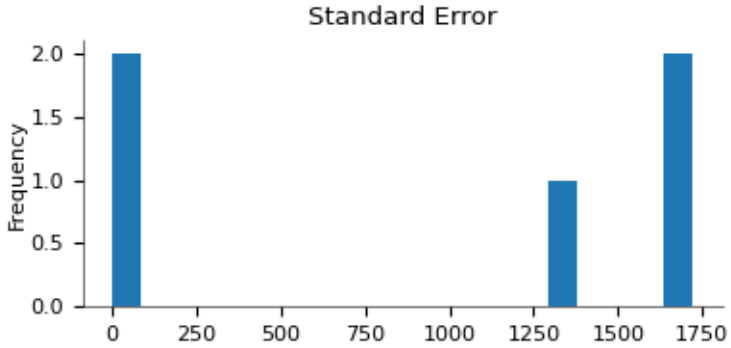
```
dfN = n - k  # Degrees of freedom for the residuals

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

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

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

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

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

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

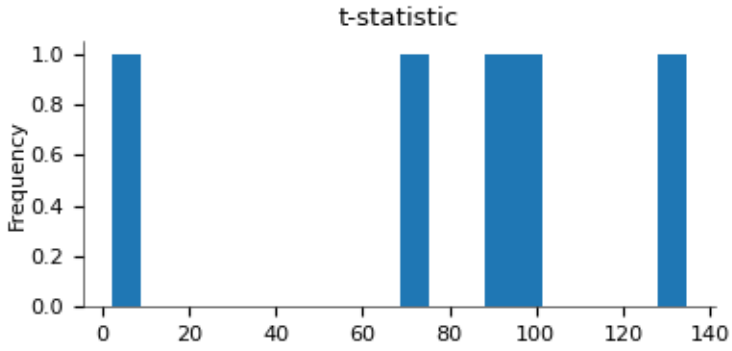
OUTPUT:

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

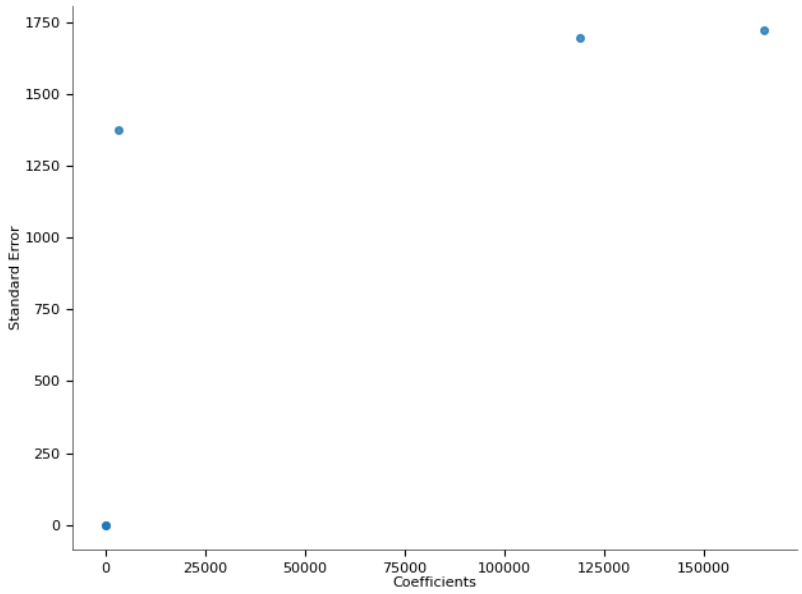
DISTRIBUTIONS:

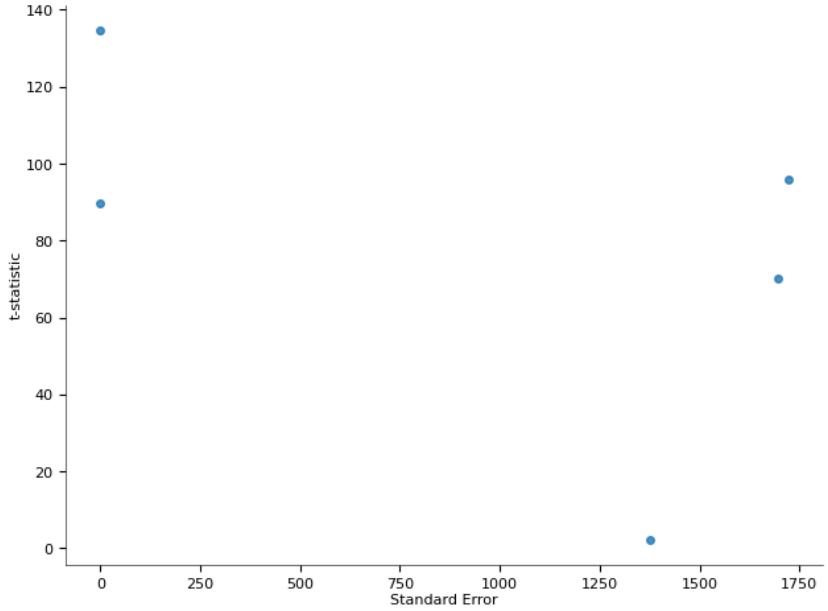




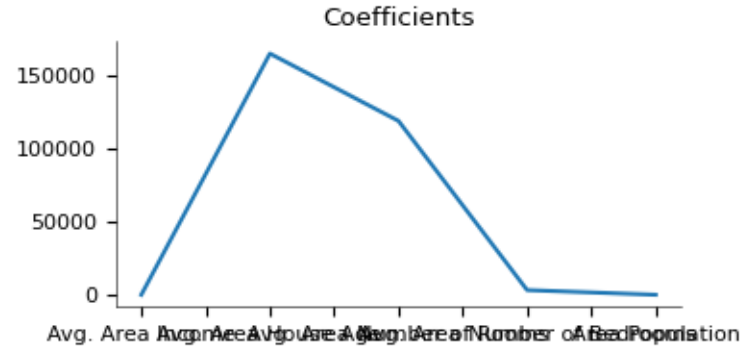


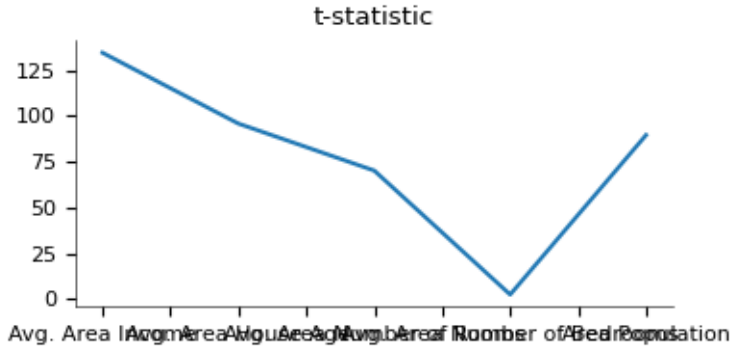
2D-DISTRIBUTIONS:


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


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

VALUES:


Coefficients
150000
100000
50000
0



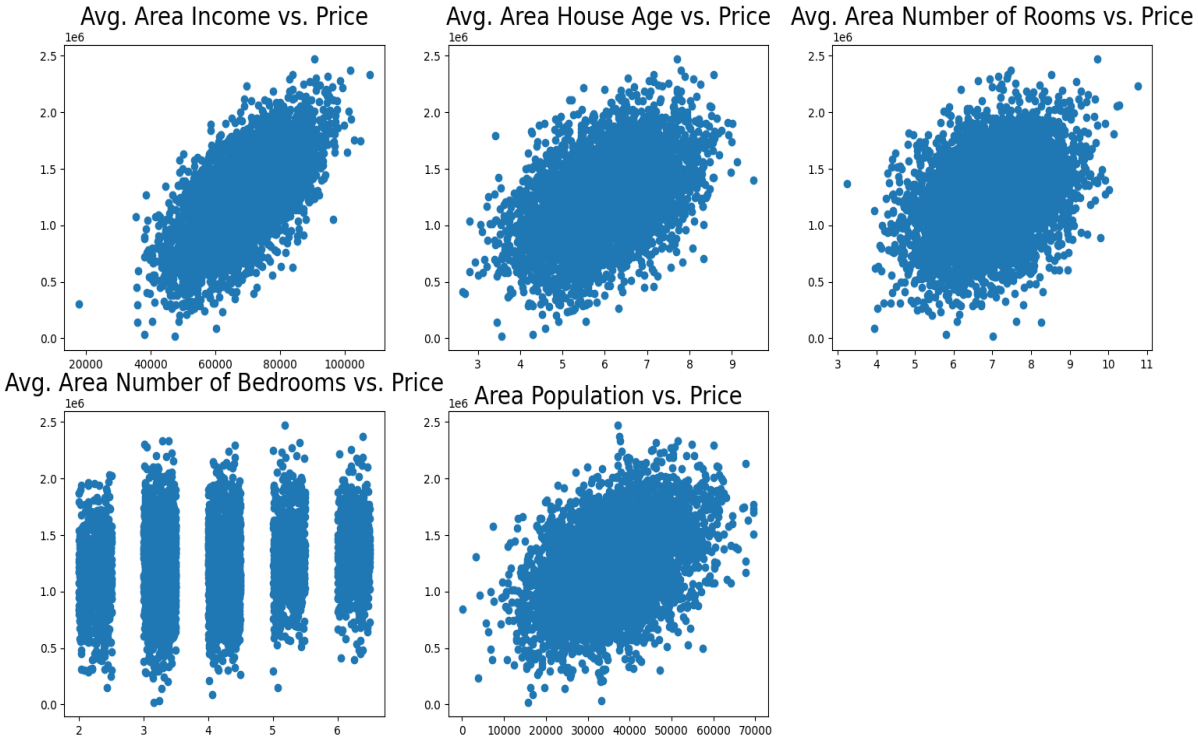

```
# Print a message indicating the features arranged in order of
importance for predicting house prices
print("Therefore, features arranged in the order of importance for
predicting the house price\n", '-'*90, sep='')

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

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

OUTPUT:

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

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

from matplotlib import gridspec

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

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

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

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

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

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

OUTPUT:




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

OUTPUT:

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

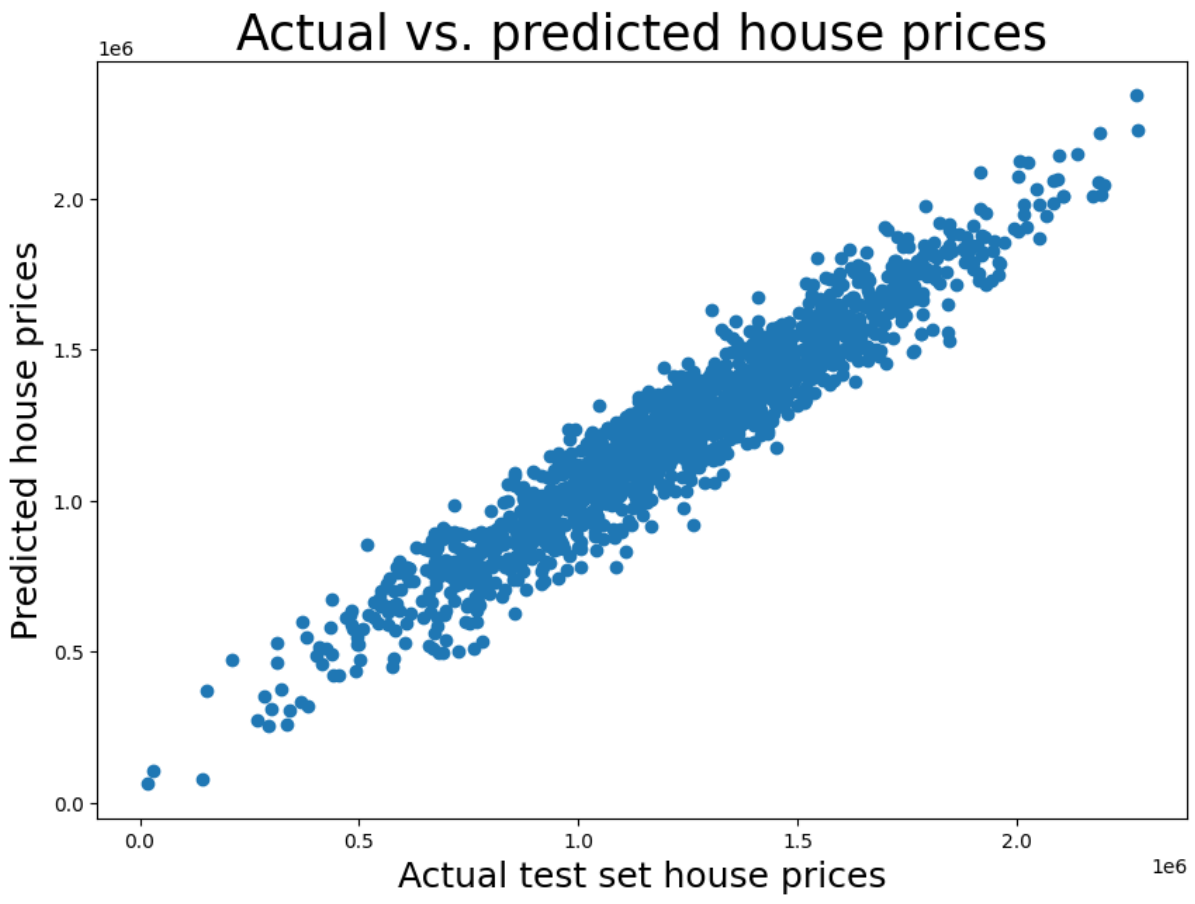
```
# Generate predictions using the linear regression model on the test
data
predictions = lm.predict(X_test)

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

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

OUTPUT:

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

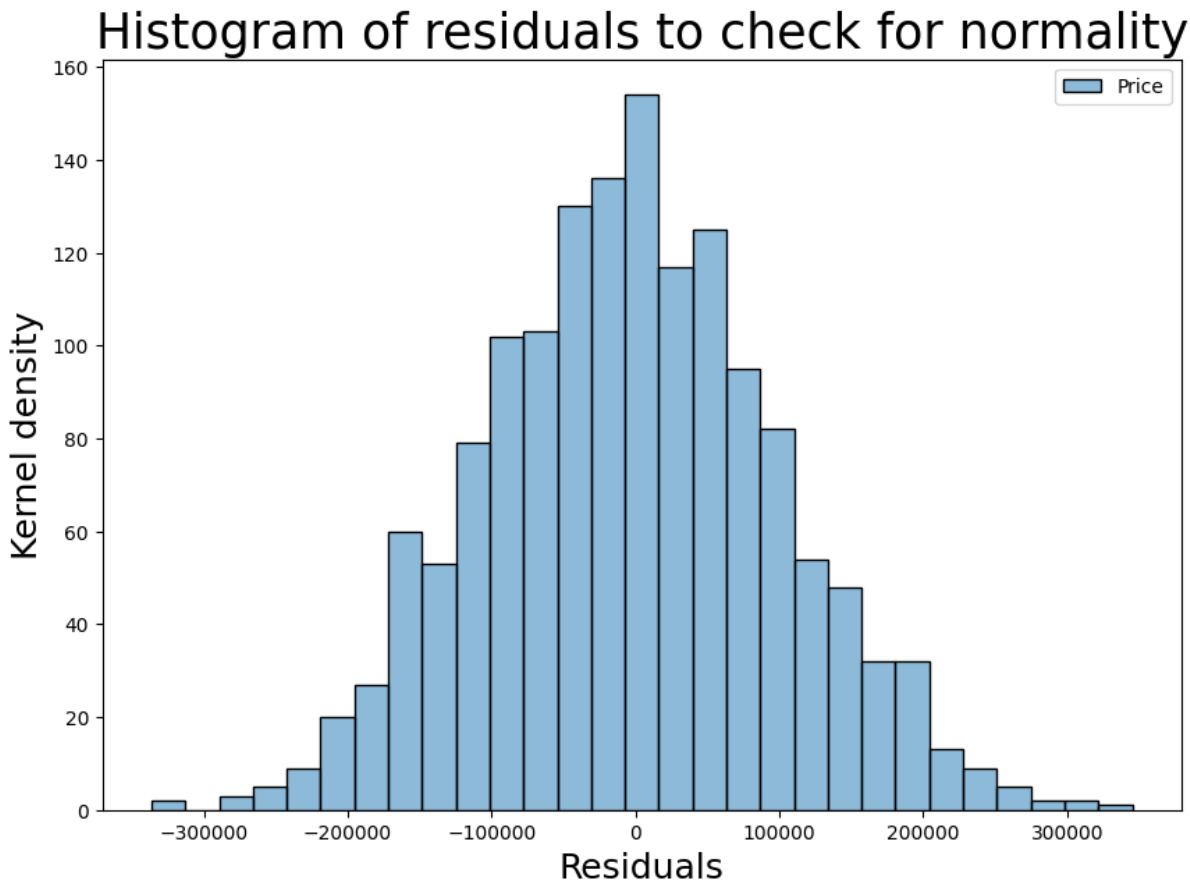
```
# Create a scatter plot to compare actual vs. predicted house prices
plt.figure(figsize=(10, 7))  # Set the figure size

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

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

OUTPUT:



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

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

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

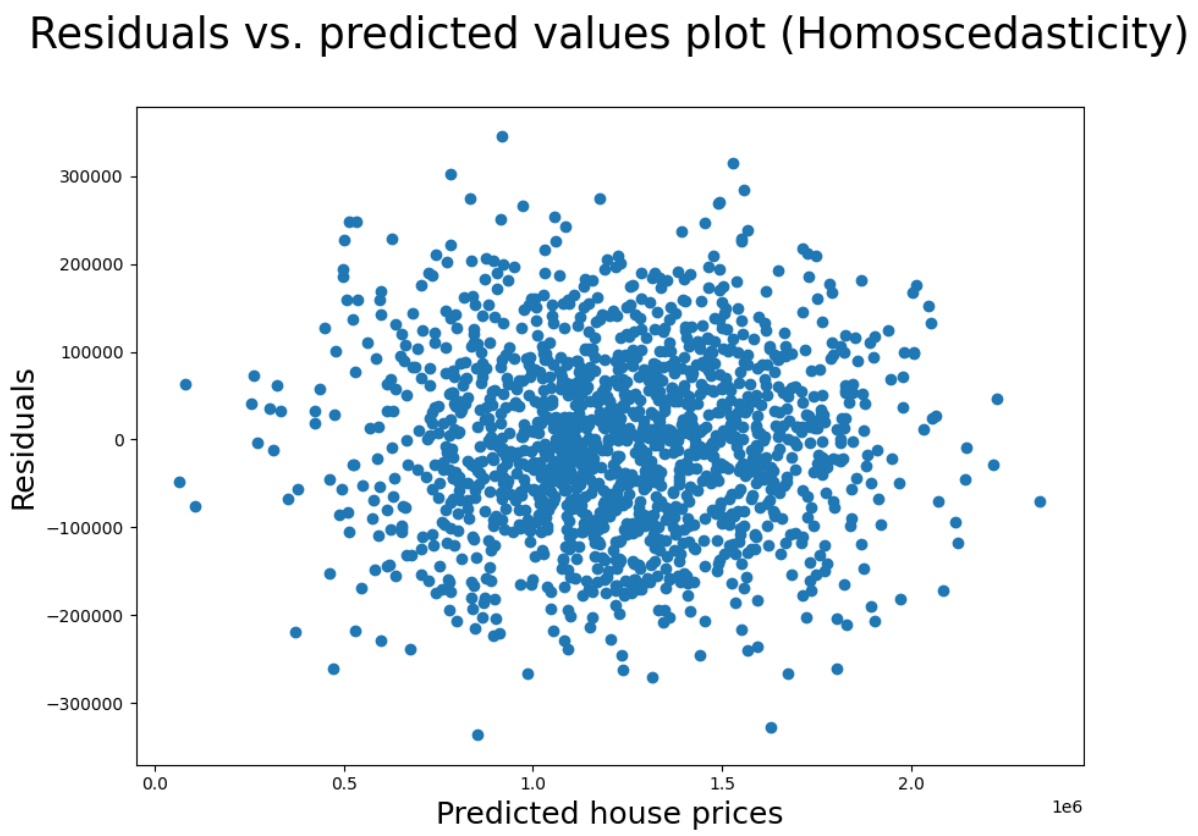
OUTPUT:




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

```
# Set the plot title and axis labels
plt.title("Residuals vs. predicted values plot (Homoscedasticity)\n",
fontsize=25)
plt.xlabel("Predicted house prices", fontsize=18)
plt.ylabel("Residuals", fontsize=18)

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

OUTPUT:




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

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

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

OUTPUT:

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

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

OUTPUT:

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

```
import numpy as np

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

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

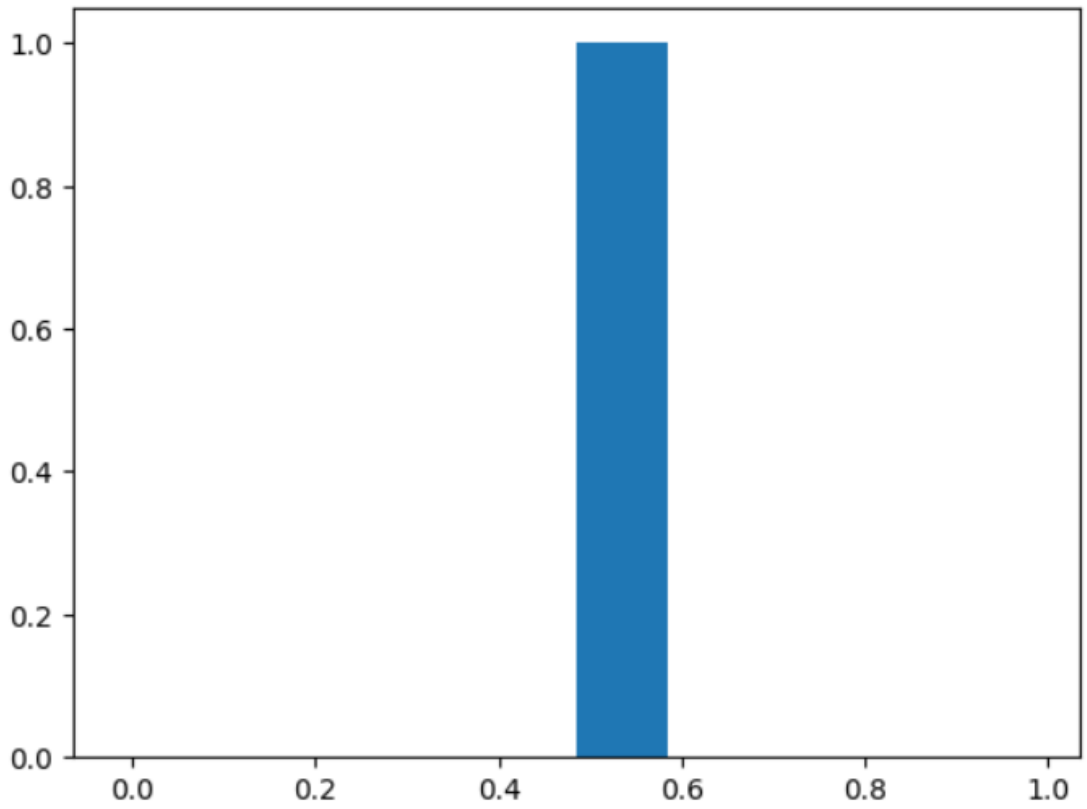
OUTPUT:

```
10.57339854753646 195.14363973516853
```

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

L

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

OUTPUT:

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



Lab-4/2203A52201/sec-AB:

Lab 04 : Implementation of Logistic Regression Model using Titanic Ship Dataset

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

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

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

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

OUTPUT:

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

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

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

[891 rows x 12 columns]
```

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

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

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

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

OUTPUT:

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

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

OUTPUT:

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

```
df.columns
```

OUTPUT:

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

```
import numpy as np
import pandas as pd
file_path="/content/titanic_train.csv"
df=pd.read_csv(file_path)
column_name='Age'
data=df[column_name]
```

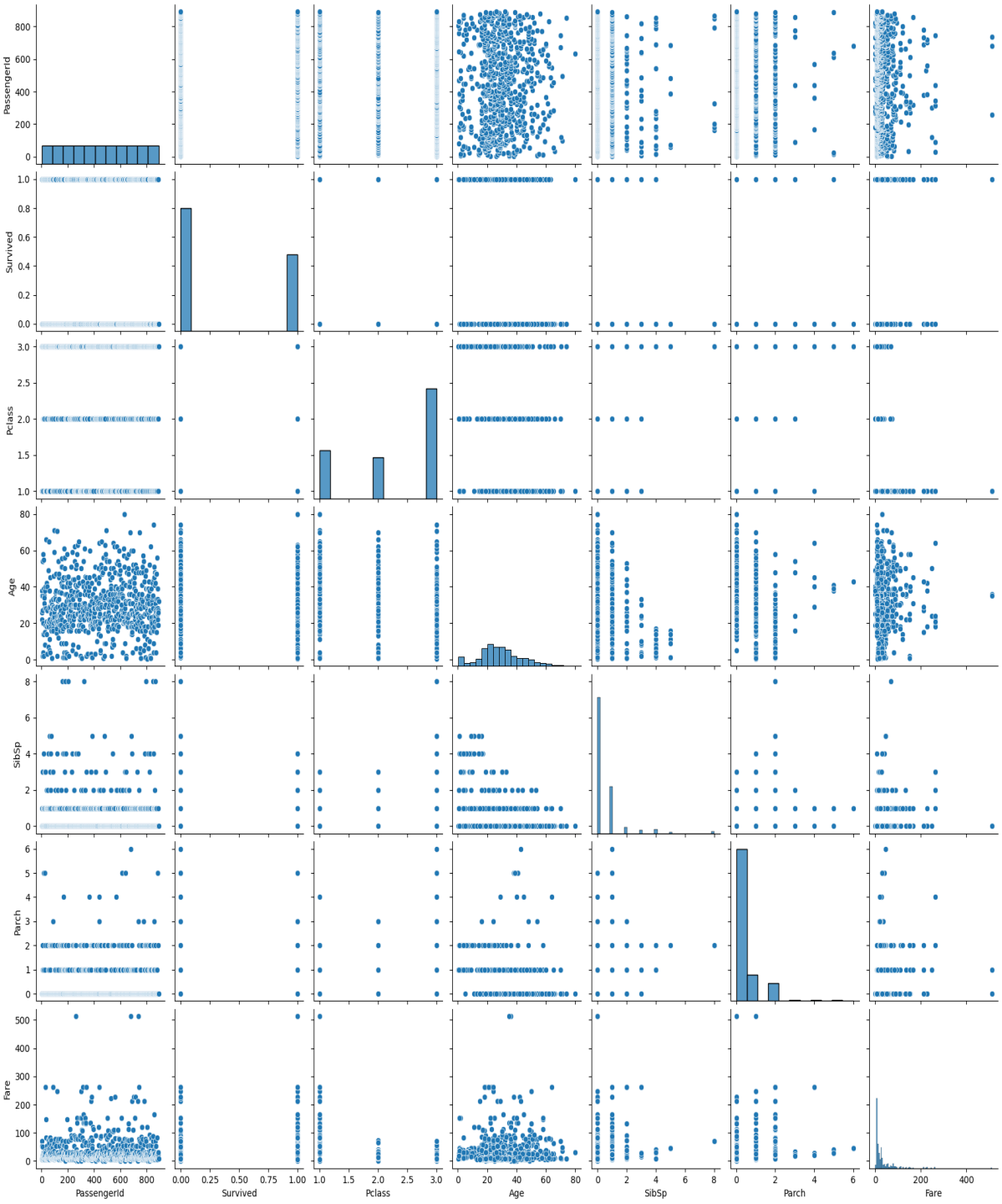
```
min_val=np.min(data)
max_val=np.max(data)
std_dev=np.std(data)
quartiles=np.percentile(data,[25,50,75,90])
print("Summary of the dataset:")
print("Minimum:",min_val)
print("Maxmimum:",max_val)
print("Standard Deviation:",std_dev)
print("25th percentile:",quartiles[0])
print("50th percentile(Median):",quartiles[1])
print("75th percentile:",quartiles[2])
print("90th percentile:",quartiles[3])
```

OUTPUT:

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

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

OUTPUT:


PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare

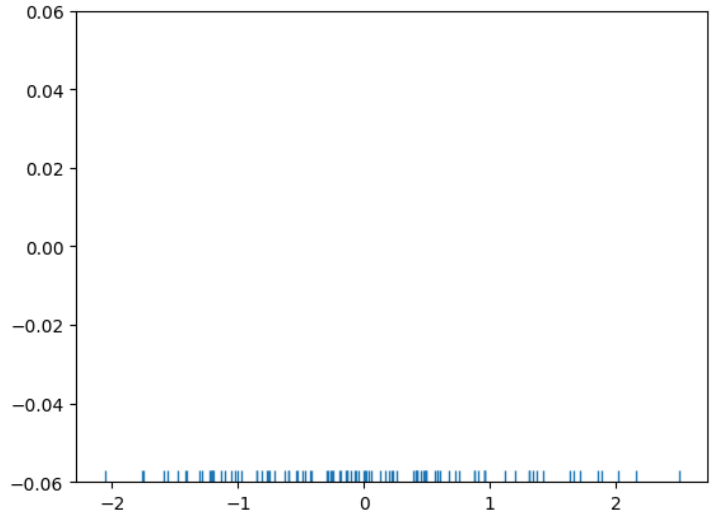
Lab-5/2203A52201/sec-AB:

Lab 05: Implementation of kernel density estimation for feature space and implementation of L1,L2 regularization using Boston Housing Dataset

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

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

#create another rugplot
sns.rugplot(dataset);

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

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

OUTPUT:




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

OUTPUT:

```
-4.555218342250088 5.597136045854967
```

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

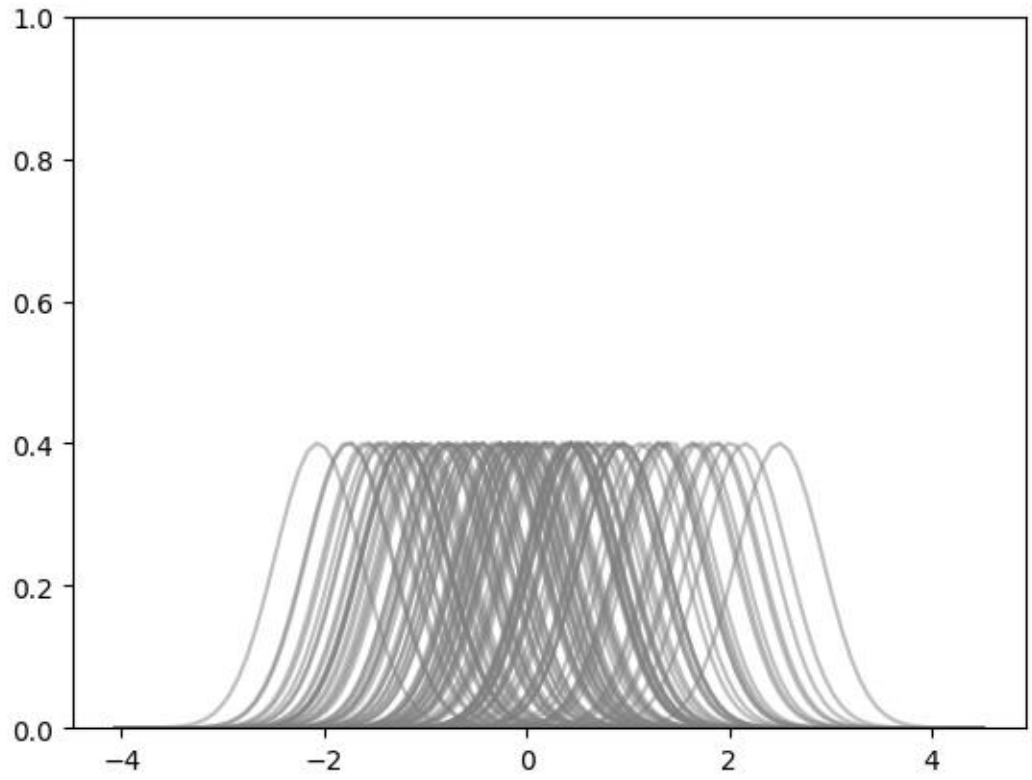
OUTPUT:

```
0.42197981688001757
```

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

#plot each basis fn
for data_point in dataset:

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

  #scale for plotting
  kernel=kernel/kernel.max()
  kernel=kernel*.4
  plt.plot(x_axis,kernel,color='grey',alpha=0.5)

plt.ylim(0,1)
```

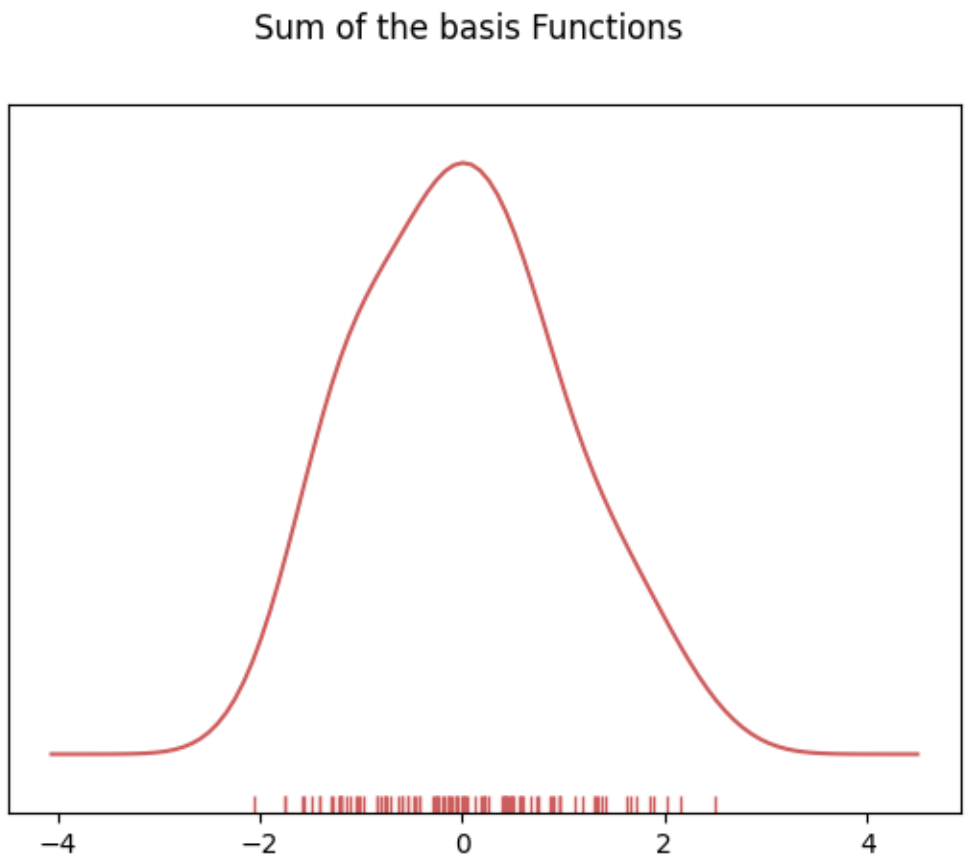
OUTPUT:

```
(0.0  , 1.0)
```




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

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

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

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

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

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

OUTPUT:



REGULARIZATION:

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

# Enable inline plotting for Jupyter Notebook
%matplotlib inline

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

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

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

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

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

OUTPUT:

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

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

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

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

OUTPUT:

```
(20640, 8)
```

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

OUTPUT:

```
(20640,)
```

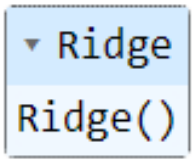
```
#ridgereg = linear_model.Ridge() creates an instance of the Ridge
regression model from scikit-learn's linear_model module.
ridgereg=linear_model.Ridge()
```

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

```
# Train the Ridge regression model on the training data
```

```
ridgereg.fit(x_train,y_train)
```

OUTPUT:




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

OUTPUT:

```
0.5233577422311327
```

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

OUTPUT:

```
0.6093458881478931
```

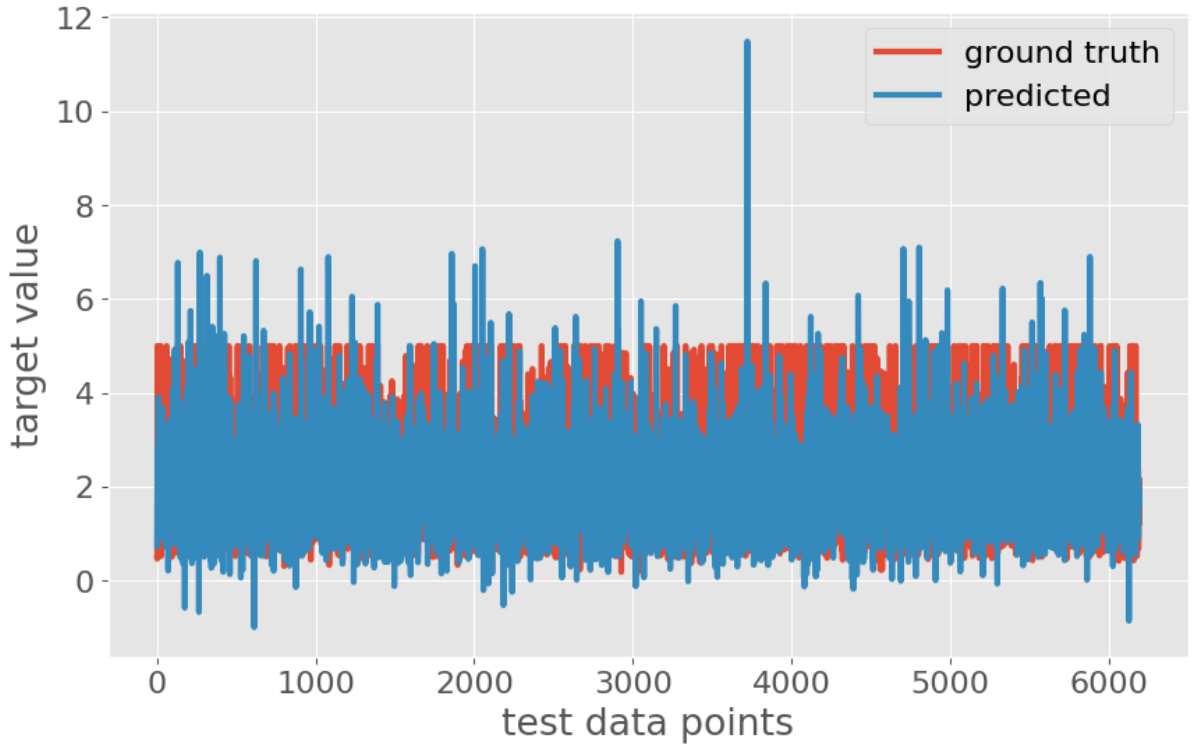
```
#make predictions on the test data using the Ridge regression model
(ridgereg) that was previously trained.
y_pred=ridgereg.predict(x_test)

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

OUTPUT:

```
0.530505269093369
```

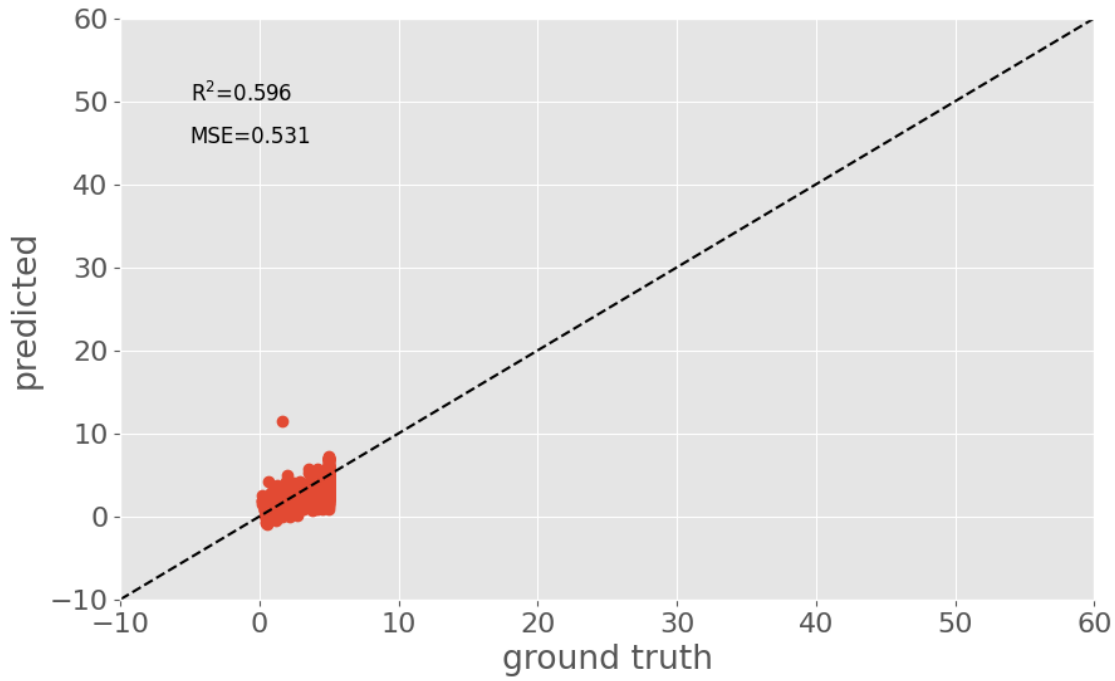
```
import matplotlib.pyplot as plt

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

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

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

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

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

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

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

OUTPUT:



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

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

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

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

# Label the x-axis as 'ground truth' and the y-axis as 'predicted'
plt.xlabel('Ground Truth')
plt.ylabel('Predicted')
# Create a string with the R-squared (coefficient of determination)
value
scorestr = r'R$^2$=%.3f' % ridgereg.score(x_test, y_test)

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

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

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

OUTPUT:



Lab-6/2203A52201/sec-AB:

Lab 06: Implementation of Dimensionality reduction using principal component analysis (PCA)

```
# For data manipulation
import pandas as pd

# For numerical operations
import numpy as np

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

# For data preprocessing
from sklearn import preprocessing

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

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

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

```
#display the table
df
```

OUTPUT:

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

150 rows × 5 columns

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

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

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

# Extract the target data from the DataFrame
y = df.loc[:, ['class']].values

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

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

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

```
# Create a new DataFrame to store the principal components
principalDataframe=pd.DataFrame(data=principalComponents,columns=['PC1'
,'PC2'])
# Extract the target variable (class) from the original DataFrame
targetDataframe = df[['class']]

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

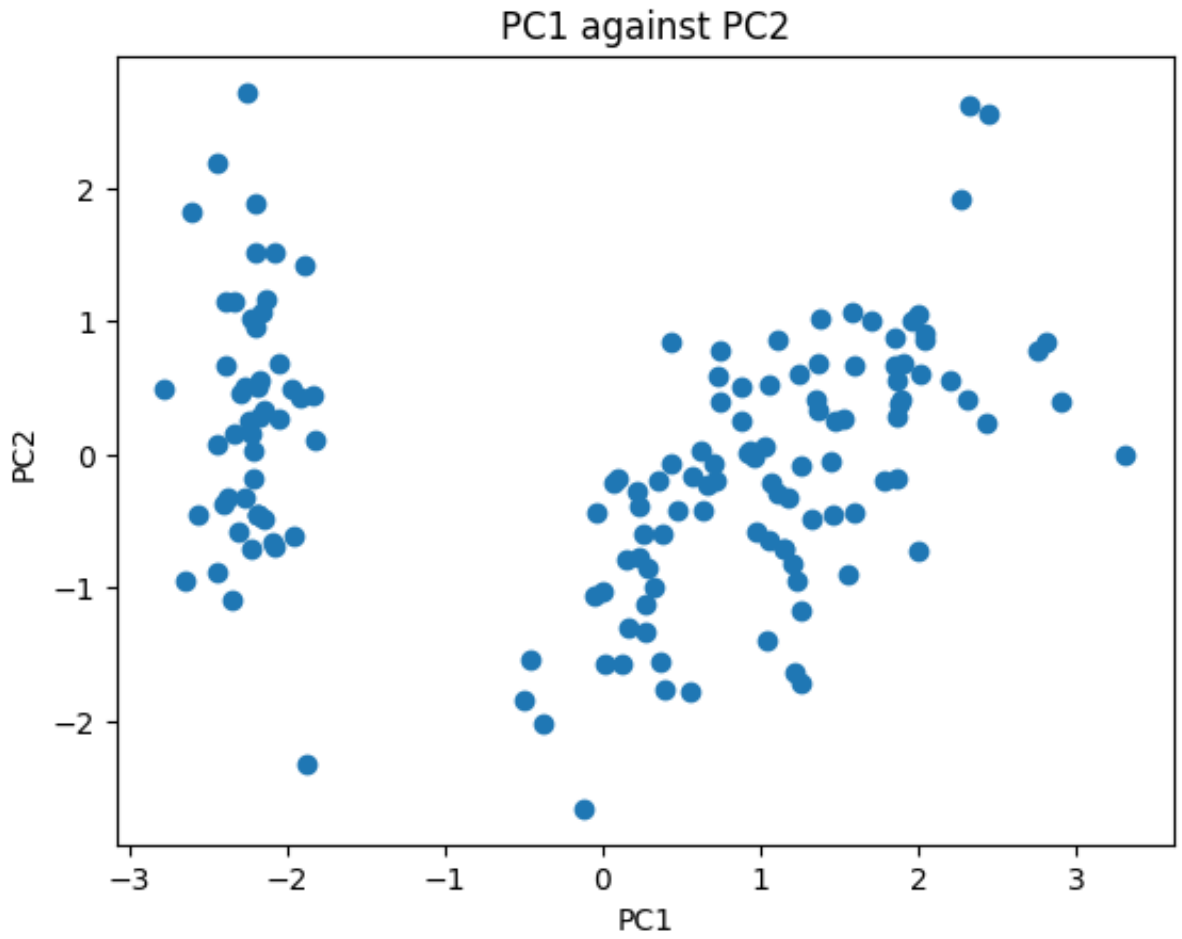
OUTPUT:

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

150 rows × 3 columns

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

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

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

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

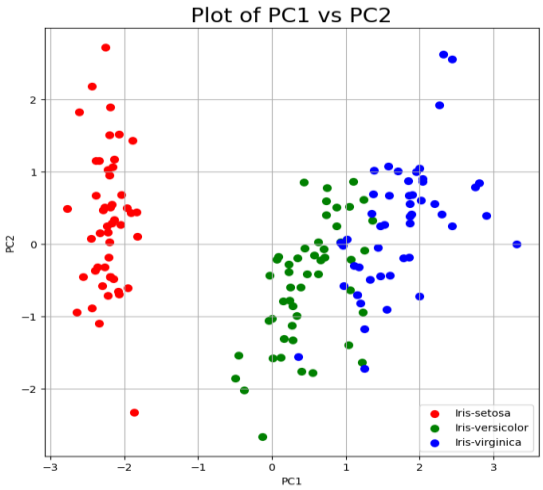
OUTPUT:




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

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

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

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

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

# For each target class and color, scatter plot the data points
for target, color in zip(targets, colors):
    # Filter the data for the current target class
    indicesToKeep = newDataframe['class'] == target
```

```
    ax.scatter(newDataframe.loc[indicesToKeep, 'PC1'],
               newDataframe.loc[indicesToKeep, 'PC2'],
               c=color,
               s=50)

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

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

OUTPUT:




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

OUTPUT:

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

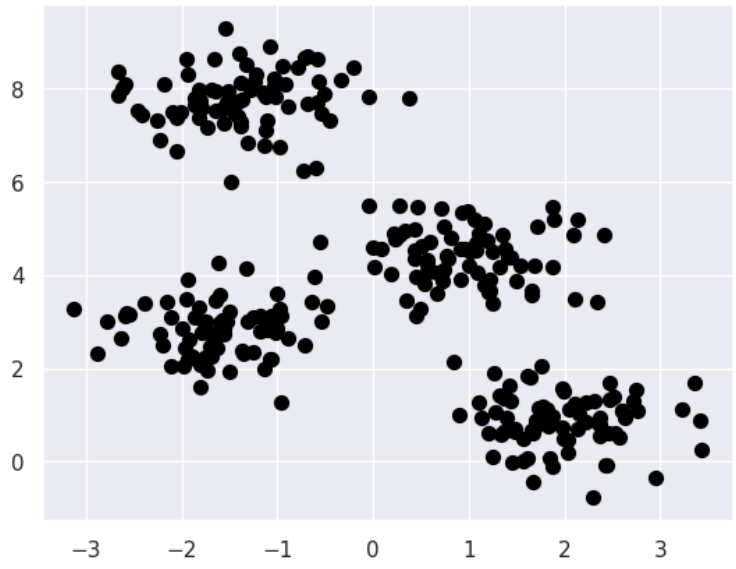
Lab-7/2203A52201/sec-AB:

Lab 07: Implementation of K-Means Clustering using Synthetic Dataset and Implementation of Gaussian Mixture Model using Synthetic Dataset.

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

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

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

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

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

OUTPUT:

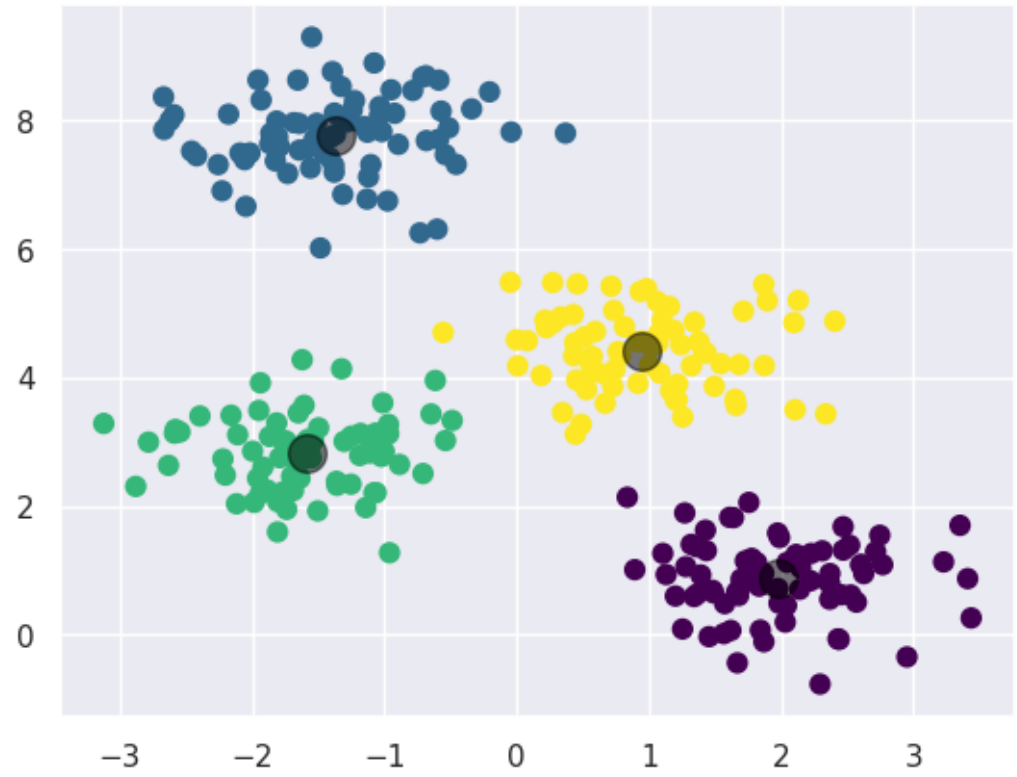



```
# Import the necessary libraries
from sklearn.cluster import KMeans  # Import the KMeans clustering
algorithm
```

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

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

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

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

# Get the cluster centers
centers = kmeans.cluster_centers_

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

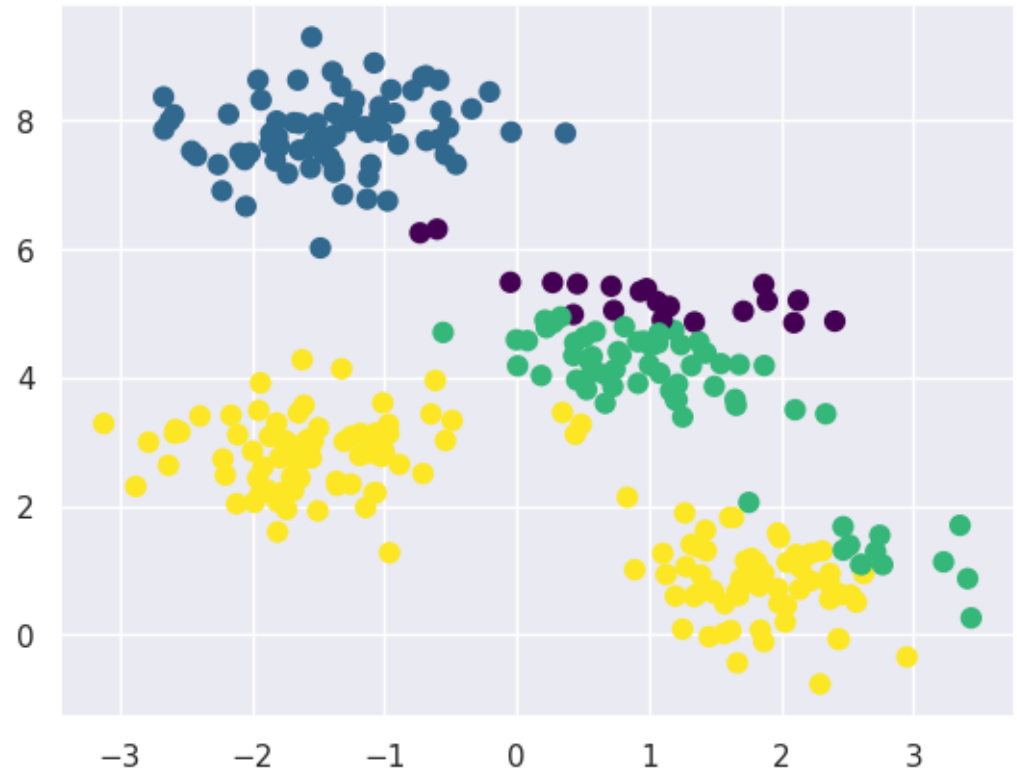
OUTPUT:




```
from sklearn.metrics import pairwise_distances_argmin
def find_clusters(X, n_clusters, rseed=2):
  # 1. Randomly choose clusters
    rng = np.random.RandomState(rseed)
```

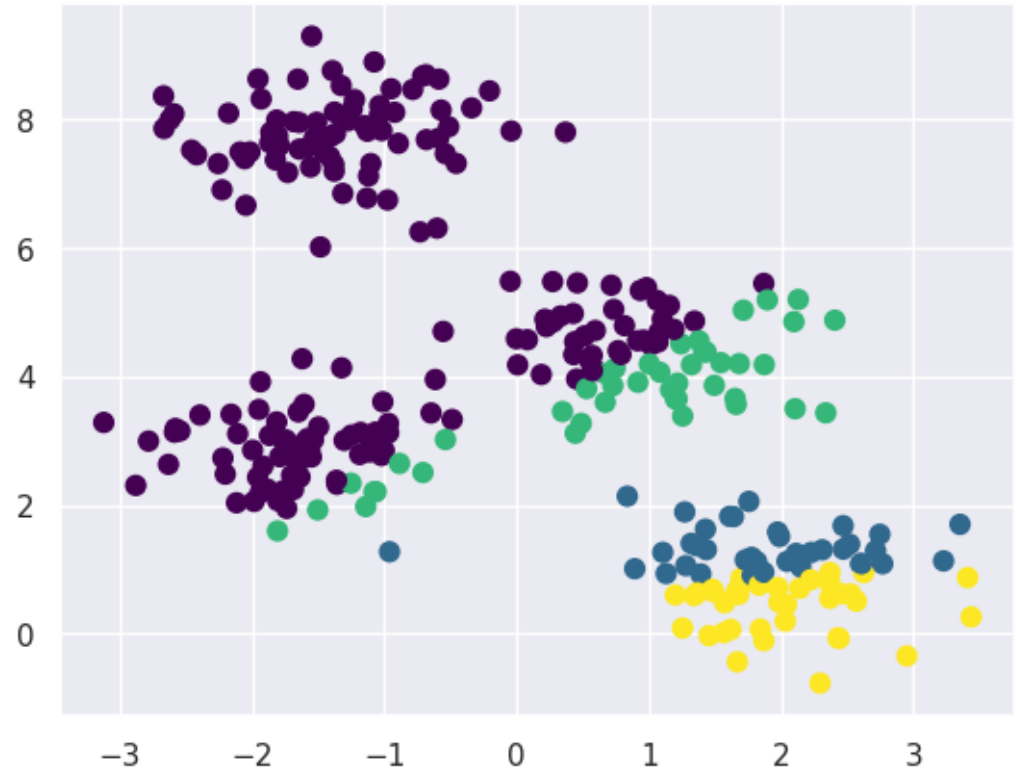
```
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]

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

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

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

OUTPUT:



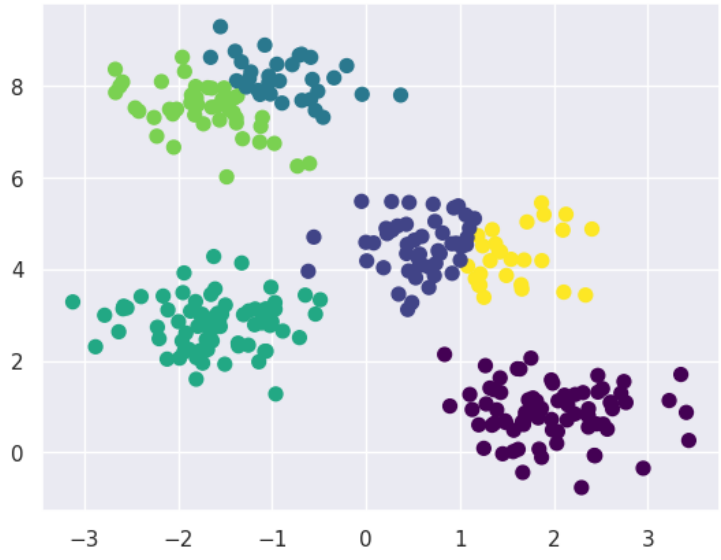

```
centers,labels=find_clusters(X,4,rseed=0)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis')
```

OUTPUT:




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

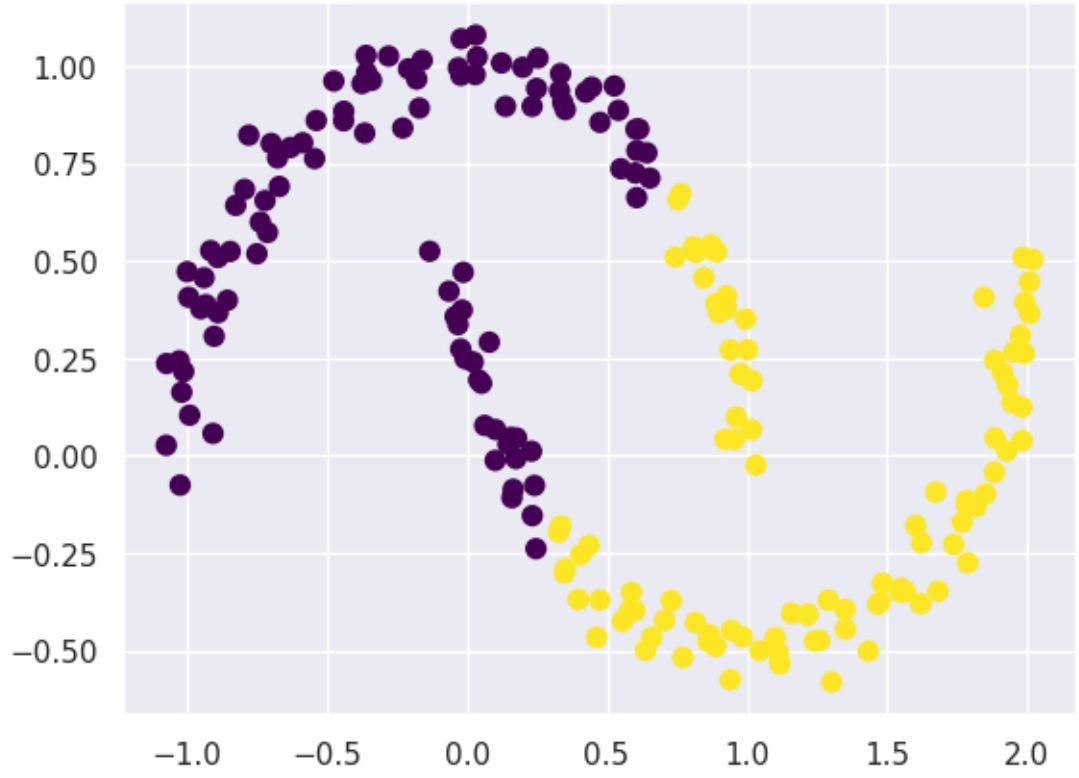
OUTPUT:



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

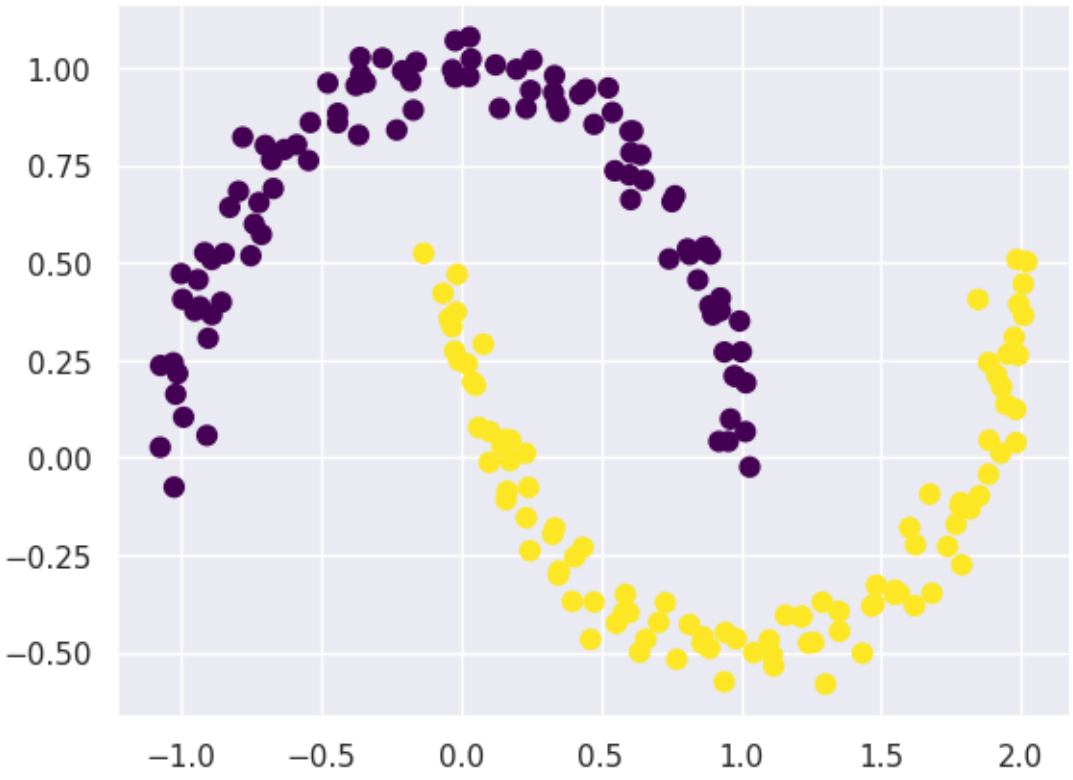
```
labels=KMeans(2,random_state=0,n_init=10).fit_predict(X)
plt.scatter(X[:,0],X[:,1],c=labels,s=50,cmap='viridis');
```

OUTPUT:




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

OUTPUT:


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

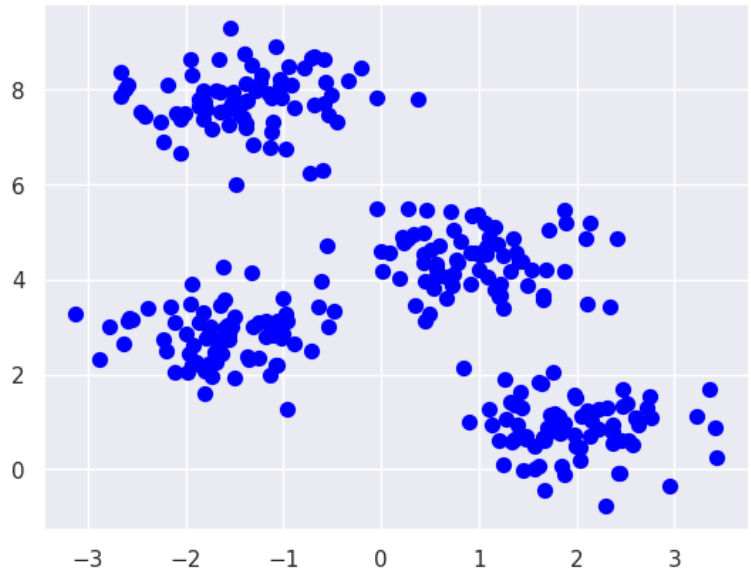
Lab-8/2203A52201/sec-AB:

Lab 08:Implementation of Gaussian Mixture Model using Synthetic Dataset.

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

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

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

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

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

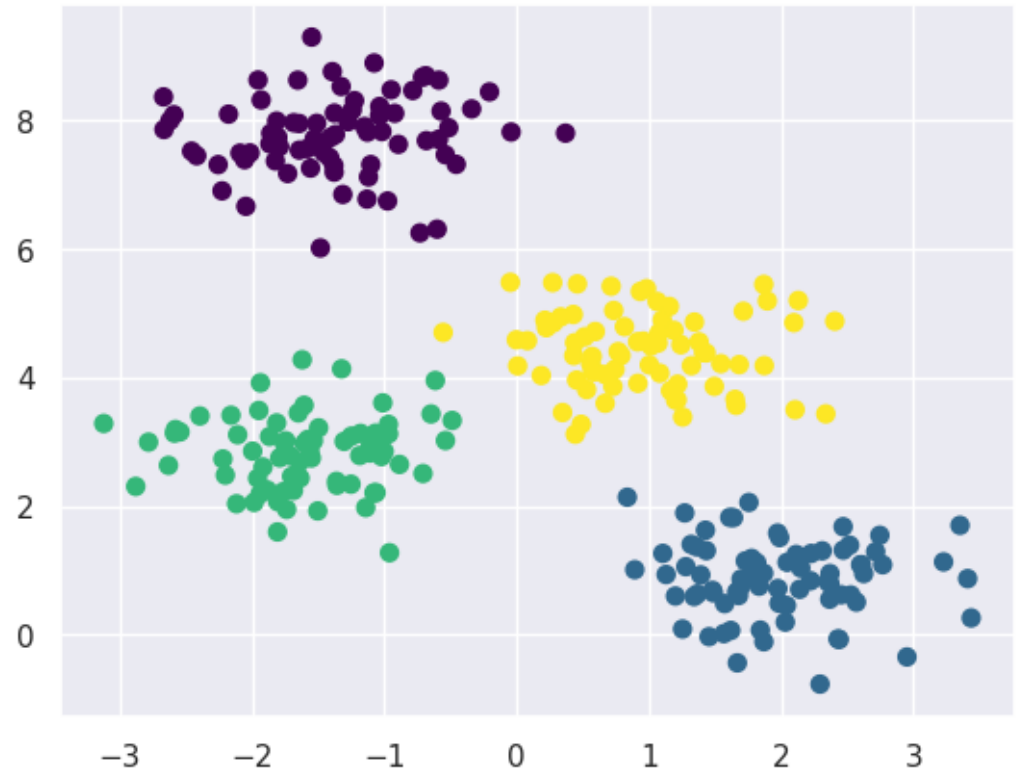
OUTPUT:




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

```
from sklearn.mixture import GaussianMixture

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

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

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

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

OUTPUT:



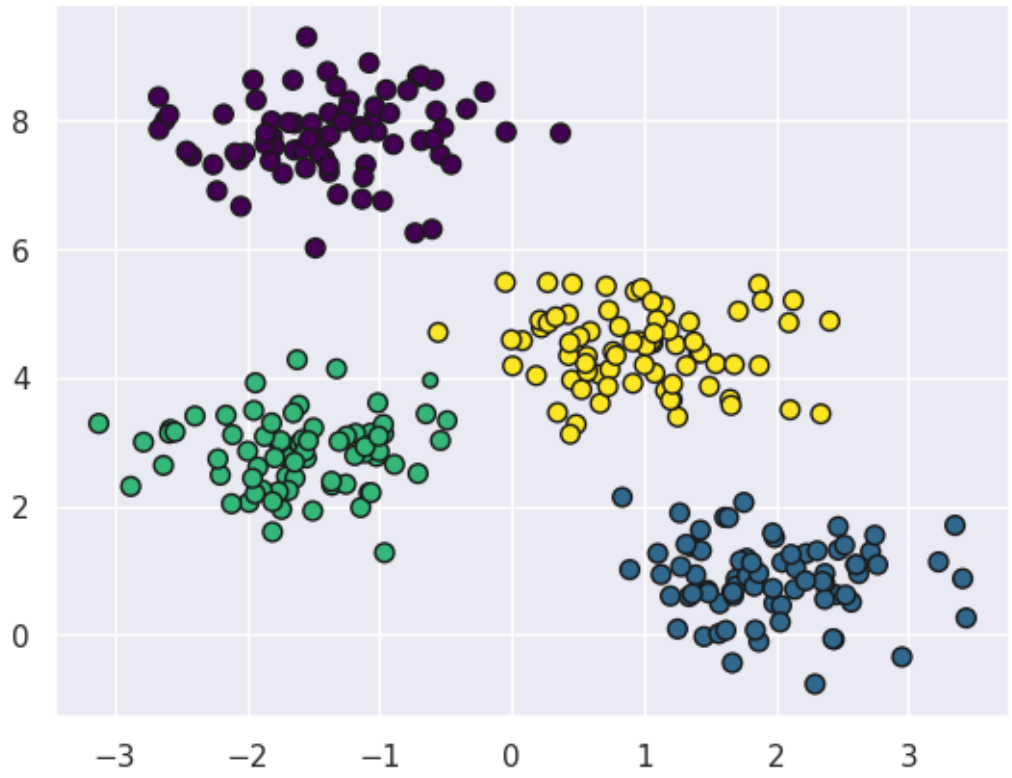
```
# Predict the probabilities of data points belonging to each cluster
probs = gmm.predict_proba(X)

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

OUTPUT:

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

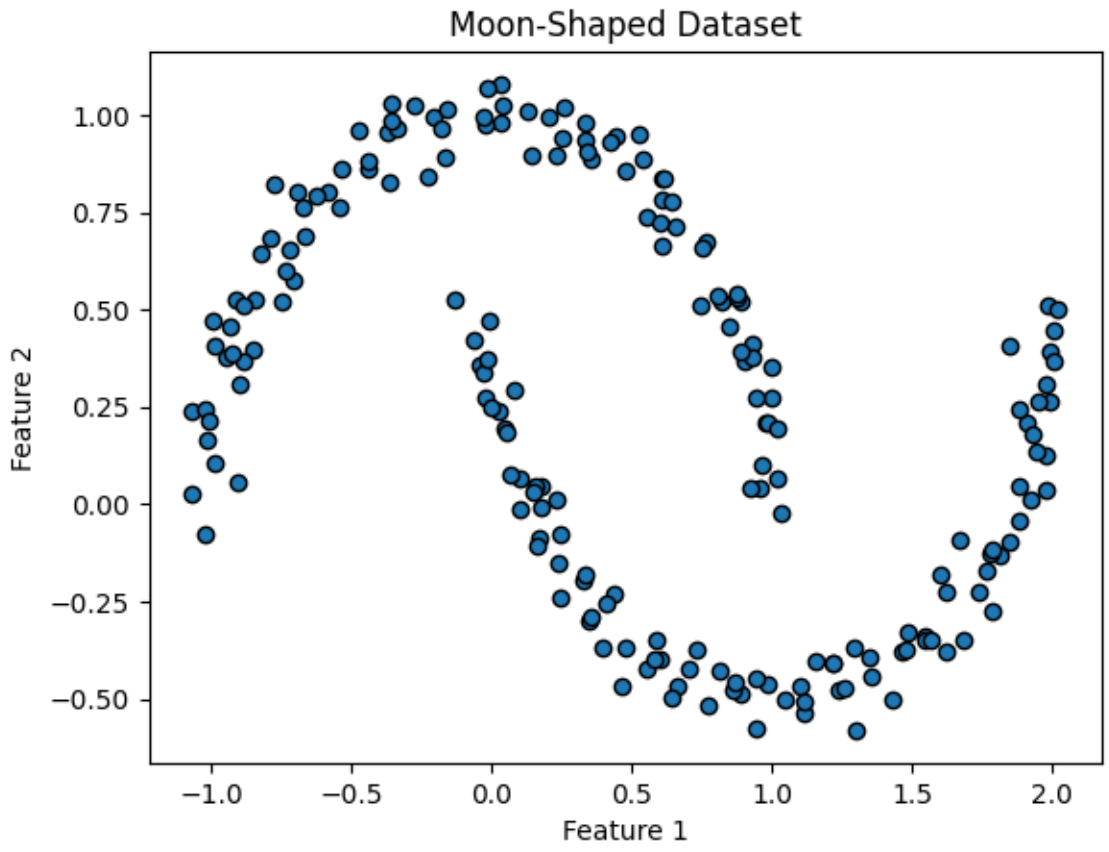
```
#print(probs.max(1))
size=probs.max(1)/0.02 #square emphasizes differences
plt.scatter(X[:,0],X[:,1],c=labels,edgecolor='k',cmap='viridis',s=size)
;
```

OUTPUT:



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

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

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

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

OUTPUT:

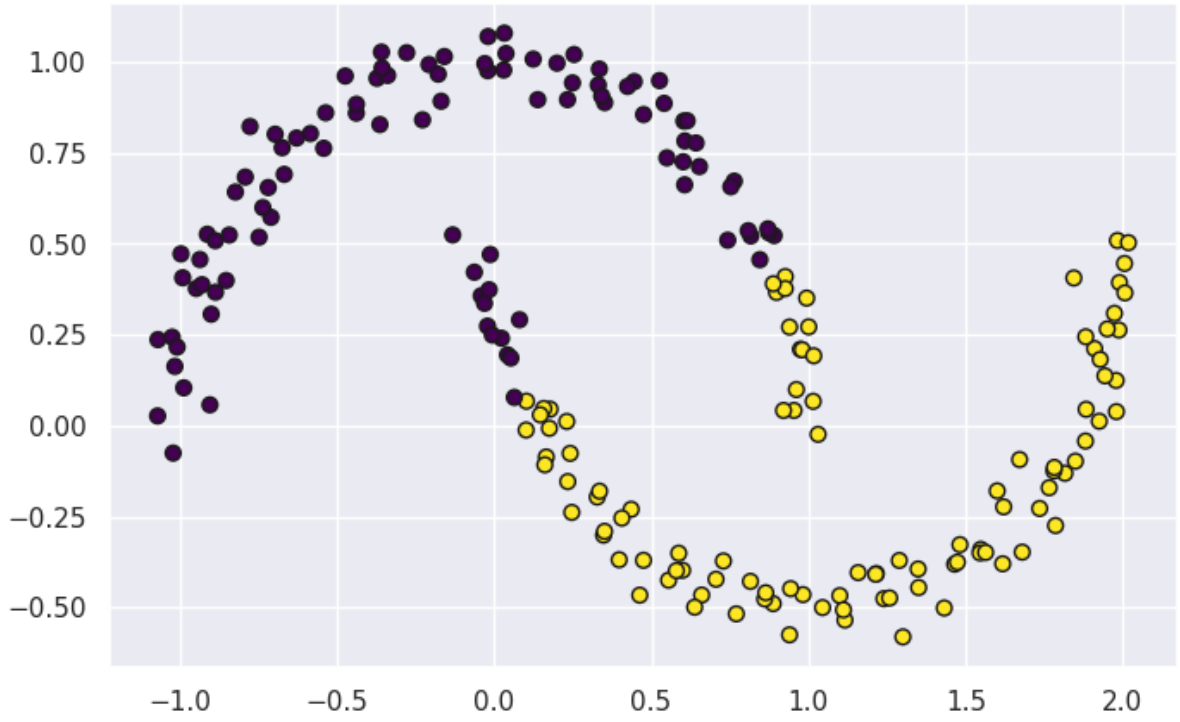


```
from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
 #Draw an ellipse with a given position and covariance"""
 ax = ax or plt.gca()
 # Convert covariance to principal axes
 if covariance.shape == (2, 2):
   U, s, Vt = np.linalg.svd(covariance)
   angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
   width, height = 2 * np.sqrt(s)
 else:
   angle = 0
   width, height = 2 * np.sqrt(covariance)
 # Draw the Ellipse
 for nsig in range(1, 4):
   ax.add_patch(Ellipse(position, nsig * width, nsig * height,angle,
**kwargs))
def plot_gmm(gmm, X, label=True, ax=None):
  ax = ax or plt.gca()
  labels = gmm.fit(X).predict(X)
  if label:
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=40,
cmap='viridis',zorder=2,edgecolor='k')
  else:
     ax.scatter(X[:, 0], X[:, 1], s=40,
zorder=2,cmap='viridis',edgecolor='k')
     ax.axis('equal')
     w_factor = 0.2 / gmm.weights_.max()
     for pos, covar, w in zip(gmm.means_, gmm.covariances_,
gmm.weights_):
         draw_ellipse(pos, covar, alpha=w * w_factor)
```

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

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

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

# Call the 'plot_gmm' function to visualize the GMM clustering
```

```
plot_gmm(gmm2, Xmoon)
```

OUTPUT:



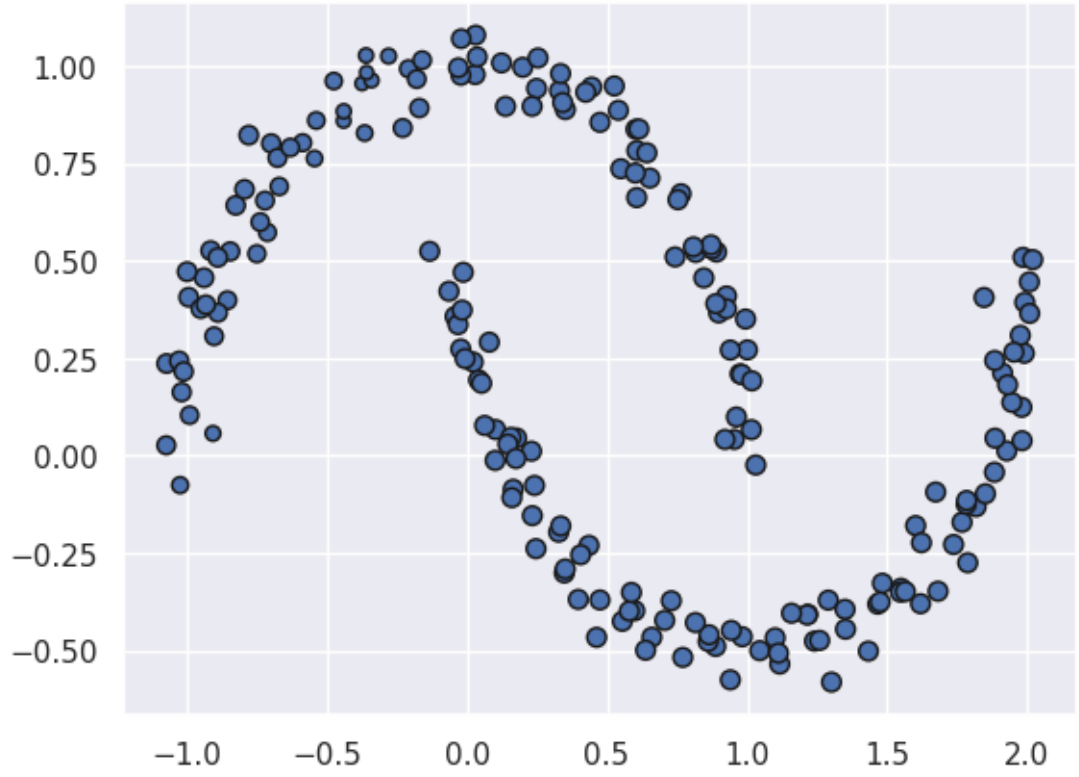

```
# Calculate the probabilities of each data point belonging to each
component
probs=gmm.predict_proba(Xmoon)

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

OUTPUT:

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

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

OUTPUT:



Lab – 9/2203A52201/sec-AB:

Lab 09:Implementation of Support Vector Machine Classification using Breast Cancer Dataset

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

from sklearn.datasets import load_breast_cancer

# Load the breast cancer dataset
cancer=load_breast_cancer()
#list of available keys.
cancer.keys()
```

OUTPUT:

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

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

OUTPUT:

```
.. _breast_cancer_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 569

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

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

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

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

SUMMARY STATISTICS

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

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

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

:Donor: Nick Street

:Date: November, 1995

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

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

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

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

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

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

.. topic:: References

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

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

OUTPUT:

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

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

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

OUTPUT:

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

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

OUTPUT:

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

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

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

OUTPUT:

0 (ZERO)

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

OUTPUT:

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

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

OUTPUT:

```
357
```

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

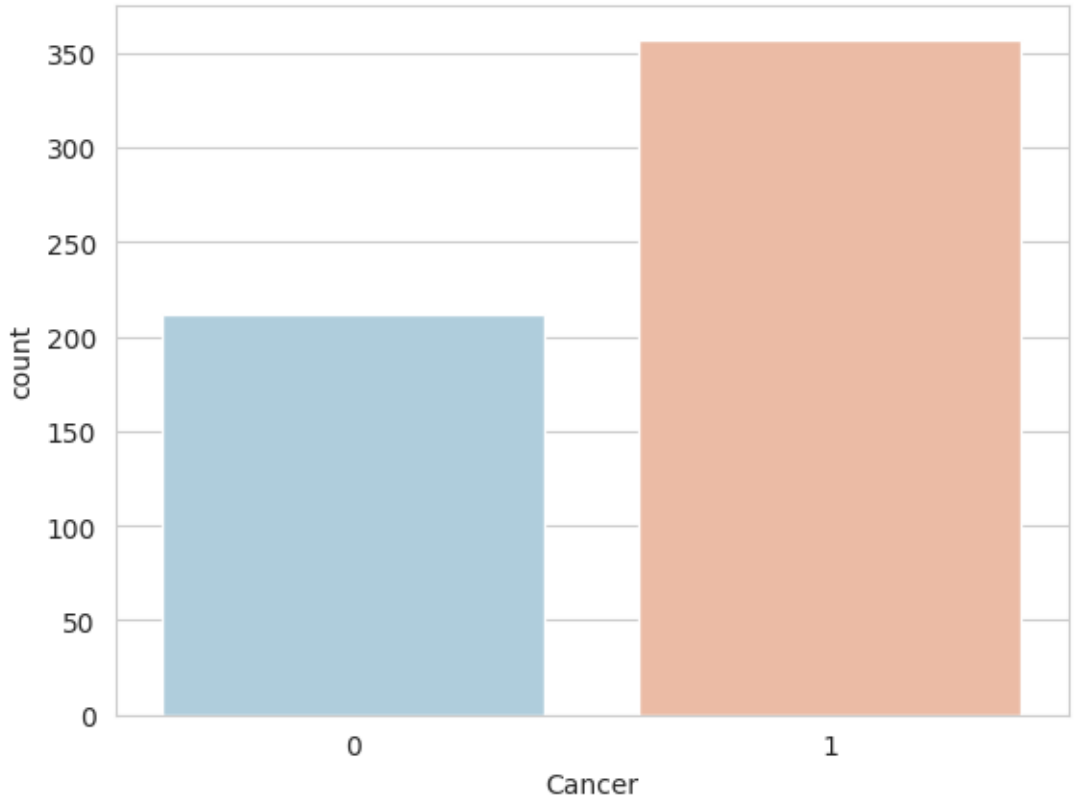
OUTPUT:

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

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

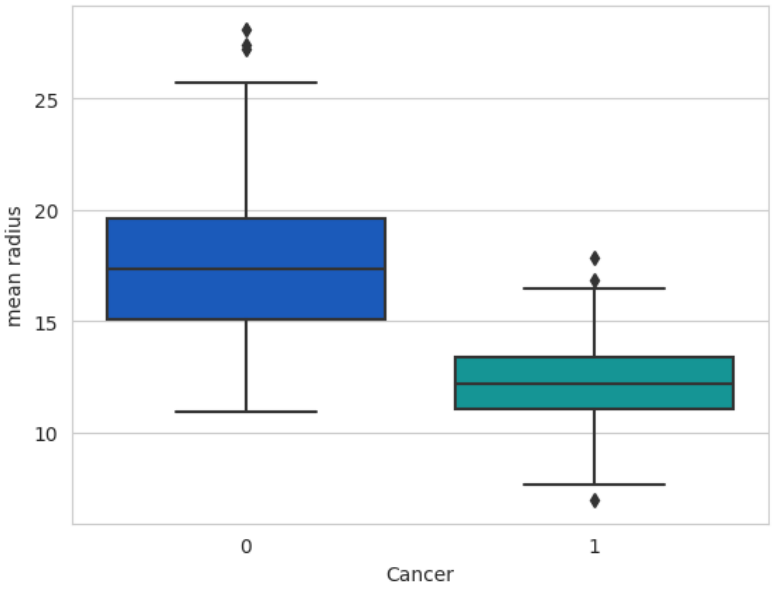
```
#countplot
sns.set_style('whitegrid')
sns.countplot(x='Cancer',data=df,palette='RdBu_r')
```

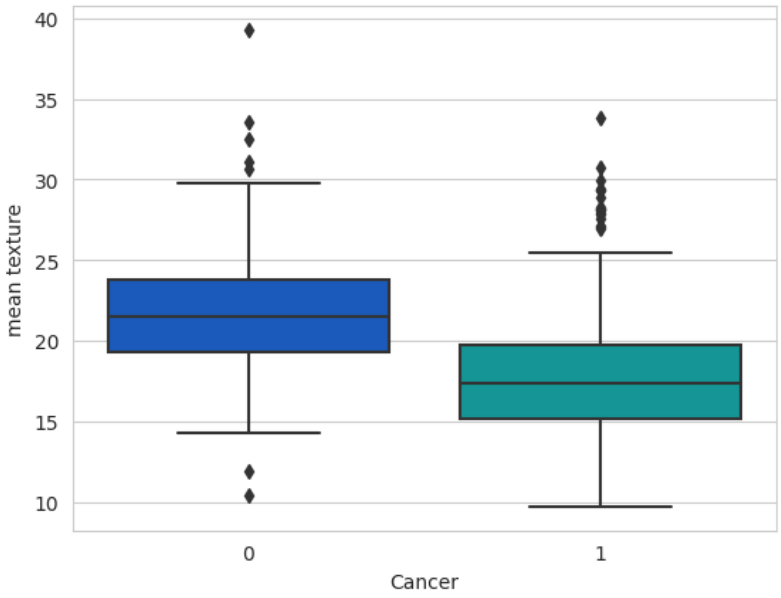
OUTPUT:

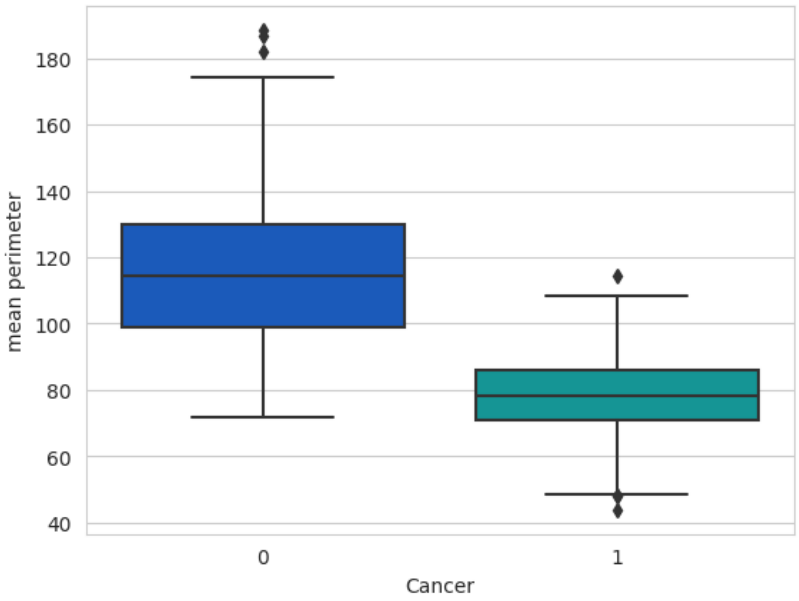


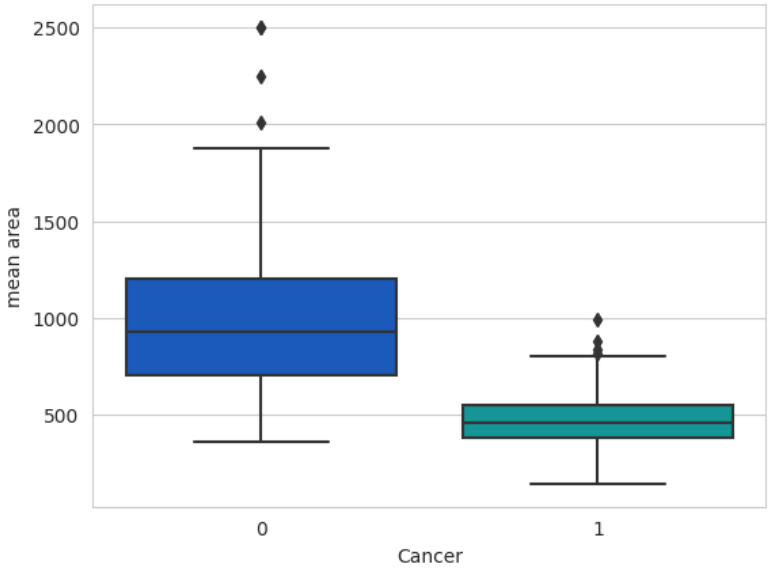

```
# List of column names (features) to create boxplots for
l = list(df.columns[0:10])

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

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

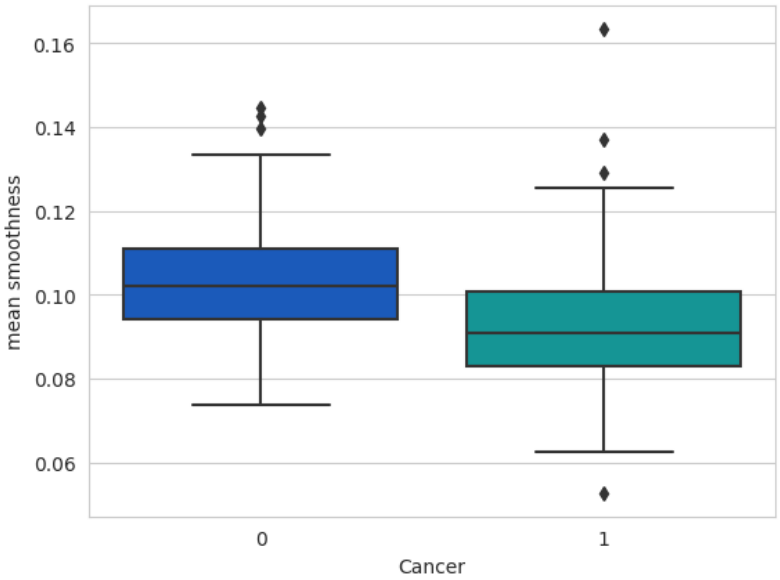
OUTPUT:

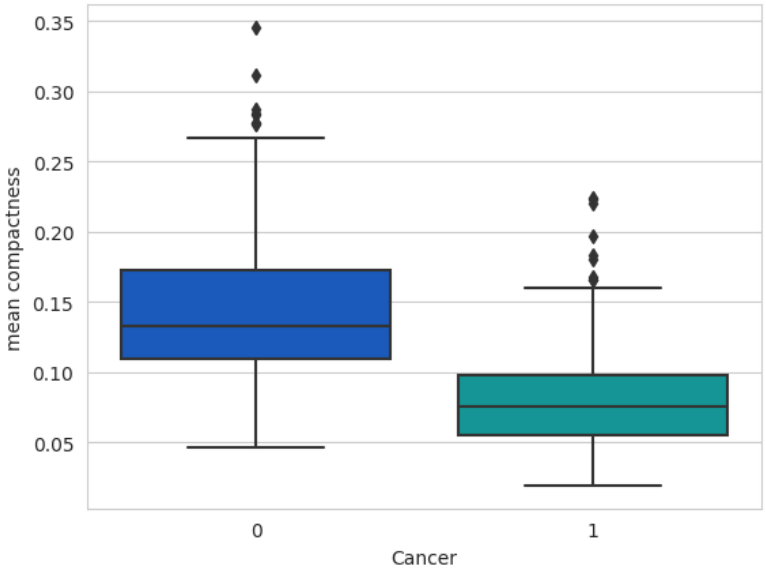

mean radius
25
20
15
10
0
1
Cancer

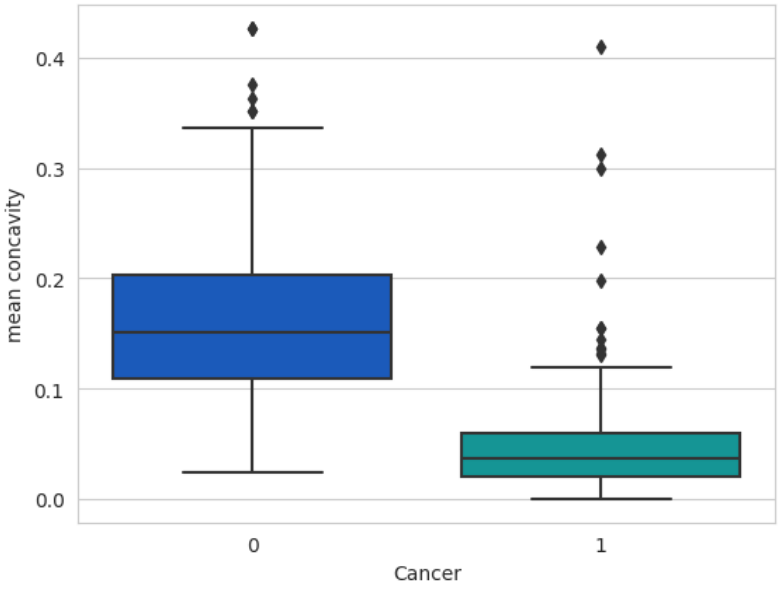

mean texture
40
35
30
25
20
15
10
0
1
Cancer

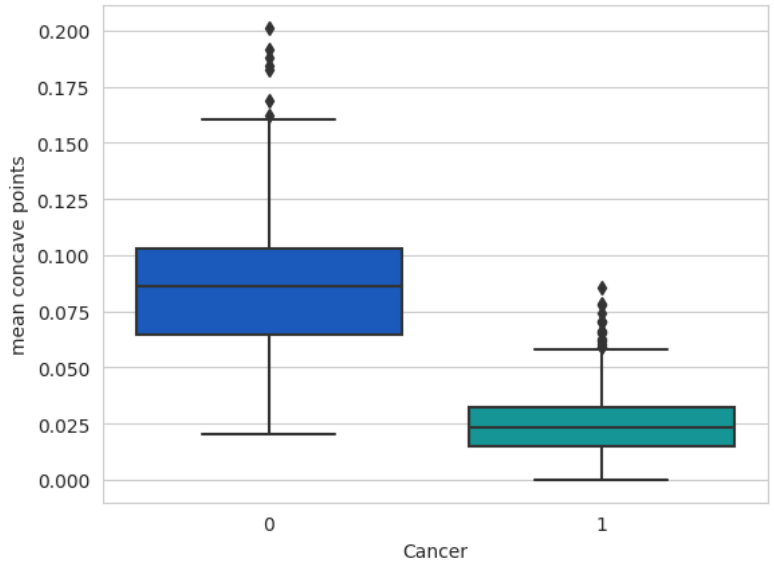

mean perimeter
180
160
140
120
100
80
60
40
0
1
Cancer

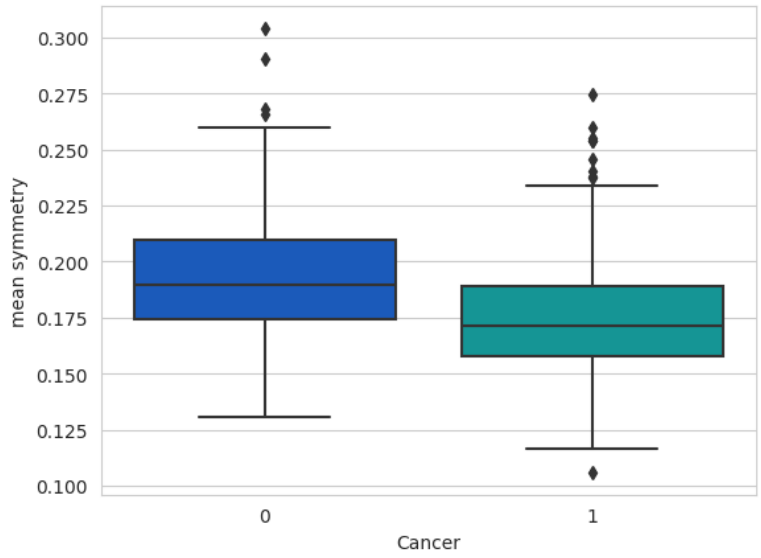

mean area
2500
2000
1500
1000
500
0
1
Cancer


0.16
0.14
0.12
0.10
0.08
0.06
mean smoothness
0
1
Cancer


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

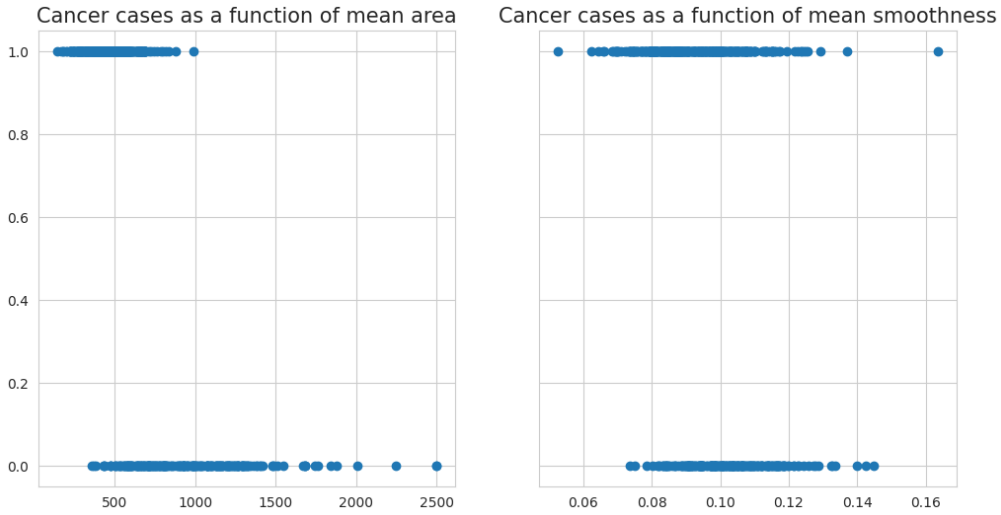

0.4
0.3
0.2
0.1
0.0
mean concavity
0
1
Cancer






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

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

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

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

OUTPUT:




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

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

OUTPUT:

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

5 rows × 30 columns

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

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

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

OUTPUT:

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

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

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

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

OUTPUT:

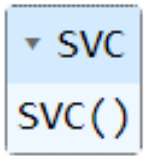
```
178    1
421    1
57     0
514    0
548    1
Name: Cancer, dtype: int64
```

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

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

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

OUTPUT:




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

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

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

OUTPUT:

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

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

OUTPUT:

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

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

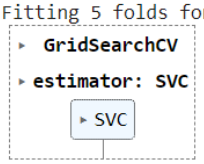
```
from sklearn.model_selection import GridSearchCV
```

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

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

OUTPUT:

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




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

OUTPUT:

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

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

OUTPUT:

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

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

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

OUTPUT:

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

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

OUTPUT:

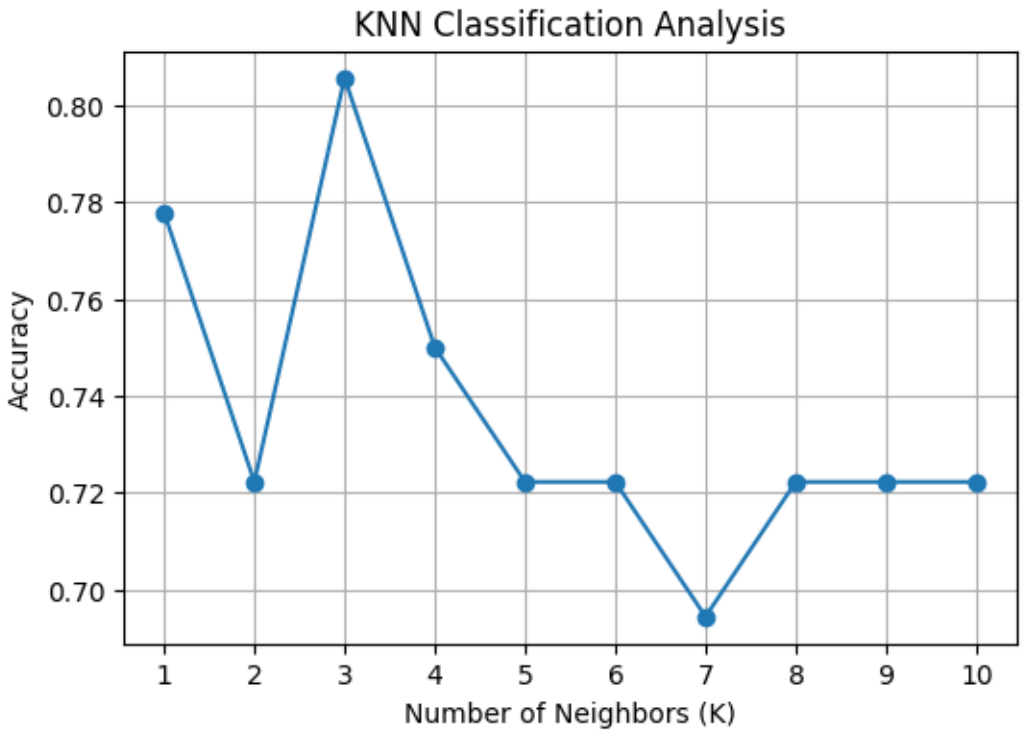
```
              precision    recall  f1-score   support

           0       0.94      0.89      0.91        66
           1       0.94      0.96      0.95       105

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

Lab-10/2203A52201/sec-AB

Lab 10: Non Parametric Algorithm (KNN) for classification, regression, and analysis of different neighbors

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Wine dataset for classification
wine = load_wine()
X_class, y_class = wine.data, wine.target
X_train, X_test, y_train, y_test = train_test_split(X_class, y_class,
test_size=0.2, random_state=42)

# Classification using KNN
k_values = range(1, 11)
classification_scores = []

for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    classification_scores.append(accuracy)

# Analysis: Plot classification performance
plt.figure(figsize=(6, 4))
plt.plot(k_values, classification_scores, marker='o')
plt.title('KNN Classification Analysis')
plt.xlabel ('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# Cross-validation for optimal K in classification
best_k_classifier = k_values[np.argmax(classification_scores)]
print(f'Best K for Classification: {best_k_classifier}')
```

OUTPUT:




```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Load the Wine dataset (assuming you have it as a DataFrame)
# X = Features, y = Target variable

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Choose an appropriate 'k' value (you can experiment with different
values)
k = 5

# Create and fit the KNN regression model
knn = KNeighborsRegressor(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)
```

```
# Evaluate the model using Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

OUTPUT:

```
Mean Squared Error: 123.456789
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Load the Wine dataset
wine_data = pd.read_csv("wine.csv")  # Replace "wine.csv" with the
actual file path

# Split the data into features (X) and target variable (y) for
classification and regression
X = wine_data.iloc[:, 1:]
y_classification = wine_data.iloc[:, 0]  # Wine class as target
variable
y_regression = wine_data["wine_quality"]  # Wine quality as target
variable

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and test sets for classification and
regression
X_train_class, X_test_class, y_train_class, y_test_class =
train_test_split(X, y_classification, test_size=0.2, random_state=42)
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X,
y_regression, test_size=0.2, random_state=42)

# Classification with KNN
k_class = 5  # Choose an appropriate 'k' value
knn_classifier = KNeighborsClassifier(n_neighbors=k_class)
```

```
knn_classifier.fit(X_train_class, y_train_class)
y_pred_class = knn_classifier.predict(X_test_class)

# Classification Analysis
accuracy_class = accuracy_score(y_test_class, y_pred_class)
classification_report_class = classification_report(y_test_class,
y_pred_class)
confusion_matrix_class = confusion_matrix(y_test_class, y_pred_class)

# Regression with KNN
k_reg = 5  # Choose an appropriate 'k' value for regression
knn_regressor = KNeighborsRegressor(n_neighbors=k_reg)
knn_regressor.fit(X_train_reg, y_train_reg)
y_pred_reg = knn_regressor.predict(X_test_reg)

# Regression Analysis
mse_reg = mean_squared_error(y_test_reg, y_pred_reg)

# Linear Regression for Regression
linear_reg = LinearRegression()
linear_reg.fit(X_train_reg, y_train_reg)
y_pred_linear_reg = linear_reg.predict(X_test_reg)
mse_linear_reg = mean_squared_error(y_test_reg, y_pred_linear_reg)

# Random Forest for Classification
random_forest_classifier = RandomForestClassifier(random_state=42)
random_forest_classifier.fit(X_train_class, y_train_class)
y_pred_rf_class = random_forest_classifier.predict(X_test_class)
accuracy_rf_class = accuracy_score(y_test_class, y_pred_rf_class)

# Support Vector Machine for Classification
svm_classifier = SVC(random_state=42)
svm_classifier.fit(X_train_class, y_train_class)
y_pred_svm_class = svm_classifier.predict(X_test_class)
accuracy_svm_class = accuracy_score(y_test_class, y_pred_svm_class)

# Analysis
# You can further analyze feature importance, model tuning, and
visualization here.

# Feature Importance
feature_importance = random_forest_classifier.feature_importances_

# Model Evaluation
linear_reg_cv_scores = cross_val_score(linear_reg, X, y_regression,
cv=5, scoring='neg_mean_squared_error')
knn_regressor_cv_scores = cross_val_score(knn_regressor, X,
y_regression, cv=5, scoring='neg_mean_squared_error')
```

```
# Visualization
# Create visualizations such as histograms, scatter plots, and feature
importance plots to gain insights.

# Example of a scatter plot for regression
plt.scatter(y_test_reg, y_pred_reg)
plt.xlabel("Actual Wine Quality")
plt.ylabel("Predicted Wine Quality (KNN Regression)")
plt.title("Actual vs. Predicted Wine Quality (KNN Regression)")
plt.show()
```

OUTPUT: