# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**ABUBAKAR MOHAMMEDSHAFEE MATTE (1BM22CS010)**

#### in partial fulfillment for the award of the degree of

**BACHELOR OF ENGINEERING**

***in***

## COMPUTER SCIENCE AND ENGINEERING

****

**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

## BENGALURU-560019

## Sep-2024 to Jan-2025

**B.M.S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**

****

##### CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by **Abubakar Mohammedshafee Matte(1BM22CS029),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Dr. Kavitha Sooda Professor & HOD

Department of CSE, BMSCE

Seema Patil

Assistant Professor Department of CSE, BMSCE

# Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 21-2-2025 | Write a python program to import and export data using Pandas library functions | 2-10 |
| 2 | 3-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 11-16 |
| 3 | 10-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 17-24 |
| 4 | 17-3-2025 | Build Logistic Regression Model for a given dataset | 25-32 |
| 5 | 24-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 33-40 |
| 6 | 7-4-2025 | Build KNN Classification model for a given dataset. | 41-49 |
| 7 | 21-4-2025 | Build Support vector machine model for a given dataset | 50-57 |
| 8 | 5-5-2025 | Implement Random forest ensemble method on a given dataset. | 58-63 |
| 9 | 5-5-2025 | Implement Boosting ensemble method on a given dataset. | 64-69 |
| 10 | 12-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 70-77 |
| 11 | 12-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 78-87 |

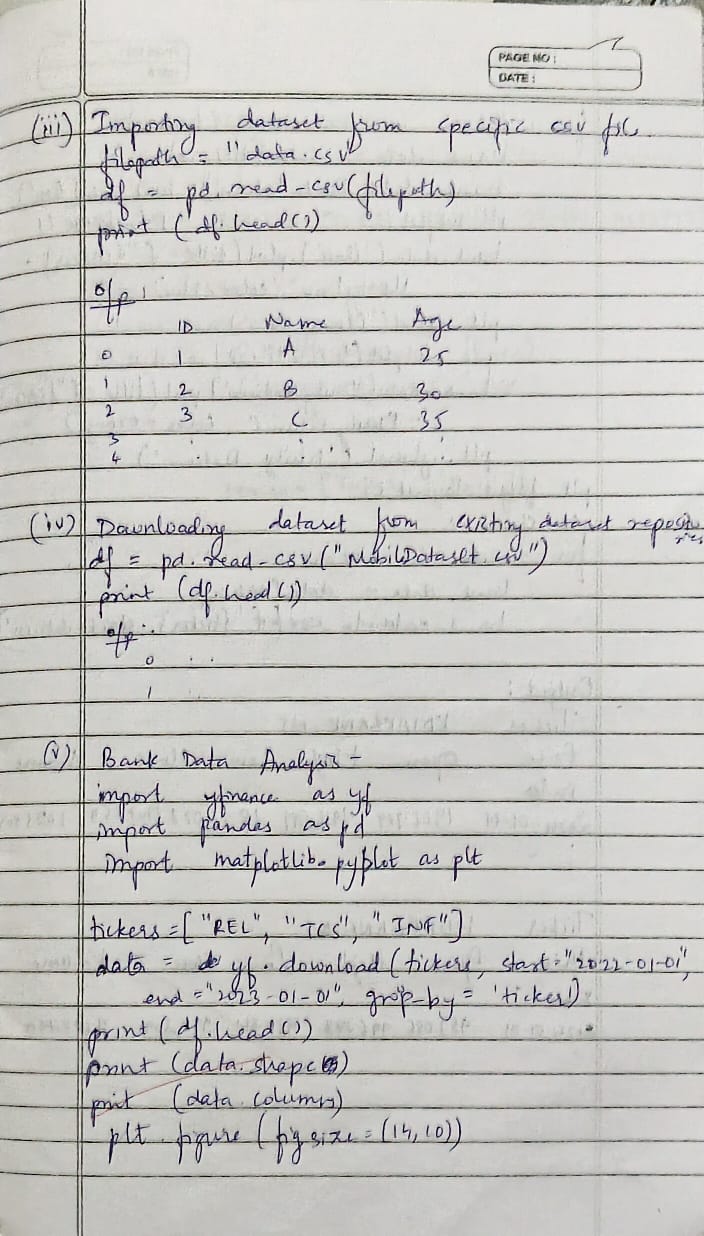
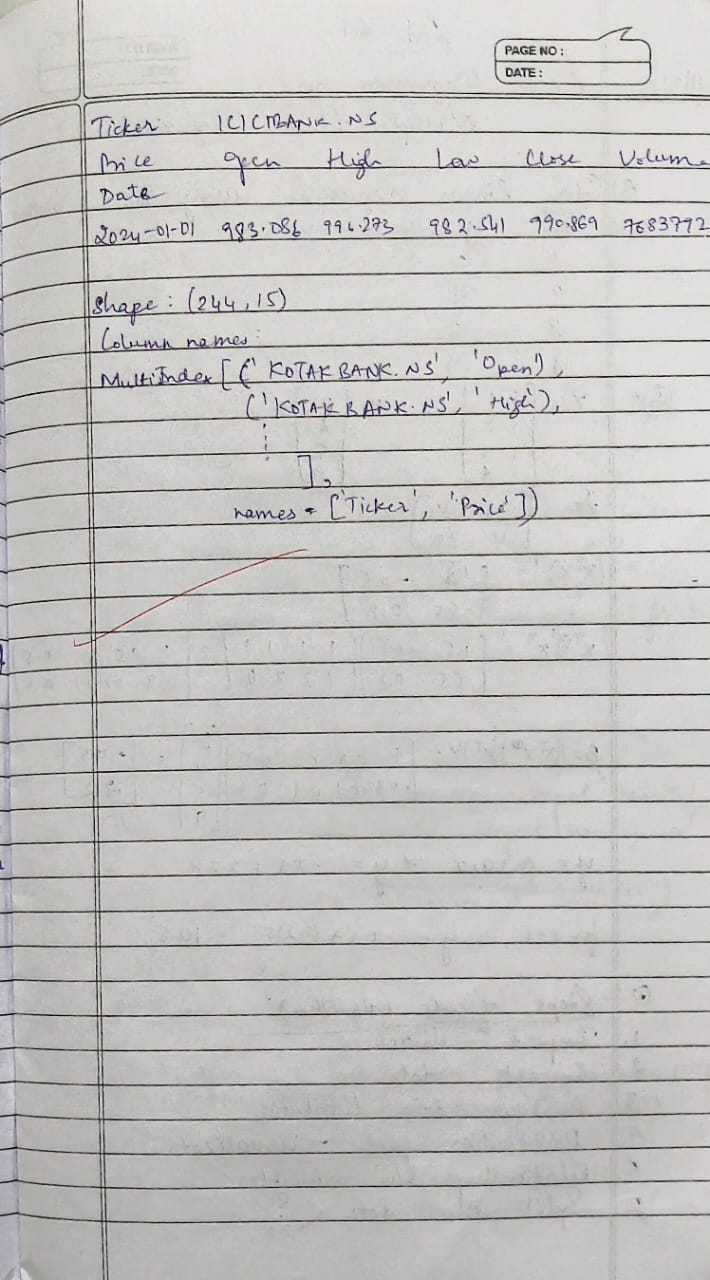
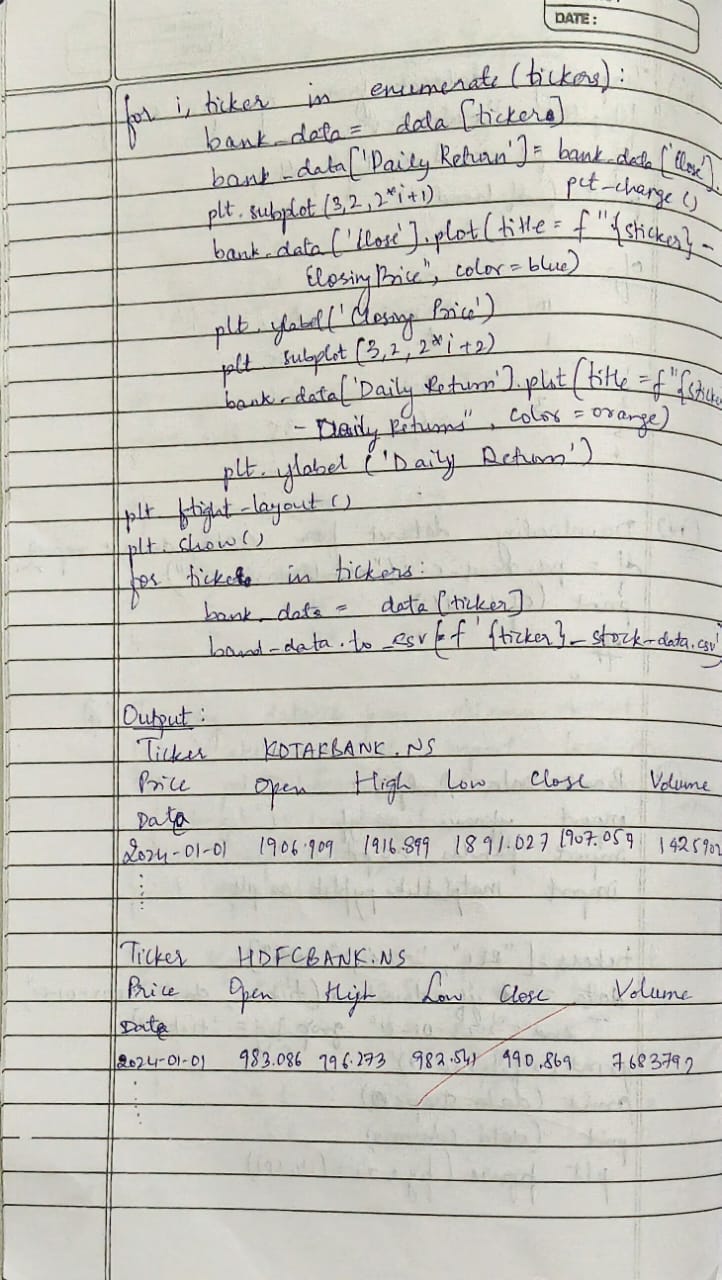
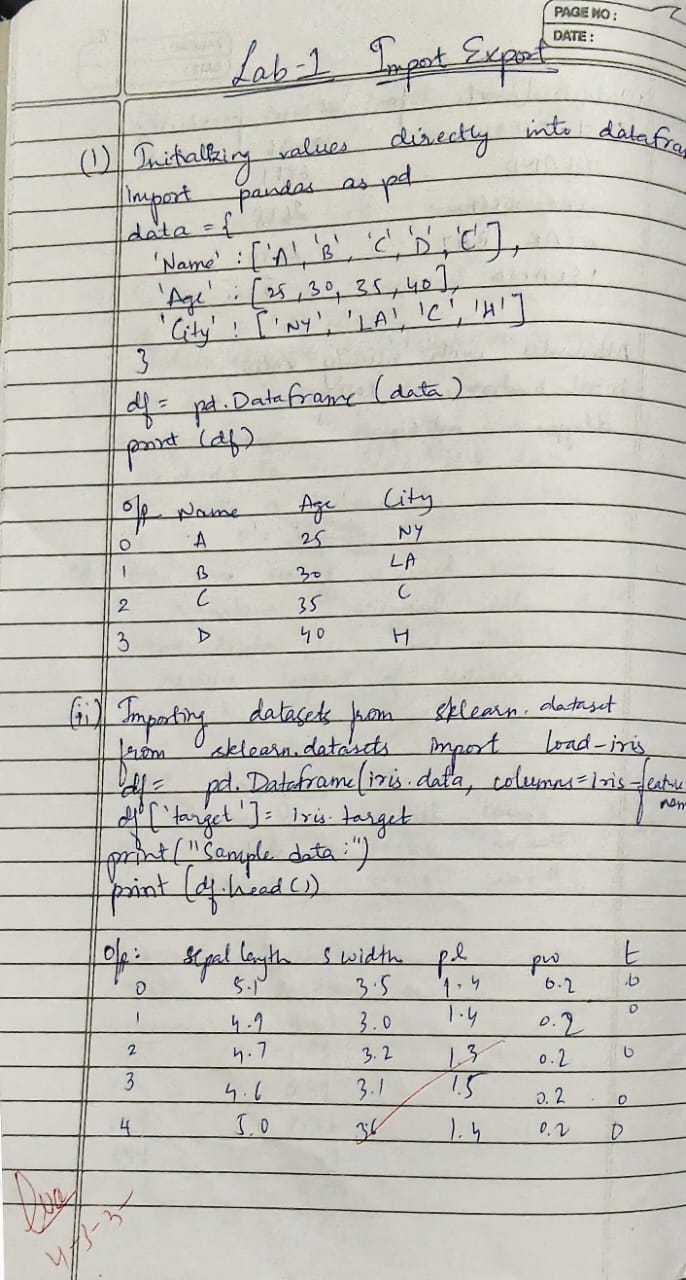
Github Link:

<https://github.com/AbubakarMatte/6A-ML-Lab-Batch1>

**Program 1**

**Write a python program to import and export data using Pandas library functions**

**Screenshot:**



**Code:**

import pandas as pd

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 35, 40],

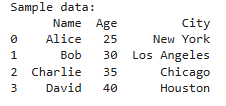
'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']

}

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())



from sklearn.datasets import load\_iris

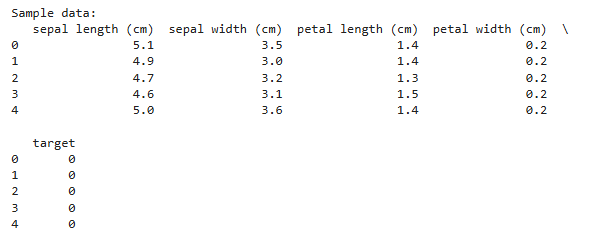
iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df['target'] = iris.target

print("Sample data:")

print(df.head())



from google.colab import files

uploaded = files.upload()

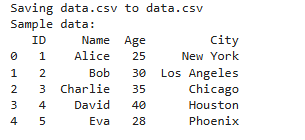
file\_path = 'data.csv' *# Ensure the file exists in the same directory*

df = pd.read\_csv(file\_path)

print("Sample data:")

print(df.head())

print("\n")



import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]

data = yf.download(tickers, start="2022-10-01", end="2023-10-01", group\_by='ticker')

print("First 5 rows of the dataset:")

print(data.head())

print("\nShape of the dataset:")

print(data.shape)

print("\nColumn names:")

print(data.columns)

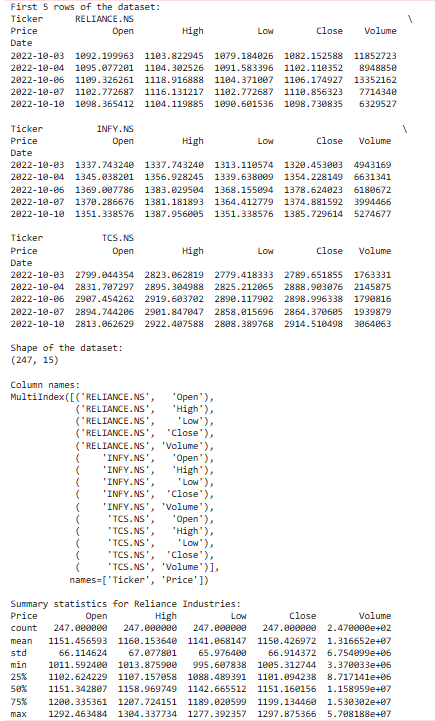
reliance\_data = data['RELIANCE.NS']

print("\nSummary statistics for Reliance Industries:")

print(reliance\_data.describe())

reliance\_data['Daily Return'] = reliance\_data['Close'].pct\_change()

reliance\_data['Daily Return'] = reliance\_data['Close'].pct\_change()



plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

reliance\_data['Close'].plot(title="Reliance Industries - Closing Price")

plt.subplot(2, 1, 2)

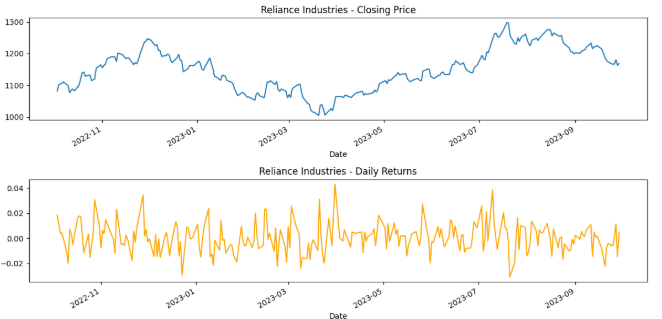
reliance\_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')

plt.tight\_layout()

plt.show()

reliance\_data.to\_csv('reliance\_stock\_data.csv')

print("\nReliance stock data saved to 'reliance\_stock\_data.csv'.")



import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group\_by='ticker')

print("First 5 rows of the dataset:")

print(data.head())

print("\nShape of the dataset:")

print(data.shape)

print("\nColumn names:")

print(data.columns)

plt.figure(figsize=(14, 10))

for i, ticker in enumerate(tickers):

    bank\_data = data[ticker]

    bank\_data['Daily Return'] = bank\_data['Close'].pct\_change()

    plt.subplot(3, 2, 2\*i+1)

    bank\_data['Close'].plot(title=f"{ticker} - Closing Price", color='blue')

    plt.ylabel('Closing Price')

    plt.subplot(3, 2, 2\*i+2)

    bank\_data['Daily Return'].plot(title=f"{ticker} - Daily Returns", color='orange')

    plt.ylabel('Daily Return')

plt.tight\_layout()

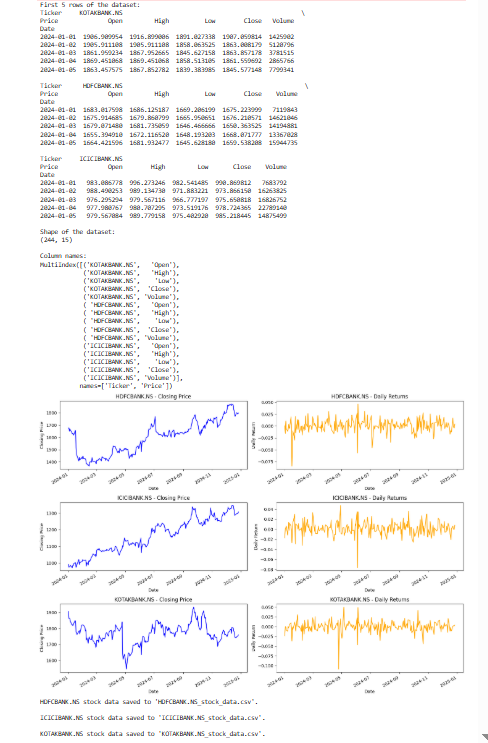
plt.show()

for ticker in tickers:

    bank\_data = data[ticker]

    bank\_data.to\_csv(f'{ticker}\_stock\_data.csv')

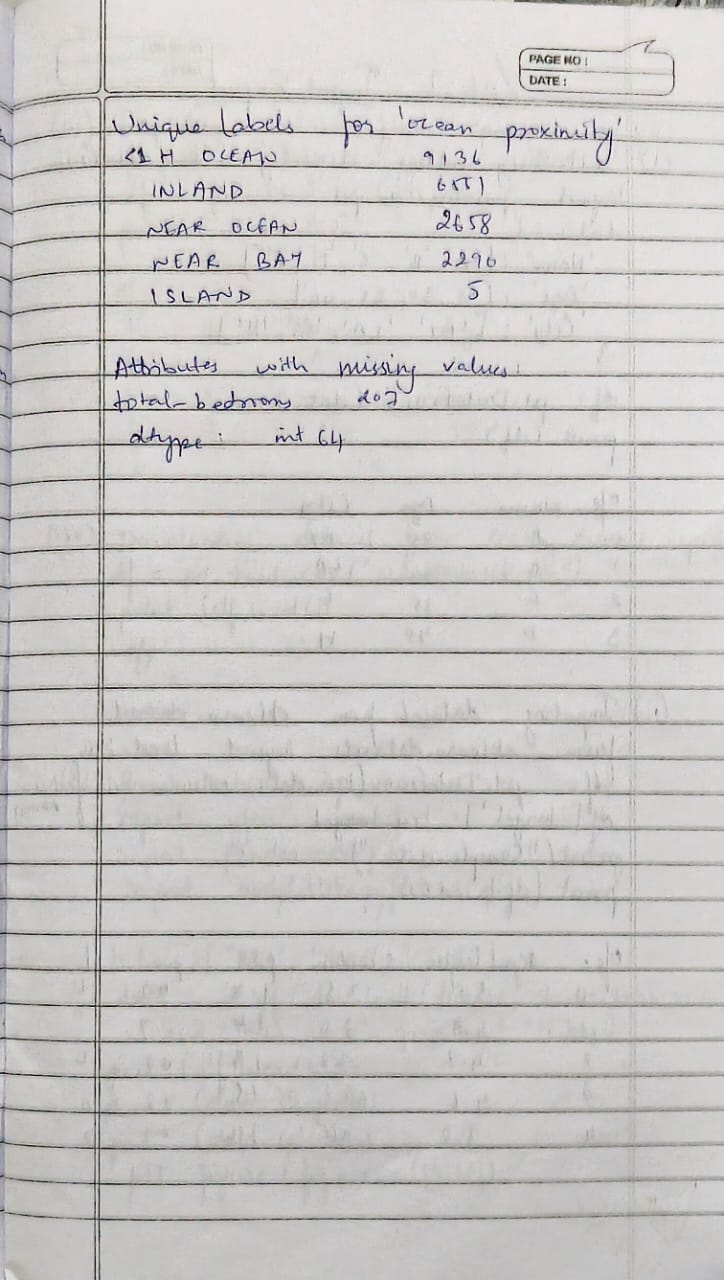
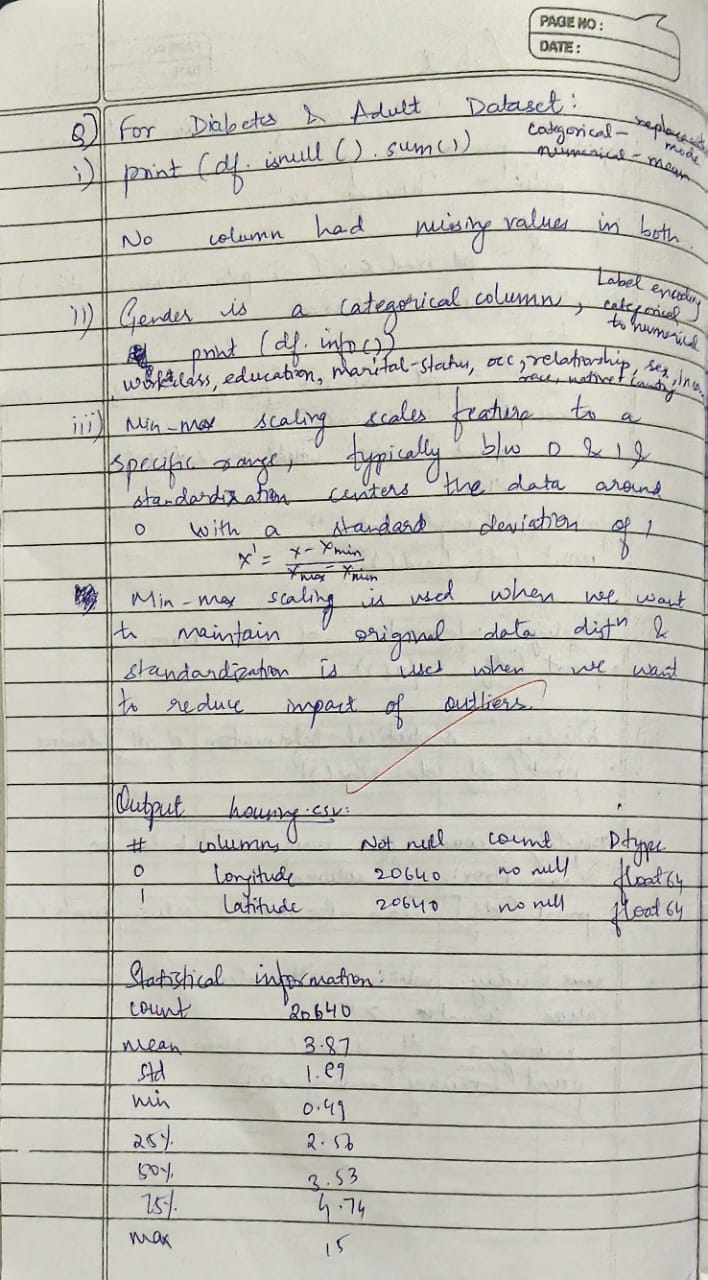
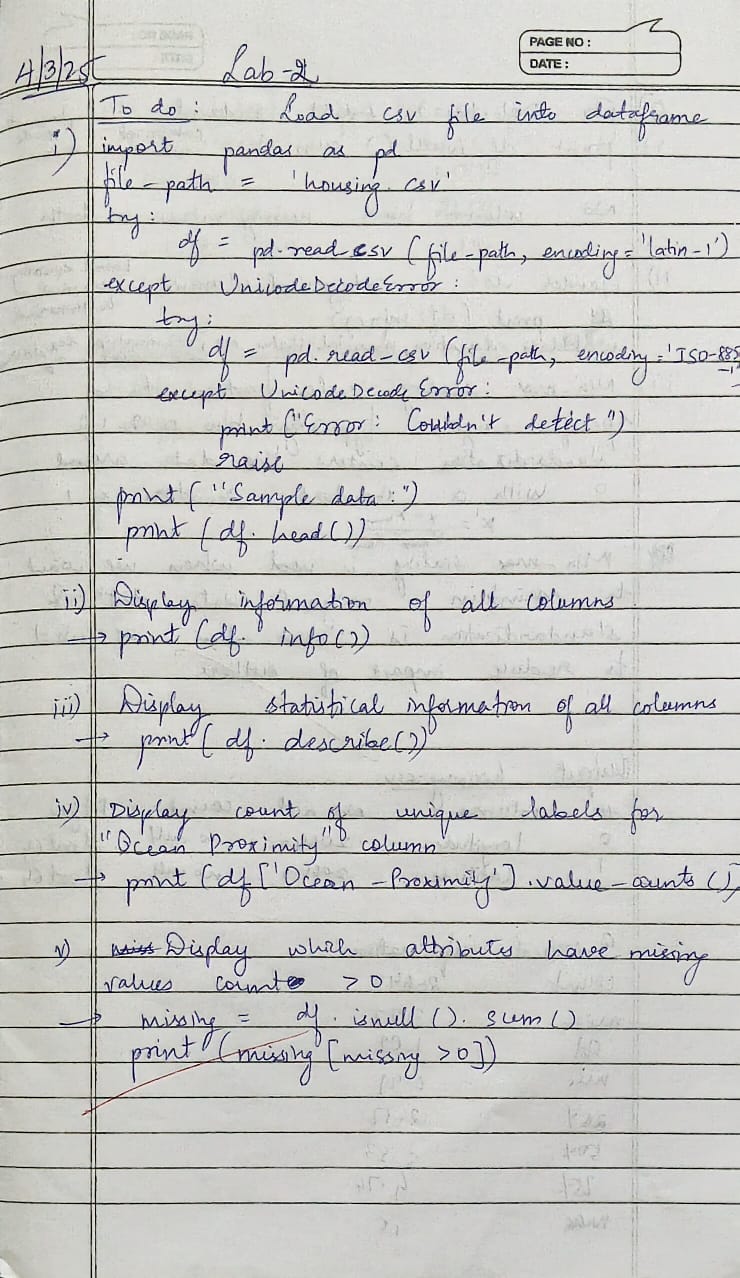
    print(f"\n{ticker} stock data saved to '{ticker}\_stock\_data.csv'.")



**Program 2**

**Demonstrate various data pre-processing techniques for a given dataset**

**Screenshot:**



**Code:**

import pandas as pd

file\_path = 'housing.csv'

df = pd.read\_csv(file\_path)

print("\nDataset Information:")

print(df.info())

print("\nStatistical Information of Numerical Columns:")

print(df.describe())

print("\nUnique Labels Count for 'Ocean Proximity' column:")

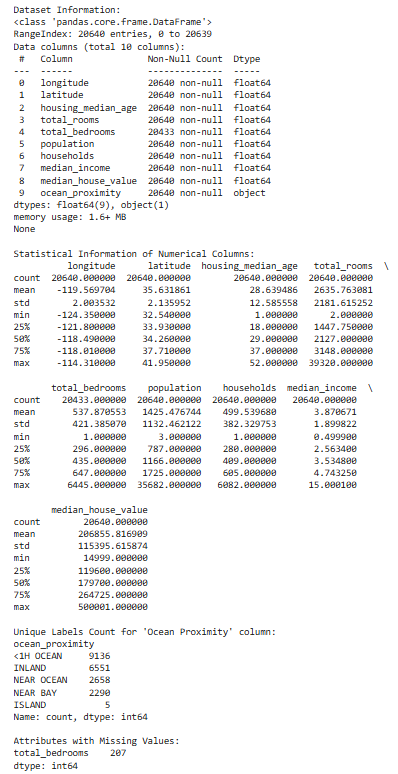
print(df['ocean\_proximity'].value\_counts())

print("\nAttributes with Missing Values:")

missing\_values = df.isnull().sum()

columns\_with\_missing\_values = missing\_values[missing\_values > 0]

print(columns\_with\_missing\_values)



import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import MinMaxScaler, StandardScaler

file\_path = "diabetes.csv"

df = pd.read\_csv(file\_path)

df\_numeric = df.select\_dtypes(include=['number']).copy() *# Select only numeric columns*

imputer = SimpleImputer(strategy="mean")

df\_numeric.iloc[:, :] = imputer.fit\_transform(df\_numeric)

df[df\_numeric.columns] = df\_numeric

Q1 = df\_numeric.quantile(0.25) *# Only compute quartiles on numeric data*

Q3 = df\_numeric.quantile(0.75) *# Only compute quartiles on numeric data*

IQR = Q3 - Q1

df = df[~((df\_numeric < (Q1 - 1.5 \* IQR)) | (df\_numeric > (Q3 + 1.5 \* IQR))).any(axis=1)]

min\_max\_scaler = MinMaxScaler()

df\_minmax = pd.DataFrame(min\_max\_scaler.fit\_transform(df\_numeric), columns=df\_numeric.columns) *# Only transform the numeric columns*

standard\_scaler = StandardScaler()

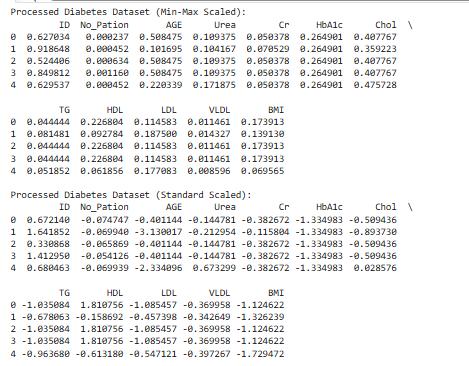
df\_standard = pd.DataFrame(standard\_scaler.fit\_transform(df\_numeric), columns=df\_numeric.columns)  *# Only transform the numeric columns*

print("\nProcessed Diabetes Dataset (Min-Max Scaled):")

print(df\_minmax.head())

print("\nProcessed Diabetes Dataset (Standard Scaled):")

print(df\_standard.head())



import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder

file\_path = "adult.csv"  *# Update the path if needed*

df = pd.read\_csv(file\_path)

df.replace("?", np.nan, inplace=True)

num\_imputer = SimpleImputer(strategy="mean")

df[df.select\_dtypes(include=['number']).columns] = num\_imputer.fit\_transform(df.select\_dtypes(include=['number']))

cat\_imputer = SimpleImputer(strategy="most\_frequent")

df[df.select\_dtypes(include=['object']).columns] = cat\_imputer.fit\_transform(df.select\_dtypes(include=['object']))

label\_encoders = {}

for col in df.select\_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

df = df[~((df < (Q1 - 1.5 \* IQR)) | (df > (Q3 + 1.5 \* IQR))).any(axis=1)]

min\_max\_scaler = MinMaxScaler()

df\_minmax = pd.DataFrame(min\_max\_scaler.fit\_transform(df), columns=df.columns)

standard\_scaler = StandardScaler()

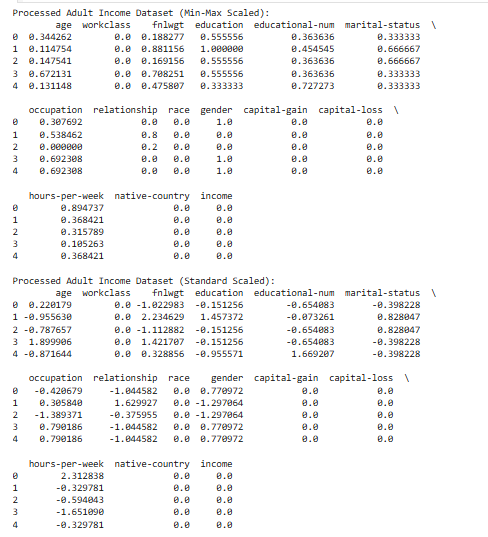
df\_standard = pd.DataFrame(standard\_scaler.fit\_transform(df), columns=df.columns)

print("\nProcessed Adult Income Dataset (Min-Max Scaled):")

print(df\_minmax.head())

print("\nProcessed Adult Income Dataset (Standard Scaled):")

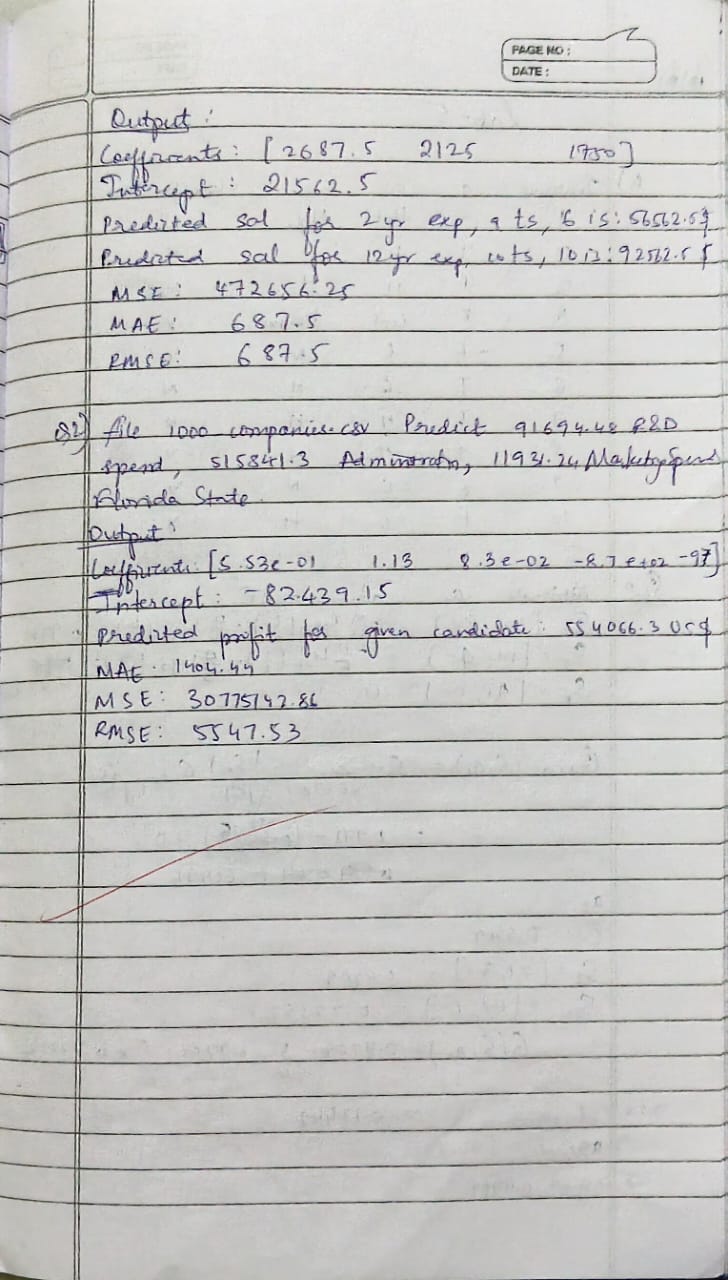
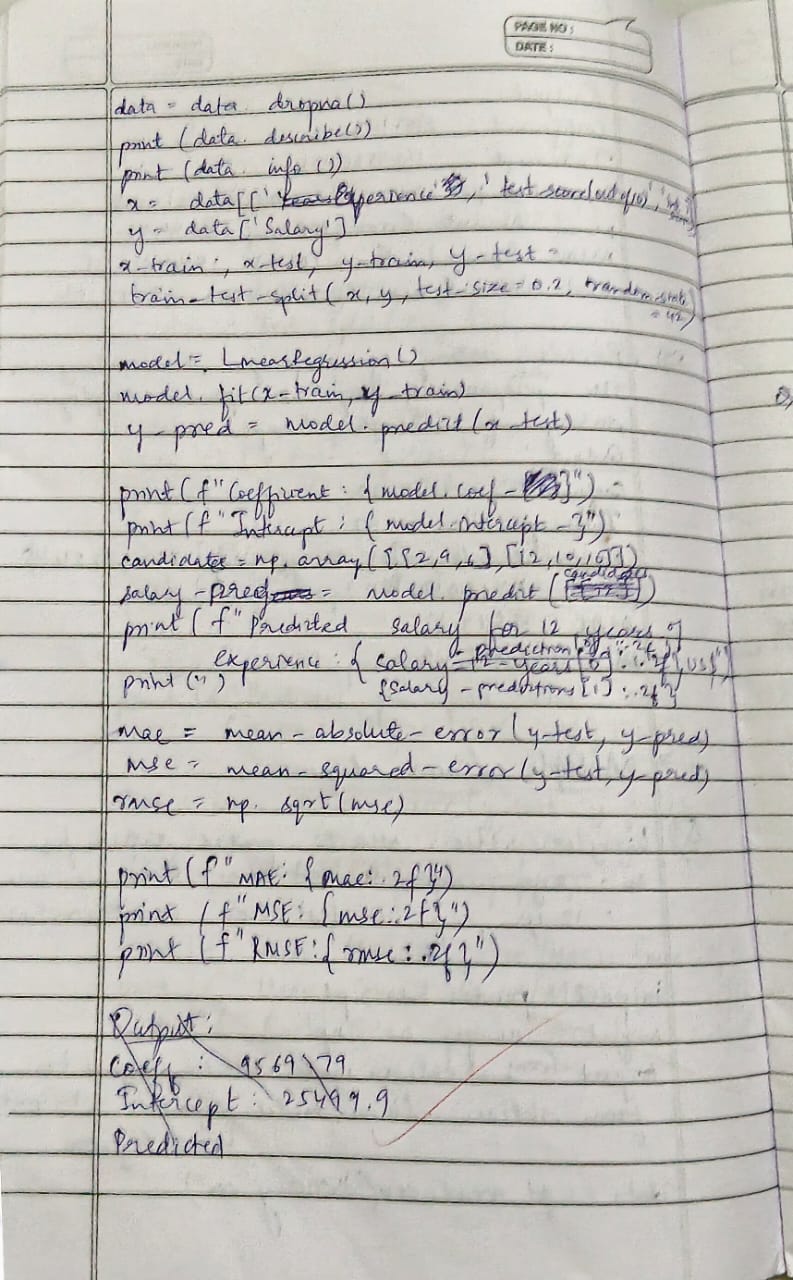
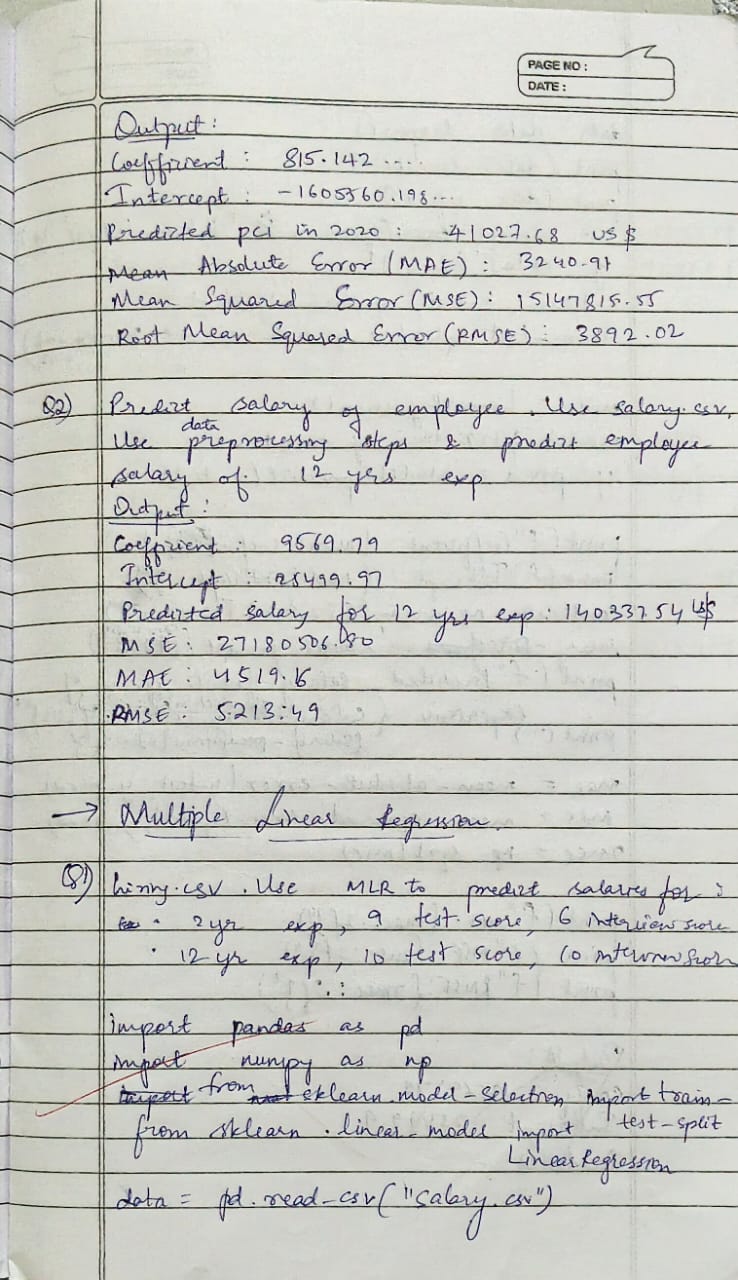
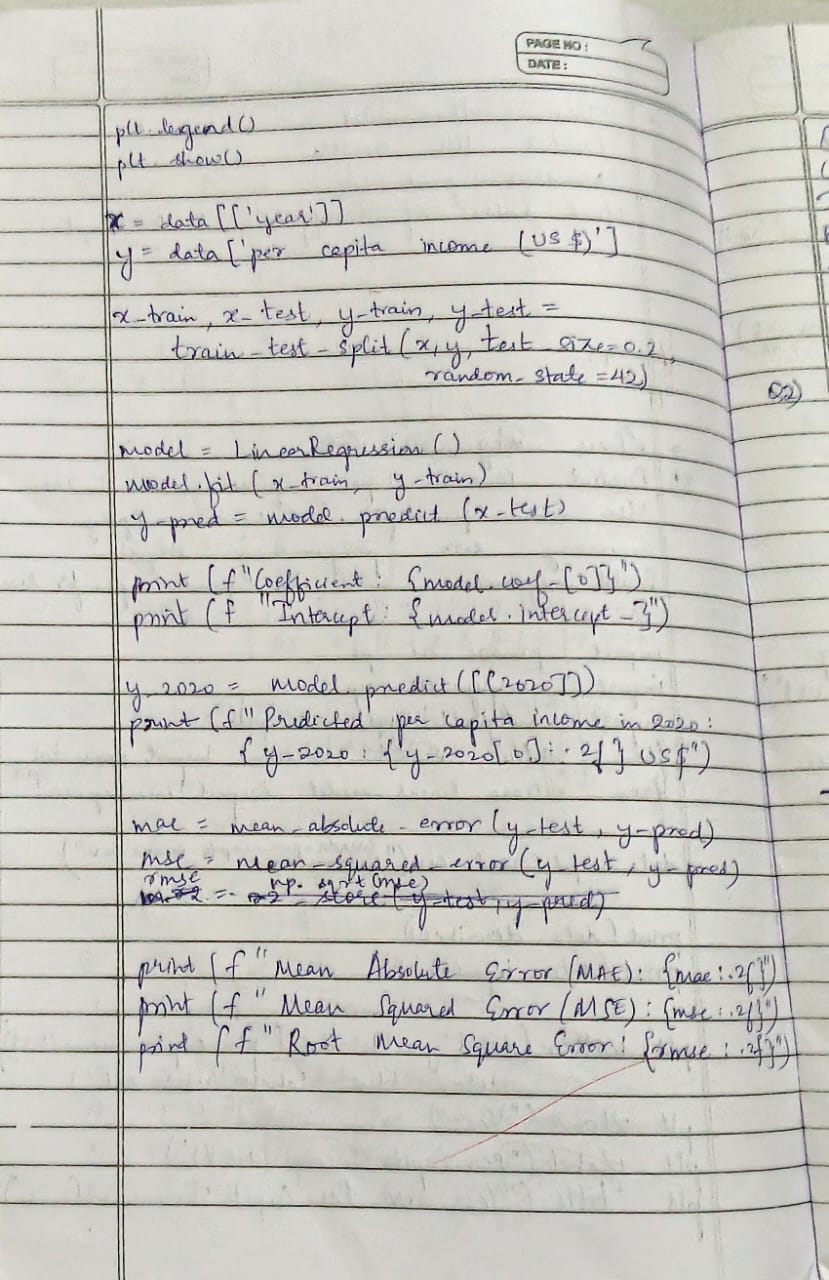
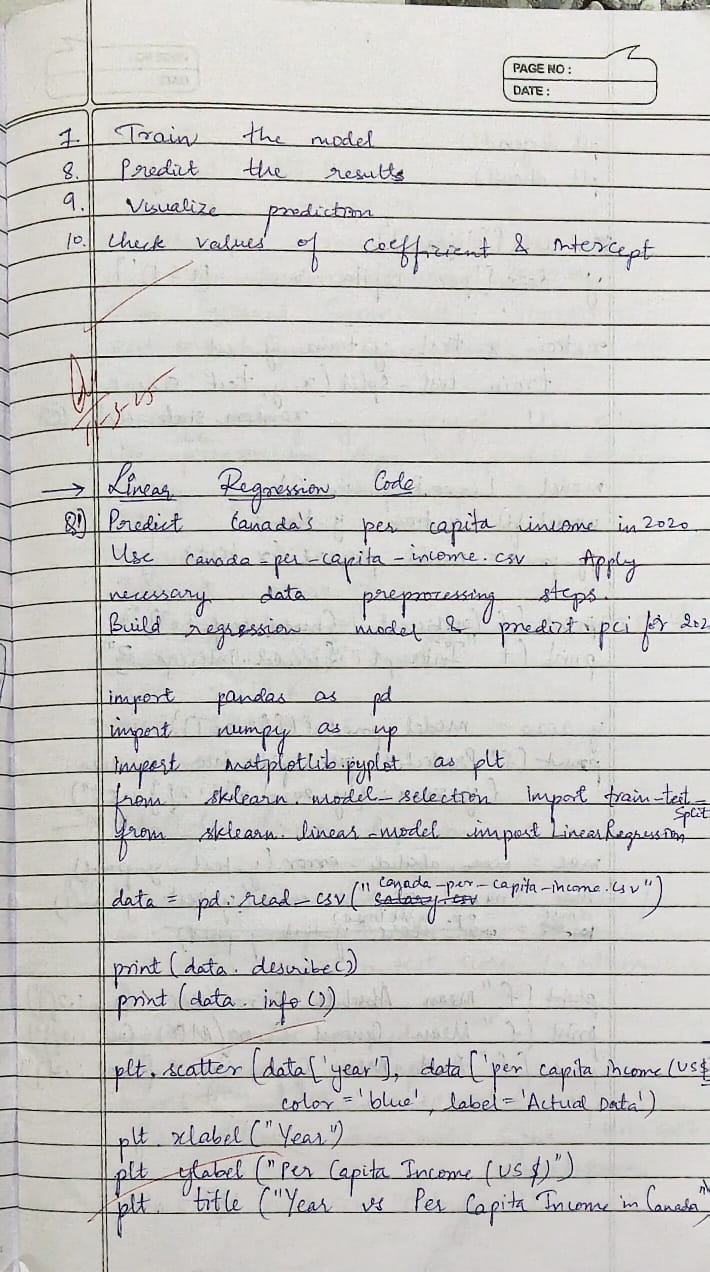
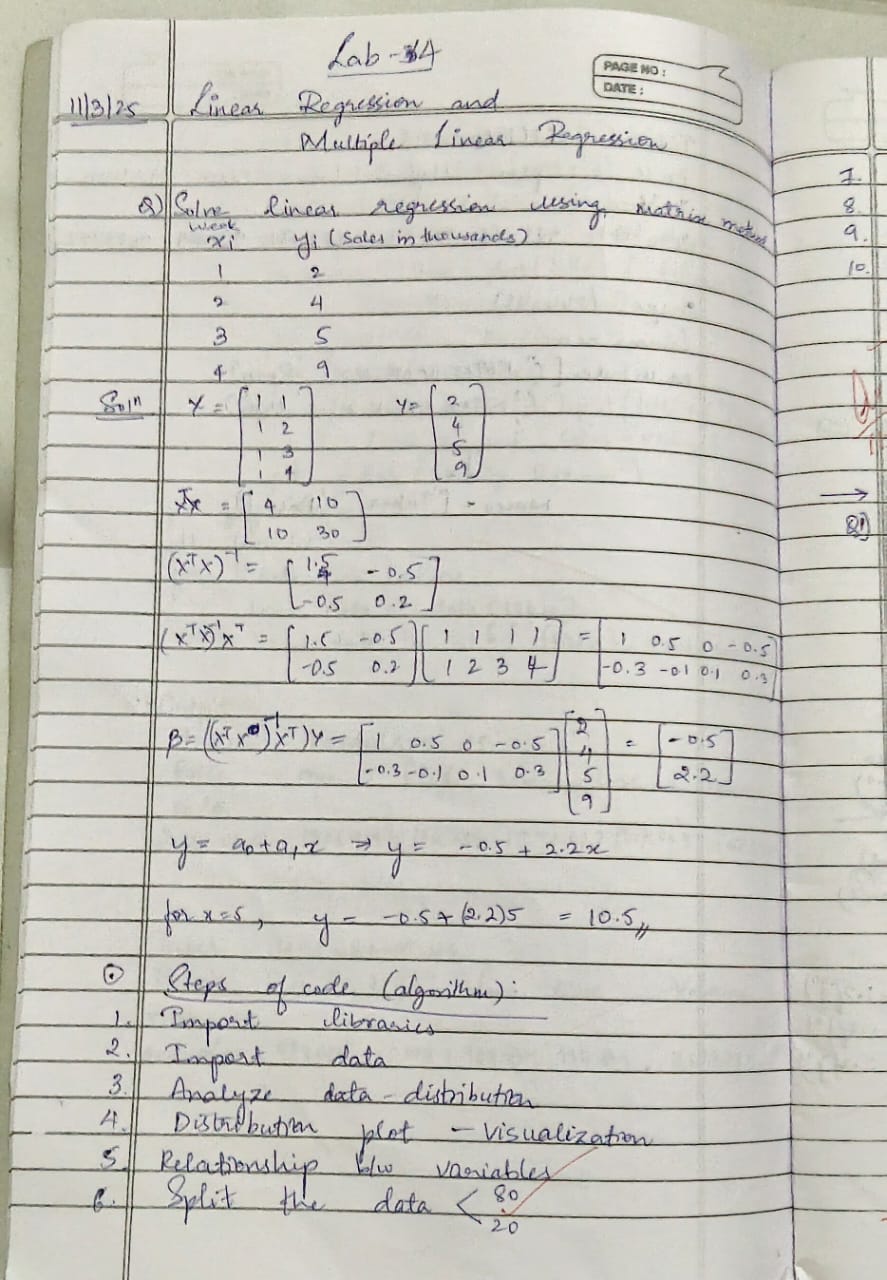
print(df\_standard.head())



**Program 3**

**Implement Linear and Multi-Linear Regression algorithm using appropriate dataset**

**Screenshot:**



**Code:**

**Linear Regression:**

import pandas as pd

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

df = pd.read\_csv('canada\_per\_capita\_income.csv')

reg = linear\_model.LinearRegression()

reg.fit(df[['year']], df['per capita income (US$)'])

print(f"Coefficient: {reg.coef\_}")

print(f"Intercept: {reg.intercept\_}")

predicted\_income = reg.predict([[2020]])

print(f"Predicted per capita income for 2020: ${predicted\_income[0]:,.2f}")

plt.scatter(df['year'], df['per capita income (US$)'], color='red')

plt.plot(df['year'], reg.predict(df[['year']]), color='blue')

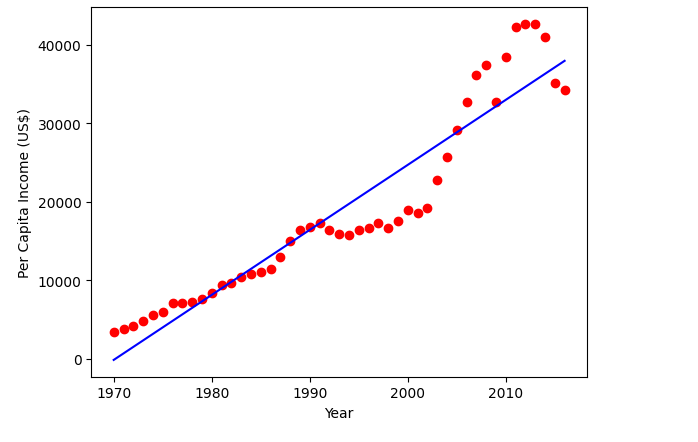
plt.xlabel('Year')

plt.ylabel('Per Capita Income (US$)')

plt.show()

print(results.head())





**Multiple Regression:**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

df\_hiring = pd.read\_csv('hiring.csv')

print("Original Data:")

print(df\_hiring.head())

df\_hiring['experience'] = df\_hiring['experience'].replace({

    'five': 5,

    'four': 4,

    'three': 3

})

df\_hiring['experience'] = pd.to\_numeric(df\_hiring['experience'], errors='coerce')

df\_hiring['test\_score(out of 10)'] = pd.to\_numeric(df\_hiring['test\_score(out of 10)'], errors='coerce')

df\_hiring['interview\_score(out of 10)'] = pd.to\_numeric(df\_hiring['interview\_score(out of 10)'], errors='coerce')

df\_hiring['salary($)'] = pd.to\_numeric(df\_hiring['salary($)'], errors='coerce')

df\_hiring['experience'] = df\_hiring['experience'].fillna(df\_hiring['experience'].median())

df\_hiring['test\_score(out of 10)'] = df\_hiring['test\_score(out of 10)'].fillna(df\_hiring['test\_score(out of 10)'].median())

df\_hiring['interview\_score(out of 10)'] = df\_hiring['interview\_score(out of 10)'].fillna(df\_hiring['interview\_score(out of 10)'].median())

df\_hiring['salary($)'] = df\_hiring['salary($)'].fillna(df\_hiring['salary($)'].median())

X\_hiring = df\_hiring[['experience', 'test\_score(out of 10)', 'interview\_score(out of 10)']]

y\_hiring = df\_hiring['salary($)']

model\_hiring = LinearRegression()

model\_hiring.fit(X\_hiring, y\_hiring)

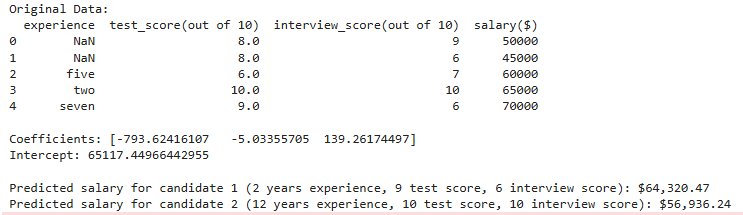
print(f"\nCoefficients: {model\_hiring.coef\_}")

print(f"Intercept: {model\_hiring.intercept\_}")

predictions\_hiring = model\_hiring.predict([[2, 9, 6], [12, 10, 10]])

print(f"\nPredicted salary for candidate 1 (2 years experience, 9 test score, 6 interview score): ${predictions\_hiring[0]:,.2f}")

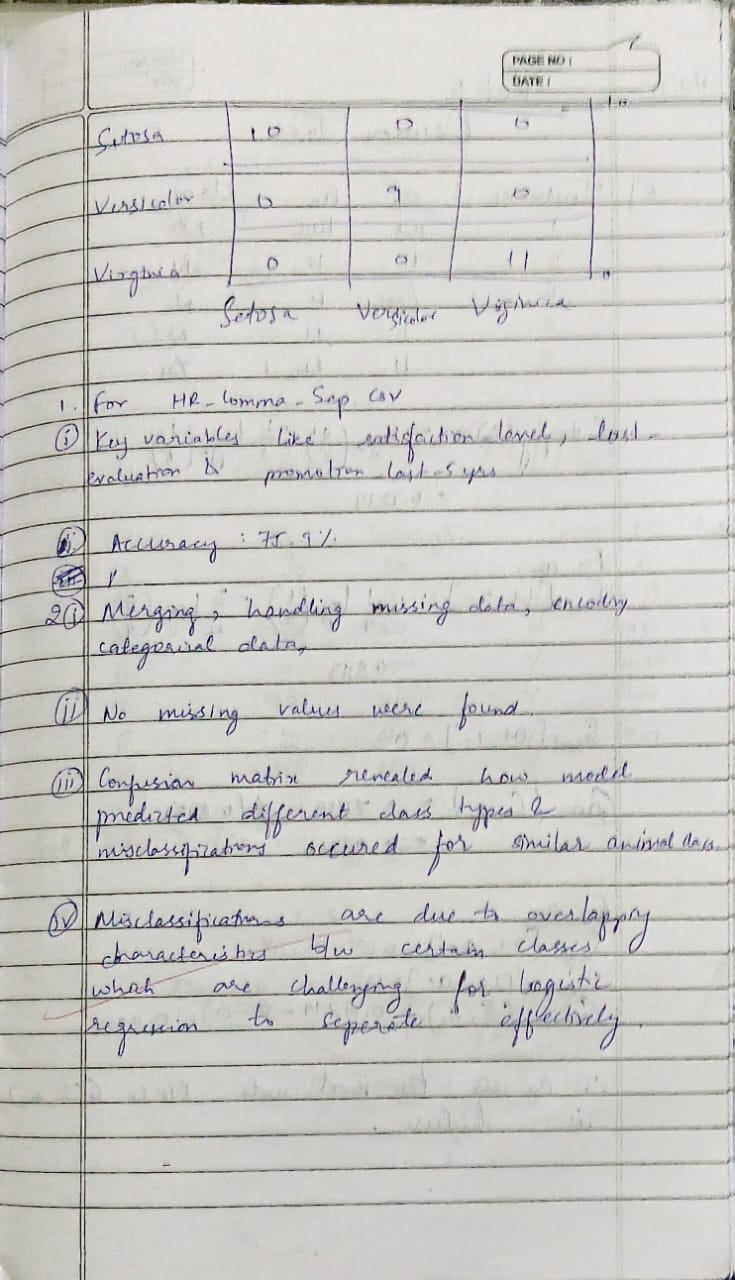
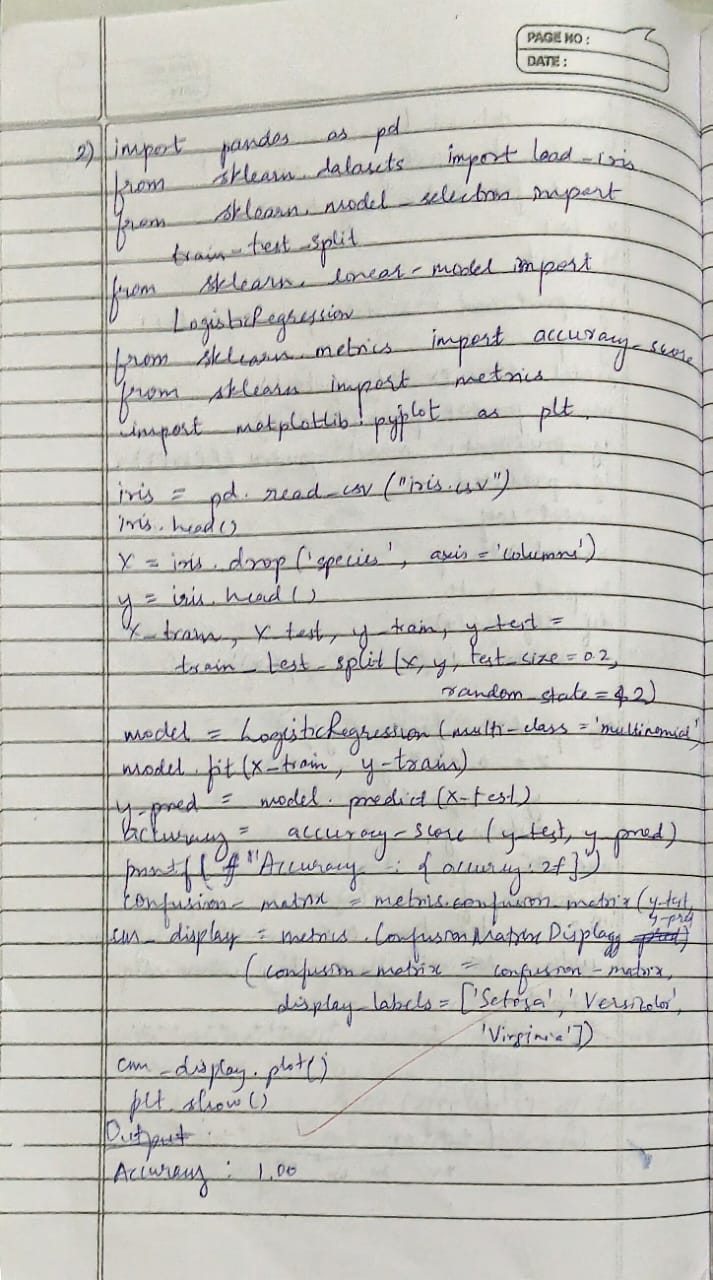
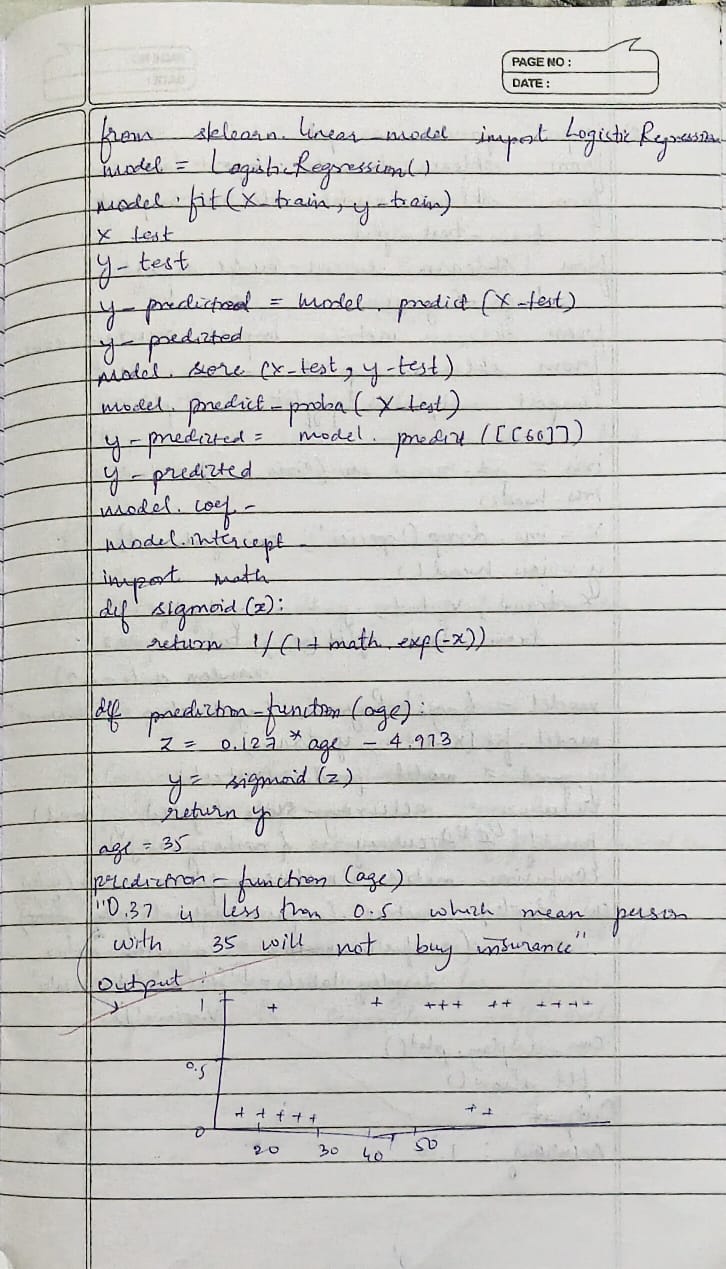
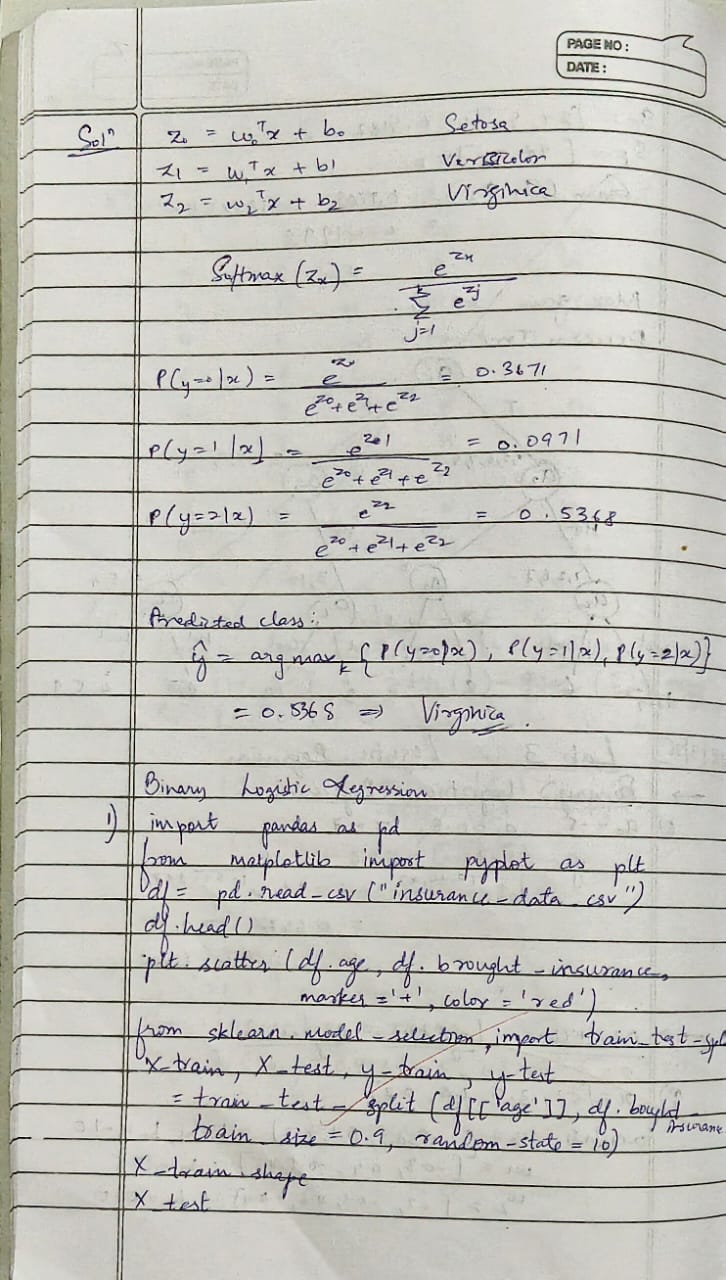
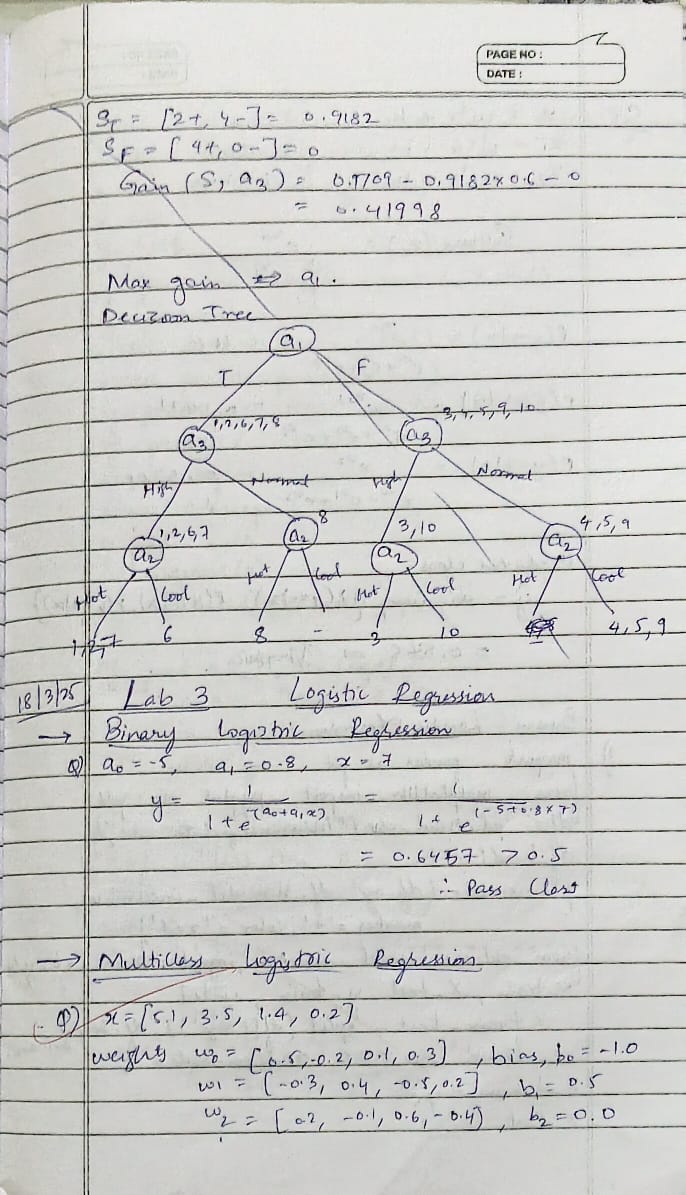
print(f"Predicted salary for candidate 2 (12 years experience, 10 test score, 10 interview score): ${predictions\_hiring[1]:,.2f}")



**Program 4**

**Build Logistic Regression Model for a given dataset**

**Screenshot:**



**Code:**

**Binary Logical Regression**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

hr\_data = pd.read\_csv("HR\_comma\_sep.csv")

print(hr\_data.head())

print("\nMissing values:\n", hr\_data.isnull().sum())

print("\nData Types:\n", hr\_data.dtypes)

print("\nUnique values in categorical columns:\n", hr\_data.select\_dtypes(include=['object']).nunique())

plt.figure(figsize=(8,5))

sns.countplot(x="salary", hue="left", data=hr\_data)

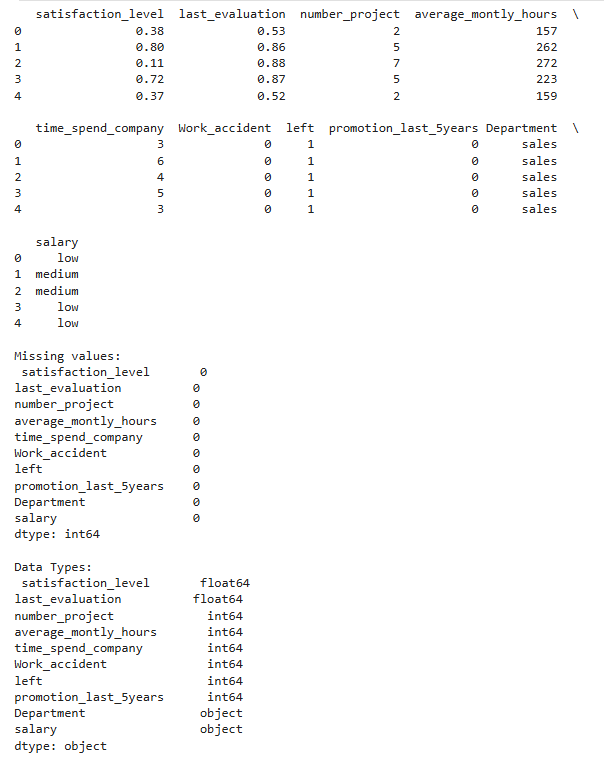
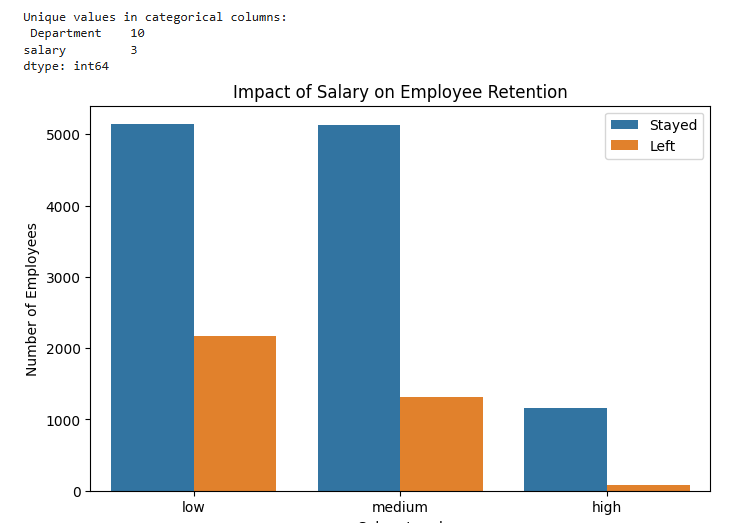
plt.title("Impact of Salary on Employee Retention")

plt.xlabel("Salary Level")

plt.ylabel("Number of Employees")

plt.legend(["Stayed", "Left"])

plt.show()

**Multi Logical classification**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

zoo\_data = pd.read\_csv("zoo-data.csv")

class\_types = pd.read\_csv("zoo-class-type.csv")

print("Zoo Data:\n", zoo\_data.head())

print("\nClass Type Data:\n", class\_types.head())

if "class\_type" not in zoo\_data.columns:

    print("\nError: 'class\_type' column is missing in zoo-data.csv. Available columns: ", zoo\_data.columns)

else:

    X = zoo\_data.drop(columns=["animal\_name", "class\_type"], errors="ignore")

    y = zoo\_data["class\_type"]

    print("\nUnique class types:", y.unique())

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    scaler = StandardScaler()

    X\_train = scaler.fit\_transform(X\_train)

    X\_test = scaler.transform(X\_test)

    model = LogisticRegression(multi\_class="multinomial", solver="lbfgs", max\_iter=200)

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    print(f"\nModel Accuracy: {accuracy:.4f}")

    cm = confusion\_matrix(y\_test, y\_pred)

    plt.figure(figsize=(8,6))

    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=y.unique(), yticklabels=y.unique())

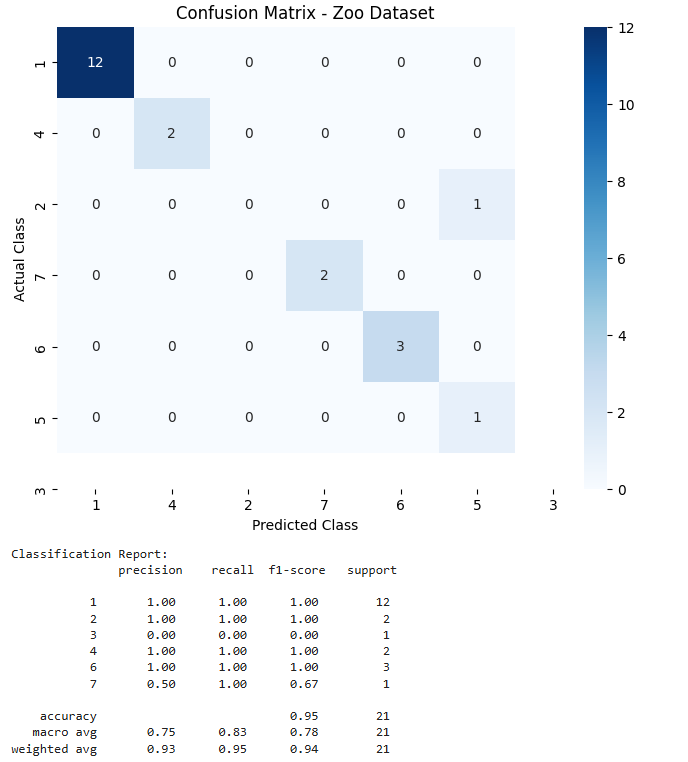
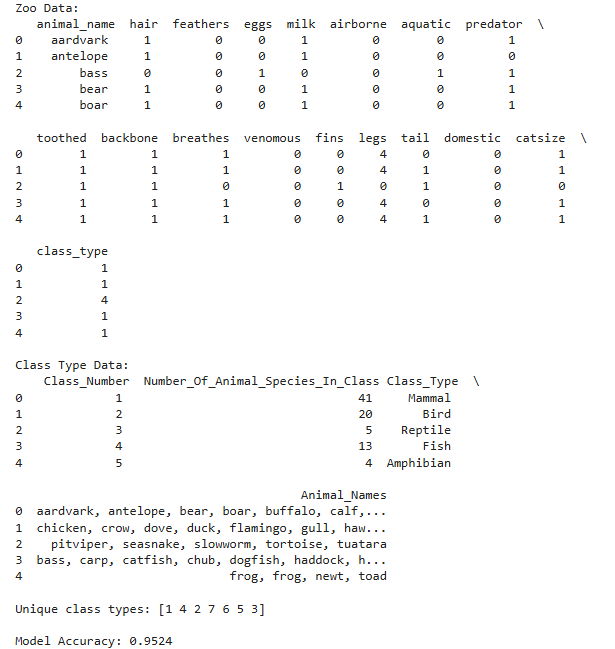
    plt.xlabel("Predicted Class")

    plt.ylabel("Actual Class")

    plt.title("Confusion Matrix - Zoo Dataset")

    plt.show()

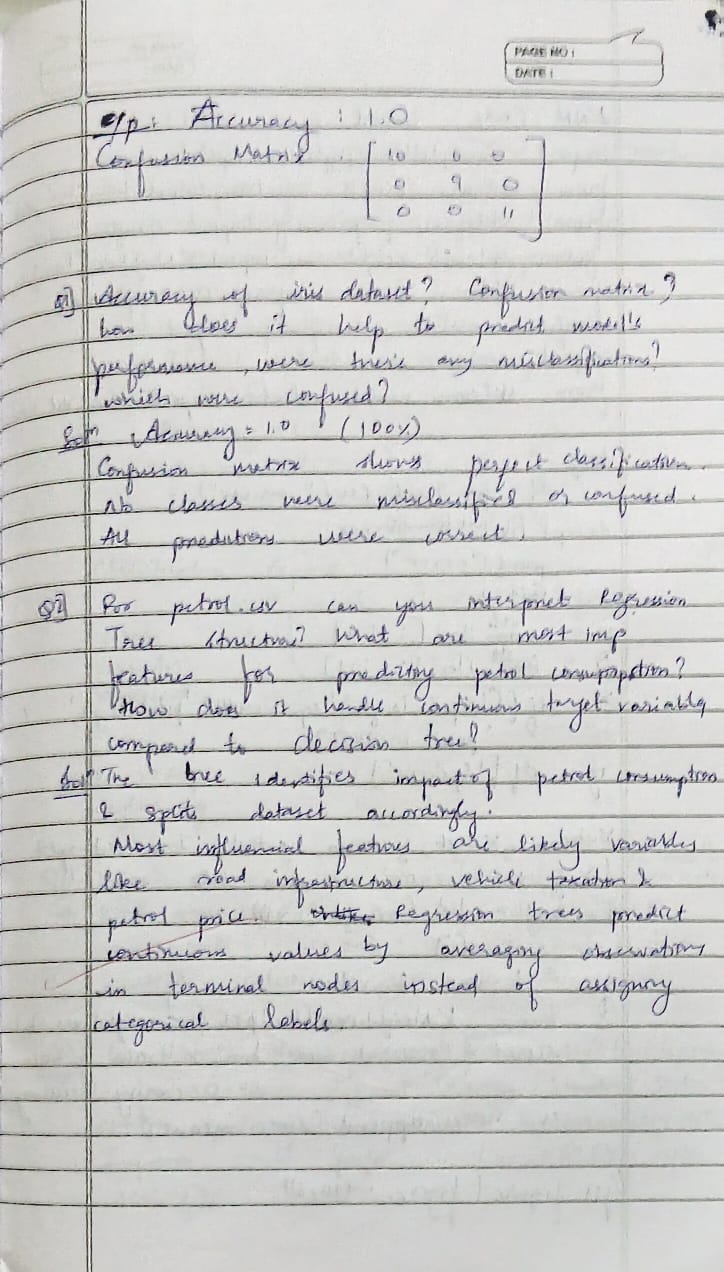
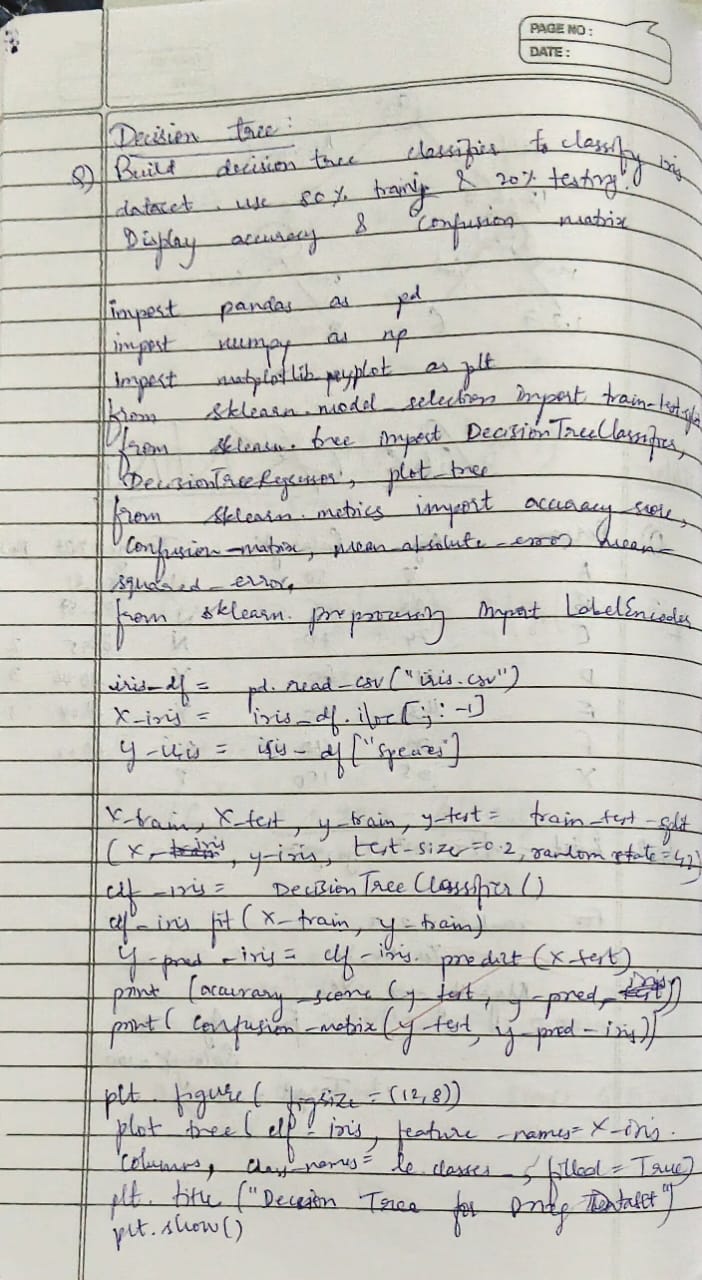
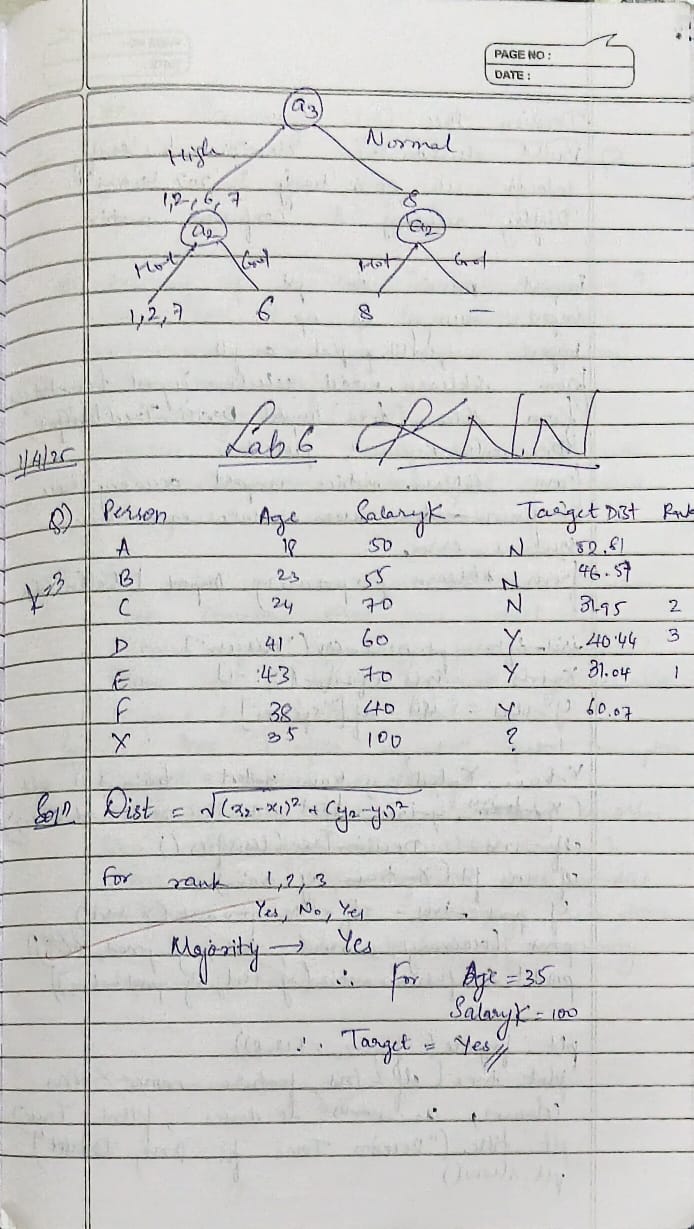
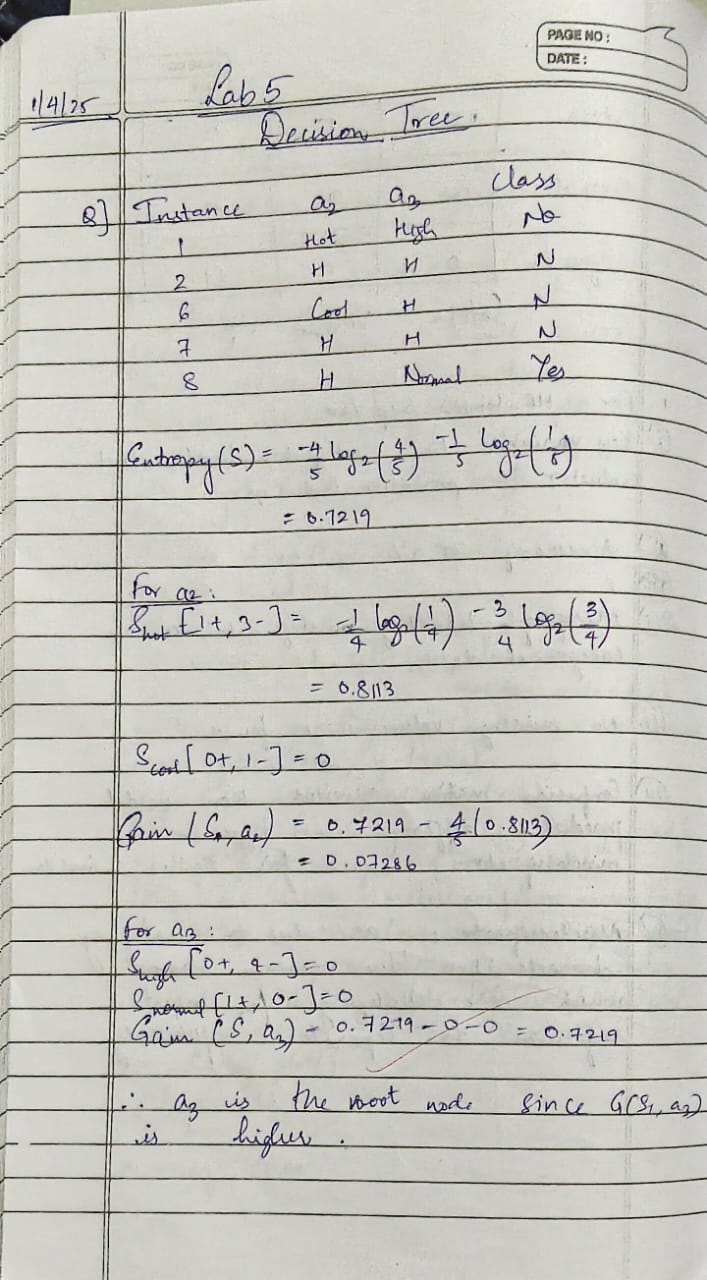
    print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))



**Program 5**

**Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.**

**Screenshot:**



**Code:**

Iris.csv

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv('iris.csv')

le = LabelEncoder()

df['species'] = le.fit\_transform(df['species'])

X = df.drop('species', axis=1)

y = df['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(criterion='entropy', 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}')

print(classification\_report(y\_test, y\_pred, target\_names=le.classes\_))

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6,4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes\_, yticklabels=le.classes\_)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

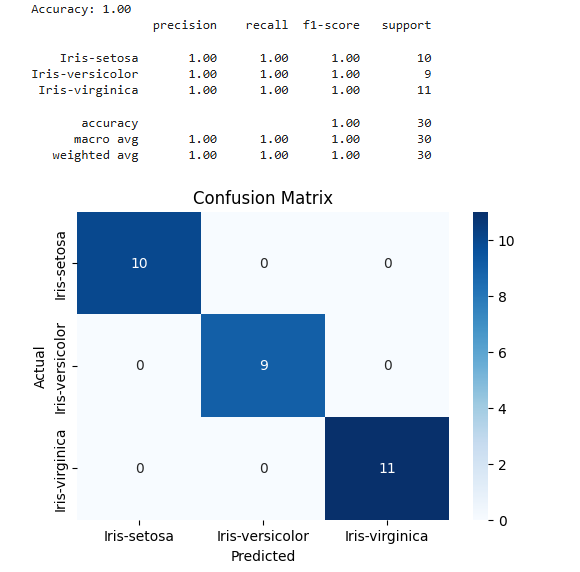
from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

plot\_tree(clf, filled=True, feature\_names=X.columns)

plt.show()



Drug.csv

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report

df = pd.read\_csv('drug.csv')

label\_encoders = {}

for column in df.columns:

    le = LabelEncoder()

    df[column] = le.fit\_transform(df[column])

    label\_encoders[column] = le

X = df.drop('Drug', axis=1)

y = df['Drug']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

*# Evaluate the classifier*

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

print(classification\_report(y\_test, y\_pred))

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6,4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes\_, yticklabels=le.classes\_)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

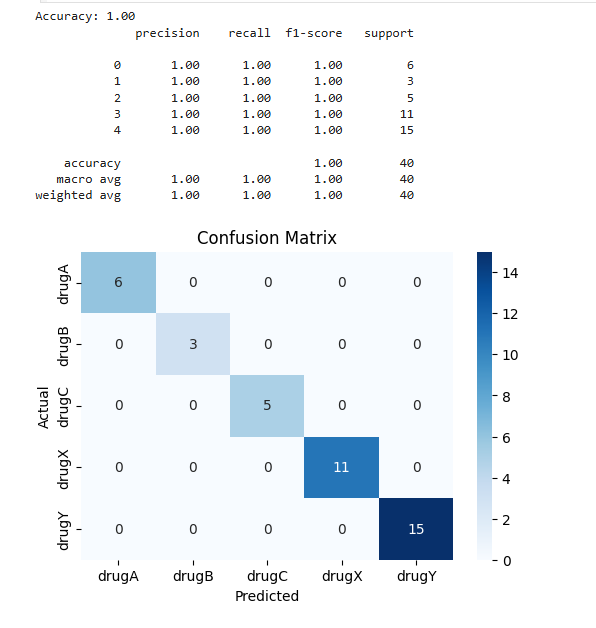
from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

plot\_tree(clf, filled=True, feature\_names=X.columns)

plt.show()



import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor, plot\_tree

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import matplotlib.pyplot as plt

import math

data = pd.read\_csv('petrol\_consumption.csv')

X = data.drop('Petrol\_Consumption', axis=1)

y = data['Petrol\_Consumption']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

regressor = DecisionTreeRegressor(random\_state=42, max\_depth=3)  *# Limiting depth for readability*

regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = math.sqrt(mse)

print("Regression Tree Evaluation Metrics:")

print(f"Mean Absolute Error (MAE): {mae:.2f}")

print(f"Mean Squared Error (MSE): {mse:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

plt.figure(figsize=(20,6))

plot\_tree(

    regressor,

    feature\_names=X.columns,

    filled=True,

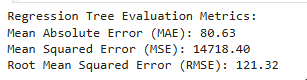
    rounded=True,

    fontsize=10,

)

plt.title("Regression Tree for Petrol Consumption Prediction", fontsize=14)

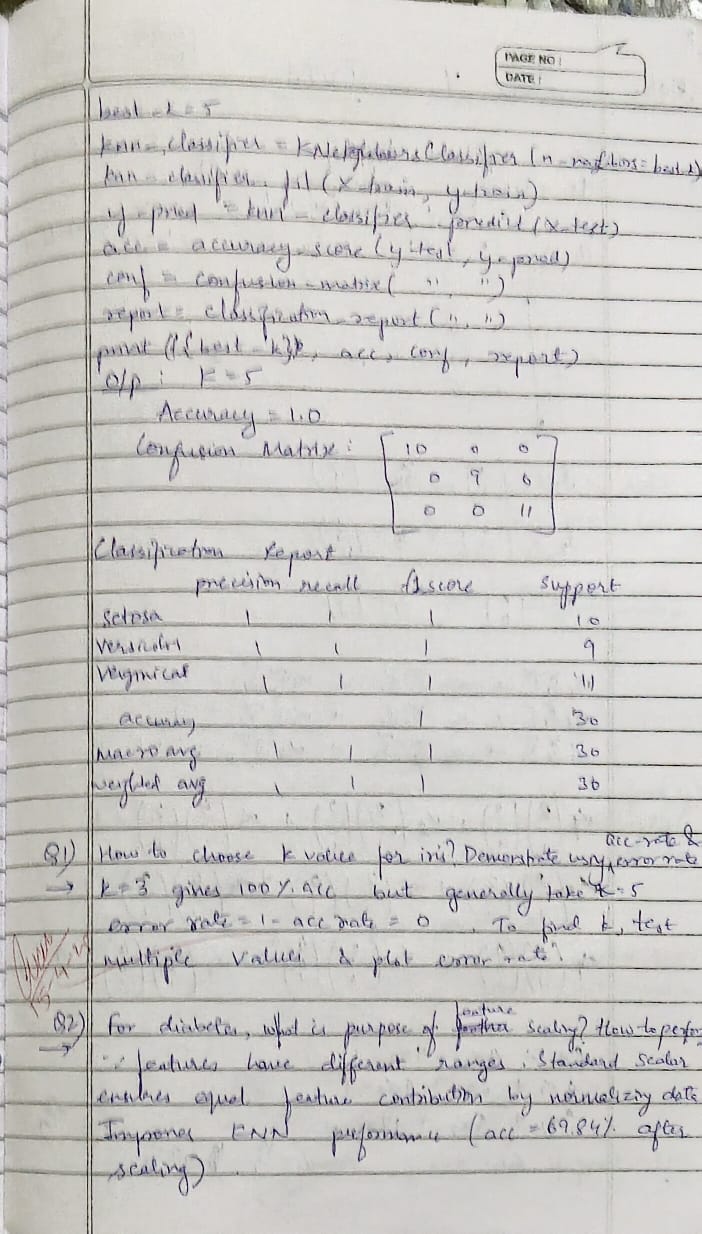
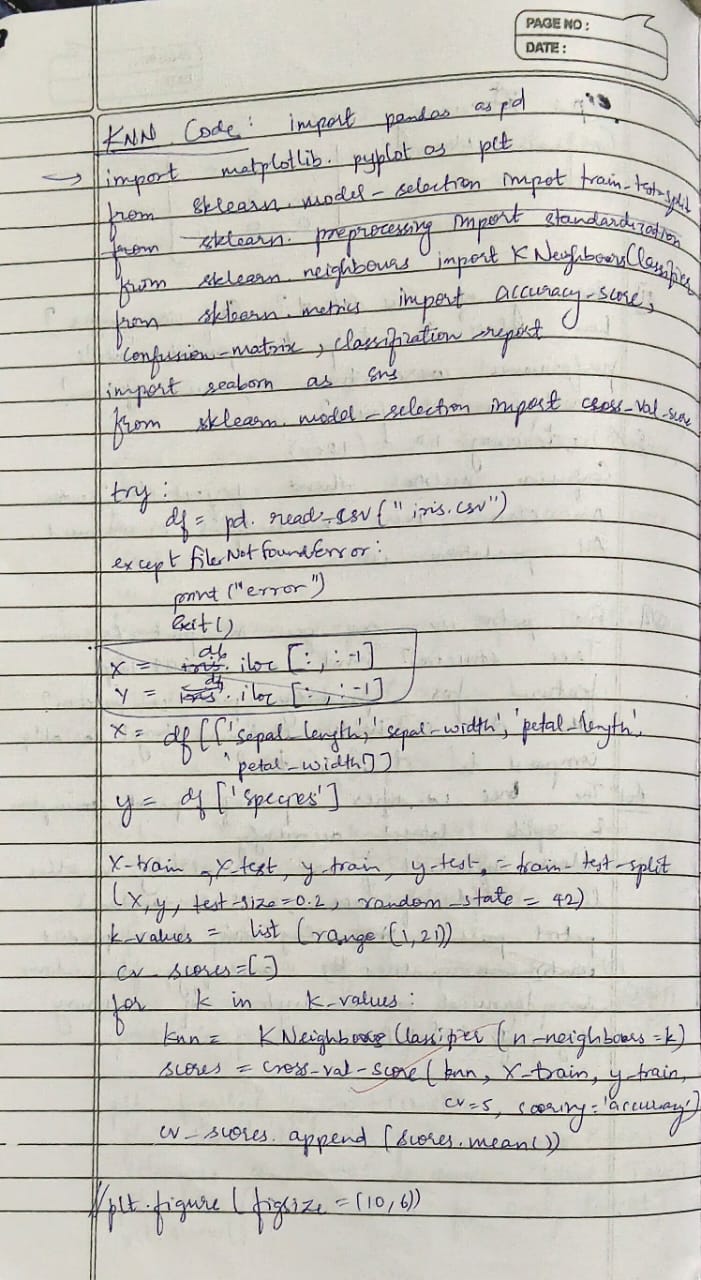
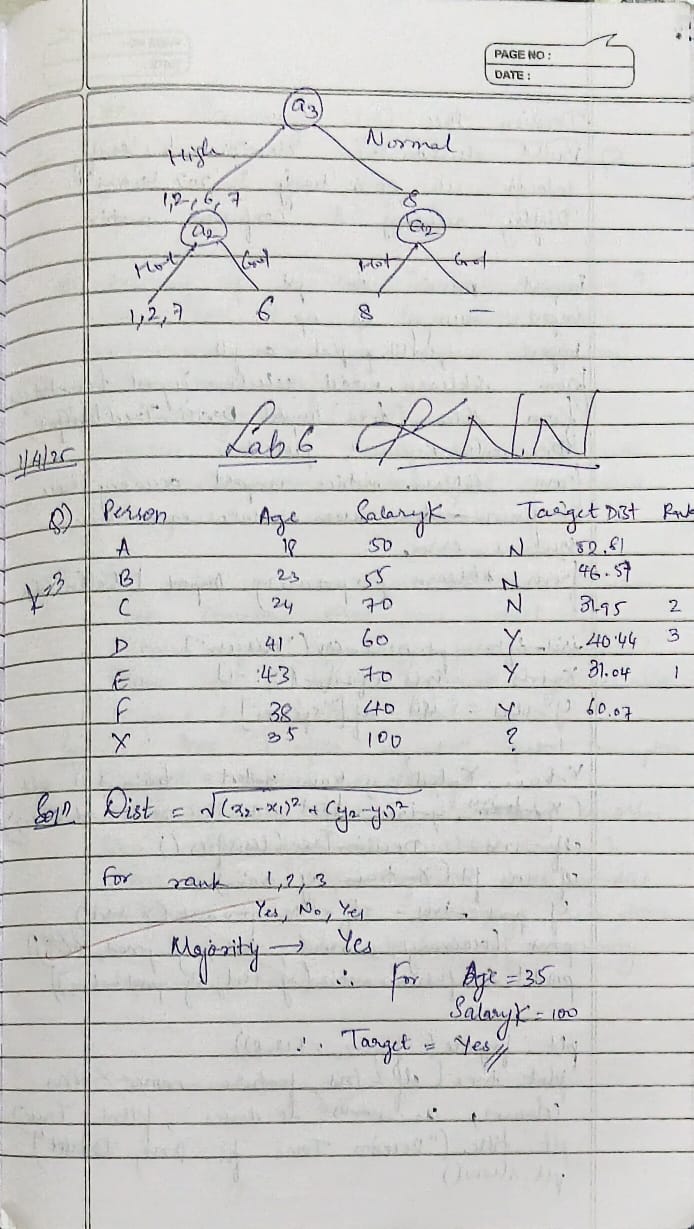
plt.show()



**Program 6**

**Build KNN Classification model for a given dataset.**

**Screenshot :**



**Code:**

**Iris.csv**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import (accuracy\_score, confusion\_matrix, classification\_report, ConfusionMatrixDisplay)

from sklearn.preprocessing import LabelEncoder

data = pd.read\_csv('iris.csv')

X = data.drop('species', axis=1)

y = data['species']

le = LabelEncoder()

y = le.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

accuracy = []

for k in range(1, 21):

    knn = KNeighborsClassifier(n\_neighbors=k)

    knn.fit(X\_train, y\_train)

    y\_pred = knn.predict(X\_test)

    accuracy.append(accuracy\_score(y\_test, y\_pred))

plt.figure(figsize=(10, 6))

plt.plot(range(1, 21), accuracy, marker='o')

plt.title('Accuracy vs K Value')

plt.xlabel('K Value')

plt.ylabel('Accuracy')

plt.xticks(range(1, 21))

plt.grid()

plt.show()

optimal\_k = np.argmax(accuracy) + 1  *# +1 because range starts at 1*

print(f"\nOptimal K value: {optimal\_k}")

knn = KNeighborsClassifier(n\_neighbors=optimal\_k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"\nAccuracy: {accuracy:.4f}")

cm = confusion\_matrix(y\_test, y\_pred)

print("\nConfusion Matrix:")

print(cm)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=le.classes\_)

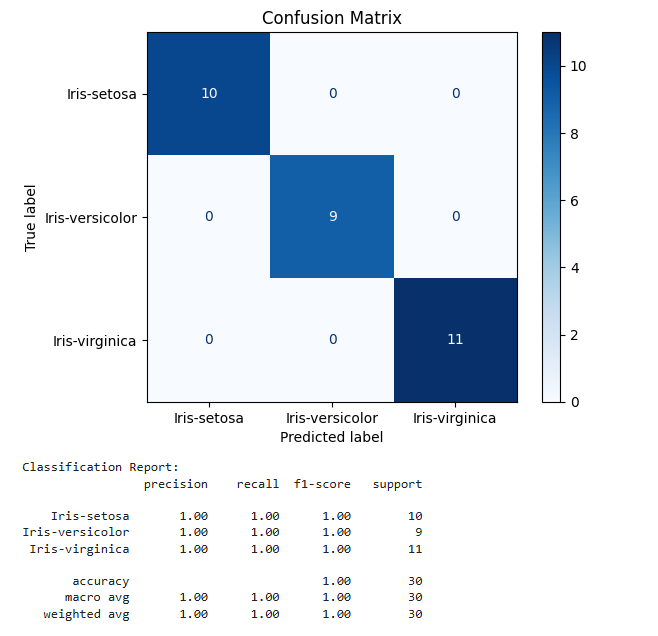
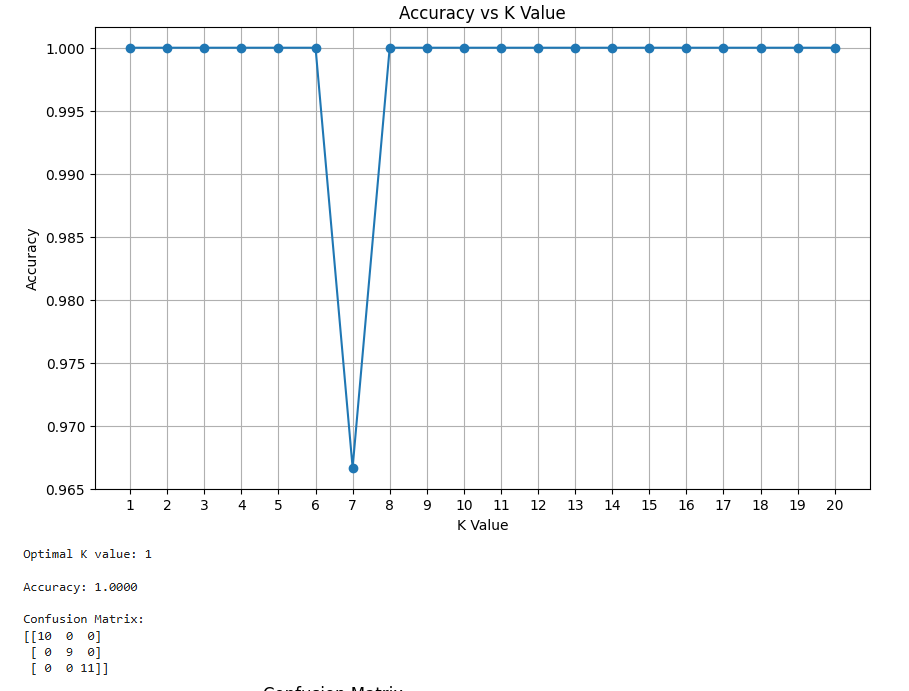
disp.plot(cmap='Blues')

plt.title('Confusion Matrix')

plt.show()

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=le.classes\_))



**Diabetes.csv**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay

data = pd.read\_csv('diabetes.csv')

X = data.drop('Outcome', axis=1)

y = data['Outcome']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

k\_values = range(1, 21)

accuracy\_scores = []

for k in k\_values:

    knn = KNeighborsClassifier(n\_neighbors=k)

    knn.fit(X\_train, y\_train)

    y\_pred = knn.predict(X\_test)

    accuracy\_scores.append(accuracy\_score(y\_test, y\_pred))

plt.figure(figsize=(10, 6))

plt.plot(k\_values, accuracy\_scores, marker='o')

plt.title('Accuracy vs K Value')

plt.xlabel('K Value')

plt.ylabel('Accuracy')

plt.xticks(k\_values)

plt.grid()

plt.show()

optimal\_k = k\_values[np.argmax(accuracy\_scores)]

print(f"Optimal K value: {optimal\_k}")

knn = KNeighborsClassifier(n\_neighbors=optimal\_k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"\nAccuracy: {accuracy:.4f}")

cm = confusion\_matrix(y\_test, y\_pred)

print("\nConfusion Matrix:")

print(cm)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['No Diabetes', 'Diabetes'])

disp.plot(cmap='Blues')

plt.title('Confusion Matrix')

plt.show()

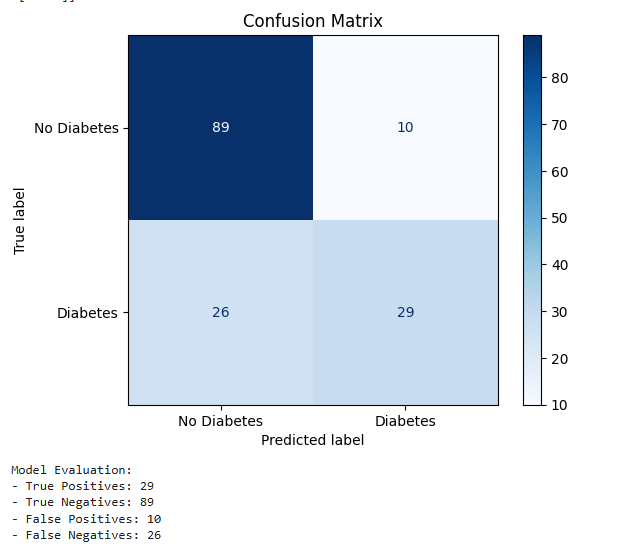
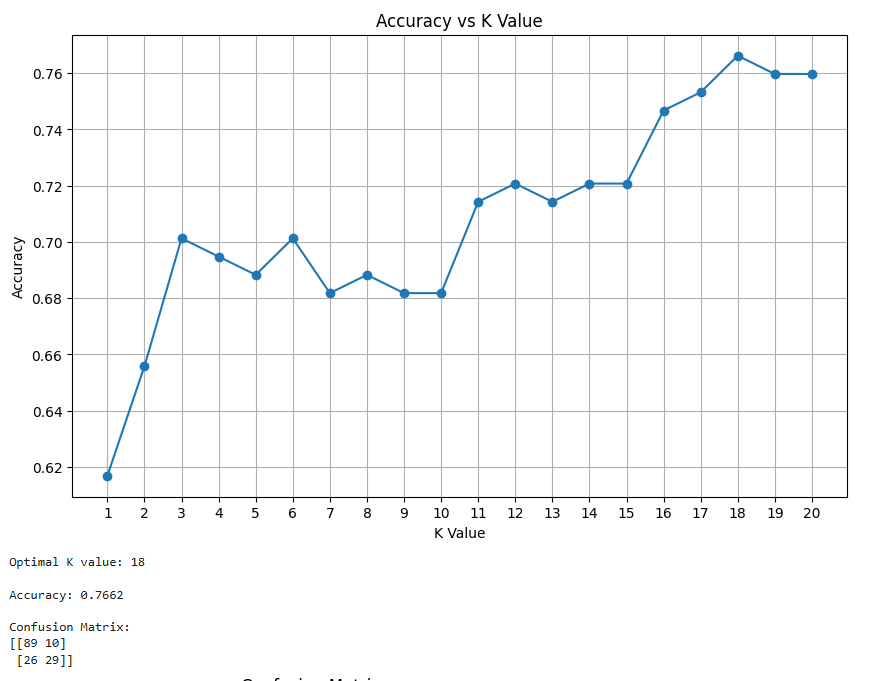
print("\nModel Evaluation:")

print(f"- True Positives: {cm[1,1]}")

print(f"- True Negatives: {cm[0,0]}")

print(f"- False Positives: {cm[0,1]}")

print(f"- False Negatives: {cm[1,0]}")



**Heart.csv**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import cross\_val\_score

try:

    df = pd.read\_csv("heart.csv")

except FileNotFoundError:

    print("Error: 'heart.csv' not found. Please make sure the file is in the correct directory.")

    exit()

X = df.drop('target', axis=1)

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

k\_values = list(range(1, 31, 2))  *# Try odd k values from 1 to 30*

cv\_scores = []

for k in k\_values:

    knn = KNeighborsClassifier(n\_neighbors=k)

    scores = cross\_val\_score(knn, X\_train\_scaled, y\_train, cv=5, scoring='accuracy')

    cv\_scores.append(scores.mean())

plt.figure(figsize=(12, 6))

plt.plot(k\_values, cv\_scores, marker='o')

plt.title('Cross-Validation Accuracy vs. K Value (Scaled Data)')

plt.xlabel('Number of Neighbors (K)')

plt.ylabel('Mean Cross-Validation Accuracy')

plt.xticks(k\_values)

plt.grid(True)

plt.show()

best\_k\_index = cv\_scores.index(max(cv\_scores))

best\_k = k\_values[best\_k\_index]

print(f"\nThe optimal K value found through cross-validation is: {best\_k}")

knn\_classifier = KNeighborsClassifier(n\_neighbors=best\_k)

knn\_classifier.fit(X\_train\_scaled, y\_train)

y\_pred = knn\_classifier.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test, y\_pred)

confusion = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print(f"\nK-Nearest Neighbors Classifier with K = {best\_k} (Scaled Data)")

print("Accuracy Score on Test Data:", accuracy)

plt.figure(figsize=(8, 6))

sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',

            xticklabels=['No Heart Disease', 'Heart Disease'], yticklabels=['No Heart Disease', 'Heart Disease'])

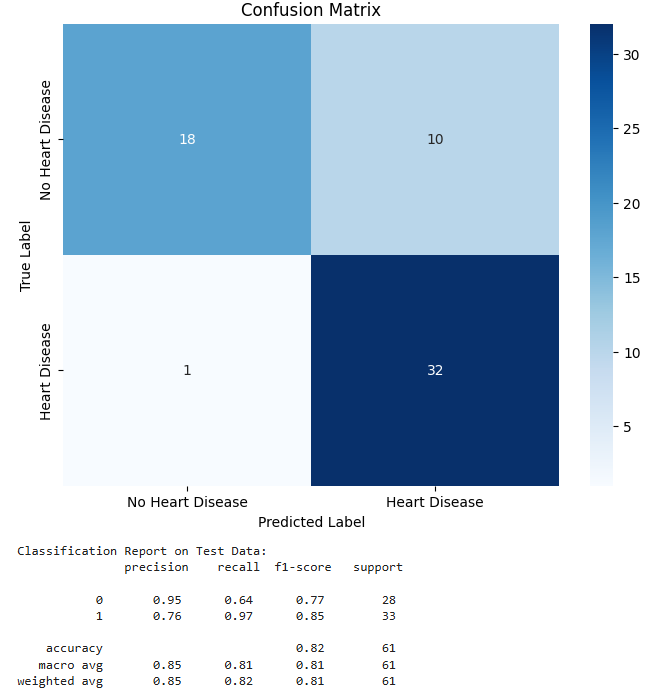
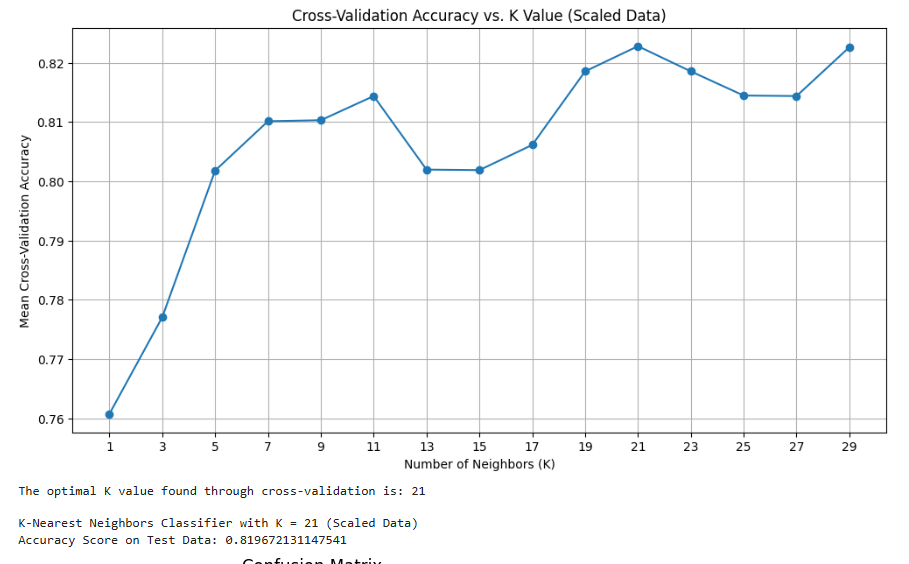
plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

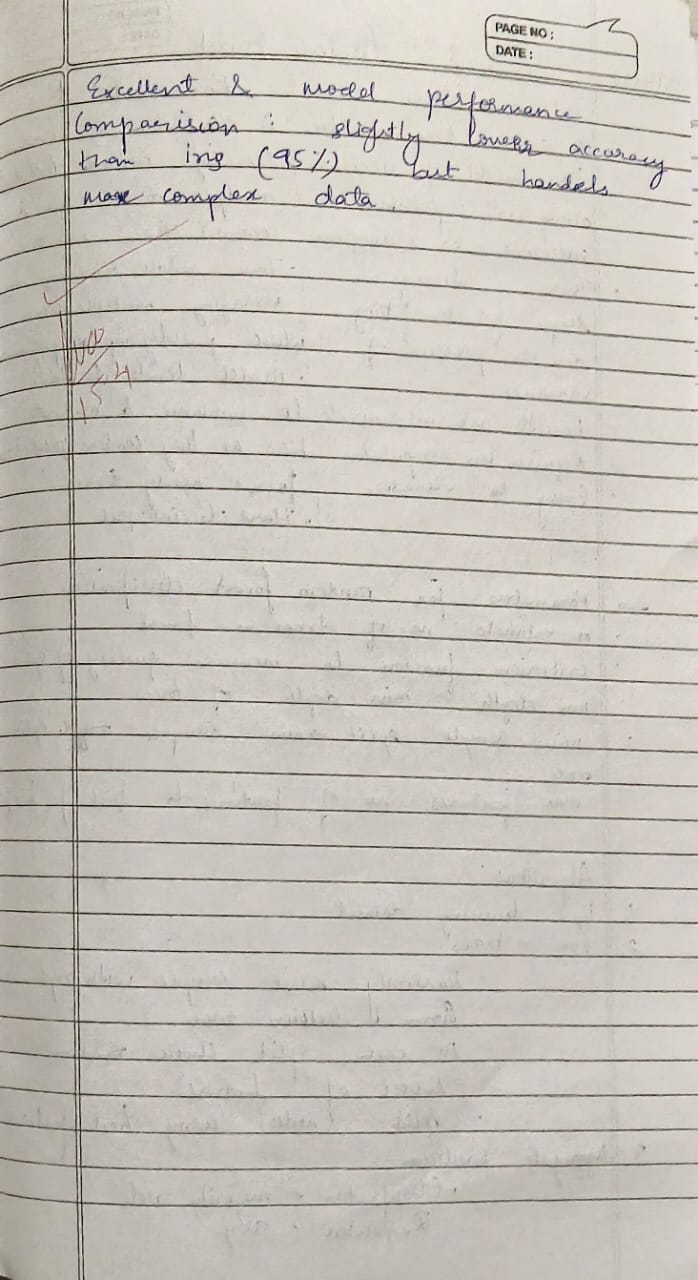
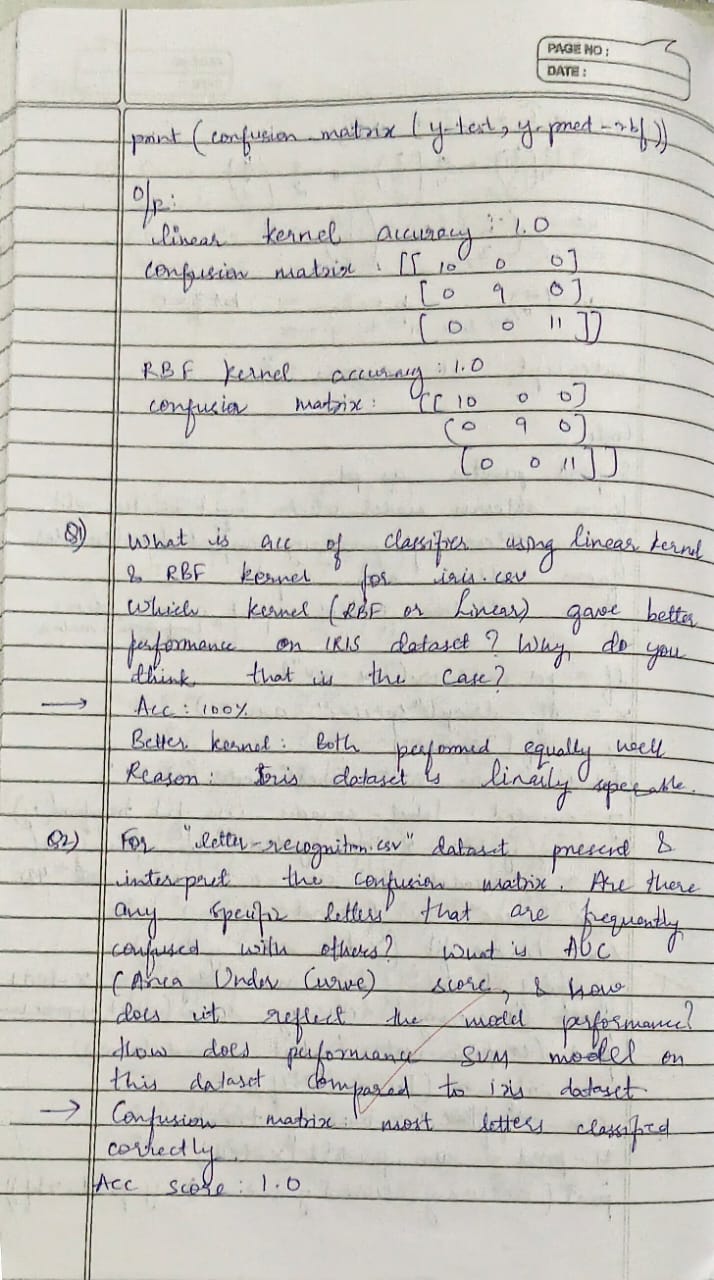
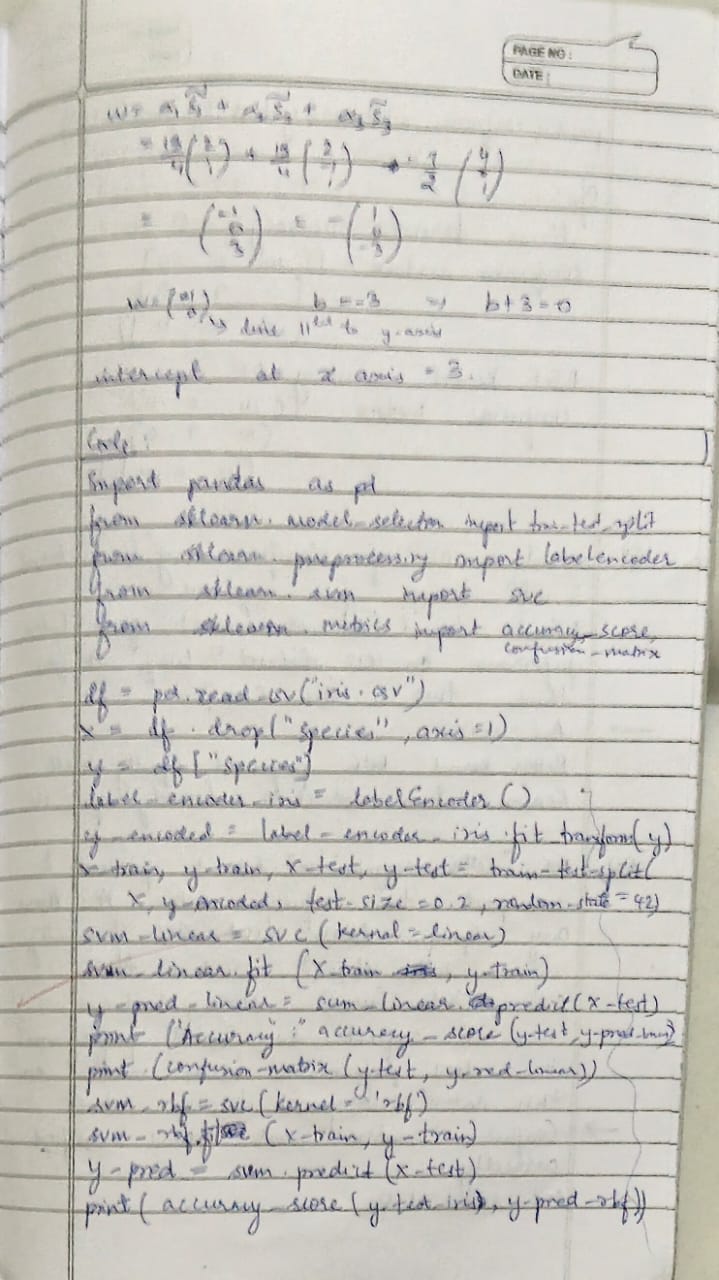
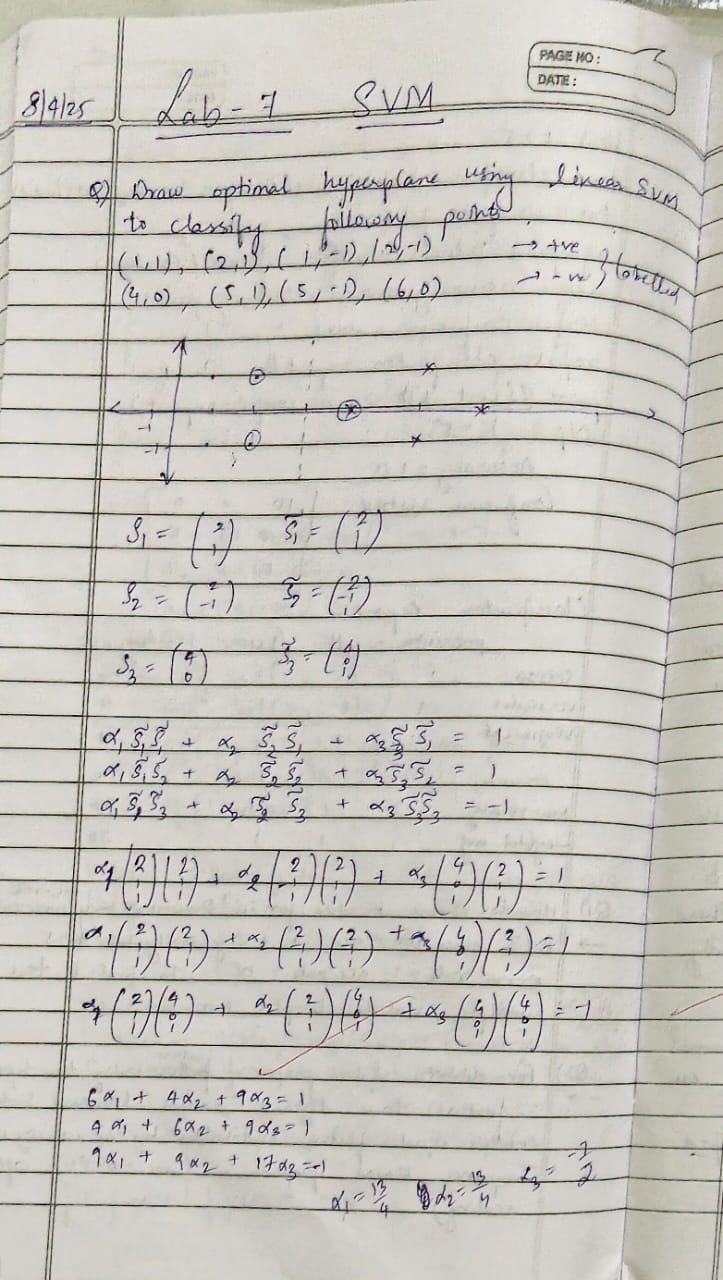
print("\nClassification Report on Test Data:\n", report)



**Program7**

**Build Support vector machine model for a given dataset**

**Screenshot:**



**Code:**

**Iris.csv**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

iris\_df = pd.read\_csv('iris.csv')

X = iris\_df.iloc[:, :-1]

y = iris\_df.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

def evaluate\_svm(kernel\_type):

    svm\_clf = SVC(kernel=kernel\_type, random\_state=42)

    svm\_clf.fit(X\_train, y\_train)

    y\_pred = svm\_clf.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    cm = confusion\_matrix(y\_test, y\_pred)

    plt.figure(figsize=(6, 4))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

                xticklabels=svm\_clf.classes\_,

                yticklabels=svm\_clf.classes\_)

    plt.title(f'Confusion Matrix ({kernel\_type} kernel)')

    plt.ylabel('Actual')

    plt.xlabel('Predicted')

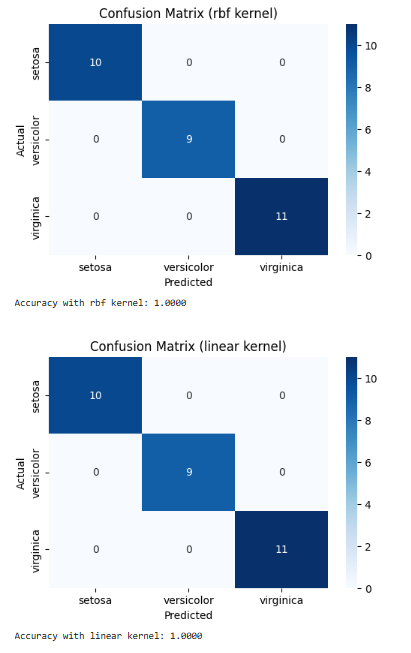
    plt.show()

    print(f"Accuracy with {kernel\_type} kernel: {accuracy:.4f}")

    print("\n")

evaluate\_svm('rbf')

evaluate\_svm('linear')



**Letter Recognition**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_auc\_score, roc\_curve

from sklearn.preprocessing import LabelBinarizer

import matplotlib.pyplot as plt

letters = pd.read\_csv("letter-recognition.csv")

X = letters.iloc[:, 1:]

y = letters.iloc[:, 0]  *# assuming first column is label*

y\_binary = (y == 'A').astype(int)  *# modify based on actual dataset*

X\_train, X\_test, y\_train\_bin, y\_test\_bin = train\_test\_split(X, y\_binary, test\_size=0.2, random\_state=42)

svm\_model = SVC(kernel='rbf', probability=True)

svm\_model.fit(X\_train, y\_train\_bin)

y\_pred = svm\_model.predict(X\_test)

y\_prob = svm\_model.predict\_proba(X\_test)[:, 1]

print("Letter Recognition Accuracy:", accuracy\_score(y\_test\_bin, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test\_bin, y\_pred))

fpr, tpr, \_ = roc\_curve(y\_test\_bin, y\_prob)

auc\_score = roc\_auc\_score(y\_test\_bin, y\_prob)

plt.figure()

plt.plot(fpr, tpr, label=f"ROC curve (AUC = {auc\_score:.2f})")

plt.plot([0, 1], [0, 1], 'k--')

plt.title("ROC Curve")

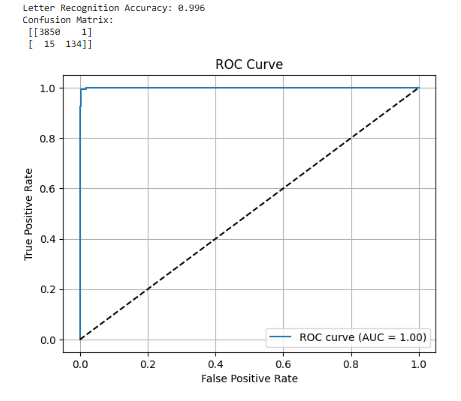
plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend(loc="lower right")

plt.grid(True)

plt.show()



**Horse Mule dataset**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder

df = pd.read\_csv("horse\_mule\_data.csv")

*# Encode 'Horse'=0, 'Mule'=1*

df['Label'] = LabelEncoder().fit\_transform(df['Label'])

X = df[['Height', 'Weight']]

y = df['Label']

model = SVC(kernel='linear', C=1000)  *# High C -> fewer support vectors*

model.fit(X, y)

support\_vectors = model.support\_vectors\_

accuracy = model.score(X, y)

print("Accuracy:", accuracy)

print("Support Vectors:\n", support\_vectors)

print("Number of Support Vectors:", len(support\_vectors))

colors = ['red' if label == 0 else 'blue' for label in y]

plt.figure(figsize=(8,6))

plt.scatter(X['Height'], X['Weight'], c=colors, label='Data Points')

plt.scatter(support\_vectors[:, 0], support\_vectors[:, 1],

            s=200, facecolors='none', edgecolors='black', label='Support Vectors')

plt.xlabel("Height")

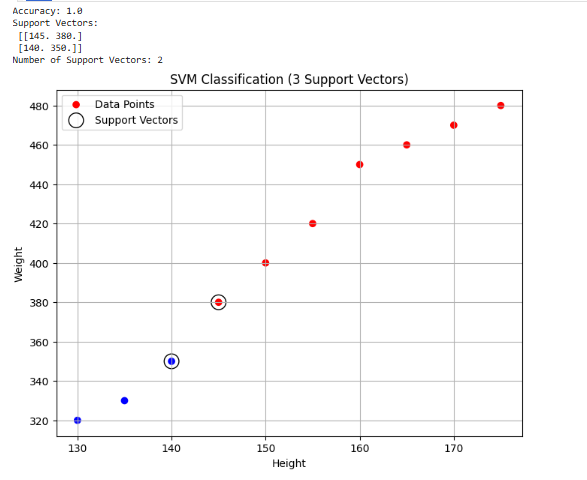
plt.ylabel("Weight")

plt.title("SVM Classification (3 Support Vectors)")

plt.legend()

plt.grid(True)

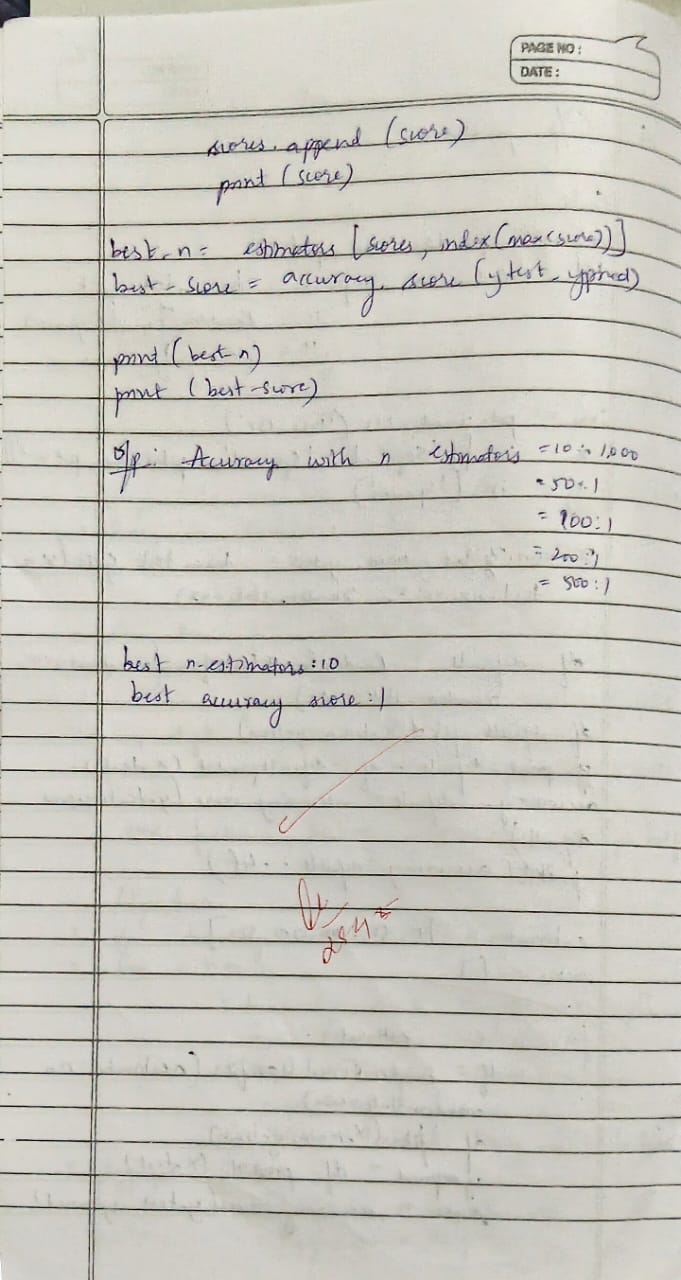
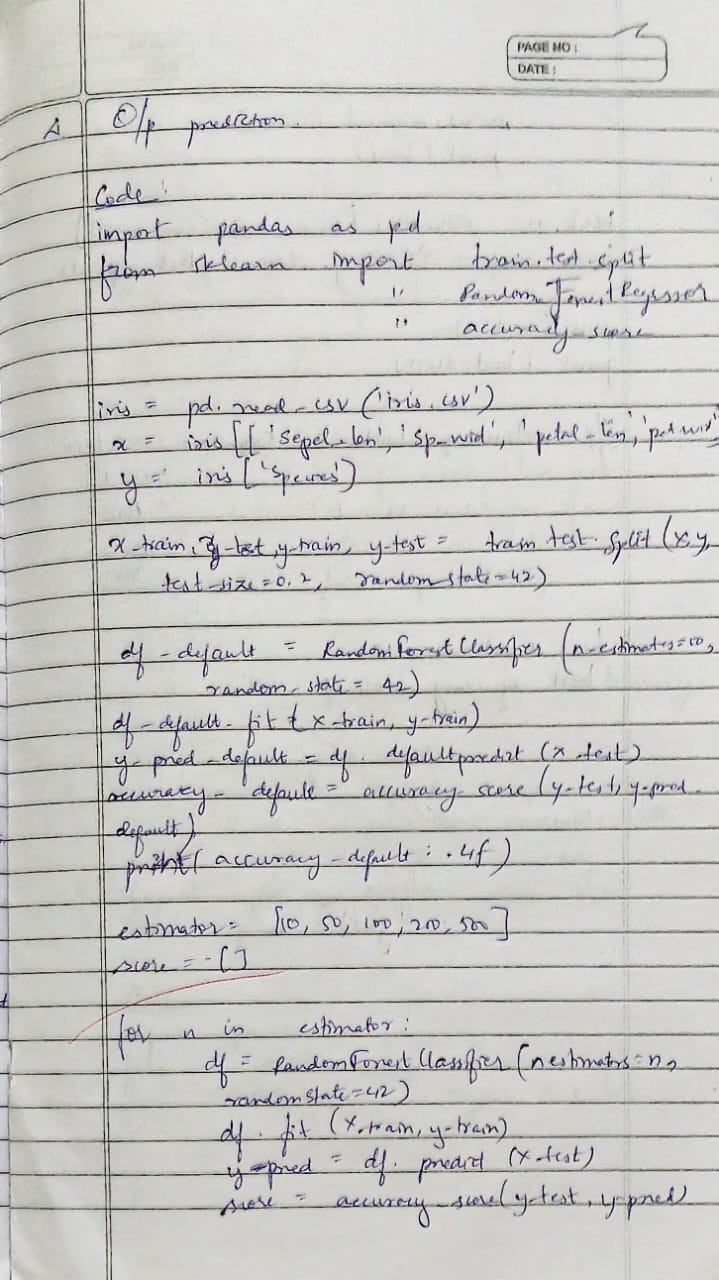
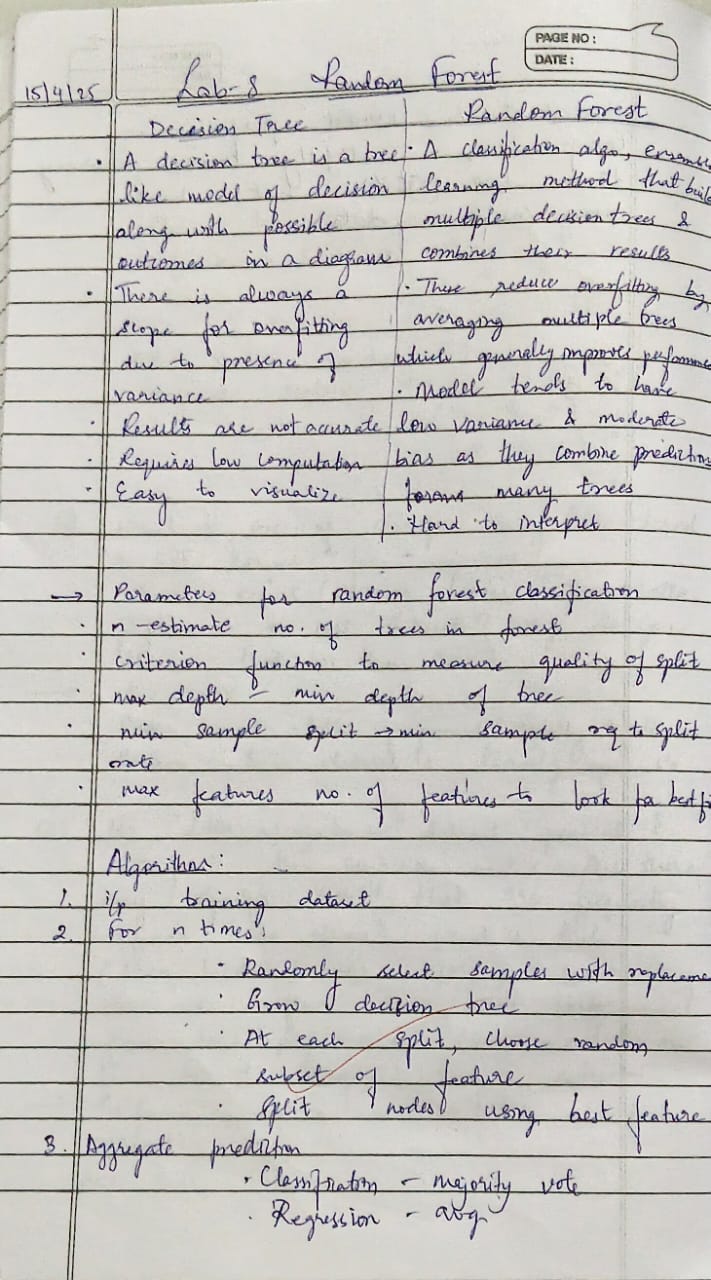
plt.show()



**Program 8**

**Implement Random forest ensemble method on a given dataset**

**Screenshot:**



**Code:**

**Iris.csv**

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

iris = pd.read\_csv("iris.csv")

X = iris.iloc[:,:-1]

y = iris.iloc[:,-1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

rf\_classifier = RandomForestClassifier(random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

y\_pred = rf\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy with default n\_estimators (10): {accuracy:.4f}")

best\_accuracy = 0

best\_n\_estimators = 0

for n\_estimators in range(10, 201, 10):

    rf\_classifier = RandomForestClassifier(n\_estimators=n\_estimators, random\_state=42)

    rf\_classifier.fit(X\_train, y\_train)

    y\_pred = rf\_classifier.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    if accuracy > best\_accuracy:

        best\_accuracy = accuracy

        best\_n\_estimators = n\_estimators

print(f"Best accuracy: {best\_accuracy:.4f} achieved with n\_estimators = {best\_n\_estimators}")



import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import roc\_auc\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

from sklearn.preprocessing import label\_binarize

from sklearn.multiclass import OneVsRestClassifier

iris = load\_iris()

X, y = iris.data, iris.target

y = label\_binarize(y, classes=[0, 1, 2])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

n\_estimators\_values = [10, 20, 30]

auc\_scores = []

for n\_estimators in n\_estimators\_values:

    rf\_classifier = OneVsRestClassifier(RandomForestClassifier(n\_estimators=n\_estimators, random\_state=42))

    rf\_classifier.fit(X\_train, y\_train)

    y\_pred\_proba = rf\_classifier.predict\_proba(X\_test)

    auc\_scores.append(roc\_auc\_score(y\_test, y\_pred\_proba, average='weighted', multi\_class='ovr'))

    print(f"AUC Score for n\_estimators = {n\_estimators}: {auc\_scores[-1]}")

plt.plot(n\_estimators\_values, auc\_scores, marker='o')

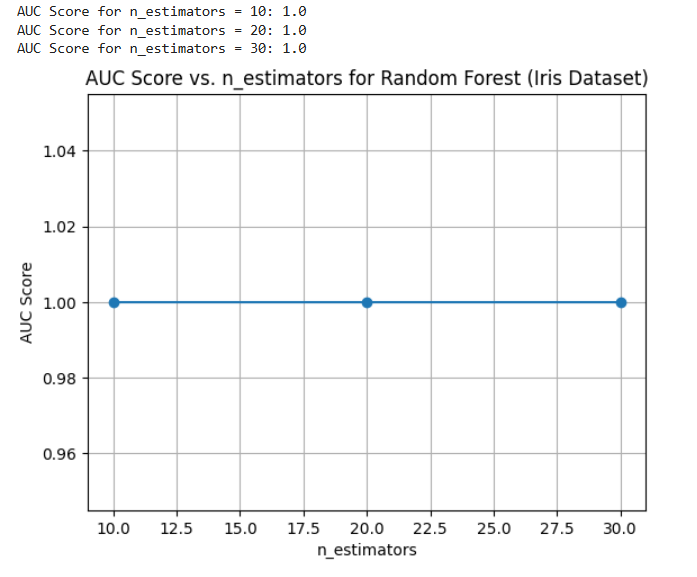
plt.xlabel('n\_estimators')

plt.ylabel('AUC Score')

plt.title('AUC Score vs. n\_estimators for Random Forest (Iris Dataset)')

plt.grid(True)

plt.show()



**Train.csv**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.preprocessing import LabelEncoder

df = pd.read\_csv("train.csv")

df = df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])

df['Age'].fillna(df['Age'].median(), inplace=True)

df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

label\_encoders = {}

for col in ['Sex', 'Embarked']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

X = df.drop(columns=['Survived'])

y = df['Survived']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.4f}")

print("Confusion Matrix:")

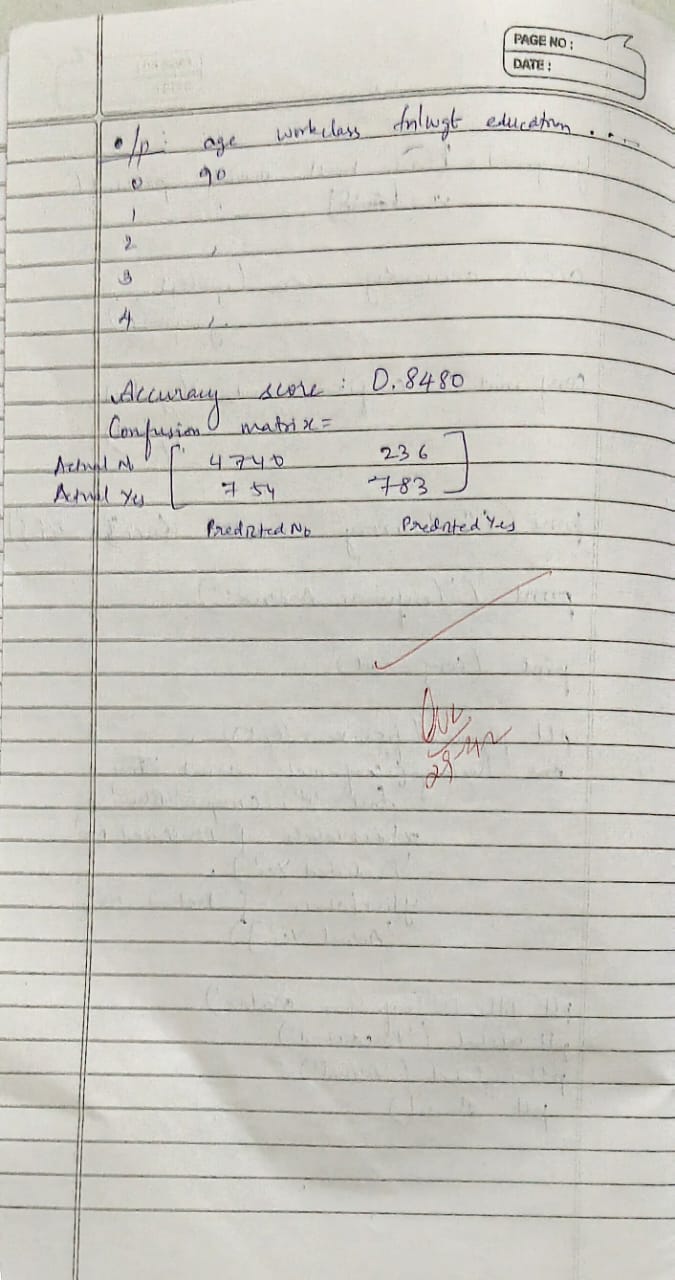
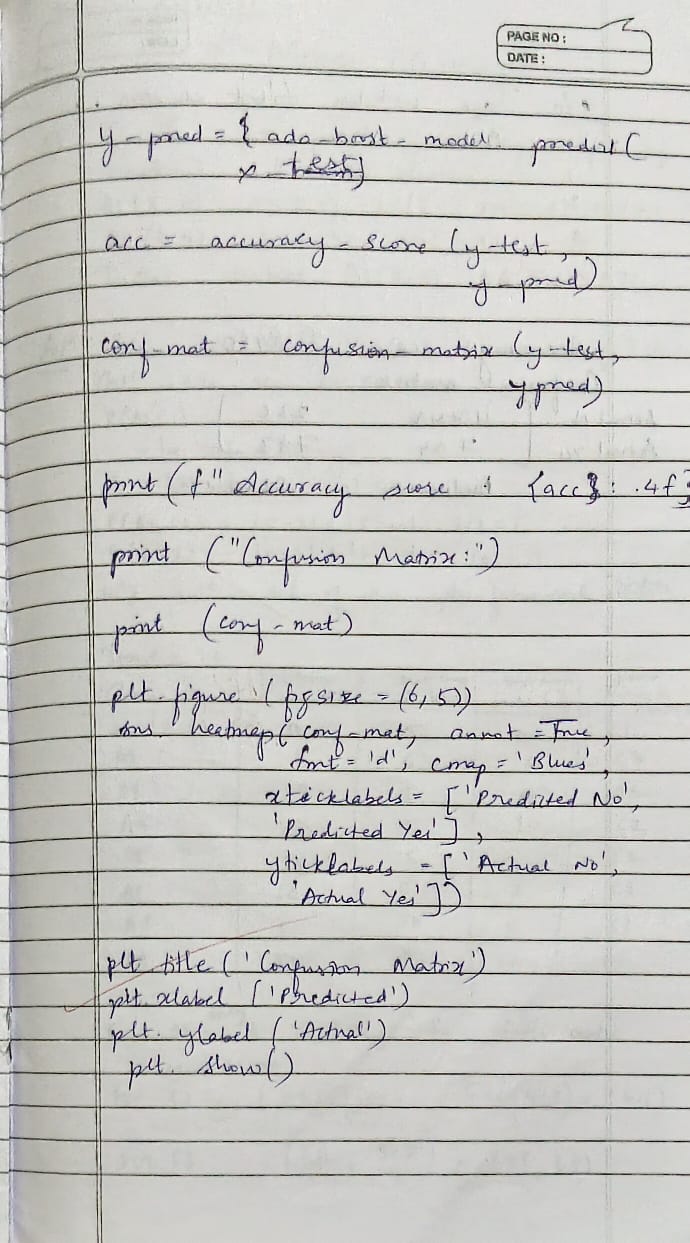
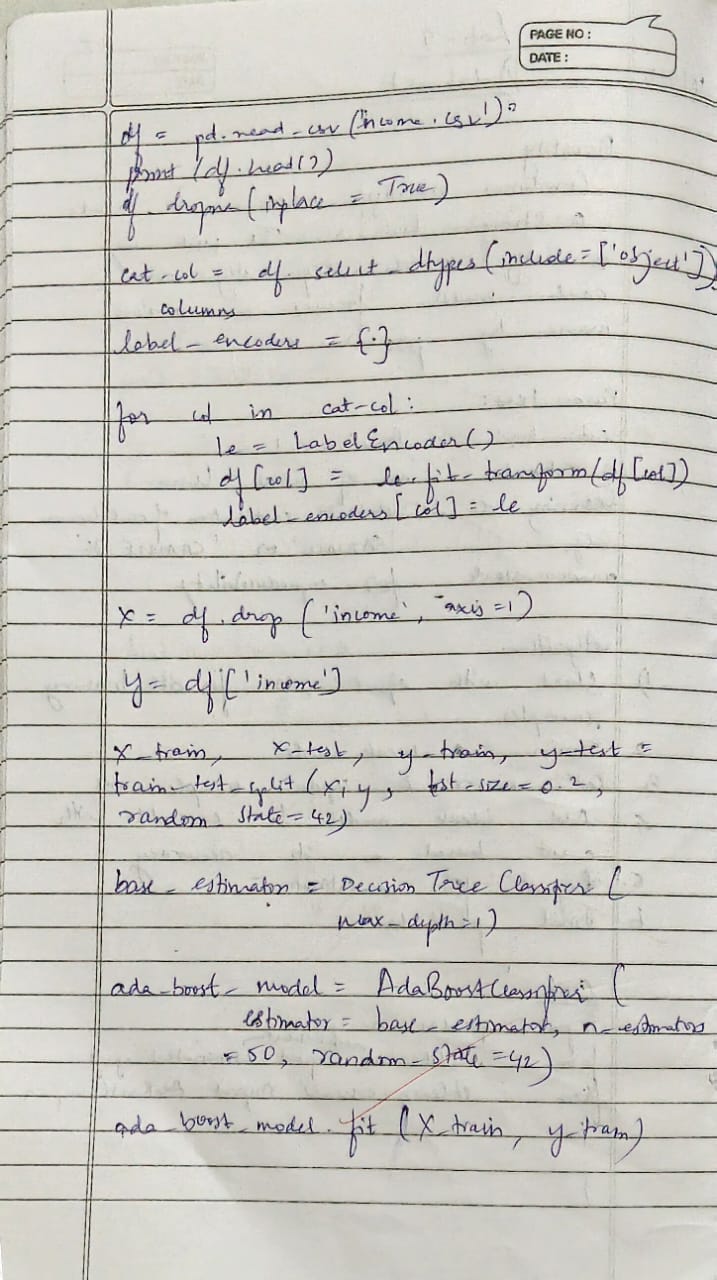
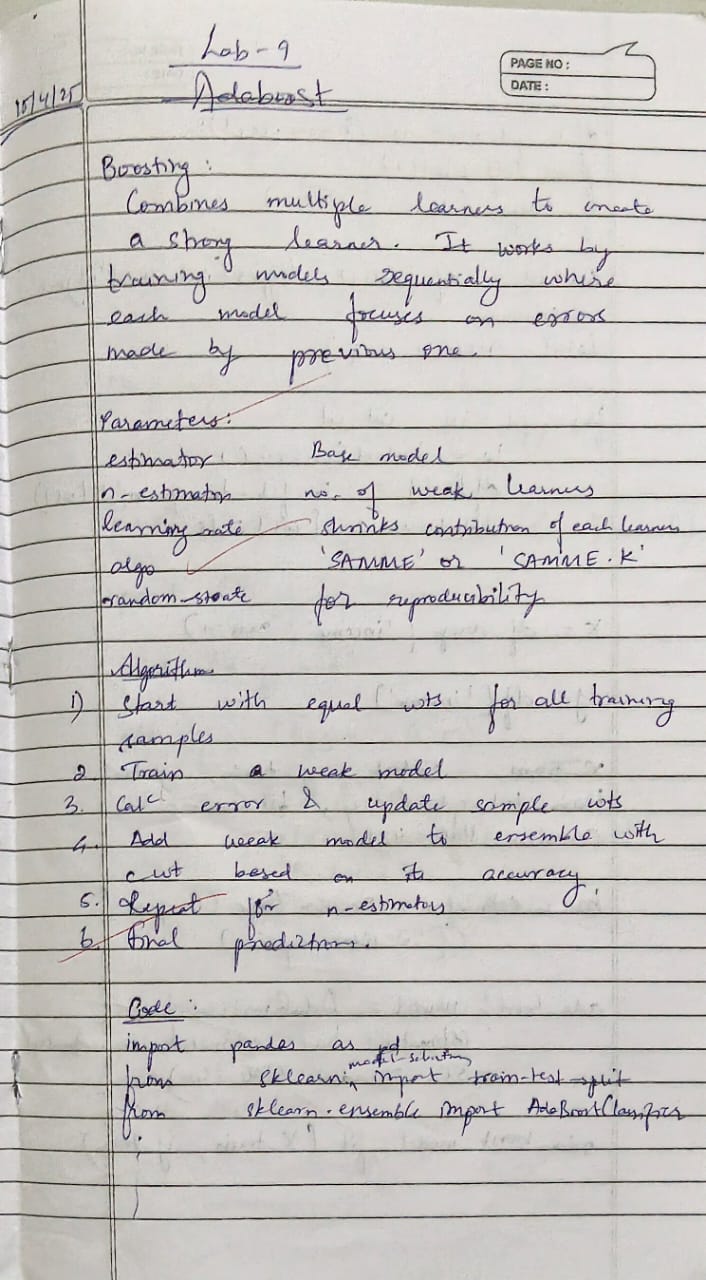
print(conf\_matrix)



**Program 9**

**Implement Boosting ensemble method on a given dataset**

**Screenshot:**



**Code:**

**Income.csv**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

data = pd.read\_csv('income.csv')

X = data.drop('income\_level', axis=1)

y = data['income\_level']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

ada\_boost = AdaBoostClassifier(n\_estimators=50, random\_state=42)

ada\_boost.fit(X\_train, y\_train)

y\_pred = ada\_boost.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Confusion Matrix:\n{conf\_matrix}")

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

            xticklabels=ada\_boost.classes\_, yticklabels=ada\_boost.classes\_)

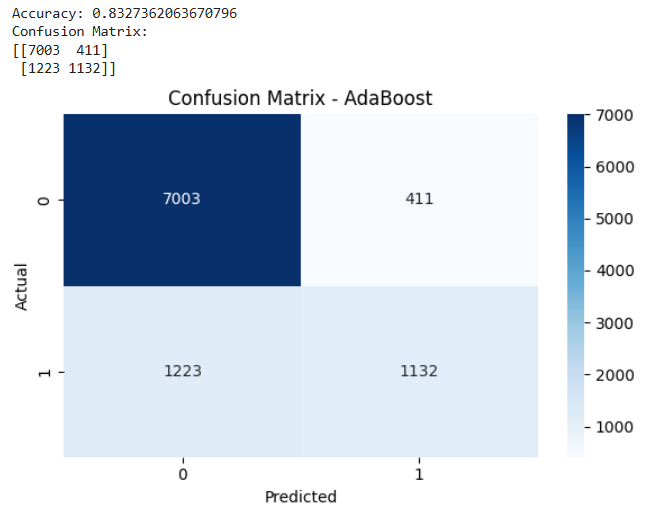
plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - AdaBoost')

plt.tight\_layout()

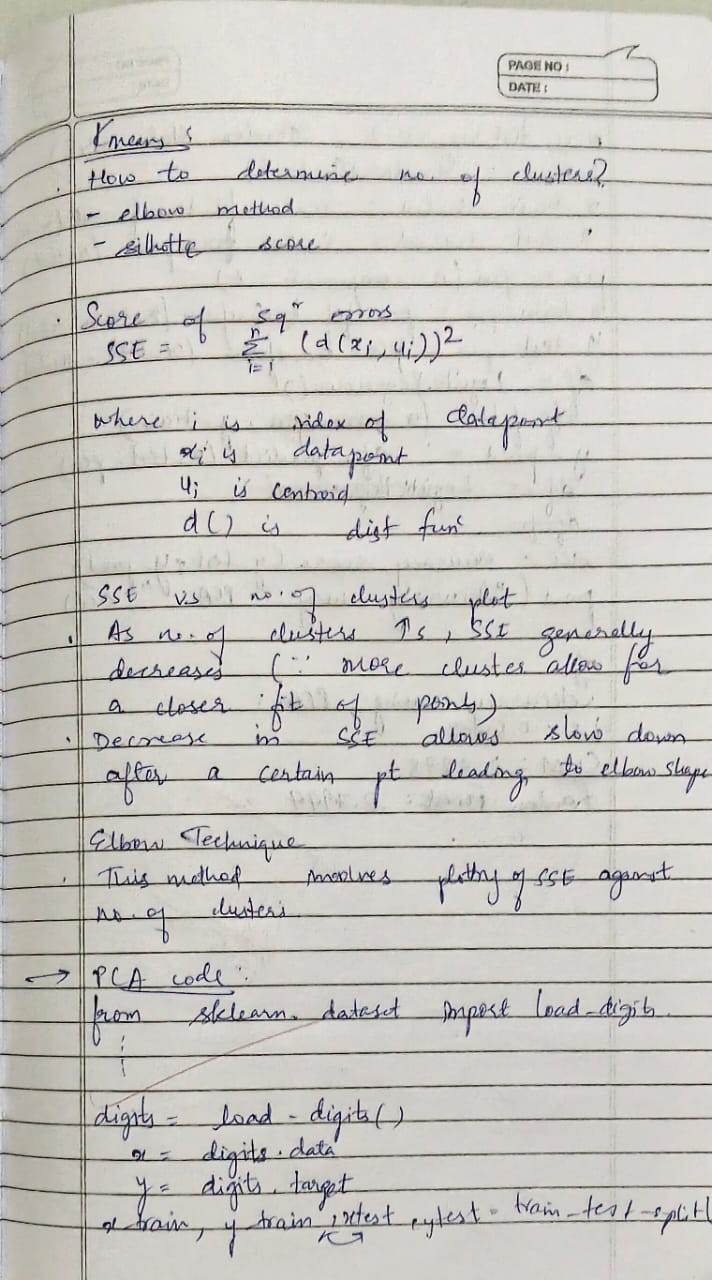
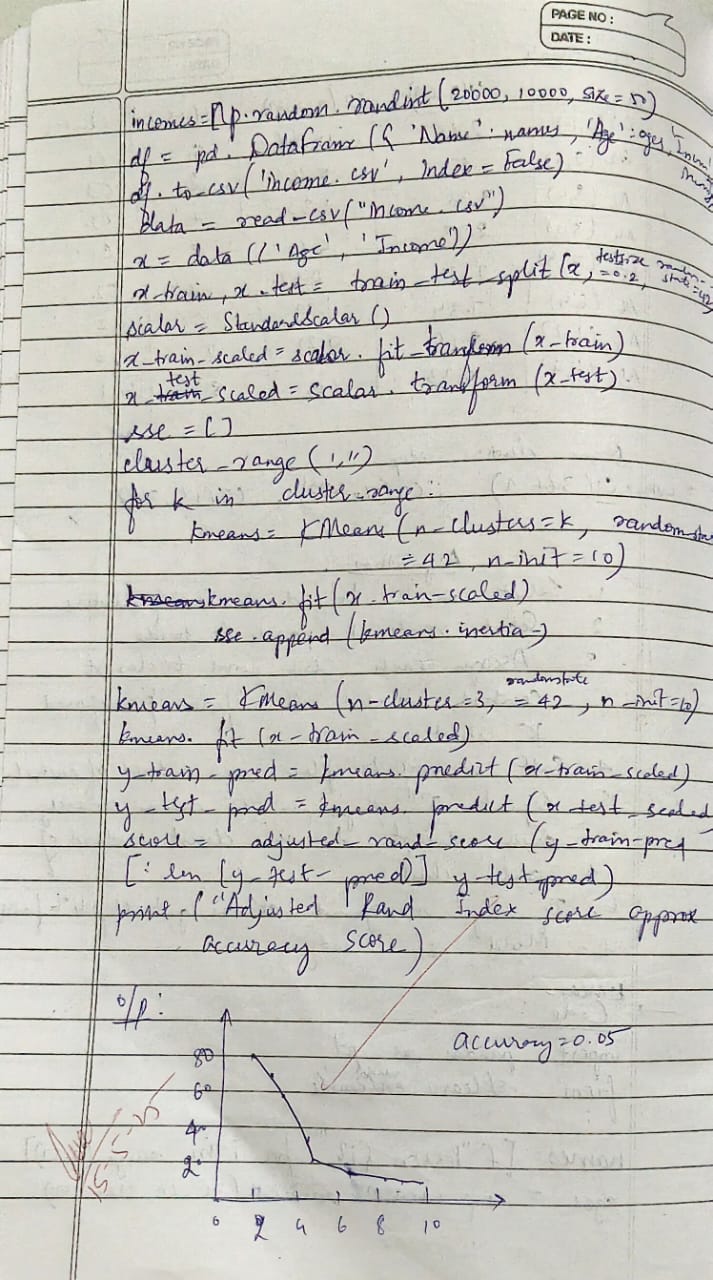
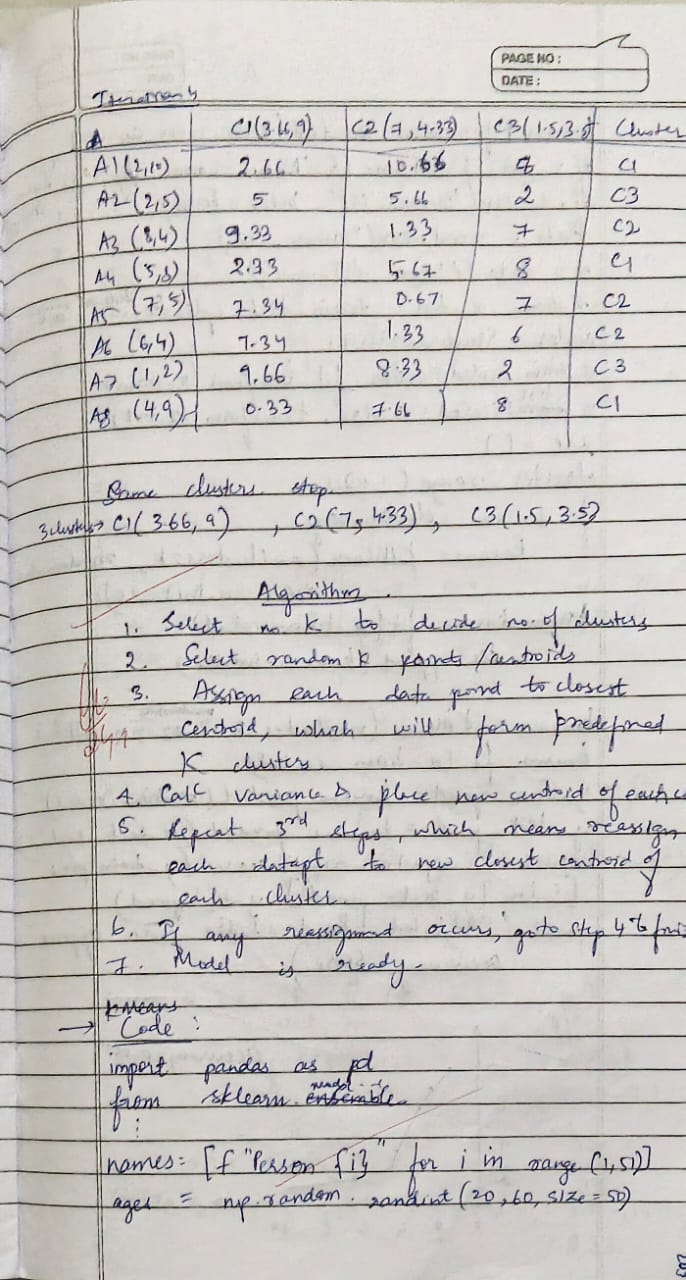
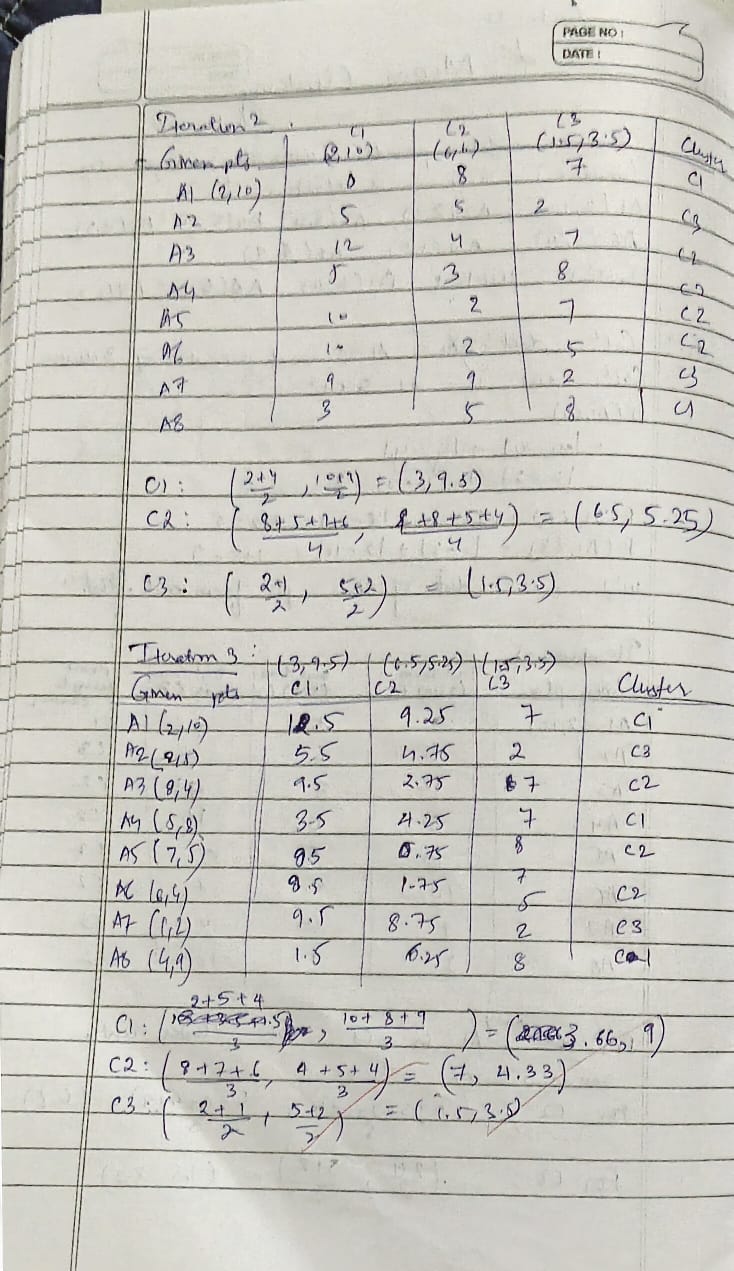
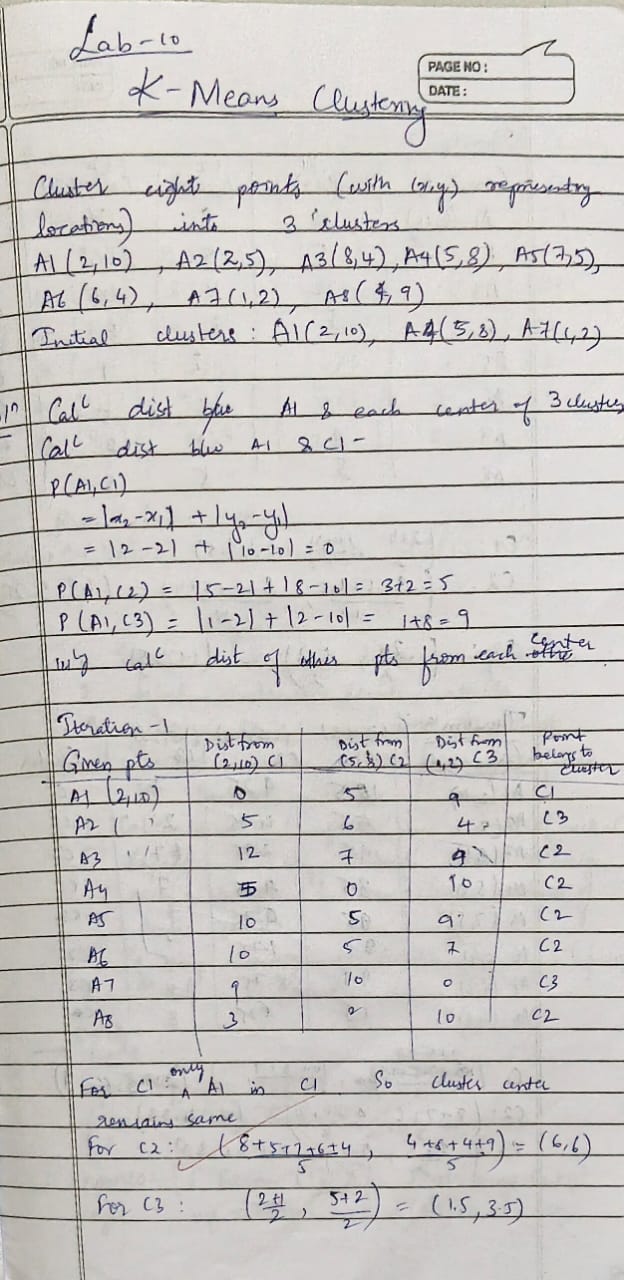
plt.show()



**Program 10**

**Build k-Means algorithm to cluster a set of data stored in a .CSV file**

**Screenshot:**



**Code:**

**Income.csv**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import adjusted\_rand\_score

import matplotlib.pyplot as plt

import random

np.random.seed(42)

names = [f"Person\_{i}" for i in range(1, 51)]

ages = np.random.randint(20, 60, 50)

incomes = np.random.randint(20000, 100000, 50)

df = pd.DataFrame({

    'Name': names,

    'Age': ages,

    'Income': incomes

})

df.to\_csv('income.csv', index=False)

print("‘income.csv' created successfully.")

from google.colab import files

files.download('income.csv')

data = pd.read\_csv('income.csv')

X = data[['Age', 'Income']]

X\_train, X\_test = train\_test\_split(X, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

sse = []

k\_range = range(1, 11)

for k in k\_range:

    kmeans = KMeans(n\_clusters=k, random\_state=42)

    kmeans.fit(X\_train\_scaled)

    sse.append(kmeans.inertia\_)

plt.plot(k\_range, sse, marker='o')

plt.xlabel('Number of Clusters')

plt.ylabel('SSE (Inertia)')

plt.title('Elbow Method: SSE vs Number of Clusters')

plt.grid(True)

plt.show()

k = 3  *# Change this based on the elbow plot*

model = KMeans(n\_clusters=k, random\_state=42)

model.fit(X\_train\_scaled)

train\_preds = model.predict(X\_train\_scaled)

test\_preds = model.predict(X\_test\_scaled)

true\_labels\_train = [random.randint(0, k-1) for \_ in range(len(train\_preds))]

accuracy = adjusted\_rand\_score(true\_labels\_train, train\_preds)

print("Adjusted Rand Index (proxy accuracy):", round(accuracy, 2))



**Iris.csv**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

iris = load\_iris()

data = pd.DataFrame(iris.data, columns=iris.feature\_names)

X = data[["petal length (cm)", "petal width (cm)"]]

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

inertia = []

K\_range = range(1, 11)

for k in K\_range:

    kmeans = KMeans(n\_clusters=k, random\_state=42)

    kmeans.fit(X\_scaled)

    inertia.append(kmeans.inertia\_)

plt.figure(figsize=(8, 5))

plt.plot(K\_range, inertia, marker='o')

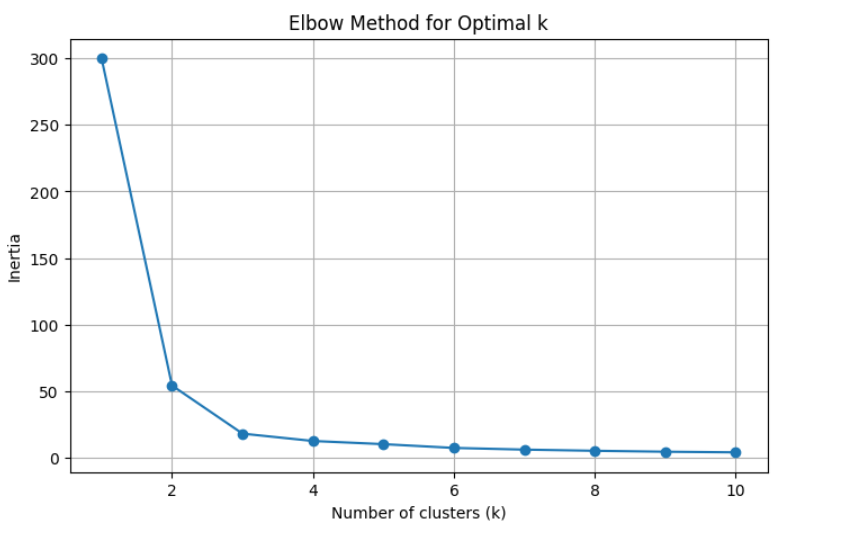
plt.title("Elbow Method for Optimal k")

plt.xlabel("Number of clusters (k)")

plt.ylabel("Inertia")

plt.grid(True)

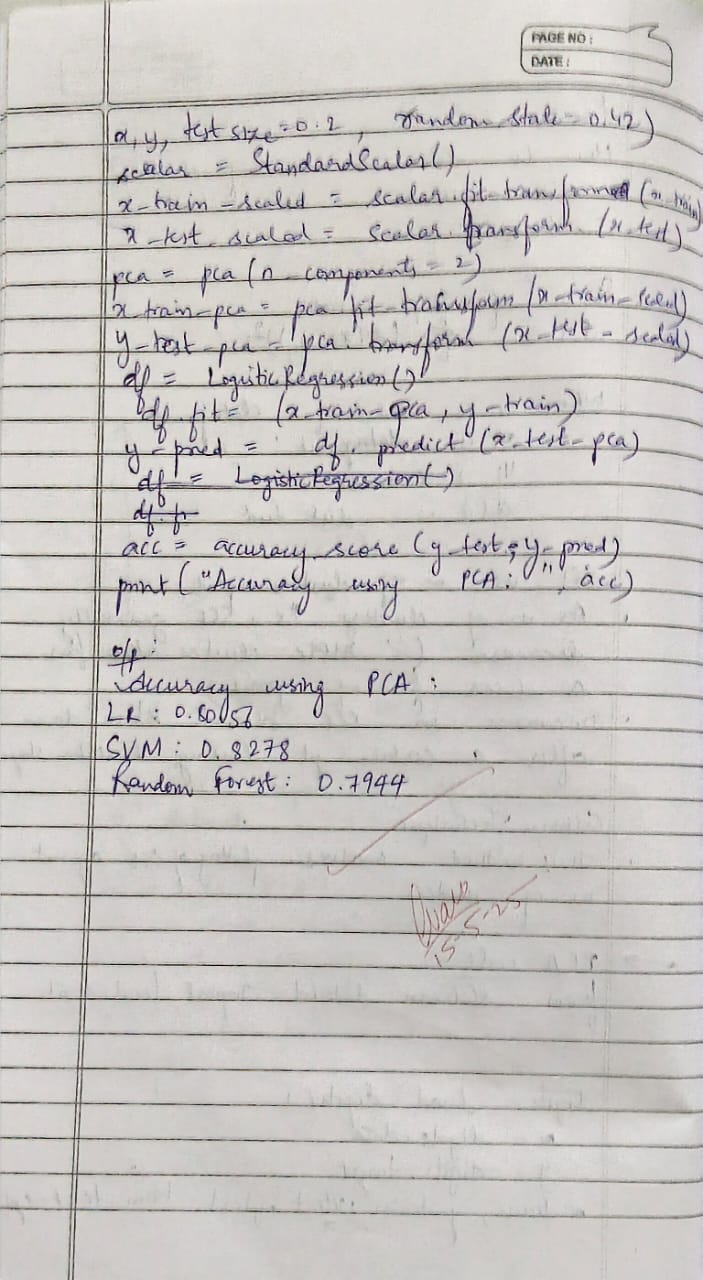
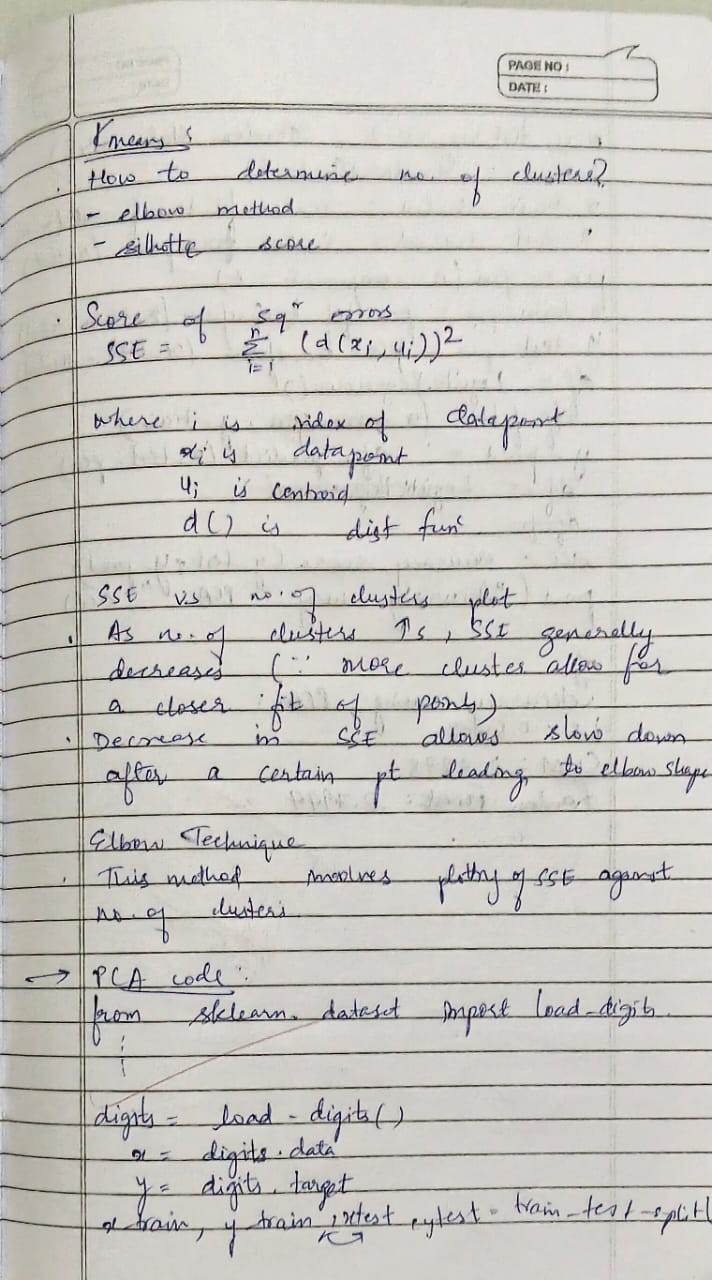
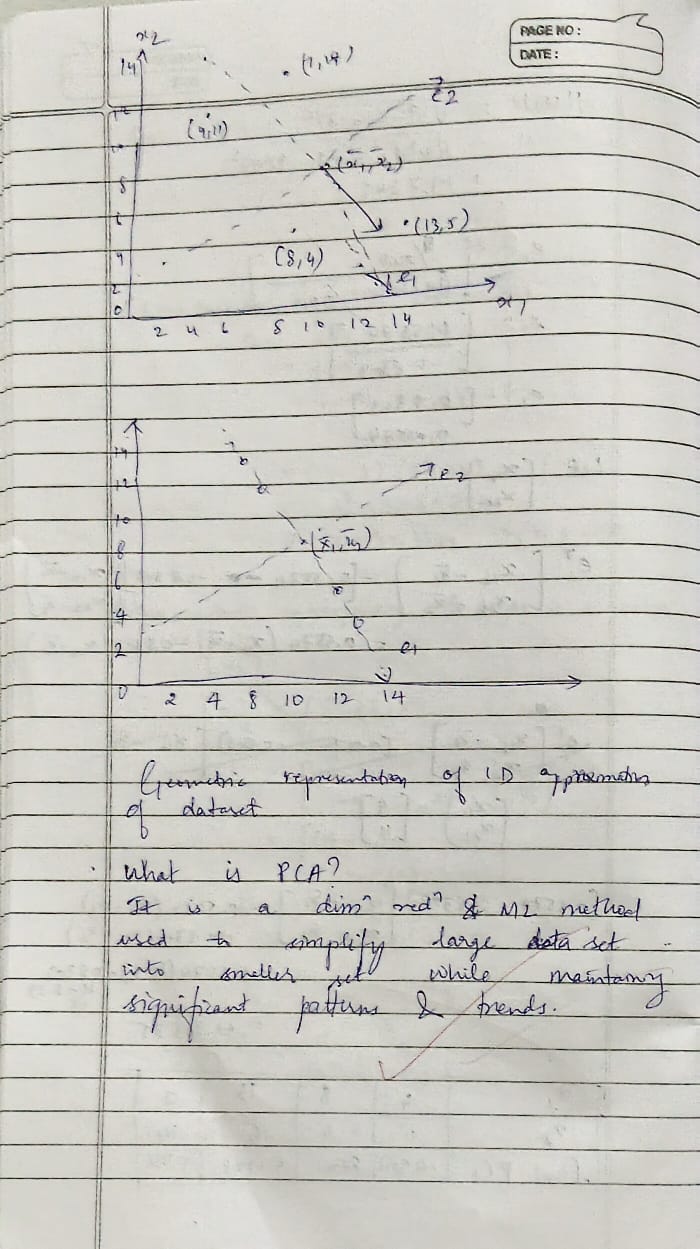
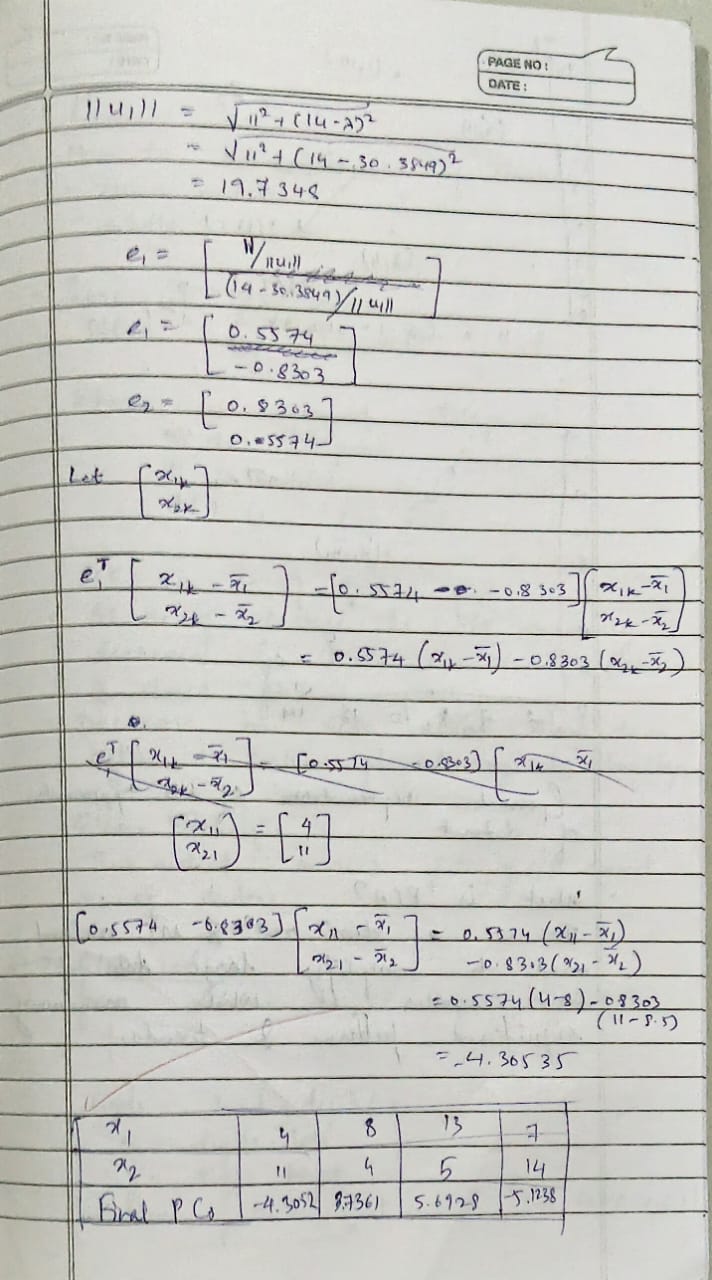
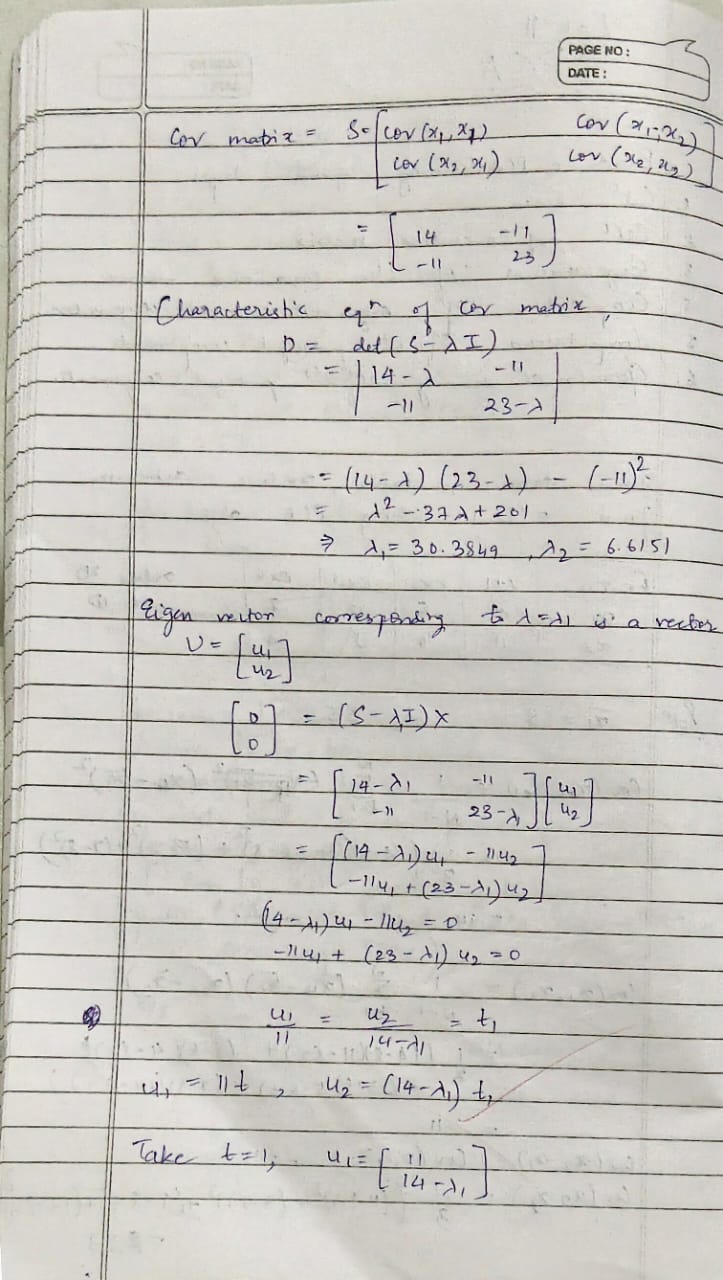
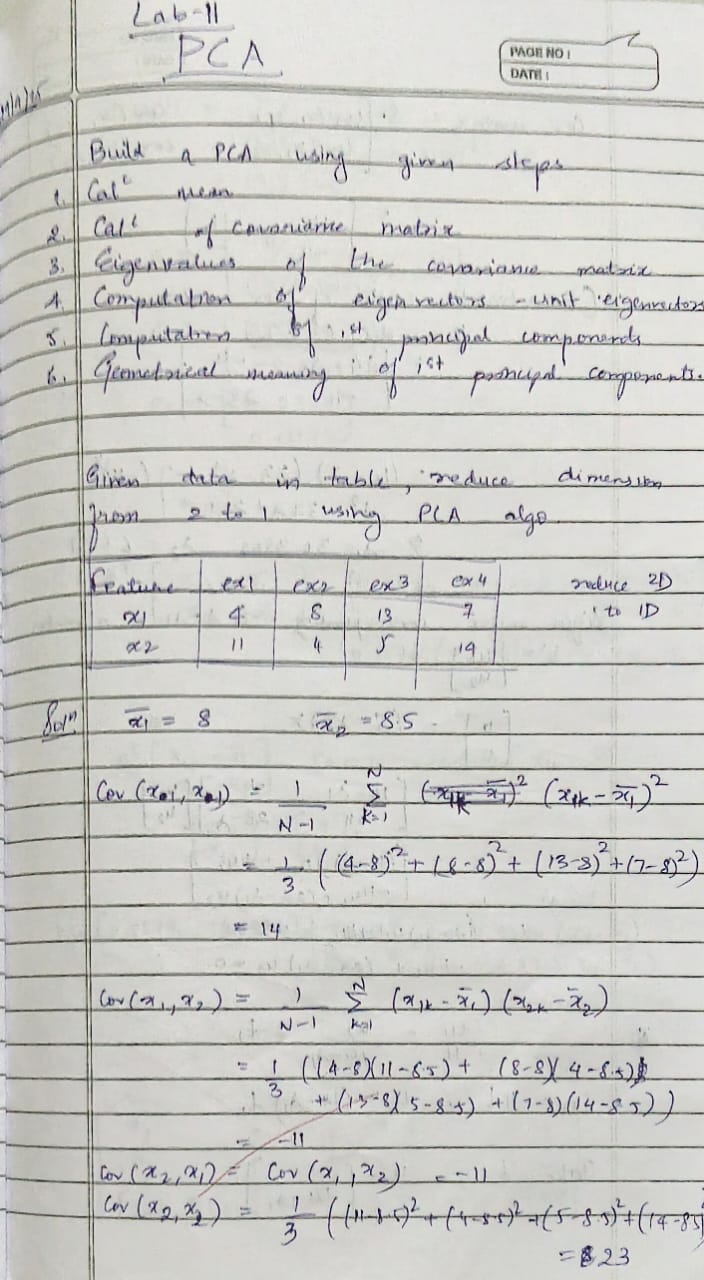
plt.show()



**Program 11**

**Implement Dimensionality reduction using Principal Component Analysis (PCA) method.**

**Screenshot:**



**Code:**

from sklearn.datasets import load\_digits

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

digits = load\_digits()

X = digits.data

y = digits.target

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X\_pca, y, test\_size=0.2, random\_state=42

)

model = LogisticRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

score = model.score(X\_test, y\_test)

print(" Model Score (accuracy using .score()):", round(score, 4))

print(" Accuracy using PCA with 2 components:", round(accuracy, 4))



import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn.decomposition import PCA

from scipy.stats import zscore

df = pd.read\_csv("/content/heart.csv")  *# Adjust path if needed*

z\_scores = np.abs(zscore(df.select\_dtypes(include=[np.number])))

df = df[(z\_scores < 3).all(axis=1)]

df\_encoded = df.copy()

for col in df\_encoded.select\_dtypes(include=["object"]).columns:

    if df\_encoded[col].nunique() <= 2:

        le = LabelEncoder()

        df\_encoded[col] = le.fit\_transform(df\_encoded[col])

    else:

        df\_encoded = pd.get\_dummies(df\_encoded, columns=[col], drop\_first=True)

X = df\_encoded.drop("target", axis=1)  *# Replace 'target' if it's named differently*

y = df\_encoded["target"]

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

models = {

    "Logistic Regression": LogisticRegression(max\_iter=1000),

    "Random Forest": RandomForestClassifier(),

    "SVM": SVC()

}

print("Model Accuracies (without PCA):")

for name, model in models.items():

    model.fit(X\_train, y\_train)

    preds = model.predict(X\_test)

    acc = accuracy\_score(y\_test, preds)

    print(f"{name}: {acc:.4f}")

pca = PCA(n\_components=0.95)  *# Retain 95% variance*

X\_pca = pca.fit\_transform(X\_scaled)

X\_train\_pca, X\_test\_pca, \_, \_ = train\_test\_split(X\_pca, y, test\_size=0.2, random\_state=42)

print("\nModel Accuracies (with PCA):")

for name, model in models.items():

    model.fit(X\_train\_pca, y\_train)

    preds = model.predict(X\_test\_pca)

    acc = accuracy\_score(y\_test, preds)

    print(f"{name}: {acc:.4f}")

