VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Sneha N Shastri (1BM22CS283)

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 **Sneha N Shastri (1BM22CS283)** 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.

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

Index

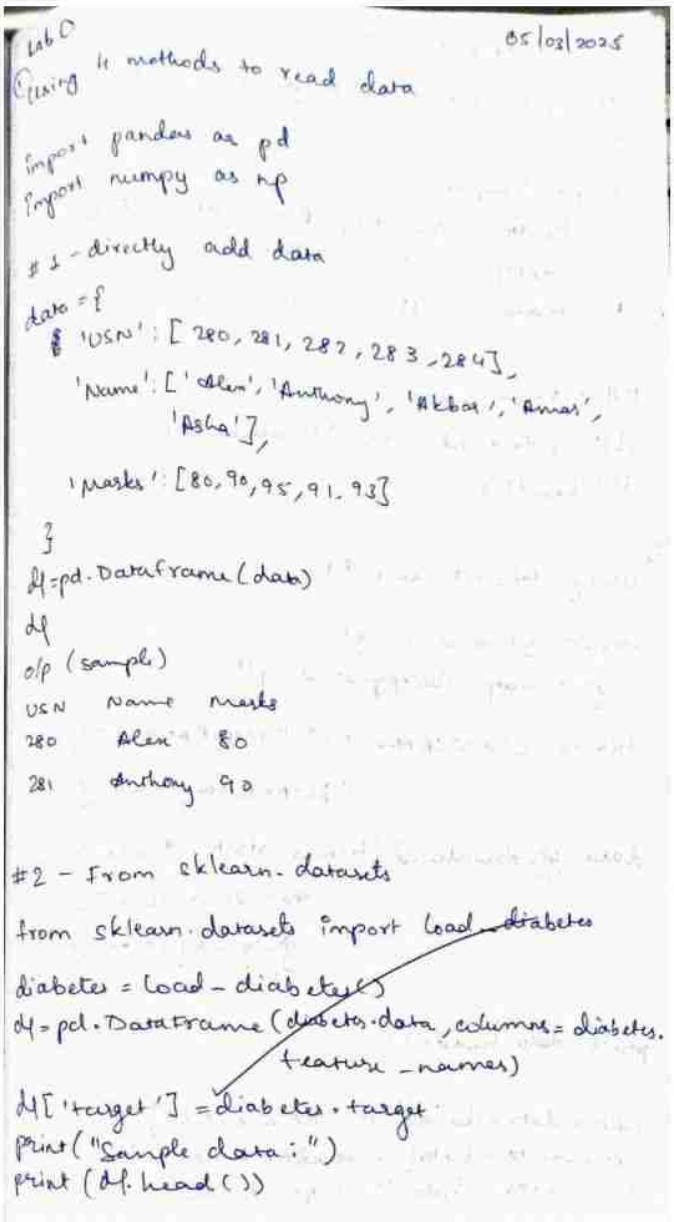
Github Link:

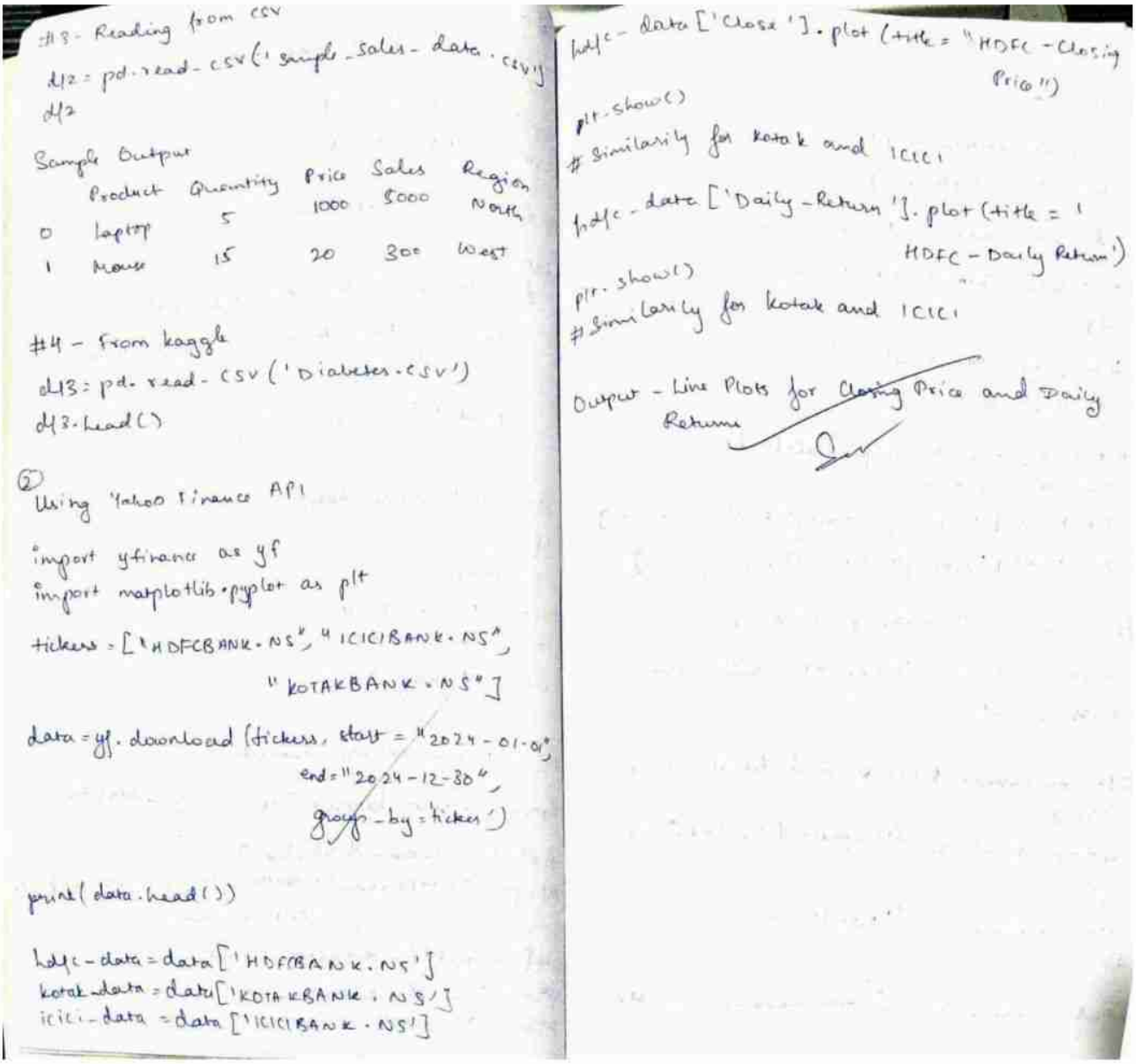
https://github.com/snehanshastri/ML

Program 1

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

Screenshot:





#3 - Reading from csv

df2 = pd.read_csv('sample_sales_data.csv')

df2

Sample Output

	Product	Quantity	Price	Sales	Region
0	laptop	5	1000	5000	North
1	Mouse	15	20	300	West

#4 - From kaggle

df3 = pd.read_csv('Diabetes.csv')

df3.head()

(2) Using Yahoo Finance API

import yfinance as yf

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(data.head())

hdfc_data = data['HDFCBANK.NS']

kotak_data = data['KOTAKBANK.NS']

icici_data = data['ICICIBANK.NS']

hdfc = data['Close'].plot(title="HDFC - Closing Price")

plt.show()

Similarly for kotak and ICICI

hdfc = data['Daily-Return'].plot(title='HDFC - Daily Return')

plt.show()

Similarly for kotak and ICICI

Output - Line Plots for Closing Price and Daily Returns

Code:

```
import pandas as pd

import numpy as np

data={
```

```
    'USN':[280,281,282,283,284],
    'Name':['Alex','Anthony','Akbar','Amar','Asha'],
    'Marks':[80,90,95,91,93]
}
df=pd.DataFrame(data)
df
from sklearn.datasets import load_diabetes


diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data,
columns=diabetes.feature_names)
df['target'] = diabetes.target
print("Sample data:")
print(df.head())
df2=pd.read_csv('/content/sample_sales_data.csv')
df2
df3=pd.read_csv('/content/Dataset of Diabetes .csv')
df3.head()
import yfinance as yf
import matplotlib.pyplot as plt
```

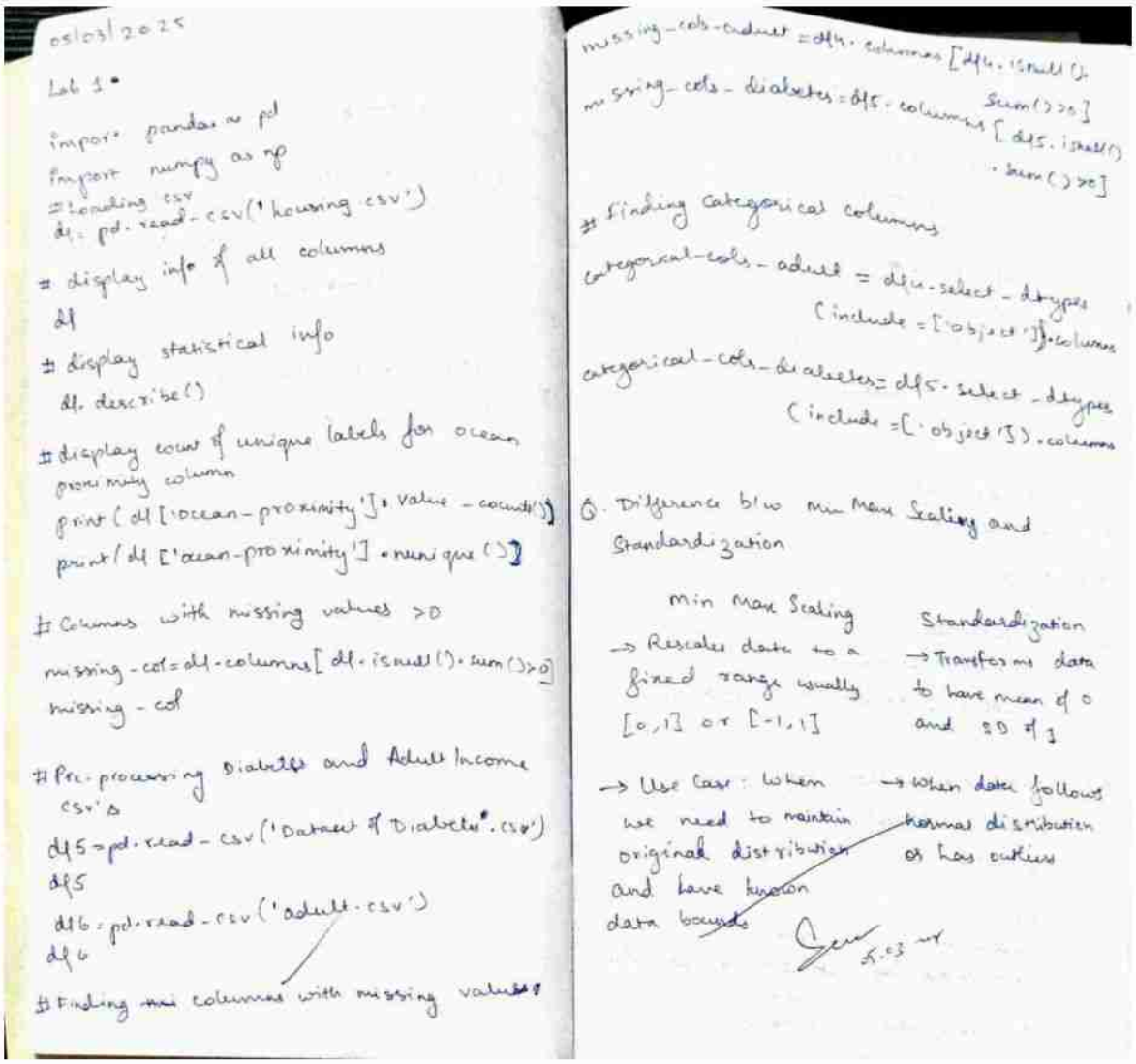
```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

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())



hdfc_data=data['HDFCBANK.NS']

kotak_data=data['KOTAKBANK.NS']

icici_data=data['ICICIBANK.NS']



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

plt.show()

kotak_data['Close'].plot(title="Kotak Bank - Closing Price")

plt.show()

icici_data['Close'].plot(title="ICICI Bank - Closing Price")

plt.show()



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

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

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

```
hdfc_data['Daily Return'].plot(title="HDFC - Daily Return")

plt.show()

kotak_data['Daily Return'].plot(title="Kotak Bank - Daily
Return")

plt.show()

icici_data['Daily Return'].plot(title="ICICI Bank - Daily
Return")

plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



Code:

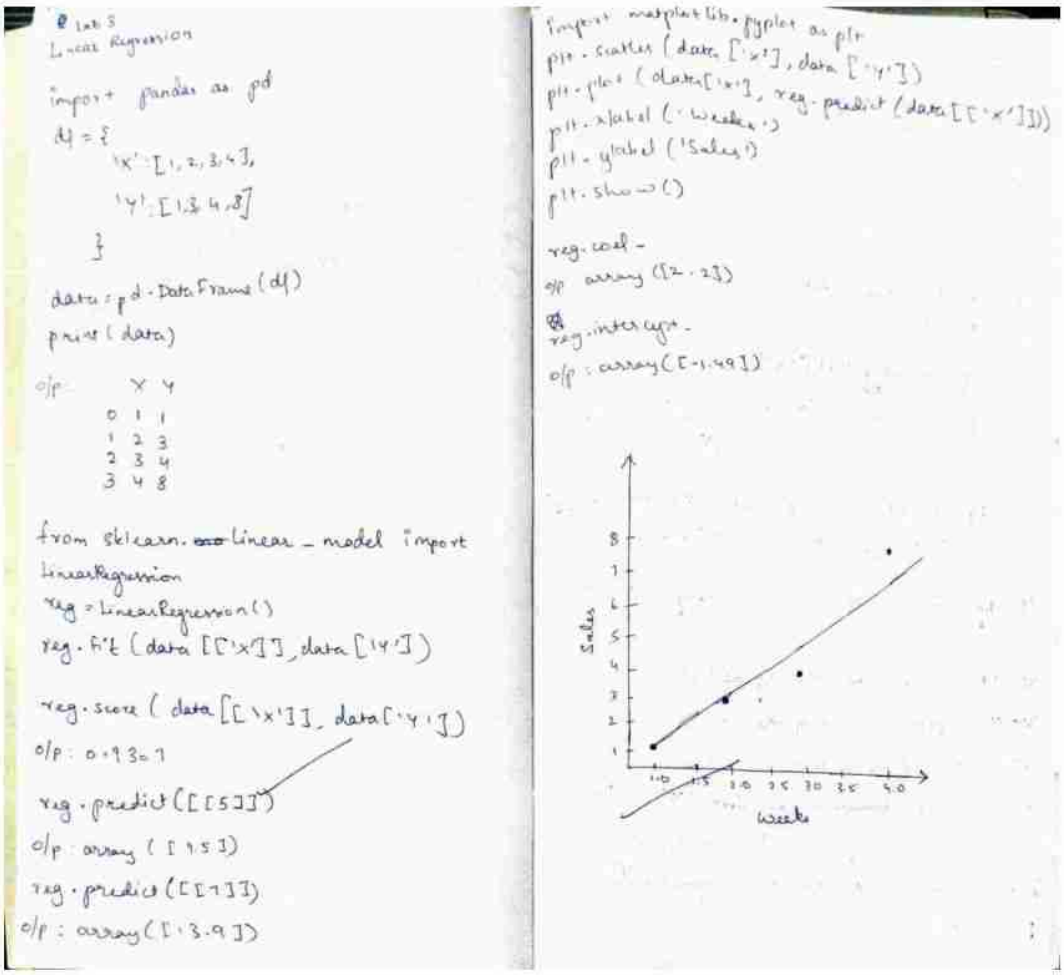
```
missing_col=df.columns[df.isnull().sum()>0]

missing_col

   # For 'adult.csv' (df4)
```

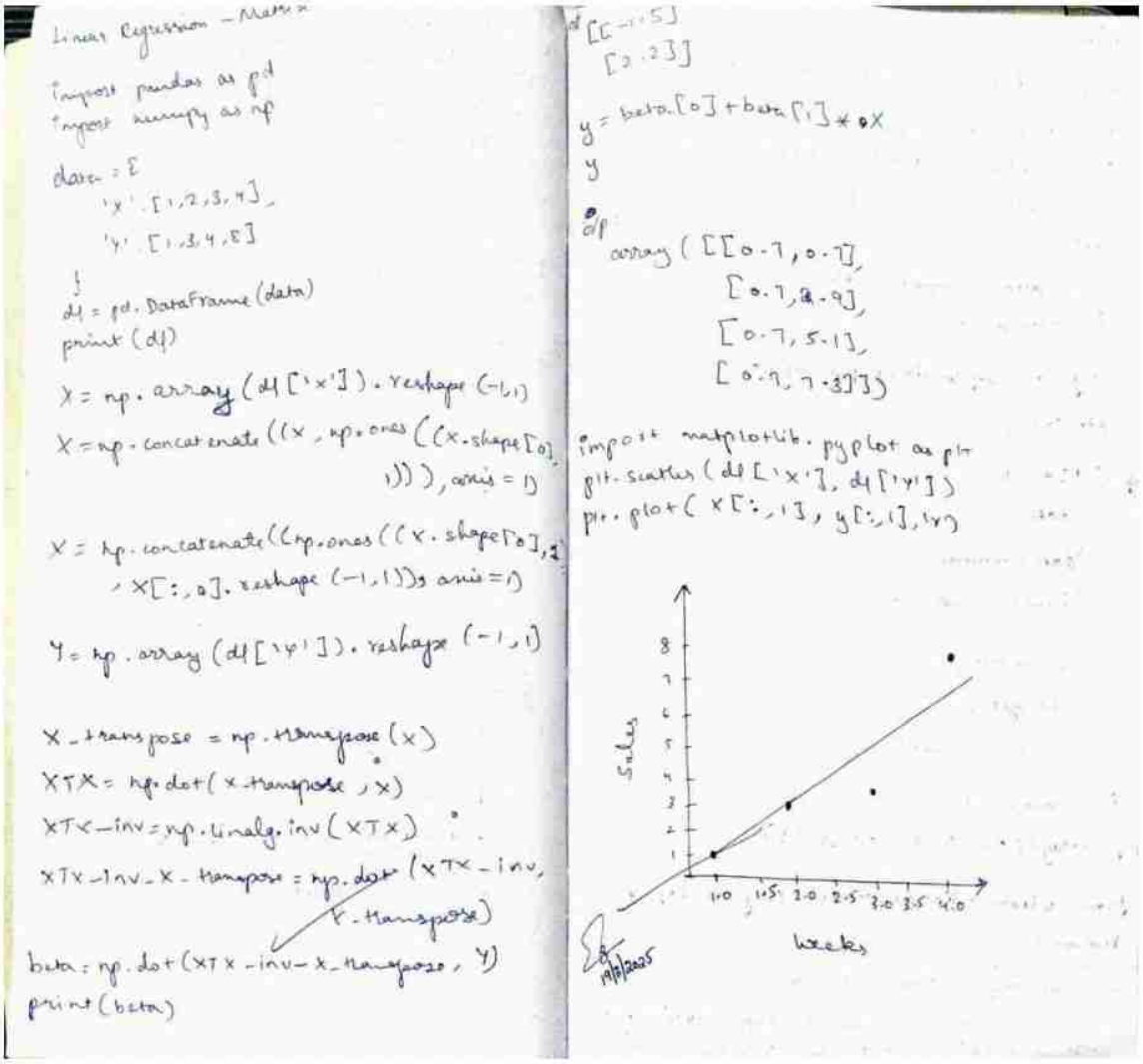
```
missing_cols_adult = df4.columns[df4.isnull().sum() > 0]

print(f"Columns with missing values in 'adult.csv':
{missing_cols_adult.tolist()}")



   # For 'Dataset of Diabetes .csv' (df5)

missing_cols_diabetes = df5.columns[df5.isnull().sum() > 0]

print(f"Columns with missing values in 'Dataset of Diabetes
.csv': {missing_cols_diabetes.tolist()}")

# For 'adult.csv' (df4)

categorical_cols_adult =
df4.select_dtypes(include=['object']).columns

print(f"Categorical columns in 'adult.csv':
{categorical_cols_adult.tolist()}")



   # For 'Dataset of Diabetes .csv' (df5)

categorical_cols_diabetes =
df5.select_dtypes(include=['object']).columns

print(f"Categorical columns in 'Dataset of Diabetes .csv':
{categorical_cols_diabetes.tolist()}")
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



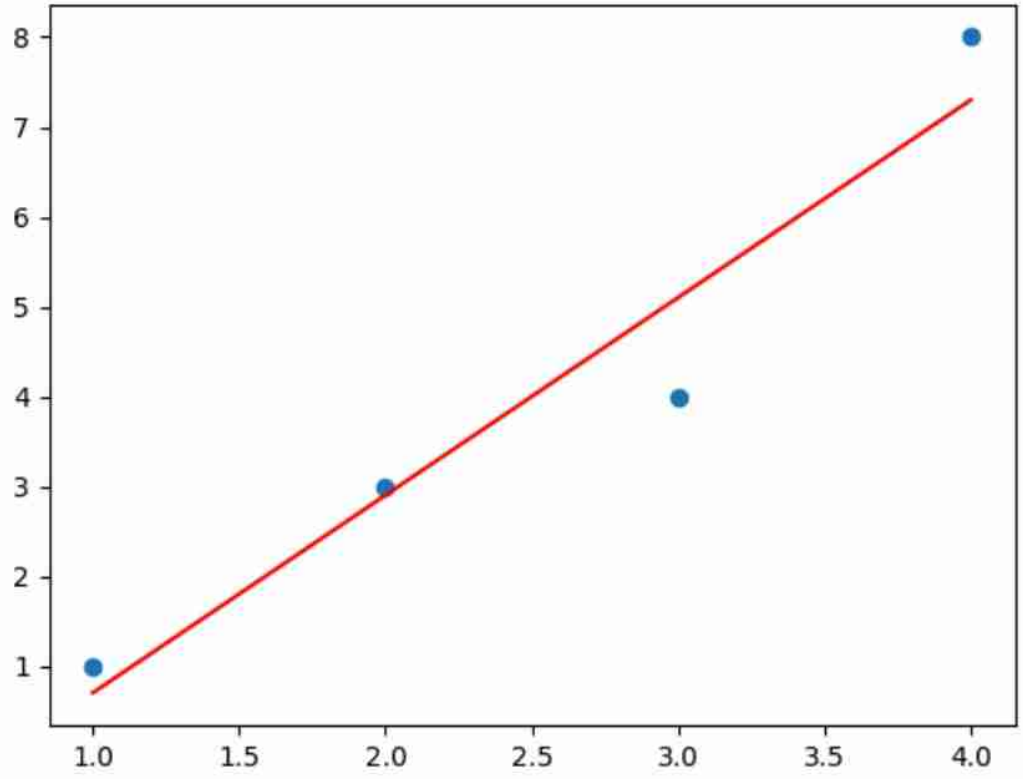


Code:

```
import pandas as pd

import numpy as np



data={

    'X':[1,2,3,4],
```

```
    'Y':[1,3,4,8]

}



df=pd.DataFrame(data)

print(df)



X=np.array(df['X']).reshape(-1,1)

X=np.concatenate((X,np.ones((X.shape[0],1))),axis=1)

X = np.concatenate((np.ones((X.shape[0], 1)),
X[:,0].reshape(-1,1)), axis=1)

Y=np.array(df['Y']).reshape(-1,1)



print(X)

print(Y)



X_transpose = np.transpose(X)

XTX = np.dot(X_transpose, X)

XTX_inv = np.linalg.inv(XTX)

XTX_inv_X_transpose = np.dot(XTX_inv, X_transpose)

beta = np.dot(XTX_inv_X_transpose, Y)

print(beta)
```

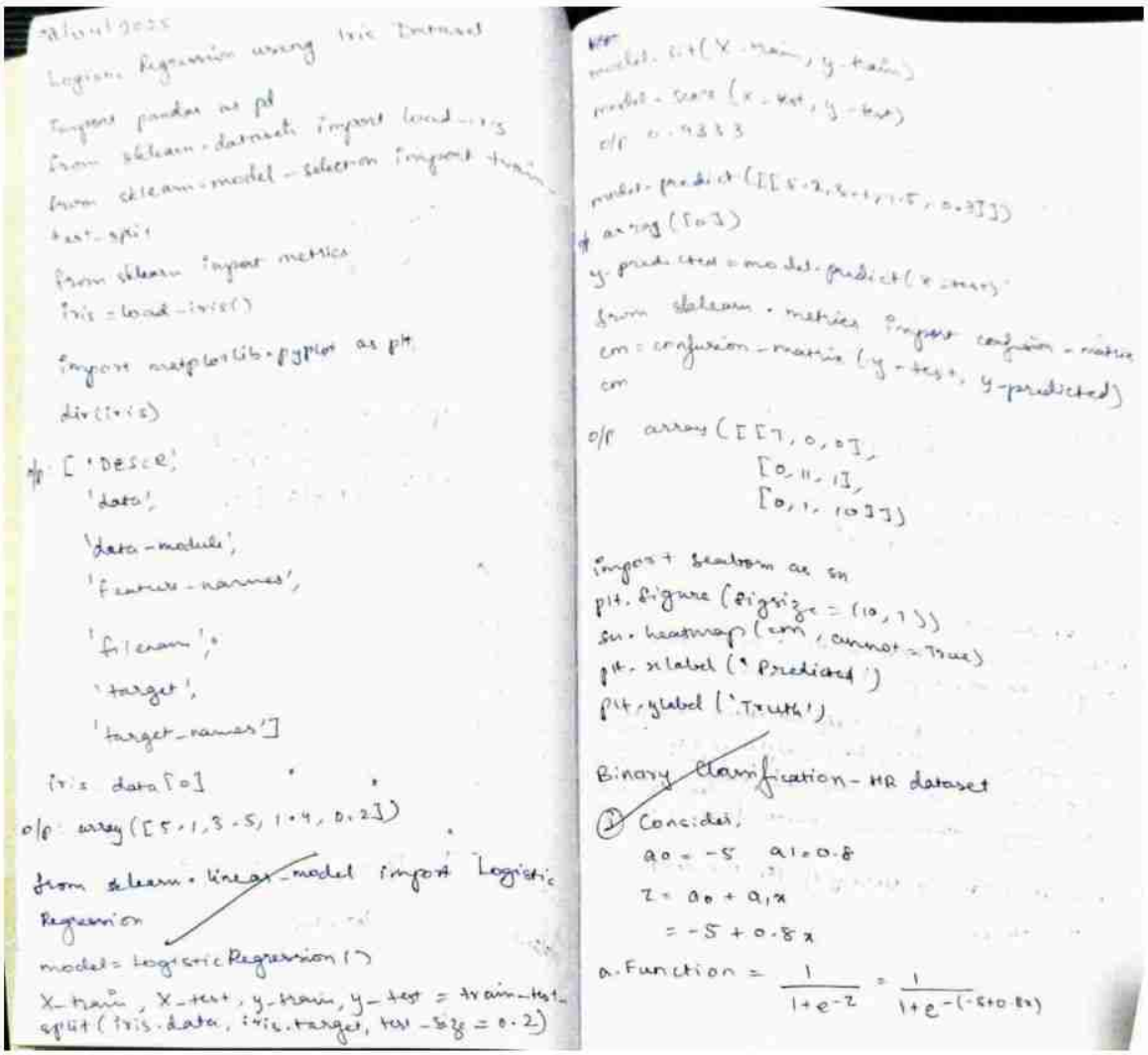
```
y=beta[0]+beta[1]*X

y

import matplotlib.pyplot as plt

plt.scatter(df['X'],df['Y'])

plt.plot(X[:,1],y[:,1],'r')
```

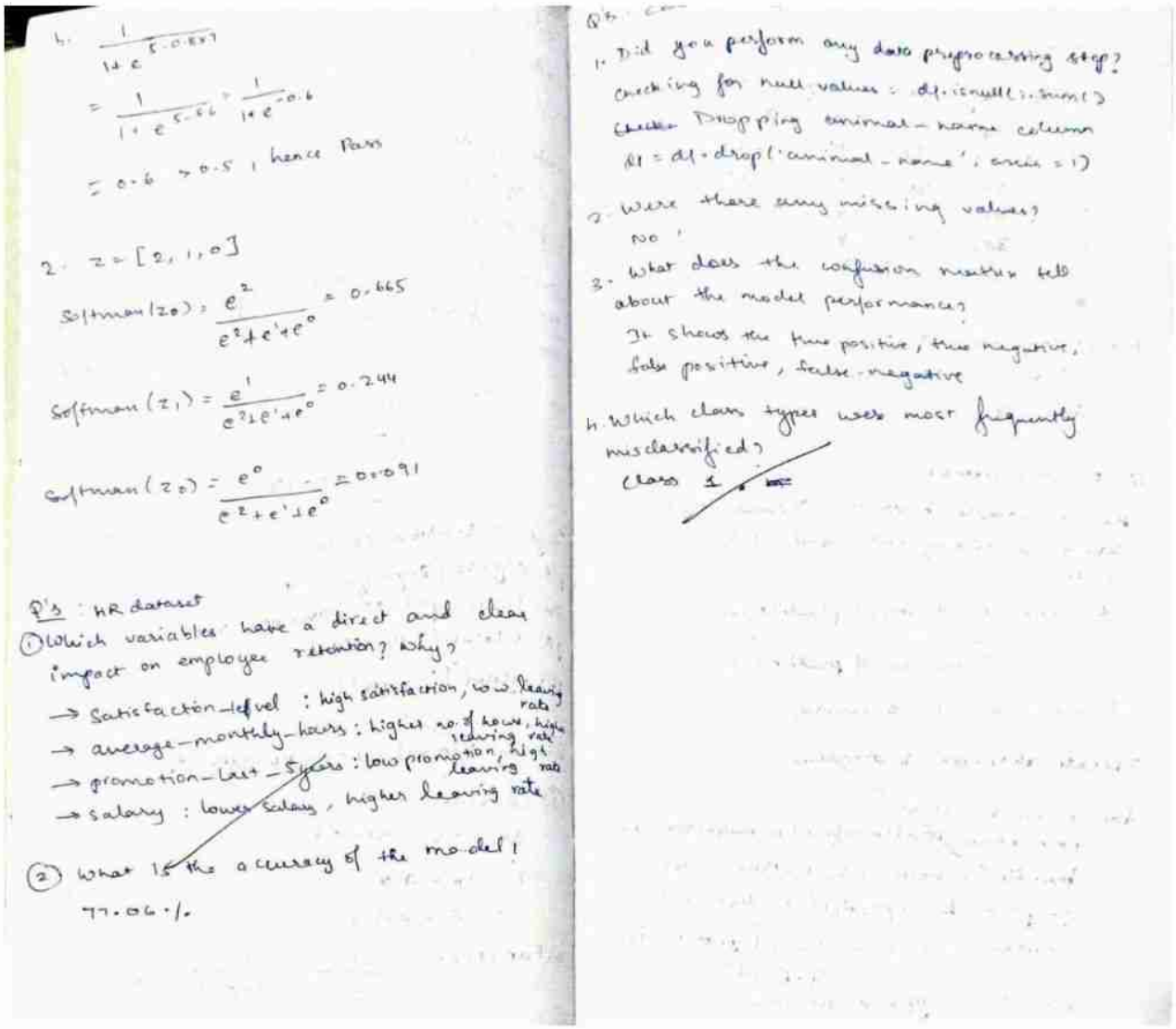


Program 4

Build Logistic Regression Model for a given dataset

Screenshot:



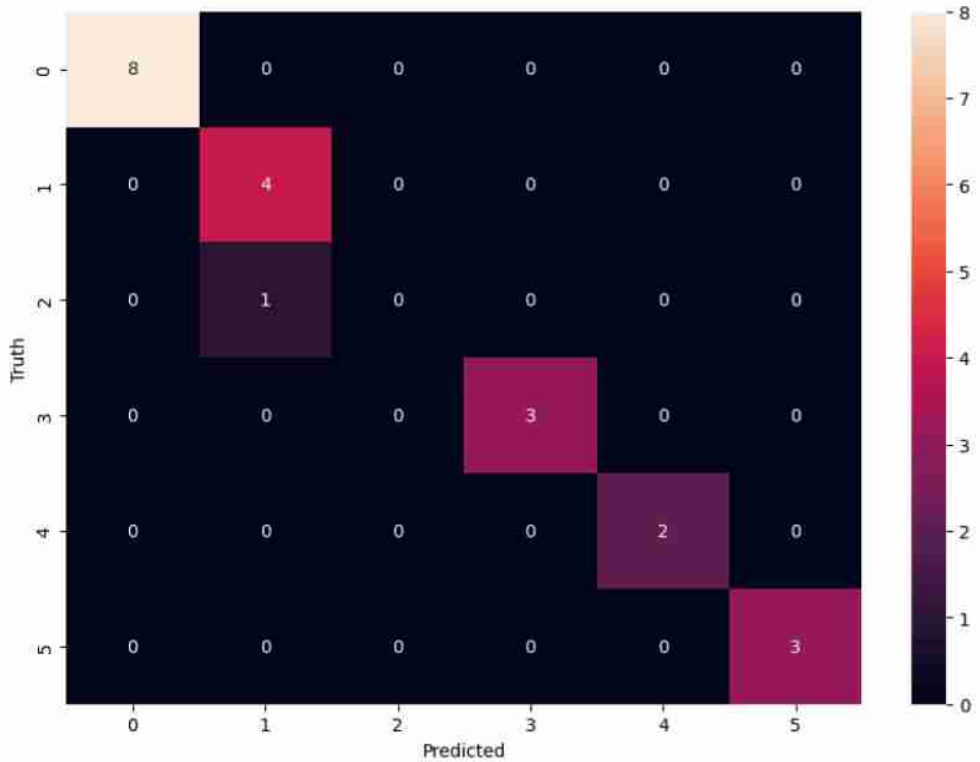


Code:

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn import metrics
```

```
import matplotlib.pyplot as plt

df=pd.read_csv('/content/zoo-data (3).csv')

df.head()

df.isnull().sum()

df=df.drop('animal_name',axis=1)

df.head()

target=df['class_type']

data=df.drop('class_type',axis=1)

data.head()

target.head()

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(data,target,tes
t_size=0.2)

from sklearn.linear_model import LogisticRegression

model=LogisticRegression()

model.fit(x_train,y_train)

from sklearn import metrics

model.score(x_test,y_test)

model.predict(x_test)

from sklearn.metrics import confusion_matrix
```

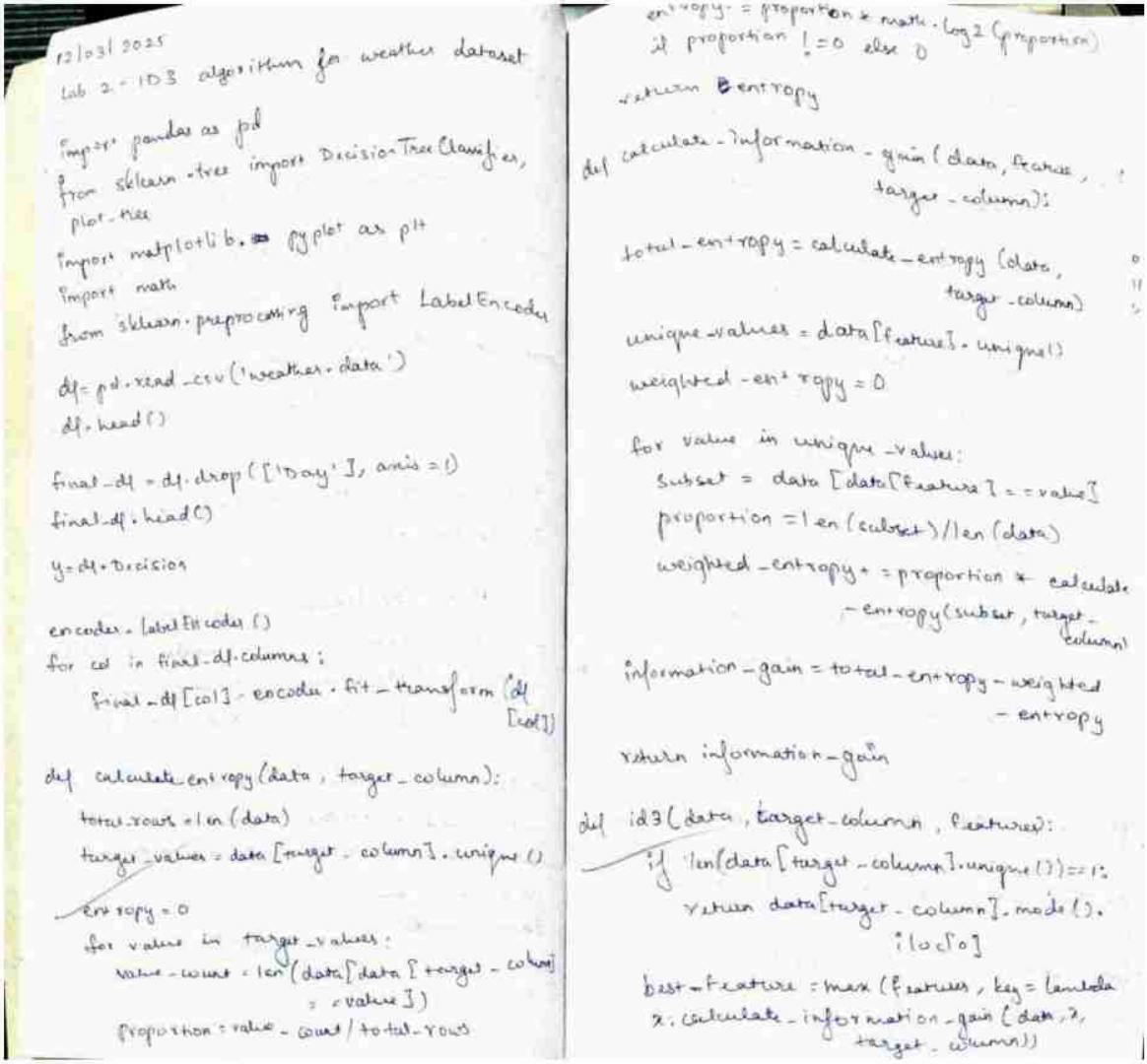
```
# Assuming 'y_test' and 'predictions' are your true labels and
model predictions

predictions = model.predict(x_test)

cm = confusion_matrix(y_test, predictions)



print(cm)

import seaborn as sn

plt.figure(figsize = (10,7))

sn.heatmap(cm, annot=True)

plt.xlabel('Predicted')

plt.ylabel('Truth')
```



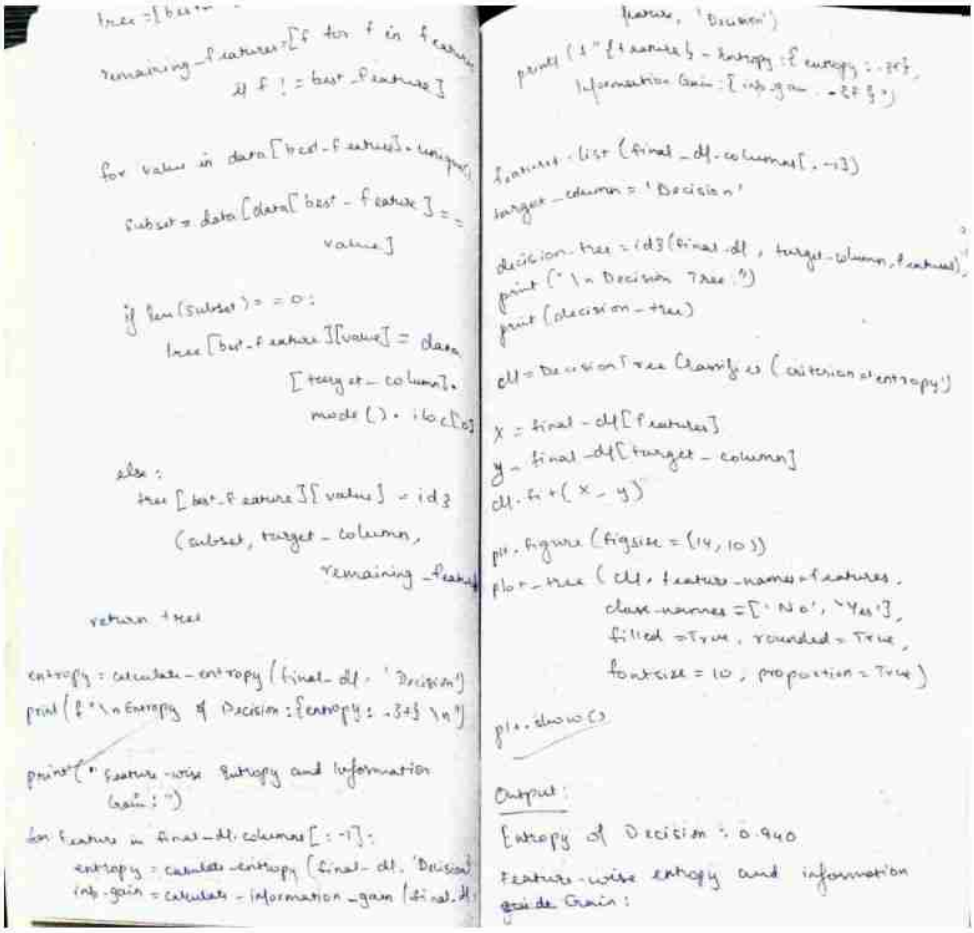
Program 5

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

Screenshot:



```
12/03/2025
Lab 2 - ID3 algorithm for weather dataset

import pandas as pd
from sklearn.tree import DecisionTreeClassifier,
plot_tree
import matplotlib.pyplot as plt
import math
from sklearn.preprocessing import LabelEncoder

df= pd.read_csv('weather.data')
df.head()

final_df = df.drop(['Day'], axis=1)
final_df.head()

y=df.Decision

encoder = LabelEncoder()
for col in final_df.columns:
    final_df[col] = encoder.fit_transform(df
    [col])

def calculate_entropy(data, target_column):
    total_rows = len(data)
    target_values = data[target_column].unique()

    entropy = 0
    for value in target_values:
        value_count = len(data[data[target_colu
                          == value])
        proportion = value_count / total_rows
        entropy += proportion * math.log2(proportion)
    if proportion != 0 else 0
    return Bentropy

def calculate_information_gain(data, feature,
                               target_column):
    total_entropy = calculate_entropy(data,
                                      target_column)
    unique_values = data[feature].unique()
    weighted_entropy = 0

    for value in unique_values:
        subset = data[data[feature] == value]
        proportion = len(subset)/len(data)
        weighted_entropy += proportion * calculate
                            _entropy(subset, target_
                                     column)
    information_gain = total_entropy - weighted
                       _entropy
    return information_gain

def id3(data, target_column, features):
    if len(data[target_column].unique())==1:
        return data[target_column].mode().
               iloc[0]
    best_feature = max(features, key= lambda
    x: calculate_information_gain(data, x,
                                  target_column))
```



```
tree = {best_feature: {}}
remaining_features = [f for f in features
                      if f != best_feature]

for value in data[best_feature].unique():
    subset = data[data[best_feature] ==
                  value]

    if len(subset) == 0:
        tree[best_feature][value] = data
            [target_column].
            mode().iloc[0]

    else:
        tree[best_feature][value] = id3
            (subset, target_column,
             remaining_features)

return tree

entropy = calculate_entropy(final_df, 'Decision')
print(f"\n Entropy of Decision: {entropy:.3f}\n")

print("Feature-wise Entropy and Information
      Gain:")
for feature in final_df.columns[:-1]:
    entropy = calculate_entropy(final_df, 'Decision')
    info_gain = calculate_information_gain(final_df,
                feature, 'Decision')
    print(f"{feature} - Entropy: {entropy:.3f},
          Information Gain: {info_gain:.3f}")

features = list(final_df.columns[:-1])
target_column = 'Decision'

decision_tree = id3(final_df, target_column, features)
print("\n Decision Tree:")
print(decision_tree)

clf = DecisionTreeClassifier(criterion='entropy')

X = final_df[features]
y = final_df[target_column]
clf.fit(X, y)

plt.figure(figsize=(14, 10))
plot_tree(clf, feature_names=features,
          class_names=['No', 'Yes'],
          filled=True, rounded=True,
          fontsize=10, proportion=True)

plt.show()
```

Output:

Entropy of Decision: 0.940

Feature-wise entropy and information gain:

Outlook - Entropy: 0.693, Information Gain: 0.247

Temp - Entropy: 0.940, Information Gain: 0.029

Humidity - Entropy: 0.940, Information Gain: 0.152

Wind - Entropy: 0.940, Information Gain: 0.048

Decision Tree:

{Outlook: {'Overcast': 'Yes', 'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}

Code:

```
import pandas as pd

from sklearn.tree import DecisionTreeClassifier, plot_tree

import matplotlib.pyplot as plt

import math

from sklearn.preprocessing import LabelEncoder



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

df.head()

final_df=df.drop(['Day'],axis=1)

final_df.head()

y=df.Decision

y

# Encode categorical features into numeric values for
DecisionTreeClassifier

encoder = LabelEncoder()

for col in final_df.columns:

    final_df[col] = encoder.fit_transform(df[col])
```

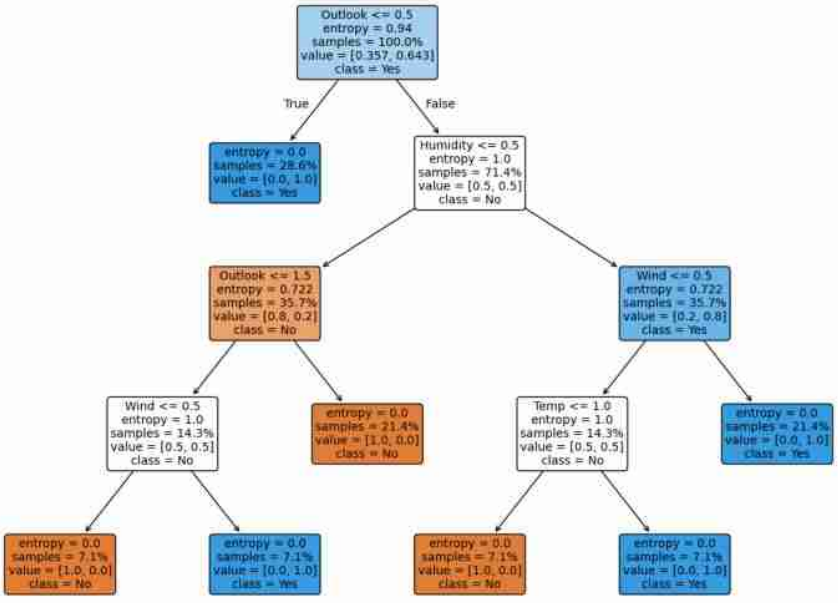
```
# Function to calculate entropy
def calculate_entropy(data, target_column):
    total_rows = len(data)
    target_values = data[target_column].unique()

    entropy = 0
    for value in target_values:
        value_count = len(data[data[target_column] == value])
        proportion = value_count / total_rows
        entropy -= proportion * math.log2(proportion) if
proportion != 0 else 0

    return entropy

# Function to calculate information gain
def calculate_information_gain(data, feature, target_column):
    total_entropy = calculate_entropy(data, target_column)

    unique_values = data[feature].unique()
    weighted_entropy = 0
```

```
    for value in unique_values:
        subset = data[data[feature] == value]
        proportion = len(subset) / len(data)
        weighted_entropy += proportion *
calculate_entropy(subset, target_column)

    information_gain = total_entropy - weighted_entropy
    return information_gain

# Function to implement the ID3 algorithm
def id3(data, target_column, features):
    # If all target values are the same, return that value
    if len(data[target_column].unique()) == 1:
        return data[target_column].iloc[0]

    # If no features left, return the most common target value
    if len(features) == 0:
        return data[target_column].mode().iloc[0]

    # Find the best feature based on information gain
    best_feature = max(features, key=lambda x:
```

```
calculate_information_gain(data, x, target_column))


    # Create a new tree node
    tree = {best_feature: {}}


    # Remove the best feature from the list of available
features
    remaining_features = [f for f in features if f !=
best_feature]


    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        if len(subset) == 0:
            tree[best_feature][value] =
data[target_column].mode().iloc[0]
        else:
            tree[best_feature][value] = id3(subset,
target_column, remaining_features)


    return tree


# Display entropy of target column
```

```
entropy = calculate_entropy(final_df, 'Decision')

print(f"\nEntropy of Decision: {entropy:.3f}\n")



# Display information gain for each feature

print("Feature-wise Entropy and Information Gain:")

for feature in final_df.columns[:-1]:

    entropy = calculate_entropy(final_df, 'Decision')

    info_gain = calculate_information_gain(final_    df,
feature, 'Decision')

    print(f"{feature} - Entropy: {entropy:.3f}, Information
Gain: {info_gain:.3f}")



# Build decision tree using ID3

features = list(final_df.columns[:-1])  # All columns except
the target

target_column = 'Decision'



decision_tree = id3(final_df, target_column, features)

print("\nDecision Tree:")

print(decision_tree)



# --- Train and plot using DecisionTreeClassifier for
```

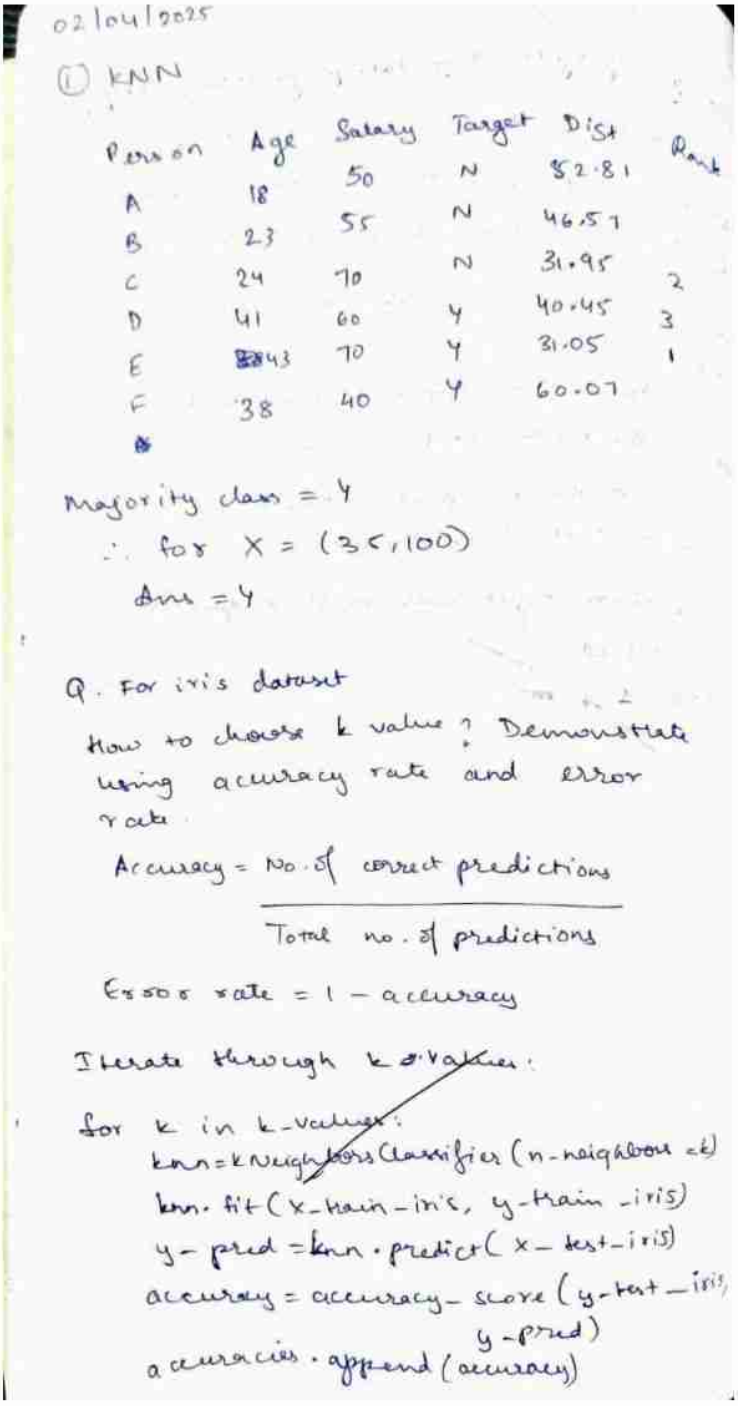
```
visualization ---

clf = DecisionTreeClassifier(criterion='entropy')



# Prepare data for the classifier

X = final_df[features]

y = final_df[target_column]

clf.fit(X, y)



# Plot the complete decision tree

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

plot_tree(clf, feature_names=features, class_names=['No',
'Yes'], filled=True, rounded=True, fontsize=10,
proportion=True)

plt.show()
```



Program 6

Build KNN Classification model for a given dataset

Screenshot:



02/04/2025

(1) KNN

Person	Age	Salary	Target	Dist	Rank
A	18	50	N	52.81	
B	23	55	N	46.57	
C	24	70	N	31.95	2
D	41	60	Y	40.45	3
E	43	70	Y	31.05	1
F	38	40	Y	60.07	

Majority class = Y

∴ for X = (35, 100)

Ans = Y

Q. For iris dataset

How to choose k value ? Demonstrate using accuracy rate and error rate

Accuracy = No. of correct predictions / Total no. of predictions

Error rate = 1 − accuracy

Iterate through k values:

```
for k in k_values:
    knn = kNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train_iris, y_train_iris)
    y_pred = knn.predict(x_test_iris)
    accuracy = accuracy_score(y_test_iris, y_pred)
    accuracies.append(accuracy)
```

error_rate.append(1 - accuracy)

Choose corresponding k-value with highest accuracy and lowest error rate

In this case : k = 8 11

Q. For diabetes dataset
What is the purpose of feature scaling?
How to perform it?

→ It is used to separate the data values from the target variables.

→ ~~y = pd.dr~~
y = df['target']
df = df.drop('target', axis = 1)

2/4/2025

Code:

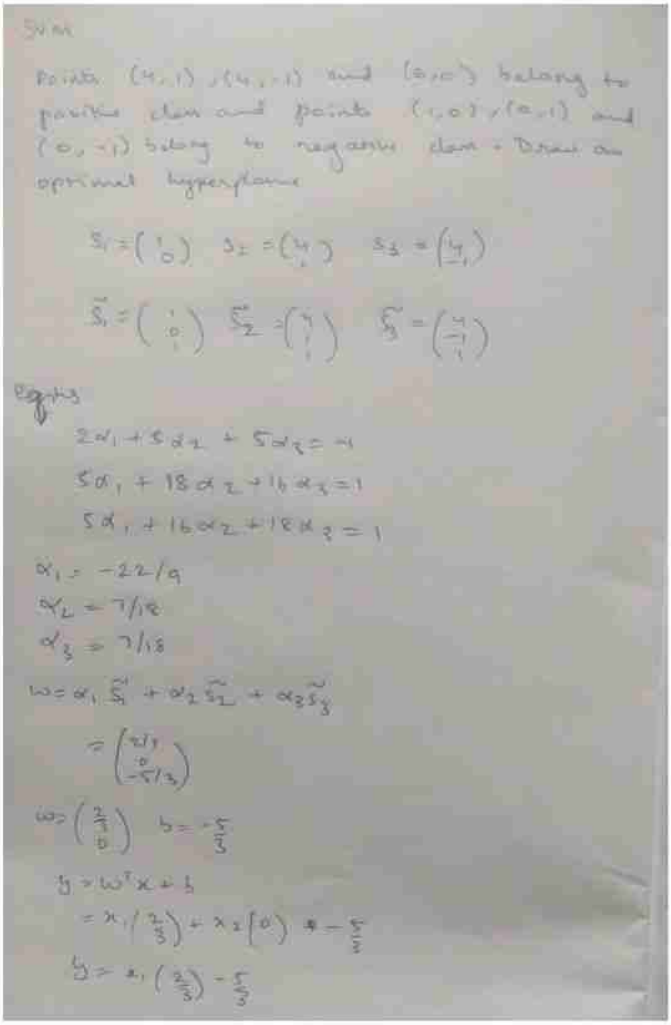
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report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


# Load Iris dataset from sklearn
dataset_iris = datasets.load_diabetes()
X_iris = dataset_iris.data  # Features
y_iris = dataset_iris.target  # Target


# Split dataset into 80% training and 20% testing
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)


# Choose an appropriate k (e.g., sqrt of number of samples)
k_iris = int(np.sqrt(len(X_train_iris)))
```

```
k_iris = k_iris if k_iris % 2 != 0 else k_iris + 1  # Ensure k
is odd


# Build and train KNN classifier
knn_iris = KNeighborsClassifier(n_neighbors=k_iris)
knn_iris.fit(X_train_iris, y_train_iris)


# Predictions and evaluation
y_pred_iris = knn_iris.predict(X_test_iris)
accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
conf_matrix_iris = confusion_matrix(y_test_iris, y_pred_iris)
report_iris = classification_report(y_test_iris, y_pred_iris)


print(f"Iris Dataset - KNN Classifier (k={k_iris})")
print(f"Accuracy: {accuracy_iris:.2f}")
print("Confusion Matrix:")
print(conf_matrix_iris)
print("Classification Report:")
print(report_iris)
```

Program 7

Build Support vector machine model for a given dataset

Screenshot:

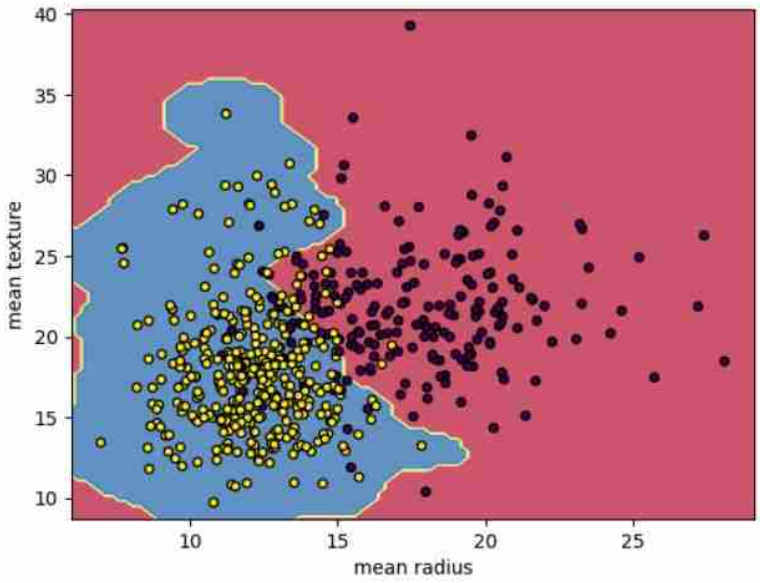


Code:

```
# Load the important packages

from sklearn.datasets import load_breast_cancer

import matplotlib.pyplot as plt

from sklearn.inspection import DecisionBoundaryDisplay
```

```
from sklearn.svm import SVC



# Load the datasets

cancer = load_breast_cancer()

X = cancer.data[:, :2]

y = cancer.target



#Build the model

svm = SVC(kernel="rbf", gamma=0.5, C=1.0)

# Trained the model

svm.fit(X, y)



# Plot Decision Boundary

DecisionBoundaryDisplay.from_estimator(

        svm,

        X,

        response_method="predict",

        cmap=plt.cm.Spectral,

        alpha=0.8,

        xlabel=cancer.feature_names[0],

        ylabel=cancer.feature_names[1],

    )
```

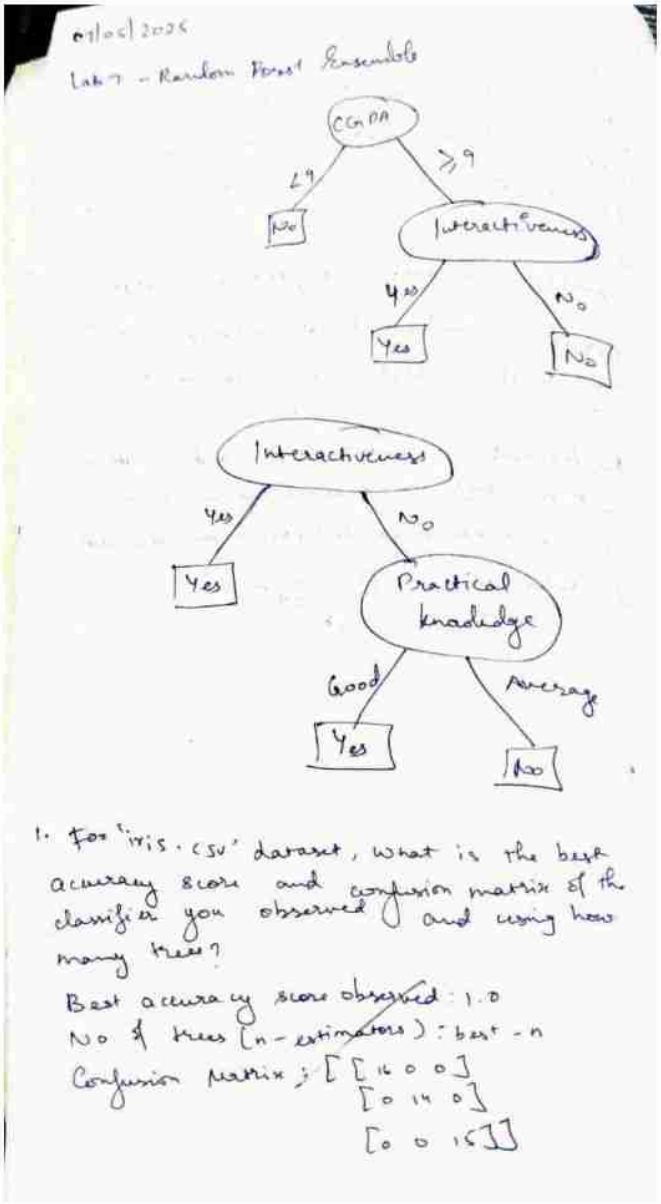
```
# Scatter plot
plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```



Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:



Code:

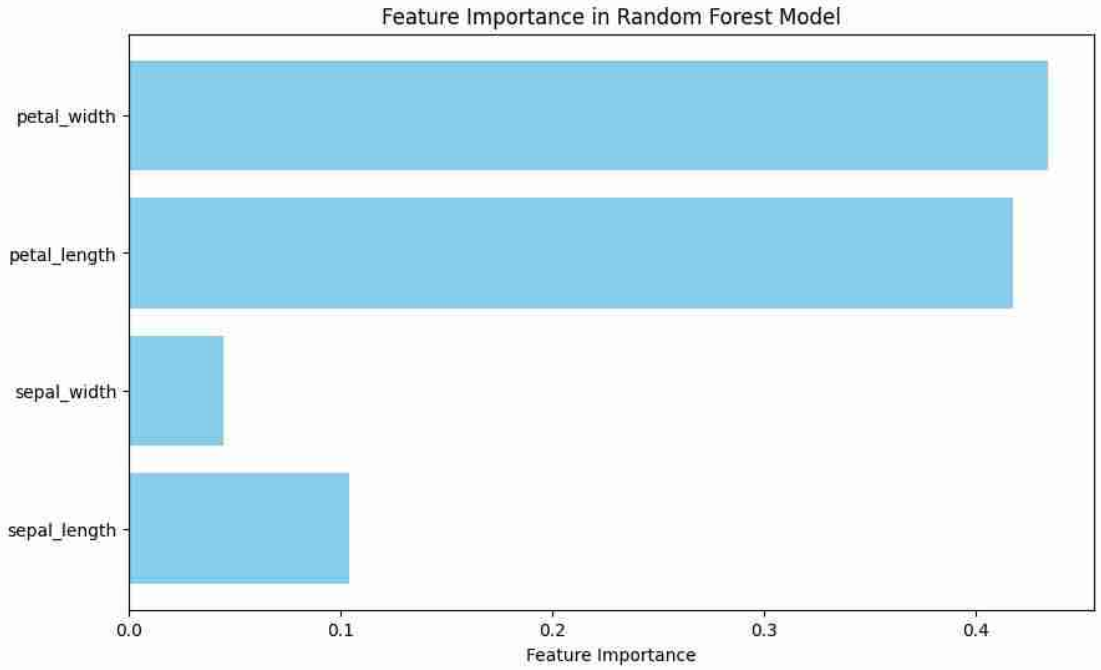
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt


# Step 1: Load the dataset using pandas read_csv

# Replace the file path with the path to your CSV file

data = pd.read_csv('/content/iris (6).csv')


# Step 2: Inspect the first few rows of the dataset

print(data.head())


# Step 3: Preprocessing (assuming the target variable is in the
last column)

X = data.iloc[:, :-1]  # Feature matrix (all columns except the
target column)

y = data.iloc[:, -1]   # Target variable (last column)


# Step 4: Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)


# Step 5: Initialize and train the Random Forest Classifier
```

```
rf_model = RandomForestClassifier(n_estimators=100,
random_state=42)

rf_model.fit(X_train, y_train)


# Step 6: Make predictions on the test set

y_pred = rf_model.predict(X_test)


# Step 7: Evaluate the model performance using accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of Random Forest model: {accuracy * 100:.2f}%")


# Step 8: Feature Importance (Optional)

feature_importances = rf_model.feature_importances_

features = X.columns


# Plotting feature importances

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

plt.barh(features, feature_importances, color='skyblue')

plt.xlabel('Feature Importance')

plt.title('Feature Importance in Random Forest Model')

plt.show()
```

```
# Step 9: Hyperparameter Tuning (Optional)

# If you want to perform hyperparameter tuning to optimize the
model

from sklearn.model_selection import GridSearchCV


param_grid = {

    'n_estimators': [100, 200, 300],

    'max_depth': [None, 10, 20, 30],

    'min_samples_split': [2, 5, 10],

    'min_samples_leaf': [1, 2, 4]

}


# Initialize GridSearchCV

grid_search = GridSearchCV(estimator=rf_model,
param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

grid_search.fit(X_train, y_train)


# Print the best parameters and best score

print(f"Best parameters found: {grid_search.best_params_}")

print(f"Best cross-validation score:
{grid_search.best_score_:.2f}")
```

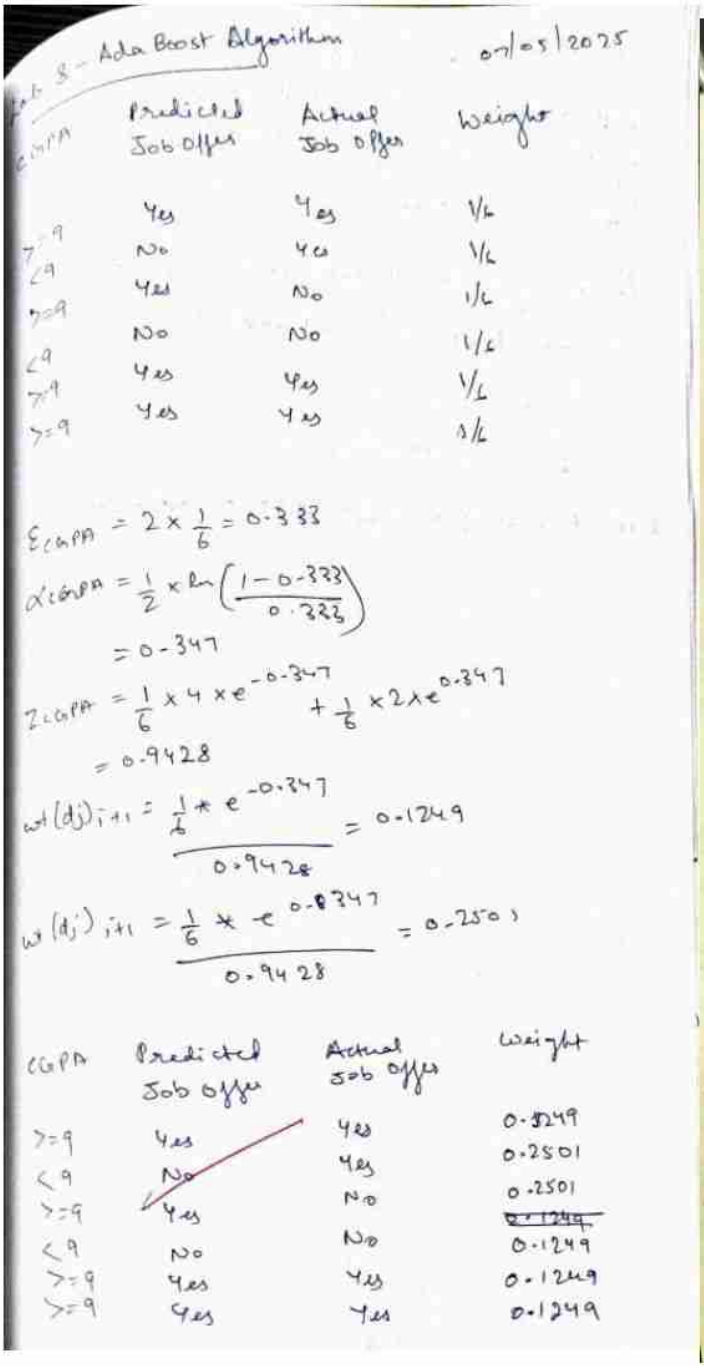
```
# Step 10: Evaluate the model with the best parameters
best_rf_model = grid_search.best_estimator_
y_pred_best = best_rf_model.predict(X_test)

accuracy_best = accuracy_score(y_test, y_pred_best)
print(f"Accuracy of tuned Random Forest model: {accuracy_best *
100:.2f}%")
```

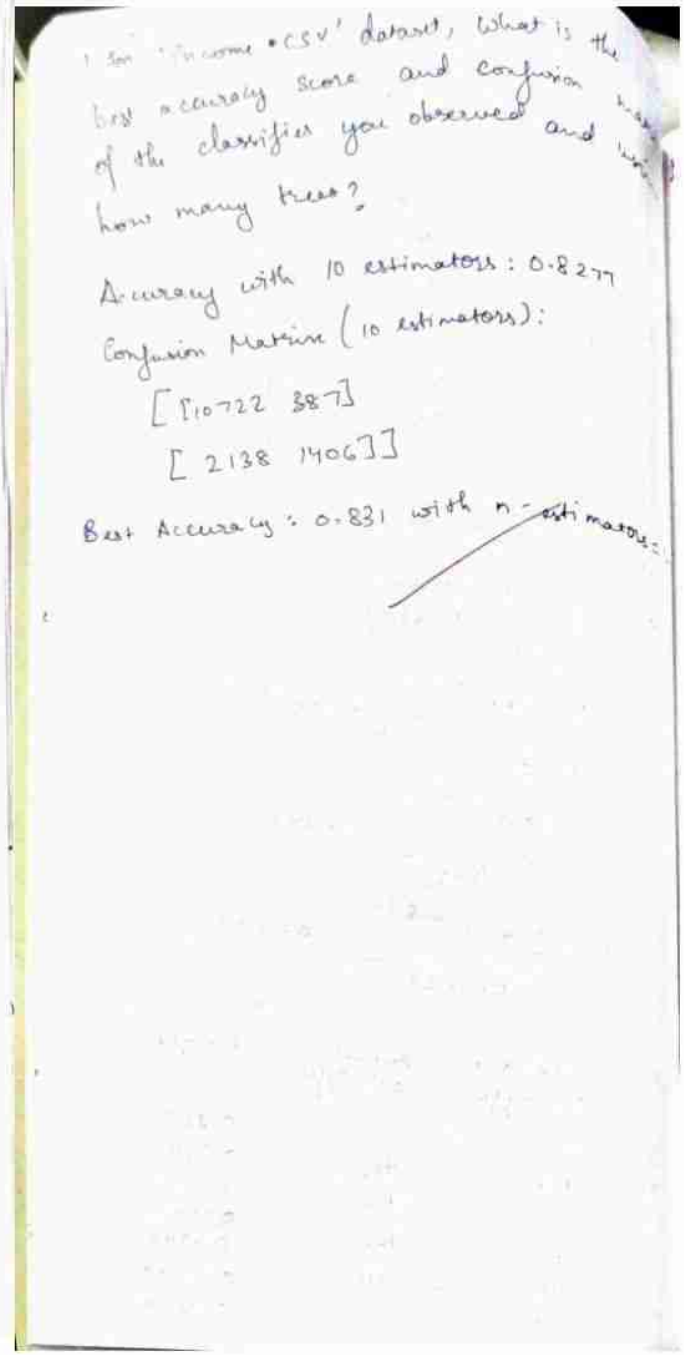


Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:





Code:

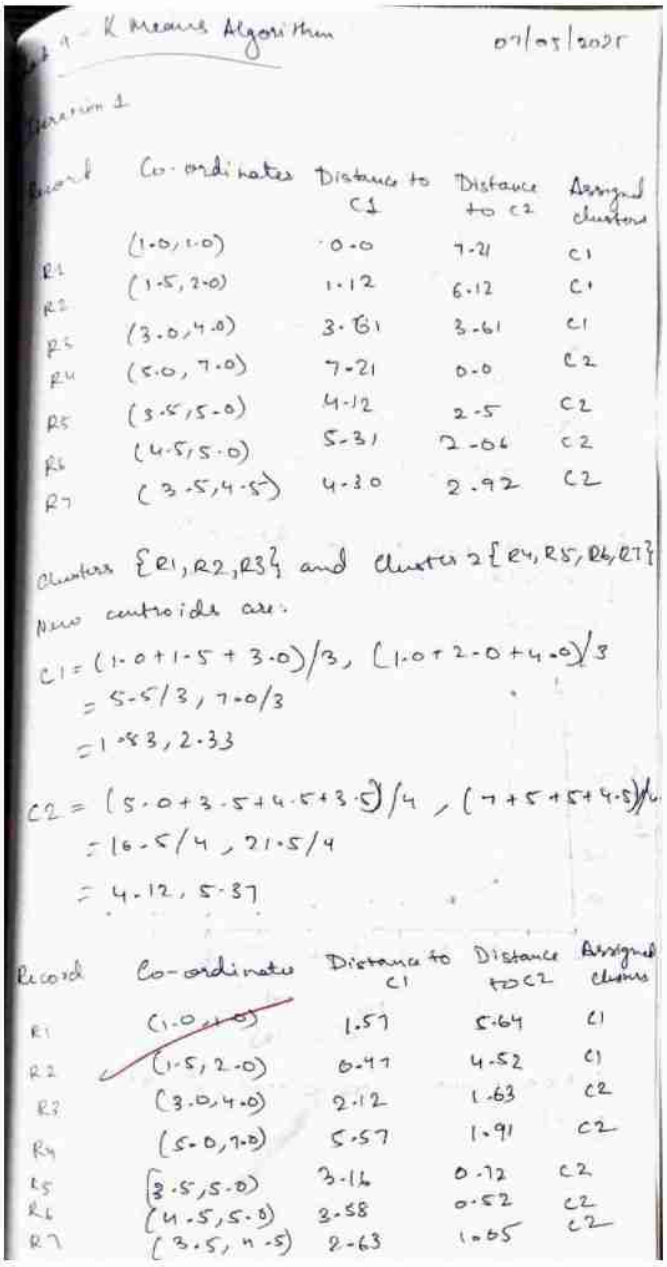
```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier  # For the
base estimator

from sklearn.metrics import accuracy_score



# 1. Load the Iris dataset:

# Make sure 'iris.csv' is in the correct location (see
previous response)

data = pd.read_csv('/content/income.csv')



# 2. Prepare the data:

X = data.iloc[:, :-1]  # Features (all columns except the
last)

y = data.iloc[:, -1]   # Target (last column)



# 3. Split into training and testing sets:

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

```
# 4. Create the base estimator (a weak learner):

# We'll use a Decision Tree with a maximum depth of 1 (a
stump)

base_estimator = DecisionTreeClassifier(max_depth=1)



# 5. Create the AdaBoost classifier:

ada_boost = AdaBoostClassifier(estimator=base_estimator,  #
Using the base estimator

                               n_estimators=50,          #
Number of boosting rounds

                               random_state=42)



# 6. Train the model:

ada_boost.fit(X_train, y_train)



# 7. Make predictions:

y_pred = ada_boost.predict(X_test)



# 8. Evaluate the model:

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of AdaBoost on Iris dataset: {accuracy *
98.7:.2f}%")
```

Program 10

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

Screenshot:



K Means Algorithm 07/05/2025

Iteration 1

Record	Co-ordinates	Distance to C1	Distance to C2	Assigned clusters
R1	(1.0, 1.0)	0.0	7.21	C1
R2	(1.5, 2.0)	1.12	6.12	C1
R3	(3.0, 4.0)	3.61	3.61	C1
R4	(5.0, 7.0)	7.21	0.0	C2
R5	(3.5, 5.0)	4.12	2.5	C2
R6	(4.5, 5.0)	5.31	2.06	C2
R7	(3.5, 4.5)	4.30	2.92	C2

Clusters {R1, R2, R3} and Cluster 2 {R4, R5, R6, R7}

New centroids are:

C1 = (1.0 + 1.5 + 3.0)/3, (1.0 + 2.0 + 4.0)/3

= 5.5/3, 7.0/3

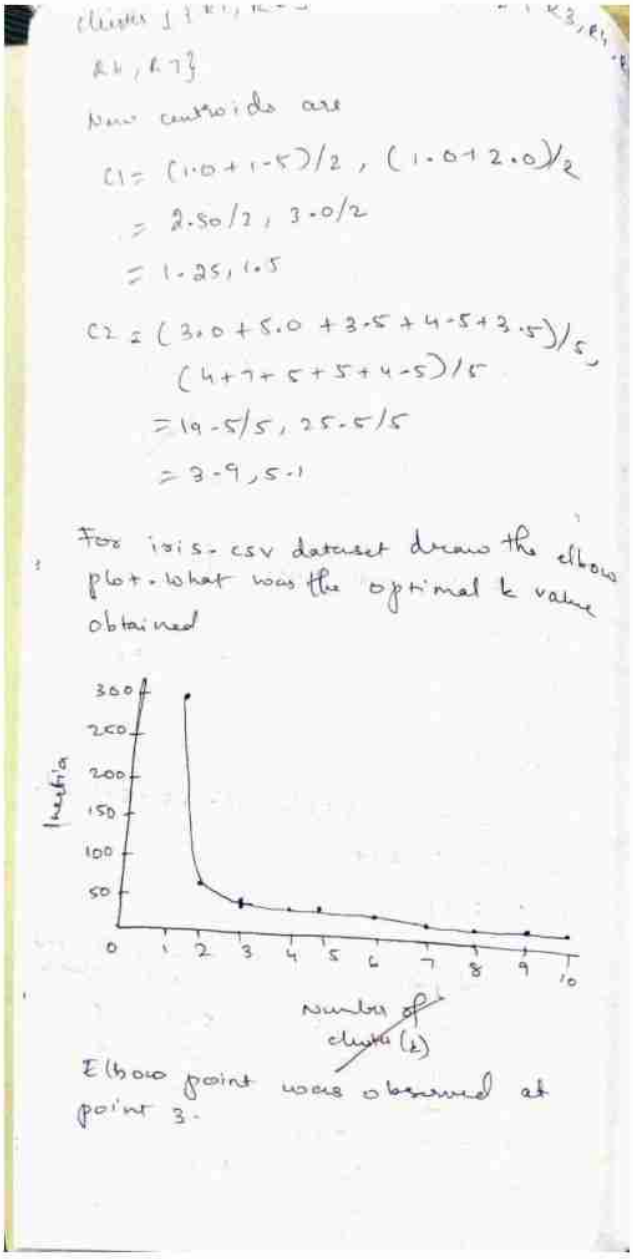
= 1.83, 2.33

C2 = (5.0 + 3.5 + 4.5 + 3.5)/4, (7 + 5 + 5 + 4.5)/4

= 16.5/4, 21.5/4

= 4.12, 5.37

Record	Co-ordinates	Distance to C1	Distance to C2	Assigned clusters
R1	(1.0, 1.0)	1.57	5.64	C1
R2	(1.5, 2.0)	0.47	4.52	C1
R3	(3.0, 4.0)	2.12	1.63	C2
R4	(5.0, 7.0)	5.57	1.91	C2
R5	(3.5, 5.0)	3.16	0.72	C2
R6	(4.5, 5.0)	3.58	0.52	C2
R7	(3.5, 4.5)	2.63	1.05	C2



R6, R7}

New centroids are

C1 = (1.0 + 1.5)/2, (1.0 + 2.0)/2

= 2.50/2, 3.0/2

= 1.25, 1.5

C2 = (3.0 + 5.0 + 3.5 + 4.5 + 3.5)/5, (4 + 7 + 5 + 5 + 4.5)/5

= 19.5/5, 25.5/5

= 3.9, 5.1

For iris.csv dataset draw the elbow plot. What was the optimal k value obtained

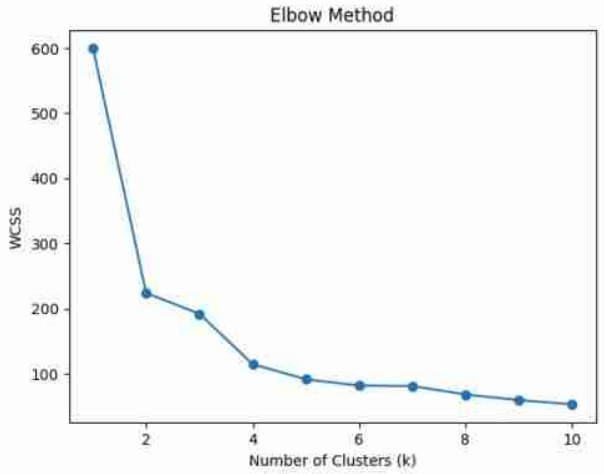
Elbow point was observed at point 3.

Code:

```
import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt



# 1. Load the Iris dataset:

data = pd.read_csv('/content/iris (6).csv')



# 2. Prepare the data:

X = data.iloc[:, :-1]  # Features (all columns except the
last)



# 3. Standardize the features (important for k-Means):

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)



# 4. Determine the optimal number of clusters (using the Elbow
method):

wcss = []  # Within-cluster sum of squares

for i in range(1, 11):
```

```
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plot the Elbow method graph
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.show()

# Based on the Elbow method, choose the optimal k (e.g., k=3)

# 5. Apply k-Means clustering with the chosen k:
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# 6. Add the cluster labels to the DataFrame:
data['Cluster'] = clusters

# 7. Visualize the clusters (if possible - 2D or 3D):
```

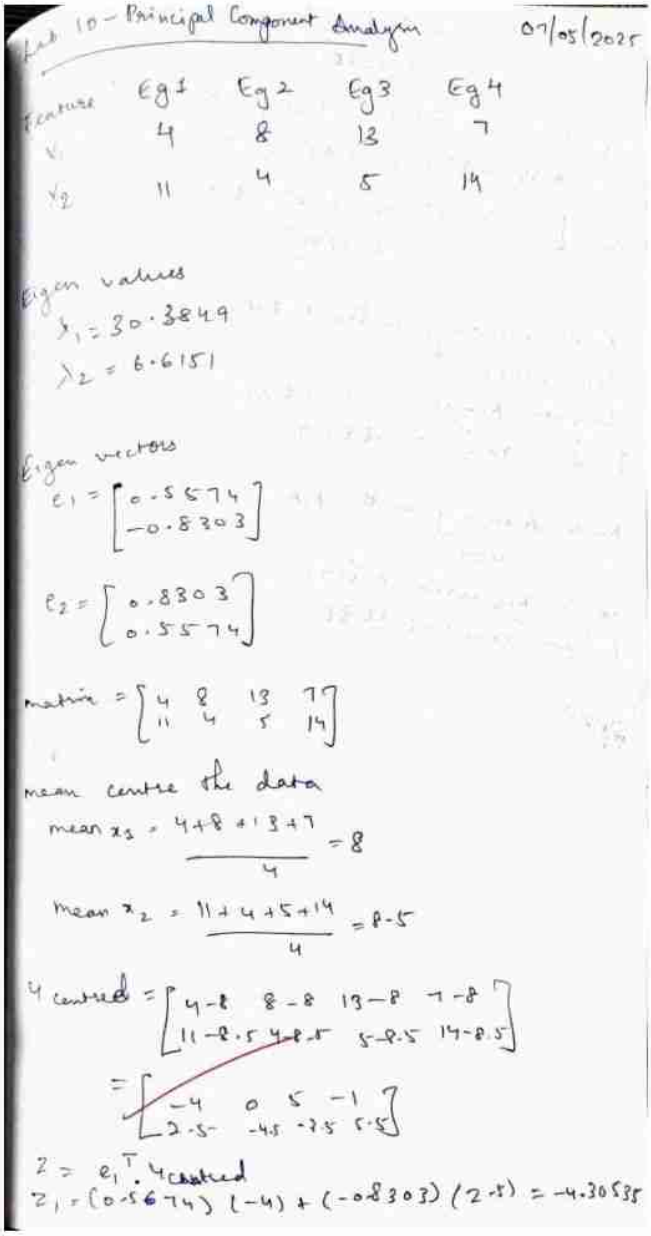
```
# Example for 2 features: Sepal Length and Sepal Width

plt.scatter(data['SepalLengthCm'], data['SepalWidthCm'],
c=data['Cluster'], cmap='viridis')

plt.title('k-Means Clustering')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.show()



# You can explore other feature combinations for
visualization.



# 8. Analyze the clusters (optional):

# You can examine the characteristics of each cluster by

# calculating the mean values of features within each cluster.

# Example:

# print(data.groupby('Cluster').mean())
```

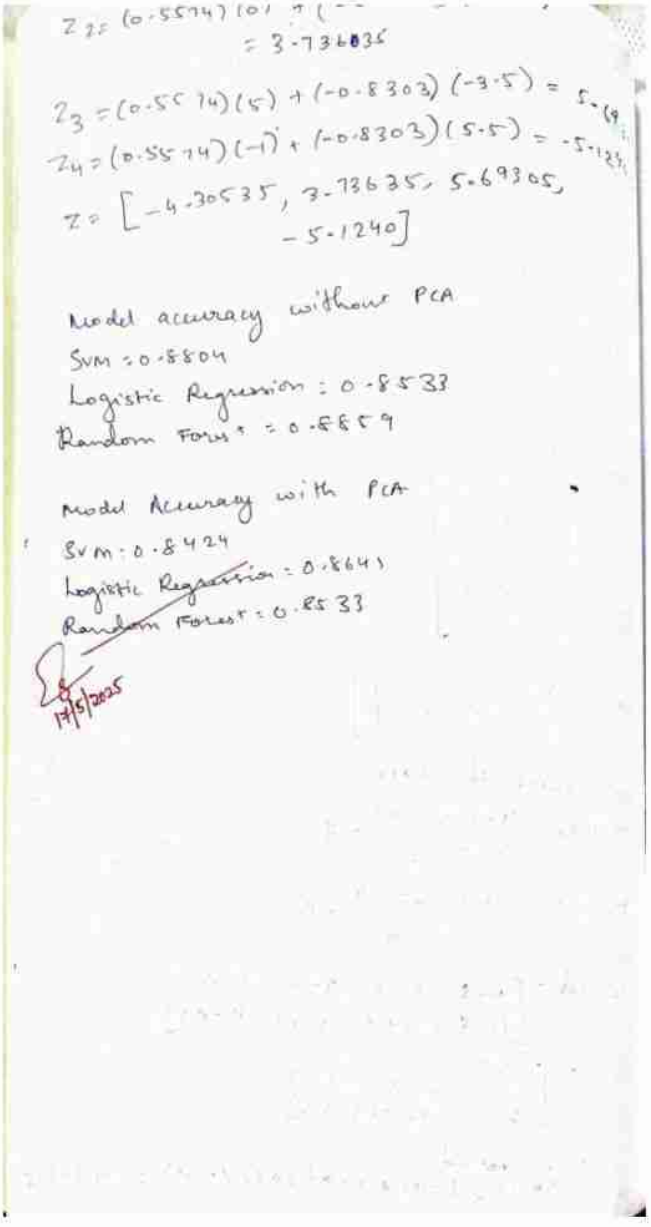


Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:





Code:

```
import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt


# 1. Load the Iris dataset:

data = pd.read_csv('/content/heart.csv')


# 2. Prepare the data:

X = data.iloc[:, :-1]  # Features (all columns except the
last)

y = data.iloc[:, -1]   # Target variable (last column)


# 3. Standardize the features (important for PCA):

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# 4. Apply PCA:

# Choose the number of components (n_components) you want to
keep

# For example, to reduce to 2 dimensions:
```

```
pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)



# 5. Create a new DataFrame with the reduced dimensions:

pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])

pca_df['Species'] = y  # Add the target variable back for
visualization



# 6. Visualize the reduced data:

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

plt.scatter(pca_df['PC1'], pca_df['PC2'],
c=pca_df['Species'].astype('category').cat.codes,
cmap='viridis')

plt.title('PCA of HeartDataset')

plt.xlabel('Principal Component 1 (PC1)')

plt.ylabel('Principal Component 2 (PC2)')

plt.show()



# 7. Explained variance ratio:

# This tells you how much variance is explained by each
principal component

explained_variance_ratio = pca.explained_variance_ratio_
```

```
print(f"Explained Variance Ratio: {explained_variance_ratio}")
```