VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Pratik Jana(1BM22CS356)

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 **Pratik Jana (1BM22CS356),** 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 a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

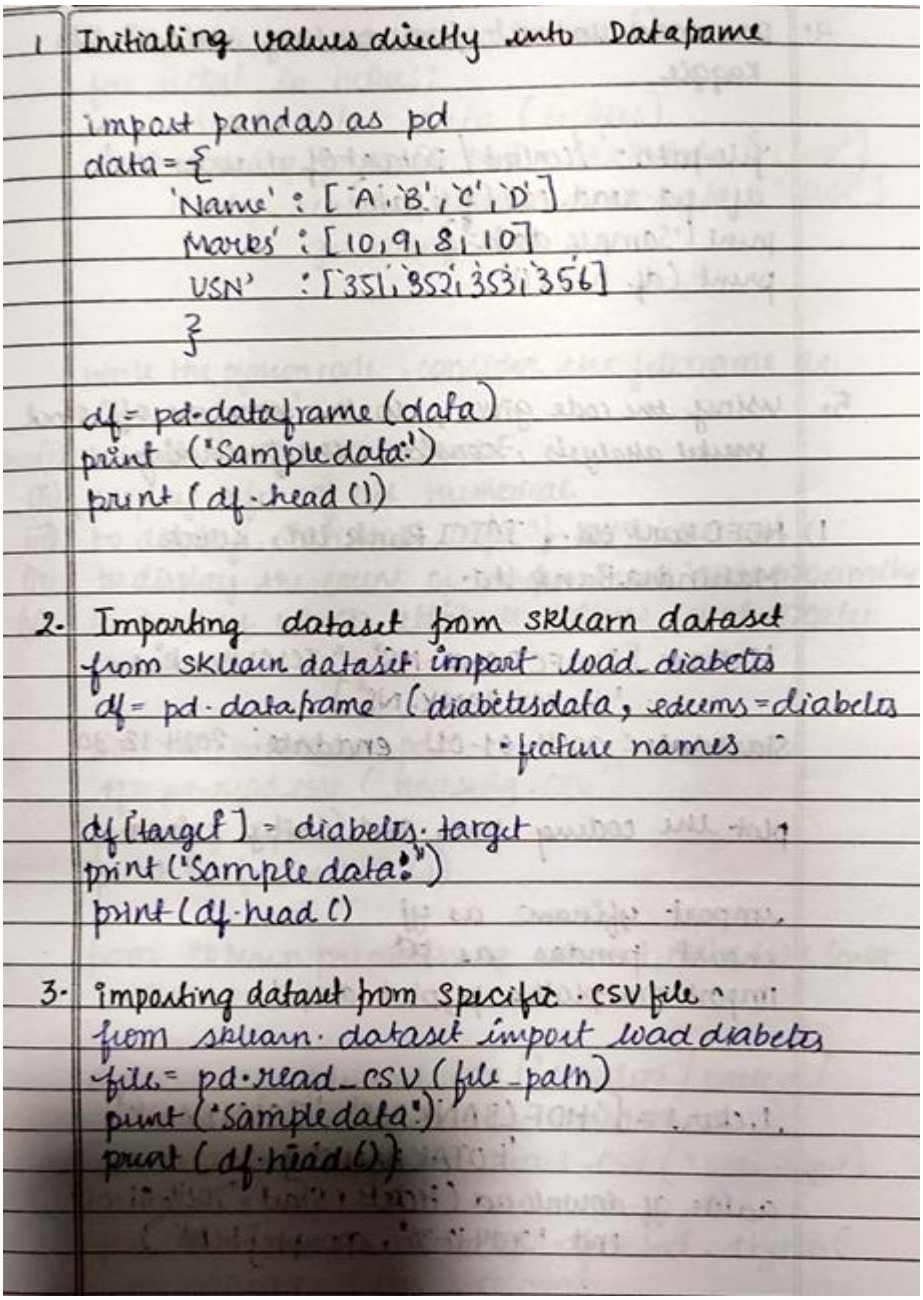
Index

GITHUB LINK: pratik03092003/ML-LAB

Program 1

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

Screenshot



1. Initialing values directly into Dataframe

```
import pandas as pd
data = {
    'Name' : ['A','B','C','D']
    Marks' : [10,9,8,10]
    USN'   : ['351','352','353','356']
    }

df = pd.dataframe(data)
print ("Sample data:")
print (df.head())
```

2. Importing dataset from sklearn dataset

```
from sklearn dataset import load_diabetes
df = pd.dataframe (diabetesdata, columns = diabetes
                   .feature names
df['target'] = diabetes.target
print ("Sample data:")
print (df.head())
```

3. importing dataset from specific .csv file

```
from sklearn.dataset import load diabetes
file = pd.read_csv(file_path)
print ("Sample data:")
print (df.head())
```

Code:

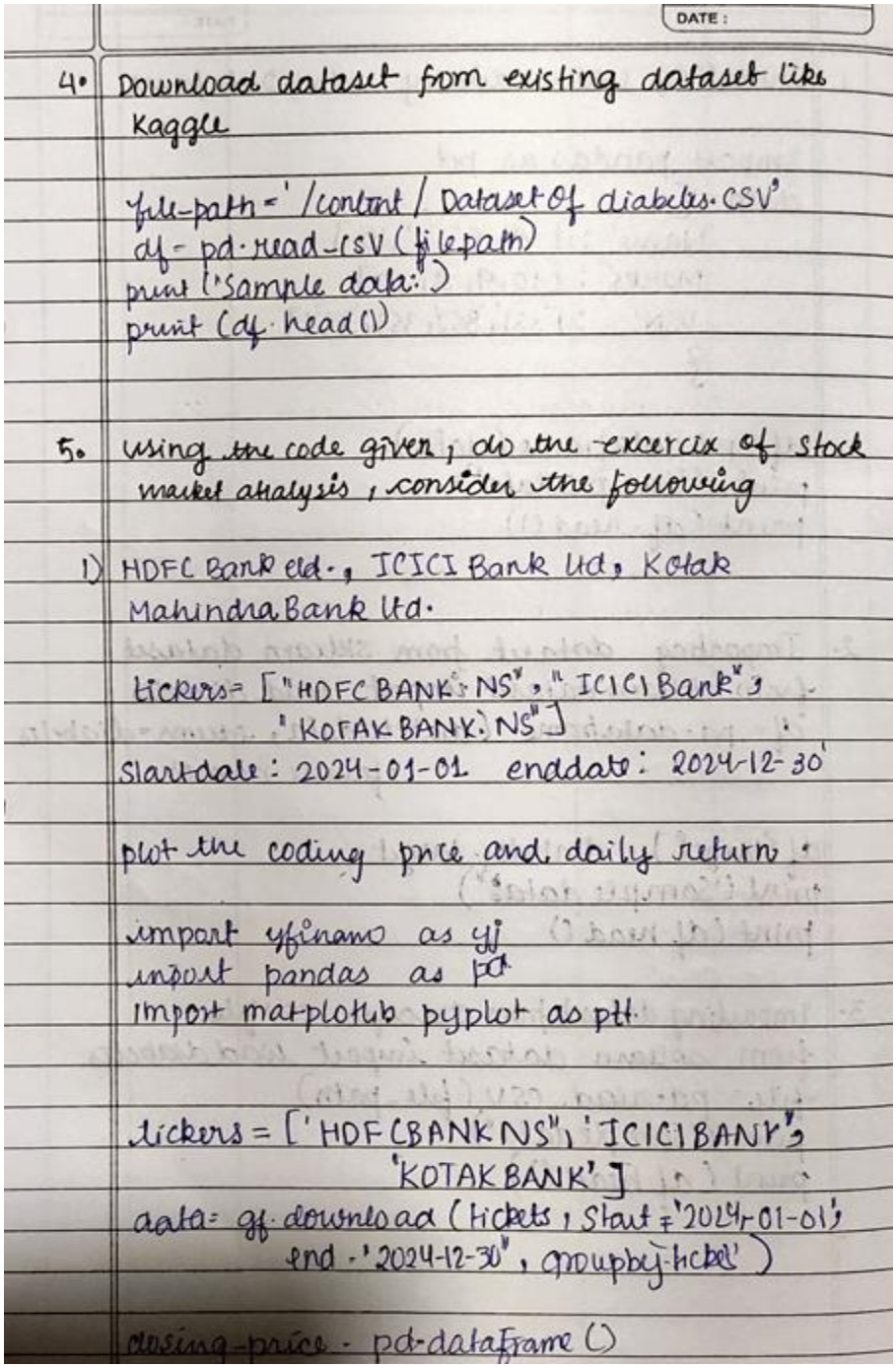
```
import pandas as pd
from sklearn.datasets import load_diabetes

# Part 1: Student data
student_data = {
    'USN': ['1A18CS001', '1A18CS002', '1A18CS003', '1A18CS004', '1A18CS005'],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Marks': [85, 78, 92, 88, 75]
}
student_df = pd.DataFrame(student_data)
print("Student Data:")
print(student_df)
print("\n" + "-"*50 + "\n")

# Part 2: Load diabetes dataset from sklearn
diabetes_data = load_diabetes()
diabetes_df = pd.DataFrame(diabetes_data.data, columns=diabetes_data.feature_names)
diabetes_df['target'] = diabetes_data.target
print("Scikit-learn Diabetes Dataset:")
print(diabetes_df.head())
print("\n" + "-"*50 + "\n")

# Part 3: Load sample sales data from CSV
```

```
try:
    sales_df = pd.read_csv('sample_sales_data.csv')
    print("Sample Sales Data:")
    print(sales_df.head())
except FileNotFoundError:
    print("sample_sales_data.csv not found.")
print("\n" + "-"*50 + "\n")

# Part 4: Load diabetes dataset from external CSV
try:
    diabetes_csv_df = pd.read_csv('Dataset of Diabetes .csv')
    print("Diabetes Dataset from CSV:")
    print(diabetes_csv_df.head())
except FileNotFoundError:
    print("Dataset of Diabetes .csv not found.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



DATE :

4. Download dataset from existing dataset like Kaggle

```
file_path = '/content/Dataset of diabetes.csv'
df = pd.read_csv(filepath)
print('Sample data:')
print(df.head())
```

5. Using the code given, do the excercise of stock market analysis, consider the following

1) HDFC Bank Ltd., ICICI Bank Ltd, Kotak Mahindra Bank Ltd.

```
tickers = ["HDFCBANK.NS", "ICICIBank",
           "KOTAKBANK.NS"]
Startdate: 2024-01-01  enddate: 2024-12-30
```

plot the coding price and daily return.

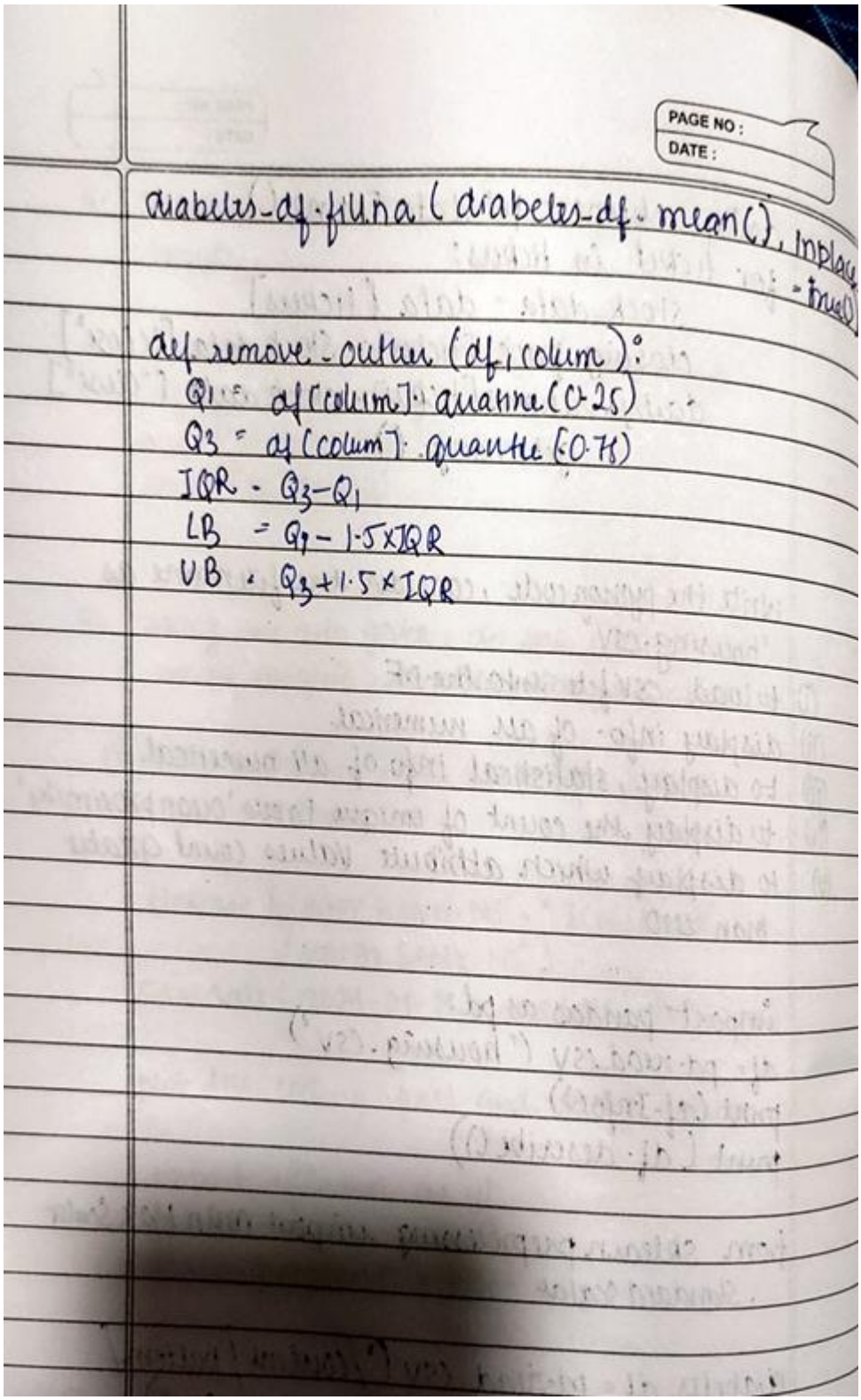
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ['HDFCBANK.NS', 'ICICIBANK',
           'KOTAKBANK']
data = yf.download(tickers, start='2024-01-01',
                   end='2024-12-30', groupby='ticker')

closing_price = pd.DataFrame()
```

```
daily_returns = pd.data_Frame()
for ticket in tickers:
    stock_data = data [Tickers]
    closing_price [Ticker] = Stock_data [:'Close"]
    daily_returns[Tickers] = stock data ["Close"]
        get_change ()
```

Write the pymon code, consider the filename as "housing.csv"

(i) to load csv file into the DF
(ii) display info. of all numerical
(iii) to display, statistical info of all numerical
(iv) to display the count of unique labels 'ocean proximity'
(v) to display which attribute values count greater than zero

```
import pandas as pd
df = pd.read_csv ('housing.csv")
print (df.Info())
print ( df.describe())

from sklearn.preprocessing import min MaxScaler
, Standard Scalar

diabetes_df = pd.read_CSV ("/content/pattern/
                            Dataset.csv")
adult_income_df = pd.read_CSV ("adult.qv")

for colum in the diabetes_df.select_dtypes(
    include = ['Obj']).columns:
  diabetes_df[column], errors' were'
```



PAGE NO :
DATE :

diabetes_df.fillna(diabetes_df.mean(), inplace=True)

def remove_outlier(df, colum):
    Q1 = df[colum].quantile(0.25)
    Q3 = df[colum].quantile(0.75)
    IQR = Q3 - Q1
    LB = Q1 - 1.5 * IQR
    UB = Q3 + 1.5 * IQR

Code:

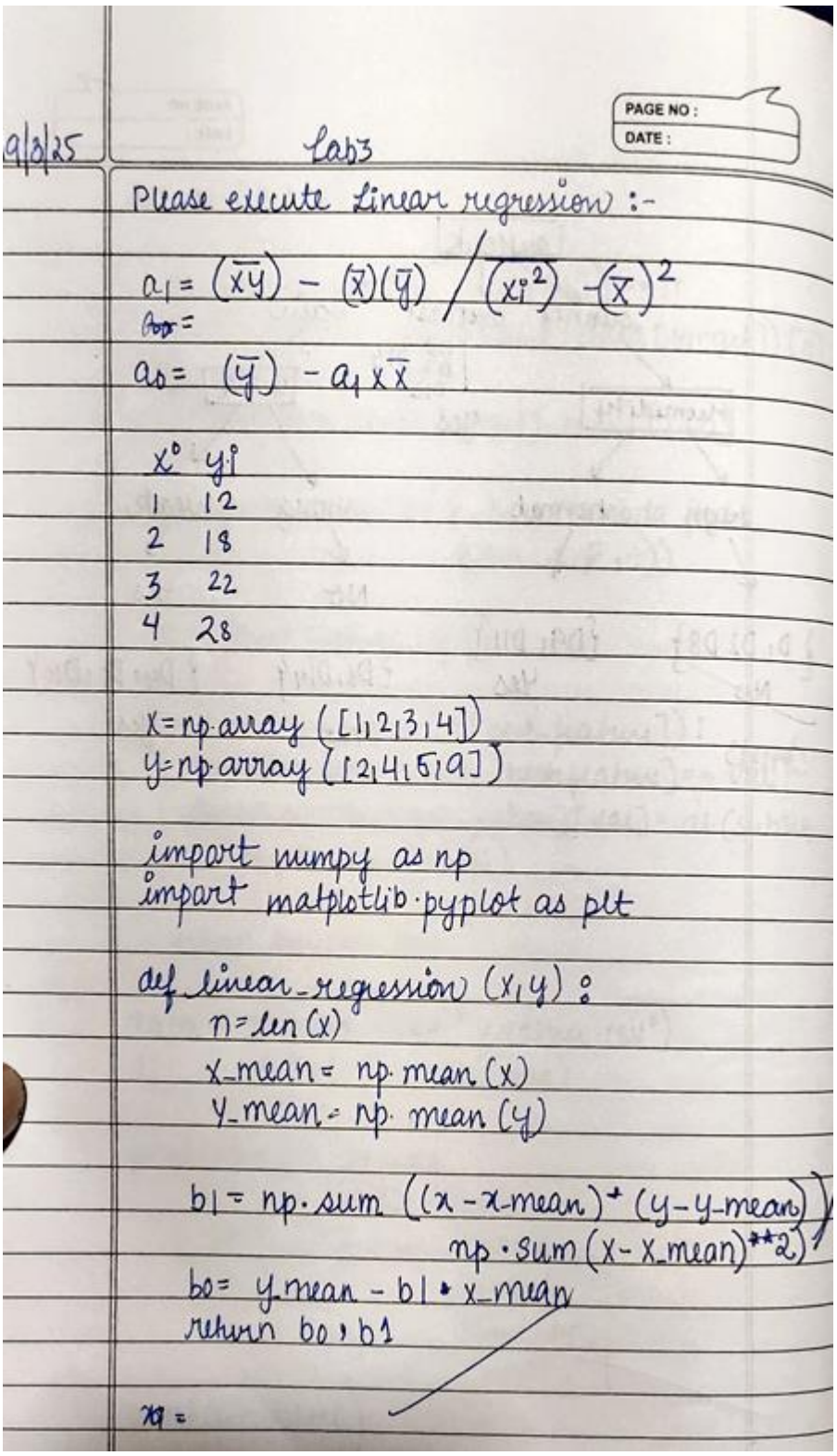
```
import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt


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


# Fetch historical data for the last 1 year

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


hdfc_data = data['HDFCBANK.NS']

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


ic_data = data['ICICIBANK.NS']

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


kb_data = data['KOTAKBANK.NS']

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


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

plt.subplot(2, 1, 1)

hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")

plt.subplot(2, 1, 2)

hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Returns", color='orange')
```

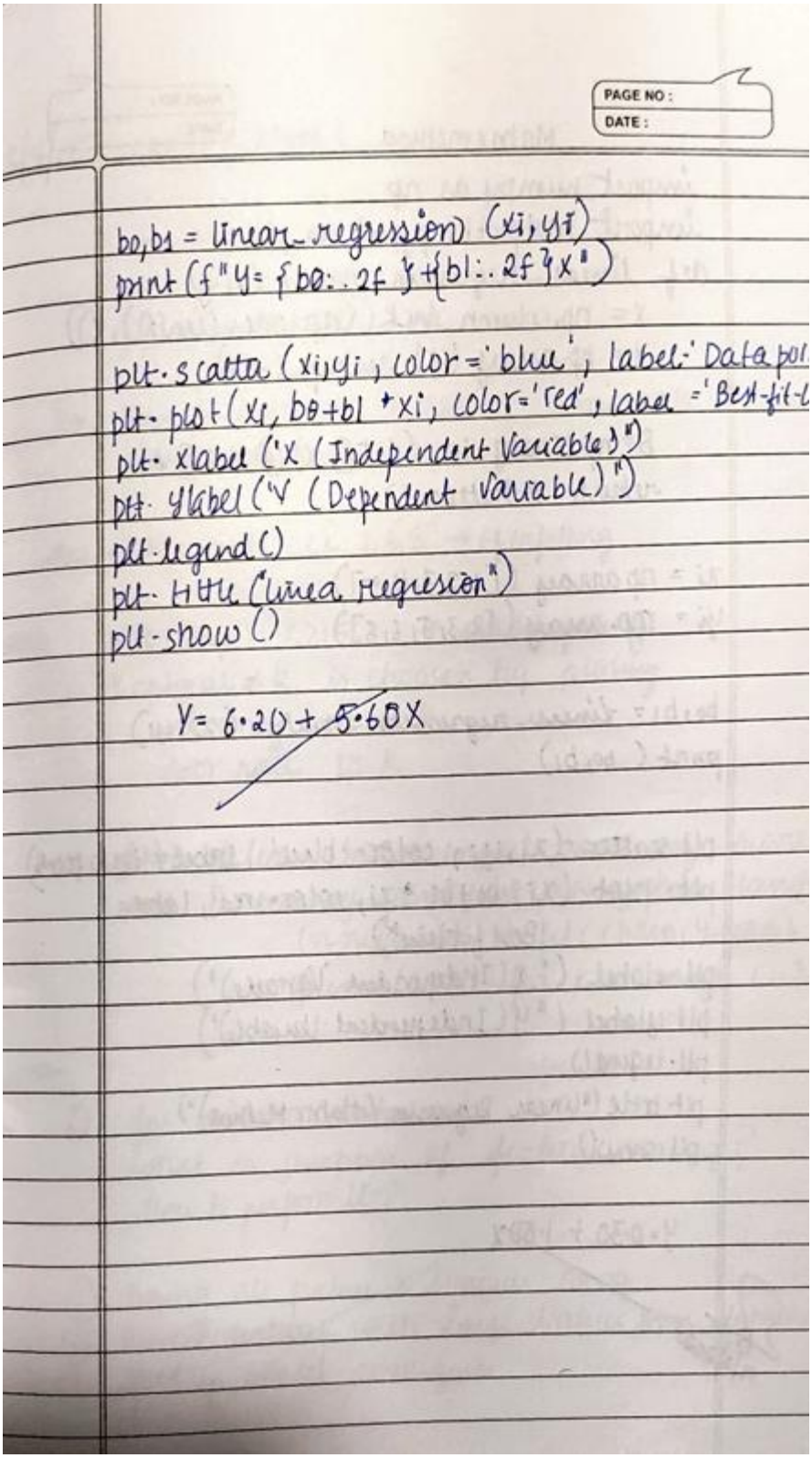
```
plt.tight_layout()
plt.show()


plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
ic_data['Close'].plot(title="ICICI Bank - Closing Price")
plt.subplot(2, 1, 2)
ic_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()


plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
kb_data['Close'].plot(title="Kotak Bank - Closing Price")
plt.subplot(2, 1, 2)
kb_data['Daily Return'].plot(title="Kotak Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot





```
b0,b1 = linear_regression(xi,yi)
print(f"y= {b0:.2f} + {b1:.2f} *x")

plt.scatter(xi,yi, color='blue', label='Data poi
plt.plot(xi, b0+b1*xi, color='red', label='Best-fit-l
plt.xlabel('X (Independent Variable)')
plt.ylabel('Y (Dependent Variable)')
plt.legend()
plt.title("Linear Regression")
plt.show()
```

$Y = 6.20 + 5.60X$

PAGE NO :
DATE :

Matrix method

```
import numpy as np
import matplotlib.pyplot as plt.
def linear_regression_matrix (x,y)
    X= np.column_stack((np.ones(len(A)),X))
    Y= np.array (y).reshape (-1,1)

    B= np.linalg.inv(X.T@ X) @ X.T@ Y
    return B.flatten ()

xi = nparray ([1,2,3,4,5])
yi = np.array ([2,3,5,6,8])

bo,b1 = linear_regression_matrix (xi,yi)
print ( bo,b1)

plt.scatter (xi,yi, color='blue', label='Data points')
plt.plot (xi, bo+b1 *xi, color='red', label=
          'Best fit line')
plt.xlabel ('X (Independent Variable)')
plt.ylabel ("Y (Independent Variable)")
plt.legend()
plt.title ("Linear Regression (Matrix Method)")
plt.show()
```

$y = 0.30 + 1.50x$

Code:

```
import matplotlib.pyplot as plt


x = list(map(float, input("Enter X values separated by space: ").split()))
y = list(map(float, input("Enter Y values separated by space: ").split()))
n = len(x)
sum_x = sum(x)
sum_y = sum(y)
sum_x2 = sum(xi**2 for xi in x)
sum_xy = sum(x[i] * y[i] for i in range(n))
a1 = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x**2)
a0 = (sum_y - a1 * sum_x) / n
print("\nLinear Regression Equation:")
print(f"y = {a0:.2f} + {a1:.2f}x")
x_test = float(input("\nEnter a value of x to predict y: "))
y_pred = a0 + a1 * x_test
print(f"Predicted y: {y_pred:.2f}")
plt.scatter(x, y, color='blue', label='Data Points')
x_line = [min(x) - 1, max(x) + 1]
y_line = [a0 + a1 * x_val for x_val in x_line]
plt.plot(x_line, y_line, color='red', label='Regression Line')
plt.scatter(x_test, y_pred, color='green', label=f'Prediction: ({x_test}, {y_pred:.2f})')
plt.xlabel('X values')
plt.ylabel('Y values')
```

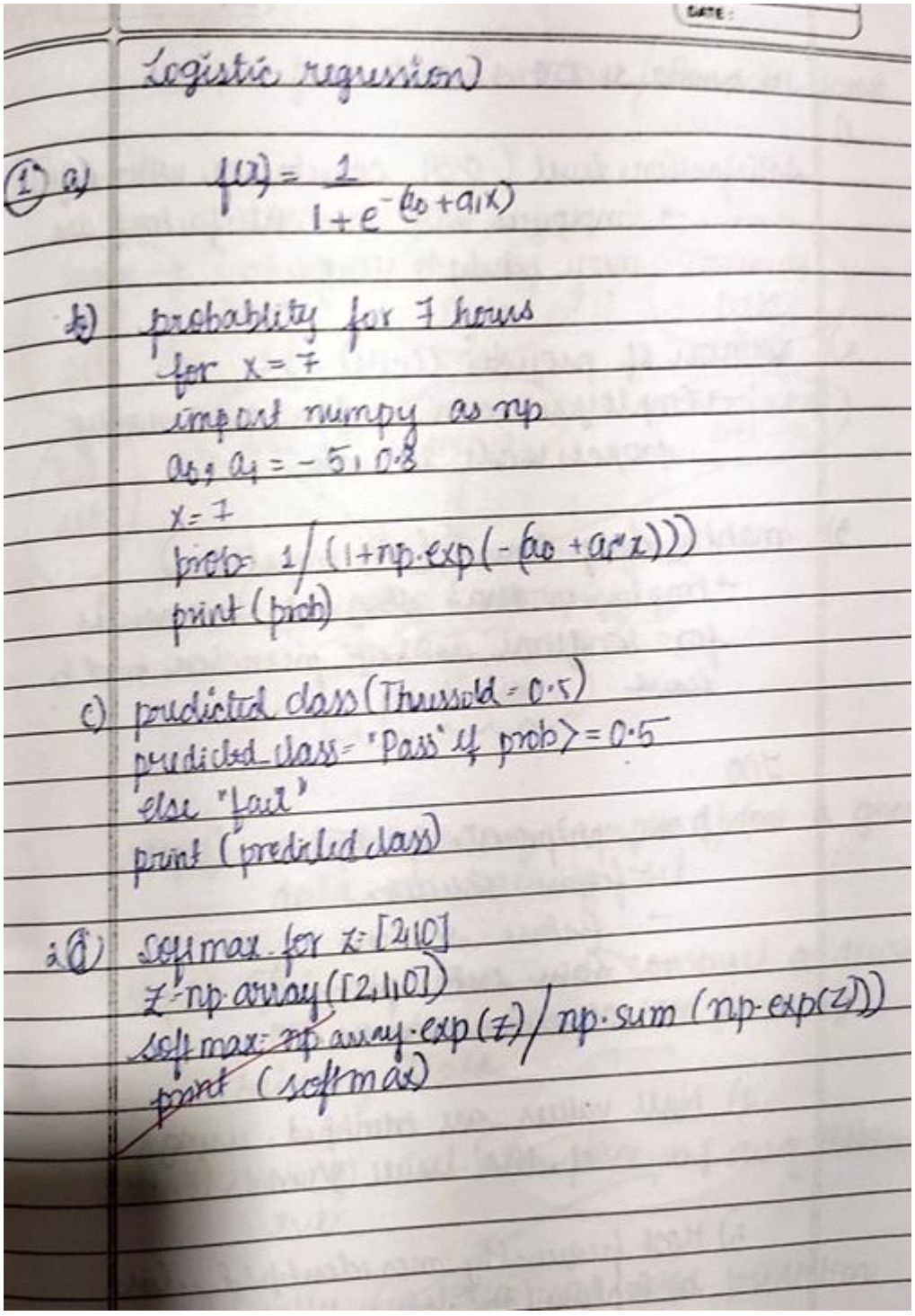
```
plt.title('Linear Regression Visualization')
plt.legend()
plt.grid(True)
plt.show()

import numpy as np
import matplotlib.pyplot as plt

x = np.array(list(map(float, input("Enter X values separated by space: ").split())))
y = np.array(list(map(float, input("Enter Y values separated by space: ").split())))
X = np.c_[np.ones(len(x)), x]
beta = np.linalg.inv(X.T @ X) @ X.T @ y
print("\nLinear Regression Equation (Matrix Form):")
print(f"y = {beta[0]:.2f} + {beta[1]:.2f}x")
x_test = float(input("\nEnter a value of x to predict y: "))
y_pred = beta[0] + beta[1] * x_test
print(f"Predicted y: {y_pred:.2f}")
plt.scatter(x, y, color='blue', label='Data Points')
x_line = np.linspace(min(x) - 1, max(x) + 1, 100)
y_line = beta[0] + beta[1] * x_line
plt.plot(x_line, y_line, color='red', label='Regression Line')
plt.scatter(x_test, y_pred, color='green', label=f'Prediction: ({x_test}, {y_pred:.2f})')
plt.xlabel('X values')
plt.ylabel('Y values')
```

```
plt.title('Linear Regression (Matrix Form)')

plt.legend()

plt.grid(True)

plt.show()
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot



HR_comma_sv.csv

1)

Satisfaction level (-0.39 correlation with left)
→ Employees with low satisfactory are more likely to leave

2) Number of projects (0.14)
→ Employees with too few or too many projects tends to leave

3) monthly avg hours (0.13 correlated)
→ Employees that stay at companies for long time without promotion tend to leave

Zoo

1) Data preprocessing steps
→ feature selection
→ feature scaling
→ train test split
→ class encoding

2) Null values are identified using:
zoo.data.isnull().sum()

3) Most frequently mes identified class
1) Reptiles (3) & Amphibians (5)
2) ~~Invertebrates~~ (7) & Bugs (6)
3) fish (4) & amphibians (5)

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 LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load dataset
df = pd.read_csv("HR_comma_sep.csv")

# Convert categorical variables to numerical
label_enc = LabelEncoder()
```

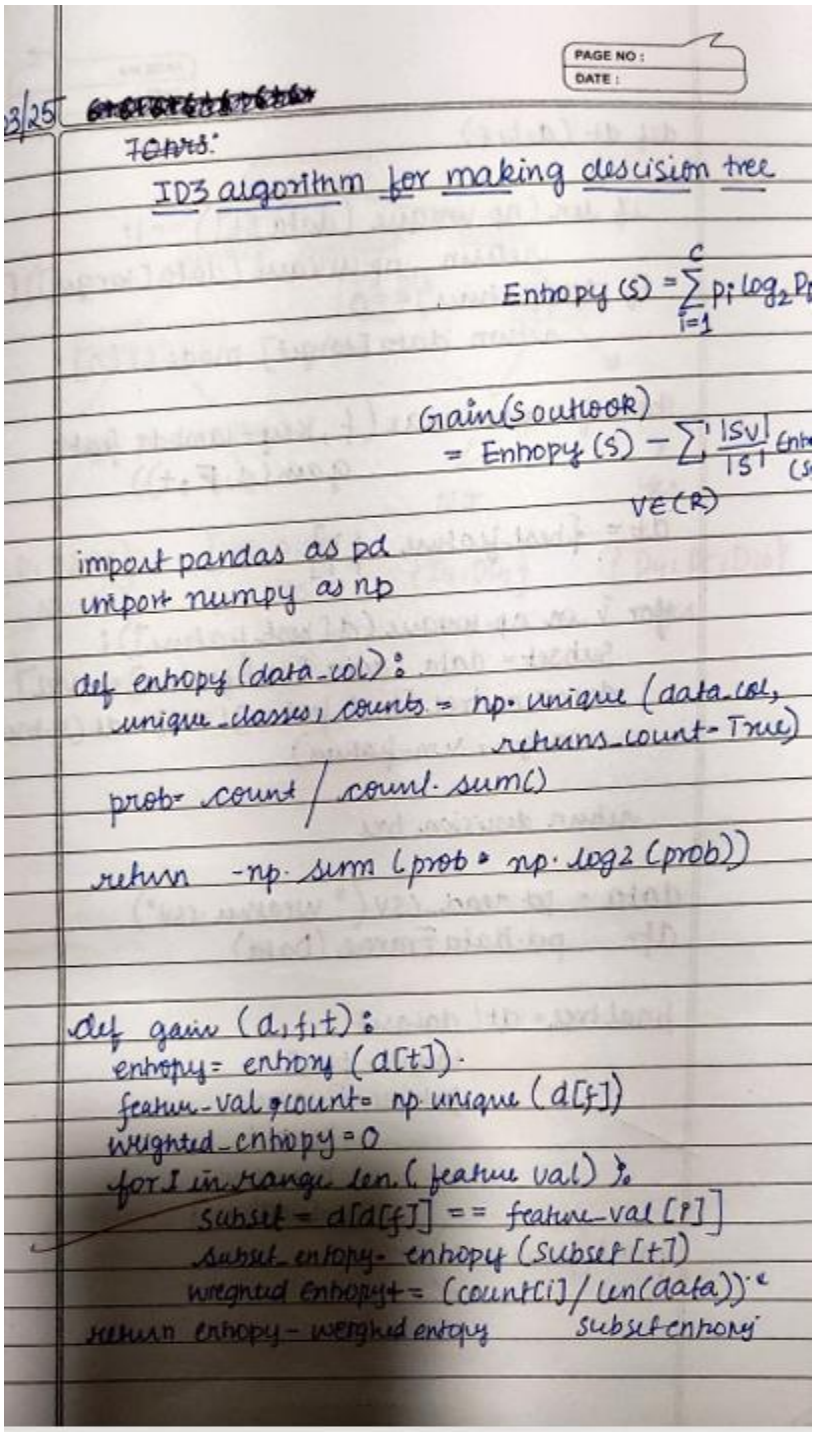
```
df["salary"] = label_enc.fit_transform(df["salary"])
df["Department"] = label_enc.fit_transform(df["Department"])

# Step 1: Exploratory Data Analysis (EDA)
correlation = df.corr()["left"].sort_values(ascending=False)
print("\nFeature Correlation with Employee Retention:\n", correlation)

# Step 2: Impact of Salary on Retention
plt.figure(figsize=(6,4))
sns.barplot(x="salary", y="left", data=df, ci=None)
plt.xlabel("Salary Level (Encoded)")
plt.ylabel("Retention Rate")
plt.title("Impact of Salary on Employee Retention")
plt.show()

# Step 3: Correlation between Department and Retention
plt.figure(figsize=(8,4))
sns.barplot(x="Department", y="left", data=df, ci=None)
plt.xlabel("Department (Encoded)")
plt.ylabel("Retention Rate")
plt.title("Department vs Employee Retention")
plt.show()

# Step 4: Logistic Regression Model
```

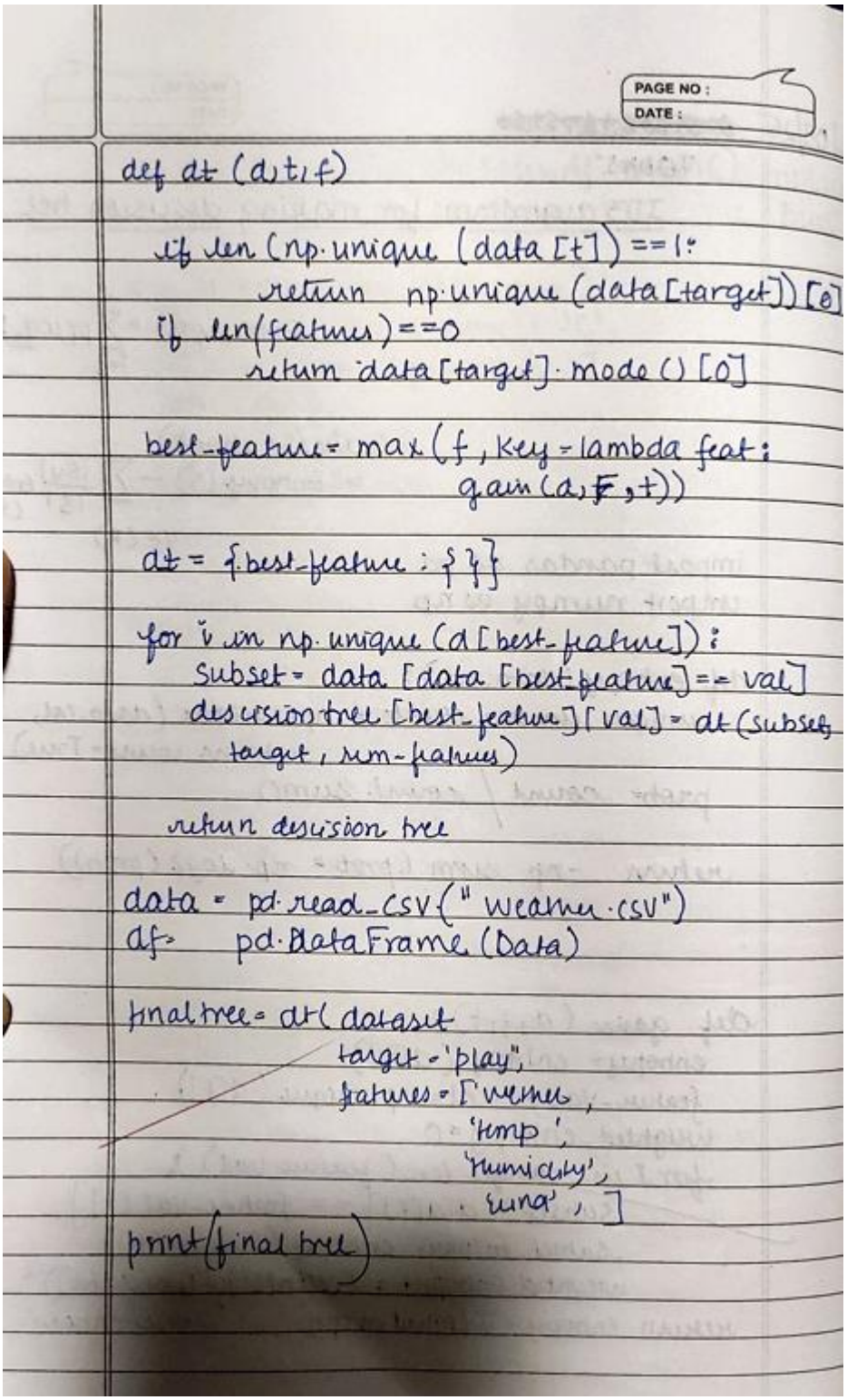
```
features = ["satisfaction_level", "last_evaluation", "number_project", "average_montly_hours",
"time_spend_company", "salary"]

X = df[features]

y = df["left"]


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


model = LogisticRegression()

model.fit(X_train, y_train)


# Step 5: Model Accuracy

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"\nModel Accuracy: {accuracy:.2f}")


# Display Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred)

print("\nConfusion Matrix:\n", conf_matrix)


# Display Classification Report

print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Program 5

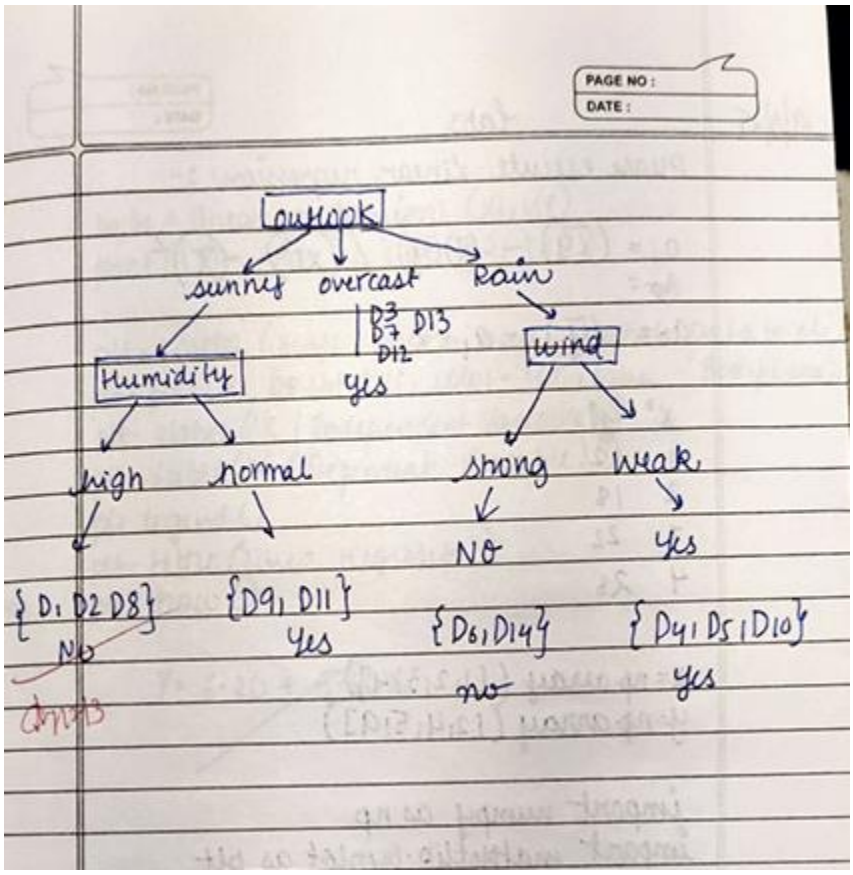
Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

Screenshot





```
def dt(d,t,f)

  if len(np.unique(data[t]))==1:
      return np.unique(data[target])[0]
  if len(features)==0
      return data[target].mode()[0]

  best-feature = max(f, key = lambda feat:
                     gain(d,F,t))

  dt = {best-feature : {}}

  for i in np.unique(d[best-feature]):
     subset = data[data[best-feature]== val]
     decision tree[best-feature][val] = dt(subset
       target, num-features)

  return decision tree

data = pd.read_csv("weather.csv")
df= pd.DataFrame(Data)

final tree = dt( dataset
                 target = 'play',
                 features = ['weather,
                             'temp',
                             'humidity',
                             'wind', ]

print(final tree)
```


PAGE NO :
DATE :
outlook
sunny
overcast
Rain
D3 D7 D12 D13
yes
Humidity
wind
high
normal
strong
weak
NO
yes
{D1 D2 D8}
No
{D9, D11}
yes
{D6, D14}
no
{D4, D5, D10}
yes

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 LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load dataset
df = pd.read_csv("HR_comma_sep.csv")

# Convert categorical variables to numerical
label_enc = LabelEncoder()
df["salary"] = label_enc.fit_transform(df["salary"])
df["Department"] = label_enc.fit_transform(df["Department"])

# Step 1: Exploratory Data Analysis (EDA)
correlation = df.corr()["left"].sort_values(ascending=False)
print("\nFeature Correlation with Employee Retention:\n", correlation)

# Step 2: Impact of Salary on Retention
plt.figure(figsize=(6,4))
```

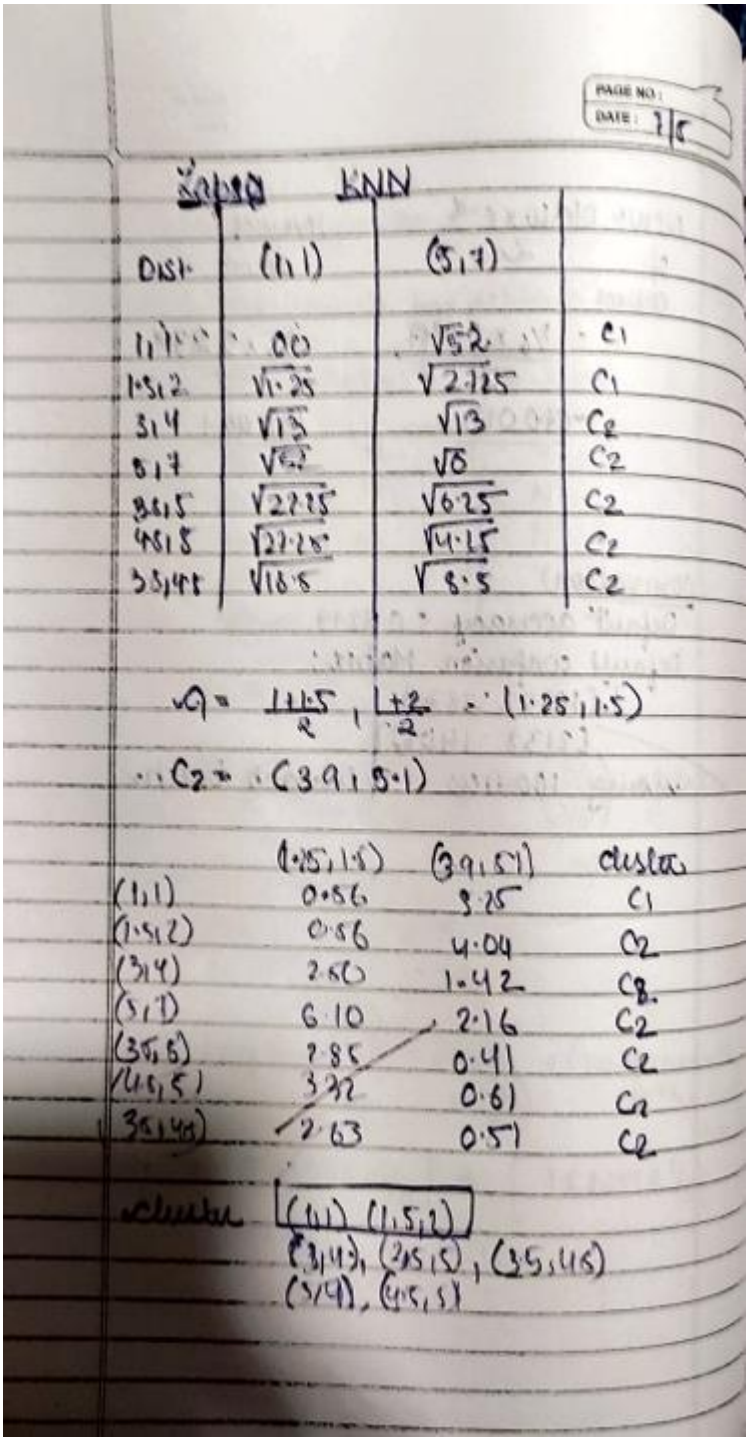
```
sns.barplot(x="salary", y="left", data=df, ci=None)

plt.xlabel("Salary Level (Encoded)")

plt.ylabel("Retention Rate")

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

plt.show()


# Step 3: Correlation between Department and Retention

plt.figure(figsize=(8,4))

sns.barplot(x="Department", y="left", data=df, ci=None)

plt.xlabel("Department (Encoded)")

plt.ylabel("Retention Rate")

plt.title("Department vs Employee Retention")

plt.show()


# Step 4: Logistic Regression Model

features = ["satisfaction_level", "last_evaluation", "number_project", "average_montly_hours",
"time_spend_company", "salary"]

X = df[features]

y = df["left"]


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


model = LogisticRegression()

model.fit(X_train, y_train)
```

```
# Step 5: Model Accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy: {accuracy:.2f}")

# Display Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)

# Display Classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Program 6

Build KNN Classification model for a given dataset

Screenshot



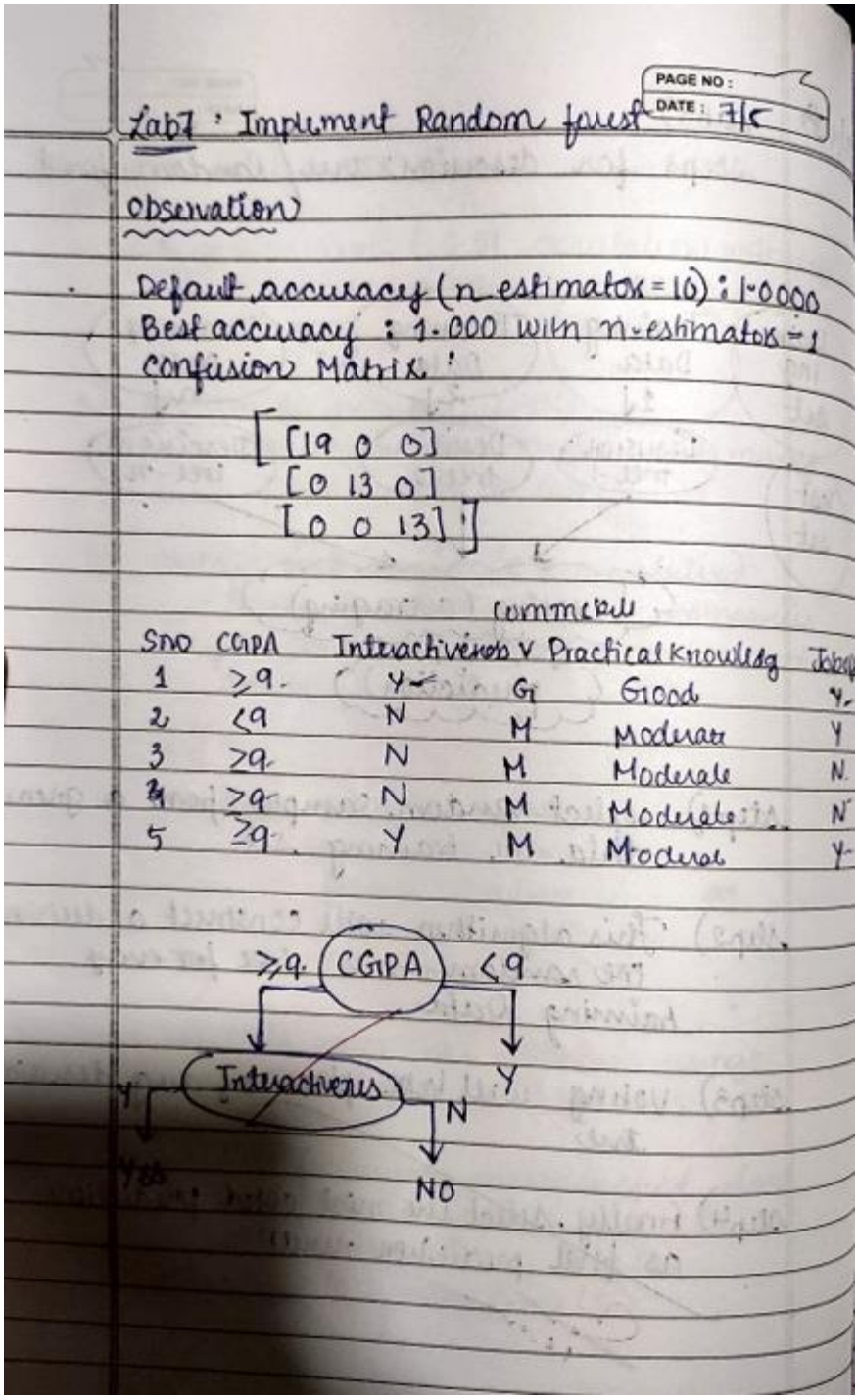
Code:

```
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.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

file_path = input("Enter the path to your CSV file: ")
df = pd.read_csv(file_path)

# Automatically select the target column (assuming it's the last column)
target_column = df.columns[-1]

# Splitting features (X) and target variable (y)
X = df.iloc[:, :-1] # All columns except last (features)
y = df.iloc[:, -1]  # Last column (target)

# Encode categorical target variable if necessary
if y.dtype == 'object':
    label_encoder = LabelEncoder()
    y = label_encoder.fit_transform(y)

# Convert categorical features to numerical if any
X = pd.get_dummies(X, drop_first=True)

# Split the dataset (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Determine optimal k (square root heuristic)
k = int(np.sqrt(len(y_train)))
if k % 2 == 0:
    k += 1

# Train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

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

```
# Display accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy Score: {accuracy:.2f}")

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)

# Display classification report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", class_report)

# Plot Confusion Matrix
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y),
yticklabels=np.unique(y))
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

Program 7

Implement Random Forest ensemble method on a given dataset



Lab7 : Implement Random forest

PAGE NO :
DATE : 7/5

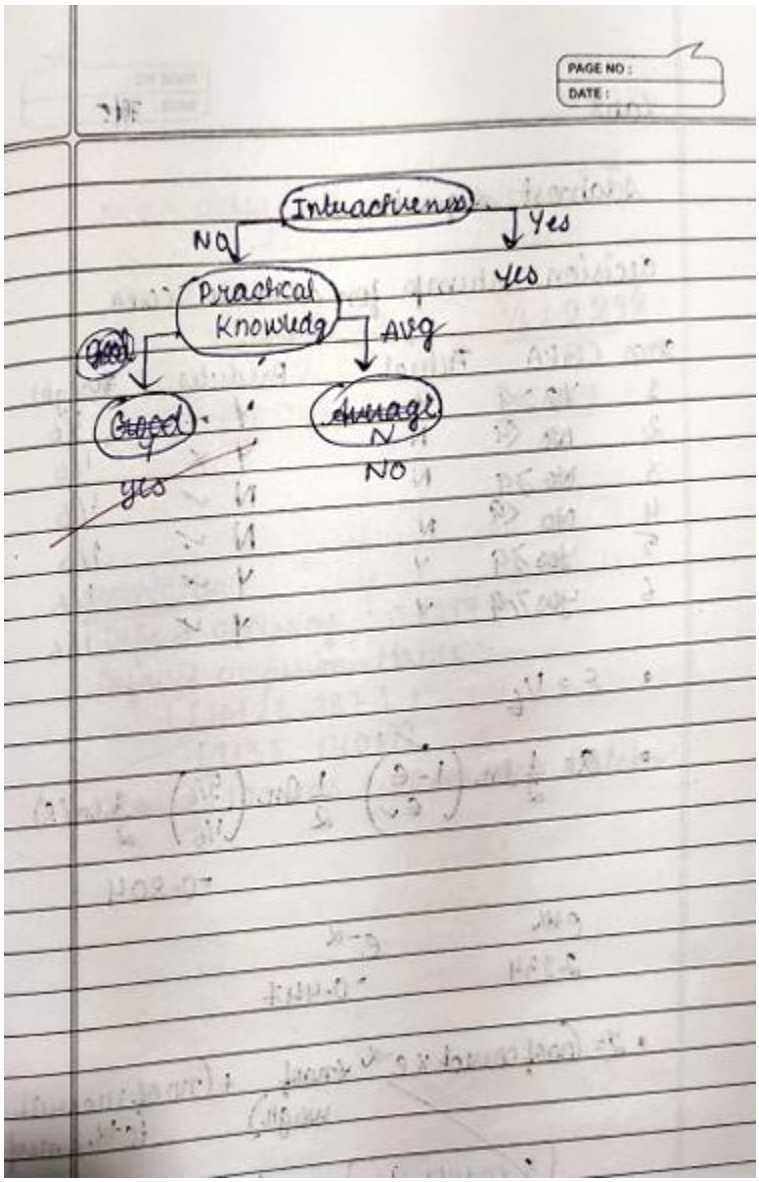
Observation)

Default accuracy (n_estimators=10) : 1.0000
Best accuracy : 1.000 with n_estimators=1
Confusion Matrix :

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

SNO	CGPA	Interactiveness	Practical Knowledge	Job
1	≥9	Y	Good	Y
2	<9	N	Moderate	Y
3	≥9	N	Moderate	N
4	≥9	N	Moderate	N
5	≥9	Y	Moderate	Y

Screenshot


PAGE NO :
DATE :
Interactiveness
NO
Yes
yes
Practical Knowledg
AVG
yes
NO

Code:

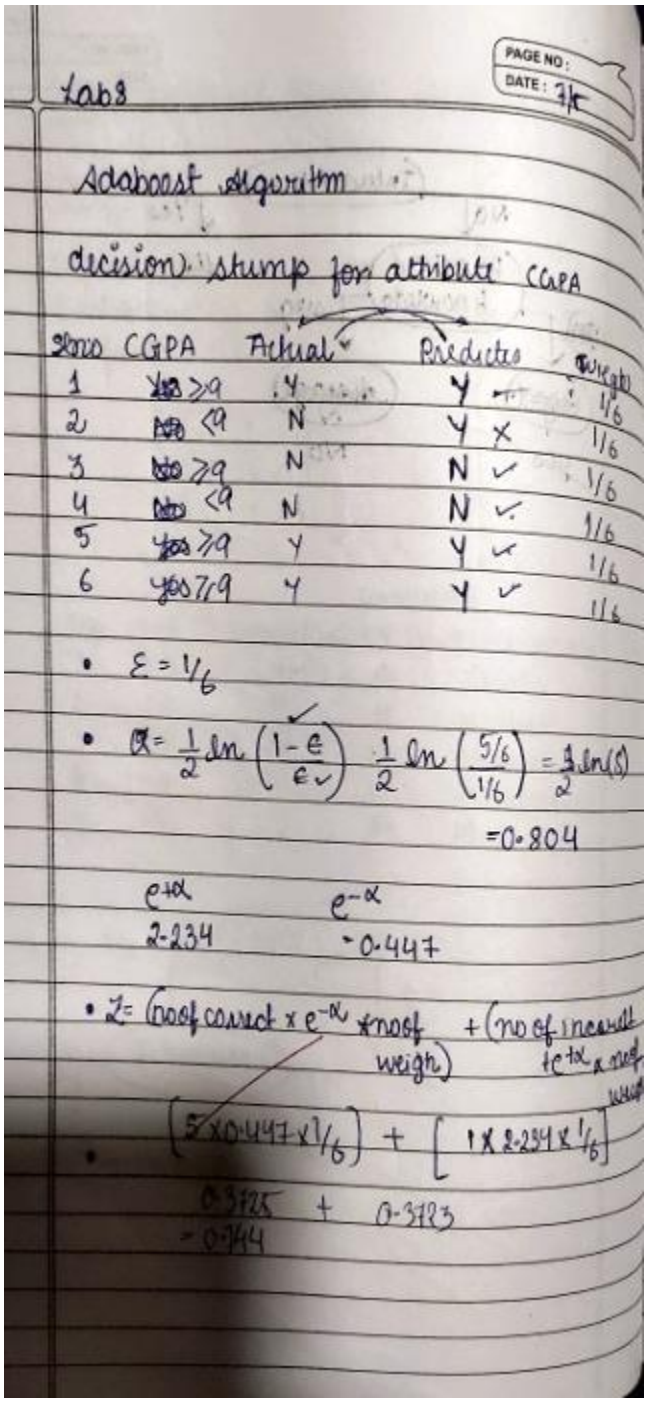
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("iris.csv")

# Features and target
X = df.drop(columns=["species"])
y = df["species"]

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 1. Train with default n_estimators = 10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)

print(f"Accuracy with default (10 trees): {default_score:.4f}")
```

```
# 2. Fine-tune n_estimators
scores = []
tree_range = range(1, 51)

for n in tree_range:
    model = RandomForestClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    scores.append(acc)
# Plotting results
plt.figure(figsize=(10,5))
plt.plot(tree_range, scores, marker='o')
plt.title("Random Forest Accuracy vs Number of Trees")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
# Best result
best_score = max(scores)
best_n = tree_range[scores.index(best_score)]

print(f"Best Accuracy = {best_score:.4f} with n_estimators = {best_n}")
```

Program 8

Implement Boosting ensemble method on a given dataset

Screenshot



PAGE NO :
DATE :

New wt of correct $= \frac{\text{old wt} \times e^{-\alpha}}{Z}$

$= \frac{1/6 \times 0.447}{Z}$

$= 0.10013$

New wt of incorr

$\frac{1/6 \times 2.234}{Z}$

0.5004

Observation)

Default accuracy = 0.8277

Default confusion Matrix :

[[10722 387]
[2138 1406]]

using 100 trees (10 in each batch)

Code:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('income.csv')

# Explore the dataset
print(data.head())
print("\nDataset info:")
print(data.info())
print("\nClass distribution:")
print(data['income_level'].value_counts())

# Split into features and target
X = data.drop('income_level', axis=1)
y = data['income_level']

# Encode categorical features (one-hot encoding)
```

```
X = pd.get_dummies(X)


# Check for missing values
if X.isnull().sum().sum() > 0:
    print("Missing values found. Filling missing values with column mean.")
    X = X.fillna(X.mean())


# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)


# Define base estimator
base_estimator = DecisionTreeClassifier(max_depth=1)  # Stump tree for AdaBoost


# Initial AdaBoost model with 10 estimators
ada_model = AdaBoostClassifier(
    estimator=base_estimator,
    n_estimators=10,
    random_state=42
)


# Train the model
ada_model.fit(X_train, y_train)
```

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

# Evaluate initial model
print("\nInitial Model with 10 estimators:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Fine-tune the number of trees
n_estimators_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200]
train_scores = []
test_scores = []

for n in n_estimators_range:
    model = AdaBoostClassifier(
        estimator=base_estimator,
        n_estimators=n,
        random_state=42
    )
    model.fit(X_train, y_train)

    # Training accuracy
    train_pred = model.predict(X_train)
```

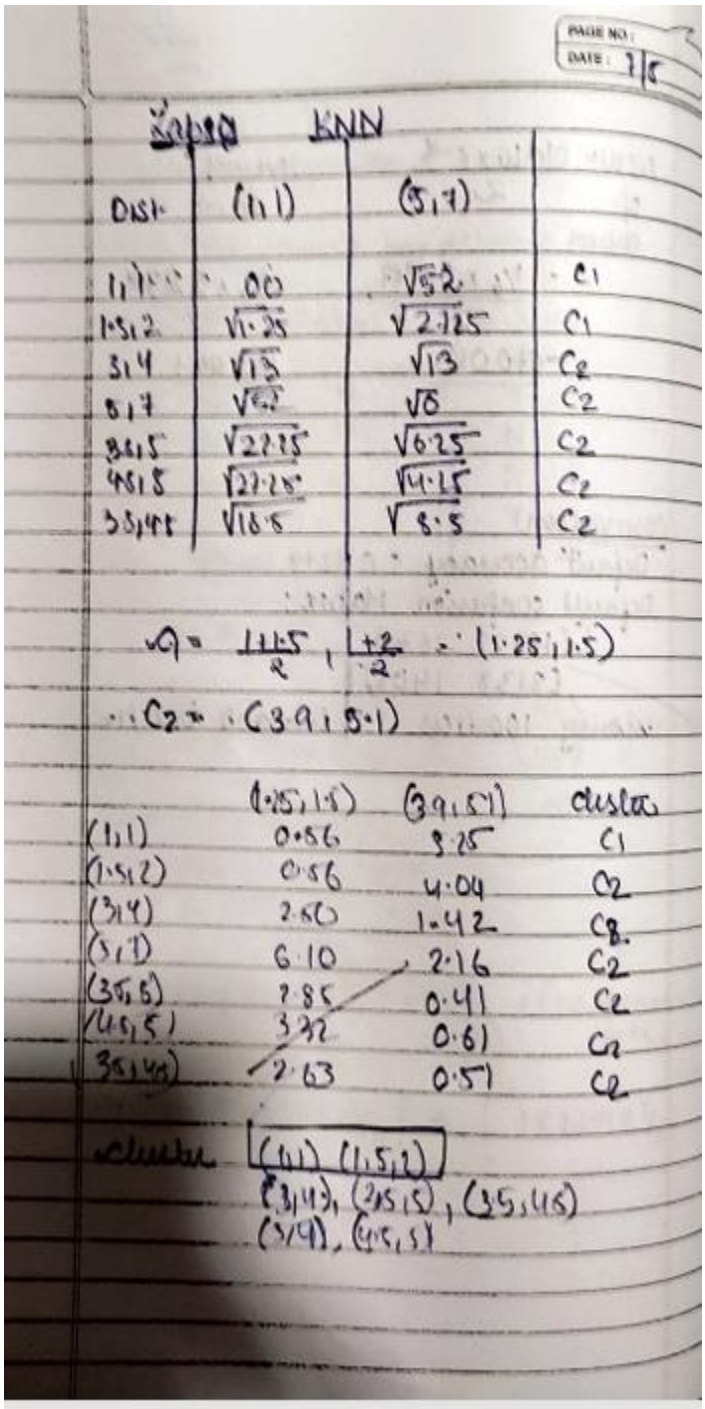
```
    train_acc = accuracy_score(y_train, train_pred)
    train_scores.append(train_acc)

    # Test accuracy
    test_pred = model.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    test_scores.append(test_acc)

    print(f"n_estimators: {n}, Train Accuracy: {train_acc:.4f}, Test Accuracy: {test_acc:.4f}")

# Find the best number of estimators
best_n = n_estimators_range[np.argmax(test_scores)]
best_score = max(test_scores)

print(f"\nBest performance: n_estimators={best_n} with test accuracy of {best_score:.4f}")

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, train_scores, label='Train Accuracy', marker='o')
plt.plot(n_estimators_range, test_scores, label='Test Accuracy', marker='o')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('AdaBoost Performance vs Number of Estimators')
plt.axvline(x=best_n, color='r', linestyle='--', label=f'Best n_estimators={best_n}')
```

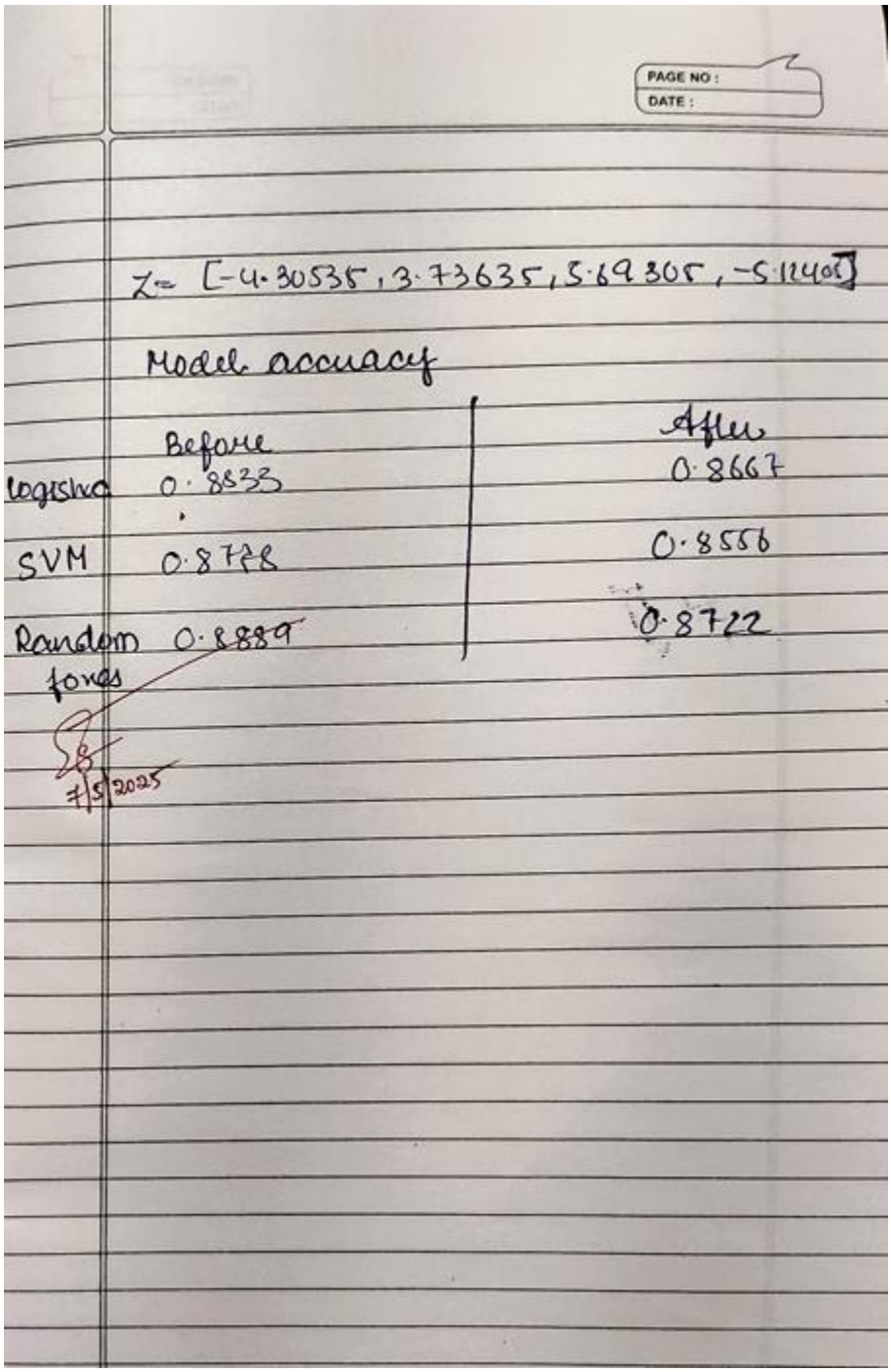
```
plt.legend()
plt.grid()
plt.show()

# Train final model with best number of estimators
final_model = AdaBoostClassifier(
    estimator=base_estimator,
    n_estimators=best_n,
    random_state=42
)
final_model.fit(X_train, y_train)

# Evaluate final model
final_pred = final_model.predict(X_test)
print("\nFinal Model Performance:")
print("Accuracy:", accuracy_score(y_test, final_pred))
print("\nClassification Report:")
print(classification_report(y_test, final_pred))

print("\nFeature Importances:")
feature_importances = pd.Series(final_model.feature_importances_, index=X.columns)
print(feature_importances.sort_values(ascending=False))
```

Program 9

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

Screenshot




PAGE NO :
DATE :

Z = [-4.30535, 3.73635, 5.69305, -5.12405]

Model accuracy

	Before	After
Logistic	0.8833	0.8667
SVM	0.8728	0.8556
Random forest	0.8889	0.8722

7/5/2025

Code:

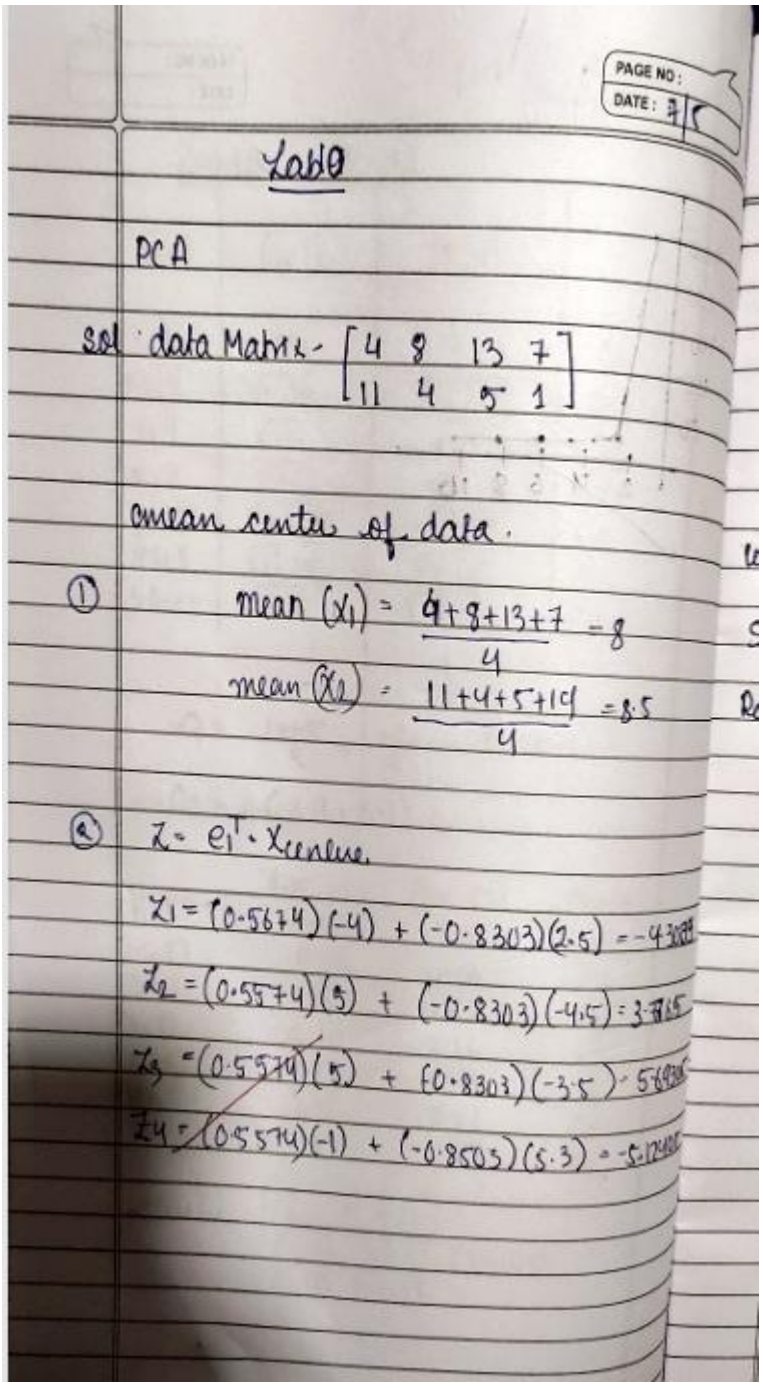
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Load your iris dataset
df = pd.read_csv("iris.csv")

# Use only Petal Length and Petal Width
X = df[["petal_length", "petal_width"]]

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Elbow Method to determine optimal k
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_)

# Plot Elbow Curve
```

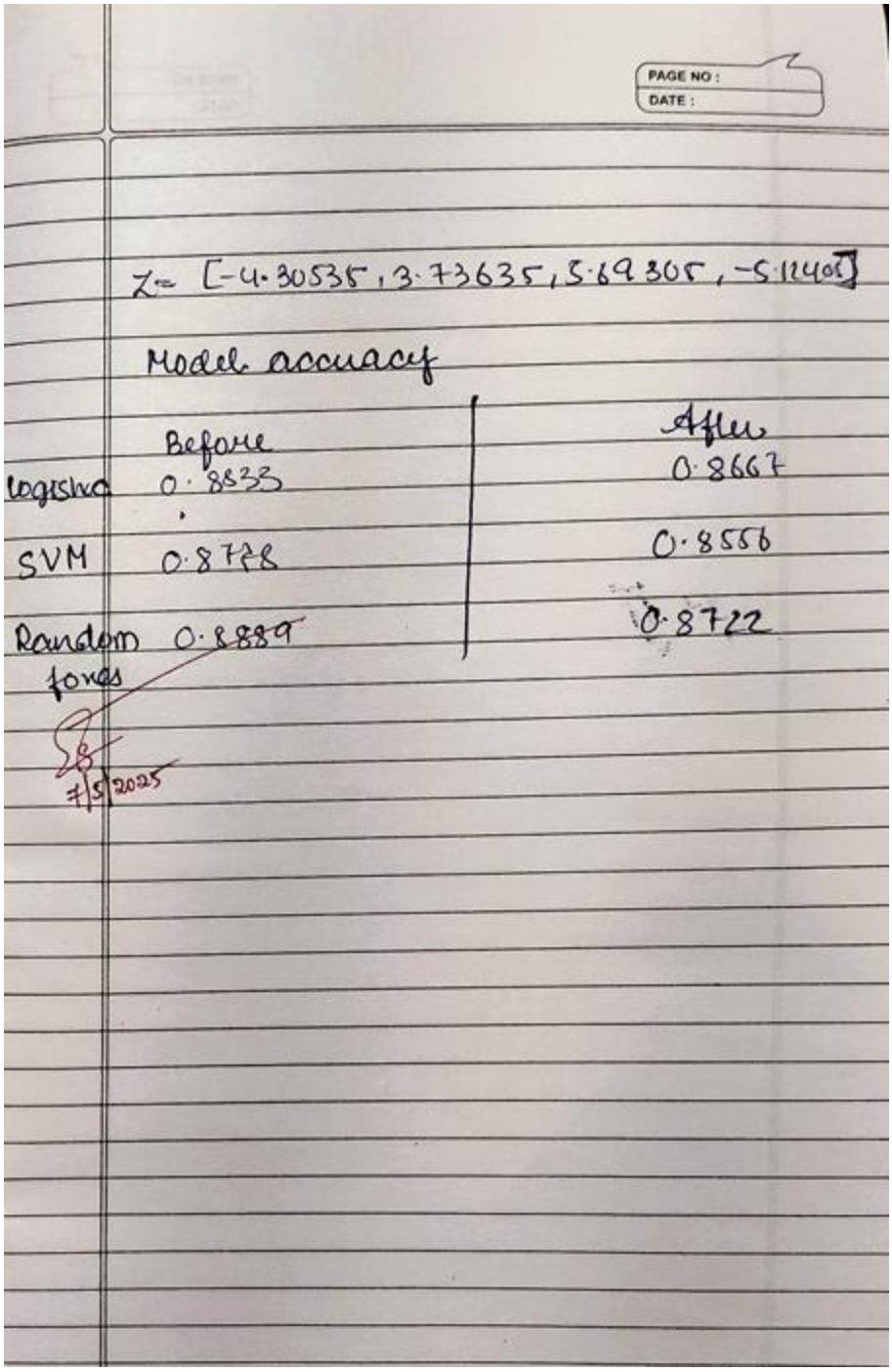
```
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.show()

# Final KMeans with optimal k (e.g., 3 from elbow)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Visualize Clusters
plt.figure(figsize=(8, 5))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis', s=50)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color='red', marker='X', label='Centroids')
plt.xlabel("Petal Length (scaled)")
plt.ylabel("Petal Width (scaled)")
plt.title(f"K-Means Clustering (k={optimal_k}) on Iris Petal Features")
plt.legend()
plt.grid(True)
plt.show()
```

Program 10

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot




PAGE NO :
DATE :

Z = [-4.30535, 3.73635, 5.69305, -5.12405]

Model accuracy

	Before	After
Logistic	0.8833	0.8667
SVM	0.8728	0.8556
Random forest	0.8889	0.8722

7/5/2025

Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline

# Load dataset
data = pd.read_csv('heart.csv')

# Separate features and target
X = data.drop('HeartDisease', axis=1)
y = data['HeartDisease']

# Identify categorical and numerical columns
cat_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
num_cols = [col for col in X.columns if col not in cat_cols]

# Label encode binary categorical columns with two categories (e.g. Sex, ExerciseAngina)
```

```
label_enc_cols = ['Sex', 'ExerciseAngina']
le = LabelEncoder()
for col in label_enc_cols:
    X[col] = le.fit_transform(X[col])

# For other categorical columns with more than two categories, apply OneHotEncoding
onehot_cols = list(set(cat_cols) - set(label_enc_cols))

# Preprocessing pipeline: OneHotEncoding + scaling numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(drop='first'), onehot_cols),
        ('scaler', StandardScaler(), num_cols)
    ],
    remainder='passthrough'  # To keep label encoded columns as is
)

# Split data into train/test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)

# Helper function to train and evaluate a model
def train_evaluate_model(model, X_train, X_test, y_train, y_test):
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
```

```
                    ('classifier', model)])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    return acc, pipeline

# Train and evaluate SVM
svm = SVC(random_state=42)
svm_acc, svm_pipeline = train_evaluate_model(svm, X_train, X_test, y_train, y_test)

# Train and evaluate Logistic Regression
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg_acc, logreg_pipeline = train_evaluate_model(logreg, X_train, X_test, y_train, y_test)

# Train and evaluate Random Forest
rf = RandomForestClassifier(random_state=42)
rf_acc, rf_pipeline = train_evaluate_model(rf, X_train, X_test, y_train, y_test)

print(f"Accuracy Scores without PCA:\nSVM: {svm_acc:.4f}\nLogistic Regression:
{logreg_acc:.4f}\nRandom Forest: {rf_acc:.4f}")

# Now apply PCA for dimensionality reduction after preprocessing (scaling + encoding)
# We modify the pipeline to include PCA before classification

def train_evaluate_model_pca(model, X_train, X_test, y_train, y_test, n_components):
```

```
    pca = PCA(n_components=n_components, random_state=42)
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('pca', pca),
        ('classifier', model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    return acc, pipeline

# Choose number of components to keep 95% variance or fixed number (e.g. 5)
# Here let's pick 5 components arbitrarily
n_components = 5

svm_pca_acc, _ = train_evaluate_model_pca(svm, X_train, X_test, y_train, y_test, n_components)
logreg_pca_acc, _ = train_evaluate_model_pca(logreg, X_train, X_test, y_train, y_test,
n_components)
rf_pca_acc, _ = train_evaluate_model_pca(rf, X_train, X_test, y_train, y_test, n_components)

print(f"\nAccuracy Scores with PCA (n_components={n_components}):")
print(f"SVM: {svm_pca_acc:.4f}")
print(f"Logistic Regression: {logreg_pca_acc:.4f}")
print(f"Random Forest: {rf_pca_acc:.4f}")
```