VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

NAVNEETH K S (1BM22CS174)

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 **Navneeth K S (1BM22CS174),** 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.

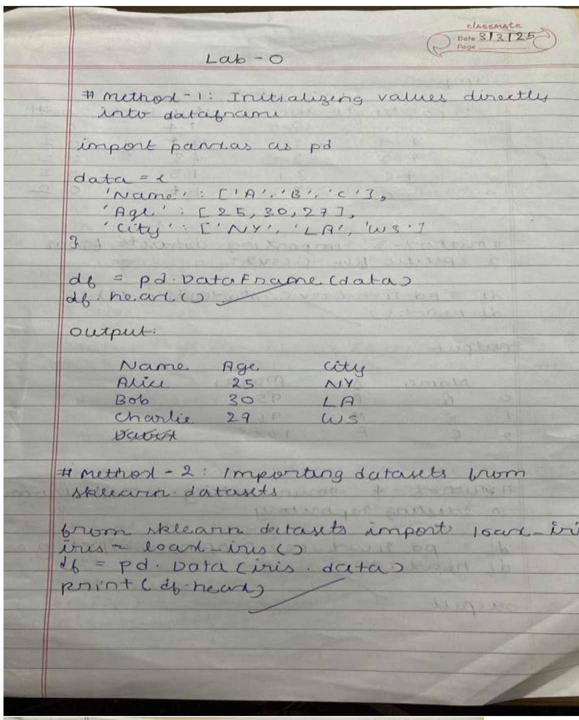
Lab Faculty Incharge Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

Program 1

Write a python program to import and export data using Panda's library functions

Screenshot



Lab - O

Date 3/3/25

method - 1 : Initializing values directly into dataframe

import pandas as pd

data = {
'Name' : ['A', 'B', 'C'],
'Age' : [25, 30, 27],
'City' : ['NY', 'LA', 'WS']
}

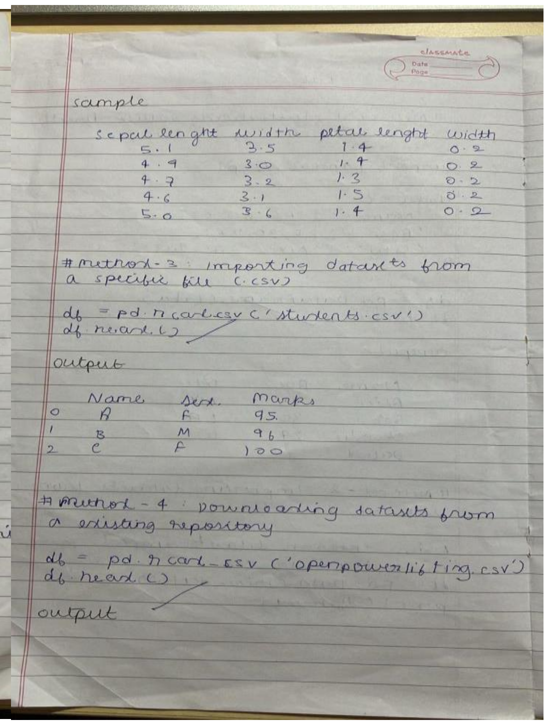
df = pd.DataFrame(data)
df.head()

output:

Name	Age	City
Alice	25	NY
Bob	30	LA
Charlie	27	WS
~~David~~		

method - 2 : Importing datasets from sklearn datasets

from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data)
print(df.head)



sample

sepal lenght	width	petal lenght	width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

method - 3 : importing datasets from a specific file (.csv)

df = pd.read_csv('students.csv')
df.head()

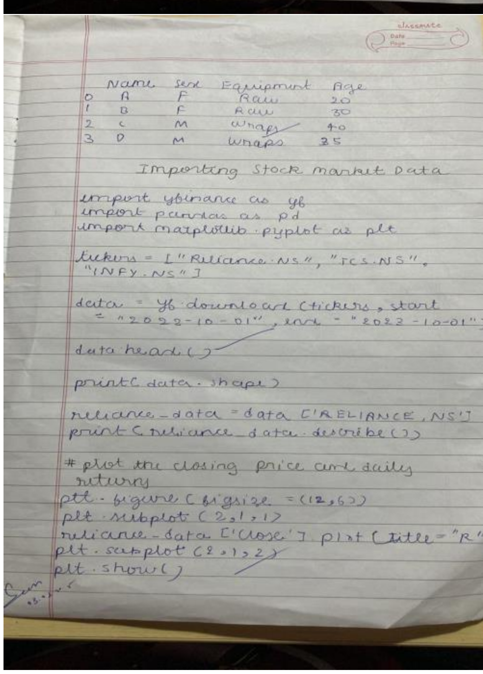
output

	Name	Sex	Marks
0	A	F	95
1	B	M	96
2	C	F	100

method - 4 : Downloading datasets from a existing repository

df = pd.read_csv('openpowerlifting.csv')
df.head()

output



	Name	Sex	Equipment	Age
0	A	F	Raw	20
1	B	F	Raw	30
2	C	M	Wraps	40
3	D	M	Wraps	35

Importing Stock market Data

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

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

data = yf.download(tickers, start = "2022-10-01", end = "2023-10-01")

data.head()

print(data.shape)

reliance_data = data['RELIANCE.NS']
print(reliance_data.describe())

# plot the closing price and daily returns
plt.figure(figsize = (12,6))
plt.subplot(2,1,1)
reliance_data['Close'].plot(title = "R"
plt.subplot(2,1,2)
plt.show()

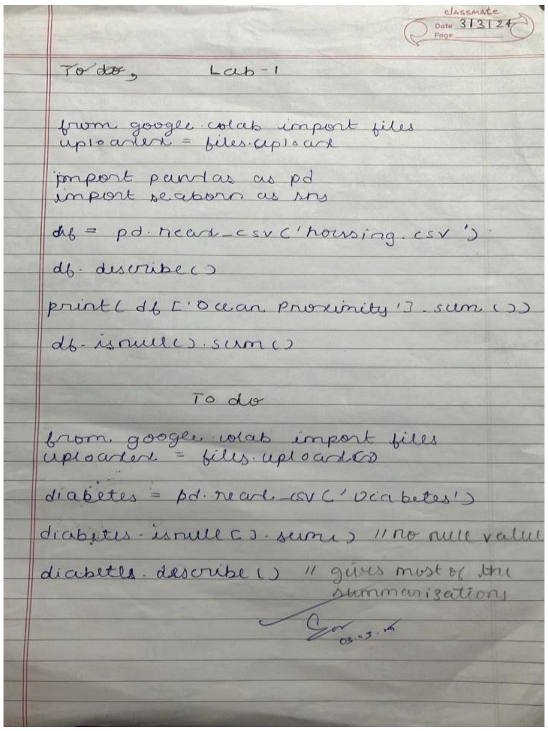
Code:

```
import pandas as pd
try:
    df = pd.read_csv('input.csv')
    print("Data imported successfully!\n")
    print(df)
except FileNotFoundError:
    print("The file 'input.csv' was not found.")
df["Processed"] = True
df.to_csv('output.csv', index=False)
print("\nData exported successfully to 'output.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshots


classmate
Date 3|3|24
Page

To do,   Lab-1

from google.colab import files
uploaded = files.upload

import pandas as pd
import seaborn as sns

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

df.describe()

print(df['Ocean Proximity'].sum())

df.isnull().sum()

To do

from google.colab import files
uploaded = files.upload()

diabetes = pd.read_csv('Diabetes')

diabetes.isnull().sum() //no null value

diabetes.describe() // gives most of the summarizations

03.03.24

Code

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler


data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', None],

    'Age': [25, 30, np.nan, 35, 29, 40],
```

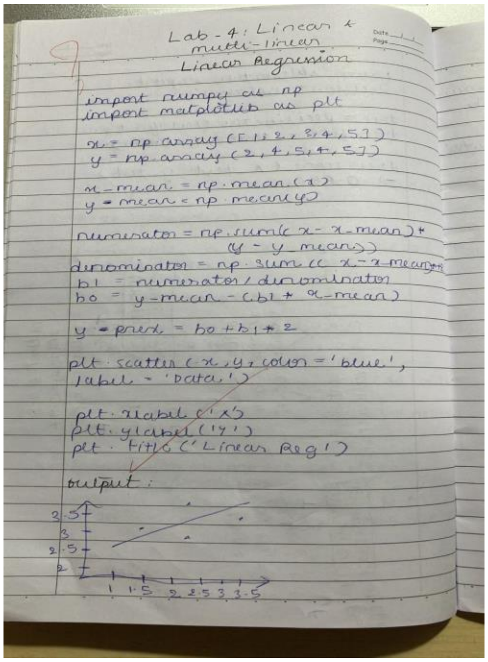
```
    'Department': ['HR', 'IT', 'Finance', 'IT', 'HR', 'Finance'],
    'Salary': [50000, 60000, 58000, 62000, np.nan, 52000]
}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Salary'].fillna(df['Salary'].median(), inplace=True)
df['Name'].fillna('Unknown', inplace=True)

le = LabelEncoder()
df['Department_Encoded'] = le.fit_transform(df['Department'])

df.drop_duplicates(inplace=True)

df.rename(columns={'Salary': 'Monthly_Salary'}, inplace=True)

df['Age'] = df['Age'].astype(int)

scaler = MinMaxScaler()
df['Salary_Normalized'] = scaler.fit_transform(df[['Monthly_Salary']])

standard_scaler = StandardScaler()
```

```
df['Age_Standardized'] = standard_scaler.fit_transform(df[['Age']])

print("\nPreprocessed DataFrame:\n", df)
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots



Code

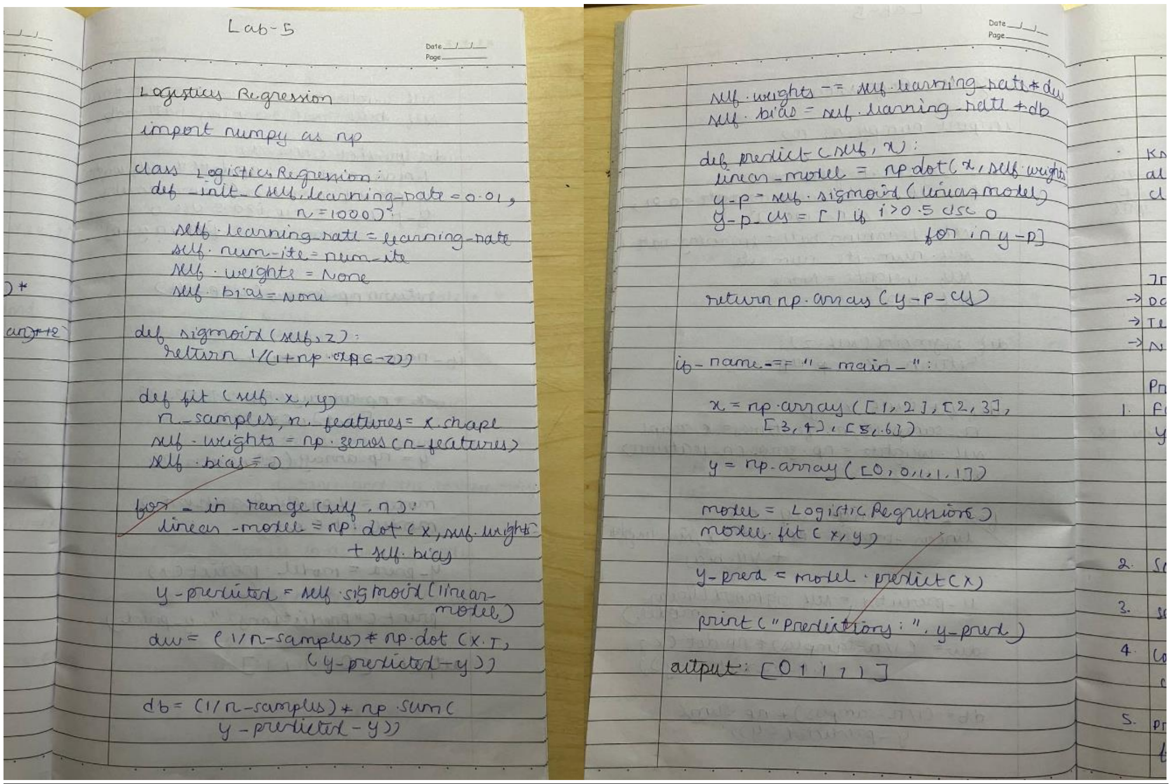
```
# Linear Regression

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

# Load dataset
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target

# Use only one feature for simple linear regression (e.g., RM = average number of rooms)
X = df[['RM']]
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
# Predict
y_pred = lr.predict(X_test)

# Output
print("Linear Regression Results")
print("Coefficients:", lr.coef_)
print("Intercept:", lr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Plot
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red')
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('House Price')
plt.title('Simple Linear Regression')
plt.show()

# Multiple Linear Regression

# Use all features
X = df.drop('PRICE', axis=1)
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
mlr = LinearRegression()
mlr.fit(X_train, y_train)

# Predict
y_pred = mlr.predict(X_test)

# Output
print("\nMultiple Linear Regression Results")
print("Coefficients:", mlr.coef_)
print("Intercept:", mlr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot's


```
Lab-5

Logistics Regression

import numpy as np

class LogisticsRegression:
    def _init_(self, learning_rate=0.01,
               n=1000):
        self.learning_rate = learning_rate
        self.num_ite = num_ite
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1/(1+np.exp(-z))

    def fit(self, x, y)
        n_samples, n_features = x.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

    for _ in range(self, n):
        linear_model = np.dot(x, self.weights)
                       + self.bias

        y_predicted = self.sigmoid(linear_
                                   model)
        dw = (1/n_samples) * np.dot(x.T,
                                    (y_predicted - y))

        db = (1/n_samples) * np.sum(
              y_predicted - y))

        self.weights -= self.learning_rate * dw
        self.bias = self.learning_rate * db

    def predict(self, x):
        linear_model = np.dot(x, self.weights)
        y_p = self.sigmoid(linear_model)
        y_p_cls = [1 if i>0.5 else 0
                   for i in y_p]

        return np.array(y_p_cls)

if _name_ == "_main_":

    x = np.array([[1,2],[2,3],
                  [3,4],[5,6]])

    y = np.array([0,0,1,1])

    model = LogisticRegression()
    model.fit(x, y)

    y_pred = model.predict(x)

    print("Predictions:", y_pred)
output: [0 1 1 1]
```

Code

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

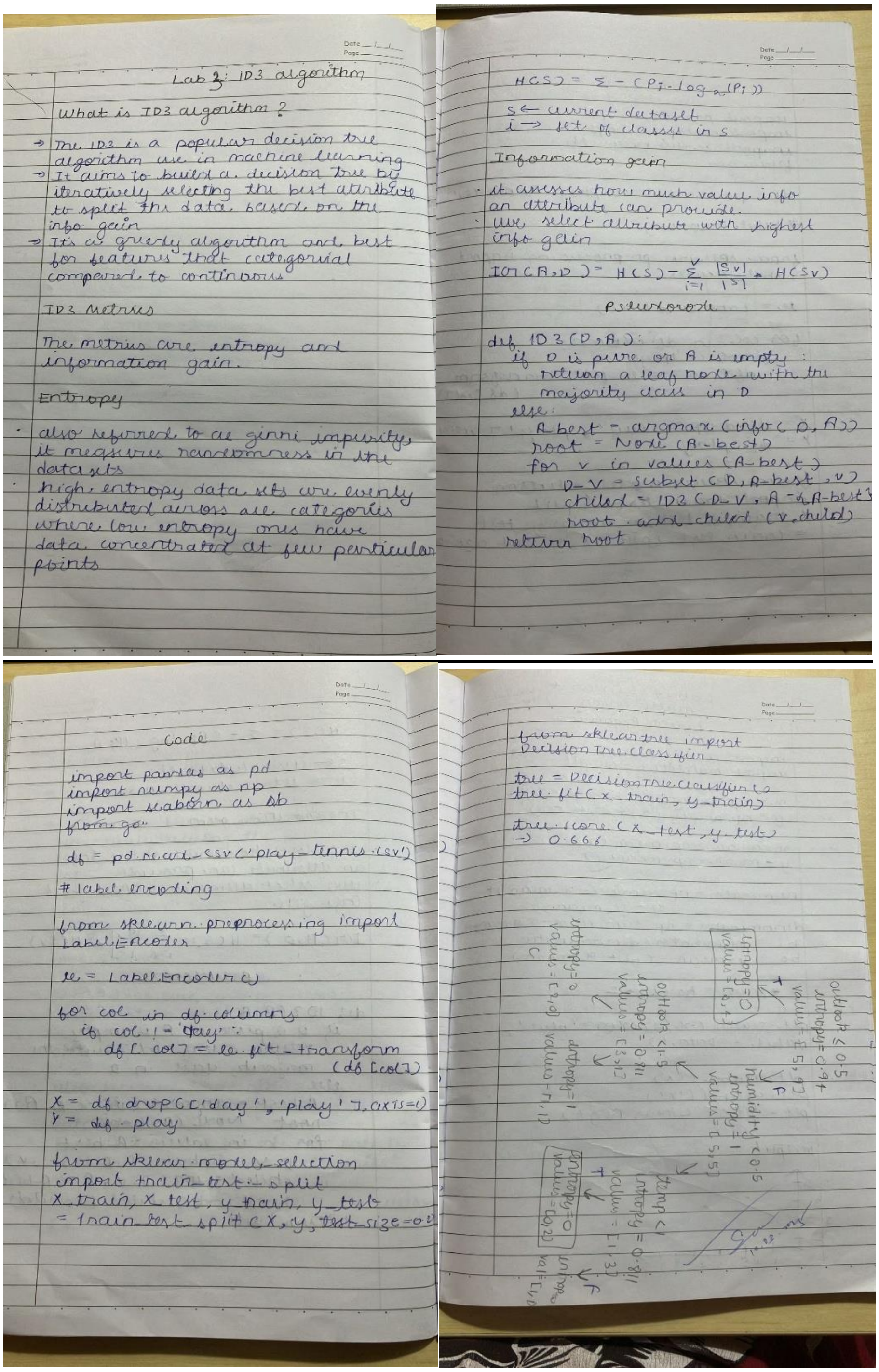
df['species'] = iris.target

```python
df_binary = df[df['species'] != 2]  # Remove class 2 (Virginica)

X = df_binary.iloc[:, :-1]   # Features
y = df_binary['species']     # Target (0 or 1)

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: Predict and evaluate
y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nAccuracy:", accuracy_score(y_test, y_pred))
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

Screenshots



Lab 3: ID3 algorithm

What is ID3 algorithm ?

→ The ID3 is a popular decision tree algorithm use in machine learning
→ It aims to build a decision tree by iteratively selecting the best attribute to split the data based on the info gain
→ It's a greedy algorithm and best for features that categorical compared to continuous

ID3 Metrics

The metrics are entropy and information gain.

Entropy

- also referred to as ginni impurity, it measures randomness in the data sets
- high entropy data sets are evenly distributed across all categories where low entropy ones have data concentrated at few particular points

$H(S) = \Sigma - (P_i \cdot \log_2 (P_i))$

S ← current dataset
i → set of classes in S

Information gain

- it assesses how much value info an attribute can provide.
- we select attribute with highest info gain

$IG(A, D) = H(S) - \sum_{i=1}^{v} \frac{|S_v|}{|S|} * H(S_v)$

Pseudocode

```
def ID3(D, A):
    if D is pure or A is empty:
        return a leaf node with the majority class in D
    else:
        A best = argmax(info(D, A))
        root = Node(A-best)
        for v in values(A-best)
            D-V = subset(D, A-best, v)
            child = ID3(D-V, A - {A-best}
            root.add_child(v, child)
    return root
```

Code

```
import pandas as pd
import numpy as np
import seaborn as sb
from go..

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

# label encoding

from sklearn.preprocessing import
LabelEncoder

le = LabelEncoder()

for col in df.columns
    if col != 'day':
        df[col] = le.fit_transform
                         (df[col])

X = df.drop(['day', 'play'], axis=1)
Y = df.play

from sklearn.model_selection
import train_test_split
X_train, X_test, y_train, y_test
= train_test_split(X, y, test_size=0.2

from sklearn.tree import
DecisionTreeClassifier

tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

tree.score(X_test, y_test)
→ 0.666
```

outlook ≤ 0.5
entropy = 0.94
values = [5, 9]

entropy = 0
values = [0, 4]

humidity < 0.5
entropy = 1
values = [5, 5]

outlook < 1.5
entropy = 0.811
values = [3, 1]

temp < 1
entropy = 0.811
values = [1, 3]

entropy = 0
values = [2, 0]

entropy = 1
values = [1, 1]

entropy = 0
values = [0, 2]

Code

import pandas as pd

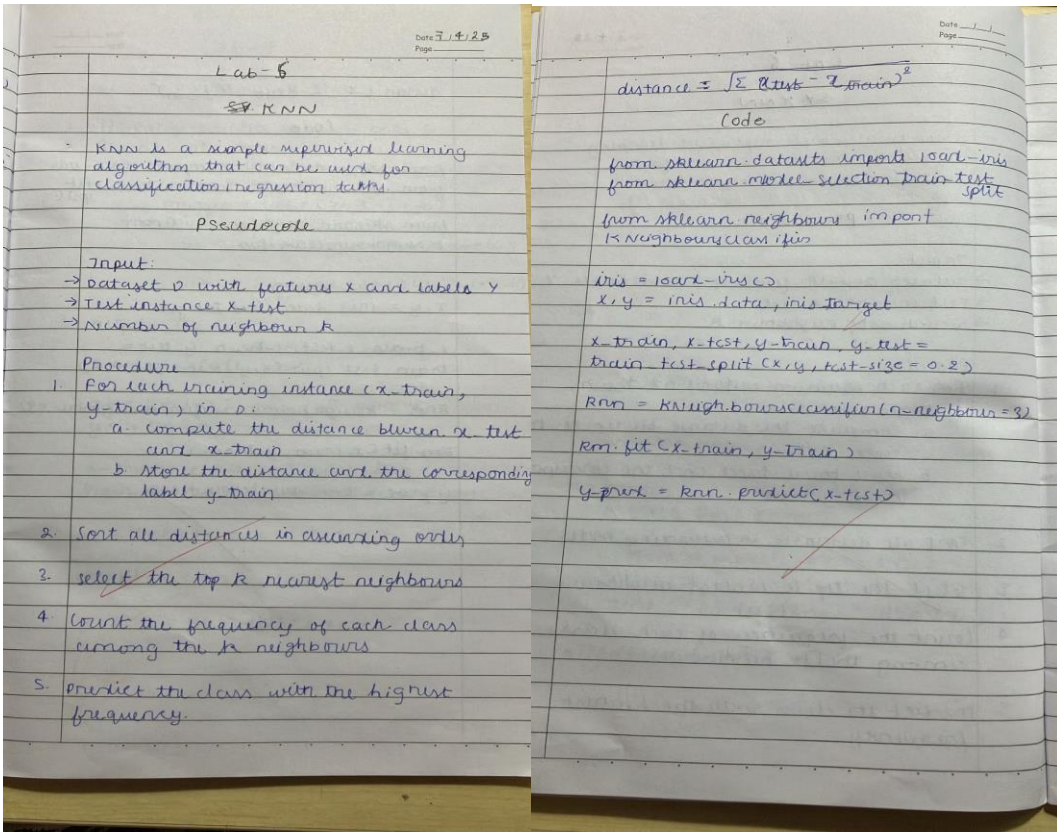
```
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast',
                'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',
                    'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',
                 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',
             'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
                   'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)
le = LabelEncoder()
for column in df.columns:
    df[column] = le.fit_transform(df[column])

# Step 3: Separate features and label
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']
clf = DecisionTreeClassifier(criterion='entropy')  # ID3 uses 'entropy'
```

```
clf = clf.fit(X, y)

print("\nDecision Tree Rules:")

tree_text = tree.export_text(clf, feature_names=X.columns.tolist())

print(tree_text)

# Example: Outlook=Rain, Temperature=Mild, Humidity=High, Wind=Weak

# Encode input sample with same label encoding order used earlier

sample = pd.DataFrame({

    'Outlook': [le.transform(['Rain'])[0]],

    'Temperature': [le.transform(['Mild'])[0]],

    'Humidity': [le.transform(['High'])[0]],

    'Wind': [le.transform(['Weak'])[0]]

})


# Predict

prediction = clf.predict(sample)

result = 'Yes' if prediction[0] == 1 else 'No'

print(f"\nPrediction for new sample (Rain, Mild, High, Weak): {result}")
```

Program 6

Build KNN Classification model for a given dataset

Screenshots



Date 7/4/25

Lab-5

KNN

• KNN is a simple supervised learning algorithm that can be used for classification/regression tasks.

Pseudocode

Input:

→ Dataset D with features X and labels Y

→ Test instance x_test

→ Number of neighbours k

Procedure

1. For each training instance (x_train, y_train) in D:
   a. compute the distance btwn x_test and x_train
   b. Store the distance and the corresponding label y_train
2. Sort all distances in ascending order
3. Select the top k nearest neighbours
4. Count the frequency of each class among the k neighbours
5. Predict the class with the highest frequency.

distance = $\sqrt{\sum (x_{test} - x_{train})^2}$

Code

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbours import KNeighboursClassifier

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

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

knn = KNeighboursClassifier(n_neighbours = 3)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

Code

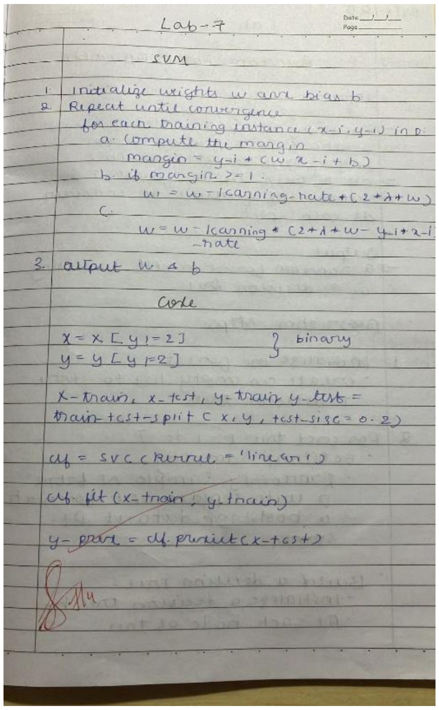
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 1: Load the Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
y = pd.Series(iris.target)

# Step 2: Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Build KNN model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Step 4: Predict and evaluate
y_pred = knn.predict(X_test)

# Results
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 7

Build Support vector machine model for a given dataset

Screenshots



Lab-7

SVM

1. initialize weights w and bias b
2. Repeat until convergence
   for each training instance (x-i, y-i) in D:
   a. compute the margin
      margin = y-i * (w. x-i + b)
   b. if margin >= 1:
      w = w - learning-rate * (2 * λ * w)
   c.
      w = w - learning-rate * (2 * λ * w - y-i * x-i)
3. output w & b

Code

```
x = x[y != 2]     } binary
y = y[y != 2]

X-train, x-test, y-train, y-test =
train-test-split(x, y, test-size = 0.2)

clf = SVC(kernel = 'linear')
clf.fit(x-train, y-train)
y-pred = clf.predict(x-test)
```

Code

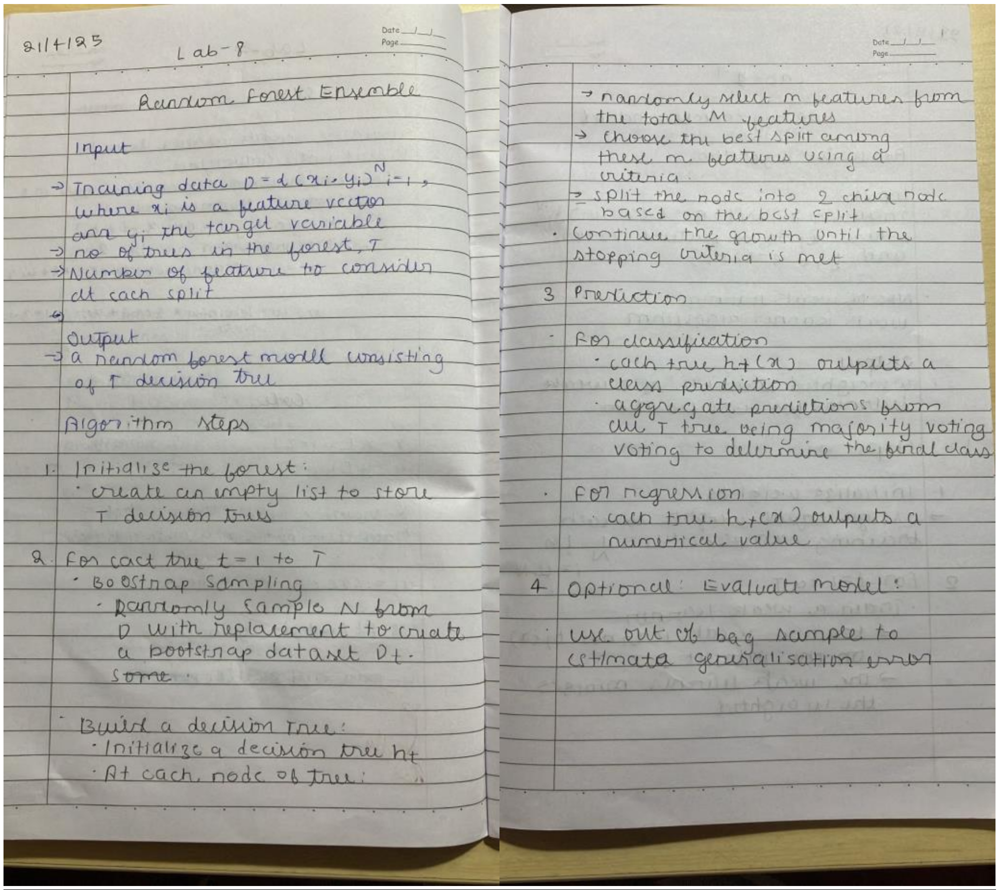
```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score


# Step 1: Load the Iris dataset

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)


# Step 2: Split the data into train and test sets (80% train, 20% test)
```

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


# Step 3: Build and train the SVM model (linear kernel)
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)


# Step 4: Predict and evaluate
y_pred = svm_model.predict(X_test)


# Output results
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 8

Implement Random Forest ensemble method on a given dataset

Screenshots



21/1/25 Lab-8

Random Forest Ensemble

Input

→ Training data $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i is a feature vector and y_i the target variable
→ no of trees in the forest, T
→ Number of feature to consider at each split

Output
→ a random forest model consisting of T decision tree

Algorithm steps

1. Initialise the forest:
   · create an empty list to store T decision trees
2. For each tree t = 1 to T
   · Bootstrap Sampling
   · Randomly sample N from D with replacement to create a bootstrap dataset D_t.
   ~~some~~
   · Build a decision tree:
   · Initialize a decision tree h_t
   · At each node of tree:
   → randomly select m features from the total M features
   → choose the best split among these m features using a criteria
   → split the node into 2 child node based on the best split
   · continue the growth until the stopping criteria is met
3. Prediction
   · For classification
   · each tree $h_t(x)$ outputs a class prediction
   · aggregate predictions from all T tree using majority voting voting to determine the final class
   · For regression
   · each tree $h_t(x)$ outputs a numerical value
4. Optional: Evaluate model:
   · use out of bag sample to estimate generalisation error

Code

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Step 1: Load Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
```

```
# Step 2: Split into train and test sets (80% train, 20% test)

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


# Step 3: Train Random Forest model

rf = RandomForestClassifier(n_estimators=100, random_state=42)  # 100 trees

rf.fit(X_train, y_train)


# Step 4: Predict and evaluate

y_pred = rf.predict(X_test)


print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

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

print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshots

~~24~~ 21/5/25

Lab-~~8~~ 9

Boosting ensemble method

AdaBoost Algorithm

Input
· Training dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ where x_i is a feature vector and $y_i \in \{-1, +1\}$

· No of weak learners
· weak learner algorithm

Output
· a weighted ensemble of T weak learners for classification

Steps:

1. Initialize weights:
→ assign equal weights to each training sample: $w_i = \frac{1}{N}$ for $i = 1, 2, \dots N$

2. For $t = 1$ to T
· Train a weak learner
→ train a weak learner $h_t(x)$ on the dataset D
→ the weak learner minimizes the weighted

· Compute weak learner weight

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

· update sample weights:

$$w_i \leftarrow w_i * \exp(-\alpha_t \cdot y_i \cdot h_t(x_i))$$

$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^{N} w_j}$$

3. Prediction

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right)$$

output the class lable: -1 or $+1$

Code

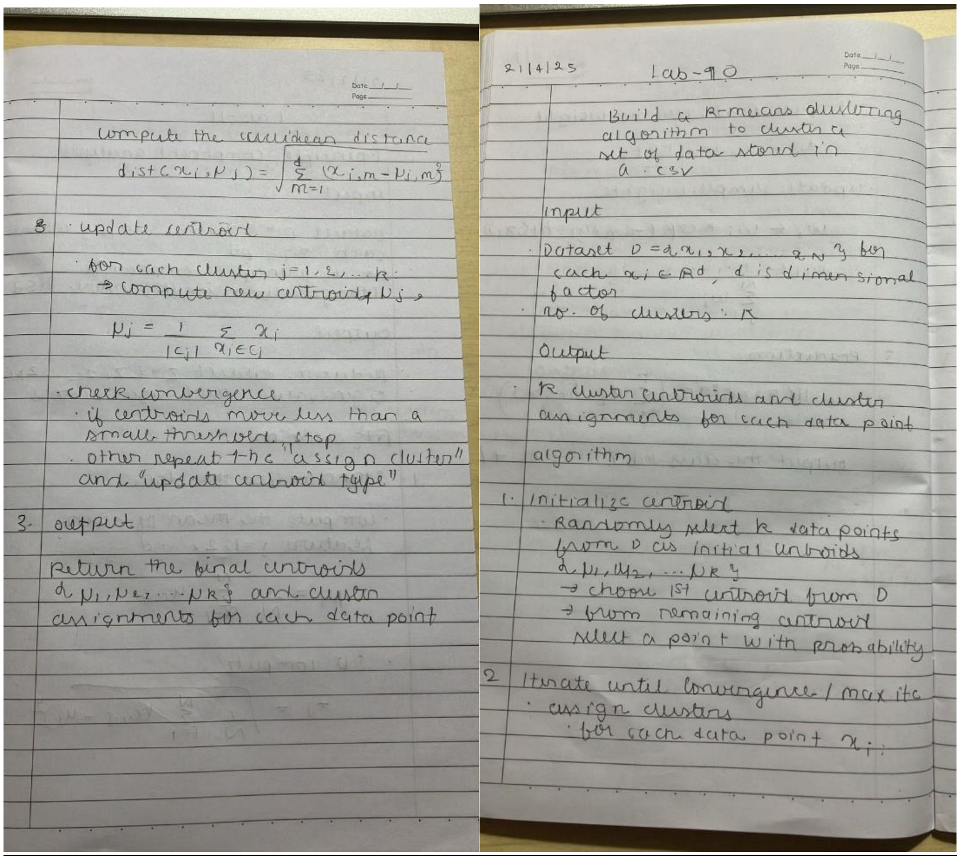
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Step 1: Load the Iris dataset
iris = load_iris()
```

```
X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)


# Step 2: Split into train and test sets (80% train, 20% test)

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


# Step 3: Build AdaBoost model with DecisionTreeClassifier as base estimator

base_estimator = DecisionTreeClassifier(max_depth=1)

model = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, learning_rate=1.0,
random_state=42)

model.fit(X_train, y_train)


# Step 5: Predict and evaluate

y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

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

print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 10

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

Screenshots


compute the euclidean distance

$$dist(x_i, \mu_j) = \sqrt{\sum_{m=1}^{d} (x_{i,m} - \mu_{j,m})^2}$$

8 · update centroid

· for each cluster $j = 1, 2, \ldots k$:

→ compute new centroid μ_j,

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

· check convergence

· if centroids move less than a small threshold, stop

· other repeat the "assign cluster" and "update centroid type"

3. output

Return the final centroids $\{\mu_1, \mu_2, \ldots \mu_k\}$ and cluster assignments for each data point

21/4/25 lab-10

Build a k-means clustering algorithm to cluster a set of data stored in a .csv

input

· Dataset $D = \{x_1, x_2, \ldots x_N\}$ for each $x_i \in R^d$, d is dimensional factor

· no. of clusters: k

Output

: k cluster centroids and cluster assignments for each data point

algorithm

1. initialize centroid

· Randomly select k data points from D as initial centroids $\{\mu_1, \mu_2, \ldots \mu_k\}$

→ choose 1st centroid from D

→ from remaining centroid select a point with probability

2 Iterate until convergence / max itr

· assign clusters

· for each data point x_i:

Code

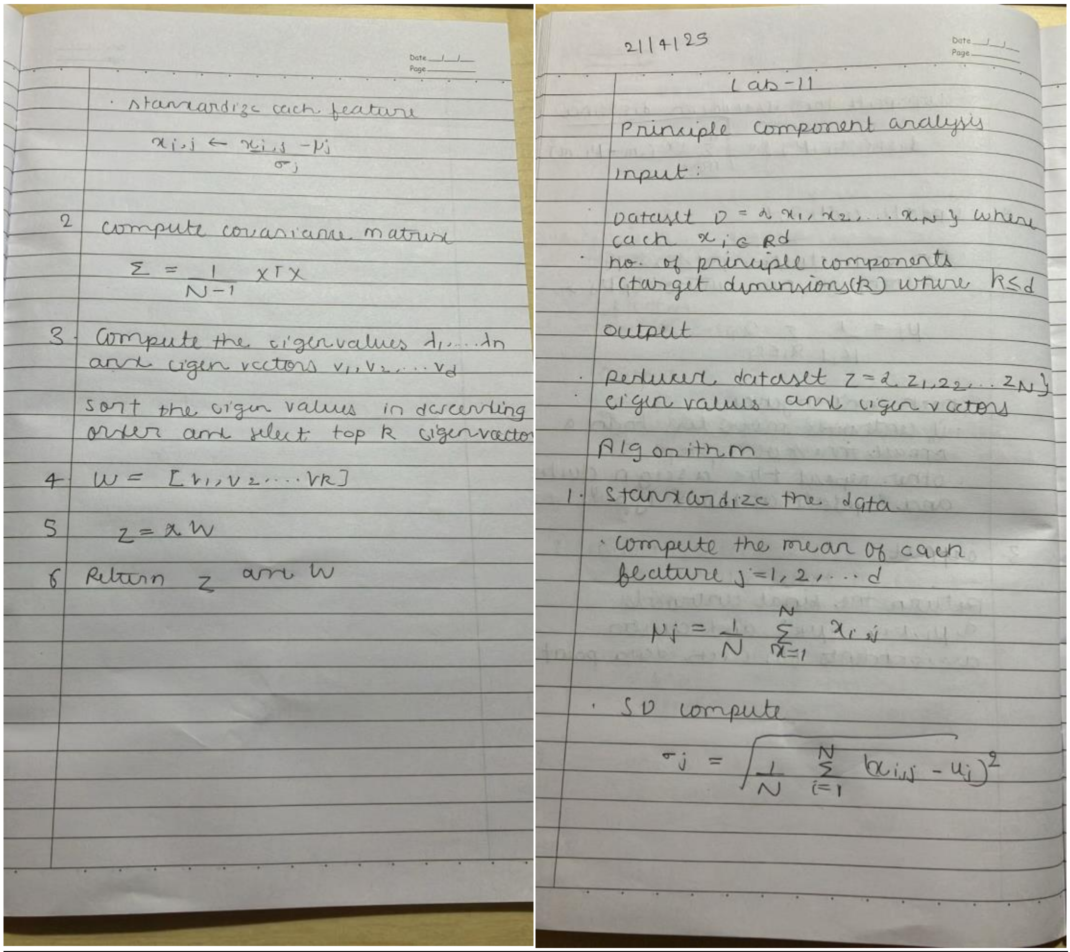
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler


# Step 1: Load dataset from CSV
df = pd.read_csv('your_dataset.csv')  # Replace with your file path


# Optional: View first few rows
print("Data Preview:\n", df.head())
```

```
# Step 2: Select relevant numeric columns for clustering
# You can specify specific columns like: df[['column1', 'column2']]
X = df.select_dtypes(include='number')

# Step 3: Scale the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: Apply K-Means clustering (e.g., 3 clusters)
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Step 5: Print cluster centers
print("Cluster Centers:\n", kmeans.cluster_centers_)

# Optional Step 6: Visualize (works well for 2D or PCA-reduced data)
if X.shape[1] >= 2:
    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis')
    plt.title("K-Means Clustering")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshots



21/4/25

Lab-11

Principle component analysis

Input:

Dataset $D = \{x_1, x_2, \ldots x_N\}$ where each $x_i \in R^d$

no. of principle components (target dimensions(k)) where $k \le d$

Output

Reduced dataset $Z = \{z_1, z_2, \ldots z_N\}$

eigen values and eigen vectors

Algorithm

1. Standardize the data

· compute the mean of each feature $j = 1, 2, \ldots d$

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

· SD compute

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2}$$

· standardize each feature

$$x_{i,j} \leftarrow \frac{x_{i,j} - \mu_j}{\sigma_j}$$

2 compute covariance matrix

$$\Sigma = \frac{1}{N-1} X^T X$$

3. Compute the eigenvalues $\lambda_1, \ldots \lambda_n$ and eigen vectors $v_1, v_2, \ldots v_d$

sort the eigen values in descending order and select top k eigenvectors

4 $W = [v_1, v_2 \ldots v_k]$

5 $Z = xW$

6 Return Z and W

Code

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt


# Step 1: Load the Iris dataset

iris = load_iris()

X = iris.data
```

```
y = iris.target
feature_names = iris.feature_names

# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA (reduce to 2 components for visualization)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Plot the 2D PCA result
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=60)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA - Iris Dataset")
plt.legend(handles=scatter.legend_elements()[0], labels=iris.target_names)
plt.grid(True)
plt.show()

# Explained variance
print("Explained variance ratio:", pca.explained_variance_ratio_)
```