VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Sanketh M Hanasi (1BM22CS242)

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 **Sanketh M Hanasi (1BM22CS242),** 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. Saritha A N** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

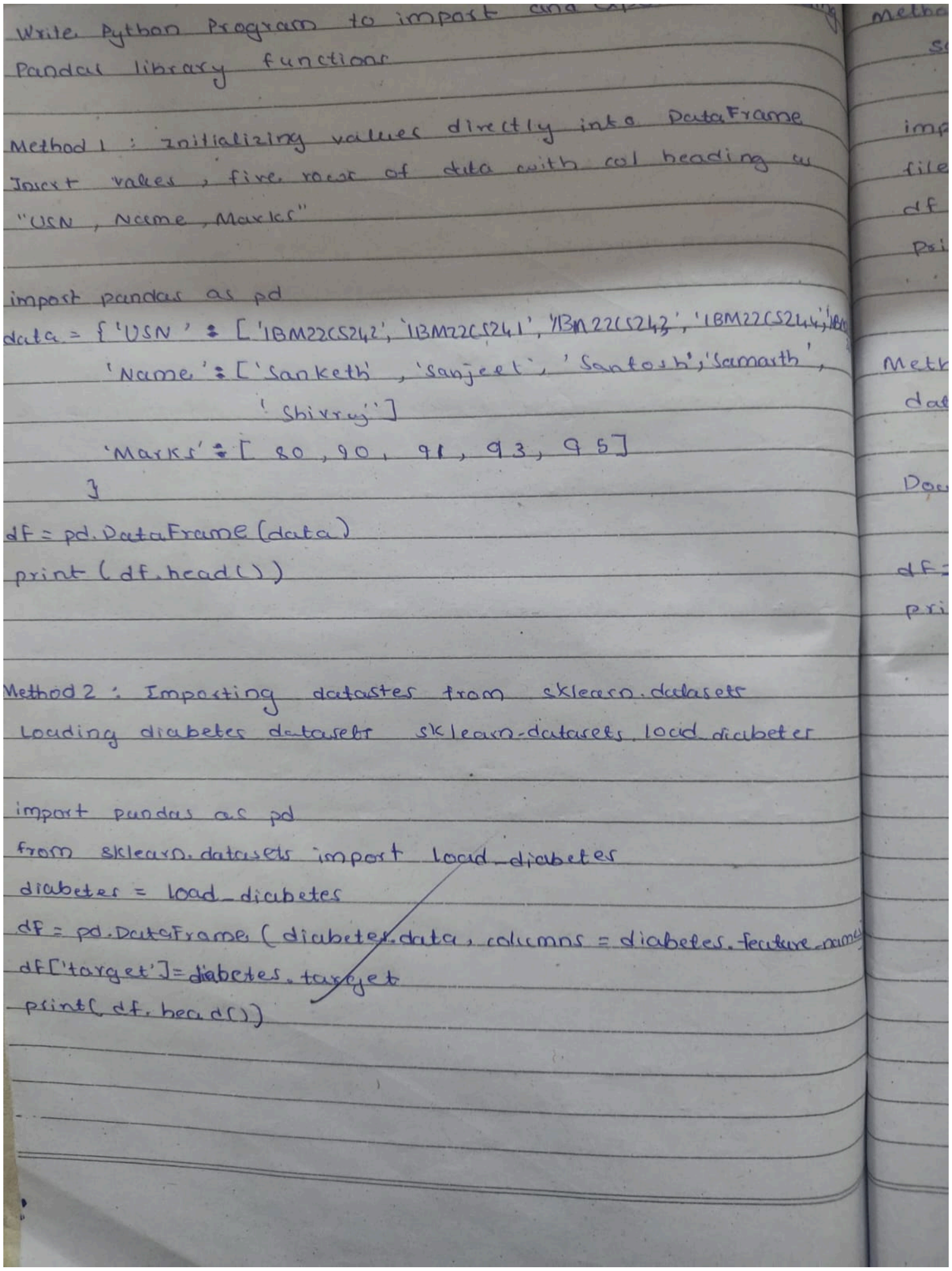
Github Link:

https://github.com/SankethHanasi/6thSem-ML-Lab/tree/main

Program 1

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

Screenshot:



Write Python Program to import and export using Pandas library functions

Method 1 : Initializing values directly into DataFrame
Insert values , five rows of data with col heading as "USN , Name , Marks"

```
import pandas as pd
data = {'USN' : ['1BM22CS242', '1BM22CS241', '1BM22CS243', '1BM22CS244', '1BM
        'Name' : ['Sanketh' , 'sanjeet', 'Santosh', 'Samarth',
                  'Shivraj']
        'Marks' : [ 80 , 90 , 91 , 93 , 95 ]
      }
df = pd.DataFrame(data)
print (df.head())
```

Method 2 : Importing datasets from sklearn.datasets
Loading diabetes dataset sklearn.datasets.load_diabeter

```
import pandas as pd
from sklearn.datasets import load_diabeter
diabeter = load_diabetes
df = pd.DataFrame ( diabetes.data , columns = diabetes.feature_names
df['target'] = diabetes.target
print( df.head())
```

ing

Method 3 : Importing datasets from a specific csv file sample-sales-data.csv

```
import pandas as pd
file path = 'sample-sales-data.csv'
df = pd.read_csv (file_path)
print(df.head())
```

Method 4 : Downloading datasets from existing dataset repositories like Mendely

Download diabetes datasets from Mendely

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

names]

Code:

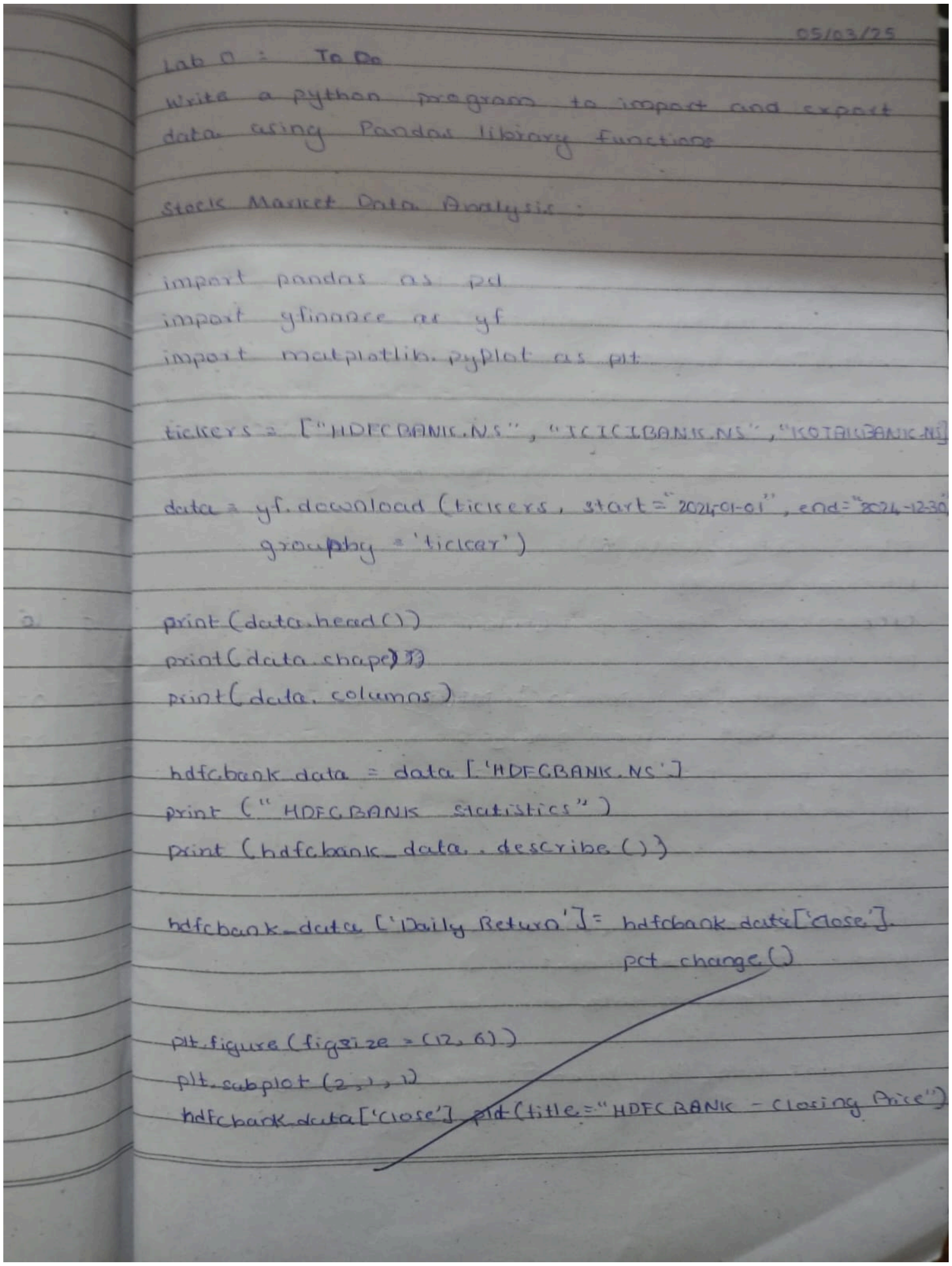
```
#Method 1 : Initializing values directly into dataframe
import pandas as pd
    data={
  'USN':['1BM22CS242','1BM22CS243','1BM22CS244','1BM22CS245','1BM22CS246'],
  'Name':['Alice','Bob','Claire','Sanketh','Samarth'],
  'Marks':[90,78,76,89,91]
}
df=pd.DataFrame(data)
print(df)

#Method 2: Importing Datasets from sklearn.datasets
from sklearn.datasets import load_diabetes
diabetes=load_diabetes()
df1=pd.DataFrame(diabetes.data,columns=diabetes.feature_names)
df1['target']=diabetes.target
print(df1.head())

#Method 3: Importing Datasets From a .csv file
df2=pd.read_csv('/content/sample_sales_data.csv')
print(df2.head())

#Method 4: Downloading Datasets from Mendely
df3=pd.read_csv('/content/Dataset of Diabetes .csv')
print(df3.head())

# Stock Market Data Analysis - ToDo
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by ='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)

HDFCBANK_data = data['HDFCBANK.NS']
```

```
print("\nSummary statistics for HDFCBANK :")
print(HDFCBANK_data.describe())
HDFCBANK_data['Daily Return'] = HDFCBANK_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
HDFCBANK_data['Close'].plot(title="HDFCBANK_data - Closing Price")
plt.subplot(2, 1, 2)
HDFCBANK_data['Daily Return'].plot(title="HDFCBANK_data - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

ICICIBANK_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICIBANK :")
print(ICICIBANK_data.describe())
ICICIBANK_data['Daily Return'] = ICICIBANK_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
ICICIBANK_data['Close'].plot(title="ICICIBANK_data  - Closing Price")
plt.subplot(2, 1, 2)
ICICIBANK_data['Daily Return'].plot(title="ICICIBANK_data - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

KOTAKBANK_data = data['KOTAKBANK.NS']
print("\nSummary statistics for KOTAKBANK :")
print(KOTAKBANK_data.describe())
KOTAKBANK_data['Daily Return'] = KOTAKBANK_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
KOTAKBANK_data['Close'].plot(title="KOTAKBANK_data  - Closing Price")
plt.subplot(2, 1, 2)
KOTAKBANK_data['Daily Return'].plot(title="KOTAKBANK_data - Daily Returns",
color='orange')
plt.tight_layout()
plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshots :



05/03/25

Lab 0 : To Do

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

Stock Market Data Analysis :

```
import pandas as pd
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",
        groupby = 'ticker')

print(data.head())
print(data.shape())
print(data.columns)

hdfcbank_data = data['HDFCBANK.NS']
print ("HDFCBANK Statistics")
print (hdfcbank_data.describe())

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

plt.figure(figsize = (12, 6))
plt.subplot (2,1,1)
hdfcbank_data['Close'].plt(title="HDFCBANK - Closing Price")
```

```
plt.subplot(2,1,2)
hdfcbank_data['Daily Return'].plot(title="HDFC BANK -
Daily Returns", color='Orange')
plt.tight_layout()
plt.show()

icicibank_data = data['ICICIBANK.NS']
print("ICICI BANK Statistics")
print(" icicibank_data.describe())

icicibank_data['Daily Returns'] = icicibank_data['Close'].
                                   pct_change()

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


kotakbank_data = data['KOTAKBANK.NS']
print("KOTAKBANK Statistics")
print( kotakbank_data.describe())

kotakbank_data['Daily Returns'] = kotakbank_data['Close']
                                   pct_change()
```

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

05.03.25

Code :

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats

df = pd.read_csv('/content/Dataset of Diabetes .csv')
df.head(10)

df['Gender']=df['Gender'].replace(to_replace='f', value='F')
d=df.groupby('Gender')['Gender'].count()
print(d)

df['CLASS']=df['CLASS'].replace(to_replace=['N ','Y '], value=['N','Y'])
d1=df.groupby('CLASS')['CLASS'].count()
print(d1)
print(df.describe())

#Code to Find Missing Values
missing_values = df.isnull().sum()
print(missing_values)

#Handling Categorical Attributes
label_encoder=LabelEncoder()
df_copy['Gender']=label_encoder.fit_transform(df_copy['Gender'])
df_copy['CLASS']=label_encoder.fit_transform(df_copy['CLASS'])
df_copy.head()
df=df_copy
nc=df.select_dtypes(include=['float64','int64']).columns
imputer=SimpleImputer(strategy='mean')
df[nc]=imputer.fit_transform(df[nc])
all_cols = df.columns
imputer = SimpleImputer(strategy='most_frequent')
df[all_cols] = imputer.fit_transform(df[all_cols])


Q1=df[nc].quantile(0.25)
Q3=df[nc].quantile(0.75)
IQR=Q3-Q1
df_clean=df[~((df[nc]<(Q1-1.5*IQR))|(df[nc]>(Q3+1.5*IQR))).any(axis=1)]
```

```
scaler_choice='minmax'
scaler=MinMaxScaler()
df_scaled=pd.DataFrame(scaler.fit_transform(df_clean[nc]),columns=nc)
df_scaled.head()
df_find=pd.concat([df_clean[cat_c],df_scaled],axis=1)
df_find.head()
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots :

LAB-3 Build Logistic Regression Model 19/03/24

1) Consider a binary classification prob where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been trained & the learned parameter are $a_0 = -5$ (intercept) & $a_1 = 0.8$ (coefficient for study hrs)

a- write the logistic regression eq for this problem

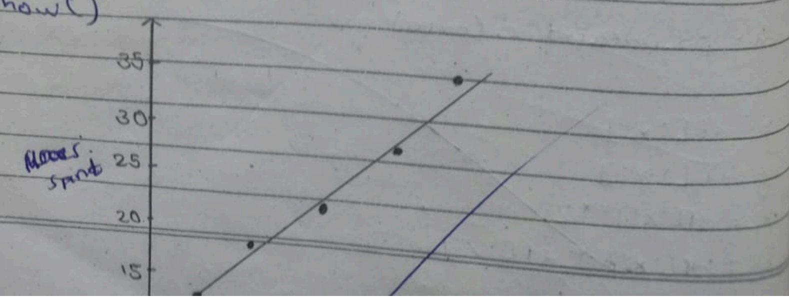
b- Cal the probability that a student who studies for 7hrs will pass

c- Determine the predicted class (pass or fail) for this student based on a threshold of 0.5

1) Linear Regression

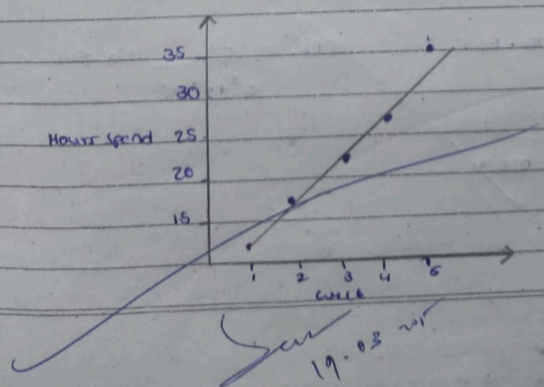
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = {
        'x': [1, 2, 3, 4, 5],
        'y': [12, 18, 22, 28, 35]
    }
df = pd.DataFrame(data)

x = df['x']
y = df['y']
x2 = df['x']**2
xy = df['x'] * df['y']
```

```
x_mean = np.mean(x)
y_mean = np.mean(y)
x2_mean = np.mean(x2)
xy_mean = np.mean(xy)

numerator = xy_mean - (x_mean * y_mean)
denominator = x2_mean - (x_mean **2)

b1 = numerator / denominator
b0 = y_mean - b1 * x_mean

print ("b1:", b1)   # b1 : 5.6000
print ("b0:", b0)   # b2 : 6A9

y5 = b0 + b1*5
y7 = b0 + b1*7
print ("week 5 :" y5)   # week 5 : 34.2
print ("week 7 :" y7)   # week 7 : 45.4

plt.scatter (x, y, label = 'Data Points')
plt.plot (x, b0 + b1 * x, color = 'red', label = 'Regression li
plt.xlabel ('x')
pt.ylabel ('y')
plt.title ('Regression')
plt.show()
```



Linear Regression using Matrix

```
x = np.column_stack ((np.ones (len(x)), x))
beta = np.linalg.inv (x.T @ x) @ x.T @ y

b0 = beta[0]
b1 = beta[1]

print ("b0 : ", b0)        # b0 : 8.60001
print ("b1 : ", b1)        # b1 : 6.20 0021

y5 = b0 + (b1 * 5)
y7 = b0 + (b1 * 7)
print ("week5 : " y5)      # week5 : 34.2
print ("week7 : " y7)      # week7 : 45.4

plt.scatter (x, y, label = "Data Points")
plt.plot (x, b0 + b1 * x , color = 'red', label = 'Regression line')
plt.xlabel ('x')
plt.ylabel ('Y')
plt.title ('Linear Regression using matrix')
plt.show()
```



19.03.25

Code :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Given data
xi = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  # Reshape for sklearn
yi = np.array([1.2, 1.8, 2.6, 3.2, 3.8])

# Train the linear regression model
model = LinearRegression()
model.fit(xi, yi)

# Get the slope (m) and intercept (c)
m = model.coef_[0]  # Slope
c = model.intercept_  # Intercept

# Predict sales for weeks 7 and 9
future_weeks = np.array([7, 9]).reshape(-1, 1)
predicted_sales = model.predict(future_weeks)

# Generate line for visualization
x_range = np.arange(1, 10, 0.1).reshape(-1, 1)
y_range = model.predict(x_range)

# Plot data points and regression line
plt.scatter(xi, yi, color='blue', label="Actual Sales")
plt.plot(x_range, y_range, color='red', label=f"Regression Line: y = {m:.2f}x + {c:.2f}")
plt.scatter(future_weeks, predicted_sales, color='green', marker='o', label="Predicted Sales (Weeks 7 & 9)")

# Labels and title
plt.xlabel("Weeks")
plt.ylabel("Sales")
plt.title("Weekly Sales Prediction using Linear Regression")
plt.legend()
plt.grid(True)

# Show plot
plt.show()
# Print equation and predicted sales
```

```
print(f"Equation of the regression line: y = {m:.2f}x + {c:.2f}")
print(f"Predicted sales for week 7: {predicted_sales[0]:.2f}")
print(f"Predicted sales for week 9: {predicted_sales[1]:.2f}")

#LINEAR REGRESSION IN MATRIX FORM
import numpy as np
import matplotlib.pyplot as plt

# Given data
xi = np.array([1, 2, 3, 4])  # Independent variable (weeks)
yi = np.array([1,3,4,8])  # Dependent variable (sales)

# Convert to matrix form: Add a column of ones for the intercept term
X = np.c_[np.ones(len(xi)), xi]  # Shape: (5,2), first column is 1 for intercept
Y = yi.reshape(-1, 1)  # Shape: (5,1)

# Compute theta using the Normal Equation: θ = (X^T X)^(-1) X^T Y
theta = np.linalg.inv(X.T @ X) @ X.T @ Y  # Matrix multiplication

# Extract slope (m) and intercept (c)
c, m = theta.flatten()  # Convert matrix to scalars

# Predict sales for weeks 7 and 9
future_weeks = np.array([1, 7, 9])  # Include 1 for intercept calculation
X_future = np.c_[np.ones(len(future_weeks)), future_weeks]
predicted_sales = X_future @ theta  # Matrix multiplication

# Generate regression line
x_range = np.linspace(1, 10, 100)  # Smooth range for plotting
y_range = c + m * x_range  # Compute y values

# Plot data points and regression line
plt.scatter(xi, yi, color='blue', label="Actual Sales")
plt.plot(x_range, y_range, color='red', label=f"Regression Line: y = {m:.2f}x + {c:.2f}")
plt.scatter([7, 9], predicted_sales[1:], color='green', marker='o', label="Predicted Sales (Weeks 7 &
9)")

# Labels and title
plt.xlabel("Weeks")
plt.ylabel("Sales")
plt.title("Weekly Sales Prediction using Matrix Approach")
plt.legend()
```

```python
plt.grid(True)

# Show plot
plt.show()

# Print equation and predicted sales
print(f"Equation of the regression line: y = {m:.2f}x + {c:.2f}")
print(f"Predicted sales for week 7: {predicted_sales[1][0]:.2f}")
print(f"Predicted sales for week 9: {predicted_sales[2][0]:.2f}")
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshots :

02/04/25 LAB-4

Consider binary classification problem where we want to predict whether a student will pass or fail based on their study hrs. The logistic regression model has been trained & the learned parameters are $a_0 = -5$, $a_1 = 0.8$.

a) Write the logistic Regression equation for this problem

$$p(y=1|x) = \frac{1}{1 + e^{-(-5+0.8x)}}$$

b) Calculate the probability that a student studies for 7 hrs will pass

$$p(\text{pass}) = \frac{1}{1 + e^{-0.6}} = 0.6479$$

c) Determine the predicted class for this student based on threshold 0.5. → if $p(\text{pass}) \geq 0.5$ → student will pass otherwise he will fail

d) Consider $z = [2,1,0]$ for three class. Apply softmax function to find the probability values of three classes

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{i=0}^{k} e^{z_i}}$$

$$p(1) = \frac{e^2}{e^2 + e^1 + e^0} = 0.665$$

$$p(2) = \frac{e^1}{e^2 + e^1 + e^0} = 0.245$$

$$p(3) = \frac{e^0}{e^2 + e^1 + e^0} = 0.09$$

Q. Dataset file : "HR_comma_sep.csv"

i) Which vars did you identify as having a direct & clear impact on emp retention & why?

-> Statisfactory level, time spent in company, no. of proj, salary

These variables are choosen based on the trends in data visulatization

ii) What was the accuracy of your logistic regression model? Do you think this is a good accuracy? Why or why not?

-> The accuracy of logistic regression was 78%. This accuracy is fairly good. It suggests that model captures most of properties affecting employee retention.

2) For zoo dataset

i) Did you perform any data preprocessing steps? If yes, what were they and why were they necessary?

-> Yes, -> Dropped 'animal_name' column, checked for missing values, converted categorical variable if needed

ii) Were there any missing or inconsistent values in the dataset? How did you handle them?

-> No missing values were found in the dataset. If they were inconsistent/present, we could use mean/mode methods to correct them.

iii) What does the confusion matrix tell you about the performance of your model?

-> It shows how well the model predicted different class types. Animals.

iv) Most frequently misclassified class : some class types are misclassified. This happend due to similarities in features b/w different animal classes.

Code :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

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

# Exploratory Data Analysis
# Plot bar chart showing impact of salaries on retention
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='salary', hue='left')
plt.title("Impact of Salary on Employee Retention")
plt.xlabel("Salary Level")
plt.ylabel("Number of Employees")
plt.legend(["Stayed", "Left"])
plt.show()

# Plot bar chart showing correlation between department and employee retention
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='Department', hue='left')
plt.title("Department-wise Employee Retention")
plt.xlabel("Department")
plt.ylabel("Number of Employees")
plt.xticks(rotation=45)
plt.legend(["Stayed", "Left"])
plt.show()

# Selecting key features based on analysis
X = df[['satisfaction_level', 'time_spend_company', 'Work_accident', 'salary']]
X = pd.get_dummies(X, columns=['salary'], drop_first=True)  # Convert categorical 'salary' to
numerical
y = df['left']

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.9, random_state=10)
```

```
# Train logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_predicted = model.predict(X_test)

# Model accuracy
accuracy = accuracy_score(y_test, y_predicted)
print(f"Model Accuracy: {accuracy:.4f}")

# Plotting actual vs predicted values
plt.figure(figsize=(6, 4))
sns.heatmap(pd.crosstab(y_test, y_predicted), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Predict probability of an employee leaving
def predict_leave_probability(features):
    return model.predict_proba([features])[0][1]  # Probability of leaving

# Example prediction
example_employee = [[0.5, 3, 0, 1, 0]]  # Sample feature values with one-hot encoded salary
probability = predict_leave_probability(example_employee[0])
print(f"Probability of leaving: {probability:.4f}")

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay

# Load dataset
file_path = "/content/zoo-data.csv"
df = pd.read_csv(file_path)

# Selecting features and target variable
X = df.drop(columns=['animal_name', 'class_type'])  # Drop non-numeric column and target
y = df['class_type']  # Target variable
```

```python
# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model for multiclass classification
model = LogisticRegression(multi_class='multinomial', max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_predicted = model.predict(X_test)

# Model accuracy
accuracy = accuracy_score(y_test, y_predicted)
print(f"Model Accuracy: {accuracy:.4f}")

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_predicted)
display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
display.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()

# Predict probability of class type for a sample animal
def predict_class_probability(features):
    return model.predict_proba([features])

# Example prediction
example_animal = X.iloc[0].values  # Taking the first animal's feature set as an example
probability = predict_class_probability(example_animal)
print(f"Predicted Class Probabilities: {probability}")
```

Program 5

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

Screenshots :

LAB-2: Implement ID3 algorithm using decision tree learning with wether dataset.

```
import pandas as pd
import numpy as np

df = pd.read_csv ("weather.csv")

def entropy(target):
    class_count = target.value_counts()
    probs = class_count/ len(target)
    return -np.sum(probs * np.log2(probs))

def info_gain(data , feature, target):
    entropy_before = entropy (target)
    feature_values = data[feature].unique()
    weighted_entropy = 0
    for value in feature_values:
        subset = target[data[feature] == value]
        weighted_entropy += (len(subset)/ len(target))*entropy(subset)
    return entropy_before - weighted_entropy

def print_entropy_and_gain (data, features, target):
    print(" Entropy and info Gain for each feature:")
    for feature in features:
        gain = info_gain (data, features, target)
        ent = entropy (target)
        print (f" Feature :{feature} | Entropy :{ent:.4f} |
                Information Gain :{gain:.4f}")
```

def

def

on tree

```
def build_tree(data, target, features):
    if len(target.unique()) == 1:
        return target.iloc[0]

    if len(features) == 0:
        return target.mode()[0]

    gains = {feature: info_gain(data, feature, target)
             for feature in features}

    best_feature = max(gains, key=gains.get)
    tree = {best_feature: {}}
    feature_values = data[best_feature].unique()
    for value in feature_values:
        subset_data = data[data[best_feature] == value]
        subset_target = target[data[best_feature] == value]

        remaining_features = [f for f in features if f !=
                                  best_feature]
        subtree = build_tree(subset_data, subset_target,
                      remaining_features)

        tree[best_feature][value] = subtree

def print_tree(tree, indent=" "):
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            print(f"{indent}{feature}:")
            for value, subtree in branches.items():
                print(f"{indent} {value} ->", end=" ")
                print_tree(subtree, indent+"  ")
    else:
        print(f"{indent}{tree}")
```

"entropy(subset

")

```
target = df['Decision']
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']
print_entropy_and_gain(df, features, target)
tree = build_tree(df, target, features)
print("Decision Tree:")
print_tree(tree, indent=" ")
```

O/P:

Entropy and Information Gain for each feature:

Feature: Outlook | Entropy: 0.9403 | Information Gain: 0.2

Feature: Temperature | Entropy: 0.9403 | Information Gain: 0.02

Feature: Humidity | Entropy: 0.9403 | Information Gain: 0.15

Feature: Wind | Entropy: 0.9403 | Information Gain: 0.04

Decision Tree:

```
Outlook:
    Sunny -> Humidity:
        High -> No
        Normal -> Yes
    Overcast -> Yes
    Rainy -> Wind:
        Weak -> Yes
        Strong -> No
```

12/3/25

Code :

```
import pandas as pd
import numpy as np

# Sample weather dataset
data = {
    'Day': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny',
    'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],
     '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'],
    'Decision': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Function to calculate entropy
def entropy(target):
    # Get the counts of each class
    class_counts = target.value_counts()
    # Calculate the entropy using the formula
    probabilities = class_counts / len(target)
    return -np.sum(probabilities * np.log2(probabilities))

# Function to calculate information gain
def information_gain(data, feature, target):
    # Calculate the entropy of the whole dataset
    entropy_before = entropy(target)

    # Get the unique values of the feature
    feature_values = data[feature].unique()

    # Calculate the weighted entropy after the split
    weighted_entropy = 0
    for value in feature_values:
        subset = target[data[feature] == value]
```

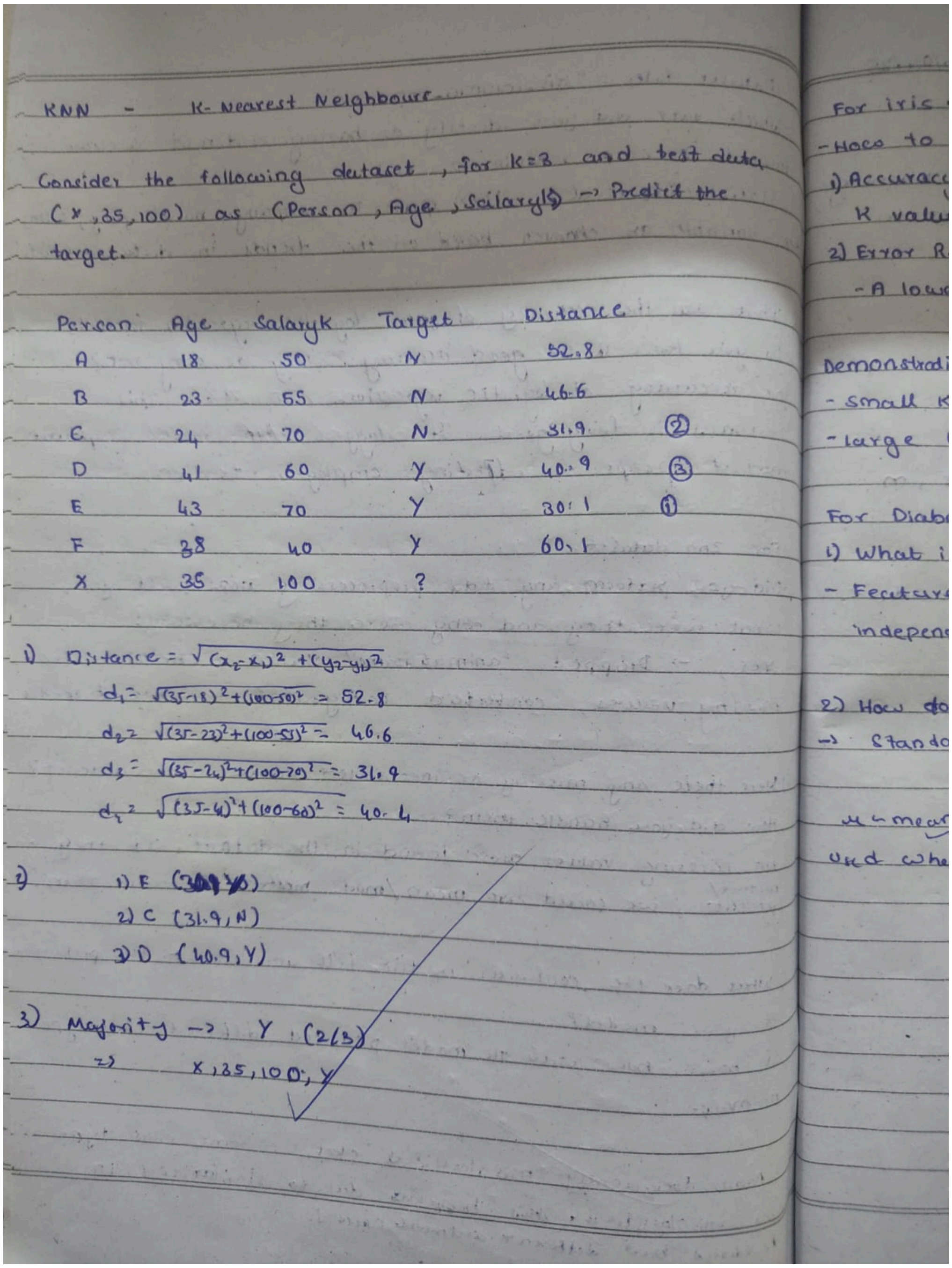
```
        weighted_entropy += (len(subset) / len(target)) * entropy(subset)

    # Information gain is the reduction in entropy
    return entropy_before - weighted_entropy

# Function to print entropy and information gain for each feature
def print_entropy_and_gain(data, features, target):
    print("\nEntropy and Information Gain for each feature:")
    for feature in features:
        gain = information_gain(data, feature, target)
        ent = entropy(target)
        print(f"Feature: {feature} | Entropy: {ent:.4f} | Information Gain: {gain:.4f}")

# Function to build the decision tree recursively
def build_tree(data, target, features):
    # Base case: If all target values are the same, return a leaf node
    if len(target.unique()) == 1:
        return target.iloc[0]

    # Base case: If no features left to split, return the majority class
    if len(features) == 0:
        return target.mode()[0]

    # Calculate information gain for each feature
    gains = {feature: information_gain(data, feature, target) for feature in features}

    # Find the feature with the highest information gain
    best_feature = max(gains, key=gains.get)

    # Create the tree node with the best feature
    tree = {best_feature: {}}

    # Get the unique values of the best feature
    feature_values = data[best_feature].unique()

    # Recursively build the tree for each subset of the data
    for value in feature_values:
        subset_data = data[data[best_feature] == value]
        subset_target = target[data[best_feature] == value]

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

```
        # Build the subtree for the subset
        subtree = build_tree(subset_data, subset_target, remaining_features)

        # Add the subtree to the tree
        tree[best_feature][value] = subtree

    return tree

# Function to print the tree in a visually structured way
def print_tree(tree, indent=""):
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            print(f"{indent}{feature}:")
            for value, subtree in branches.items():
                print(f"{indent}  {value} ->", end=" ")
                print_tree(subtree, indent + "    ")
    else:
        print(f"{indent}{tree}")

# Target variable
target = df['Decision']

# Features
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']

# Step 1: Print entropy and information gain for each feature
print_entropy_and_gain(df, features, target)

# Step 2: Build the decision tree
tree = build_tree(df, target, features)

# Step 3: Print the decision tree (formatted)
print("\nDecision Tree:")
print_tree(tree, indent="    ")
```

Program 6

Build KNN Classification model for a given dataset

Screenshots :



KNN - K-Nearest Neighbours

Consider the following dataset, for K=3 and test data (X, 35, 100) as (Person, Age, Salary(k$)) → Predict the target.

Person	Age	Salaryk	Target	Distance	
A	18	50	N	52.8	
B	23	55	N	46.6	
C	24	70	N	31.9	2
D	41	60	Y	40.9	3
E	43	70	Y	30.1	1
F	38	40	Y	60.1	
X	35	100	?		

1) Distance = $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

$d_1 = \sqrt{(35-18)^2+(100-50)^2} = 52.8$

$d_2 = \sqrt{(35-23)^2+(100-55)^2} = 46.6$

$d_3 = \sqrt{(35-24)^2+(100-70)^2} = 31.9$

$d_4 = \sqrt{(35-41)^2+(100-60)^2} = 40.4$

2) 1) E (30.1, Y)
2) C (31.9, N)
3) D (40.9, Y)

3) Majority → Y (2/3)
⇒ X, 35, 100, Y

For iris dataset:

- How to choose the K value?

1) Accuracy rate approach: We train the model with diff K values and calculate the accuracy for each K

2) Error Rate Approach: Error Rate = 1 - Accuracy

- A lower error rate indicates a better K value.

Demonstration of accuracy rate and error rate:

- small K value may lead to overfitting
- large K value may lead to underfitting

For Diabetes Dataset:

1) What is the purpose of feature scaling?

- Feature scaling is used to normalize the range of independent variables.

2) How to perform feature scaling?

→ Standardization $X_{scaled} = \frac{x - \mu}{\sigma}$

μ - mean, σ - standard deviation

Used when data follows a normal distribution.

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

# Load the Iris dataset
iris = pd.read_csv("/content/iris.csv")
X_iris = iris.drop(columns=['species'])
y_iris = iris['species']

# Split the dataset into training and testing sets
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 the best k value (testing k from 1 to 20)
k_values = range(1, 21)
accuracy_list = []

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_list.append(accuracy_score(y_test_iris, y_pred))

best_k_iris = k_values[np.argmax(accuracy_list)]
print(f"Optimal k value for Iris dataset: {best_k_iris}")

# Train KNN with the best k value
knn_iris = KNeighborsClassifier(n_neighbors=best_k_iris)
knn_iris.fit(X_train_iris, y_train_iris)
y_pred_iris = knn_iris.predict(X_test_iris)

# Display Accuracy and Confusion Matrix for Iris dataset
print("Iris Dataset Accuracy:", accuracy_score(y_test_iris, y_pred_iris))
print("Confusion Matrix for Iris:")
print(confusion_matrix(y_test_iris, y_pred_iris))
print("Classification Report for Iris:")
```

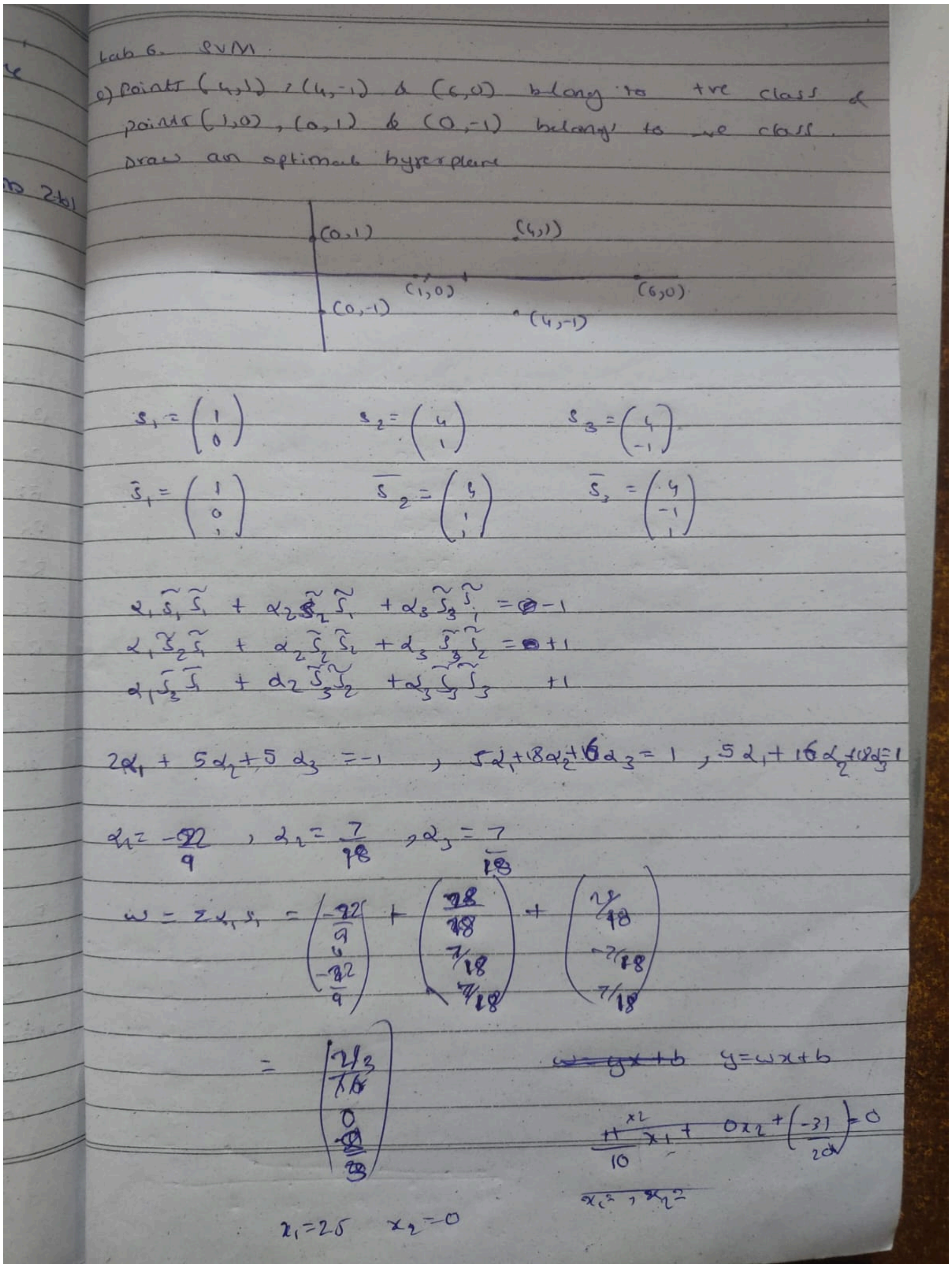
```
print(classification_report(y_test_iris, y_pred_iris))

# Load the Diabetes dataset
diabetes = pd.read_csv("/content/diabetes.csv")
X_diabetes = diabetes.drop(columns=['Outcome'])
y_diabetes = diabetes['Outcome']

# Split into training and testing
X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes = train_test_split(X_diabetes,
y_diabetes, test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train_diabetes = scaler.fit_transform(X_train_diabetes)
X_test_diabetes = scaler.transform(X_test_diabetes)

# Train KNN Classifier for Diabetes dataset
best_k_diabetes = 7  # Assume 7 as a reasonable k value (can be tuned further)
knn_diabetes = KNeighborsClassifier(n_neighbors=best_k_diabetes)
knn_diabetes.fit(X_train_diabetes, y_train_diabetes)
y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)

# Display Accuracy and Confusion Matrix for Diabetes dataset
print("Diabetes Dataset Accuracy:", accuracy_score(y_test_diabetes, y_pred_diabetes))
print("Confusion Matrix for Diabetes:")
print(confusion_matrix(y_test_diabetes, y_pred_diabetes))
print("Classification Report for Diabetes:")
print(classification_report(y_test_diabetes, y_pred_diabetes))


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
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
ConfusionMatrixDisplay

# Load the Heart dataset
heart = pd.read_csv("/content/heart.csv")
```

```
X_heart = heart.drop(columns=['target'])
y_heart = heart['target']

# Split the dataset into training and testing sets
X_train_heart, X_test_heart, y_train_heart, y_test_heart = train_test_split(X_heart, y_heart,
test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train_heart = scaler.fit_transform(X_train_heart)
X_test_heart = scaler.transform(X_test_heart)

# Choose the best k value (testing k from 1 to 20)
k_values = range(1, 21)
accuracy_list = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_heart, y_train_heart)
    y_pred = knn.predict(X_test_heart)
    accuracy_list.append(accuracy_score(y_test_heart, y_pred))

best_k_heart = k_values[np.argmax(accuracy_list)]
print(f"Optimal k value for Heart dataset: {best_k_heart}")

# Train KNN with the best k value
knn_heart = KNeighborsClassifier(n_neighbors=best_k_heart)
knn_heart.fit(X_train_heart, y_train_heart)
y_pred_heart = knn_heart.predict(X_test_heart)

# Display Accuracy and Confusion Matrix for Heart dataset
print("Heart Dataset Accuracy:", accuracy_score(y_test_heart, y_pred_heart))
print("Confusion Matrix for Heart:")
conf_matrix_heart = confusion_matrix(y_test_heart, y_pred_heart)
display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_heart)
display.plot(cmap='Blues')
plt.title("Confusion Matrix for Heart Dataset")
plt.show()

# Classification Report
print("Classification Report for Heart Dataset:")
print(classification_report(y_test_heart, y_pred_heart))
```

Program 7

Build Support vector machine model for a given dataset

Screenshots :



Lab 6. SVM

Q) Points (4,1), (4,-1) & (6,0) belong to +ve class & points (1,0), (0,1) & (0,-1) belongs to -ve class. Draw an optimal hyperplane

$s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $s_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$ $s_3 = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$

$\tilde{s}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ $\tilde{s}_2 = \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}$ $\tilde{s}_3 = \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}$

$\alpha_1 \tilde{s}_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_1 + \alpha_3 \tilde{s}_3 \tilde{s}_1 = -1$

$\alpha_1 \tilde{s}_2 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_2 = +1$

$\alpha_1 \tilde{s}_3 \tilde{s}_1 + \alpha_2 \tilde{s}_3 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_3 \quad +1$

$2\alpha_1 + 5\alpha_2 + 5\alpha_3 = -1$, $5\alpha_1 + 18\alpha_2 + 16\alpha_3 = 1$, $5\alpha_1 + 16\alpha_2 + 18\alpha_3 = 1$

$\alpha_1 = -\frac{22}{9}$, $\alpha_2 = \frac{7}{18}$, $\alpha_3 = \frac{7}{18}$

$w = \sum \alpha_i s_i = \begin{pmatrix} -\frac{22}{9} \\ 0 \\ -\frac{22}{9} \end{pmatrix} + \begin{pmatrix} \frac{28}{18} \\ \frac{7}{18} \\ \frac{7}{18} \end{pmatrix} + \begin{pmatrix} \frac{28}{18} \\ -\frac{7}{18} \\ \frac{7}{18} \end{pmatrix}$

$= \begin{pmatrix} \frac{2}{3} \\ 0 \\ -\frac{5}{3} \end{pmatrix}$

~~w = yx+b~~ $y = wx + b$

$\frac{11}{10} x_1 + 0 x_2 + \left(\frac{-31}{20}\right) = 0$

$x_1 = 2.5 \quad x_2 = 0$

Code :

```
import numpy as np
import matplotlib.pyplot as plt

class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        y = np.where(y <= 0, -1, 1) # Convert labels to -1 and 1
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                    self.b += self.lr * y[idx]

    def predict(self, X):
        approx = np.dot(X, self.w) + self.b
        return np.sign(approx)

    def visualize(self, X, y, new_point=None, prediction=None):
        def get_hyperplane(x, w, b, offset):
            return (-w[0] * x + b + offset) / w[1]

        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)

        # Plot existing data points
        for i, sample in enumerate(X):
            if y[i] == 1:
```

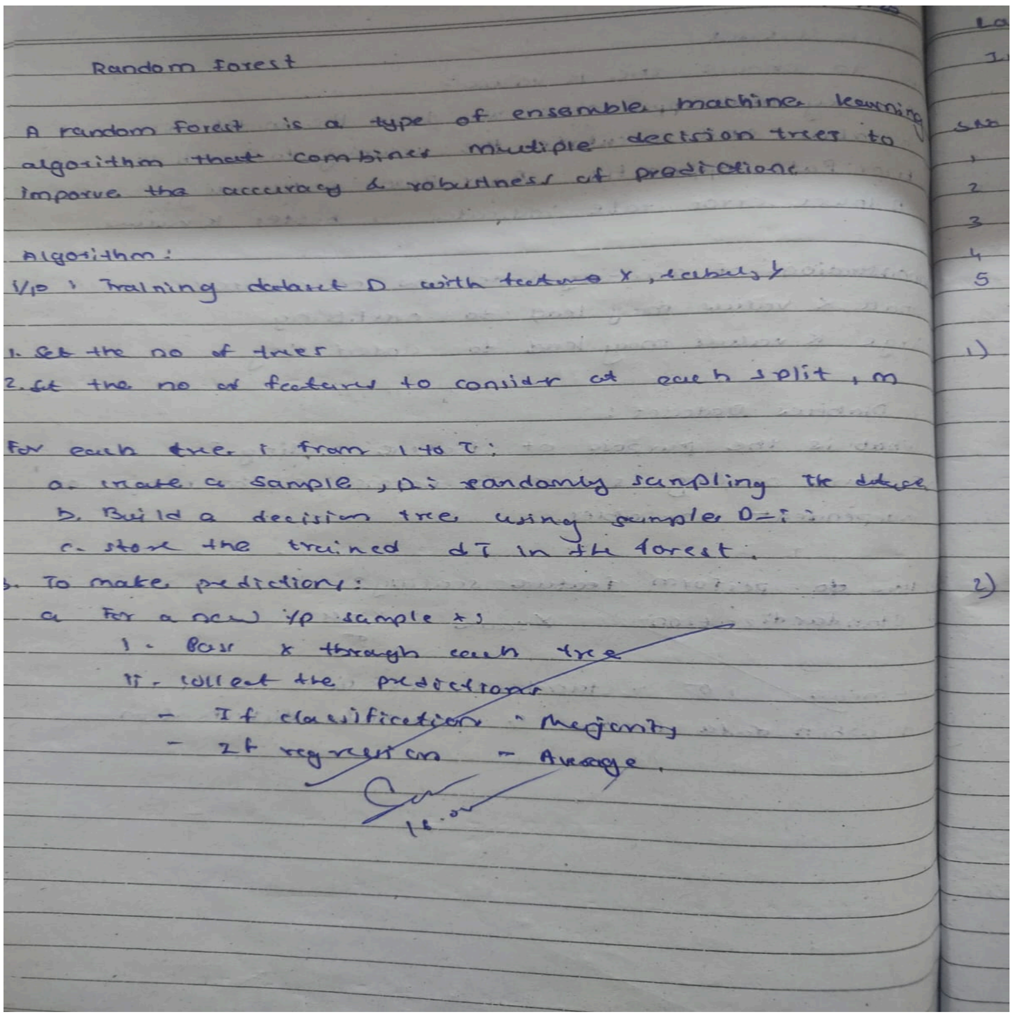
```
            plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +1' if i == 0 else "")
        else:
            plt.scatter(sample[0], sample[1], marker='x', color='red', label='Class -1' if i == 0 else "")

    # Plot decision boundary
    x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1, 100)
    x1 = get_hyperplane(x0, self.w, self.b, 0)
    x1_m = get_hyperplane(x0, self.w, self.b, -1)
    x1_p = get_hyperplane(x0, self.w, self.b, 1)

    ax.plot(x0, x1, 'k-', label='Decision Boundary')
    ax.plot(x0, x1_m, 'k--', label='Margins')
    ax.plot(x0, x1_p, 'k--')

    # Plot the new point
    if new_point is not None:
        color = 'green' if prediction == 1 else 'orange'
        label = f'New Point: Class {"1" if prediction == 1 else "0"}'
        plt.scatter(new_point[0], new_point[1], c=color, s=100, edgecolors='black', label=label,
        marker='*')

    ax.legend()
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("SVM with New Point Prediction")
    plt.grid(True)
    plt.show()



if __name__ == "__main__":
  # Training data
  X = np.array([
    [1, 0],
    [0, 1],
    [0, -1],
    [4, -1],
    [4, 1],
    [6, 0]
  ])
  y = np.array([0, 0, 0, 1, 1, 1]) # 0 -> -1, 1 -> +1
```

```
# New point to classify
new_point = np.array([[5, 5]])

# Train and predict
svm = SVM()
svm.fit(X, y)
prediction = svm.predict(new_point)[0]

# Visualize
svm.visualize(X, y, new_point=new_point[0], prediction=prediction)

# Print prediction
print(f"New point {new_point[0]} classified as: {'Class 1' if prediction == 1 else 'Class 0'}")
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshots :



Random forest

A random forest is a type of ensemble machine learning algorithm that combines multiple decision trees to imporve the accuracy & robustness of predictions

Algorithm:

I/P : Training dataset D with features X, labels y

1. Set the no of trees
2. Set the no of features to consider at each split, m

For each tree i from 1 to T:

a. create a sample, Di randomly sampling the dataset
b. Build a decision tree using sample Di
c. store the trained dT in the forest.

3. To make predictions:

a For a new i/p sample x:

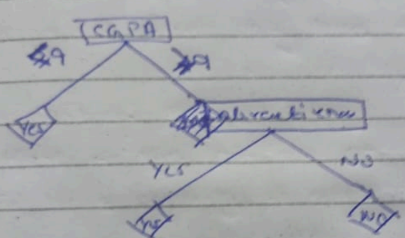
i. Pass x through each tree
ii. collect the predictions
- If classification - Majority
- If regression - Average

Lab-7

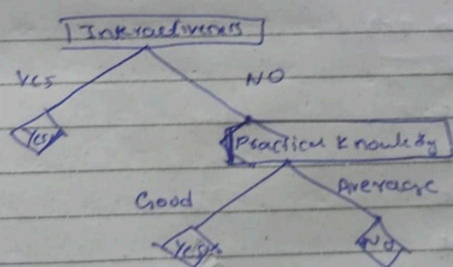
Implement Random forest ensemble method

S.No	CGPA	Interactiveness	Communication skills	Practical knowledge	Job offer
1	≥ 9	Yes	Good	Good	Yes
2	< 9	No	Moderate	Good	Yes
3	≥ 9	No	Moderate	Average	No
4	≥ 9	No	Moderate	Average	No
5	≥ 9	Yes	Moderate	Good	Yes

1)



2)



Code :

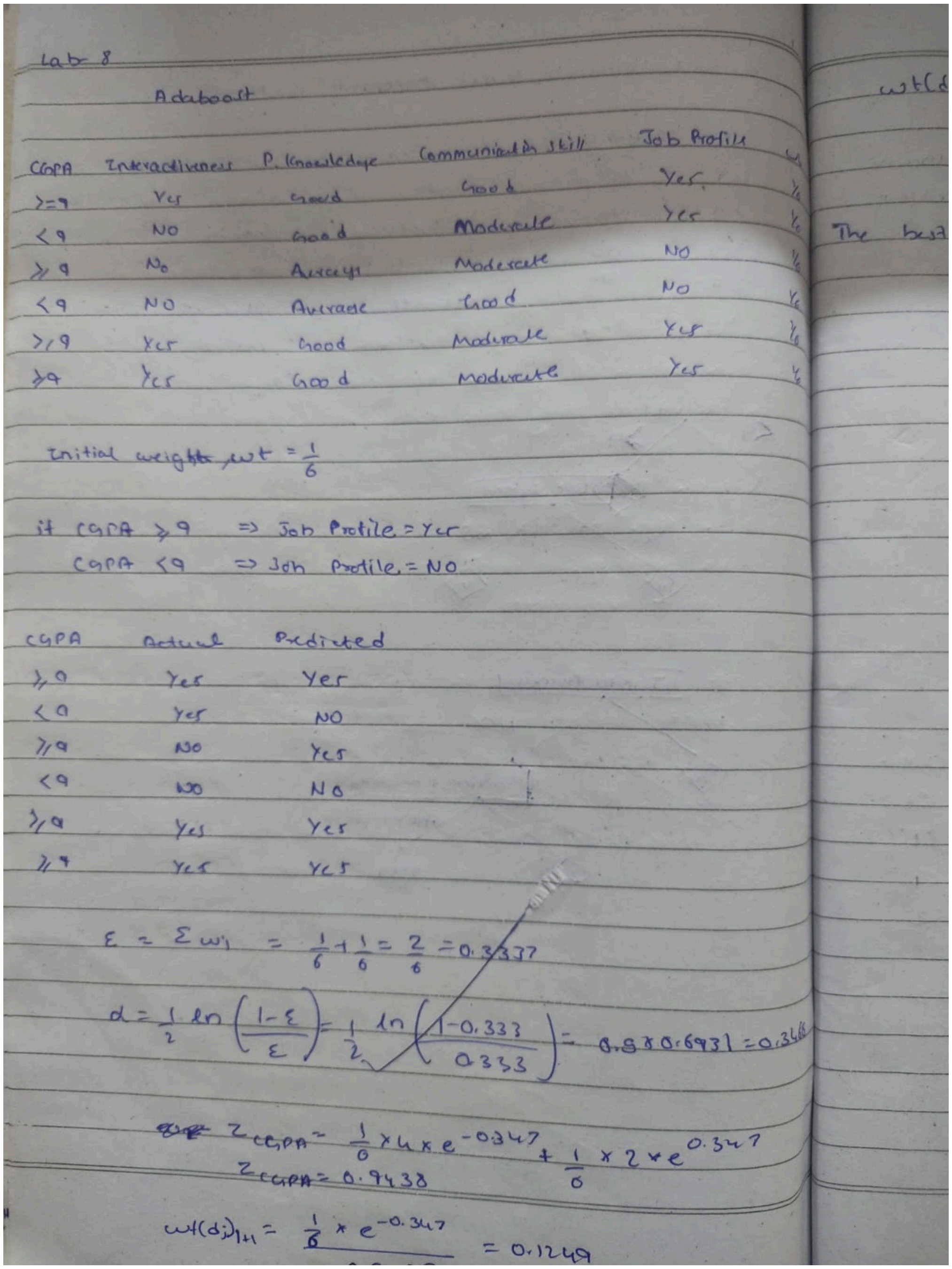
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("iris.csv")  # Make sure this file is in your directory

# Split features and target
X = df.drop("species", axis=1)
y = df["species"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train default Random Forest (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)

# Accuracy
default_accuracy = accuracy_score(y_test, y_pred_default)
print(f"Default accuracy (n_estimators=10): {default_accuracy:.4f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_default, labels=rf_default.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf_default.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (n_estimators=10)")
plt.show()

# Fine-tune: try different values of n_estimators
accuracies = []
tree_range = range(1, 101)

for n in tree_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
```

```
    accuracies.append(accuracy_score(y_test, y_pred))

# Best accuracy and number of trees
best_accuracy = max(accuracies)
best_n = tree_range[accuracies.index(best_accuracy)]
print(f"Best accuracy: {best_accuracy:.4f} using {best_n} trees")

# Plot accuracy vs number of trees
plt.figure(figsize=(10, 5))
plt.plot(tree_range, accuracies, 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()
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshots :



Lab-8

Adaboost

CGPA	Interactiveness	P. Knowledge	Communication skill	Job Profile	
>=9	Yes	Good	Good	Yes	1/6
<9	NO	Good	Moderate	Yes	1/6
≥9	No	Average	Moderate	NO	1/6
<9	NO	Average	Good	NO	1/6
≥9	Yes	Good	Moderate	Yes	1/6
≥9	Yes	Good	Moderate	Yes	1/6

Initial ~~weight~~, $wt = \frac{1}{6}$

if CGPA ≥ 9 ⇒ Job Profile = Yes

CGPA < 9 ⇒ Job Profile = NO

CGPA	Actual	Predicted
≥9	Yes	Yes
<9	Yes	NO
≥9	NO	Yes
<9	NO	NO
≥9	Yes	Yes
≥9	Yes	Yes

$$\varepsilon = \Sigma w_i = \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = 0.3337$$

$$\alpha = \frac{1}{2} \ln\left(\frac{1-\varepsilon}{\varepsilon}\right) = \frac{1}{2} \ln\left(\frac{1-0.333}{0.333}\right) = 0.5 \times 0.6931 = 0.3466$$

$$Z_{CGPA} = \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} \times 2 \times e^{0.347}$$

$$Z_{CGPA} = 0.9438$$

$$wt(d_i)_{i+1} = \frac{\frac{1}{6} \times e^{-0.347}}{\ldots} = 0.1249$$

wt(d

The best

$$wt(d_i)_{t+1} = \frac{1/6 \times e^{0.347}}{0.4428} = 0.2601$$

The best accuracy score obtained 83.3%

		Predicted	
		0	1
	0	7101	308
True	1	1303	1052

Code :

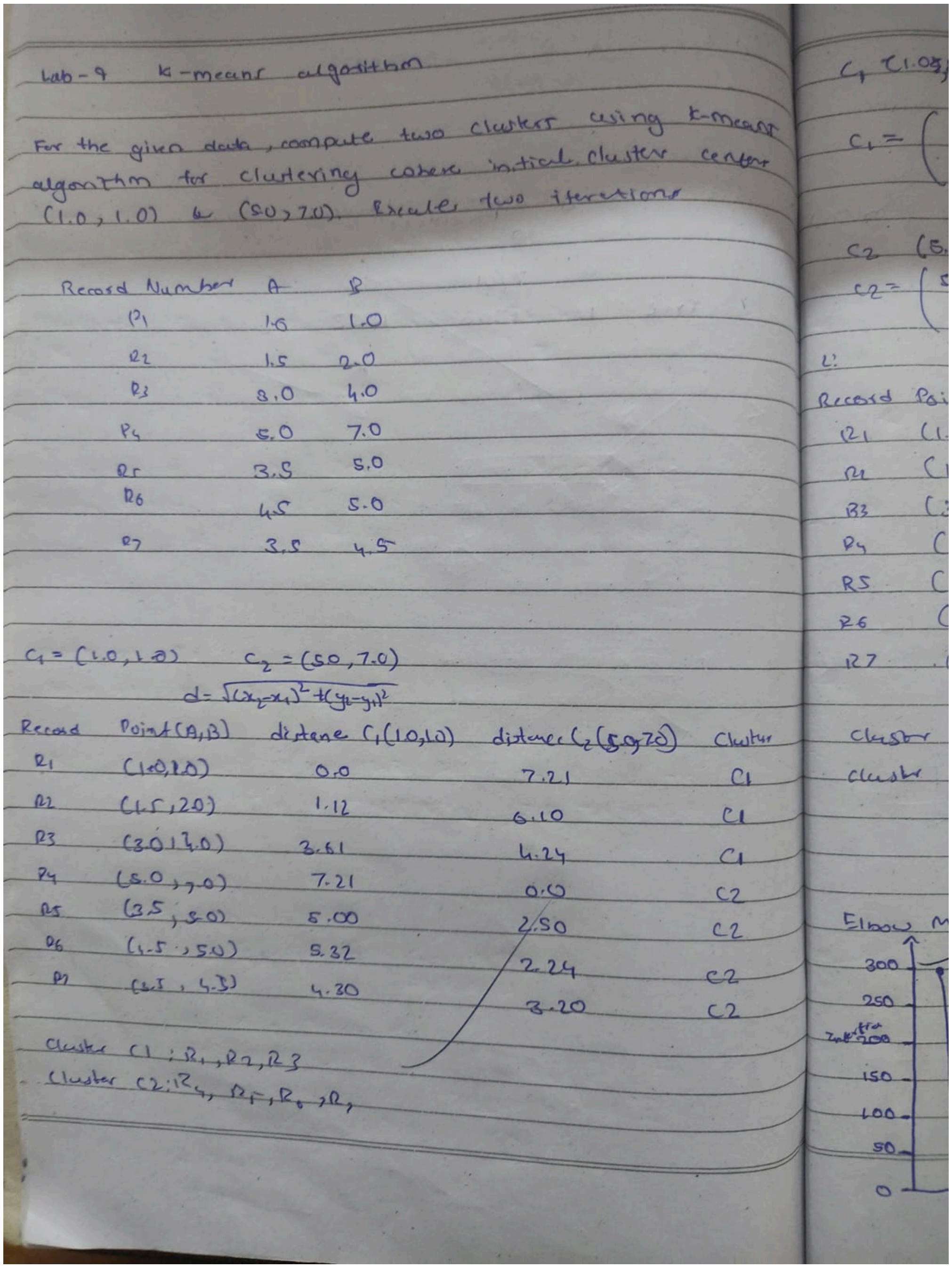
```
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("income.csv")  # Update path if needed

# Features and target
X = df.drop("income_level", axis=1)
y = df["income_level"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train AdaBoost classifier with default n_estimators=10
ada_default = AdaBoostClassifier(n_estimators=10, random_state=42)
ada_default.fit(X_train, y_train)
y_pred_default = ada_default.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred_default)

print(f"Default accuracy (n_estimators=10): {default_accuracy:.4f}")

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_default)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (n_estimators=10)")
plt.show()

# Fine-tune: test n_estimators from 1 to 100
accuracies = []
estimator_range = range(1, 101)

for n in estimator_range:
    ada = AdaBoostClassifier(n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)
    y_pred = ada.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))
```

```
# Best accuracy and corresponding number of estimators
best_accuracy = max(accuracies)
best_n = estimator_range[accuracies.index(best_accuracy)]

print(f"Best accuracy: {best_accuracy:.4f} using {best_n} estimators")

# Plot accuracy vs number of estimators
plt.figure(figsize=(10, 5))
plt.plot(estimator_range, accuracies, marker='o')
plt.title("AdaBoost Accuracy vs Number of Estimators")
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

Program 10

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

Screenshots :



Lab - 9 k-means algorithm

For the given data, compute two clusters using k-means algorithm for clustering where initial cluster centers (1.0, 1.0) & (5.0, 7.0). Execute two iterations

Record Number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

$C_1 = (1.0, 1.0)$ $C_2 = (5.0, 7.0)$

$d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$

Record	Point (A,B)	distance $C_1(1.0,1.0)$	distance $C_2(5.0,7.0)$	Cluster
R1	(1.0, 1.0)	0.0	7.21	C1
R2	(1.5, 2.0)	1.12	6.10	C1
R3	(3.0, 4.0)	3.61	4.24	C1
R4	(5.0, 7.0)	7.21	0.0	C2
R5	(3.5, 5.0)	5.00	2.50	C2
R6	(4.5, 5.0)	5.32	2.24	C2
R7	(3.5, 4.5)	4.30	3.20	C2

Cluster C1 : R_1, R2, R3

Cluster C2 : R_4, R_5, R_6, R_7

C_1 {(1.0, 1.0), (1.5, 2.0) (3.0, 4.0)}

$$C_1 = \left(\frac{1.0 + 1.5 + 3.0}{3}, \frac{1.0 + 2.0 + 4.0}{3} \right) = (1.83, 2.33)$$

C_2 (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)

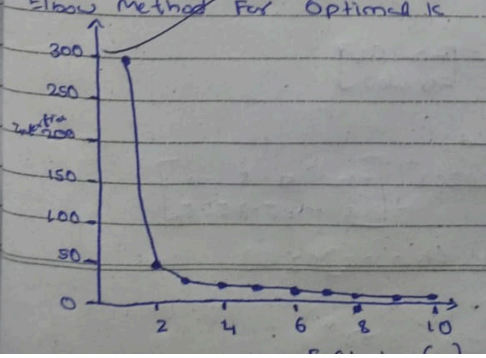
$$C_2 = \left(\frac{5.0 + 3.5 + 4.5 + 3.5}{4}, \frac{7.0, 5.0, 5.0 + 4.5}{4} \right) = (4.125, 5.375)$$

Record	Point (A, B)	distance (1.83, 2.33)	distance (4.125, 5.375)	Cluster
R1	(1.0, 1.0)	1.57	5.62	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)	5.71	1.85	C2
R5	(3.5, 5.0)	3.53	0.78	C2
R6	(4.5, 5.0)	3.92	0.53	C2
R7	(3.5, 4.5)	3.07	1.01	C2

Cluster C1 = R1 (1.0, 1.0) R2 (1.5, 2.0)

Cluster C2 = R3 (3.0, 4.0) R4 (5.0, 7.0) R5 (3.5, 5.0) R6 (4.5, 5.0) R7 (3.5, 4.5)

Elbow Method For Optimal K



Code :

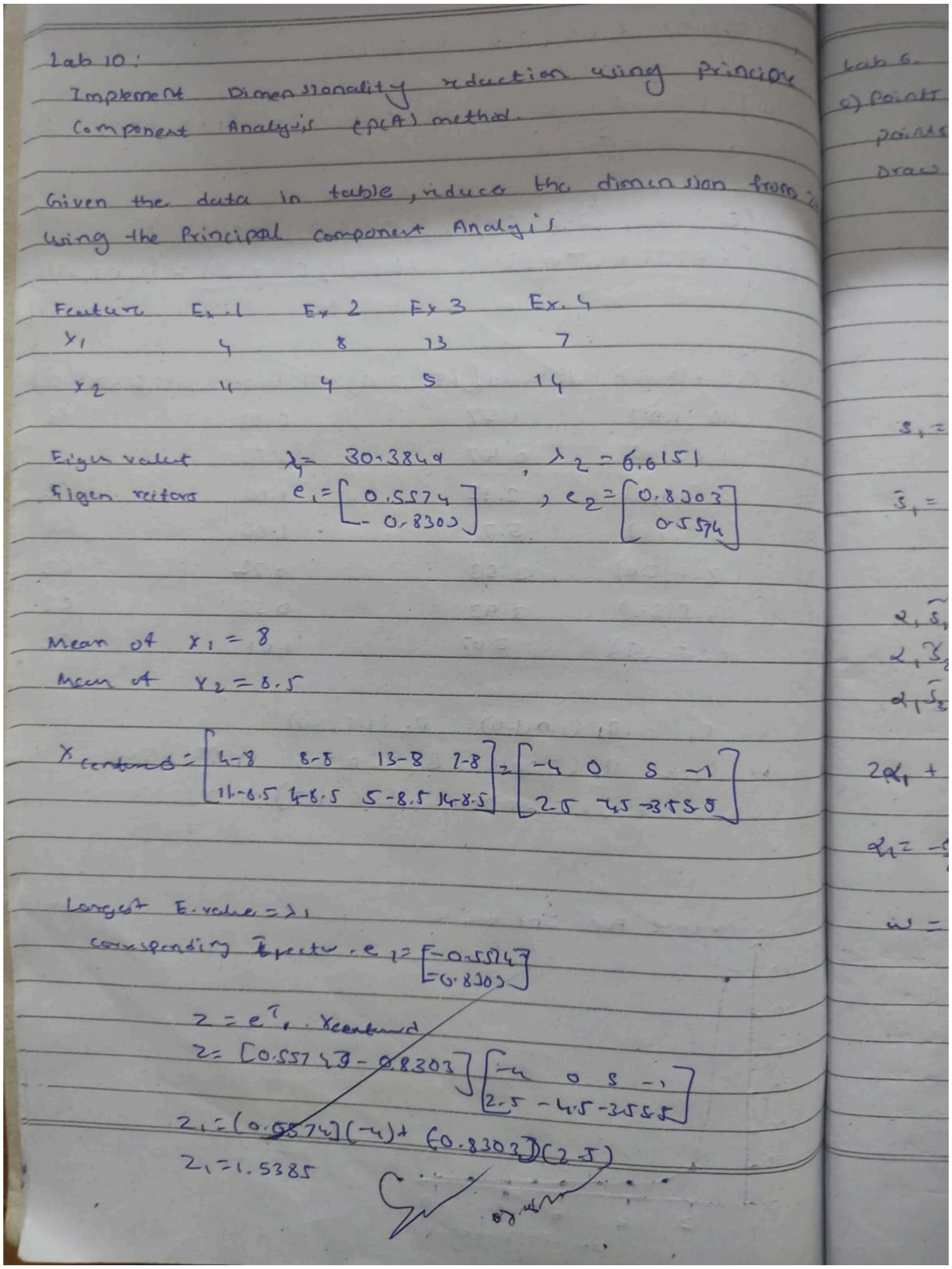
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

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

# Display the first few rows to understand its structure
print(df.head())

# Select only petal width and petal length features for clustering
X = df[['petal_width', 'petal_length']]

# Check if scaling helps with clustering (optional)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 1. Use the elbow method to find the optimal value of k (number of clusters)
inertia = []

# Try different values of k from 1 to 10 and calculate the inertia (sum of squared distances)
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)  # Use scaled data for KMeans
    inertia.append(kmeans.inertia_)

# Plot the elbow plot
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()

# 2. Fit KMeans with the optimal number of clusters (k=3 for example, based on elbow plot)
optimal_k = 3
```

```python
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)

# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y_kmeans, s=50, cmap='viridis')

# Plot the centroids
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='X', label='Centroids')
plt.title(f'K-Means Clustering with k={optimal_k}')
plt.xlabel('Petal Width (scaled)')
plt.ylabel('Petal Length (scaled)')
plt.legend()
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshots :



Lab 10:

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

Given the data in table, reduce the dimension from 2 using the Principal component Analysis

Feature	Ex. 1	Ex. 2	Ex. 3	Ex. 4
x_1	4	8	13	7
x_2	11	4	5	14

Eigen value $\lambda_1 = 30.3849$, $\lambda_2 = 6.6151$

Eigen vectors $e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$, $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Mean of $x_1 = 8$

Mean of $x_2 = 8.5$

$X_{centered} = \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 14-8.5 \end{bmatrix} = \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$

Longest E.value $= \lambda_1$

corresponding E.vector, $e_1 = \begin{bmatrix} -0.5574 \\ -0.8303 \end{bmatrix}$

$z = e^T_1 \cdot X_{centered}$

$z = [0.5574 \; -0.8303] \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$

$z_1 = (0.5574)(-4) + (0.8303)(2.5)$

$z_1 = 1.5385$

Code :

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

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

# Display the first few rows of the dataset
print(df.head())

# Assuming the target column is 'target' and the rest are features
# Split dataset into features (X) and target (y)
X = df.drop('Cholesterol', axis=1)
y = df['Cholesterol']

# Step 1: Convert text columns to numbers using label encoding and one hot encoding
# Identify categorical columns (e.g., if they are object types or have string values)
categorical_columns = X.select_dtypes(include=['object']).columns

# Apply Label Encoding for binary or ordinal categorical columns
label_encoder = LabelEncoder()
for col in categorical_columns:
    X[col] = label_encoder.fit_transform(X[col])

# Step 2: Apply Scaling (StandardScaler)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Build classification models and check the best accuracy

# Split data into training and testing sets
```

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

# Initialize models
models = {
    'SVM': SVC(),
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42)
}

# Train models and evaluate accuracy
accuracies = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies[model_name] = accuracy
    print(f'{model_name} Accuracy: {accuracy:.4f}')

# Step 4: Apply PCA to reduce dimensions and retrain the models

# Apply PCA (choose the number of components based on explained variance)
pca = PCA(n_components=0.95)  # Keep 95% of the variance

X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Retrain models using PCA-transformed data
pca_accuracies = {}
for model_name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred_pca = model.predict(X_test_pca)
    accuracy_pca = accuracy_score(y_test, y_pred_pca)
    pca_accuracies[model_name] = accuracy_pca
    print(f'{model_name} Accuracy with PCA: {accuracy_pca:.4f}')

# Step 5: Comparison of accuracies
print("\n--- Model Performance Comparison ---")
for model_name in models.keys():
    print(f'{model_name} Accuracy without PCA: {accuracies[model_name]:.4f}')
    print(f'{model_name} Accuracy with PCA: {pca_accuracies[model_name]:.4f}')
```