# VISVESVARAYATECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# Machine Learning(23CS6PCMAL)

***Submitted by***

**Aditya(1BM22CS015)**

***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 **Aditya (1BM22CS015),** 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 anMachine Learning (23CS6PCMAL) work prescribed for the said degree.

| | |
|---|---|
| Dr. Seema Patil<br>Assistant Professor<br>DepartmentofCSE,BMSCE | Dr.Kavitha Sooda<br>Professor & HOD<br>Department of CSE, BMSCE |

# Index

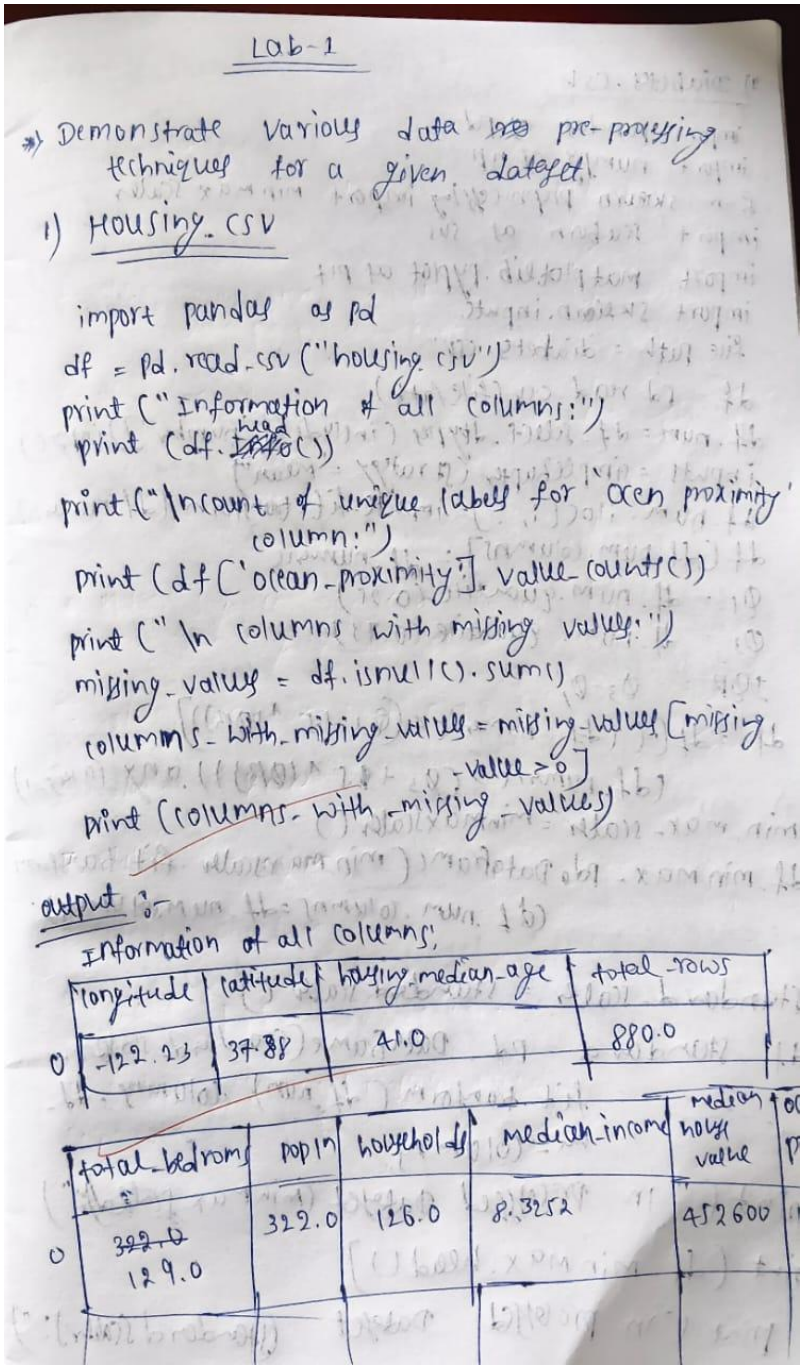


GithubLink: https://github.com/AdityaMK15/machine-learning-lab

**Program 1**

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

Screenshot



Lab-1

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

1) Housing.csv

```
import pandas as pd
df = pd.read_csv("housing.csv")
print("Information of all columns:")
print(df.head())
print("\nCount of unique labels for 'ocen proximity' column:")
print(df['ocean_proximity'].value_counts())
print("\n In columns with missing values:")
missing_values = df.isnull().sum()
columns_with_missing_values = missing_values[missing_values > 0]
print(columns_with_missing_values)
```

output :-

Information of all columns:

| | longitude | latitude | housing_median_age | total_rows |
|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41.0 | 880.0 |

| | total_bedrooms | popln | households | median_income | median house value |
|---|---|---|---|---|---|
| 0 | 129.0 | 322.0 | 126.0 | 8.3252 | 452600 |

2) Diabetes.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import minmax scaler
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn.inputs.
file_path = 'diabetes.csv'
df = pd.read_csv(file_path)
df_num = df.select_dtypes(include=['number']).copy()
inputs = SimpleImputer(strategy="mean")
df_num.iloc[:, :] = impute.fit_transform(df_num)
df[df_num.columns] = df_numeric
Q1 = df_num.quantile(0.25)
Q3 = df_num.quantile(0.75)
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_min_max = pd.DataFrame(min_max_scaler.fit_transform
                (df_num.columns=df_num.columns).
standard_scaler = standard scaler()
df_standard = pd.DataFrame(standard_scaler.
        fit_transform(df_num), columns=df_
print(df_standard.head()).
```

Diabetes Dataset

1) missing values are present in numerical columns i.e pregnant, which are replaced by the mean of the respective column.

2) No categorical columns, hence no encoding.

3) min-max scaling transforms data to a fixed range [0, 1] using

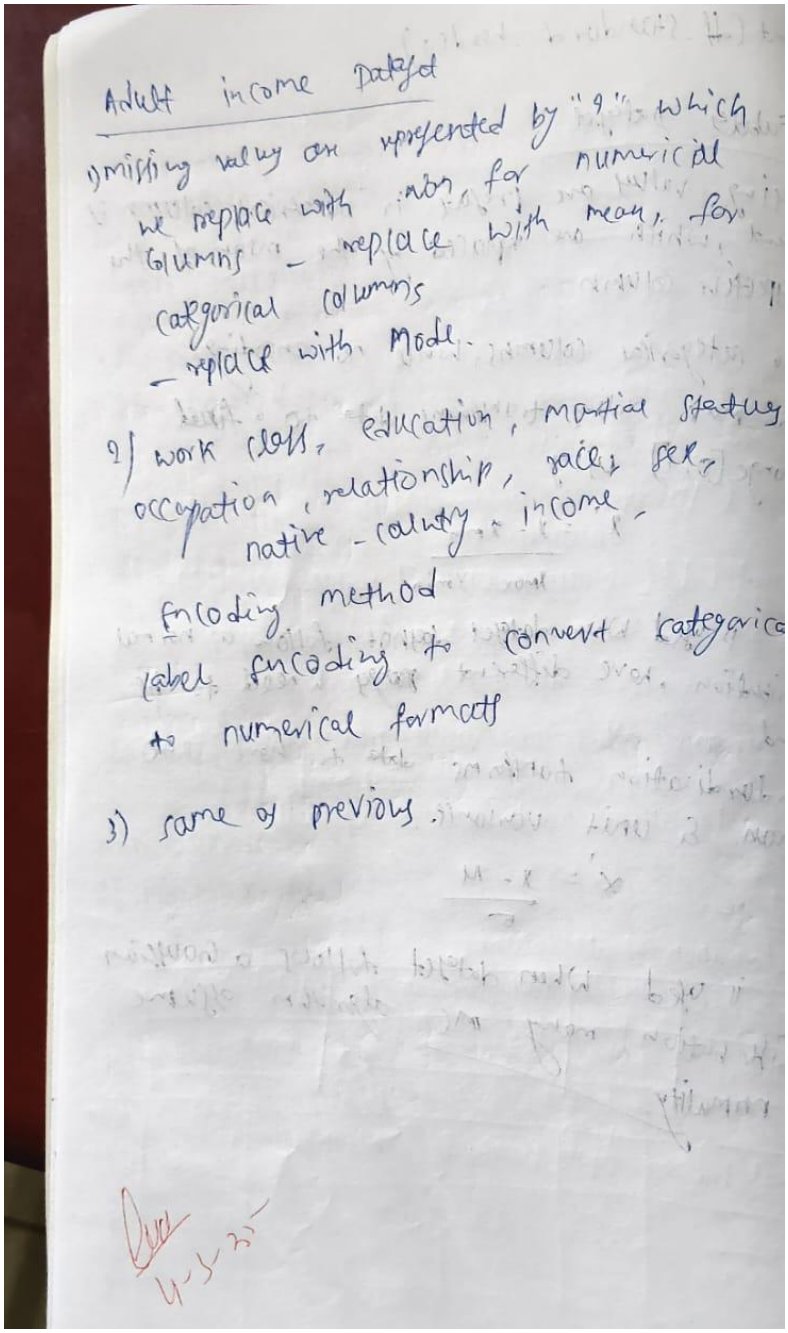
$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

It is used when dataset doesnot follow a normal distribution, have different ranges & need to be bound

Standardization transforms data to have zero mean & unit variance.

$$x' = \frac{x - M}{\sigma}$$

It is used when dataset follows a Gaussian distribution, many ML algorithm assume normality.




Code:

import yfinance as yf

import pandas as pd

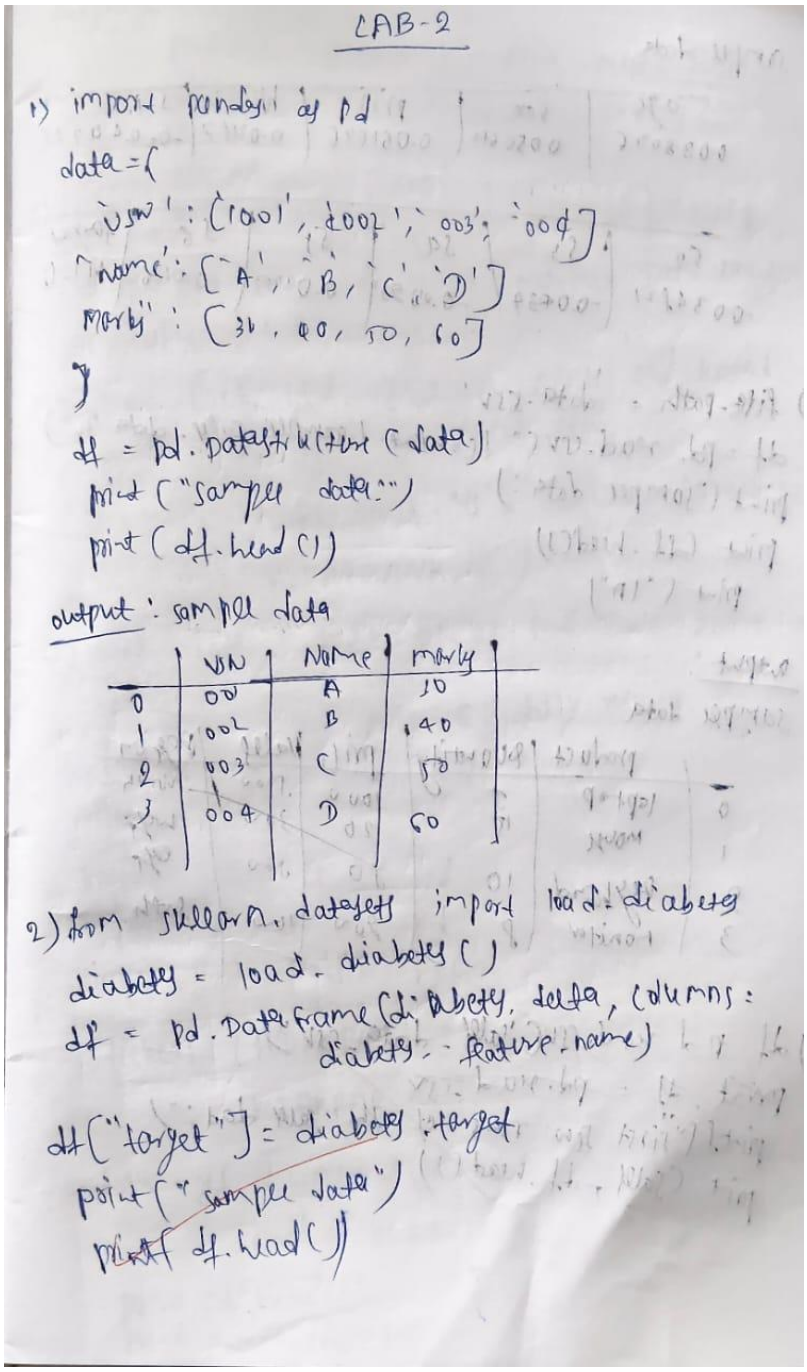
```
import matplotlib.pyplot as plt

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)

hdfc_data = data['HDFCBANK.NS']

print("\nSummary statistics for HDFC Bank:")

print(hdfc_data.describe())

hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

icici_data = data['ICICIBANK.NS']

print("\nSummary statistics for ICICI Bank:")

print(icici_data.describe())

icici_data['Daily Return'] = icici_data['Close'].pct_change()

kotak_data = data['KOTAKBANK.NS']

print("\nSummary statistics for Kotak Mahindra Bank:")

print(kotak_data.describe())


kotak_data['Daily Return'] = kotak_data['Close'].pct_change()

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

plt.subplot(3, 2, 1)
```

```
hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")
plt.subplot(3, 2, 2)
hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Returns", color='orange')
plt.subplot(3, 2, 3)
icici_data['Close'].plot(title="ICICI Bank - Closing Price")
plt.subplot(3, 2, 4)
icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')
plt.subplot(3, 2, 5)
kotak_data['Close'].plot(title="Kotak Mahindra Bank - Closing Price")
plt.subplot(3, 2, 6)
kotak_data['Daily Return'].plot(title="Kotak Mahindra Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

hdfc_data.to_csv('hdfc_bank_data.csv')
icici_data.to_csv('icici_bank_data.csv')
kotak_data.to_csv('kotak_bank_data.csv')

print("\nHDFC Bank data saved to 'hdfc_bank_data.csv'.")
print("ICICI Bank data saved to 'icici_bank_data.csv'.")
print("Kotak Bank data saved to 'kotak_bank_data.csv'.")
```

**Program 2**

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



LAB-2

1) import pandas as pd

```
data = {
    'USN': ['001', '002', '003', '004'],
    'name': ['A', 'B', 'C', 'D'],
    'marks': [30, 40, 50, 60]
}
df = pd.DataFrame(data)
print("sample data:")
print(df.head())
```

output: sample data

| | USN | Name | marks |
|---|---|---|---|
| 0 | 001 | A | 30 |
| 1 | 002 | B | 40 |
| 2 | 003 | C | 50 |
| 3 | 004 | D | 60 |

2) from sklearn.datasets import load_diabetes

```
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_name)
df['target'] = diabetes.target
print("sample data")
print(df.head())
```

Sample data

| age | sex | bmi | bp | s1 |
|---|---|---|---|---|
| 0.038076 | 0.050680 | 0.061696 | 0.021872 | -0.044223 |

| s2 | s3 | s4 | s5 | s6 | target |
|---|---|---|---|---|---|
| -0.034821 | -0.043401 | -0.002592 | 0.019907 | -0.017646 | 151.0 |

3) file_path = 'data.csv'
df = pd.read_csv("/content/sample_sales_data.csv")
print("sample data:")
print(df.head())
print("\n")

Output:
sample data

| | product | quantity | price | sales | region |
|---|---|---|---|---|---|
| 0 | laptop | 5 | 1000 | 5000 | North |
| 1 | Mouse | 15 | 20 | 300 | West |
| 2 | keyboard | 10 | 50 | 500 | East |
| 3 | Monitor | 8 | 200 | 1600 | South |

4) df = pd.read_csv('sales_data.csv')
print(df = pd.read_csv
print("first few rows of the sales data:")
print(sales, df.head())

output:-

| | product | quantity | price | sales | Region |
|---|---|---|---|---|---|
| 0 | laptop | 5 | 1000 | 5000 | North |

5) Bank Data Analysis

using the code of stock market Data analysis consider the following

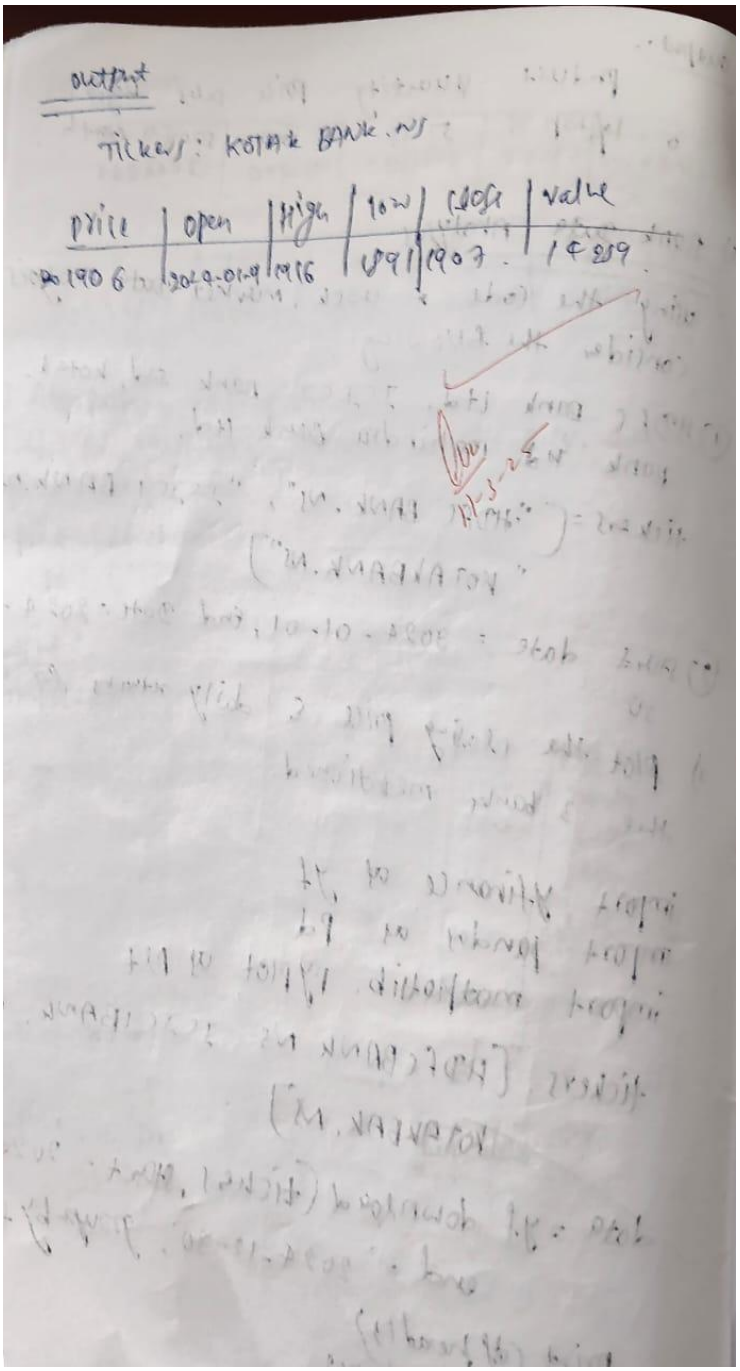
(1) HDFC Bank Ltd, ICICI Bank Ltd, kotak Bank Ltd, mahindra Bank Ltd.

tickers = ["HDFC BANK.NS", "ICICI BANK.NS", "KOTAKBANK.NS"]

(2) start date = 2024-01-01, End Date = 2024-12-30

1) plot the closing price & daily returns for all the 3 banks mentioned.

import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers. [HDFCBANK.NS ICICIBANK.NS, KOTAKBAK.NS]
data = yf.download(tickers, start = 2024-01-01'




Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
```

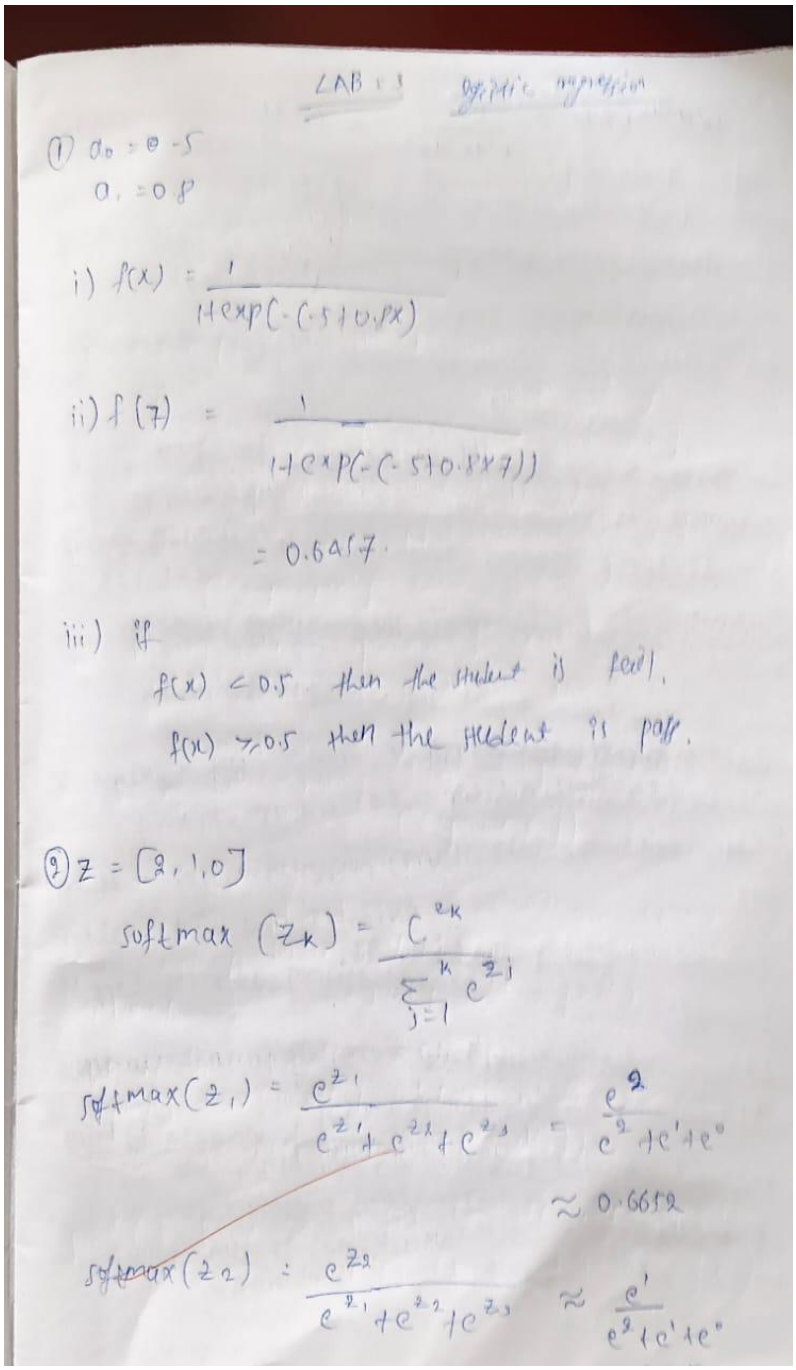
```
import matplotlib.pyplot as plt

diabetes_data = pd.read_csv('/content/Dataset of Diabetes .csv')
adult_income_data = pd.read_csv('/content/adult.csv')

print("Diabetes Dataset:")
print(diabetes_data.head())

print("\nAdult Income Dataset:")
print(adult_income_data.head())

diabetes_numerical_cols = diabetes_data.select_dtypes(include=[np.number]).columns
diabetes_categorical_cols = diabetes_data.select_dtypes(include=[object]).columns

diabetes_imputer_num = SimpleImputer(strategy='median')
diabetes_data[diabetes_numerical_cols] =
diabetes_imputer_num.fit_transform(diabetes_data[diabetes_numerical_cols])

diabetes_imputer_cat = SimpleImputer(strategy='most_frequent')
diabetes_data[diabetes_categorical_cols] =
diabetes_imputer_cat.fit_transform(diabetes_data[diabetes_categorical_cols])

adult_income_numerical_cols = adult_income_data.select_dtypes(include=[np.number]).columns
adult_income_categorical_cols = adult_income_data.select_dtypes(include=[object]).columns

adult_income_imputer_num = SimpleImputer(strategy='median')
adult_income_data[adult_income_numerical_cols] =
adult_income_imputer_num.fit_transform(adult_income_data[adult_income_numerical_cols])

adult_income_imputer_cat = SimpleImputer(strategy='most_frequent')
adult_income_data[adult_income_categorical_cols] =
adult_income_imputer_cat.fit_transform(adult_income_data[adult_income_categorical_cols])

categorical_columns_adult = adult_income_data.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()

for col in categorical_columns_adult:
    adult_income_data[col] = label_encoder.fit_transform(adult_income_data[col])

def detect_and_remove_outliers(df):
    numerical_df = df.select_dtypes(include=[np.number])
    Q1 = numerical_df.quantile(0.25)
    Q3 = numerical_df.quantile(0.75)
    IQR = Q3 - Q1
    return df[~((numerical_df< (Q1 - 1.5 * IQR)) | (numerical_df> (Q3 + 1.5 * IQR))).any(axis=1)]

diabetes_data_cleaned = detect_and_remove_outliers(diabetes_data)
adult_income_data_cleaned = detect_and_remove_outliers(adult_income_data)
```

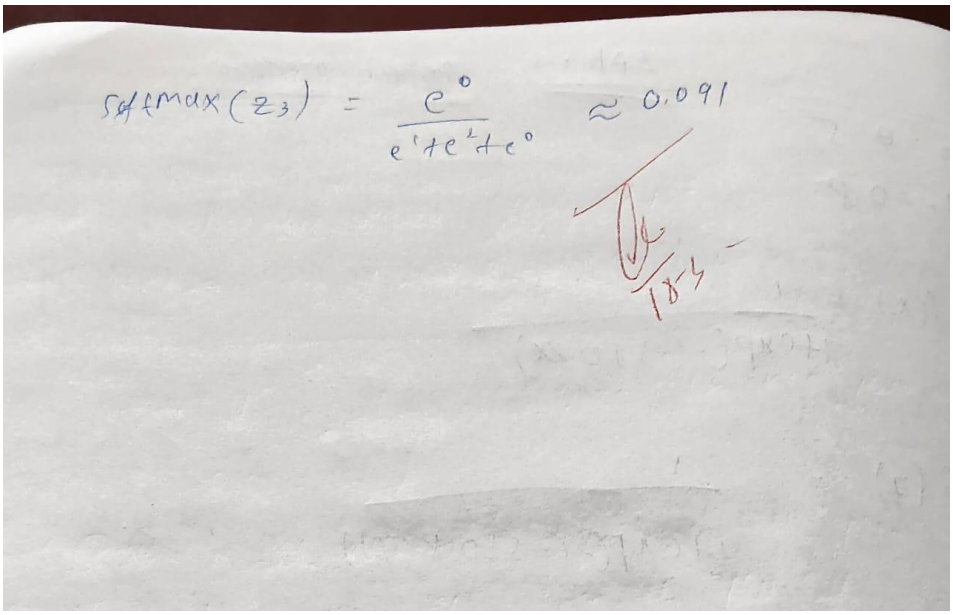
```
min_max_scaler = MinMaxScaler()

diabetes_numerical_cols = diabetes_data_cleaned.select_dtypes(include=[np.number]).columns
diabetes_data_normalized = diabetes_data_cleaned.copy()

diabetes_data_normalized[diabetes_numerical_cols] =
min_max_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_numerical_cols =
adult_income_data_cleaned.select_dtypes(include=[np.number]).columns
adult_income_data_normalized = adult_income_data_cleaned.copy()

adult_income_data_normalized[adult_income_numerical_cols] =
min_max_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])

standard_scaler = StandardScaler()

diabetes_data_standardized = diabetes_data_cleaned.copy()
diabetes_data_standardized[diabetes_numerical_cols] =
standard_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_data_standardized = adult_income_data_cleaned.copy()
adult_income_data_standardized[adult_income_numerical_cols] =
standard_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])
```

## Program 3

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

Screenshot:






Code:

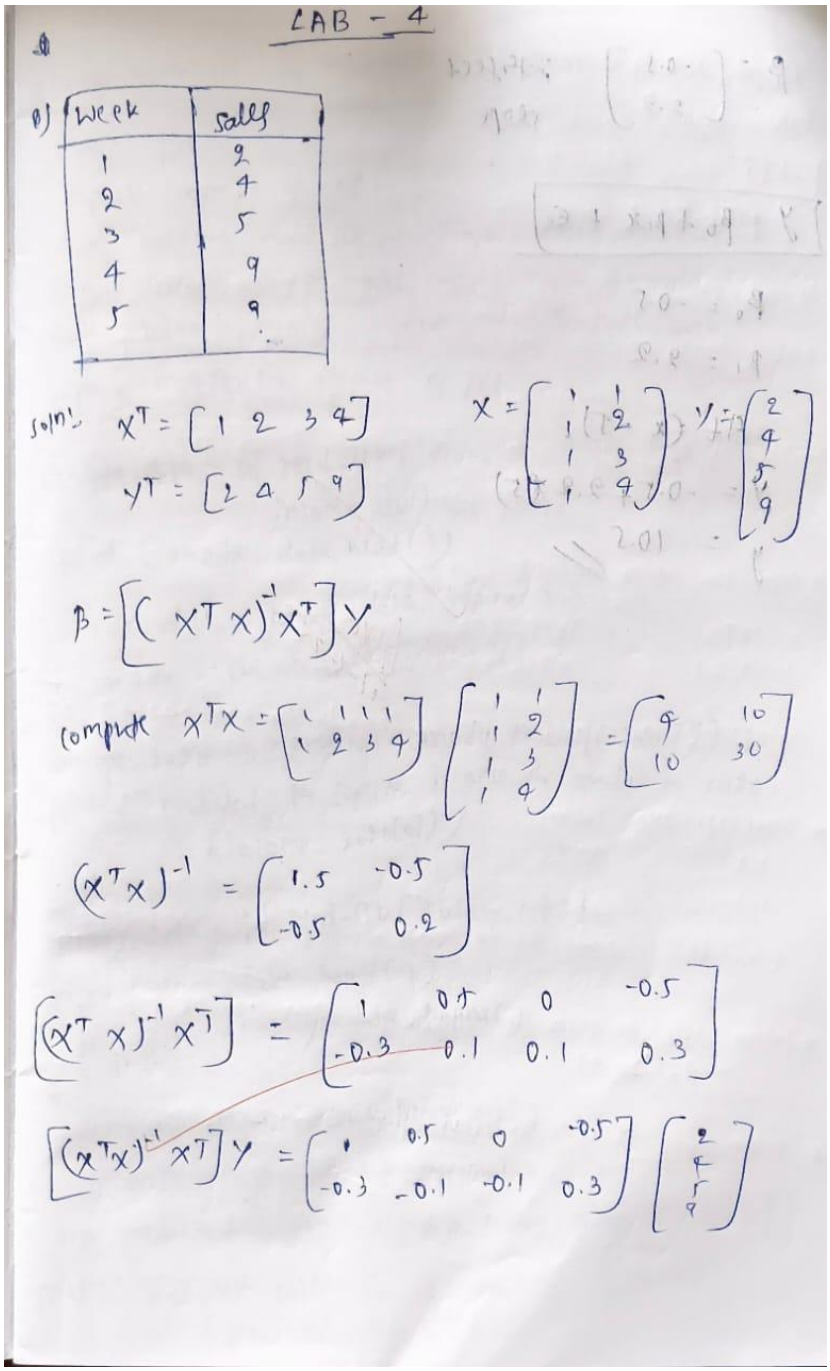
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
mean_absolute_error, mean_squared_error
from sklearn.preprocessing import LabelEncoder

iris = pd.read_csv("/content/iris (4).csv")
drug = pd.read_csv("/content/drug.csv")
petrol = pd.read_csv("/content/petrol_consumption.csv")

X_iris = iris.iloc[:, :-1]
y_iris = iris.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("Decision Tree Classification for IRIS Dataset:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

X_drug = drug.iloc[:, :-1]
```

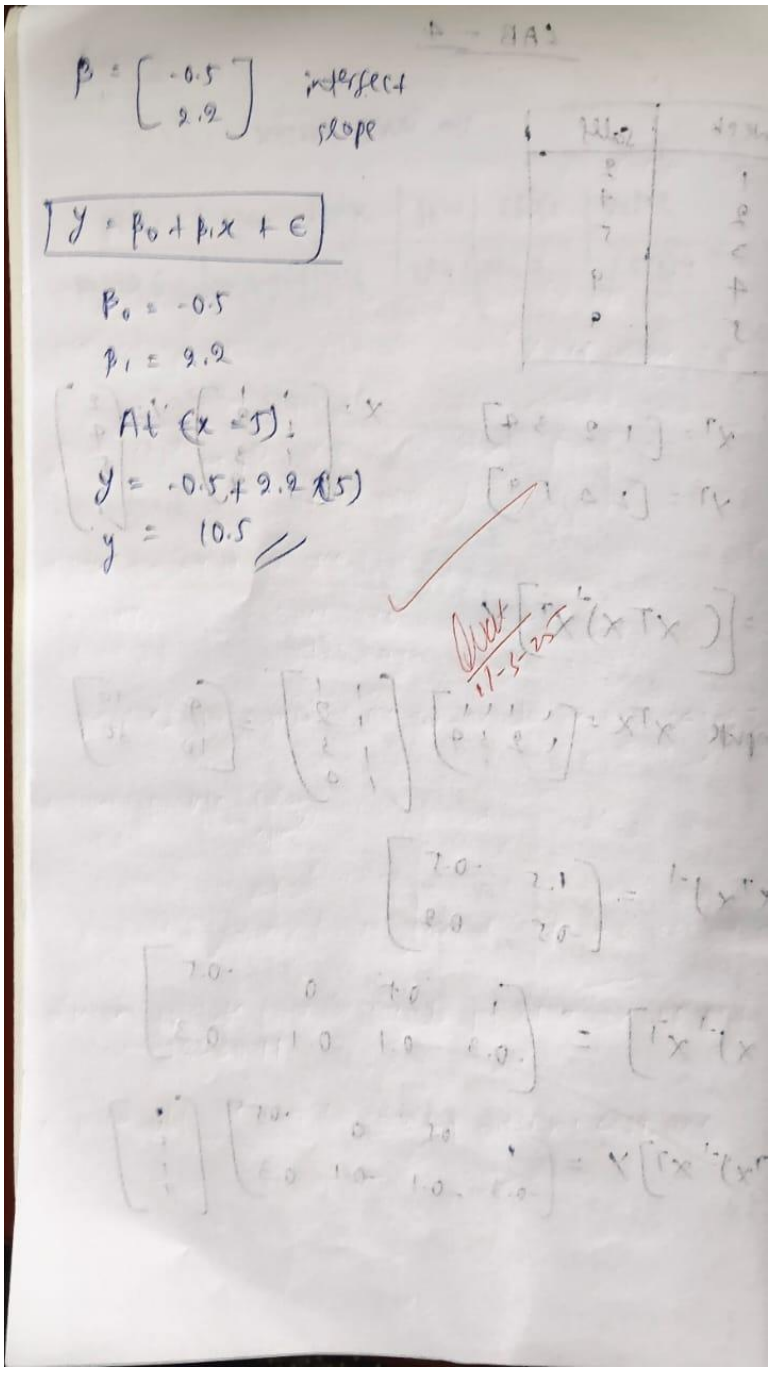
```
y_drug = drug.iloc[:, -1]

le = LabelEncoder()

for col in X_drug.select_dtypes(include=['object']).columns:
    X_drug[col] = le.fit_transform(X_drug[col])

X_train, X_test, y_train, y_test = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("\nDecision Tree Classification for Drug Dataset:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

X_petrol = petrol.iloc[:, :-1]
y_petrol = petrol.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_petrol, y_petrol, test_size=0.2, random_state=42)

dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
y_pred = dtr.predict(X_test)

print("\nDecision Tree Regression for Petrol Consumption:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

**Program 4**

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot






Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error
```

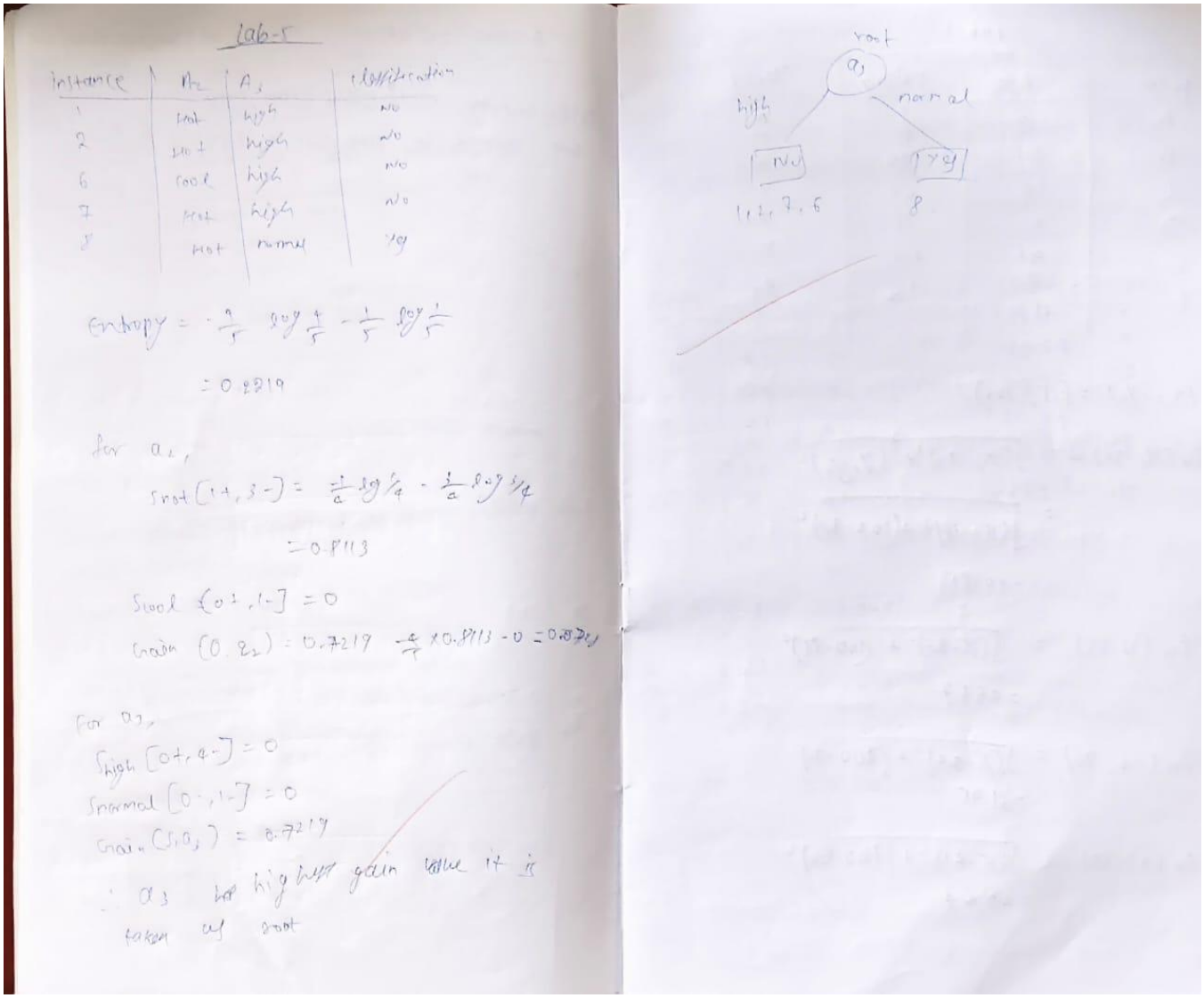
```
import matplotlib.pyplot as plt

hiring_data = pd.read_csv('hiring.csv')
print(hiring_data.head())
hiring_data = hiring_data.dropna()

experience_mapping = {
    'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'seven': 7, 'eight': 8,
    'nine': 9, 'ten': 10, 'eleven': 11, 'twelve': 12, 'thirteen': 13, 'fourteen': 14,
}

hiring_data['experience'] = hiring_data['experience'].replace(experience_mapping)
hiring_data['experience'] = pd.to_numeric(hiring_data['experience'], errors='coerce')

if hiring_data['experience'].isnull().any():
    print("Warning: There are still non-numeric values in the 'experience' column.")
    hiring_data = hiring_data.dropna(subset=['experience'])

X_hiring = hiring_data[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = hiring_data['salary($)']

X_train_hiring, X_test_hiring, y_train_hiring, y_test_hiring = train_test_split(X_hiring, y_hiring,
test_size=0.2, random_state=42)

regressor_hiring = LinearRegression()
regressor_hiring.fit(X_train_hiring, y_train_hiring)

candidate_1 = np.array([[2, 9, 6]])
candidate_2 = np.array([[12, 10, 10]])

salary_1 = regressor_hiring.predict(candidate_1)
salary_2 = regressor_hiring.predict(candidate_2)

print(f"Predicted salary for candidate 1 (2 yr experience, 9 test score, 6 interview score):
{salary_1[0]}")
print(f"Predicted salary for candidate 2 (12 yr experience, 10 test score, 10 interview score):
{salary_2[0]}")



companies_data = pd.read_csv('/content/1000_Companies.csv')
print(companies_data.head())
companies_data = companies_data.dropna()

label_encoder = LabelEncoder()
companies_data['State'] = label_encoder.fit_transform(companies_data['State'])

X_companies = companies_data[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
```

```
y_companies = companies_data['Profit']

X_train_companies, X_test_companies, y_train_companies, y_test_companies =
train_test_split(X_companies, y_companies, test_size=0.2, random_state=42)

regressor_companies = LinearRegression()
regressor_companies.fit(X_train_companies, y_train_companies)

input_data = np.array([[91694.48, 515841.3, 11931.24, label_encoder.transform(['Florida'])[0]]])
predicted_profit = regressor_companies.predict(input_data)

print(f"Predicted profit for the given inputs (Florida State): {predicted_profit[0]}")

y_pred_hiring = regressor_hiring.predict(X_test_hiring)
mae_hiring = mean_absolute_error(y_test_hiring, y_pred_hiring)
print(f"Mean Absolute Error for Salary Prediction: {mae_hiring}")

y_pred_companies = regressor_companies.predict(X_test_companies)
mae_companies = mean_absolute_error(y_test_companies, y_pred_companies)
print(f"Mean Absolute Error for Profit Prediction: {mae_companies}")
```

**Program 5**

Build Logistic Regression Model for a given dataset

Screenshot



Code:

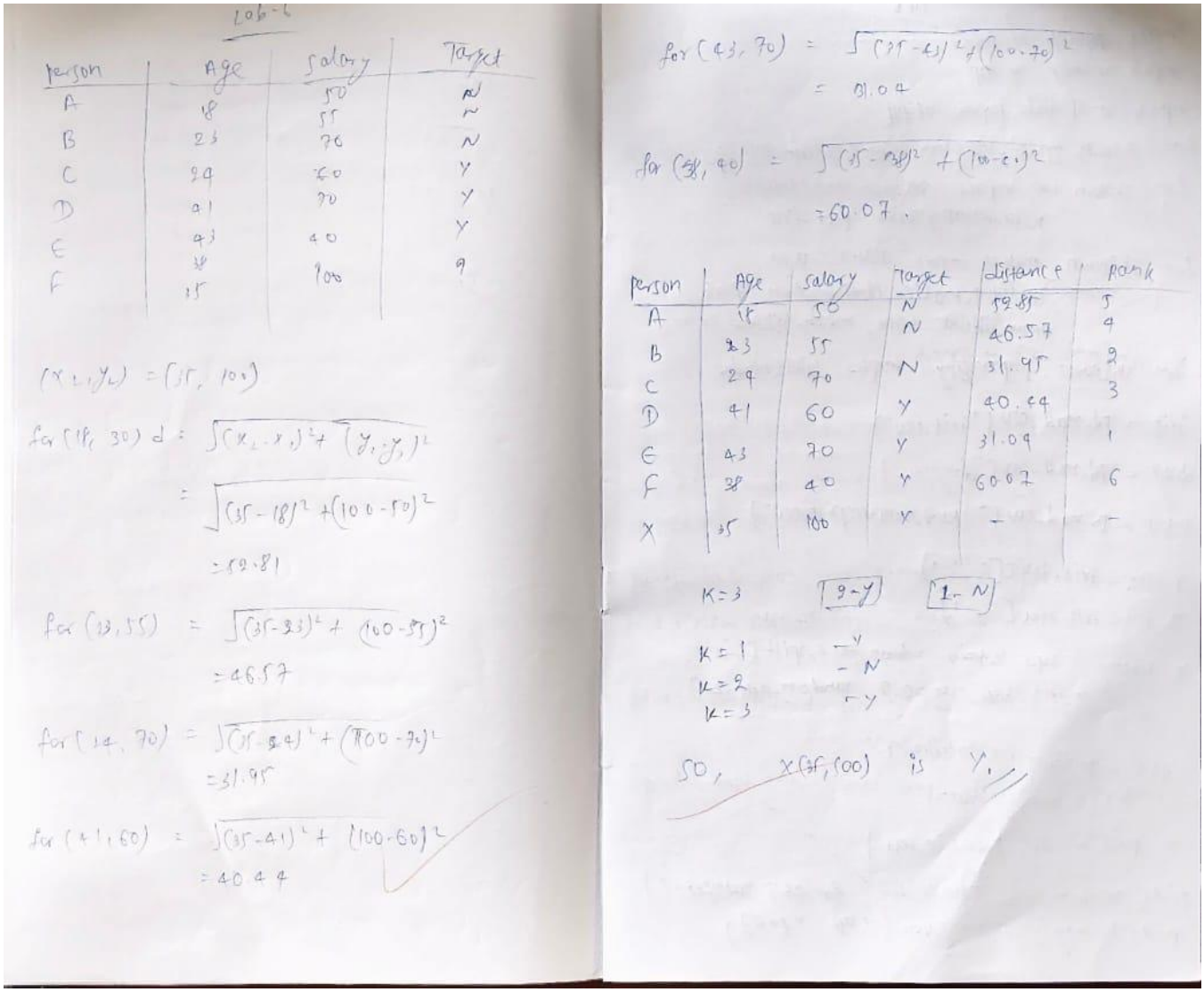
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

file_path = 'HR_comma_sep.csv'
data = pd.read_csv(file_path)

print(data.info())

print(data.head())

print(data.describe())

plt.figure(figsize=(8, 5))
sns.countplot(x='salary', hue='left', data=data)
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(x='Department', hue='left', data=data)
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.xticks(rotation=45)
plt.show()

data_encoded = pd.get_dummies(data, columns=['salary', 'Department'], drop_first=True)

print(data_encoded.info())

X = data_encoded.drop('left', axis=1)
y = data_encoded['left']

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
logreg = LogisticRegression(max_iter=1000)

logreg.fit(X_train_scaled, y_train)

y_pred = logreg.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression Model: {accuracy * 100:.2f}%")

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Stayed', 'Left'],
yticklabels=['Stayed', 'Left'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

**Program 6**

Build KNN Classification model for a given dataset.

Screenshot



Code:

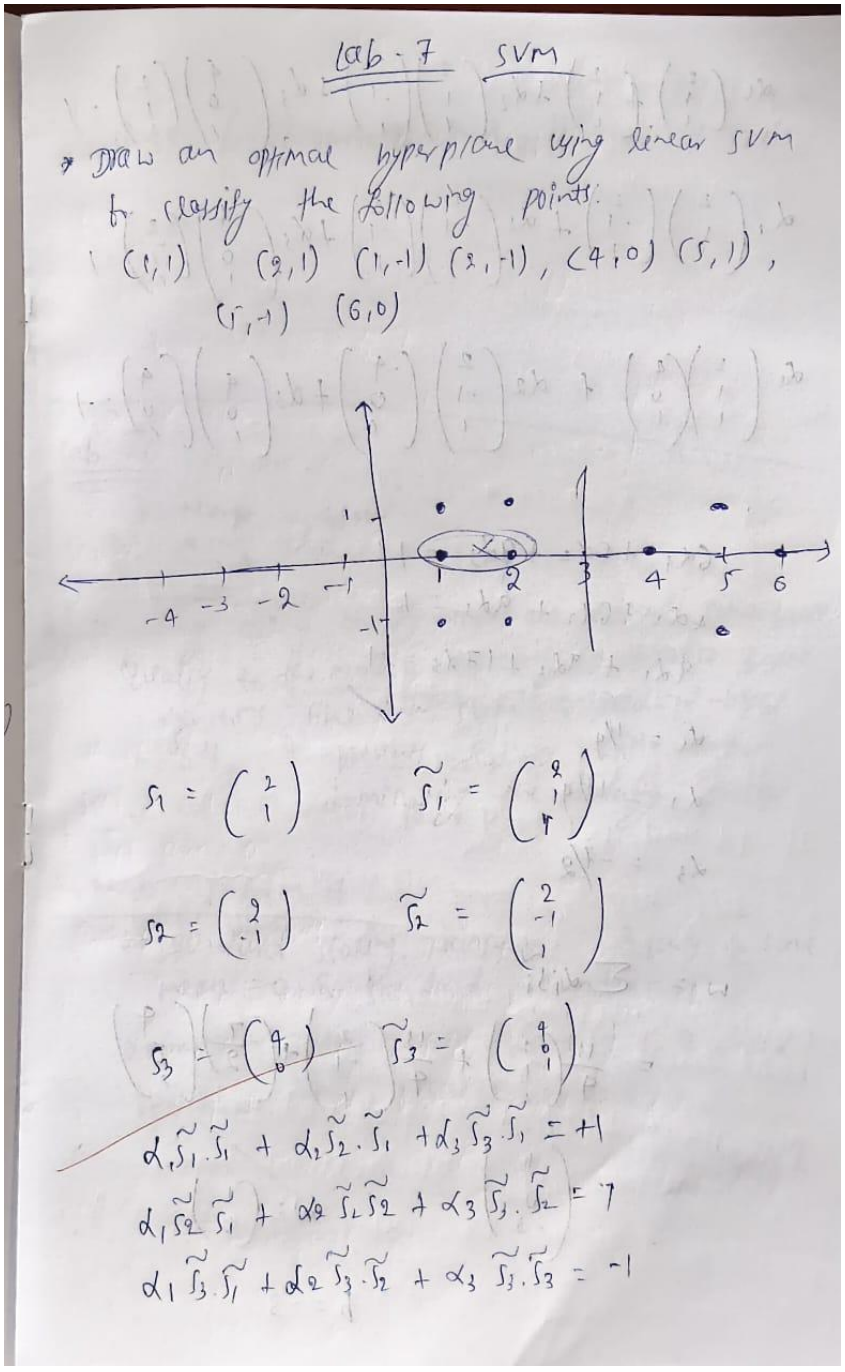
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

iris_df = pd.read_csv('/content/iris (3).csv')

print(iris_df.head())

X_iris = iris_df.drop(columns=['species'])
y_iris = iris_df['species']

X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_iris = scaler.fit_transform(X_train_iris)
X_test_iris = scaler.transform(X_test_iris)

knn_iris = KNeighborsClassifier(n_neighbors=3)

knn_iris.fit(X_train_iris, y_train_iris)

y_pred_iris = knn_iris.predict(X_test_iris)

accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
print(f"Accuracy on Iris test data: {accuracy_iris * 100:.2f}%")

cm_iris = confusion_matrix(y_test_iris, y_pred_iris)
sns.heatmap(cm_iris, annot=True, fmt="d", cmap="Blues", xticklabels=knn_iris.classes_,
yticklabels=knn_iris.classes_)
plt.title("Confusion Matrix for Iris Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Iris Dataset:")
print(classification_report(y_test_iris, y_pred_iris))

diabetes_df = pd.read_csv('diabetes.csv')
print(diabetes_df.head())
```

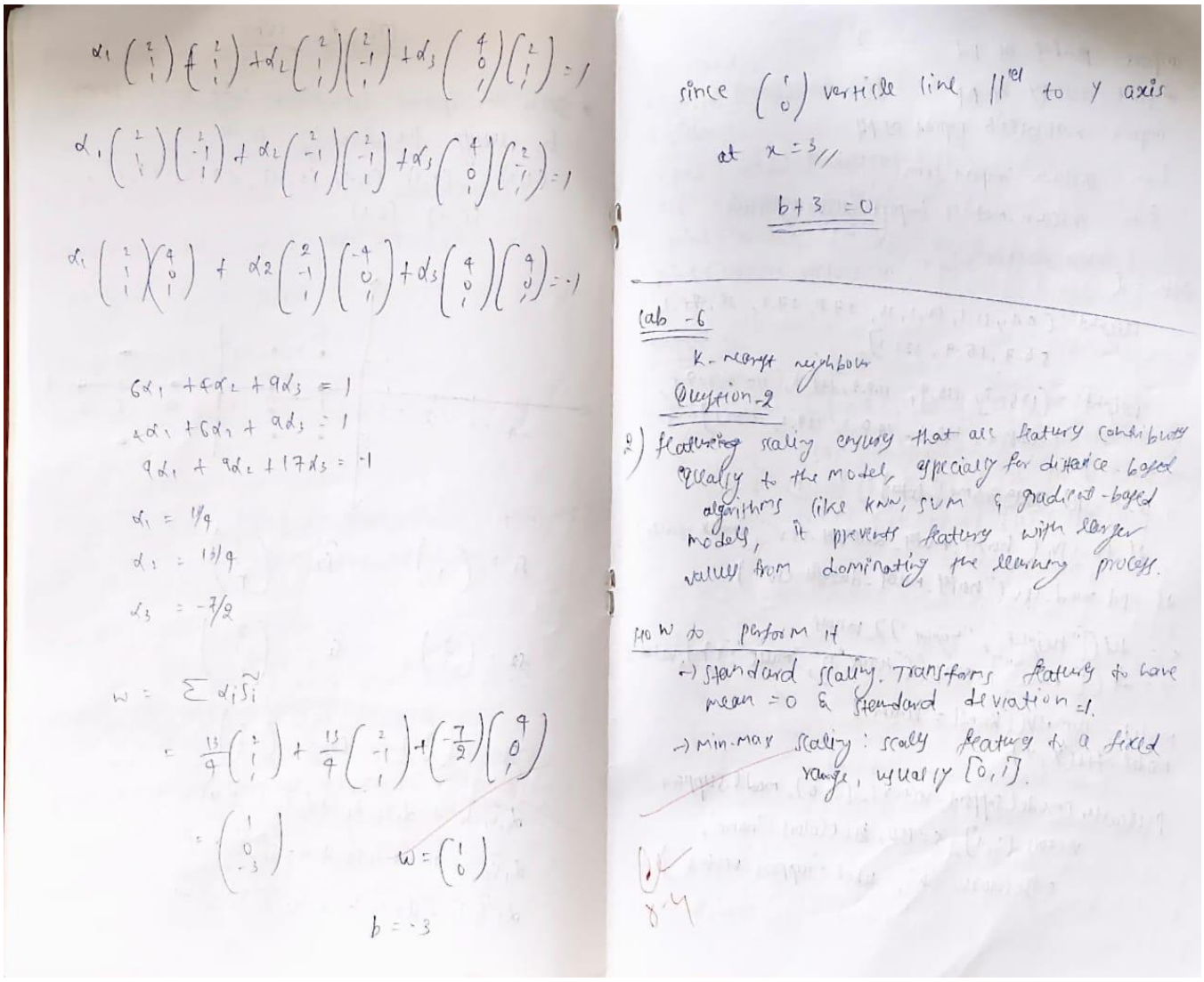
```
X_diabetes = diabetes_df.drop(columns=['Outcome'])
y_diabetes = diabetes_df['Outcome']

X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes = train_test_split(X_diabetes,
y_diabetes, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_diabetes = scaler.fit_transform(X_train_diabetes)
X_test_diabetes = scaler.transform(X_test_diabetes)

knn_diabetes = KNeighborsClassifier(n_neighbors=5)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)

accuracy_diabetes = accuracy_score(y_test_diabetes, y_pred_diabetes)
print(f"Accuracy on Diabetes test data: {accuracy_diabetes * 100:.2f}%")

cm_diabetes = confusion_matrix(y_test_diabetes, y_pred_diabetes)
sns.heatmap(cm_diabetes, annot=True, fmt="d", cmap="Blues", xticklabels=knn_diabetes.classes_,
yticklabels=knn_diabetes.classes_)
plt.title("Confusion Matrix for Diabetes Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Diabetes Dataset:")
print(classification_report(y_test_diabetes, y_pred_diabetes))
```

**Program 7**

Build Support vector machine model for a given dataset

Screenshot





since $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ vertical line, ⊥ to y axis

at $x = 3$

$b + 3 = 0$

lab - 6

K - nearest neighbour

Question-2

2) Feature scaling ensures that all features contribute equally to the model, especially for distance based algorithms like KNN, SVM & gradient-based models, it prevents features with larger values from dominating the learning process.

How to perform it

→ Standard scaling: Transforms feature to have mean = 0 & standard deviation = 1.

→ Min-Max scaling: scale feature to a fixed range, usually [0, 1].

Code:

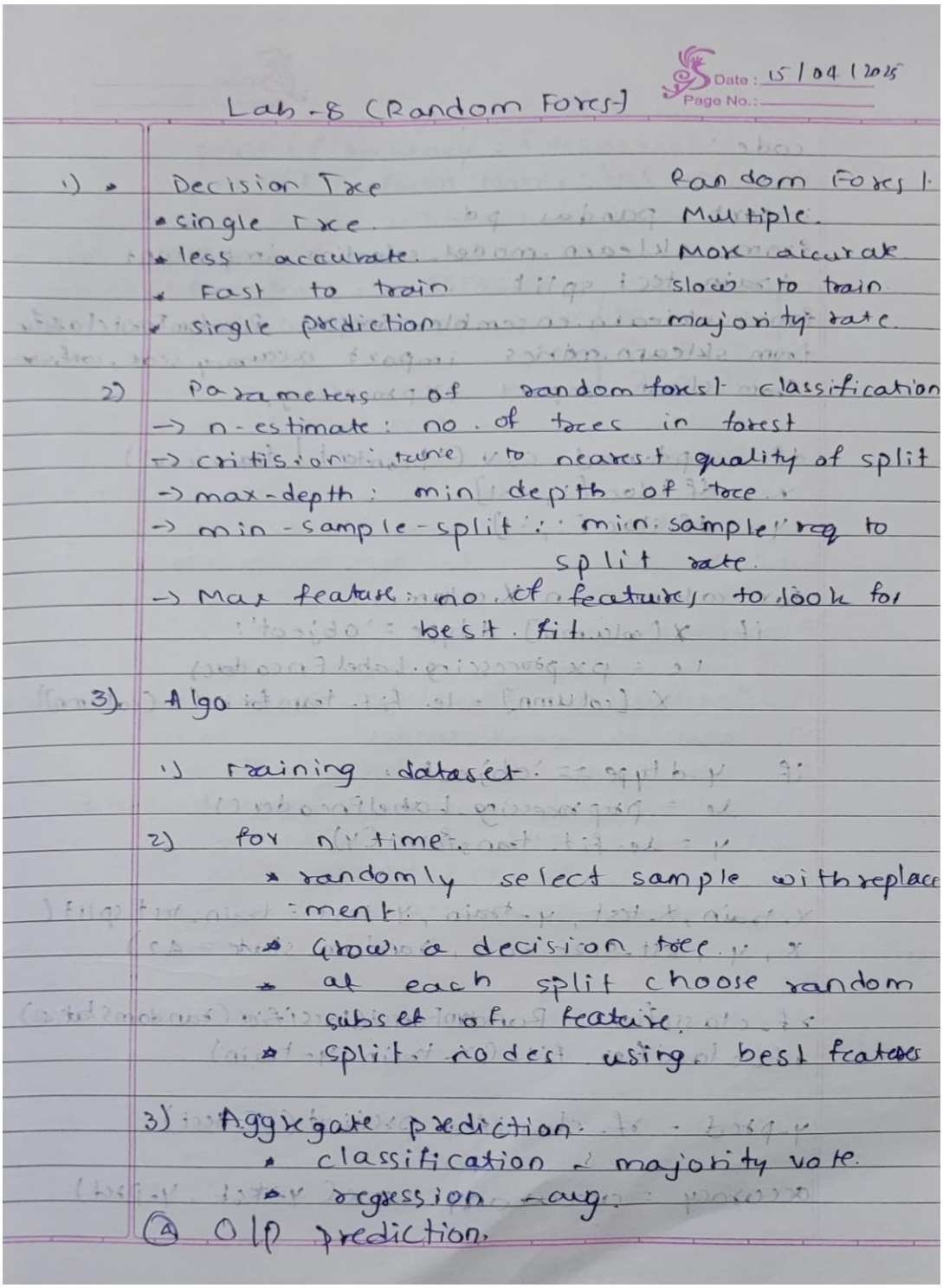
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import LabelEncoder, label_binarize
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

df = pd.read_csv("/content/letter-recognition.csv")

top_classes = df['letter'].value_counts().head(5).index.tolist()
df = df[df['letter'].isin(top_classes)]

X = df.iloc[:, 1:]
y = df.iloc[:, 0]

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

y_bin = label_binarize(y_encoded, classes=np.unique(y_encoded))
n_classes = y_bin.shape[1]

X_train, X_test, y_train, y_test_bin = train_test_split(X, y_bin, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear', probability=True)
svm_model.fit(X_train, y_train.argmax(axis=1))
y_score = svm_model.predict_proba(X_test)

y_pred = svm_model.predict(X_test)
y_true = y_test_bin.argmax(axis=1)

print("Accuracy:", accuracy_score(y_true, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))

plt.figure()
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:,i], y_score[:,i])
    auc = roc_auc_score(y_test_bin[:,i], y_score[:,i])
    plt.plot(fpr, tpr, label=f"{label_encoder.inverse_transform([i])[0]} AUC={auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve (Top 5 Classes)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()

macro_auc = roc_auc_score(y_test_bin, y_score, average="macro")
print("Macro AUC Score:", macro_auc)
```

**Program 8**

Implement Random forest ensemble method on a given dataset.

Screenshot



Lab-8 (Random Forest)

Date: 15/04/2025
Page No.:

1) Decision Tree
- single tree
- less accurate
- Fast to train
- single prediction

Random Forest
- Multiple
- More accurate
- slow to train
- majority rate

2) Parameters of random forest classification
-> n-estimate: no. of trees in forest
-> criterion: to measure quality of split
-> max-depth: min depth of tree
-> min-sample-split: min sample req to split node
-> Max feature: no. of features to look for best fit

3) Algo
1) training dataset
2) for n times
- randomly select sample with replacement
- Grow a decision tree
- at each split choose random subset of features
- split nodes using best features
3) Aggregate prediction
- classification - majority vote
- regression - avg
4) O/p prediction

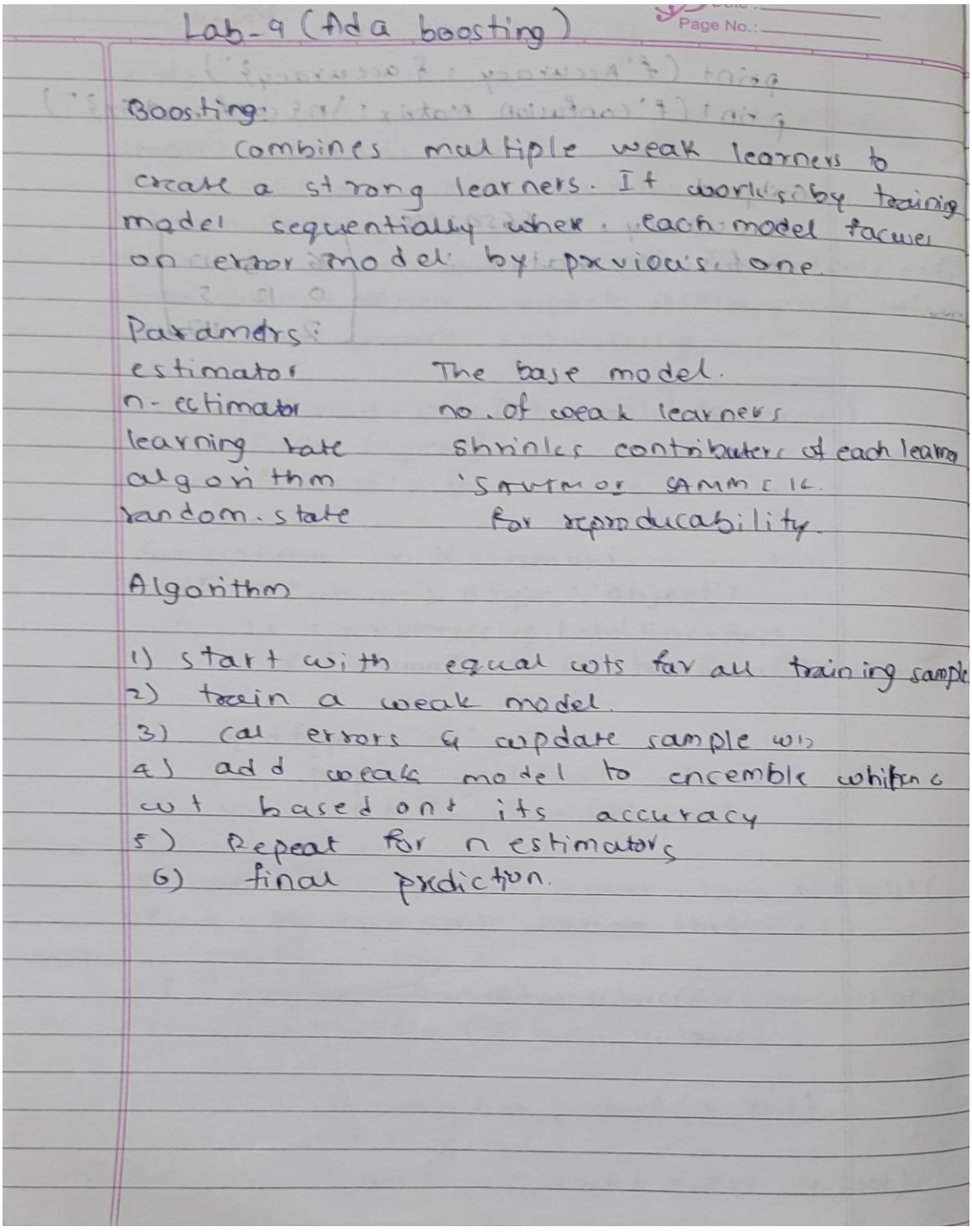
Code:

```
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 import preprocessing

df = pd.read_csv('/content/train.csv')

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

for column in X.columns:
    if X[column].dtype == 'object':
      le = preprocessing.LabelEncoder()
      X[column] = le.fit_transform(X[column])

if y.dtype == 'object':
  le = preprocessing.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)

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)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

**Program 9**

Implement Boosting ensemble method on a given dataset.

Screenshot



Code:

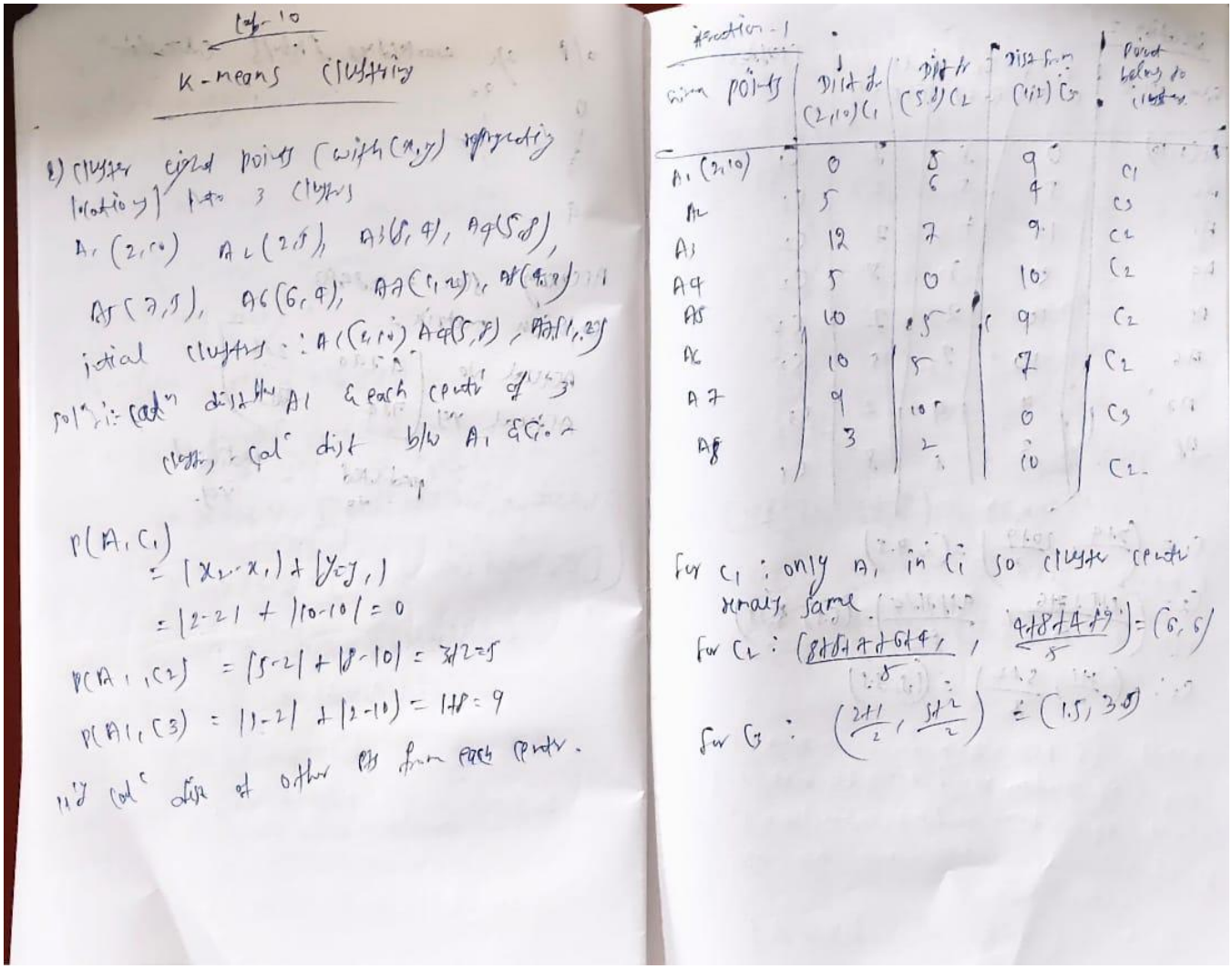
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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, random_state=42)

results = []

n_estimators_list = [10, 50, 100]
learning_rates = [0.01, 0.1, 1]

for n in n_estimators_list:
    for lr in learning_rates:
        tree_base = DecisionTreeClassifier(max_depth=1)
        model = AdaBoostClassifier(estimator=tree_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'DecisionTree',
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

for n in n_estimators_list:
    for lr in learning_rates:
        log_reg_base = LogisticRegression(max_iter=1000)
        model = AdaBoostClassifier(estimator=log_reg_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'LogisticRegression',
```

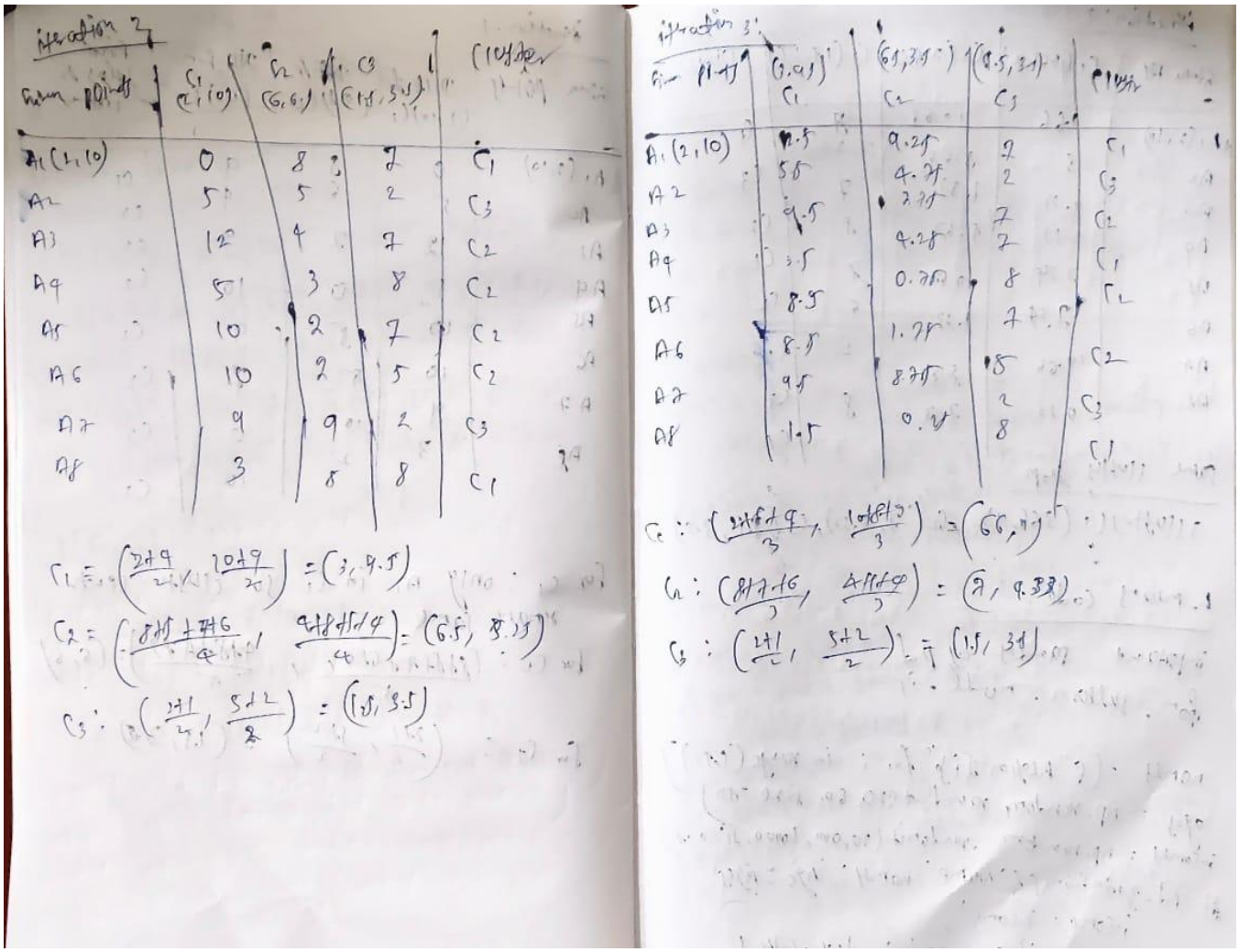
```
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

results_df = pd.DataFrame(results)
print(results_df)

import seaborn as sns
plt.figure(figsize=(12, 6))
sns.barplot(x='n_estimators', y='Accuracy', hue='Base', data=results_df, ci=None)
plt.title('AdaBoost Accuracy with Different Estimators and n_estimators')
plt.show()
```

## Program 10

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

Screenshot





Iteration 2

| Data points | $C_1$ (2, 10) | $C_2$ (6, 6) | $C_3$ (1.5, 3.5) | Cluster |
|---|---|---|---|---|
| $A_1$ (2, 10) | 0 | 8 | 7 | $C_1$ |
| $A_2$ | 5 | 5 | 2 | $C_3$ |
| $A_3$ | 12 | 4 | 7 | $C_2$ |
| $A_4$ | 5 | 3 | 8 | $C_2$ |
| $A_5$ | 10 | 2 | 7 | $C_2$ |
| $A_6$ | 10 | 2 | 5 | $C_2$ |
| $A_7$ | 4 | 4 | 2 | $C_3$ |
| $A_8$ | 3 | 8 | 8 | $C_1$ |

$C_1 = \left(\frac{2+4}{2}, \frac{10+4}{2}\right) = (3, 4.5)$

$C_2 = \left(\frac{6+5+2+2}{4}, \frac{6+3+2+2}{4}\right) = (6.5, 3.25)$

$C_3 = \left(\frac{1+1}{2}, \frac{3+2}{2}\right) = (1.5, 2.5)$

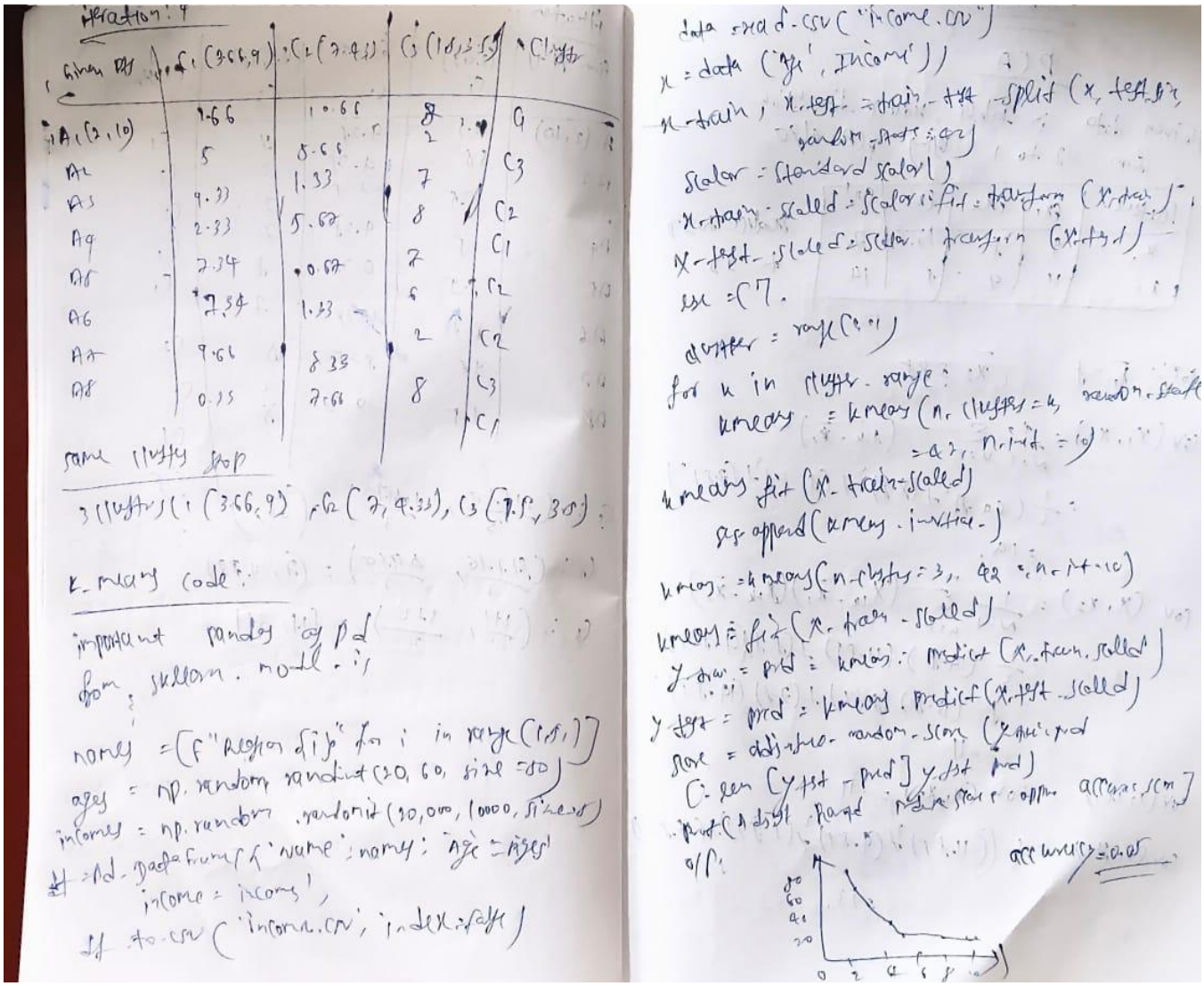
Iteration 3:

| Data points | (3, 4.5) $C_1$ | (6.5, 3.25) $C_2$ | (1.5, 2.5) $C_3$ | Cluster |
|---|---|---|---|---|
| $A_1$ (2, 10) | 6.5 | 9.25 | 7 | $C_1$ |
| $A_2$ | 5.5 | 4.75 | 2 | $C_3$ |
| $A_3$ | 9.5 | 2.75 | 7 | $C_2$ |
| $A_4$ | 3.5 | 4.25 | 7 | $C_2$ |
| $A_5$ | 8.5 | 0.75 | 8 | $C_2$ |
| $A_6$ | 8.5 | 1.75 | 7 | $C_2$ |
| $A_7$ | 4.5 | 3.75 | 5 | $C_2$ |
| $A_8$ | 1.5 | 0.25 | 2 | $C_3$ |

$C_1 = \left(\frac{2+4}{2}, \frac{10+4}{2}\right) = (3, 4.5)$

$C_2 = \left(\frac{6+5+2+2}{4}, \frac{6+3+2+2}{4}\right) = (6.5, 3.25)$

$C_3 = \left(\frac{1+1}{2}, \frac{3+2}{2}\right) = (1.5, 2.5)$



Code:

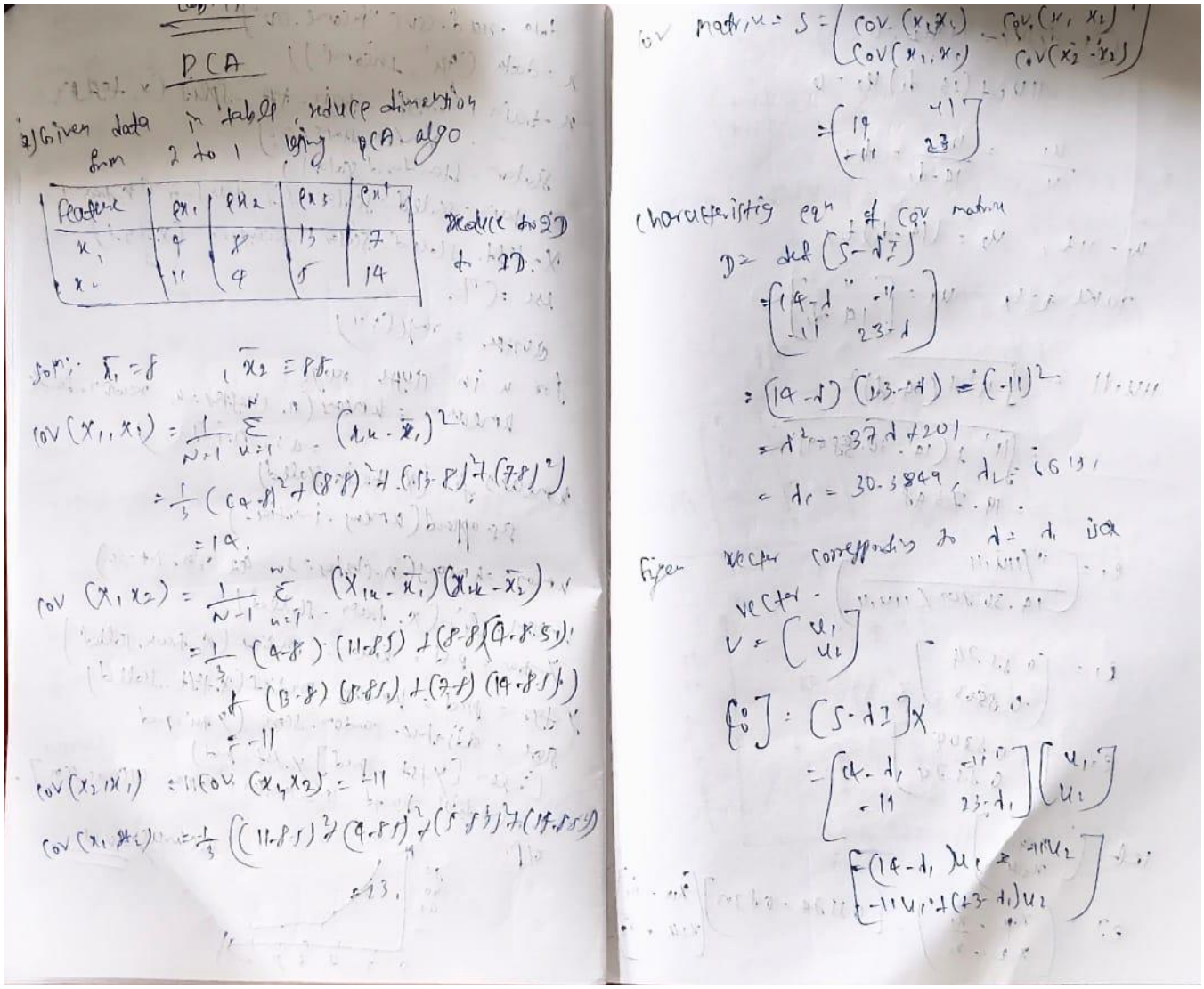
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = {
    'Name': [f'Person_{i+1}' for i in range(50)],
    'Age': np.random.randint(18, 70, size=50),
    'Income': np.random.randint(20000, 120000, size=50)
```

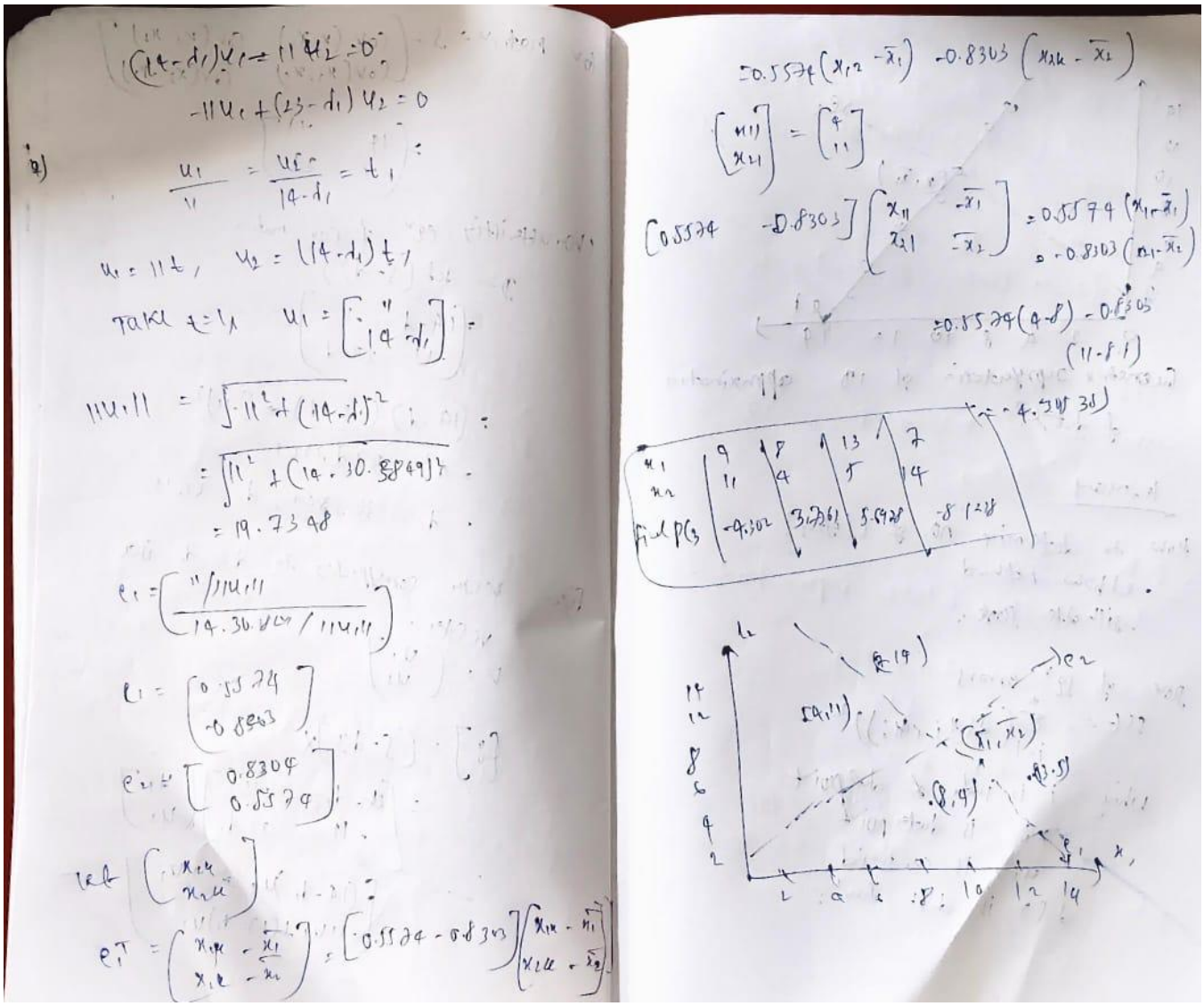
```
}

df = pd.DataFrame(data)

df.to_csv('income.csv', index=False)

df = pd.read_csv('income.csv')

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

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test = train_test_split(X_scaled, test_size=0.2, random_state=42)

sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train)
    sse.append(kmeans.inertia_)

plt.plot(k_range, sse, marker='o')
plt.title('SSE vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.show()

optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train)
y_pred = kmeans.predict(X_test)

print(f'Predicted Clusters for Test Data: {y_pred}')
```

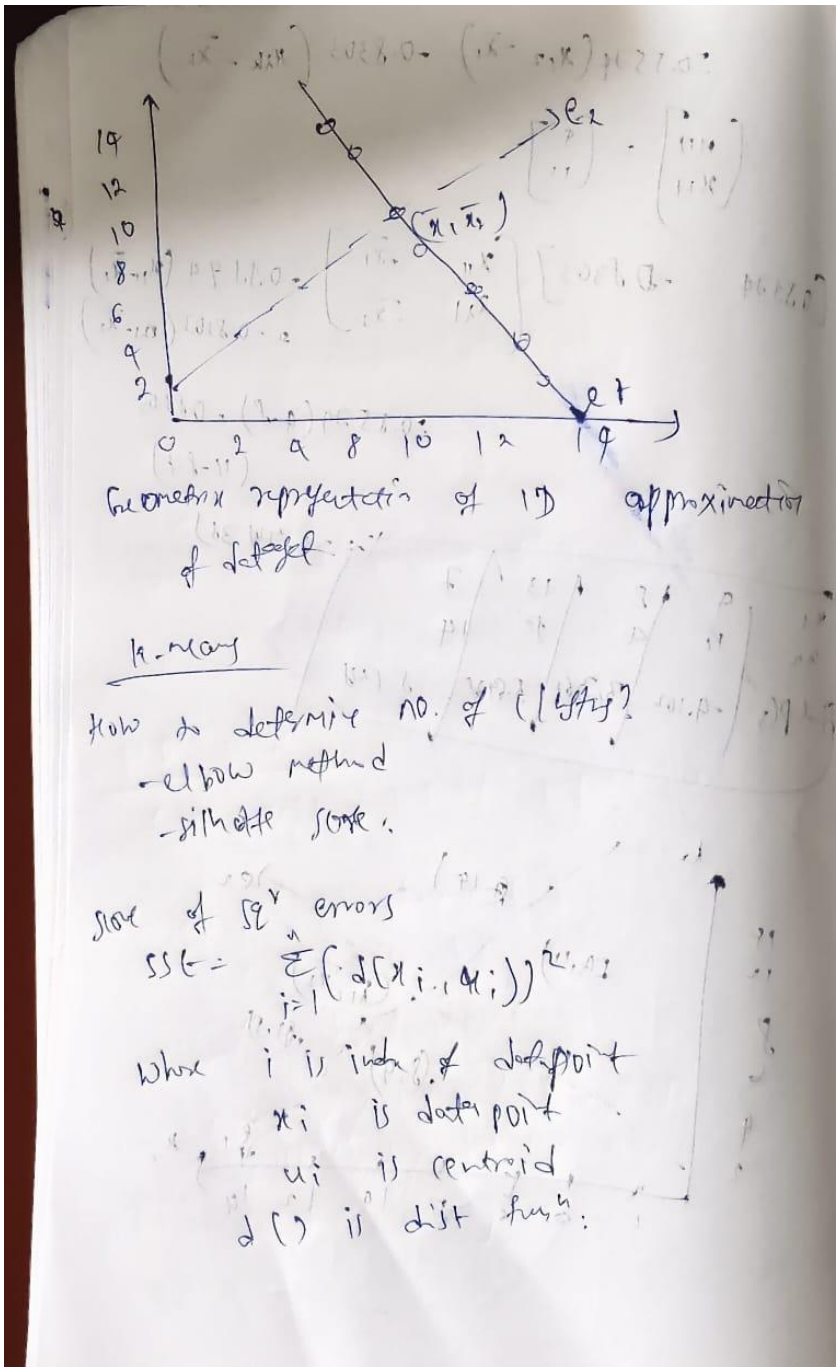
**Program 11**

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

Screenshot








Geometric representation of 1D approximation of dataset

K-means

How to determine no. of clusters?

- elbow method
- silhouette score.

Sum of SQ errors

$$SSE = \sum_{i=1}^{n} (d(x_i, u_i))^2$$

Where i is index of datapoint
$x_i$ is data point
$u_i$ is centroid
$d()$ is dist fun.

Code:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from scipy import stats

df = pd.read_csv('heart (2).csv')

z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
df_no_outliers = df[(z_scores< 3).all(axis=1)]

df_cleaned = df_no_outliers.copy()
for col in df_cleaned.select_dtypes(include='object').columns:
    df_cleaned[col] = LabelEncoder().fit_transform(df_cleaned[col])

X = df_cleaned.drop('HeartDisease', axis=1)
y = df_cleaned['HeartDisease']

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,
stratify=y)

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}

print("Accuracy without PCA:")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")

pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42,
stratify=y)

print("\nAccuracy with PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")
```