

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT
on**

Machine Learning (23CS6PCMAL)

Submitted by

Shreya Soni (1BM22CS268)

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 **Shreya Soni (1BM22CS268)**, 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 Sheetal V A Assistant Professor Department of CSE, BMSCE | Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE |

Index

| Sl. No. | Date | Experiment Title | Page No. |
|--------------------|-------------|--|---------------------|
| 1 | 5-3-2025 | Write a python program to import and export data using Pandas library functions | 4 |
| 2 | 5-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 7 |
| 3 | 19-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 10 |
| 4 | 2-4-2025 | Build Logistic Regression Model for a given dataset | 13 |
| 5 | 12-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample | 17 |
| 6 | 2-4-2025 | Build KNN Classification model for a given dataset | 20 |
| 7 | 9-4-2025 | Build Support vector machine model for a given dataset | 25 |
| 8 | 7-5-2025 | Implement Random Forest ensemble method on a given dataset | 27 |
| 9 | 7-5-2025 | Implement Boosting ensemble method on a given dataset | 29 |
| 10 | 7-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file | 31 |
| 11 | 7-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method | 33 |

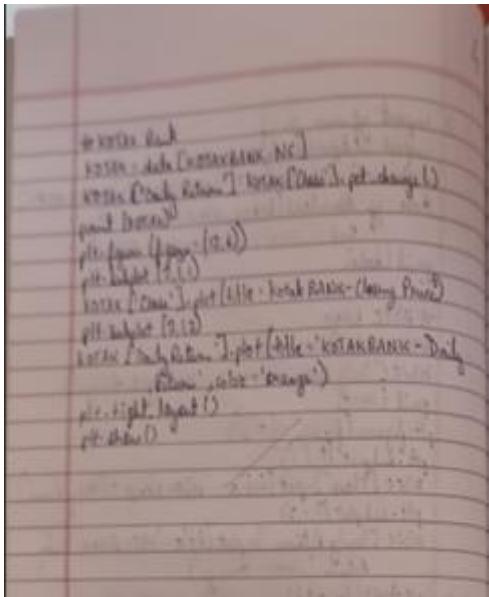
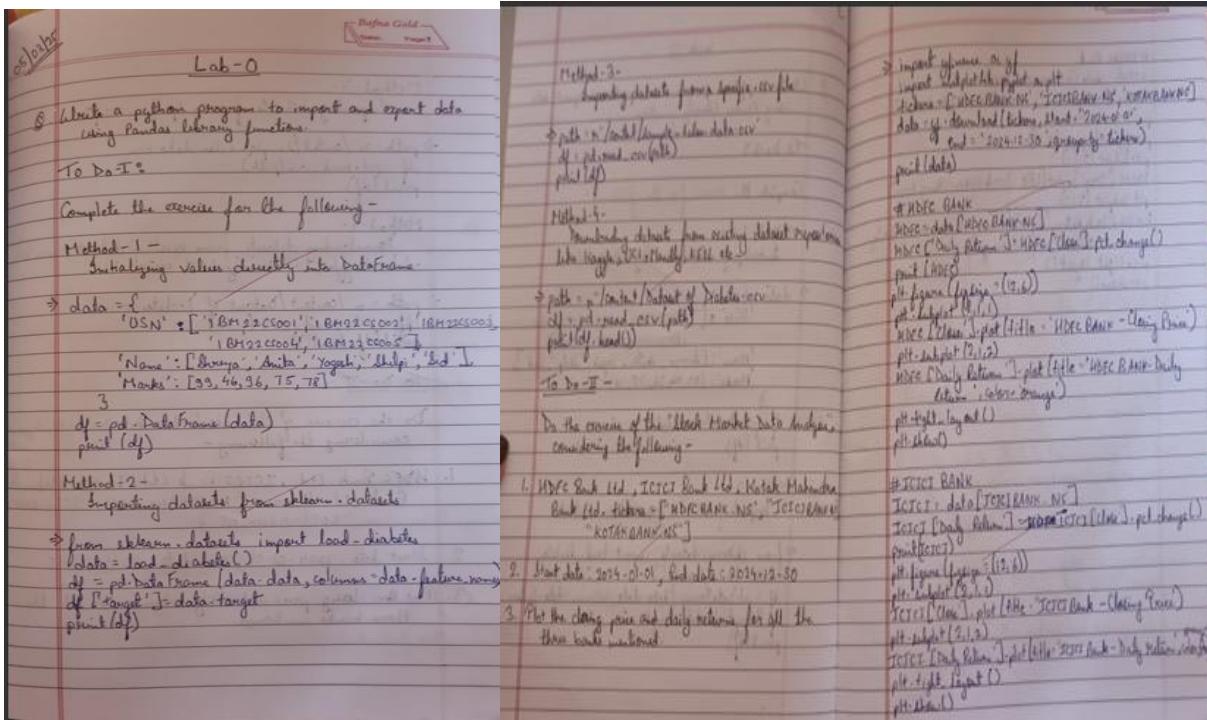
Github Link:

<https://github.com/SHREYASONI28/6thSem-ML-Lab>

Program 1

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

Screenshot



Code:

```
import pandas as pd

data={
    'USN':['1BM22CS001','1BM22CS002','1BM22CS003','1BM22CS004','1BM22CS005'],
    'Name':['Ankita','Anita','Amit','Anish','Arun'],
    'Marks':[99,56,96,85,45]
}

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

from sklearn.datasets import load_diabetes
data=load_diabetes()
df=pd.DataFrame(data.data,columns=data.feature_names)
df['target']=data.target
print(df)

path=r"/content/sample_sales_data.csv"
df=pd.read_csv(path)
print(df)

path=r"/content/Dataset of Diabetes .csv"
df=pd.read_csv(path)
print(df.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=tickers)
print(data)

#HDFCBANK
HDFC=data['HDFCBANK.NS']
HDFC['Daily Return']=HDFC['Close'].pct_change()
print(HDFC)

plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
HDFC['Close'].plot(title='HDFC BANK - Closing Price')
plt.subplot(2,1,2)
HDFC['Daily Return'].plot(title='HDFC BANK - Daily Return',color='orange')
```

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

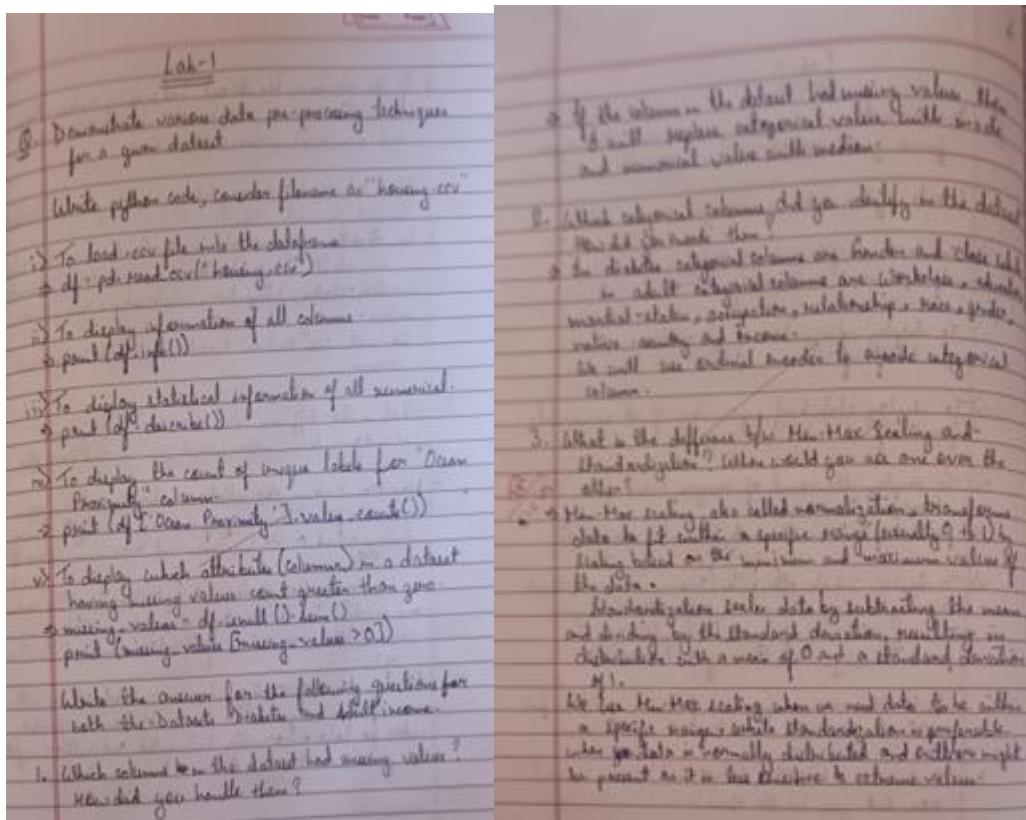
#ICICIBANK
ICICI=data['ICICIBANK.NS']
ICICI['Daily Return']=ICICI['Close'].pct_change()
print(ICICI)

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

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
df = pd.read_csv(r"/content/Dataset of Diabetes .csv")
df.head(10)
df.shape
print(df.info())
print(df.describe())
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```

#Set the values to some value (zero, the mean, the median, etc.).
# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["AGE"]])
imputer2.fit(df_copy[["BMI"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column
df_copy["AGE"] = imputer1.transform(df[["AGE"]])
df_copy["BMI"] = imputer2.transform(df[["BMI"]])

# Verify that there are no missing values left
print(df_copy["AGE"].isnull().sum())
print(df_copy["BMI"].isnull().sum())
#Handling Categorical Attributes
#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column

# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[["M", "F", "f"]])
# Fit and transform the data
df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column
encoded_data = onehot_encoder.fit_transform(df[["CLASS"]])

# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()

# Convert to DataFrame for better visualization
encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)

df_encoded.drop("Gender", axis=1, inplace=True)
df_encoded.drop("CLASS", axis=1, inplace=True)

print(df_encoded.head())
normalizer = MinMaxScaler()
df_encoded[['BMI']] = normalizer.fit_transform(df_encoded[['BMI']])
df_encoded.head()

```

```

scaler = StandardScaler()
df_encoded[['AGE']] = scaler.fit_transform(df_encoded[['AGE']])
df_encoded.head()
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['BMI'].quantile(0.25)
Q3 = df_encoded_copy1['BMI'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['BMI'] = np.where(df_encoded_copy1['BMI'] > upper_bound, upper_bound,
                                    np.where(df_encoded_copy1['BMI'] < lower_bound, lower_bound, df_encoded_copy1['BMI']))

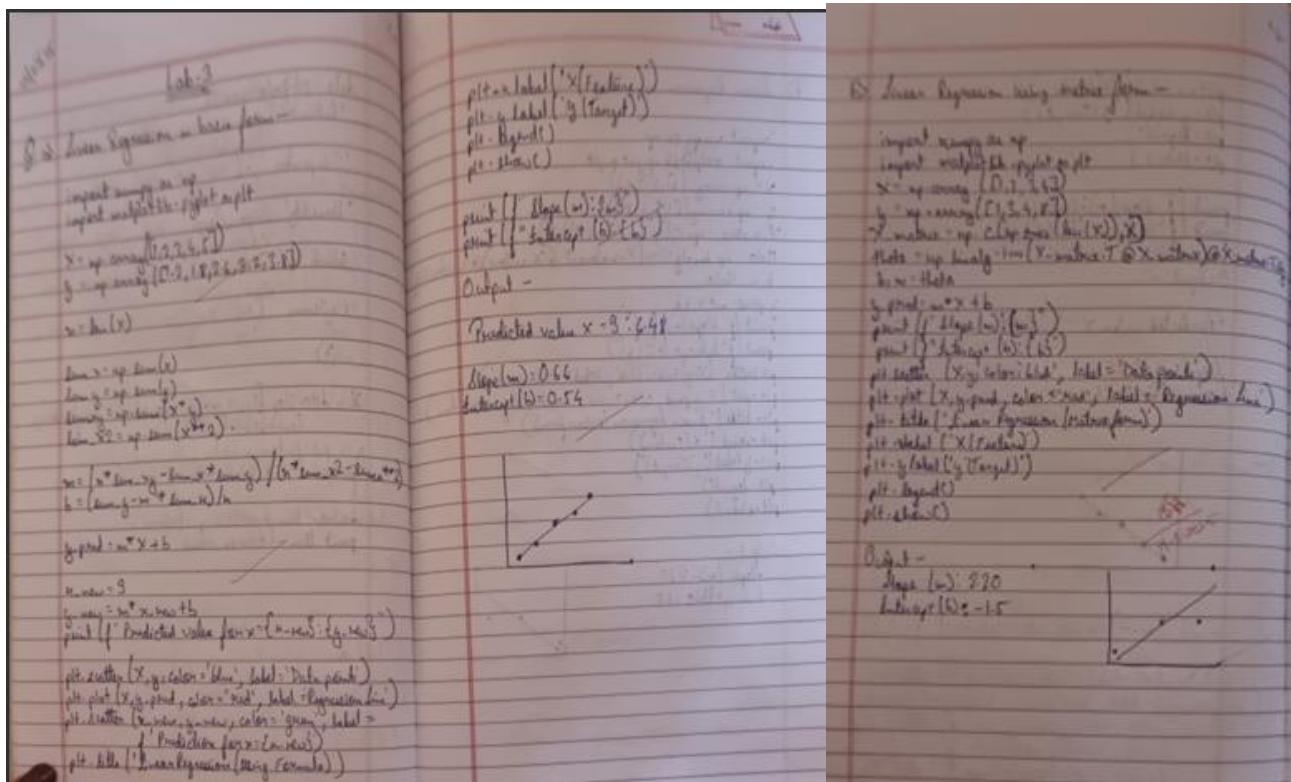
print(df_encoded_copy1.head())
df_encoded_copy2['BMI_zscore'] = stats.zscore(df_encoded_copy2['BMI'])
df_encoded_copy2['BMI'] = np.where(df_encoded_copy2['BMI_zscore'].abs() > 3, np.nan, df_encoded_copy2['BMI'])
# Replace outliers with NaN
print(df_encoded_copy2.head())
df_encoded_copy3['BMI_zscore'] = stats.zscore(df_encoded_copy3['BMI'])
median_salary = df_encoded_copy3['BMI'].median()
df_encoded_copy3['BMI'] = np.where(df_encoded_copy3['BMI_zscore'].abs() > 3, median_salary,
df_encoded_copy3['BMI'])
print(df_encoded_copy3.head())

```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



Code:

```

import pandas as pd
import matplotlib.pyplot as plt

data={"X":[1,2,3,4,5],
      "Y":[1.2,1.8,2.6,3.2,3.8]}
df=pd.DataFrame(data)
df

Xi=df["X"].mean()
Yi=df["Y"].mean()

df["Xi^2"]=[Xi**2 for Xi in df["X"]]
Xisq=df["Xi^2"].mean()

xiyi=[]
x=df["X"]
y=df["Y"]
for i in range(len(x)):

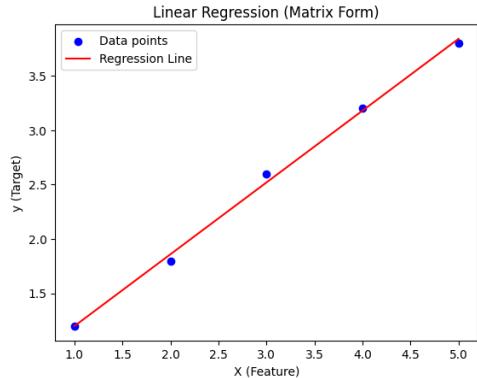
```

```

xiyi.append(x[i]*y[i])
df["XiYi"] = xiyi
print(df["XiYi"])
XiYi2=df["XiYi"].mean()
print(XiYi2)
a1 = (df["XiYi"].sum() - len(df) * Xi * Yi) / (df["X"].apply(lambda x: x**2).sum() - len(df) * Xi**2)
a0 = Yi - a1 * Xi

x=9
Y=a0+a1*x
print(Y)
plt.scatter(df["X"], df["Y"], color='blue', label='Data points') # Scatter plot of original data
plt.plot(df["X"], a0 + a1 * df["X"], color='red', label='Regression Line') # Correct regression line
plt.title('Linear Regression (Matrix Form)')
plt.xlabel('X (Feature)')
plt.ylabel('y (Target)')
plt.legend()
plt.show()

```



```

import numpy as np
import matplotlib.pyplot as plt

X = np.array([1, 2, 3, 4])
y = np.array([1, 3, 4, 8])

X_matrix = np.c_[np.ones(len(X)), X]

theta = np.linalg.inv(X_matrix.T @ X_matrix) @ X_matrix.T @ y

b, m = theta

y_pred = m * X + b

```

```

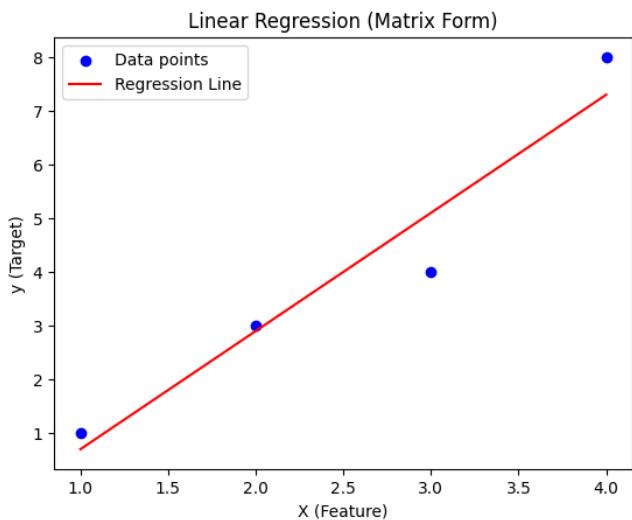
print(f"Slope (m): {m}")
print(f"Intercept (b): {b}")

```

Slope (m): 2.2000000000000006

Intercept (b): -1.5

```
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.title('Linear Regression (Matrix Form)')
plt.xlabel('X (Feature)')
plt.ylabel('y (Target)')
plt.legend()
plt.show()
```



Program 4

Build Logistic Regression Model for a given dataset

Screenshot

Lab-4

Logistic Regression -

- Consider a binary classification problem where we want to predict whether skilled or unskilled based on their weekly hours. The logistic regression model has been trained and the learned parameters are $a_0 = -5$ (intercept) and $a_1 = 0.8$ (coefficient for weekly hours).
- What is the logistic regression equation for this problem?
- Calculate the probability that a skilled employee (one who works 40 hours) will leave.
- Determine the predicted class (leave or not) for this student based on a threshold of 0.5.

Logistic Regression -

$$P(\text{Leave}) = \frac{e^{-a_0 - a_1 \cdot \text{Hours}}}{1 + e^{-a_0 - a_1 \cdot \text{Hours}}} = \frac{e^{-(-5) - 0.8 \cdot 40}}{1 + e^{-(-5) - 0.8 \cdot 40}} = \frac{e^{31.2}}{1 + e^{31.2}} = 0.001$$

$$\text{Leave}(z) = \frac{e^z}{1 + e^z} = 0.001$$

The probability of this class - 0.001, 0.499, 99.9%.

- After building the logistic regression model, write the equation for the following questions -
- For dataset file "HR_Comp_Salary".
- Which variables did you identify as having a direct and strong impact on employee retention?
- Variable impacting employee retention -
- The dataset had a strong direct impact on whether employees with lower salaries are more likely to leave.

What does the equation indicate? All you about the performance of your model?

- This can be well described with class separation ratio - good predictions.
- If sigmoid value > 0.5
- Then this type can most frequently predicted to be true that the employee
- or under particular circumstances leave.
- or make this prediction.
- Logistic regression curve has decision boundary which only set fully linear relationship.

Average monthly hours and number of projects also influence retention. Employees with less salary and no promotion are more likely to leave.

What are the accuracy of your logistic regression model? Do you think this is a good accuracy? Is the model useful?

The logistic regression model achieved an accuracy of 97%.

For accuracy -

- If above 80% \rightarrow You get a good model.
- If below 70% \rightarrow It may need improvement.
- If around 75% \rightarrow It can be improved.

For 2nd dataset -

- Did your performance (the data preprocessing step)? If yes, what were they and why were they necessary?
- Dropped several rows (not useful for classification)
- Bad data (0% training, 100% testing)
- Used Multinomial Logistic Regression for multiclass classification

Were there any missing or invalid values in the dataset? If yes, did you handle them?

No explicit missing values found.

If present, they should be handled by filling or dropping values.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Load dataset
df = pd.read_csv("/content/HR_comma_sep (2).csv")

# Scatter plot: Employee satisfaction vs Retention
plt.scatter(df.satisfaction_level, df.left, marker='+', color='red')
plt.xlabel("Satisfaction Level")
plt.ylabel("Left (1) / Stayed (0)")
plt.title("Impact of Satisfaction Level on Employee Retention")
plt.show()

# Define features (X) and target (y)
X = df[['satisfaction_level']]
y = df['left']

# Split dataset (90% train, 10% test)
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()
model.fit(X_train, y_train)

# Predictions
y_predicted = model.predict(X_test)

# Model Accuracy
print(f"Model Accuracy: {model.score(X_test, y_test):.4f}")

# Probability predictions
print("Predicted Probabilities:")
print(model.predict_proba(X_test))

# Predict for a specific satisfaction level (e.g., 0.4)
predicted_status = model.predict([[0.4]])
print(f"Prediction for Satisfaction Level 0.4: {'Left' if predicted_status[0] == 1 else 'Stayed'}")

# Logistic function
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

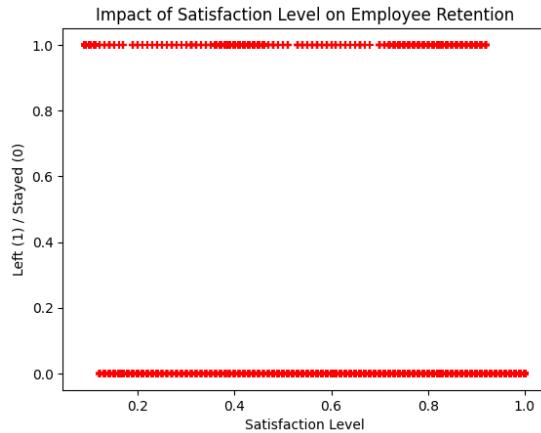
# Custom prediction function
```

```

m, b = model.coef_[0][0], model.intercept_[0]
def prediction_function(satisfaction):
    z = m * satisfaction + b
    y = sigmoid(z)
    return y

satisfaction_test = 0.4
print(f"Sigmoid Prediction for Satisfaction Level {satisfaction_test}:
{prediction_function(satisfaction_test):.4f}")

```



Model Accuracy: 0.7707
 Predicted Probabilities:
 [[0.81879598 0.18120402]
 [0.64435551 0.35564449]
 [0.67008191 0.32991809]
 ...
 [0.85026544 0.14973456]
 [0.93858587 0.06141413]
 [0.90306111 0.09693889]]
 Prediction for Satisfaction Level 0.4: Stayed
 Sigmoid Prediction for Satisfaction Level 0.4: 0.3644

```

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

```

```

# Load the Zoo dataset
file_path = "/content/zoo-data (1).csv"
zoo_data = pd.read_csv(file_path)

# Drop the 'animal_name' column as it is not a relevant feature
X = zoo_data.drop(['animal_name', 'class_type'], axis=1) # Features
y = zoo_data['class_type'] # Target variable

```

```

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

# Initialize the Logistic Regression model for multi-class classification
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=200)
# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Multinomial Logistic Regression model: {accuracy:.2f}")

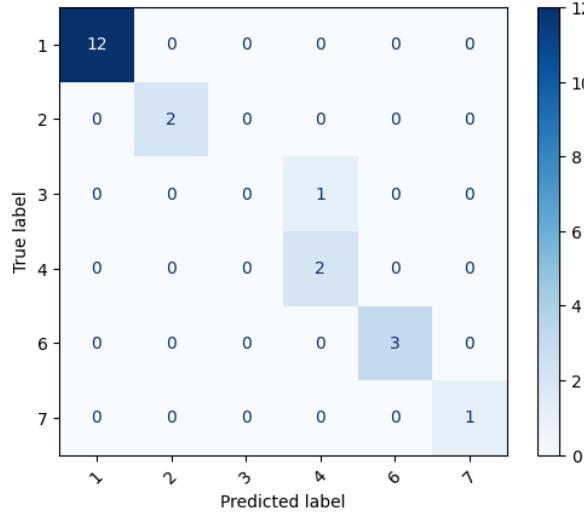
# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Adjust display labels to match actual present labels in the test set
unique_classes_in_test = sorted(y_test.unique())

# Display confusion matrix
cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=unique_classes_in_test)
cm_display.plot(cmap='Blues', xticks_rotation=45)
plt.show()

```

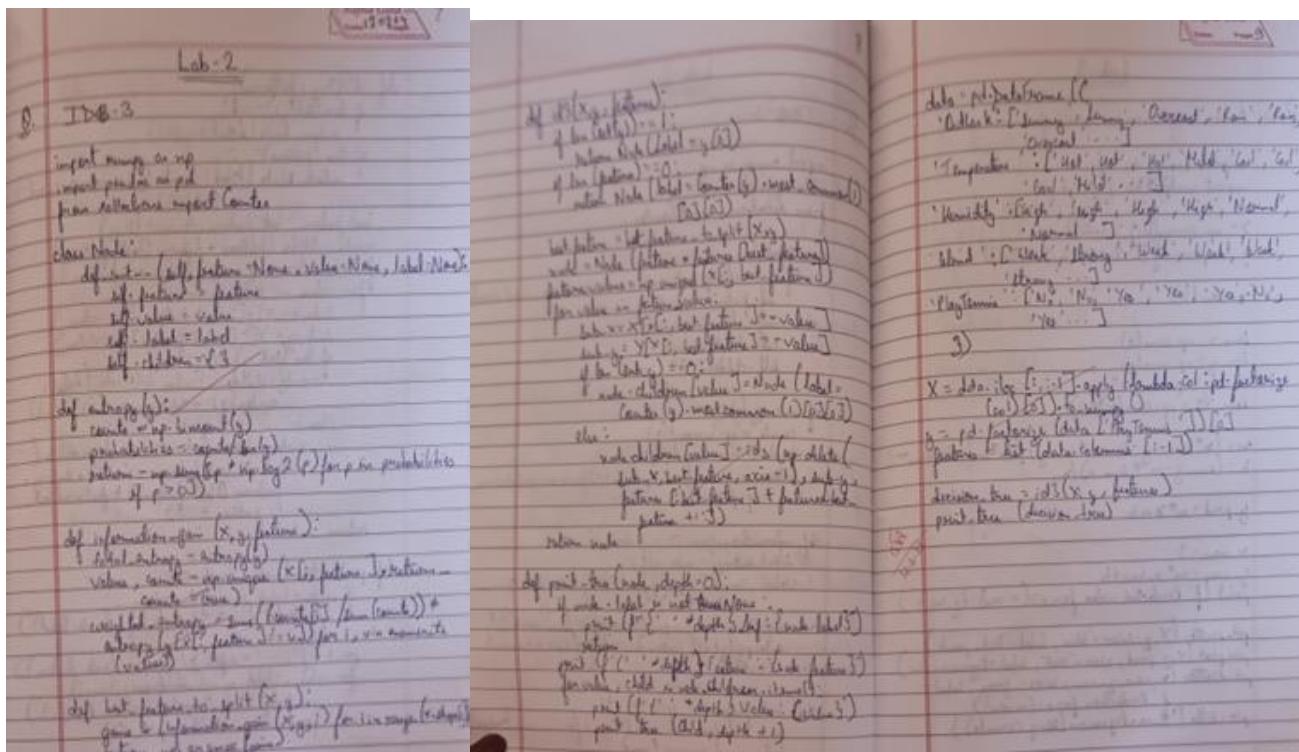
Accuracy of the Multinomial Logistic Regression model: 0.95



Program 5

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

Screenshot



Code:

```

import numpy as np
import pandas as pd
from collections import Counter
class Node:
    def __init__(self, feature=None, value=None, label=None):
        self.feature = feature # Attribute to split on
        self.value = value # Value of the attribute
        self.label = label # Label if it's a leaf node
        self.children = {} # Dictionary of child nodes

    def entropy(y):
        counts = np.bincount(y)
        probabilities = counts / len(y)
        return -np.sum([p * np.log2(p) for p in probabilities if p > 0])

    def information_gain(X, y, feature):
        total_entropy = entropy(y)
        values, counts = np.unique(X[:, feature], return_counts=True)
        weighted_entropy = sum([counts[i] / len(X) * entropy(y[y[:, feature] == values[i]]) for i in range(len(values))])
        return total_entropy - weighted_entropy

    def best_feature_to_split(x):
        gains = [information_gain(x, y, i) for i in range(len(x[0]))]
        return np.argmax(gains)

    def build_tree(x, y, feature):
        if len(np.unique(y)) == 1:
            return Node(label=y[0])
        if len(x[0]) == 0:
            return Node(label=Counter(y).most_common(1)[0][0])
        if self.feature is None:
            self.feature = best_feature_to_split(x)
            self.value = np.unique(x[:, self.feature])[0]
        if self.value not in x[:, self.feature]:
            self.label = Counter(y).most_common(1)[0][0]
            return Node(label=self.label)
        else:
            left_index = x[:, self.feature] < self.value
            right_index = x[:, self.feature] >= self.value
            left_x, right_x = x[left_index], x[right_index]
            left_y, right_y = y[left_index], y[right_index]
            self.left = build_tree(left_x, left_y, None)
            self.right = build_tree(right_x, right_y, None)
            return self

    def print_tree(node, depth=0):
        if node.label is not None:
            print(f'{" " * depth}label: {node.label}')
        else:
            print(f'{" " * depth}feature: {node.feature}, value: {node.value}')
            print(f'{" " * depth}left: {node.left}')
            print(f'{" " * depth}right: {node.right}')

    def predict(node, x):
        if node.label is not None:
            return node.label
        if x[node.feature] < node.value:
            return predict(node.left, x)
        else:
            return predict(node.right, x)
    
```

```

weighted_entropy = sum((counts[i] / sum(counts)) * entropy(y[X[:, feature] == v])) for i, v in
enumerate(values))
return total_entropy - weighted_entropy

def best_feature_to_split(X, y):
    gains = [information_gain(X, y, i) for i in range(X.shape[1])]
    return np.argmax(gains)

def id3(X, y, features):
    if len(set(y)) == 1:
        return Node(label=y[0])
    if len(features) == 0:
        return Node(label=Counter(y).most_common(1)[0][0])
    best_feature = best_feature_to_split(X, y)
    node = Node(feature=features[best_feature])
    feature_values = np.unique(X[:, best_feature])
    for value in feature_values:
        sub_X = X[X[:, best_feature] == value]
        sub_y = y[X[:, best_feature] == value]
        if len(sub_y) == 0:
            node.children[value] = Node(label=Counter(y).most_common(1)[0][0])
        else:
            node.children[value] = id3(np.delete(sub_X, best_feature, axis=1), sub_y, features[:best_feature] +
features[best_feature+1:])
    return node
    if node.label is not None:
        print(f"{' ' * depth}Leaf: {node.label}")
        return
    print(f"{' ' * depth}Feature: {node.feature}")
    for value, child in node.children.items():
        print(f"{' ' * depth}Value: {value}")
        print_tree(child, depth + 1)

# Example dataset
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny',
    'Overcast', 'Overcast', 'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'Normal',
    'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', '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', 'Yes', 'No']
})
X = data.iloc[:, :-1].apply(lambda col: pd.factorize(col)[0]).to_numpy()
y = pd.factorize(data['PlayTennis'])[0]
features = list(data.columns[:-1])
decision_tree = id3(X, y, features)
print_tree(decision_tree)

```

Feature: Outlook

Value: 0

 Feature: Humidity

 Value: 0

 Leaf: 0

 Value: 1

 Leaf: 1

 Value: 1

 Leaf: 1

 Value: 2

 Feature: Wind

 Value: 0

 Leaf: 1

 Value: 1

 Leaf: 0

Program 6

Build KNN Classification model for a given dataset

Screenshot

Lab 5

Q1. Build KNN Classification model for a given dataset.

Consider the following dataset for k=3 and test data (X=35, Y=1) to find the class of the test data using KNN classifier model and predict target.

| Person | Age | Height | Weight | Class |
|--------|-----|--------|--------|-------|
| A | 18 | 50 | N | 52.83 |
| B | 25 | 55 | N | 46.59 |
| C | 24 | 70 | N | 31.94 |
| D | 41 | 60 | Y | 48.44 |
| E | 35 | 70 | Y | 31.06 |
| F | 38 | 40 | Y | 60.08 |
| X | 35 | 100 | - | - |

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

$$A: \sqrt{(35-18)^2 + (100-50)^2} = 52.83$$

$$B: \sqrt{(35-25)^2 + (100-55)^2} = 46.59$$

$$C: \sqrt{(35-24)^2 + (100-70)^2} = 31.94$$

$$D: \sqrt{(35-41)^2 + (100-60)^2} = 48.44$$

$$E: \sqrt{(35-40)^2 + (100-70)^2} = 31.06$$

$$\therefore \sqrt{(35-35)^2 + (100-70)^2} = 30.00$$

$$D-3 \rightarrow Y = YA$$

Code:

```

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

# Function to train and evaluate KNN model
def knn_classification(data_path, target_column, dataset_name, k=5):
    # Load dataset
    df = pd.read_csv(data_path)

    # Split features and target
    X = df.drop(columns=[target_column])
    y = df[target_column]

```

```

# Split 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)

# Feature scaling for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train KNN model
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of KNN on {dataset_name} dataset: {accuracy:.4f}')
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - {dataset_name}')
plt.show()

# Run KNN classification on both datasets
knn_classification('/content/iris (3).csv', 'species', 'Iris', k=5)
knn_classification('/content/diabetes.csv', 'Outcome', 'Diabetes', k=5)

```

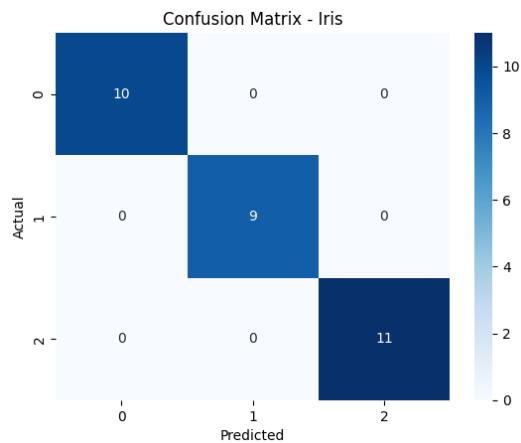
Accuracy of KNN on Iris dataset: 1.0000

Classification Report:

| | precision | recall | f1-score | support |
|--|-----------|--------|----------|---------|
|--|-----------|--------|----------|---------|

| | | | | |
|------------|------|------|------|----|
| setosa | 1.00 | 1.00 | 1.00 | 50 |
| versicolor | 1.00 | 1.00 | 1.00 | 50 |
| virginica | 1.00 | 1.00 | 1.00 | 50 |

| | | | | |
|--------------|------|------|------|----|
| accuracy | | 1.00 | 30 | |
| macro avg | 1.00 | 1.00 | 1.00 | 30 |
| weighted avg | 1.00 | 1.00 | 1.00 | 30 |



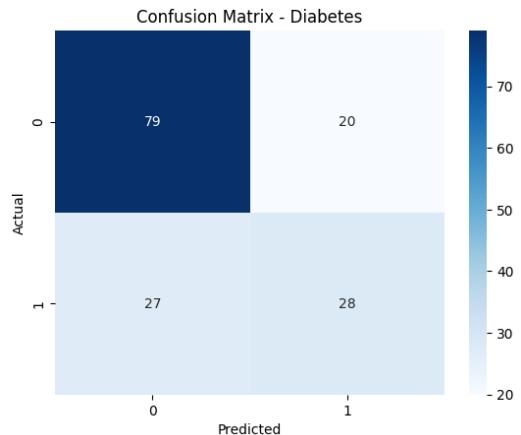
Accuracy of KNN on Diabetes dataset: 0.6948

Classification Report:

| | precision | recall | f1-score | support |
|--|-----------|--------|----------|---------|
|--|-----------|--------|----------|---------|

| | | | | |
|---|------|------|------|----|
| 0 | 0.75 | 0.80 | 0.77 | 99 |
| 1 | 0.58 | 0.51 | 0.54 | 55 |

| | accuracy | 0.69 | 154 |
|--------------|----------|------|------|
| macro avg | 0.66 | 0.65 | 0.66 |
| weighted avg | 0.69 | 0.69 | 0.69 |



```

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

```

Load dataset

```
df = pd.read_csv('/content/heart.csv')
```

```

# Define features and target
X = df.drop(columns=['target']) # Assuming 'target' is the classification column
y = df['target']

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

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the best K value
k_values = range(1, 21)
accuracy_scores = []
for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))

best_k = k_values[np.argmax(accuracy_scores)]
print(f'Best K value: {best_k}')

# Train model with best K
best_model = KNeighborsClassifier(n_neighbors=best_k)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy with best K ({best_k}): {accuracy:.4f}')
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - KNN (K={best_k})')
plt.show()

# Plot K values vs. Accuracy
plt.plot(k_values, accuracy_scores, marker='o')
plt.xlabel('K Value')
plt.ylabel('Accuracy')

```

```
plt.title('K Value vs Accuracy')
plt.show()
```

Best K value: 7

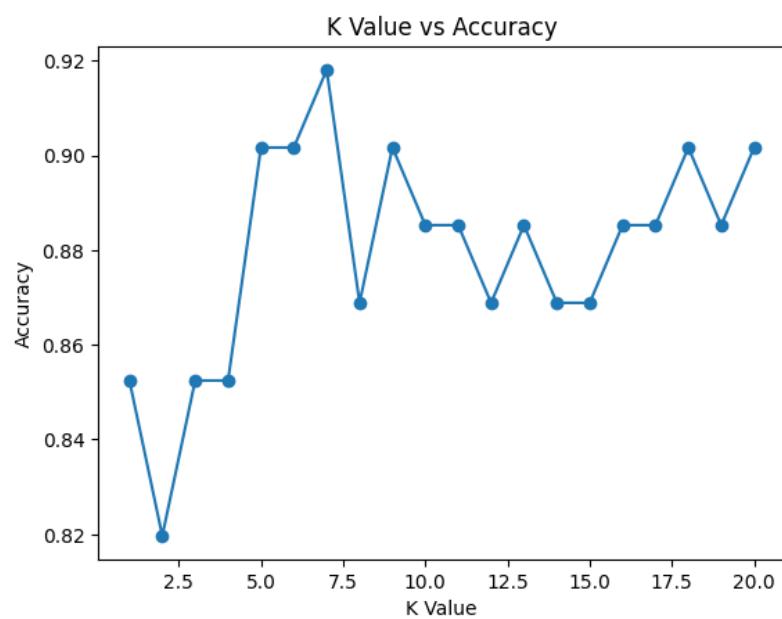
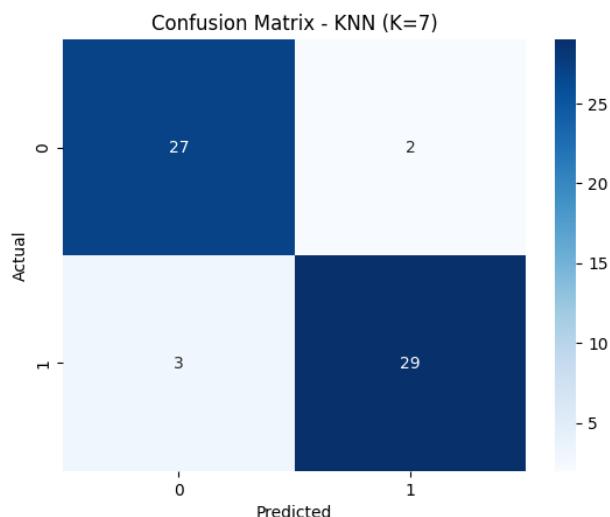
Accuracy with best K (7): 0.9180

Classification Report:

| | precision | recall | f1-score | support |
|--|-----------|--------|----------|---------|
|--|-----------|--------|----------|---------|

| | | | | |
|---|------|------|------|----|
| 0 | 0.90 | 0.93 | 0.92 | 29 |
| 1 | 0.94 | 0.91 | 0.92 | 32 |

| | | | | |
|--------------|------|------|------|----|
| accuracy | | 0.92 | | 61 |
| macro avg | 0.92 | 0.92 | 0.92 | 61 |
| weighted avg | 0.92 | 0.92 | 0.92 | 61 |



Program 7

Build Support vector machine model for a given dataset

Code:

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

# Define the Linear SVM class
class LinearSVM:
    def __init__(self, learning_rate=0.001, reg_strength=0.1, num_iterations=1000):
        self.learning_rate = learning_rate
        self.reg_strength = reg_strength
        self.num_iterations = num_iterations

    def fit(self, X, y):
        # Initialize weights and bias
        num_samples, num_features = X.shape
        self.W = np.zeros(num_features) # Weights
        self.b = 0 # Bias

        # Gradient Descent
        for _ in range(self.num_iterations):
            # Compute the margin (decision function)
            margins = 1 - y * (np.dot(X, self.W) + self.b)
            # Compute gradient
            dw = -2 * np.dot(X.T, (y * (margins > 0))) / num_samples + 2 * self.reg_strength * self.W
            db = -2 * np.sum(y * (margins > 0)) / num_samples

            # Update weights and bias
            self.W -= self.learning_rate * dw
            self.b -= self.learning_rate * db

    def predict(self, X):
        # Make predictions
        return np.sign(np.dot(X, self.W) + self.b)

# Generate toy data (binary classification)
np.random.seed(42)
num_samples = 100
X = np.random.randn(num_samples, 2)
y = np.ones(num_samples)
y[X[:, 0] < X[:, 1]] = -1 # Assign different class based on condition

# Train the Linear SVM
svm = LinearSVM(learning_rate=0.001, reg_strength=0.1, num_iterations=1000)
svm.fit(X, y)
```

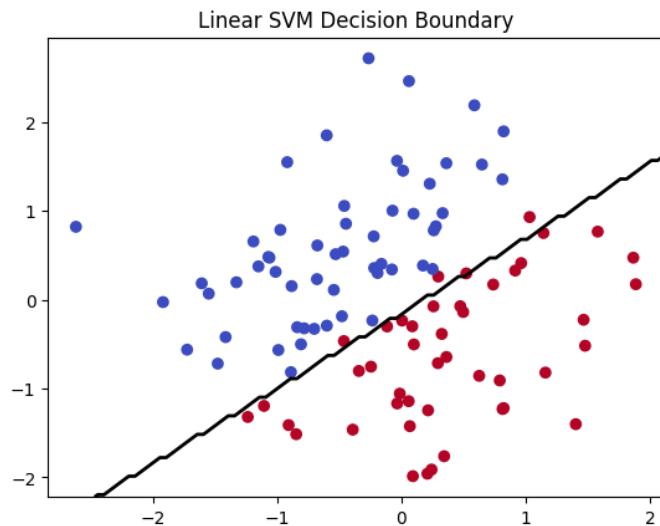
```

# Predict
y_pred = svm.predict(X)

# Visualize the decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100), np.linspace(ylim[0], ylim[1], 100))
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.title("Linear SVM Decision Boundary")
plt.show()

# Print accuracy (simple comparison)
accuracy = np.mean(y_pred == y)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

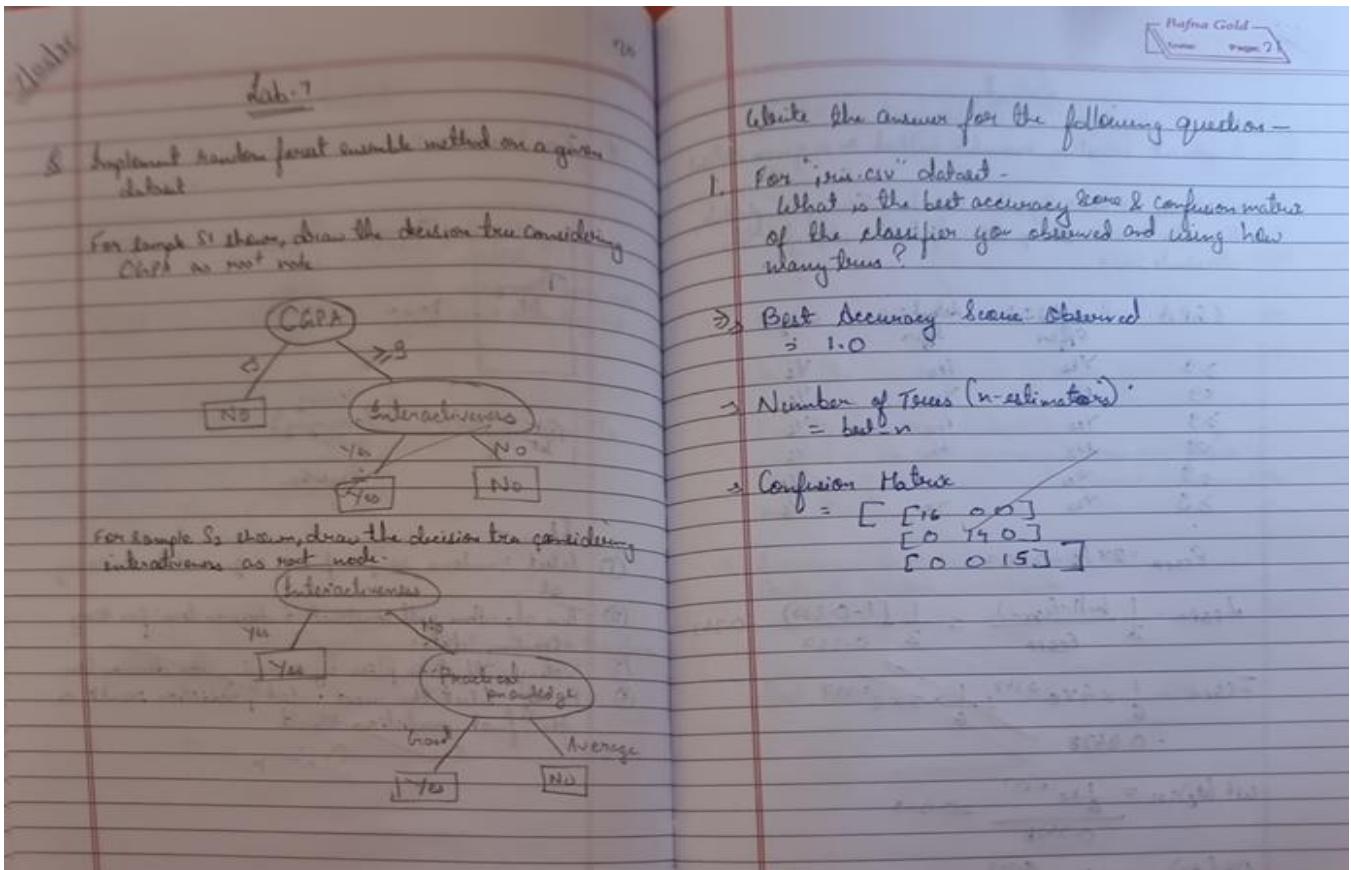


Accuracy: 96.00%

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot



Code:

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

```
# Load the iris dataset from CSV
df = pd.read_csv("/content/iris (2).csv")
```

```
# Assuming last column is the label
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

```
# Split into training and test sets (70% train, 30% test)
```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# 1. Train RF Classifier with default n_estimators=10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
accuracy_default = accuracy_score(y_test, y_pred_default)

print(f"Default RF Accuracy (n_estimators=10): {accuracy_default:.4f}")

best_accuracy = 0
best_n = 0
accuracies = []

for n in range(1, 101):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

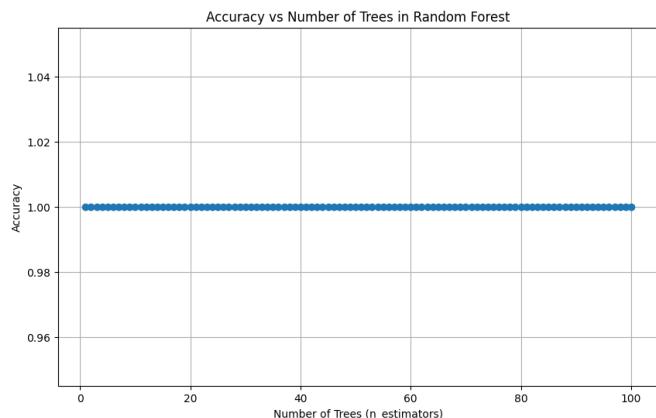
    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n

print(f"Best RF Accuracy: {best_accuracy:.4f} with n_estimators = {best_n}")
# Plot accuracy vs. number of trees
plt.figure(figsize=(10, 6))
plt.plot(range(1, 101), accuracies, marker='o')
plt.title("Accuracy vs Number of Trees in Random Forest")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()

```

Default RF Accuracy (n_estimators=10): 1.0000

Best RF Accuracy: 1.0000 with n_estimators = 1



Program 9

Implement Boosting ensemble method on a given dataset

Screenshot

Lab-3

Q. Implement Boosting ensemble method on a given dataset considering AdaBoost algorithm for the following template data. Then the decision stump calculation steps for the attribute CGPA.

| CGPA | Predicted Job Offer | Actual Job Offer | Weight |
|------|---------------------|------------------|--------|
| > 9 | Yes | Yes | 1/6 |
| < 9 | No | Yes | 1/6 |
| > 9 | Yes | No | 1/6 |
| < 9 | No | No | 1/6 |
| > 9 | Yes | Yes | 1/6 |
| > 9 | Yes | Yes | 1/6 |

$$E_{\text{GPA}} = 2 \times \frac{1}{6} = 0.333$$

$$\Delta E_{\text{GPA}} = \frac{1}{2} \ln(1 - E_{\text{GPA}}) = \frac{1}{2} \ln(1 - 0.333) = 0.247$$

$$\Delta E_{\text{GPA}} = \frac{1}{6} \times 4 \times e^{-0.247} + \frac{1}{6} \times 2 \times e^{0.247} \\ = 0.8478$$

$$Wt(d_0)_{i+1} = \frac{\frac{1}{6} \times e^{-0.247}}{0.8478} = 0.1248$$

$$Wt(d_0)_{i+1} = \frac{\frac{1}{6} \times e^{0.247}}{0.8478} = 0.2501$$

| CGPA | Predicted Job Offer | Actual Job Offer | Weight |
|------|---------------------|------------------|--------|
| > 9 | Yes | Yes | 0.1248 |
| < 9 | No | Yes | 0.1248 |
| > 9 | Yes | No | 0.2501 |
| < 9 | No | No | 0.2501 |
| > 9 | Yes | Yes | 0.1248 |
| > 9 | Yes | Yes | 0.1248 |

Write the answer for the following question -

1. For "income.csv" dataset what is the best accuracy score and confusion matrix of the classifier you observed and why how many trees?

→ Accuracy with 10 estimators: 0.8277
Confusion Matrix (10 estimators):

$$\begin{bmatrix} 1072 & 387 \\ 2138 & 14081 \end{bmatrix}$$

Best accuracy: 0.831 with n_estimators=42

Code:

```

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

# Step 1: Load the dataset
df = pd.read_csv("/content/income.csv")

# Step 2: Split into features and target
X = df.drop(columns=['income_level'])
y = df['income_level']

# Step 3: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: AdaBoost with 10 estimators
model_10 = AdaBoostClassifier(n_estimators=10, random_state=42)
model_10.fit(X_train, y_train)
y_pred_10 = model_10.predict(X_test)
accuracy_10 = accuracy_score(y_test, y_pred_10)
conf_matrix_10 = confusion_matrix(y_test, y_pred_10)

```

```

print("Accuracy with 10 estimators:", round(accuracy_10, 4))
print("Confusion Matrix (10 estimators):\n", conf_matrix_10)

# Step 5: Fine-tune number of trees (1 to 50)
best_accuracy = 0
best_n = 0
accuracies = []

for n in range(1, 51):
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n

print(f"\nBest Accuracy: {round(best_accuracy, 4)} with n_estimators = {best_n}")

# Step 6: Plot accuracy vs. number of estimators
plt.figure(figsize=(10, 6))
plt.plot(range(1, 51), accuracies, marker='o', linestyle='-', color='blue')
plt.title('Accuracy vs Number of Trees (n_estimators)')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.tight_layout()
plt.show()

```

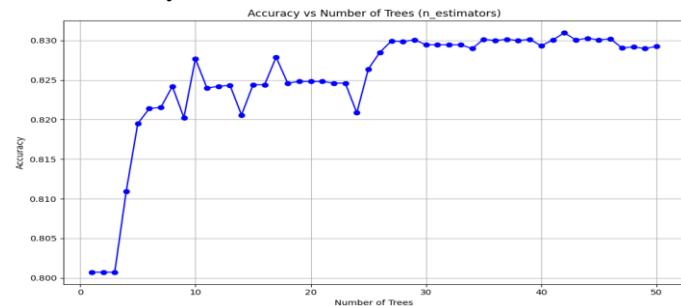
Accuracy with 10 estimators: 0.8277

Confusion Matrix (10 estimators):

`[[10722 387]`

`[2138 1406]]`

Best Accuracy: 0.831 with n_estimators = 42



Program 10

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

Screenshot

Lab. 9

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

Compute two clusters using k-means algorithm for clustering where initial cluster centers are $(1.0, 1.0)$ and $(5.0, 7.0)$. Execute for two iterations.

→ Iteration 1:

| Record Number | Class to C ₁ $(1.0, 1.0)$ | Class to C ₂ $(5.0, 7.0)$ | Assign to cluster |
|---------------------------|---|---|-------------------|
| R ₁ (1.0, 1.0) | 0.0 | 7.21 | C ₁ |
| R ₂ (1.5, 2.0) | 1.12 | 6.12 | C ₂ |
| R ₃ (3.0, 4.0) | 3.61 | 3.61 | C ₁ |
| R ₄ (5.0, 7.0) | 7.21 | 0.0 | C ₂ |
| R ₅ (3.5, 5.0) | 4.12 | 2.5 | C ₁ |
| R ₆ (4.5, 5.0) | 5.31 | 2.06 | C ₂ |
| R ₇ (3.5, 4.5) | 4.30 | 2.92 | C ₂ |

Cluster 1 {R₁, R₂, R₃} 2 Cluster 2 {R₃, R₄, R₅, R₆, R₇}

Their new centroids are :-

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

$$= 5.5/3 = 1.83, 7.0/3 = 2.33$$

$$C_2 = \frac{(5.0 + 3.5 + 4.5 + 2.5)}{4}, \frac{(7.0 + 5.0 + 4.0)}{4}$$

$$= 16.5/4 = 4.12, 21.5/4 = 5.37$$

Iteration 2:

| Record Number | Class to C ₁ $(1.0, 1.0)$ | Class to C ₂ $(5.0, 7.0)$ | Assign to cluster |
|---------------------------|---|---|-------------------|
| R ₁ (1.0, 1.0) | 1.87 | 5.64 | C ₁ |
| R ₂ (1.5, 2.0) | 0.47 | 4.52 | C ₁ |
| R ₃ (3.0, 4.0) | 2.12 | 1.63 | C ₂ |
| R ₄ (5.0, 7.0) | 5.87 | 1.31 | C ₁ |
| R ₅ (3.5, 5.0) | 3.16 | 0.72 | C ₂ |
| R ₆ (4.5, 5.0) | 3.58 | 0.62 | C ₂ |
| R ₇ (3.5, 4.5) | 2.63 | 1.05 | C ₂ |

Clusters 1 {R₁, R₂, R₃} and Cluster 2 {R₃, R₄, R₅, R₆, R₇}.

Their new centroids are -

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

$$= 5.5/3 = 1.83, 7.0/3 = 2.33$$

$$C_2 = \frac{(5.0 + 3.5 + 4.5 + 2.5)}{4}, \frac{(7.0 + 5.0 + 4.0)}{4}$$

$$= 16.5/4 = 4.12, 21.5/4 = 5.37$$

Write the answer for the following question -

- For "iris.csv" dataset, draw the elbow plot. What was the optimal k-value obtained?

Elbow plot for k-means clustering

Sum of squared distances vs K

K=3

Code:

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

```

# Load the dataset
df = pd.read_csv("/content/iris (2).csv")

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

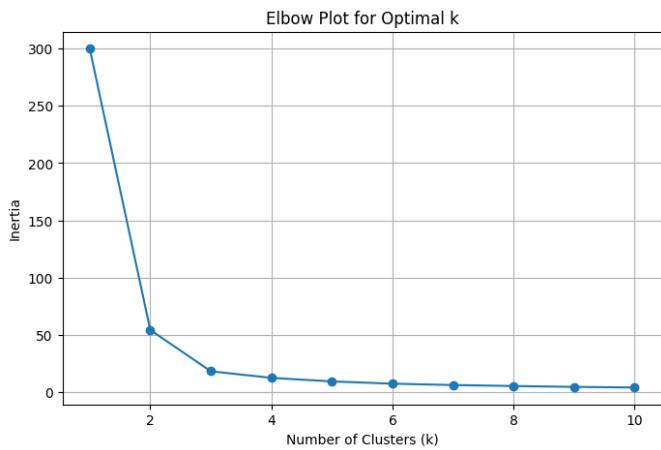
# Optional: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow method to determine optimal k
inertia = []
k_range = range(1, 11)

for k in k_range:
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    model.fit(X_scaled)
    inertia.append(model.inertia_)

# Plot the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Plot for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()

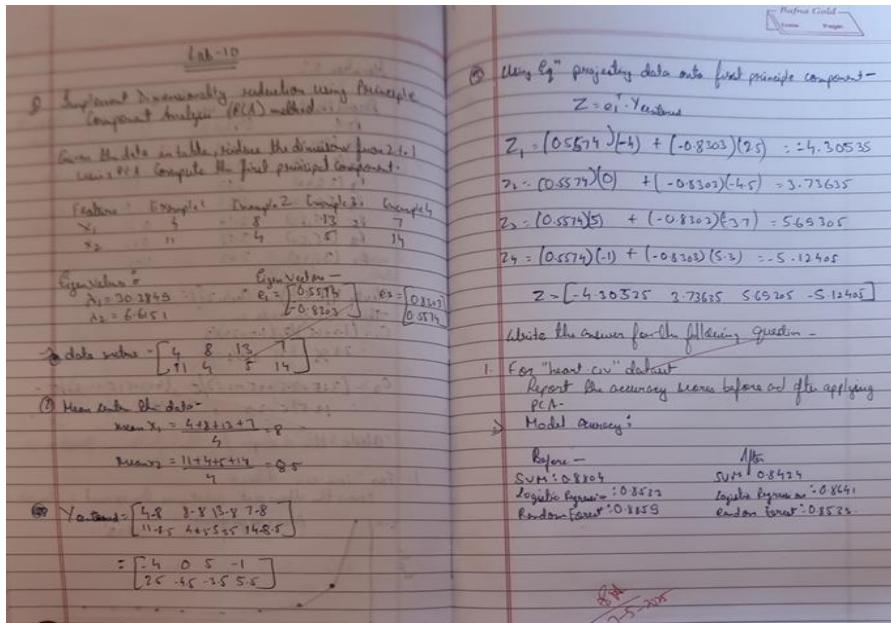
```



Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot



Code:

```

import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv("/content/heart (1).csv") # Update to match your file path if needed

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

# Identify categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

# Encode categorical columns
for col in categorical_cols:
    if X[col].nunique() == 2:

```

```

X[col] = LabelEncoder().fit_transform(X[col])
else:
    X = pd.get_dummies(X, columns=[col])

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
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()
}

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

# Apply PCA (reduce to 5 components)
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Train and evaluate models (with PCA)
print("\n✖️ Accuracy with PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train_pca)
    y_pred_pca = model.predict(X_test_pca)
    print(f"{name}: {accuracy_score(y_test_pca, y_pred_pca):.4f}")

🔍 Accuracy without PCA:
SVM: 0.8804
Logistic Regression: 0.8533
Random Forest: 0.8859

✖️ Accuracy with PCA:
SVM: 0.8424
Logistic Regression: 0.8641
Random Forest: 0.8533

```