PRACTICAL JOURNAL IN

APPLIED ARTIFICIAL INTELLIGENCE

MACHINE LEARNING

SUBMITTED BY:

SHUBHAM LAXMAN PATTADE

ROLL NO: 2414568

IN PARTIAL FULLFILMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY PART – II SEMESTER III

ACADEMIC YEAR
2024-2025

PARLE TILAK VIDYALAYA ASSOCIATION'S
MULUND COLLEGE OF COMMERCE(AUTONOMOUS)
(AFFILIATED TO UNIVERSITY OF MUMBAI)
NAAC RE-ACCREDITED A GRADE – III CYCLE
MULUND WEST, MUMBAI 400080
MAHARASHTRA, INDIA
2024-25

Parle Tilak Vidyalaya Association's

MULUND COLLEGE OF COMMERCE (AUTONOMOUS)

(Affiliated to University of Mumbai)

MULUND WEST, MUMBAI 400080

MAHARASHTRA, INDIA

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that **Shubham L Pattade** of **M.Sc. I.T. Part II** Roll No **2414568** has successfully completed the practical work in **Applied Artificial Intelligence** in partial fulfilment of the requirements for the Semester III of **M.Sc. I.T. Part II** during the academic year **2024-25**.

Teacher In-charge and Coordinator

Examiner

Date:

College Seal

Parle Tilak Vidyalaya Association's

MULUND COLLEGE OF COMMERCE (AUTONOMOUS)

(Affiliated to University of Mumbai)

MULUND WEST, MUMBAI 400080

MAHARASHTRA, INDIA

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that **Shubham L Pattade** of **M.Sc. I.T. Part II** Roll No **2414568** has successfully completed the practical work in **Machine Learning** in partial fulfilment of the requirements for the Semester III of **M.Sc. I.T. Part II** during the academic year **2024-25**.

Teacher In-charge and Coordinator

Examiner

Date:

College Seal

Table of Contents

Practical 1: Design a bot using AIML

Std-startup.xml

```
<aiml version="1.0.1" encoding="UTF-8">
    <!-- std-startup.xml -->
    <!-- Category is an atomic AIML unit -->
    <category>
        <!-- Pattern to match in user input -->
        <!-- If user enters "LOAD AIML B" -->
        <pattern>LOAD AIML B</pattern>
        <!-- Template is the response to the pattern -->
        <!-- This learn an aiml file -->
        <template>
            <learn>basic_chat.aiml</learn>
            <!-- You can add more aiml files here -->
            <!--<learn>more_aiml.aiml</learn>-->
        </template>
    </category>
</aiml>
```

Basic_chat.aiml

```
<aiml version="1.0.1" encoding="UTF-8">
    <!-- basic_chat.aiml -->
    <category>
        <pattern>HELLO</pattern>
        <template>
            Well, hello!
        </template>
    </category>
    <category>
        <pattern>WHAT ARE YOU</pattern>
        <template> I'm a bot, silly! </template>
    </category>
    <category>
```

```
<pattern>MY NAME IS *</pattern>
        <template>
            <set name="username">
                <star />
            </set> is the nice name. </template>
    </category>
    <category>
        <pattern>I LIKE *</pattern>
        <template>
            <set name="liking">
                <star />
            </set> is also my favourite. </template>
    </category>
    <category>
        <pattern>MY DOG NAME IS *</pattern>
        <template> THAT IS INTERESTING THAT YOU HAVE A DOG NAMED <set name="dog">
                <star />
            </set> . </template>
    </category>
    <category>
        <pattern>BYE</pattern>
        <template> Bye!!! <get name="username" /> Thanks for talking with me. </template>
    </category>
</aiml>
```

Chatbot.py

```
import aiml # Create the kernel and learn AIML files
kernel = aiml.Kernel()
kernel.learn("std-startup.xml")
kernel.respond("load aiml b") # Press CTRL-C to break this loop
while True:
    message = input("Enter your message to the bot: ")
    if message == "quit":
        break
```

else:

    bot_response = kernel.respond(message)

    print(bot_response)

Output:

```
================ RESTART: E:/MSc/Sem3/AAI Prac/Pranay/chatbot.py ===========
Loading std-startup.xml...done (0.23 seconds)
Loading basic_chat.aiml...done (0.00 seconds)
Enter your message to the bot: hello
Well, hello!
Enter your message to the bot: what are you
I'm a bot, silly!
Enter your message to the bot: quit
```

Practical 2: Design an Expert system using AIML

Std_startup.xml

```
<aiml version="1.0.1" encoding="UTF-8">
    <!-- std-startup.xml -->
    <!-- Category is an atomic AIML unit -->
    <category>
        <!-- Pattern to match in user input -->
        <!-- If user enters "LOAD AIML B" -->
        <pattern>LOAD AIML B</pattern>
        <!-- Template is the response to the pattern -->
        <!-- This learn an aiml file -->
        <template>
            <learn>basic_chat.aiml</learn>
            <!-- You can add more aiml files here -->
            <!--<learn>more_aiml.aiml</learn>-->
        </template>
    </category>
</aiml>
```

Basic_chat.aiml

```
<aiml version="1.0.1" encoding="UTF-8">
    <!-- basic_chat.aiml -->
    <category>
    <pattern>HELLO</pattern>
    <template>
    WHAT WOULD YOU LIKE TO DISCUSS? : HEALTH, MOVIES
    </template>
    </category>
    <category>
    <pattern>MOVIES</pattern>
    <template>
        YES <set name = "topic">MOVIES</set>
        </template>
```

```
</category>
    <category>
    <pattern>HEALTH</pattern>
    <template> YES <set name = "topic">HEALTH</set> </template>
    </category>
    <topic name ="MOVIES">
    <category>
    <pattern>*</pattern>
    <template>
    DO YOU LIKE COMEDY MOVIES?
    </template>
    </category>
    <category> <pattern>YES</pattern>
    <template>
    I TOO LIKE COMEDY MOVIES
    </template>
    </category>
    <category>
    <pattern>NO</pattern>
    <template>
    BUT I LIKE COMEDY MOVIES
    </template>
    </category>
    </topic>
    <topic name ="HEALTH">
    <category>
    <pattern>*</pattern>
    <template>
    DO YOU HAVE FEVER?
    </template>
    </category>
    <category>
    <pattern>YES</pattern>
    <template>
```

```
PLEASE TAKE MEDICINES AND PROPER REST
</template></category>
<category>
<pattern>NO</pattern>
<template>
GO OUT FOR A WALK AND LISTEN MUSIC
</template>
</category>
</topic>
<category>
<pattern>NICE TALKING TO YOU</pattern>
<template>
SAME HERE...!!
</template>
</category>
</aiml>
```

Chatbot.py

```
import aiml
# Create the kernel and learn AIML files
kernel = aiml.Kernel()
kernel.learn("std-startup.xml")
kernel.respond("LOAD AIML B")
# Press CTRL-C to break this loop
while True:
    message = input("Enter your message to the bot: ")
    if message == "quit":
        break
    else:
        bot_response = kernel.respond(message)
        print(bot_response)
```


Output:

```
================== RESTART: E:/MSC/Sem3/AAI Prac/Pranay/P2.py ==================
Loading std-startup.xml...done (0.03 seconds)
Loading basic_chat.aiml...done (0.00 seconds)
Enter your message to the bot: hello
WHAT WOULD YOU LIKE TO DISCUSS? : HEALTH, MOVIES
Enter your message to the bot: health
YES HEALTH
Enter your message to the bot: i am feeling tired
DO YOU HAVE FEVER?
Enter your message to the bot: no
GO OUT FOR A WALK AND LISTEN MUSIC
Enter your message to the bot: Movies
YES MOVIES
Enter your message to the bot: I love movies
DO YOU LIKE COMEDY MOVIES?
Enter your message to the bot: yes
I TOO LIKE COMEDY MOVIES
Enter your message to the bot: Thanks
DO YOU LIKE COMEDY MOVIES?
Enter your message to the bot: Quit
DO YOU LIKE COMEDY MOVIES?
Enter your message to the bot: yes
I TOO LIKE COMEDY MOVIES
Enter your message to the bot: Yes
I TOO LIKE COMEDY MOVIES
Enter your message to the bot: quit
```

Practical 3: Implement Bayes Theorem using Python

```
# calculate the probability of cancer patient and diagnostic test
# calculate P(A|B) given P(A), P(B|A), P(B|not A)
def bayes_theorem(p_a, p_b_given_a, p_b_given_not_a):
    # calculate P(not A)
    not_a = 1 - p_a
    # calculate P(B)
    p_b = p_b_given_a * p_a + p_b_given_not_a * not_a
    # calculate P(A|B)
    p_a_given_b = (p_b_given_a * p_a) / p_b
    return p_a_given_b
# P(A)
p_a = 0.0002
# P(B|A)
p_b_given_a = 0.85
# P(B|not A)
p_b_given_not_a = 0.05
# calculate P(A|B)
result = bayes_theorem(p_a, p_b_given_a, p_b_given_not_a)
# summarize
print('P(A|B) = %.3f%%' % (result * 100))
```

Output:

```
================= RESTART: E:\MSC\Sem3\AAI Prac\Pranay\P3.py =================
P(A|B) = 0.339%
```

Practical 4: Implement Conditional Probability and joint probability using Python.

```
import enum, random
class Kid(enum.Enum):
    BOY = 0
    GIRL = 1
def random_kid() -> Kid:
    return random.choice([Kid.BOY, Kid.GIRL])
both_girls = 0
older_girl = 0
either_girl = 0
random.seed(0)
for _ in range(10000):
    younger = random_kid()
    older = random_kid()
    if older == Kid.GIRL:
        older_girl += 1
    if older == Kid.GIRL and younger == Kid.GIRL:
        both_girls += 1
    if older == Kid.GIRL or younger == Kid.GIRL:
        either_girl += 1
print("older girl: ", older_girl)
print("both girl: ", both_girls)
print("either girl: ", either_girl)
print("P(both | older):", both_girls / older_girl)
print("P(both | either):", both_girls / either_girl)
```

Output:

```
================== RESTART: E:\MSC\Sem3\AAI Prac\Pranay\P4.py ==================
older girl:  4937
both girl:  2472
either girl:  7464
P(both | older): 0.5007089325501317
P(both | either): 0.3311897106109325
```

Practical 5: Write a program to implement Rule based system. (Prolog)

```
go:-
hypothesis(Disease),
write('I believe that the patient have '),
write(Disease),
nl,
write('TAKE CARE '),
undo.
/*Hypothesis that should be tested*/
hypothesis(cold) :- cold, !.
hypothesis(flu) :- flu, !.
hypothesis(typhoid) :- typhoid, !.
hypothesis(measles) :- measles, !.
hypothesis(malaria) :- malaria, !.
hypothesis(unknown). /* no diagnosis*/
/*Hypothesis Identification Rules*/
cold :-
verify(headache),
verify(runny_nose),
verify(sneezing),
verify(sore_throat),
write('Advices and Sugestions:'),
nl,
write('1: Tylenol/tab'),
nl,
write('2: panadol/tab'),
nl,
write('3: Nasal spray'),
nl,
write('Please wear warm cloths Because'),
nl.
```

```
flu :-

verify(fever),
verify(headache),
verify(chills),
verify(body_ache),
write('Advices and Sugestions:'),
nl,
write('1: Tamiflu/tab'),
nl,
write('2: panadol/tab'),
nl,
write('3: Zanamivir/tab'),
nl,
write('Please take a warm bath and do salt gargling Because'),
nl.
typhoid :-
verify(headache),
verify(abdominal_pain),
verify(poor_appetite),
verify(fever),
write('Advices and Sugestions:'),
nl,
write('1: Chloramphenicol/tab'),
nl,
write('2: Amoxicillin/tab'),
nl,
write('3: Ciprofloxacin/tab'),
nl,
write('4: Azithromycin/tab'),
nl,
write('Please do complete bed rest and take soft Diet Because'),
nl.
measles :-
verify(fever),
```

```
verify(runny_nose),

verify(rash),
verify(conjunctivitis),
write('Advices and Sugestions:'),
nl,
write('1: Tylenol/tab'),
nl,
write('2: Aleve/tab'),
nl,
write('3: Advil/tab'),
nl,
write('4: Vitamin A'),
nl,
write('Please Get rest and use more liquid Because'),
nl.
malaria :-
verify(fever),
verify(sweating),
verify(headache),
verify(nausea),
verify(vomiting),
verify(diarrhea),
write('Advices and Sugestions:'),
nl,
write('1: Aralen/tab'),
nl,
write('2: Qualaquin/tab'),
nl,
write('3: Plaquenil/tab'),
nl,
write('4: Mefloquine'),
nl,
write('Please do not sleep in open air and cover your full skin Because'),
nl.
```

```
/* how to ask questions */

ask(Question) :-
write('Does the patient have following symptom:'),
write(Question),
write('? '),
read(Response),
nl,
( (Response == yes ; Response == y)
->
assert(yes(Question)) ;
assert(no(Question)), fail).
:- dynamic yes/1,no/1.
/*How to verify something */
verify(S) :-
(yes(S)
->
true ;
(no(S)
->
fail ;
ask(S))).
/* undo all yes/no assertions*/
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```

Output:

```
?- go.
Does the patient have following symptom:headache? y.

Does the patient have following symptom:runny_nose? |: y.

Does the patient have following symptom:sneezing? |: y.

Does the patient have following symptom:sore_throat? |: y.

Advices and Sugestions:
1: Tylenol/tab
2: panadol/tab
3: Nasal spray
Please wear warm cloths Because
I believe that the patient have cold
TAKE CARE
true.

?-
```

Practical 6: Design a Fuzzy based application using Python / R

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
from skfuzzy import control as ctrl
from mpl_toolkits.mplot3d import Axes3D # Required for 3D plotting
# New Antecedent/Consequent objects hold universe variables and membership
# functions
quality = ctrl.Antecedent(np.arange(0, 10, 0.1), 'quality')
service = ctrl.Antecedent(np.arange(0, 10, 0.1), 'service')
tip = ctrl.Consequent(np.arange(0, 25, 0.1), 'tip')


quality['poor'] = fuzz.zmf(quality.universe, 0,5)
quality['average'] = fuzz.gaussmf(quality.universe,5,1)
quality['good'] = fuzz.smf(quality.universe,5,10)


service['poor'] = fuzz.zmf(service.universe, 0,5)
service['average'] = fuzz.gaussmf(service.universe,5,1)
service['good'] = fuzz.smf(service.universe,5,10)


tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

quality['average'].view()
plt.title('Quality')
```



```
service['poor'].view()
```

```
plt.title('Service')

tip['medium'].view()
plt.title('Tip Medium')

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
rule1.view()
plt.title('Rule 1')
rule2.view()
plt.title('Rule 2')
rule3.view()
plt.title('Rule 3')
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8
tipping.compute()
plt.title('Result')
plt.show(block=True)
print(tipping.output['tip'])
tip.view(sim=tipping)
```

Output:

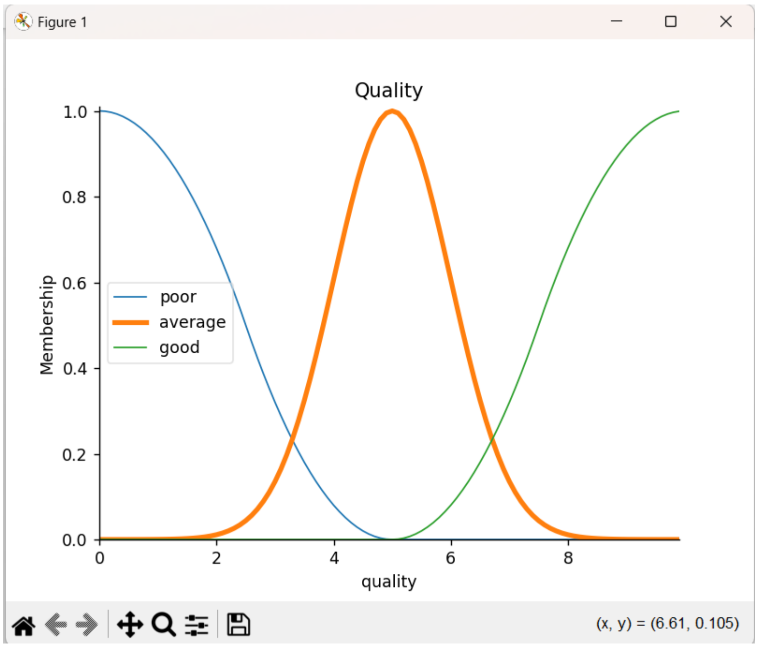


Figure 1
Quality
Membership
quality
poor
average
good
(x, y) = (6.61, 0.105)

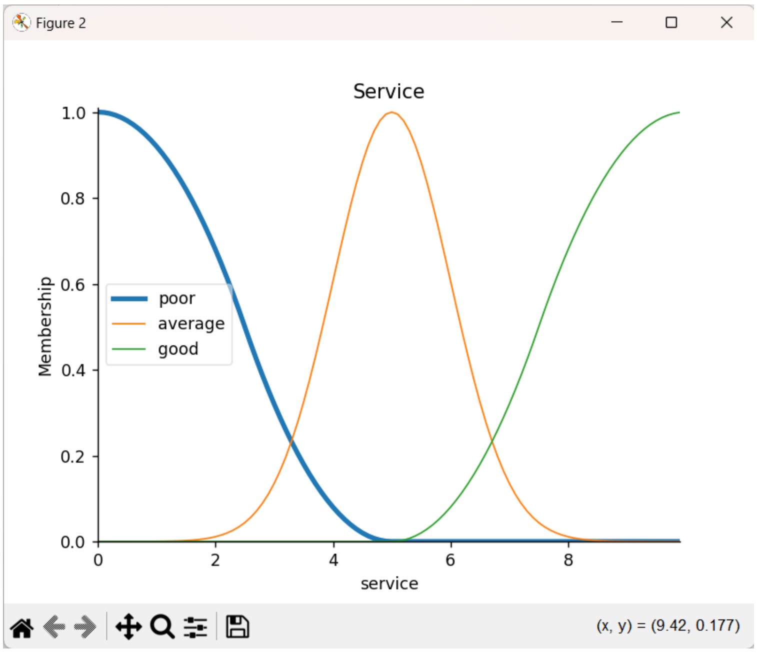


Figure 2
Service
Membership
service
poor
average
good
(x, y) = (9.42, 0.177)

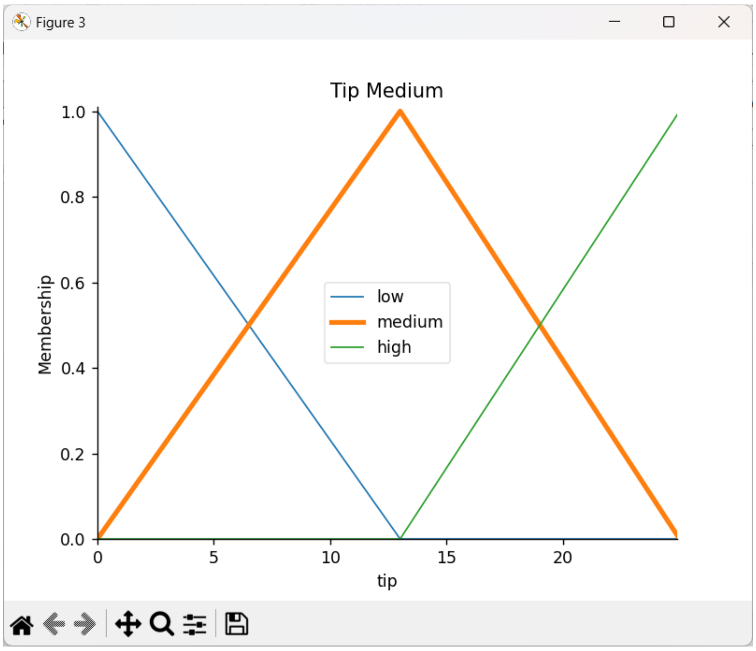


Figure 3
Tip Medium
1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
5
10
15
20
tip
low
medium
high

Practical 7A: Write an application to stimulate supervised learning model.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
iris=datasets.load_iris()
x = iris.data
y = iris.target
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
#To make predictions on our test data
y_pred=classifier.predict(x_test)
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:

```
================= RESTART: E:/MSC/Sem3/AAI Prac/Pranay/P7_A.py =================
sepal-length sepal-width petal-length petal-width
Squeezed text (150 lines).
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 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 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 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Confusion Matrix
[[19  0  0]
 [ 0 16  0]
 [ 0  1  9]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        16
           2       1.00      0.90      0.95        10

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

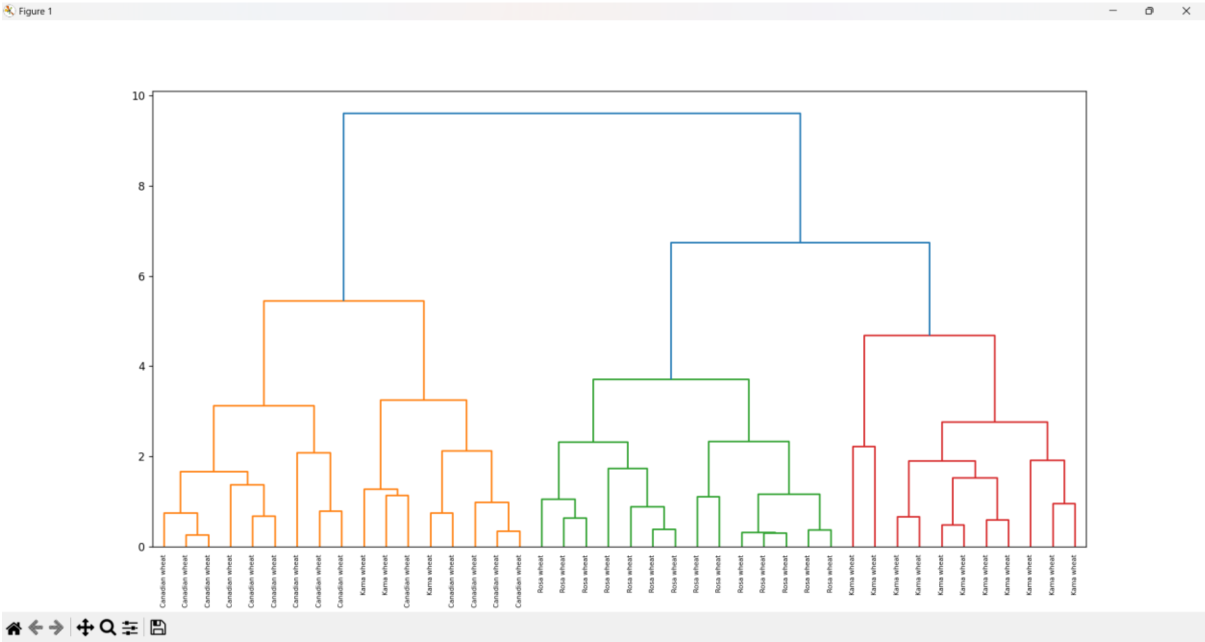
7B: Write an application to stimulate unsupervised learning model.

```
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt
import pandas as pd

# Reading the DataFrame
seeds_df = pd.read_csv("seeds-less-rows.csv")
# Remove the grain species from the DataFrame, save for later
varieties = list(seeds_df.pop('grain_variety'))
# Extract the measurements as a NumPy array
samples = seeds_df.values
"""
Perform hierarchical clustering on samples using the
linkage() function with the method='complete' keyword argument.
Assign the result to mergings.
"""
mergings = linkage(samples, method='complete')
"""
Plot a dendrogram using the dendrogram() function on mergings,
specifying the keyword arguments labels=varieties, leaf_rotation=90,
and leaf_font_size=6.
"""
dendrogram(mergings,
           labels=varieties,
           leaf_rotation=90,
           leaf_font_size=6,
          )
```



```
plt.show()
```

Output:

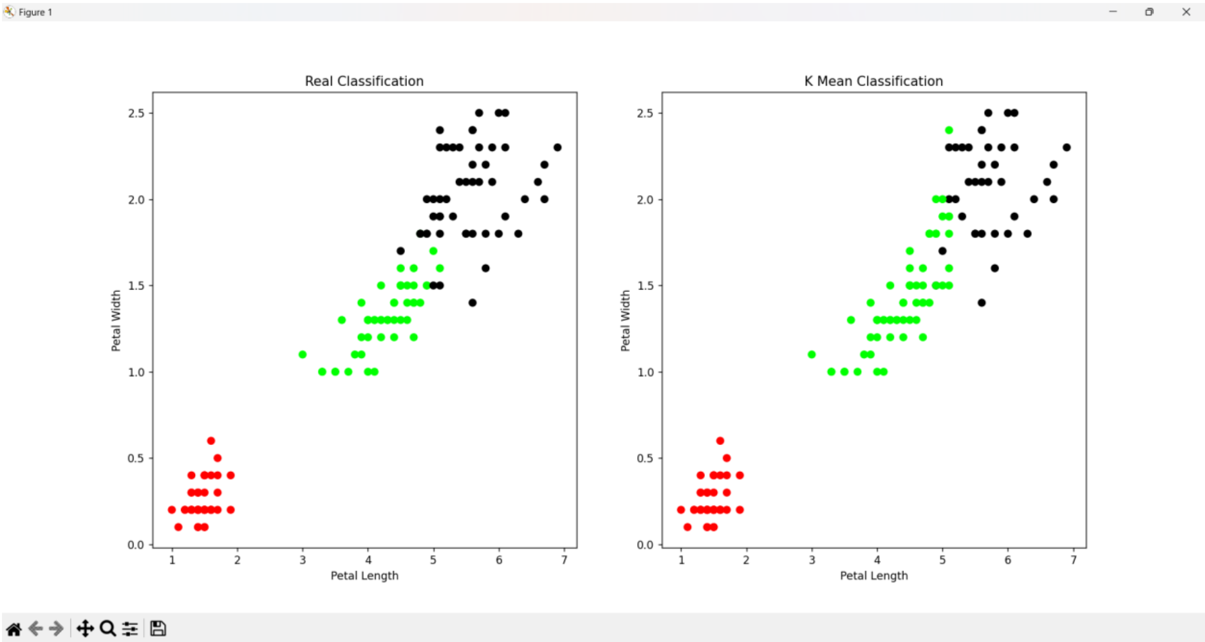


Practical 8: Write an application to implement clustering algorithm.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
model = KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

```
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))

print('The Confusion matrix of K-Mean: ',sm.confusion_matrix(y, model.labels_))
```

Output:



Practical 9: Write an application to implement support vector machine algorithm.

```
#Import scikit-learn dataset library
from sklearn import datasets
#Import svm model
from sklearn import svm
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

#Load dataset
cancer = datasets.load_breast_cancer()

# print the names of the 13 features
print("Features: ", cancer.feature_names)

# print the label type of cancer('malignant' 'benign')
print("Labels: ", cancer.target_names)

# print data(feature)shape
cancer.data.shape

# print the cancer data features (top 5 records)
print(cancer.data[0:5])

# print the cancer labels (0:malignant, 1:benign)
print(cancer.target)
```



```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
test_size=0.3,random_state=109) # 70% training and 30% test
```

```
#Create a svm Classifier

clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

Output:

```
================== RESTART: E:/MSC/Sem3/AAI Prac/Pranay/P9.py ==================
Features:  ['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']
Labels:  ['malignant' 'benign']
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
  2.364e-01 7.678e-02]]
```

```
[0 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
 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 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 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]
Accuracy: 0.9649122807017544
Precision: 0.9811320754716981
Recall: 0.9629629629629629
>>
```

Practical 10: Simulate artificial neural network model with both feedforward and backpropagation approach.

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # two inputs [sleep,study]

y = np.array(([92], [86], [89]), dtype=float) # one output [Expected % in Exams]

X = X / np.amax(X, axis=0) # maximum of X array longitudinally

y = y / 100


# Sigmoid Function

def sigmoid(x):

    return 1 / (1 + np.exp(-x))


# Derivative of Sigmoid Function

def derivatives_sigmoid(x):

    return x * (1 - x)


# Variable initialization

epoch = 5000 # Setting training iterations

lr = 0.1 # Setting learning rate

inputlayer_neurons = 2 # number of features in data set

hiddenlayer_neurons = 3 # number of hidden layers neurons

output_neurons = 1 # number of neurons at output layer


# weight and bias initialization

wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons)) # weight of the link from input node to hidden node

bh = np.random.uniform(size=(1, hiddenlayer_neurons)) # bias of the link from input node to hidden node

wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons)) # weight of the link from hidden node to output node

bout = np.random.uniform(size=(1, output_neurons)) # bias of the link from hidden node to output node
```

draws a random range of numbers uniformly of dim x*y

```
for i in range(epoch):
    # Forward Propogation
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)
    # Backpropagation
    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    # how much hidden layer weights contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) * lr
wh += X.T.dot(d_hiddenlayer) * lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n", output)
```

Output:

```
================== RESTART: E:/MSC/Sem3/AAI Prac/Pranay/P10.py =============
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.69044378]
 [0.67208465]
 [0.68799528]]
>>
```

Practical 1

1.1 Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.

```
import seaborn as sns

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

plt.rcParams["figure.figsize"] = [8,6]
sns.set(style="darkgrid")

df = sns.load_dataset('titanic')

print(df.head())
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN Southampton     no   True
```

```
print(df.isnull().sum())
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
```

```
alone             0
dtype: int64
```

```
df = df[["age", "embarked"]]
print(df.head())
```

```
    age embarked
0  22.0        S
1  38.0        C
2  26.0        S
3  35.0        S
4  35.0        S
```

```
df.loc[:, 'age'] = df.age.fillna(df.age.median())
df = df.dropna(subset=["embarked"])

print(df.head(20))
```

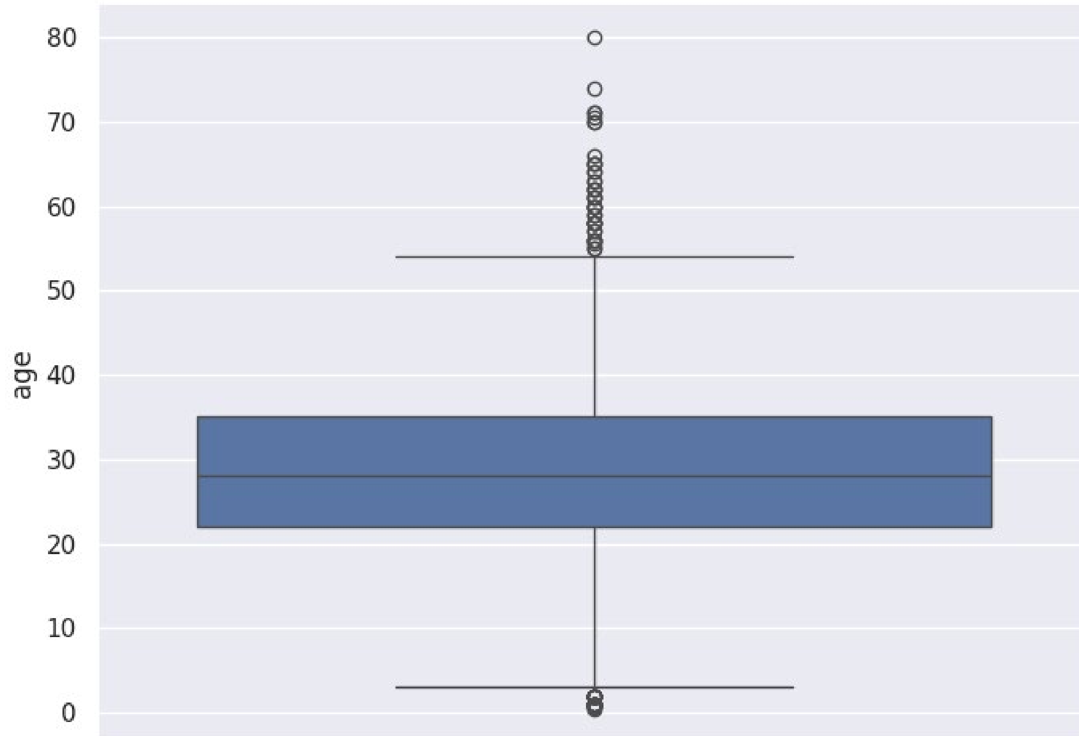
```
     age embarked
0   22.0        S
1   38.0        C
2   26.0        S
3   35.0        S
4   35.0        S
5   28.0        Q
6   54.0        S
7    2.0        S
8   27.0        S
9   14.0        C
10   4.0        S
11  58.0        S
12  20.0        S
13  39.0        S
14  14.0        S
15  55.0        S
16   2.0        Q
17  28.0        S
18  31.0        S
19  28.0        C
```

```
df.loc[:, 'embarked'] = df.embarked.str.upper()
print(df.embarked.unique())
```

```
['S' 'C' 'Q']
```

```
sns.boxplot(data=df.age)
```

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

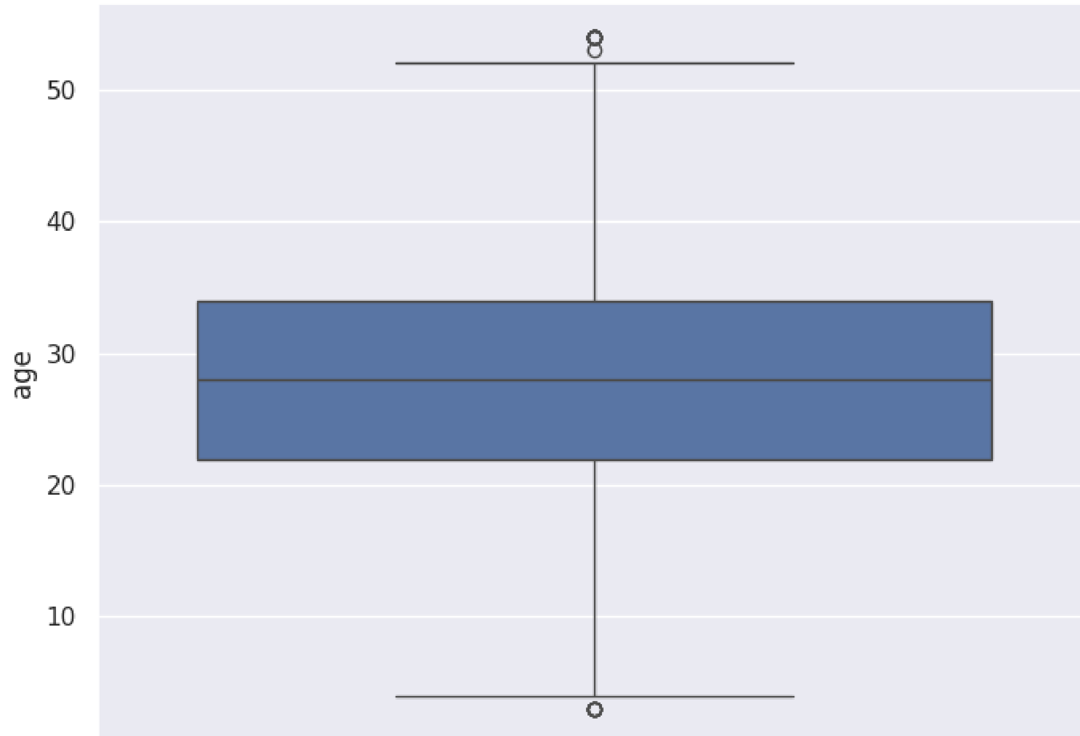



```
Q1 = df.age.quantile(0.25)
Q3 = df.age.quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df = df[(df.age >= lower_bound) & (df.age <= upper_bound)]

print(df.head())
```

```
    age embarked
0  22.0        S
1  38.0        C
2  26.0        S
3  35.0        S
4  35.0        S
```

```
sns.boxplot(data=df.age)
```

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


50
40
30
20
10
age

1.2 Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables

```
import pandas as pd import
numpy as np import seaborn as
sns

import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = [8,6]
sns.set(style="darkgrid")

df = sns.load_dataset('iris')
print(df.head())
```

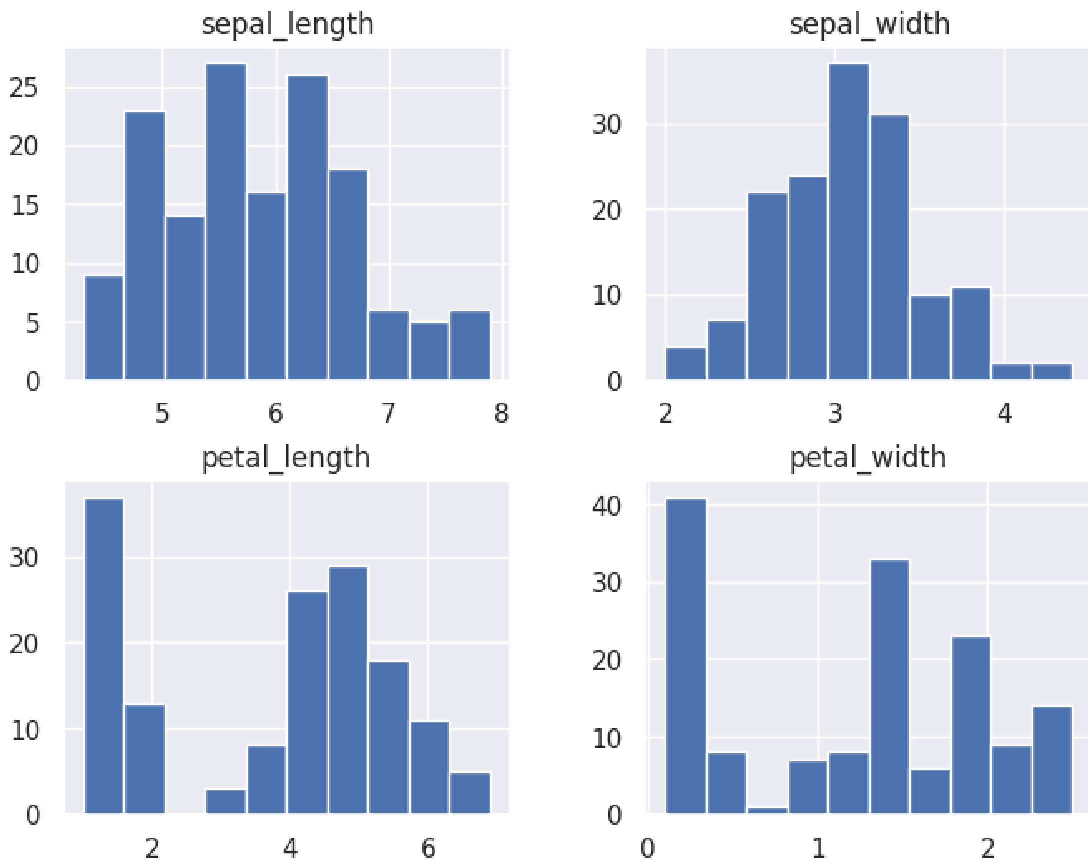
```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

```
summary_statistics = df.describe()
print(summary_statistics)
```

```
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

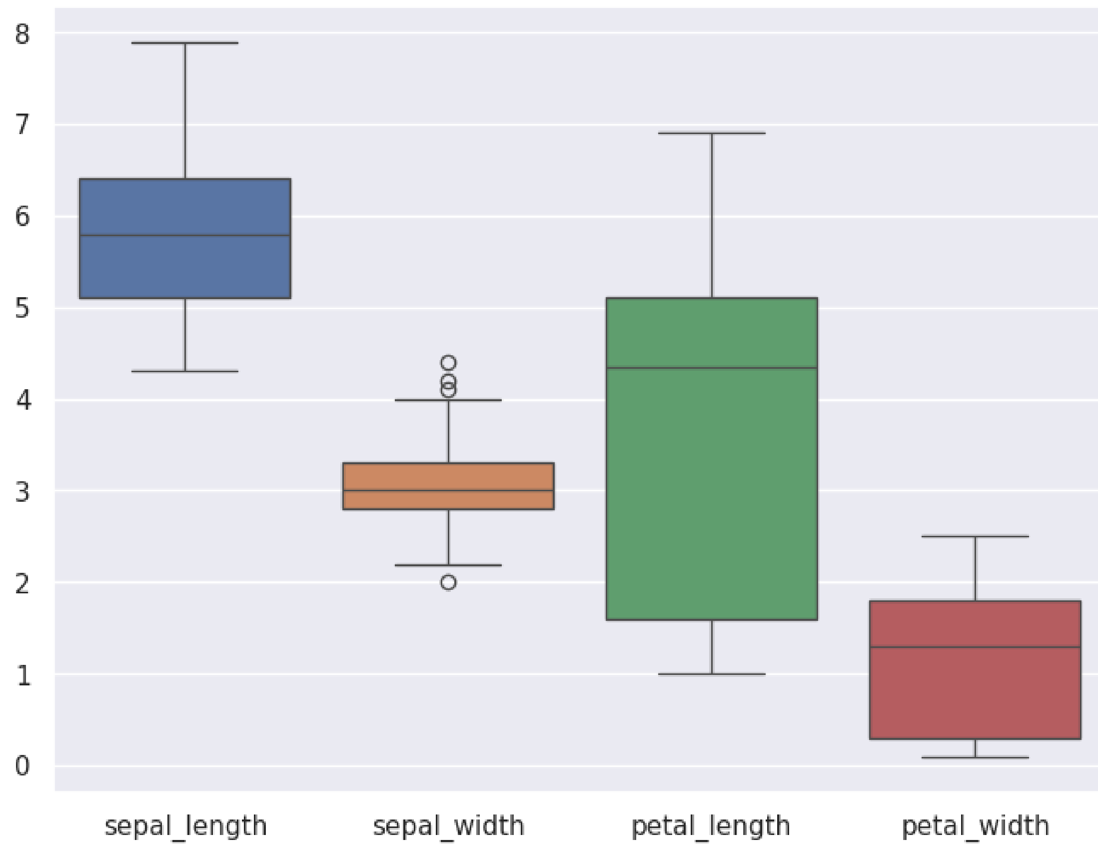
```
#Univariate Visualizations

df.hist()
```

```
array([[<Axes: title={'center': 'sepal_length'}>,
        <Axes: title={'center': 'sepal_width'}>],
       [<Axes: title={'center': 'petal_length'}>,
        <Axes: title={'center': 'petal_width'}>]], dtype=object)
```



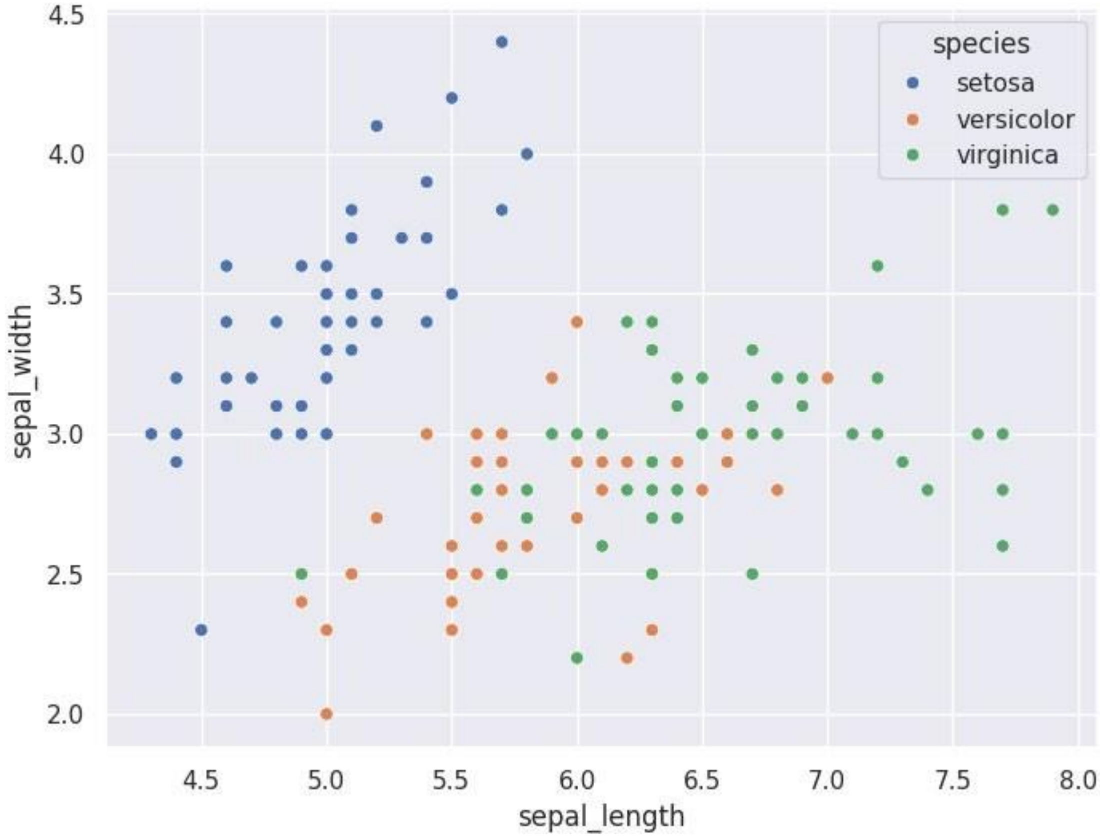

```
sns.boxplot(data=df)
```

```
<Axes: >
```



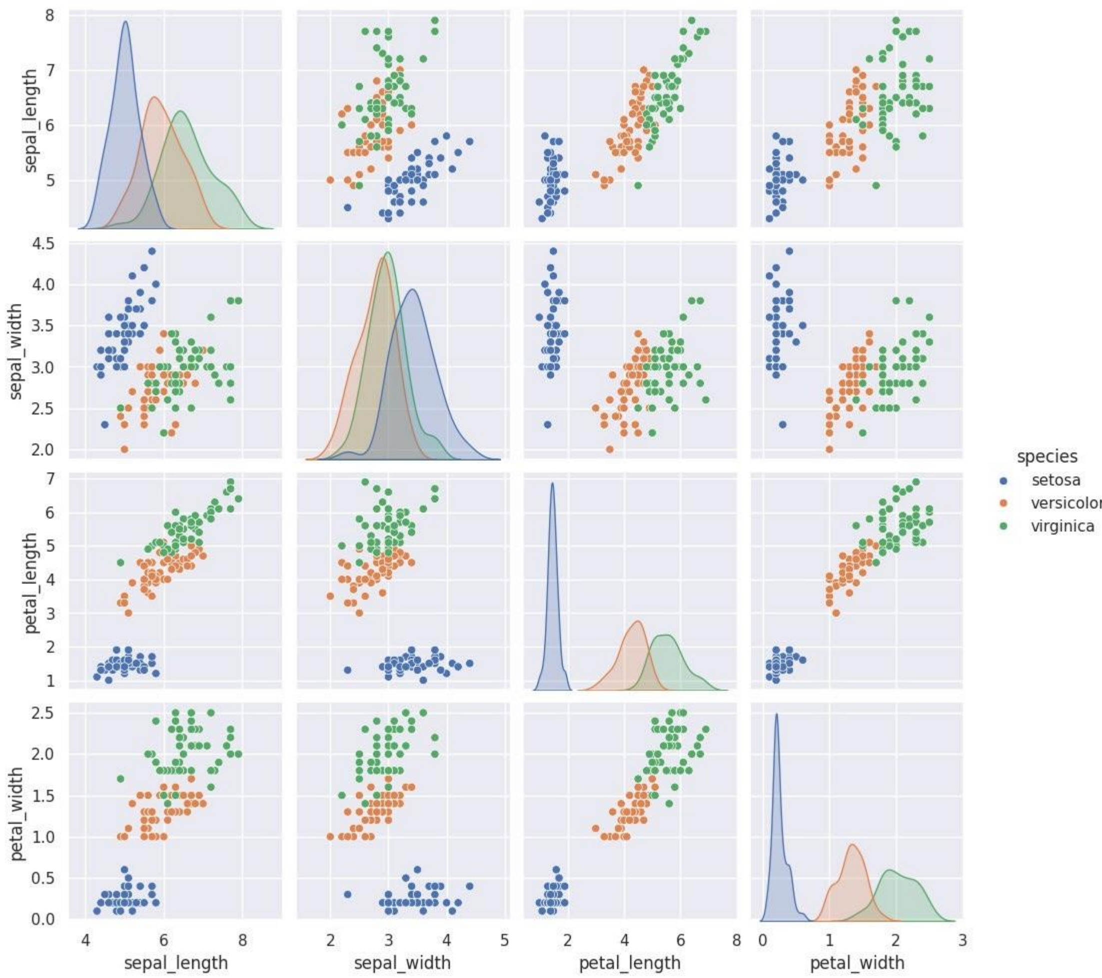

```
sns.scatterplot(data=df, x='sepal_length', y='sepal_width', hue='species')
```

```
<Axes: xlabel='sepal_length', ylabel='sepal_width'>
```



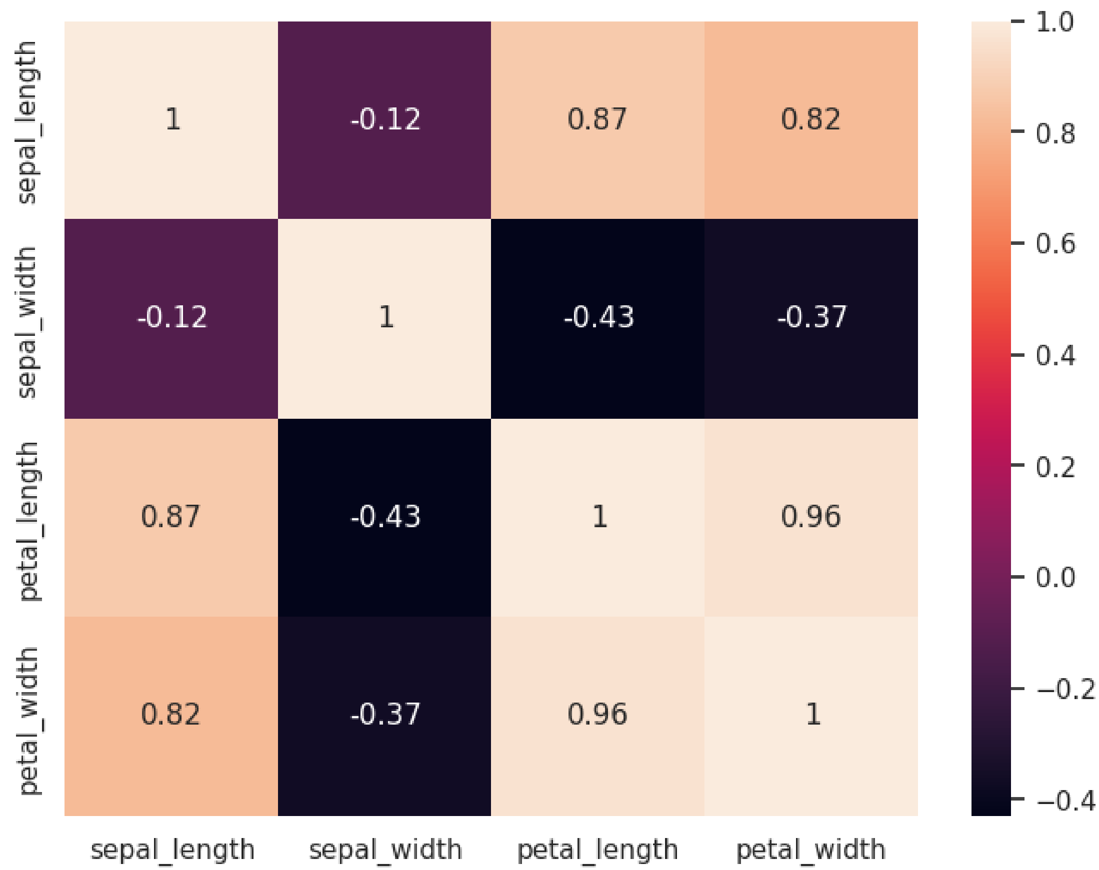

```
sns.pairplot(df, hue="species")
```

<seaborn.axisgrid.PairGrid at 0x7fc52c528610>




```
#Correleation

numeric_df = df.select_dtypes(include=[np.number])
correlation_matrix = numeric_df.corr()
sns.heatmap(correlation_matrix, annot=True)
```

```
<Axes: >
```




```
potential_features = df.select_dtypes(include=[np.number]).columns.tolist(
)
target_variable = 'species'

print("Potential Features: ", potential_features)
print("Target Variable: ", target_variable)
```

```
Potential Features:  ['sepal_length', 'sepal_width', 'petal_length', 'peta
l_width']
Target Variable:  species
```

1.3 Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization.

```
import seaborn as sns

import pandas as pd

from sklearn.preprocessing import LabelEncoder from
sklearn.preprocessing import StandardScaler from
sklearn.preprocessing import Binarizer
```

```
df = sns.load_dataset('titanic')
print(df.head())
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class
\
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

```
label_encoder = LabelEncoder()
df['sex'] = label_encoder.fit_transform(df['sex'])
print(df.sex.head())
```

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

```
scaler = StandardScaler()
df[['age', 'fare']] = scaler.fit_transform(df[['age', 'fare']])
print(df[['age', 'fare']].head())
```

```
        age      fare
0 -0.530377 -0.502445
1  0.571831  0.786845
2 -0.254825 -0.488854
```

```python
binarizer = Binarizer(threshold=0)
df[['fare']] = binarizer.fit_transform(df[['fare']])
print(df.fare.head())
```

```
0    0.0
1    1.0
2    0.0
3    1.0
4    0.0
Name: fare, dtype: float64
```

Practical 2

2.1 Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset)

```
import pandas as pd

data = {
    'Material': ['Plastic', 'Metal', 'Glass', 'Metal'],
    'Color': ['White', 'Silver', 'Green', 'Grey'],
    'Size': ['Small', 'Large', 'Small', 'Large'],
    'Recyclable': ['Yes', 'Yes', 'Yes', 'No'],
    'E-Waste': ['No', 'Yes', 'No', 'Yes']
}


df = pd.DataFrame(data)
df.to_csv('training_data.csv', index=False)

data = pd.read_csv('training_data.csv')
X = data.iloc[:, :-1]
y  =  data.iloc[:,  -1]

hypothesis = ['0'] * X.shape[1]

for i in range(len(X)):

  if y[i] == 'Yes':
    for j in range(X.shape[1]):

      if hypothesis[j] == '0':
        hypothesis[j] = X.iloc[i, j]
      elif hypothesis[j] != X.iloc[i, j]:
        hypothesis[j] = '?'

print(hypothesis)

['Metal', '?', 'Large', '?']
```

Practical 3

3.1 Simple Linear Regression Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
np.random.seed(42)
X = 2.5 * np.random.rand(100, 1)
y = 5 + 2 * X + np.random.randn(100, 1)

data = pd.DataFrame({'Feature': X.flatten(), 'Target': y.flatten()})
print(data.head())
```

```
    Feature    Target
0  0.936350  6.959748
1  2.376786  9.454564
2  1.829985  8.751730
3  1.496646  6.005724
4  0.390047  5.560421
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Feature', y='Target', data=data)
plt.title('Feature vs Target')
plt.show()
```

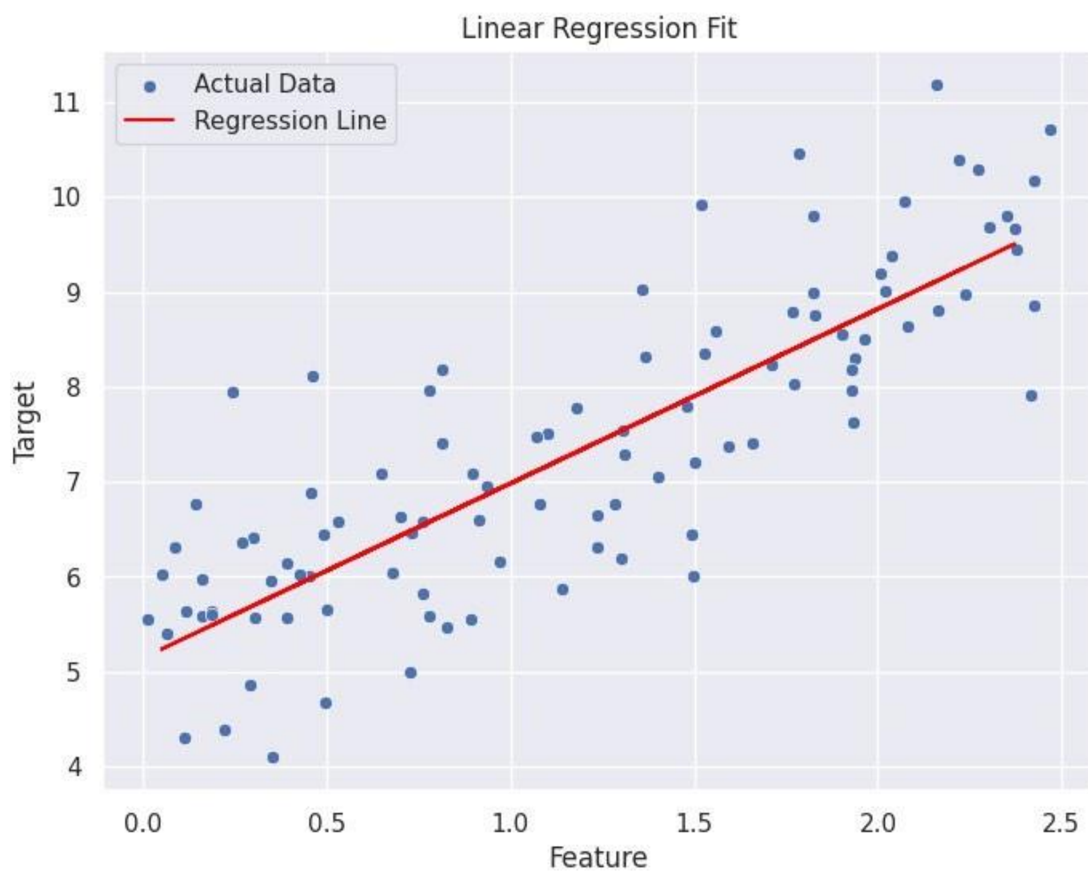

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=42)

model = LinearRegression()

model.fit(X_train, y_train)

print(f"Intercept: {model.intercept_[0]:.2f}")
print(f"Coefficient: {model.coef_[0][0]:.2f}")

Intercept: 5.14
Coefficient: 1.84

y_pred = model.predict(X_test)

pred_df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.fl
atten()})
print(pred_df.head())

     Actual  Predicted
0  5.974345   5.435196
1  8.970661   9.257909
2  7.624273   8.694195
```

```
3  7.403224   8.189620
4  7.084932   6.332951
```

```python
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")

r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2:.2f}")
```

```
Mean Squared Error (MSE): 0.65
R-squared: 0.73
```

```python
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Feature', y='Target', data=data, label='Actual Data')
plt.plot(X_test, y_pred, color='red', label='Regression Line')
plt.title('Linear Regression Fit')
plt.legend()
plt.show()
```

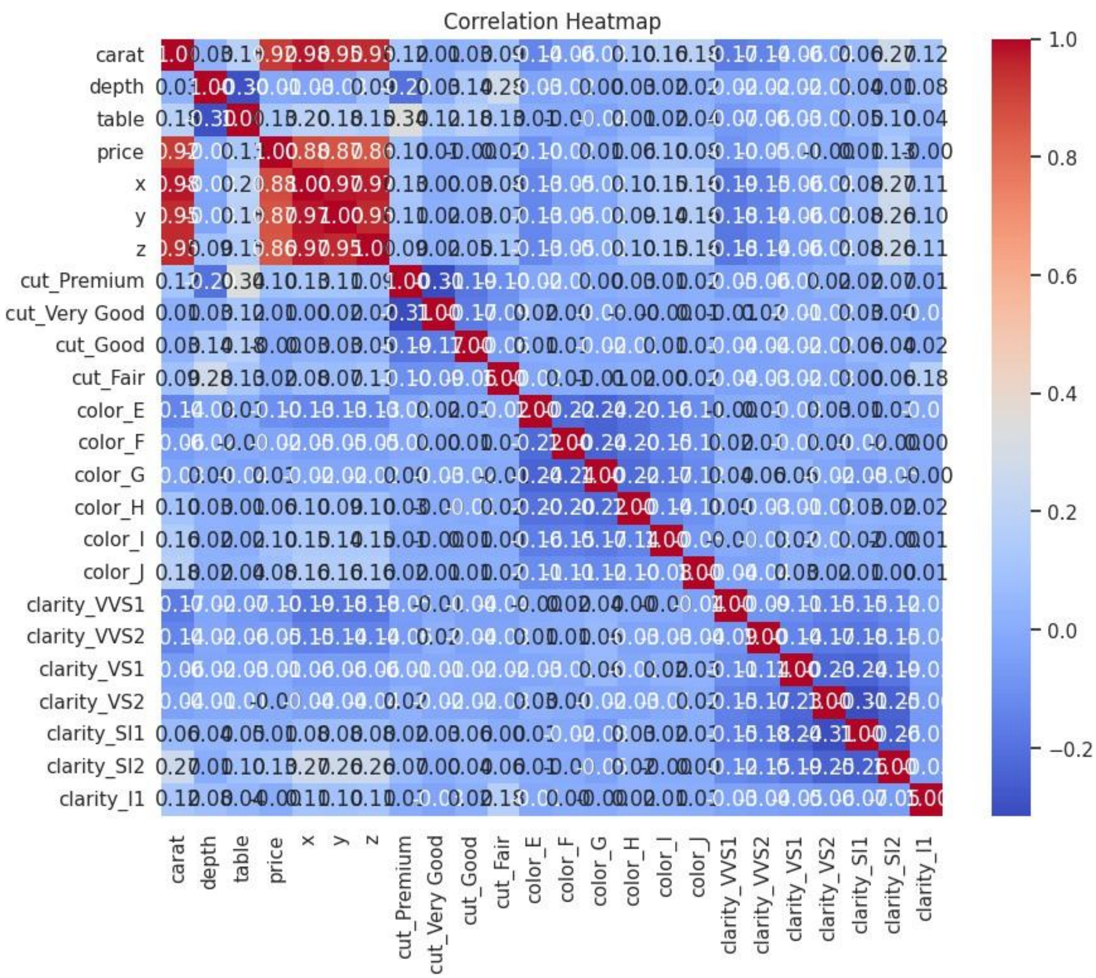


3.2 Multiple Linear Regression Extend linear regression to multiple features. Handle feature selection and potential multicollinearity.

```
import seaborn as sns import
pandas as pd import numpy as
np

import matplotlib.pyplot as plt

import statsmodels.api as sm

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor

df = sns.load_dataset('diamonds')

print("Missing values in the dataset:")
print(df.isnull().sum())

df = pd.get_dummies(df, drop_first=True)
```

```
Missing values in the dataset:
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap")
plt.show()
```




```
X = df[['carat', 'depth', 'table', 'x', 'y', 'z',
        'cut_Premium', 'cut_Good', 'cut_Very Good',
        'color_E', 'color_F', 'clarity_VVS2', 'clarity_VS1']]
y = df['price']

y = y.astype(float)

X_with_constant = sm.add_constant(X)

X_with_constant = X_with_constant.astype(int)

vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X_with_constant.values, i+1) for i
in range(len(X.columns))]
print(vif)

         Features        VIF
0           carat   3.613538
1           depth   1.211270
```

```
2           table   1.530672
3               x  19.224267
4               y  15.677513
5               z   5.789510
6     cut_Premium   1.548643
7        cut_Good   1.295429
8   cut_Very Good   1.346362
9         color_E   1.079848
10        color_F   1.060367
11   clarity_VVS2   1.051655
12    clarity_VS1   1.031049
```

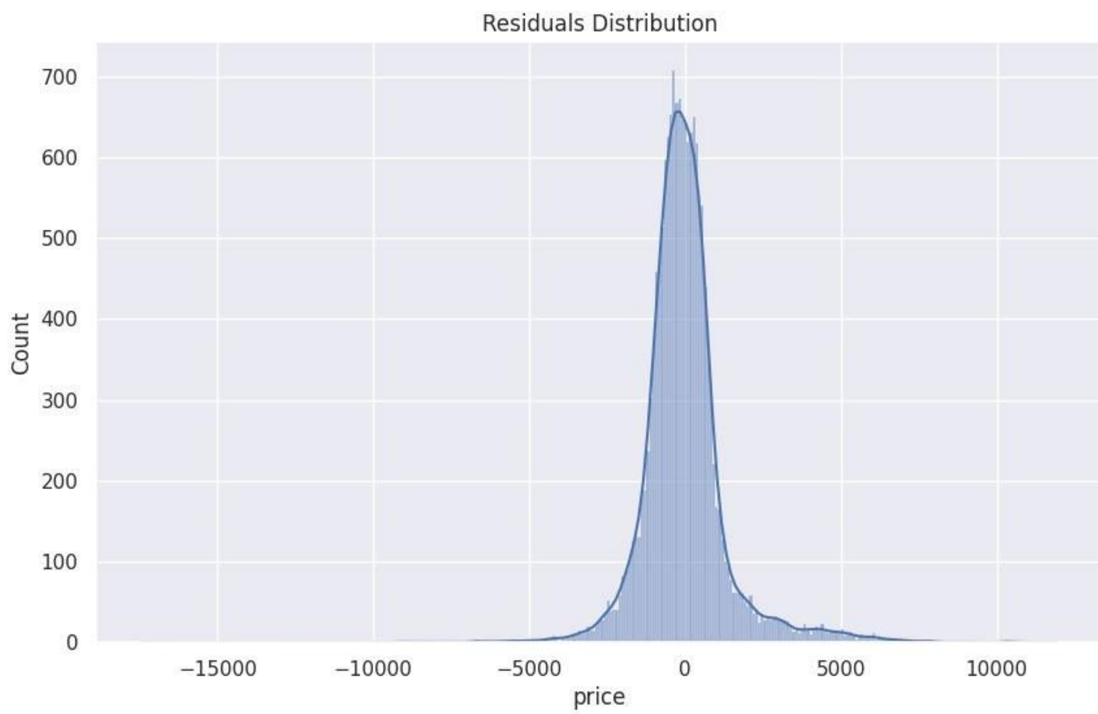
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)

print("Data types of X_train:")
print(X_train.dtypes)
print("Data type of y_train:", y_train.dtype)
```

```
Data types of X_train:
carat            float64
depth            float64
table            float64
x                float64
y                float64
z                float64
cut_Premium         bool
cut_Good            bool
cut_Very Good       bool
color_E             bool
color_F             bool
clarity_VVS2        bool
clarity_VS1         bool
dtype: object
Data type of y_train: float64
```

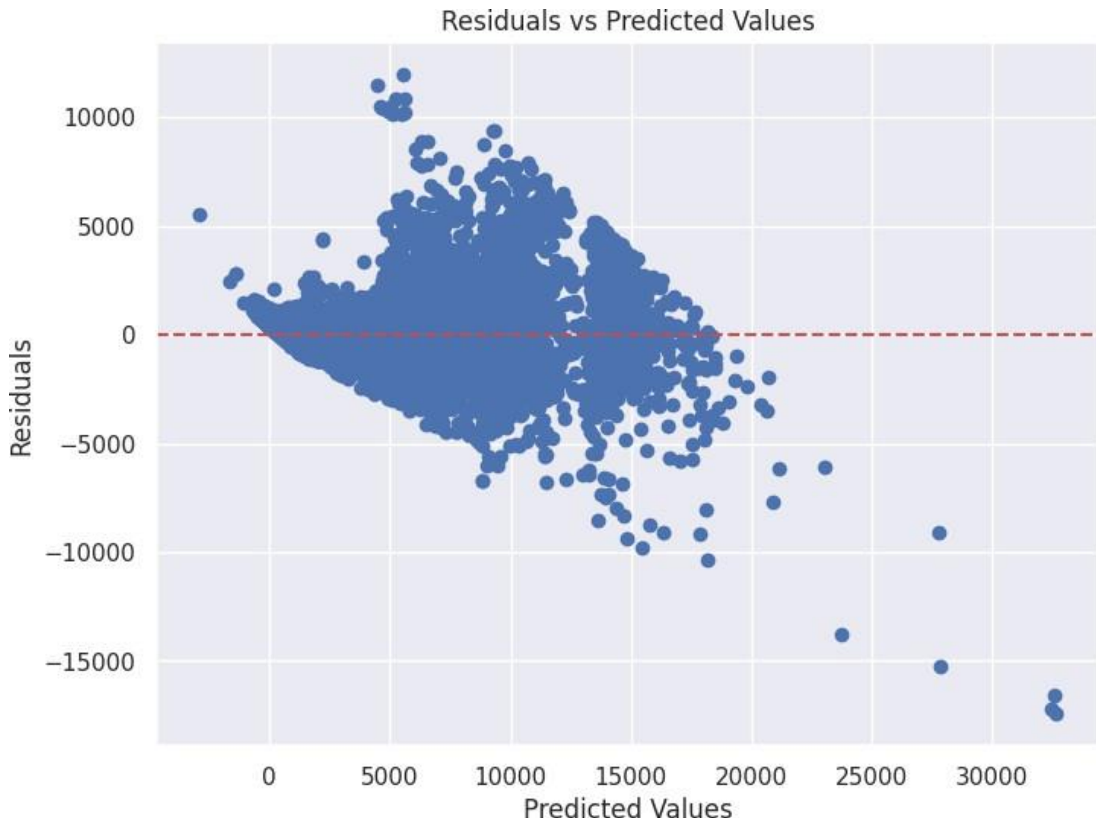
```
X_train = X_train.astype(float)
y_train = y_train.astype(float)

model = LinearRegression()
model.fit(X_train, y_train)

print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

```
Intercept: 17520.480548853404
Coefficients: [ 1.06799558e+04 -1.74848962e+02 -8.87411825e+01 -1.17618393
e+03
  3.03543071e+01  8.16330490e+00 -3.94552416e+01 -1.98436785e+02
```

```
 -1.87877044e+01  4.36586414e+02  4.74099129e+02  1.02842658e+03
  6.62232418e+02]

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

residuals = y_test - y_pred

Mean Squared Error: 2018911.748442661
R-squared: 0.8705490836162249

plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title("Residuals Distribution")
plt.show()
```




```
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted Values")
plt.show()
```



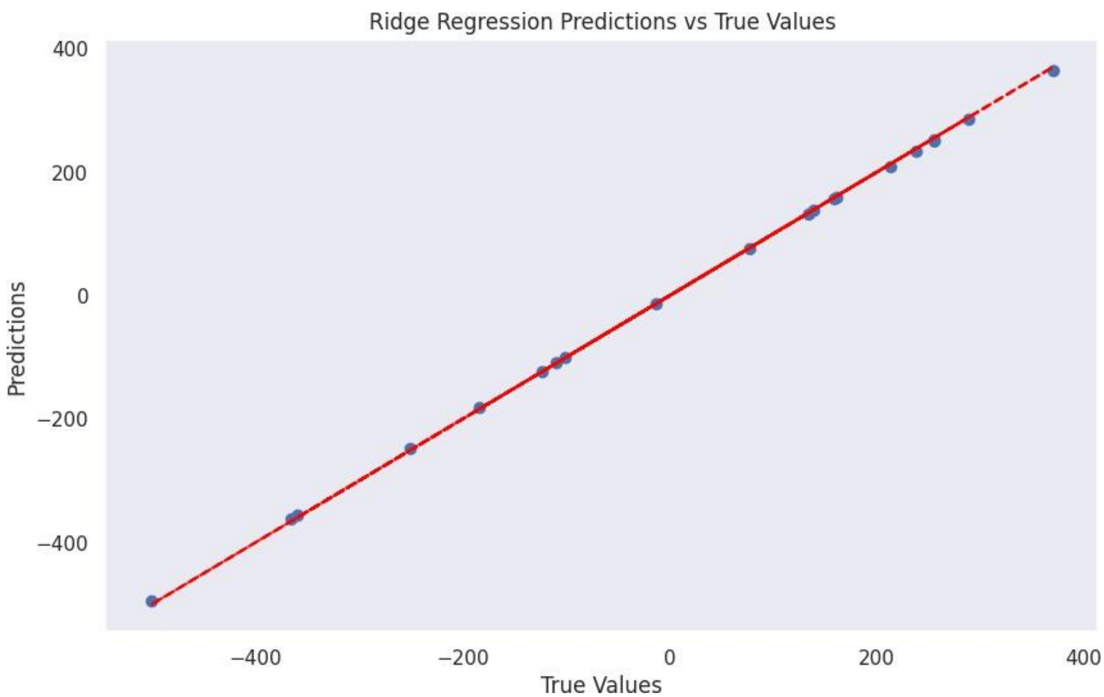

```
X_train_sm = sm.add_constant(X_train)
ols_model = sm.OLS(y_train, X_train_sm).fit()
print(ols_model.summary())
```

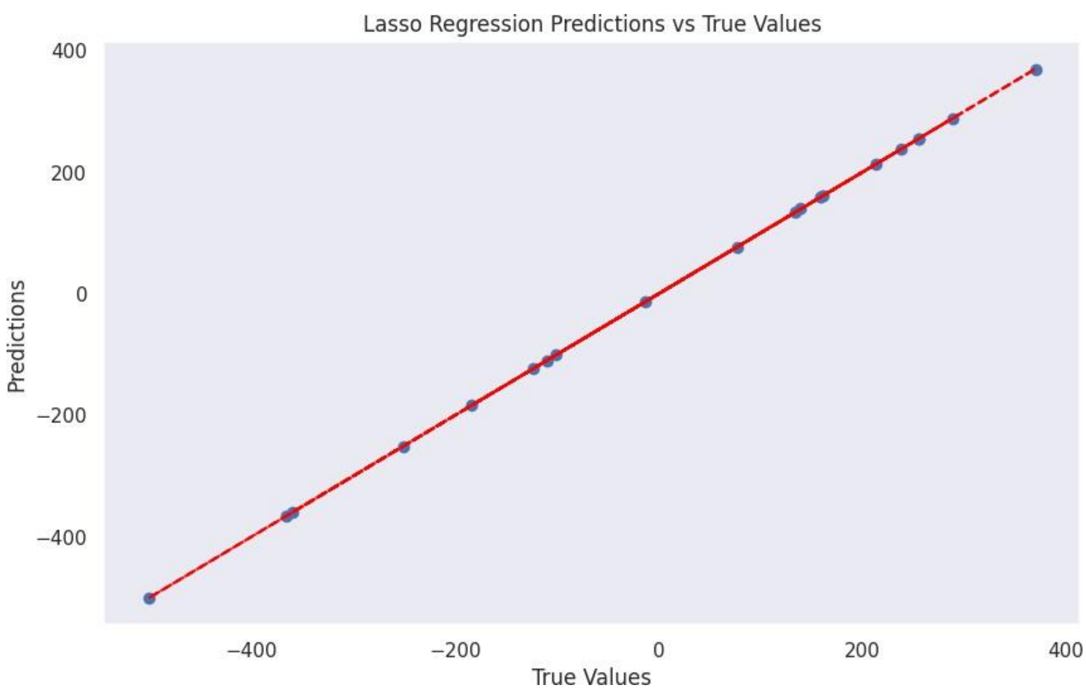
```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                  price   R-squared:                         0
.870
Model:                            OLS   Adj. R-squared:                    0
.869
Method:                 Least Squares   F-statistic:                   1.935
e+04
Date:                Thu, 24 Oct 2024   Prob (F-statistic):
0.00
Time:                        07:44:35   Log-Likelihood:              -3.2835
e+05
No. Observations:               37758   AIC:                           6.567
e+05
Df Residuals:                   37744   BIC:                           6.569
e+05
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
```

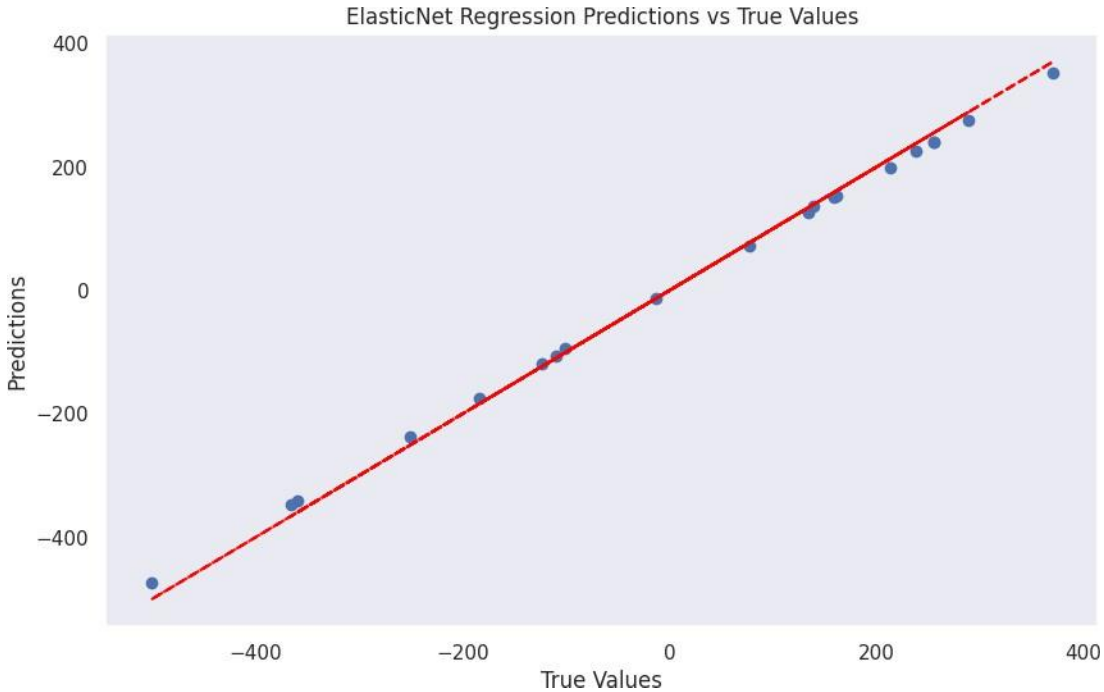
```
                    coef    std err          t      P>|t|      [0.025      
0.975]
------------------------------------------------------------------------------
------------
const          1.752e+04    537.976     32.567      0.000    1.65e+04    1
.86e+04
carat          1.068e+04     72.935    146.431      0.000    1.05e+04    1
.08e+04
depth          -174.8490      6.336    -27.595      0.000    -187.268    -
162.430
table           -88.7412      4.144    -21.414      0.000     -96.864     
-80.619
x             -1176.1839     46.936    -25.059      0.000   -1268.179   -1
084.188
y                30.3543     27.897      1.088      0.277     -24.325     
85.034
z                 8.1633     43.889      0.186      0.852     -77.861     
94.188
cut_Premium     -39.4552     21.233     -1.858      0.063     -81.073     
2.163
cut_Good       -198.4368     29.625     -6.698      0.000    -256.502    -
140.372
cut_Very Good   -18.7877     20.712     -0.907      0.364     -59.383     
21.808
color_E         436.5864     20.108     21.712      0.000     397.174     
475.999
color_F         474.0991     20.084     23.606      0.000     434.735     
513.463
clarity_VVS2   1028.4266     26.287     39.123      0.000     976.903    1
079.950
clarity_VS1     662.2324     21.075     31.423      0.000     620.925     
703.540
==================================================================

====
Omnibus:                     9105.593   Durbin-Watson:                   1
.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           327369
.326
Skew:                           0.453   Prob(JB):                         
0.00
Kurtosis:                      17.397   Cond. No.                     6.14
e+03
==================================================================

====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

3.3 Regularized Linear Models (Ridge, Lasso, ElasticNet) Implement regression variants like LASSO and Ridge on any generated dataset.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import make_regression

from sklearn.model_selection import train_test_split from
sklearn.linear_model import Ridge, Lasso, ElasticNet from sklearn.metrics
import mean_squared_error

X, y, coef = make_regression(n_samples=100, n_features=10, noise=0.1, coef
=True, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=42)

ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
ridge_pred = ridge_model.predict(X_test)

lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
lasso_pred = lasso_model.predict(X_test)

elastic_model = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_model.fit(X_train, y_train)
elastic_pred = elastic_model.predict(X_test)

def plot_results(y_test, predictions, model_name):
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, predictions)
    plt.plot(y_test, y_test, color='red', linestyle='--')  # y=x line
    plt.title(f'{model_name} Predictions vs True Values')
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.grid()
    plt.show()

plot_results(y_test, ridge_pred, 'Ridge Regression')
plot_results(y_test, lasso_pred, 'Lasso Regression')
plot_results(y_test, elastic_pred, 'ElasticNet Regression')
```


Ridge Regression Predictions vs True Values
Predictions
True Values
400
200
0
−200
−400
−400
−200
0
200
400


Lasso Regression Predictions vs True Values
Predictions
True Values
400
200
0
−200
−400
−400
−200
0
200
400




```
print("Mean Squared Error (MSE):")
print(f"Ridge: {mean_squared_error(y_test, ridge_pred):.2f}")
print(f"Lasso: {mean_squared_error(y_test, lasso_pred):.2f}")
print(f"ElasticNet: {mean_squared_error(y_test, elastic_pred):.2f}")
```

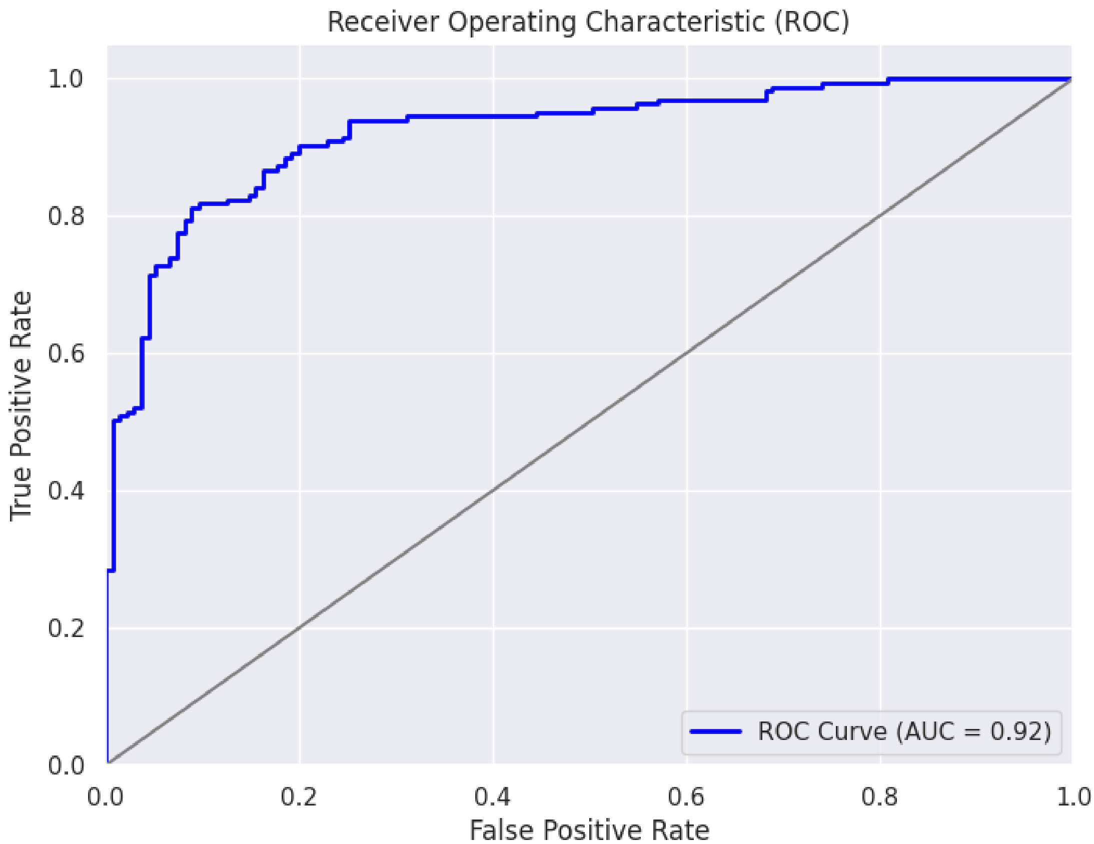
```
Mean Squared Error (MSE):
Ridge: 11.84
Lasso: 0.18
ElasticNet: 176.03
```

Practical 4

4.1 Logistic Regression Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
roc_curve, auc
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=10, n_classes=2, ran
dom_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)
logistic_reg_model = LogisticRegression()
logistic_reg_model.fit(X_train, y_train)
y_pred = logistic_reg_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")

Accuracy: 0.85
Precision: 0.89
Recall: 0.82

y_prob = logistic_reg_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC Curve (AUC = {roc_auc:.
2f})")
plt.plot([0, 1], [0, 1], color='gray', linestyle='-')  # Diagonal line for random classifier

plt.xlim([0.0, 1.0])
```

```
plt.legend(loc='lower right')
plt.show()
```



4.2 Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.

```python
from sklearn import datasets

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report

iris = datasets.load_iris()
X  =  iris.data
y = iris.target

df = pd.DataFrame(data=X, columns=iris.feature_names)
df['target'] = y

print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm
)  \)  \
0  0           5.1 5.1           3.5 3.5            1.4 1.4           0.  0.
2  2
1  1           4.9 4.9           3.0 3.0            1.4 1.4           0.  0.
2  2
2  2           4.7 4.7           3.2 3.2            1.3 1.3           0.  0.
2  2
3  3           4.6 4.6           3.1 3.1            1.5 1.5           0.  0.
2  2
4  4           5.0 5.0           3.6 3.6            1.4 1.4           0.  0.
2  2
```

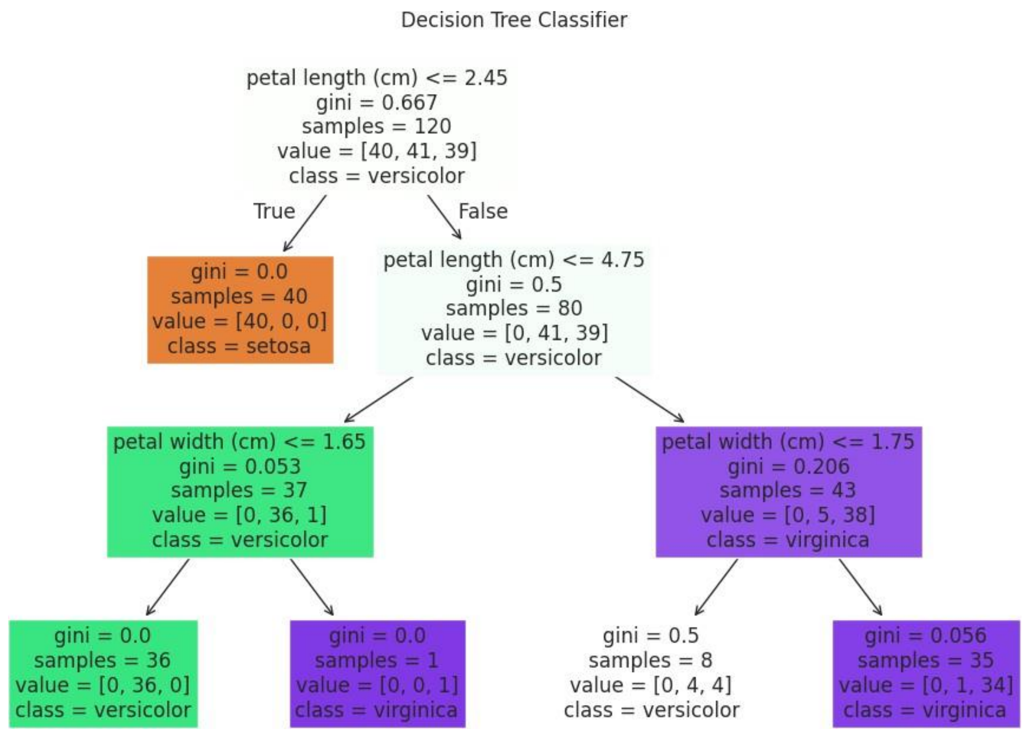
```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
y = y.astype('category').cat.codes

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=42)
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

for i in range(len(y_test)):

    print(f'Predicted: {iris.target_names[y_pred[i]]}, Actual: {iris.targe
t_names[y_test.iloc[i]]}')

Predicted: versicolor, Actual: versicolor
Predicted: setosa, Actual: setosa
Predicted: virginica, Actual: virginica
Predicted: versicolor, Actual: versicolor
Predicted: versicolor, Actual: versicolor
Predicted: setosa, Actual: setosa
Predicted: versicolor, Actual: versicolor
Predicted: virginica, Actual: virginica
Predicted: versicolor, Actual: versicolor
Predicted: versicolor, Actual: versicolor
Predicted: virginica, Actual: virginica
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa
Predicted: versicolor, Actual: versicolor
Predicted: virginica, Actual: virginica
Predicted: versicolor, Actual: versicolor
Predicted: versicolor, Actual: versicolor
Predicted: virginica, Actual: virginica
Predicted: setosa, Actual: setosa
Predicted: virginica, Actual: virginica
Predicted: setosa, Actual: setosa
Predicted: virginica, Actual: virginica
Predicted: virginica, Actual: virginica
Predicted: virginica, Actual: virginica
Predicted: virginica, Actual: virginica
Predicted: virginica, Actual: virginica
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa

accuracy = accuracy_score(y_test, y_pred)
```

```
print(classification_report(y_test, y_pred, target_names=iris.target_names
))
```

```
Accuracy: 100.00%
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

4.3 Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, pl
ot_tree
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, mean_squared_error

iris = load_iris()
X  =  iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=42)

clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

Accuracy: 1.00

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=
iris.target_names)
plt.title("Decision Tree Classifier")
plt.show()
```

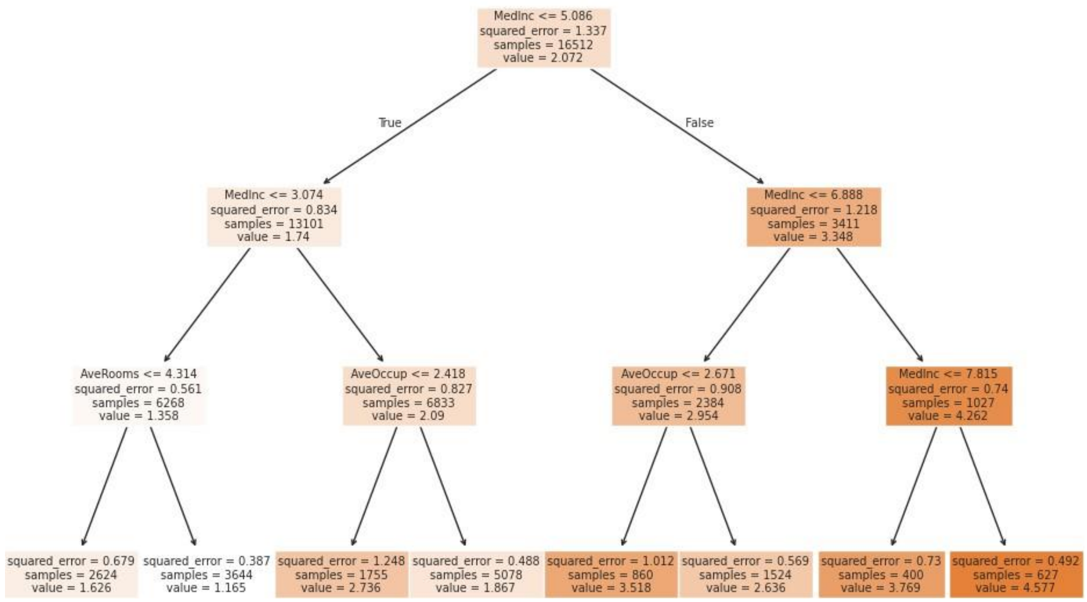



```
housing = fetch_california_housing()
X_housing = housing.data
y_housing = housing.target

X_train_housing, X_test_housing, y_train_housing, y_test_housing = train_t
est_split(X_housing, y_housing, test_size=0.2, random_state=42)

reg = DecisionTreeRegressor(max_depth=3, random_state=42)
reg.fit(X_train_housing, y_train_housing)

y_pred_housing = reg.predict(X_test_housing)

mse = mean_squared_error(y_test_housing, y_pred_housing)
print(f'Mean Squared Error: {mse:.2f}')
```

```
Mean Squared Error: 0.64
```

```
plt.figure(figsize=(12,8))
plot_tree(reg, filled=True, feature_names=housing.feature_names)
plt.title("Decision Tree Regressor")
plt.show()
```

Decision Tree Regressor



4.4 Implement a Support Vector Machine for any relevant dataset.

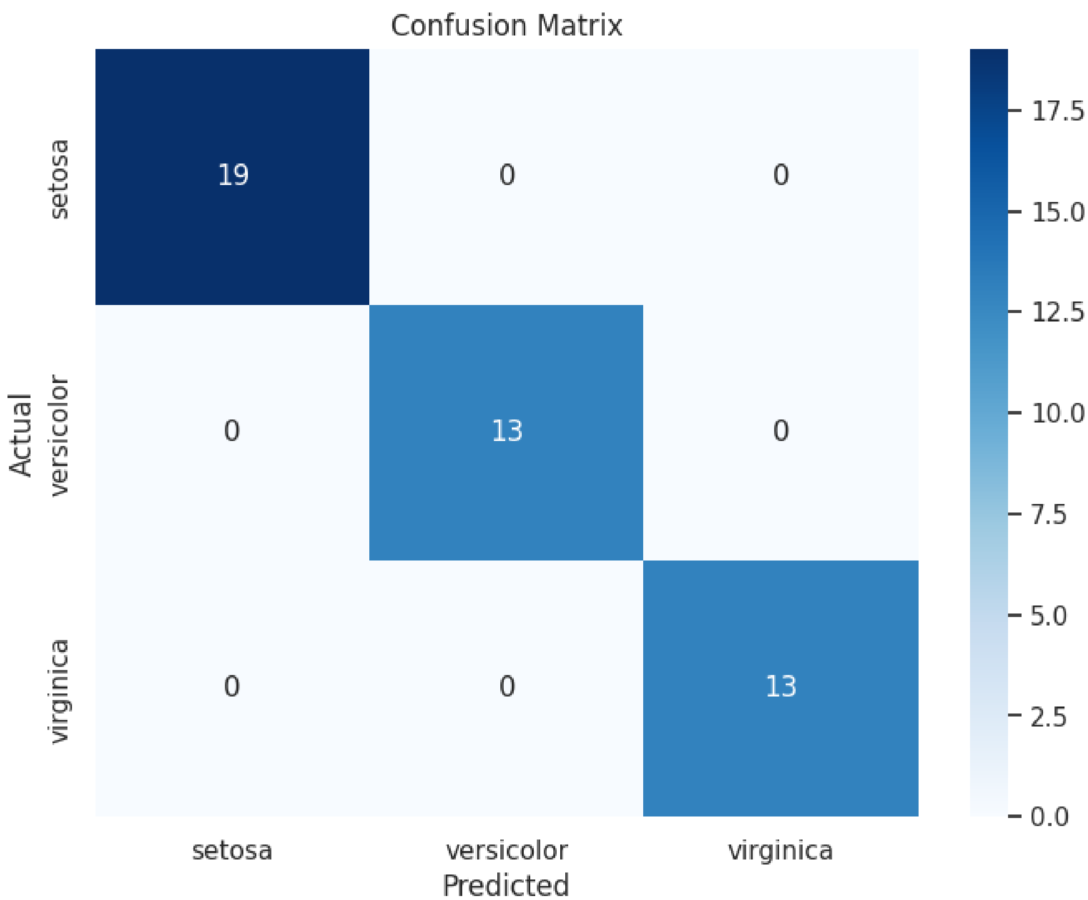
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix, classificati
on_report
import seaborn as sns

iris = datasets.load_iris()
X  =  iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train, y_train)

y_pred = svm_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

class_report = classification_report(y_test, y_pred, target_names=iris.tar
get_names)
print("\nClassification Report:")
print(class_report)
```

```
Accuracy: 1.00

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Classification Report:
              precision    recall  f1-score   support
```

```
      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13
```

```
    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=ir
is.target_names, yticklabels=iris.target_names)
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



4.5 Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split from
sklearn.tree  import  DecisionTreeClassifier from sklearn.ensemble
import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classificati
on_report
import seaborn as sns

iris = datasets.load_iris()
X  =  iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)
tree_classifier = DecisionTreeClassifier(random_state=42)
tree_classifier.fit(X_train, y_train)
y_pred_tree = tree_classifier.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print(f"Decision Tree Accuracy: {accuracy_tree:.2f}\n")
```

```
Decision Tree Accuracy: 1.00
```

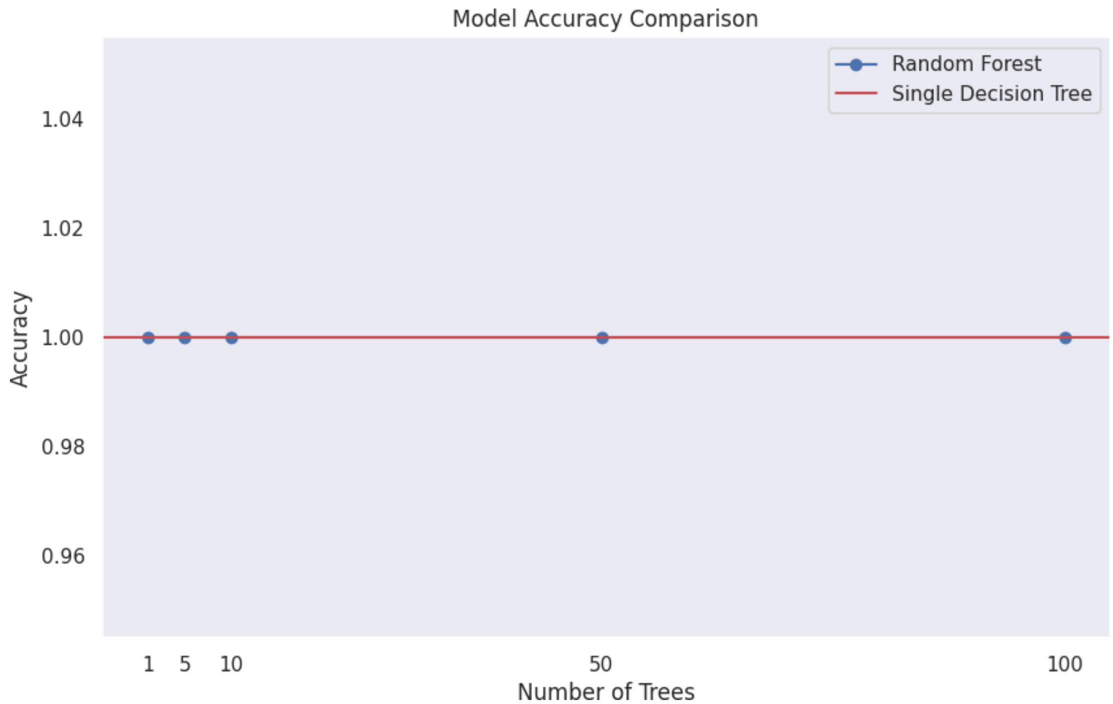
```
n_trees = [1, 5, 10, 50, 100]
accuracy_forest = []
for n in n_trees:

    forest_classifier = RandomForestClassifier(n_estimators=n, random_stat
e=42)
    forest_classifier.fit(X_train, y_train)
    y_pred_forest = forest_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred_forest)
    accuracy_forest.append(accuracy)
    print(f"Random Forest with {n} trees Accuracy: {accuracy:.2f}")
```

```
Random Forest with 1 trees Accuracy: 1.00
Random Forest with 5 trees Accuracy: 1.00
Random Forest with 10 trees Accuracy: 1.00
Random Forest with 50 trees Accuracy: 1.00
```

```
Random Forest with 100 trees Accuracy: 1.00
```

```
plt.figure(figsize=(10, 6))
plt.plot(n_trees, accuracy_forest, marker='o', label='Random Forest')
plt.axhline(y=accuracy_tree, color='r', linestyle='-', label='Single Decis
ion Tree')
plt.title('Model Accuracy Comparison')
```

```
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.xticks(n_trees)
plt.legend()
plt.grid()
plt.show()
```



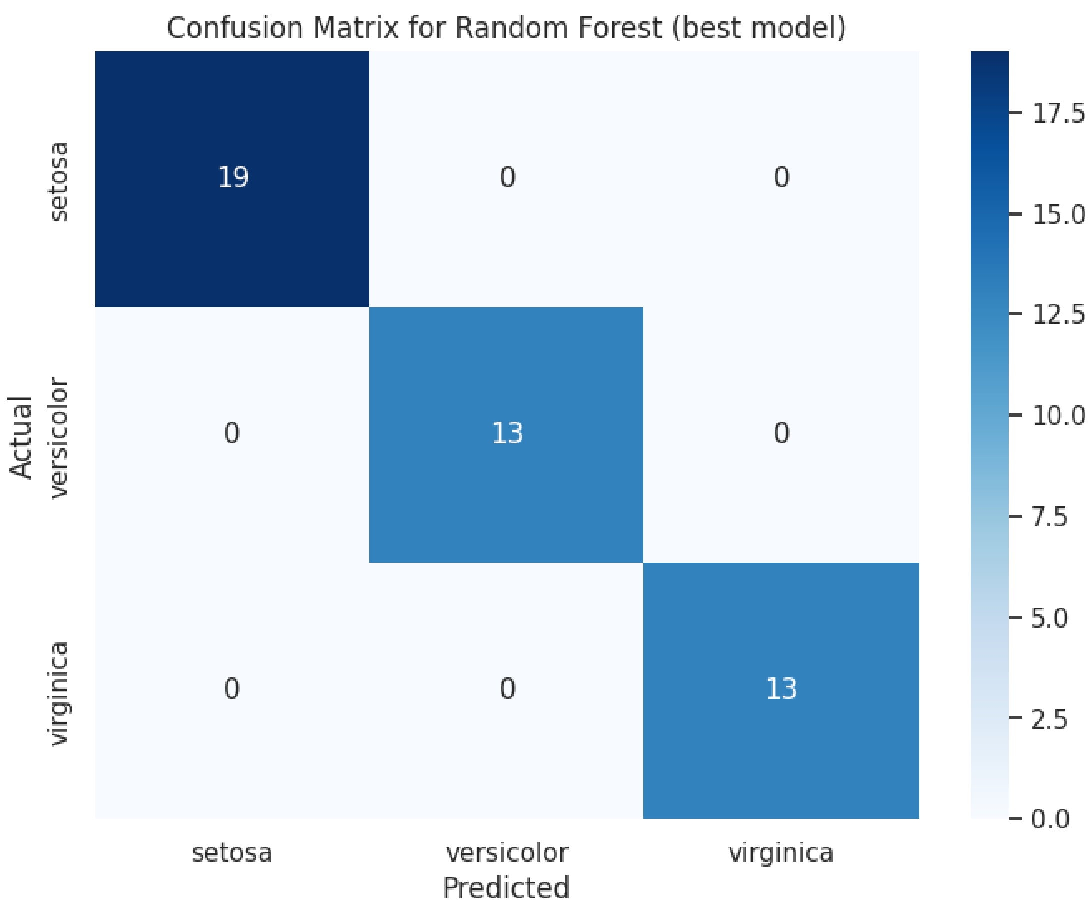
best_n = n_trees[np.argmax(accuracy_forest)] *# Get the best performing nu mber of trees*

```
best_forest_classifier = RandomForestClassifier(n_estimators=best_n, rando
m_state=42)
best_forest_classifier.fit(X_train, y_train)
y_pred_best_forest = best_forest_classifier.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred_best_forest)
print("\nConfusion Matrix for Random Forest (best model):")
print(conf_matrix)

Class_report = classification_report(y_test, y_pred_best_forest, target_na
mes=iris.target_names)
print("\nClassification Report for Random Forest (best model):")
print(class_report)
```

```
Confusion Matrix for Random Forest (best model):
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

```
Classification Report for Random Forest (best model):
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=ir
is.target_names, yticklabels=iris.target_names)
plt.title('Confusion Matrix for Random Forest (best model)')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



4.6 Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classificati
on_report

iris = datasets.load_iris()
X  = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb_model.fit(X_train, y_train)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, sco
ring='accuracy', cv=3, verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print("Best parameters from GridSearch:", best_params)
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best parameters from GridSearch: {'learning_rate': 0.01, 'max_depth': 4, '
n_estimators': 100, 'subsample': 0.8}
```

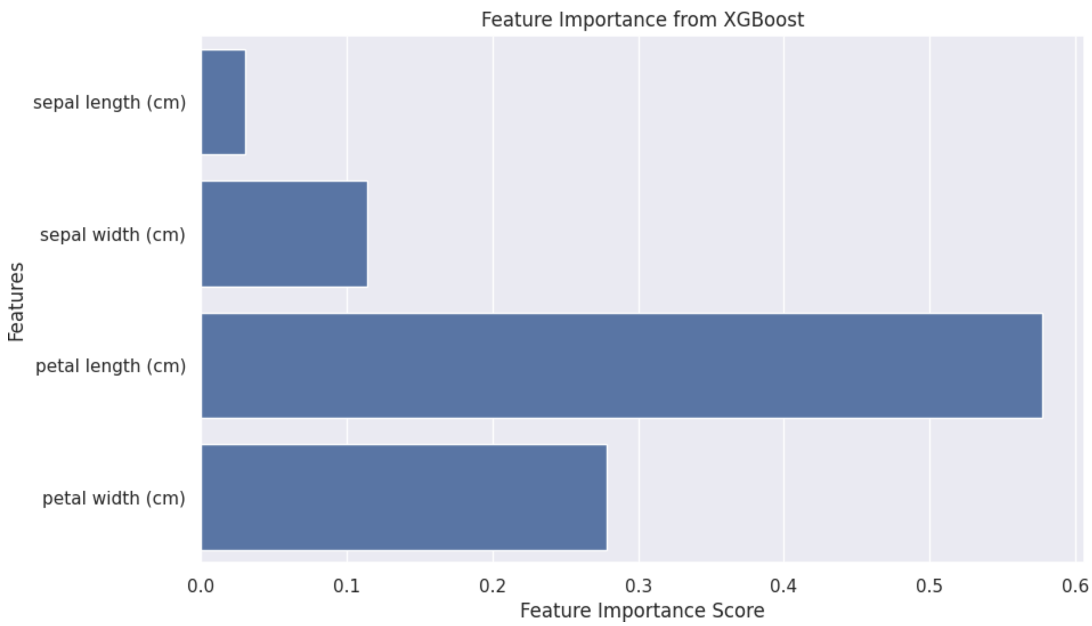
```
best_xgb_model = grid_search.best_estimator_
y_pred = best_xgb_model.predict(X_test)
```

```python
accuracy = accuracy_score(y_test, y_pred)
print(f"\nXGBoost Accuracy: {accuracy:.2f}\n")

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
class_report = classification_report(y_test, y_pred, target_names=iris.tar
get_names)
print("\nClassification Report:")
print(class_report)
```

```
XGBoost Accuracy: 1.00

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x=best_xgb_model.feature_importances_, y=iris.feature_names)
plt.title('Feature Importance from XGBoost')
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.show()
```


Feature Importance from XGBoost
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
Features
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Feature Importance Score

Practical 5

5.1 Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, classification_report, confusi
on_matrix

iris = load_iris()
X = iris.data
y = iris.target

df = pd.DataFrame(data=X, columns=iris.feature_names)
df['target'] = y

print(df.head())
```

```
sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm
\  )  \
   0           5.1   5.1           3.5   3.5           1.4   1.4           0.    0.
   2
   1           4.9   4.9           3.0   3.0           1.4   1.4           0.    0.
   2
   2           4.7   4.7           3.2   3.2           1.3   1.3           0.    0.
   2
   3           4.6   4.6           3.1   3.1           1.5   1.5           0.    0.
   2
   4           5.0   5.0           3.6   3.6           1.4   1.4           0.    0.
   2
```

```python
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

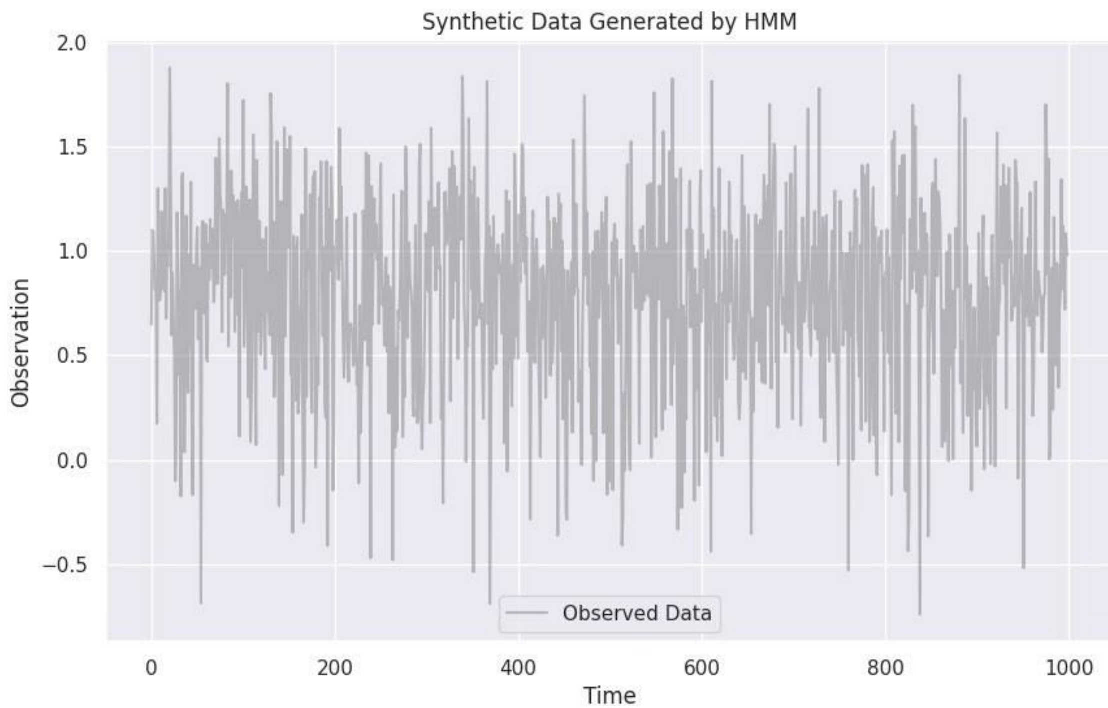
```
Accuracy: 0.98
Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

5.2 Implement Hidden Markov Models using hmmlearn

```
!pip install hmmlearn
import numpy as np

import matplotlib.pyplot as plt

from hmmlearn import hmm
```

```
Requirement already satisfied: hmmlearn in /usr/local/lib/python3.10/dist-packages (0.3.2)
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.10/dist-packages (from hmmlearn) (1.26.4)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in /usr/local/lib/python3.10/dist-packages (from hmmlearn) (1.5.2)
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.10/dist-packages (from hmmlearn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (3.5.0)
```

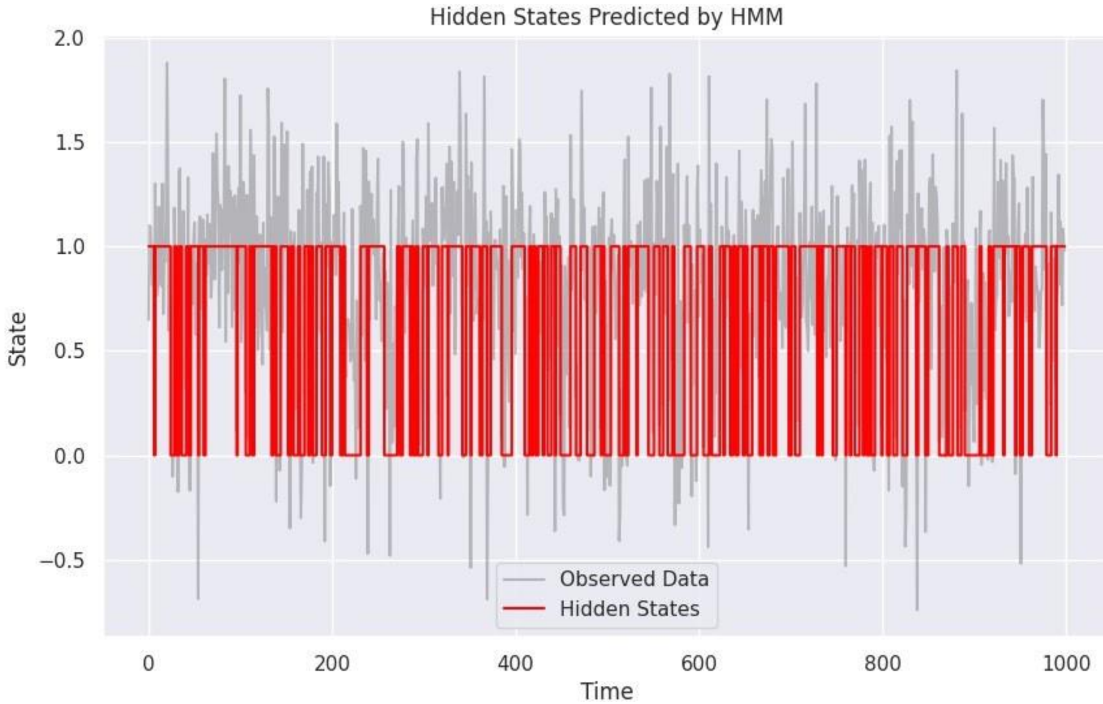
```
np.random.seed(42)

n_samples = 1000
n_states = 2
trans_probs = np.array([[0.7, 0.3],
                        [0.4, 0.6]])
means = np.array([[1.0], [0.5]])
covars = np.array([[0.1], [0.2]])

model = hmm.GaussianHMM(n_components=n_states, covariance_type="diag", n_iter=100)
model.startprob_ = np.array([0.6, 0.4])
model.transmat_ = trans_probs
model.means_ = means
model.covars_ = covars

X, Z = model.sample(n_samples)

plt.figure(figsize=(10, 6))
plt.plot(X, label='Observed Data', color='grey', alpha=0.5)
plt.title('Synthetic Data Generated by HMM')
plt.xlabel('Time')
plt.ylabel('Observation')
plt.legend()
plt.show()
```




```
model.fit(X)
hidden_states = model.predict(X)
```

```
WARNING:hmmlearn.base:Even though the 'startprob_' attribute is set, it wi
ll be overwritten during initialization because 'init_params' contains 's'
WARNING:hmmlearn.base:Even though the 'transmat_' attribute is set, it wil
l be overwritten during initialization because 'init_params' contains 't'
WARNING:hmmlearn.base:Even though the 'means_' attribute is set, it will b
e overwritten during initialization because 'init_params' contains 'm'
WARNING:hmmlearn.base:Even though the 'covars_' attribute is set, it will
be overwritten during initialization because 'init_params' contains 'c'
```

```
plt.figure(figsize=(10, 6))
plt.plot(X, label='Observed Data', color='grey', alpha=0.5)
plt.step(range(n_samples), hidden_states, where="post", label='Hidden Stat
es', color='red')
plt.title('Hidden States Predicted by HMM')
plt.xlabel('Time')
plt.ylabel('State')
plt.legend()
plt.show()
```




```
print("Transition matrix:\n", model.transmat_)
print("Means:\n", model.means_)
print("Covariances:\n", model.covars_)
```

```
Transition matrix:
 [[0.65865532 0.34134468]
 [0.3121865  0.6878135 ]]
Means:
 [[0.54954006]
 [1.00338912]]
Covariances:
 [[[0.22176075]]

 [[0.09283459]]]
```

Practical 6

6.1 Implement Bayesian Linear Regression to explore prior and posterior distribution.

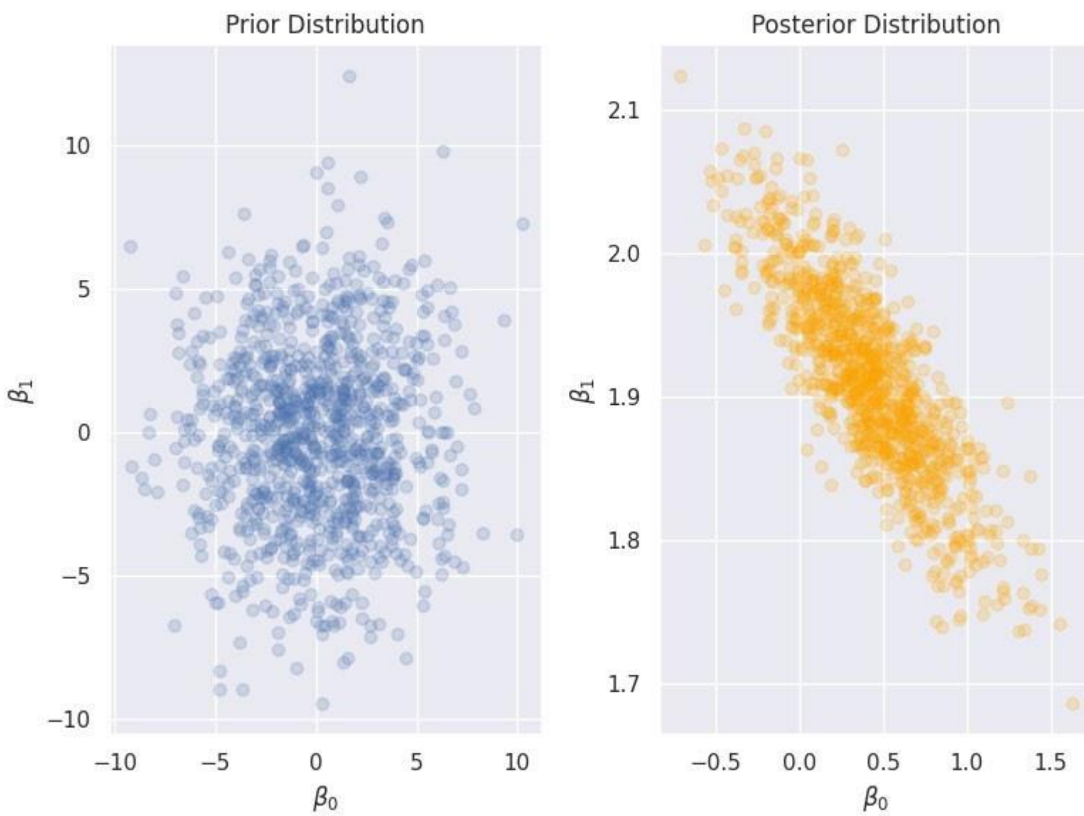
```
import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import multivariate_normal
```

```
np.random.seed(42)
X = np.random.rand(100, 1) * 10
true_beta = np.array([2.0])
y = 2.0 * X.flatten() + np.random.normal(0, 2, size=X.shape[0])
X_b = np.c_[np.ones((X.shape[0], 1)), X]
sigma_0 = 10
sigma_n = 4
sigma_0_inv = 1 / sigma_0
sigma_n_inv = 1 / sigma_n
N = X_b.shape[0]
beta_prior_mean = np.zeros(X_b.shape[1])
beta_prior_cov = sigma_0 * np.eye(X_b.shape[1])
posterior_cov = np.linalg.inv(sigma_n_inv * (X_b.T @ X_b) + sigma_0_inv *
np.eye(X_b.shape[1]))
posterior_mean = posterior_cov @ (sigma_n_inv * (X_b.T @ y))
beta_samples = np.random.multivariate_normal(posterior_mean, posterior_cov
, size=1000)
plt.figure(figsize=(10, 6))
beta_prior_samples = np.random.multivariate_normal(beta_prior_mean, beta_p
rior_cov, size=1000)
```

```
<Figure size 1000x600 with 0 Axes>
```

```
plt.subplot(1, 2, 1)
plt.title("Prior Distribution")
plt.scatter(beta_prior_samples[:, 0], beta_prior_samples[:, 1], alpha=0.2)
plt.xlabel("$\\beta_0$")
plt.ylabel("$\\beta_1$")
plt.subplot(1, 2, 2)
plt.title("Posterior Distribution")
plt.scatter(beta_samples[:, 0], beta_samples[:, 1], alpha=0.2, color='oran
ge')
plt.xlabel("$\\beta_0$")
plt.ylabel("$\\beta_1$")
plt.tight_layout()
```

```
plt.show()
```

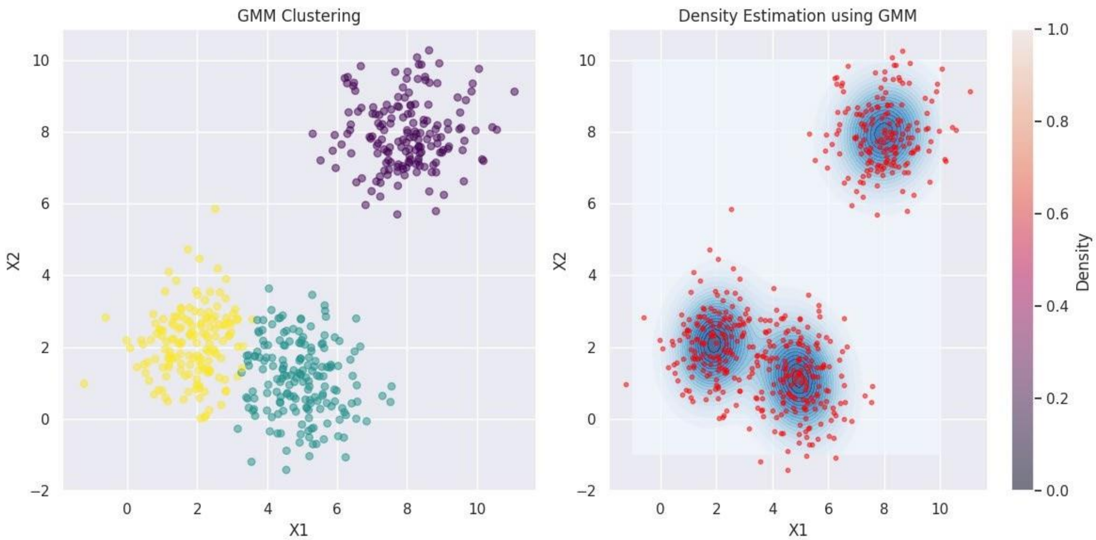



```
print("Posterior Mean:", posterior_mean)
print("Posterior Covariance:\n", posterior_cov)
```

```
Posterior Mean: [0.42825291 1.90809351]
Posterior Covariance:
 [[ 0.1389247  -0.0211579 ]
 [-0.0211579   0.00451795]]
```

6.2 Implement Gaussian Mixture Models for density estimation and unsupervised clustering

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.mixture import GaussianMixture

np.random.seed(42)
means = [[2, 2], [8, 8], [5, 1]]
covariances = [[[1, 0], [0, 1]], [[1, 0], [0, 1]], [[1, 0], [0, 1]]]
n_samples = 500
data = np.vstack([
    np.random.multivariate_normal(mean, cov, n_samples // len(means))
    for mean, cov in zip(means, covariances)
])

n_components = len(means)  # Number of clusters

gmm = GaussianMixture(n_components=n_components, covariance_type='full')
gmm.fit(data)
labels = gmm.predict(data)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(data[:, 0], data[:, 1], c=labels, s=30, cmap='viridis', alpha=
0.5)
plt.title('GMM Clustering')
plt.xlabel('X1')
plt.ylabel('X2')
x = np.linspace(-1, 10, 100)
y = np.linspace(-1, 10, 100)
X, Y = np.meshgrid(x, y)
XX = np.column_stack([X.ravel(), Y.ravel()])
logprob = gmm.score_samples(XX)
pdf = np.exp(logprob).reshape(X.shape)
plt.subplot(1, 2, 2)
plt.contourf(X, Y, pdf, levels=20, cmap='Blues', alpha=0.7)
plt.scatter(data[:, 0], data[:, 1], c='red', s=10, alpha=0.5)
plt.title('Density Estimation using GMM')
plt.xlabel('X1')
plt.ylabel('X2')
plt.colorbar(label='Density')
plt.tight_layout()
plt.show()
```


GMM Clustering
X1
X2
Density Estimation using GMM
Density

Practical 7

7.1 Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation

```
import numpy as np

from sklearn.model_selection import KFold, StratifiedKFold

from sklearn.metrics import accuracy_score

from sklearn.datasets import load_iris

from sklearn.ensemble import RandomForestClassifier
```

```
data = load_iris()
X, y = data.data, data.target
model = RandomForestClassifier()
kf = KFold(n_splits=5, shuffle=True, random_state=42)
kf_scores = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    score = accuracy_score(y_test, predictions)
    kf_scores.append(score)
print(f'K-Fold Accuracy: {np.mean(kf_scores):.2f} ± {np.std(kf_scores):.2f
}')
```

```
K-Fold Accuracy: 0.96 ± 0.02
```

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
skf_scores = []
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    score = accuracy_score(y_test, predictions)
    skf_scores.append(score)
print(f'Stratified K-Fold Accuracy: {np.mean(skf_scores):.2f} ± {np.std(sk
f_scores):.2f}')
```

```
Stratified K-Fold Accuracy: 0.95 ± 0.03
```

7.2 Systematically explore combinations of hyperparameters to optimize model performance.(use grid and randomized search)

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split, GridSearchCV, Random
izedSearchCV
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

data = load_iris()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)
model = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
}

grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring
='accuracy', cv=5)
grid_search.fit(X_train, y_train)
print("Grid Search Best Parameters:", grid_search.best_params_)
print("Grid Search Best Score:", grid_search.best_score_)

Grid Search Best Parameters: {'max_depth': 5, 'min_samples_split': 5, 'n_e
stimators': 10}
Grid Search Best Score: 0.9636363636363636

param_dist = {
    'n_estimators': np.arange(10, 200, 10),
    'max_depth': [None] + list(np.arange(1, 20, 1)),
    'min_samples_split': np.arange(2, 20, 2),
}

random_search = RandomizedSearchCV(estimator=model, param_distributions=pa
ram_dist, n_iter=50, scoring='accuracy', cv=5, random_state=42)
random_search.fit(X_train, y_train)
print("Randomized Search Best Parameters:", random_search.best_params_)
print("Randomized Search Best Score:", random_search.best_score_)

Randomized Search Best Parameters: {'n_estimators': 120, 'min_samples_spli
```

```
t': 16, 'max_depth': None}
Randomized Search Best Score: 0.9636363636363636
```

```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

Practical 8

8.1 Implement Bayesian Learning using inferences

```
import numpy as np

P_A = 0.5
P_B = 0.5

def likelihood_heads(coin, flips):
    if coin == 'A':
        return (0.5 * flips) * (0.5 * (10 - flips))
    elif coin == 'B':

        return (0.9 * flips) * (0.1 * (10 - flips))

observed_heads = 8
total_flips = 10

likelihood_A = likelihood_heads('A', observed_heads)
likelihood_B = likelihood_heads('B', observed_heads)

marginal_likelihood = (likelihood_A * P_A) + (likelihood_B * P_B)

posterior_A = (likelihood_A * P_A) / marginal_likelihood
posterior_B = (likelihood_B * P_B) / marginal_likelihood

print(f"Posterior Probability of Coin A: {posterior_A:.4f}")
print(f"Posterior Probability of Coin B: {posterior_B:.4f}")
```

```
Posterior Probability of Coin A: 0.7353
Posterior Probability of Coin B: 0.2647
```

Practical 9

9.1 Set up a generator network to produce samples and a discriminator network to distinguish between real and generated data. (Use a simple small dataset)

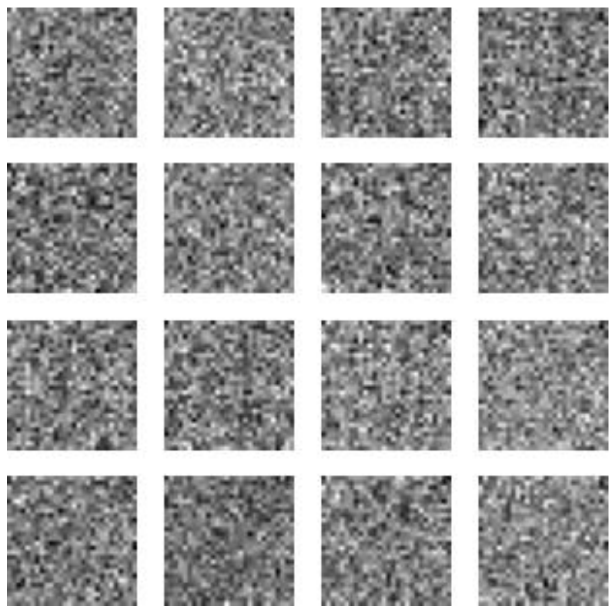
```
import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

(X_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
X_train = (X_train.astype(np.float32) - 127.5) / 127.5
X_train = np.expand_dims(X_train, axis=-1)

latent_dim = 100
num_examples_to_generate = 16

def build_generator():
    model = models.Sequential()
    model.add(layers.Dense(256, input_dim=latent_dim))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(512))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(1024))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(28 * 28 * 1, activation='tanh'))
    model.add(layers.Reshape((28, 28, 1)))
    return model

def build_discriminator():
    model = models.Sequential()
    model.add(layers.Flatten(input_shape=(28, 28, 1)))
    model.add(layers.Dense(512))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Dense(256))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model
```

```
generator = build_generator()
discriminator = build_discriminator()
discriminator.compile(optimizer='adam', loss='binary_crossentropy', metric
s=['accuracy'])
```

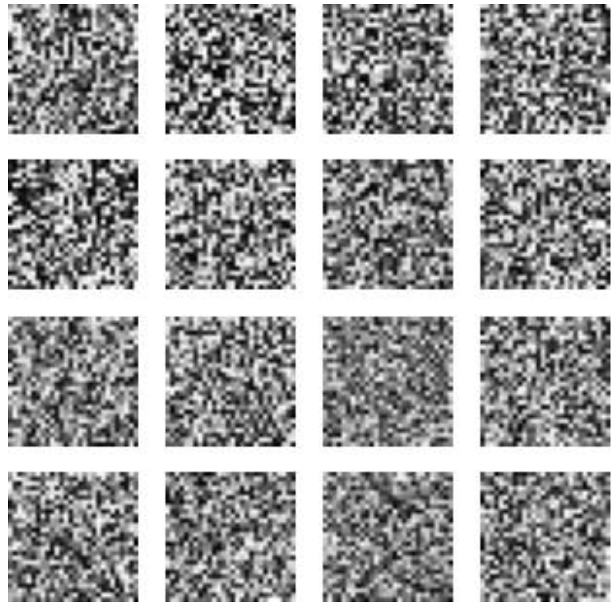
```python
discriminator.trainable = False
gan_input = layers.Input(shape=(latent_dim,))
generated_image = generator(gan_input)
gan_output = discriminator(generated_image)
gan = models.Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy')

def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input)
    predictions = (predictions.numpy() + 1) / 2  # Rescale to [0, 1]


    plt.figure(figsize=(4, 4))
    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i + 1)
        plt.imshow(predictions[i, :, :, 0], cmap='gray')
        plt.axis('off')
    plt.savefig(f'gan_epoch_{epoch}.png')
    plt.show()

def train_gan(epochs, batch_size):
    random_latent_vectors = tf.random.normal(shape=(num_examples_to_genera
te, latent_dim))

    for epoch in range(epochs):

        idx = np.random.randint(0, X_train.shape[0], batch_size)
        real_images = X_train[idx]

        noise = tf.random.normal(shape=(batch_size, latent_dim))
        fake_images = generator(noise)

        combined_images = tf.concat([real_images, fake_images], axis=0)

        labels = tf.constant([[1.0]] * batch_size + [[0.0]] * batch_size)

        d_loss = discriminator.train_on_batch(combined_images, labels)

        noise = tf.random.normal(shape=(batch_size, latent_dim))
        misleading_labels = tf.constant([[1.0]] * batch_size)

        g_loss = gan.train_on_batch(noise, misleading_labels)

        if epoch % 100 == 0:
            print(f"Epoch: {epoch}")
            print(f"Discriminator Loss: {d_loss[0]}")
            print(f"Generator Loss: {g_loss}")

            generate_and_save_images(generator, epoch, random_latent_vecto
rs)
```

```
epochs = 300
batch_size = 64

train_gan(epochs, batch_size)
```

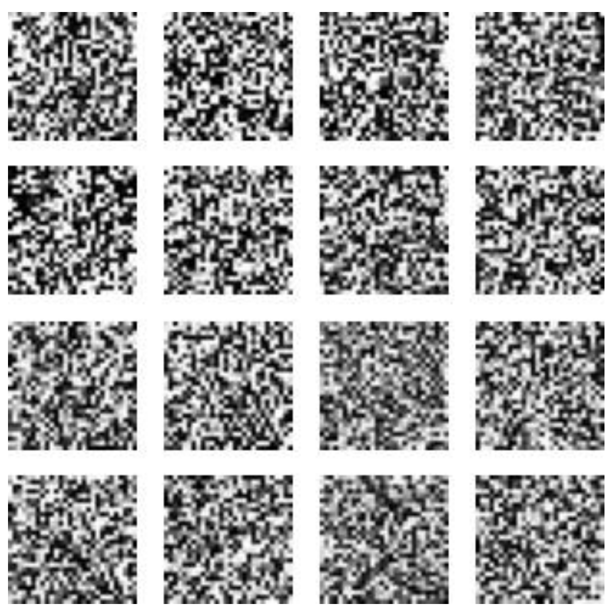
```
Epoch: 0
Discriminator Loss: 0.7258248329162598
Generator Loss: [array(0.72582483, dtype=float32), array(0.72582483, dtype
=float32), array(0.390625, dtype=float32)]
```



```
Epoch: 100
Discriminator Loss: 2.1150412559509277
Generator Loss: [array(2.1150413, dtype=float32), array(2.1150413, dtype=f
loat32), array(0.20482673, dtype=float32)]
```



```
Epoch: 200
Discriminator Loss: 2.8626105785369873
Generator Loss: [array(2.8626106, dtype=float32), array(2.8626106, dtype=f
loat32), array(0.20747824, dtype=float32)]
```



Practical 10

10.1 Develop an API to deploy your model and perform predictions

```
# Required Libraries

!pip install pyngrok flask scikit-learn

# Importing Libraries

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split from
sklearn.ensemble import RandomForestClassifier import pickle

from flask import Flask, request, jsonify

from pyngrok import ngrok


# Load dataset

iris = load_iris()
X, y = iris.data, iris.target

# Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=42)

# Train a model

model = RandomForestClassifier()
model.fit(X_train, y_train)

# Save the model

with open('model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)

# Load the model

with open('model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)

# Create Flask app app =
Flask(_name_) port =
"5000"
```

```python
@app.route('/')
def home():

    return "Welcome to the Iris Prediction API! Use the /predict endpoint
to make predictions."

@app.route('/predict', methods=['POST'])
def predict():

    data = request.json
    features = data.get('features')
```

```
    # Ensure the features are in the correct format
    if not features or len(features) != 4:  # Assuming 4 features for iris dataset
        return jsonify({'error': 'Invalid input format. Please provide 4 f
eatures.'}), 400

    try:
        prediction = model.predict([features])  # Wrap features in a list
to create 2D array
        return jsonify({'prediction': int(prediction[0])})  # Convert pred iction to int
    except Exception as e:
        return jsonify({'error': str(e)}), 500

# Start ngrok and print the public URL
ngrok.set_auth_token("api_auth_token") public_url =
ngrok.connect(port).public_url print("Public URL:", public_url)


# Run the Flask app
if __name__ == '__main__':
    app.run(port=port)
```

```
Requirement already satisfied: pyngrok in /usr/local/lib/python3.10/dist-p
ackages (7.2.0)
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-pac
kages (2.2.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/d
ist-packages (1.5.2)
Requirement already satisfied: PyYAML>=5.1 in /usr/local/lib/python3.10/di
st-packages (from pyngrok) (6.0.2)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.1
0/dist-packages (from flask) (3.0.4)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/di
st-packages (from flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3
.10/dist-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dis
t-packages (from flask) (8.1.7)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/
dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/d
ist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/
dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/pyth
on3.10/dist-packages (from scikit-learn) (3.5.0)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.1
0/dist-packages (from Jinja2>=3.0->flask) (2.1.5)
Public URL: https://2f62-49-43-24-101.ngrok-free.app
```

```
 * Serving Flask app '__main__'
 * Debug mode: off

INFO:werkzeug:WARNING: This is a development server. Do not use it in a pr
oduction deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug:127.0.0.1 - - [24/Oct/2024 07:50:42] "POST /predict HTTP/1.1
" 200 -
```