VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

PRANAV SRINIVAS (1BM22CS203)

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

February 2025 — June 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 **Pranav Srinivas (1BM22CS203),** 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.

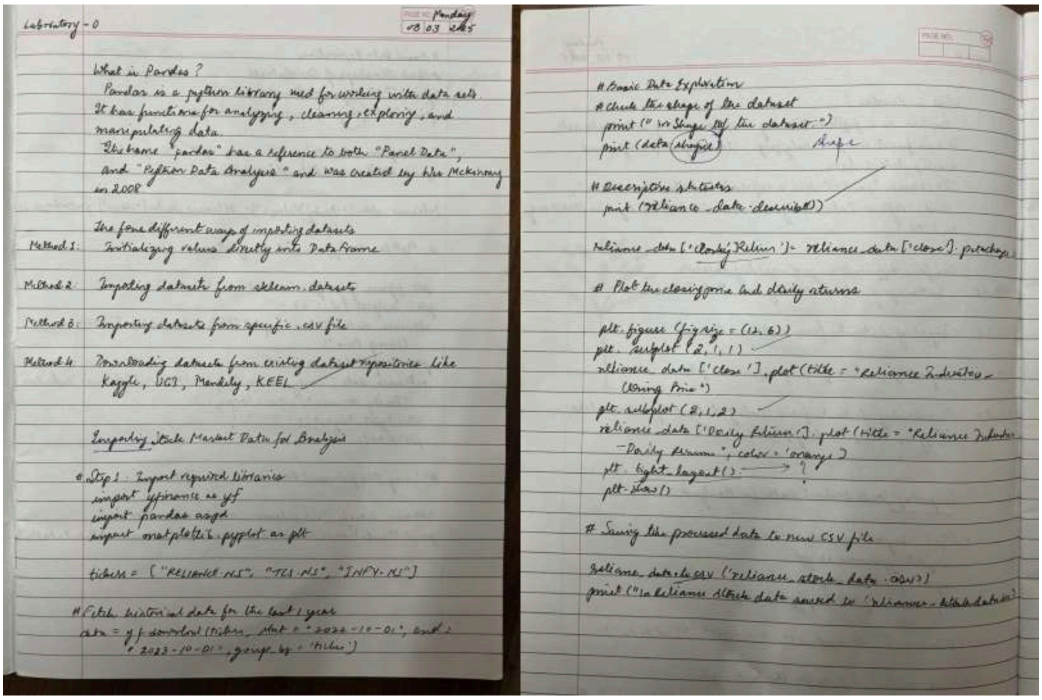
Prof. Geetha N Assistant Professor, Department of CSE, BMSCE.	**Dr. Kavitha Sooda** Professor & Head, Department of CSE, BMSCE.

Index

Github Link: https://github.com/pranavsrinivasdof/MACHINE-LEARNING-LAB

Program 1: Write a python program to import and export data using Pandas library functions

Observation Snapshot:


Laboratory - 0

Monday 03 03 2025

What is Pandas?

Pandas is a python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008

The four different ways of importing datasets

Method 1: Initializing values directly into DataFrame

Method 2: Importing datasets from sklearn.datasets

Method 3: Importing datasets from specific .csv file

Method 4: Downloading datasets from existing dataset repositories like Kaggle, UCI, Mendely, KEEL

Importing Stock Market Data for Analysis

Step 1: Import required libraries
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

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

# Fetch historical data for the last 1 year
data = yf.download(tickers, start = "2022-10-01", end = "2023-10-01", group_by = 'ticker')

# Basic Data Exploration
# Check the shape of the dataset
print("\n Shape of the dataset:")
print(data.shape)

# Descriptive statistics
print(reliance_data.describe())

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

Plot the closing price and daily returns

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

Saving the processed data to new CSV file

reliance_data.to_csv('reliance_stock_data.csv')
print("\n Reliance stock data saved to 'reliance_stock_data.csv'")

Code:

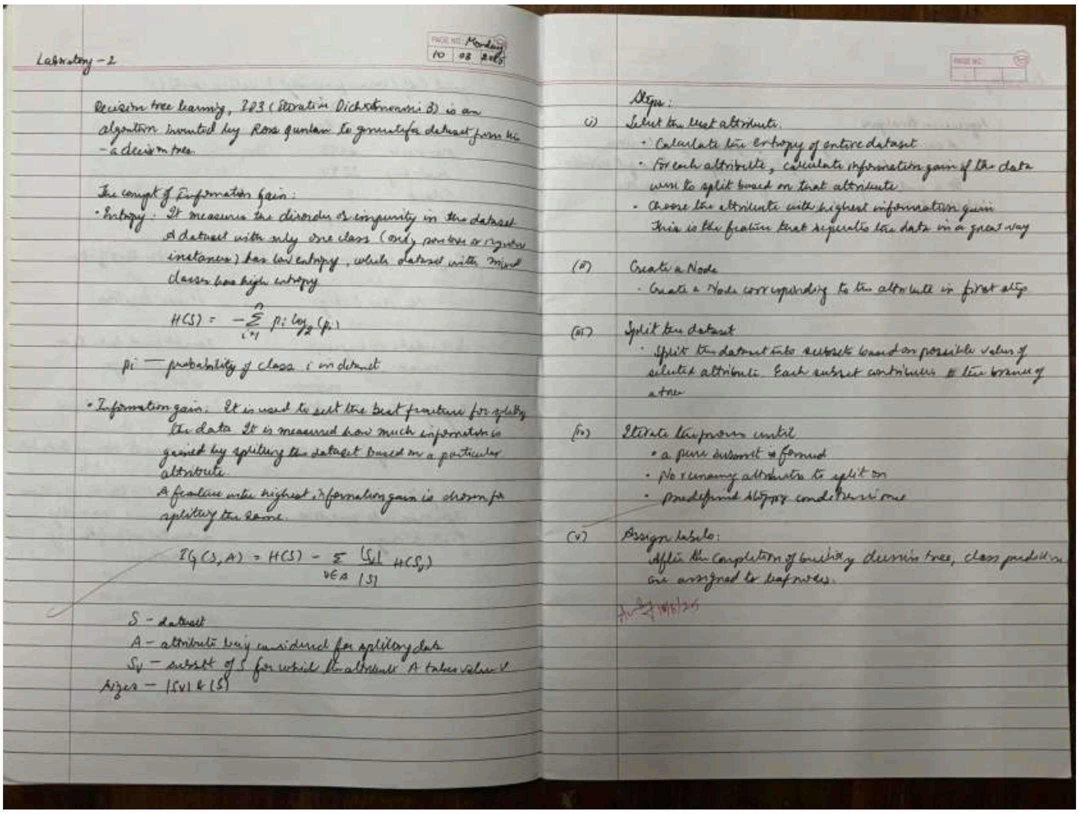
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats

#**Diabetes Dataset**
df=pd.read_csv('/content/Dataset of Diabetes .csv')
df.head()
df.shape
print(df.info())
# Summary statistics
print(df.describe())
missing_values=df.isnull().sum()
print(missing_values[missing_values > 0])
categorical_cols = df.select_dtypes(include=['object']).columns
```

```
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd

numerical_cols = df.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_minmax = df.copy()  # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df[numerical_cols])

scaler = StandardScaler()
df_standard = df.copy()
df_standard[numerical_cols] = scaler.fit_transform(df[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())

#**Adult Income Dataset**
df1=pd.read_csv('/content/adult.csv')
df1.head()
df1.shape
print(df1.info())
# Summary statistics
print(df.describe())
missing_values=df1.isnull().sum()
print(missing_values[missing_values > 0])
categorical_cols = df1.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
    df1 = pd.get_dummies(df1, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd
```

```
numerical_cols = df1.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_minmax = df1.copy()  # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df1[numerical_cols])

scaler = StandardScaler()
df_standard = df1.copy()
df_standard[numerical_cols] = scaler.fit_transform(df1[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())
```

Program 2: Demonstrate various data pre-processing techniques for a given dataset

Observation Snapshot:



Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv('housing.csv')
df.head(2)
df.describe()
df.info()
sns.histplot(df['median_income'], kde=True, color='green')
sns.histplot(df['housing_median_age'])
from sklearn.model_selection import train_test_split

X = df.drop("median_house_value", axis=1)
y = df["median_house_value"]

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

```
X = df.drop("median_house_value", axis=1)
y = df["median_house_value"]
df["income_cat"] = pd.cut(df["median_house_value"],
bins=[0, 100000, 200000, 300000, 400000, np.inf],
labels=[1, 2, 3, 4, 5])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=df["income_cat"])
train_set = X_train.copy()
train_set["median_house_value"] = y_train
train_set.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,s=train_set["population"]/100,
label="population",figsize=(10,7), c="median_house_value", cmap=plt.get_cmap("jet"),
colorbar=True)
plt.legend()
numerical_columns = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numerical_columns.corr()
print(correlation_matrix["median_house_value"].sort_values(ascending=False))
df.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
# Combine 'median_income' and 'households'
df["income_households"] = df["median_income"] * df["households"]

numerical_columns = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numerical_columns.corr()
print(correlation_matrix["median_house_value"].sort_values(ascending=False))
df.plot(kind="scatter", x="income_households", y="median_house_value", alpha=0.1)
plt.show()
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
h=df
h.dropna(subset=["total_bedrooms"])
from sklearn.preprocessing import OneHotEncoder
df1=pd.read_csv('housing.csv')
hc=df1[["ocean_proximity"]]
encoder=OneHotEncoder()
hc_encoded=encoder.fit_transform(hc).toarray()
hc_1hot_df = pd.DataFrame(hc_encoded, columns=encoder.get_feature_names_out(hc.columns))
hc_1hot_df.head()
```

Feature scaling is crucial in machine learning for several reasons, particularly when using algorithms that are sensitive to the scale of features. Here's a breakdown of its importance:

1. **Improved Performance of Distance-Based Algorithms:**

2. **Faster Convergence of Gradient Descent:**

3. **Improved Regularization:**

4. **Better Interpretation of Coefficients:**

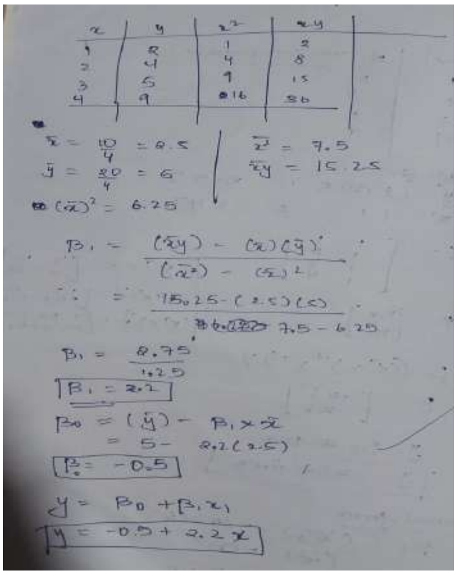
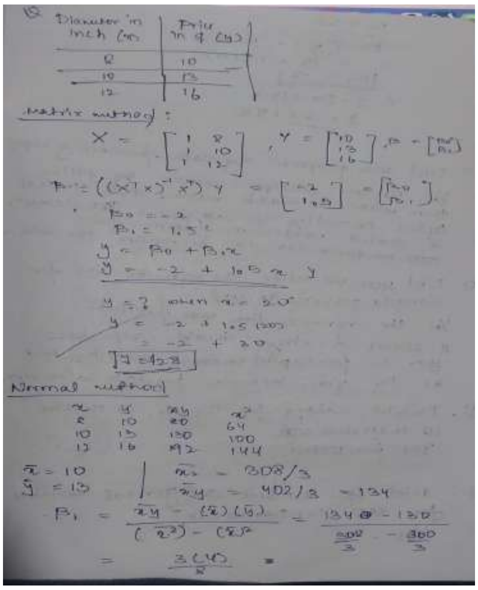
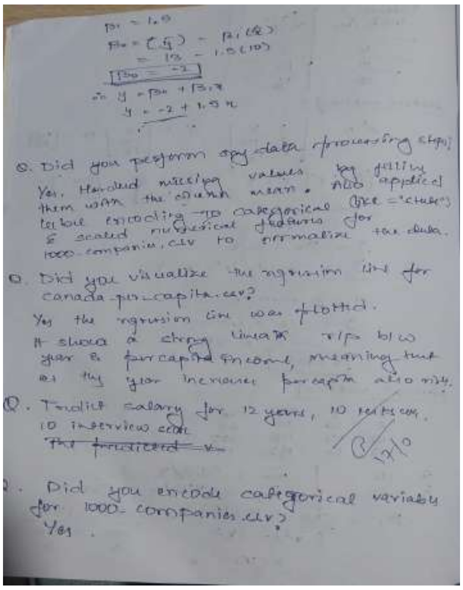
5. **Numerical Stability:**

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

# Custom transformer to add engineered attributes
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        # Assumes X is a NumPy array with the following columns:
        # total_rooms (index 3), total_bedrooms (index 2), population (index 4), households (index 5)
        rooms_per_household = X[:, 3] / X[:, 5]
        population_per_household = X[:, 4] / X[:, 5]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, 2] / X[:, 3]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

# Identify numerical and categorical columns
num_attribs = df1.drop("ocean_proximity", axis=1).columns  # All numeric columns
cat_attribs = ["ocean_proximity"]

# Build numerical pipeline: impute missing values, add new attributes, then scale
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

# Build the full pipeline combining numerical and categorical processing
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])
```

```
# Process the dataset using the pipeline
housing_prepared = full_pipeline.fit_transform(housing)
print("Shape of processed data:", housing_prepared.shape)
```

Program 3: Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Observation Snapshot:

Code:

```
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

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

```
# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')

new_df = df.drop('price',axis='columns')
new_df

price = df.price
price

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,price)

"""(1) Predict price of a home with area = 3300 sqr ft"""

reg.predict([[3300]])

reg.coef_

reg.intercept_

"""Y = m * X + b (m is coefficient and b is intercept)"""

3300*135.78767123 + 180616.43835616432

"""(1) Predict price of a home with area = 5000 sqr ft"""

reg.predict([[5000]])
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from sklearn import linear_model

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

"""Data Preprocessing: Fill NA values with median value of a column"""

df.bedrooms.median()

df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
```

```
df

reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)

reg.coef_

reg.intercept_

"""Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old"""

reg.predict([[3000, 3, 40]])

112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384
import pandas as pd
from sklearn.linear_model import LinearRegression

# Load the dataset
df1 = pd.read_csv('/content/canada_per_capita_income.csv')

# Prepare the data
X = df1.year.values.reshape(-1, 1)  # Features (year)
y = df1['per capita income (US$)']   # Target (per capita income)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict per capita income for 2020
year_2020 = [[2020]]
predicted_income = model.predict(year_2020)

print(f"Predicted per capita income for Canada in 2020: {predicted_income[0]:.2f}")

import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load the dataset (canada_per_capita_income.csv)
df1 = pd.read_csv('/content/canada_per_capita_income.csv')

# Prepare the data
X = df1.year.values.reshape(-1, 1)  # Features (year)
y = df1['per capita income (US$)']   # Target (per capita income)
```

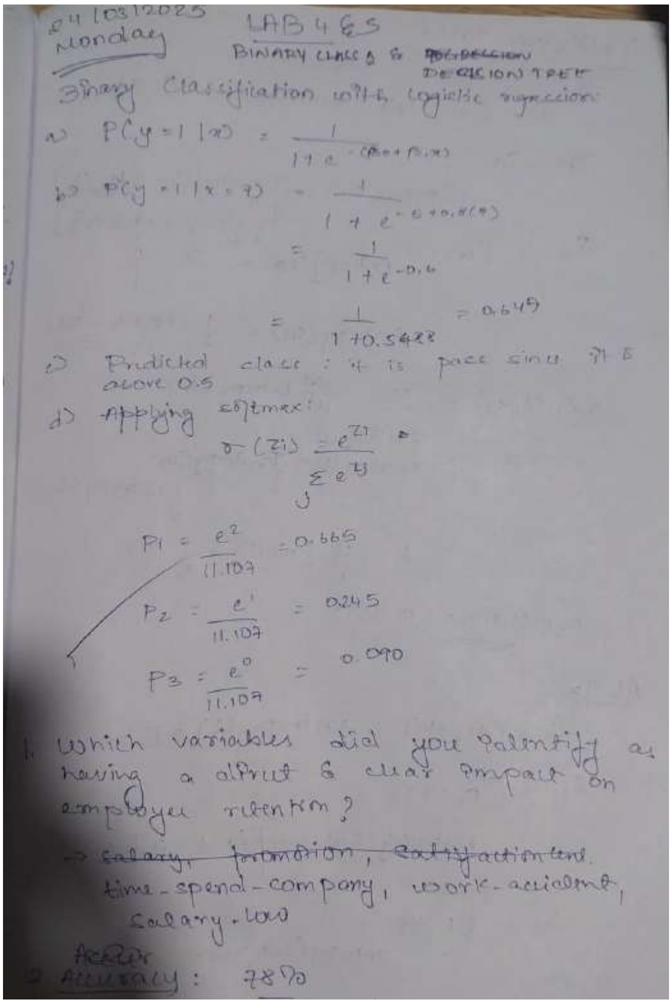
```
# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Create the plot
plt.figure(figsize=(8, 6))
plt.scatter(X, y, color='blue', label='Data Points') # Now using the correct X and y
plt.plot(X, model.predict(X), color='red', label='Regression Line')
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)')
plt.title('Per Capita Income in Canada over Time')
plt.legend()
plt.grid(True)
plt.show()
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv('/content/salary.csv')

# Prepare the data
X = df.iloc[:, :-1].values  # Features (years of experience)
y = df.iloc[:, 1].values    # Target (salary)

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean') # Create an imputer object with strategy as mean
X = imputer.fit_transform(X) # Fit and transform the imputer on feature data 'X'

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict salary for 12 years of experience
years_experience = [[12]]
predicted_salary = model.predict(years_experience)

print(f"Predicted salary for 12 years of experience: {predicted_salary[0]:.2f}")
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

# Load the dataset
```

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

# Handle missing values
# Convert 'experience' column to numeric, replacing non-numeric with NaN
df['experience'] = pd.to_numeric(df['experience'], errors='coerce')

imputer = SimpleImputer(strategy='mean')
df['experience'] = imputer.fit_transform(df[['experience']])
df['test_score(out of 10)'] = imputer.fit_transform(df[['test_score(out of 10)']])

# Prepare the data
X = df.drop('salary($)', axis='columns')
y = df['salary($)']

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict salaries for the given candidates
candidate1 = [[2, 9, 6]]
candidate2 = [[12, 10, 10]]

predicted_salary1 = model.predict(candidate1)
predicted_salary2 = model.predict(candidate2)

print(f"Predicted salary for candidate 1: ${predicted_salary1[0]:.2f}")
print(f"Predicted salary for candidate 2: ${predicted_salary2[0]:.2f}")
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Load the dataset
df = pd.read_csv('/content/1000_Companies.csv')

# Separate features (X) and target (y)
X = df.iloc[:, :-1].values
y = df.iloc[:, 4].values

# Encode categorical data (State)
labelencoder = LabelEncoder()
X[:, 3] = labelencoder.fit_transform(X[:, 3])
```

```
ct = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(), [3])],
    remainder='passthrough'
)
X = ct.fit_transform(X)

# Avoid dummy variable trap (remove one encoded column)
X = X[:, 1:]

# 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=0)

# Create and train the multiple linear regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict profit for the given values
new_prediction = regressor.predict([[1, 0, 91694.48, 515841.3, 11931.24]])
print(f"Predicted Profit: {new_prediction[0]:.2f}")
```

Program 4: Build Logistic Regression Model for a given dataset

Observation Snapshot:



24/03/2025
Monday

LAB 4 & 5
BINARY CLASS & ~~REGRESSION~~ DECISION TREE

Binary Classification with logistic regression

a) $P(y=1 \mid x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$

b) $P(y=1 \mid x=7) = \frac{1}{1+e^{-5+0.8(7)}}$

$= \frac{1}{1+e^{-0.6}}$

$= \frac{1}{1+0.5488} = 0.645$

c) Predicted class : it is pass since it is above 0.5

d) Applying softmax:

$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$

$P_1 = \frac{e^2}{11.107} = 0.665$

$P_2 = \frac{e^1}{11.107} = 0.245$

$P_3 = \frac{e^0}{11.107} = 0.090$

1. Which variables did you identify as having a direct & clear impact on employee retention?

→ ~~salary, promotion, satisfaction level~~
time-spend-company, work-accident, salary-low

2. ~~Accur~~ Accuracy : 78%

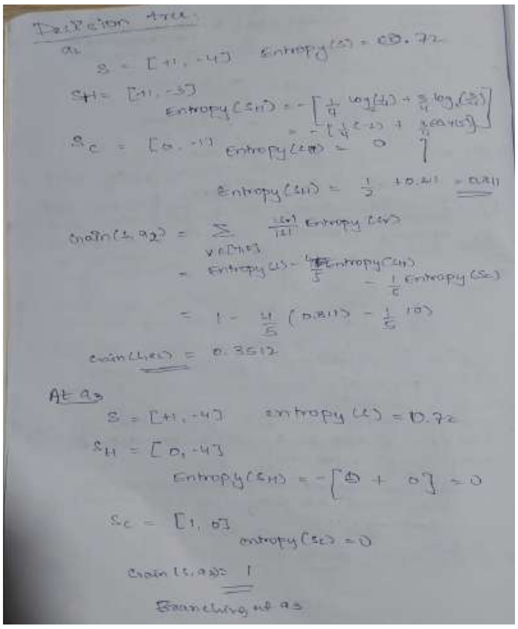
Code:

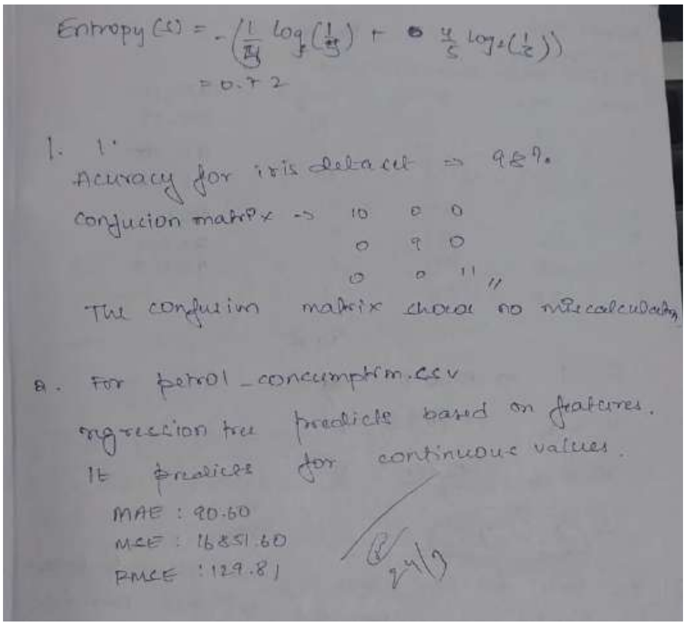
```
import pandas as pd
import numpy as np
df=pd.read_csv("/content/HR_comma_sep.csv")
df.head(3)
print(df.isnull().sum())
print(df.groupby('left').mean(numeric_only=True))
print(df.groupby('salary').mean(numeric_only=True))
import matplotlib.pyplot as plt
pd.crosstab(df.salary,df.left).plot(kind='bar')
plt.title('Employee Retention vs Salary')
plt.xlabel('Salary')
plt.ylabel('Number of Employees')
plt.show()
```

```
pd.crosstab(df.Department,df.left).plot(kind='bar')
plt.title('Employee Retention vs Department')
plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.show()
salary_dummies = pd.get_dummies(df.salary, prefix="salary")
dept_dummies = pd.get_dummies(df.Department, prefix="dept")

df_with_dummies = pd.concat([df, salary_dummies, dept_dummies], axis=1)

df_with_dummies = df_with_dummies.drop(['salary', 'Department'], axis=1)

X_features = ['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours',
'time_spend_company', 'Work_accident', 'promotion_last_5years'] + list(salary_dummies.columns) +
list(dept_dummies.columns)
X = df_with_dummies[X_features]
y = df_with_dummies.left


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the model:", accuracy)
```

Program 5: Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Observation Snapshot:





Code:

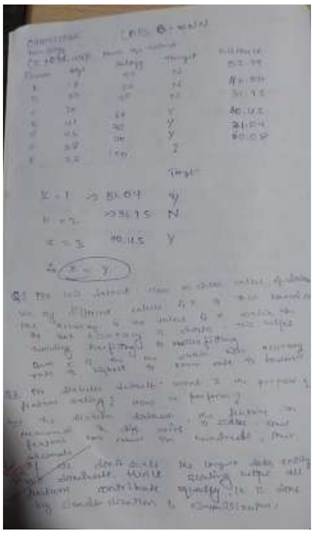
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import tree
import matplotlib.pyplot as plt

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

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

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)


plt.figure(figsize=(12, 8))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import tree
import matplotlib.pyplot as plt

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

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

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)


plt.figure(figsize=(12, 8))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np # import numpy
```

```
data = pd.read_csv("petrol_consumption.csv")

X = data[['Petrol_tax', 'Average_income', 'Paved_Highways',
        'Population_Driver_licence(%)']]
y = data['Petrol_Consumption']

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

regressor = DecisionTreeRegressor()

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))
# Assuming 'data' is your original pandas DataFrame
plot_tree(regressor, feature_names=data[['Petrol_tax', 'Average_income', 'Paved_Highways',
'Population_Driver_licence(%)']].columns, filled=True, rounded=True)
plt.show()
```

Program 6: Build K-Nearest Neighbours Classification model for a given dataset.

Observation Snapshot:



Code:

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

try:
    data = pd.read_csv('/content/iris (1).csv')
except FileNotFoundError:
    print("Error: 'iris.csv' not found. Please upload the file to your Colab environment.")
    exit()

X = data.drop('species', axis=1)
y = data['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)

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

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

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

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

try:
    diabetes = pd.read_csv('diabetes.csv')
except FileNotFoundError:
    print("Error: 'diabetes.csv' not found. Please ensure the file is in the current directory.")
    exit()

X = diabetes.drop('Outcome', axis=1)
y = diabetes['Outcome']

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
sns.heatmap(cm, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))

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

try:
  heart = pd.read_csv('heart.csv')
except FileNotFoundError:
  print("Error: 'heart.csv' not found. Please ensure the file is in the current directory.")
  exit()

X = heart.drop('target', axis=1)
y = heart['target']

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
best_k = 1
best_accuracy = 0

for k in range(1, 21):
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(X_train, y_train)
  y_pred = knn.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred)
  if accuracy > best_accuracy:
    best_accuracy = accuracy
    best_k = k

print(f"Best k: {best_k} with accuracy {best_accuracy}")
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
```

```
print(cm)

sns.heatmap(cm, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report:")
print(classification_report(y_test, y_pred))

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
print(classification_report(y_test, y_pred))

# prompt: For Iris dataset
# How to choose the k value? Demonstrate using accuracy rate and error
# rate. Give theory

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

# Load the Iris dataset
try:
    data = pd.read_csv('/content/iris (1).csv')
except FileNotFoundError:
    print("Error: 'iris (1).csv' not found. Please upload the file to your Colab environment.")
    exit()

# Prepare the data
```

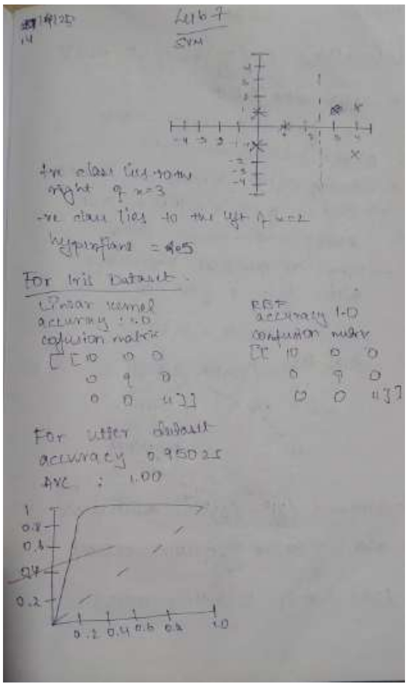
```
X = data.drop('species', axis=1)
y = data['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the data (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the optimal k value
error_rates = []
for k in range(1, 31):  # Test k values from 1 to 30
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    error_rates.append(1 - accuracy_score(y_test, y_pred)) # Error rate = 1 - accuracy

# Plot error rates
plt.figure(figsize=(10, 6))
plt.plot(range(1, 31), error_rates, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()


# Theory for choosing k:
# The optimal 'k' value minimizes the error rate.
# Very small k (e.g., 1) can lead to overfitting, being too sensitive to noise.
# Very large k (e.g., 30) can lead to underfitting, smoothing out the decision boundaries too much.
# We seek a k that balances these extremes, as shown by the error rate plot.

#Select k based on the minimum error rate observed in the plot
best_k = error_rates.index(min(error_rates)) + 1  #Add 1 as the index starts from 0


# Train and evaluate the model with the best k
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv('/content/iris (1).csv')
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Store accuracy and error rate
accuracy = []
error_rate = []

# Try k from 1 to 20
for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    preds = knn.predict(X_test)
    acc = accuracy_score(y_test, preds)
    accuracy.append(acc)
    error_rate.append(1 - acc)

# Plot
plt.figure(figsize=(10,5))
plt.plot(range(1, 21), accuracy, label='Accuracy')
```

```
plt.plot(range(1, 21), error_rate, label='Error Rate')
plt.xlabel('K Value')
plt.ylabel('Rate')
plt.title('K vs Accuracy and Error Rate')
plt.legend()
plt.show()

import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load data
df = pd.read_csv('/content/diabetes.csv')
X = df.drop('Outcome', axis=1)  # Features
y = df['Outcome']               # Target

# Perform scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert back to DataFrame (optional)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
```

Program 7: Build Support vector machine model for a given dataset

Observation Snapshot:



Code:

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

positive_class = np.array([[4, 1], [4, -1], [6, 0]])
negative_class = np.array([[1, 0], [0, 1], [0, -1]])

plt.figure(figsize=(8, 6))
plt.scatter(positive_class[:, 0], positive_class[:, 1], color='red', label='Positive Class', s=100,
edgecolors='black')
plt.scatter(negative_class[:, 0], negative_class[:, 1], color='blue', label='Negative Class', s=100,
edgecolors='black')

all_points = np.concatenate([positive_class, negative_class])
labels = ["(4,1)", "(4,-1)", "(6,0)", "(1,0)", "(0,1)", "(0,-1)"]

for i, txt in enumerate(labels):
    plt.annotate(txt, (all_points[i][0], all_points[i][1]), textcoords="offset points", xytext=(0,5),
ha='center', fontsize=10)

x_values = np.linspace(-1, 7, 100)
y_values = np.zeros_like(x_values)
```

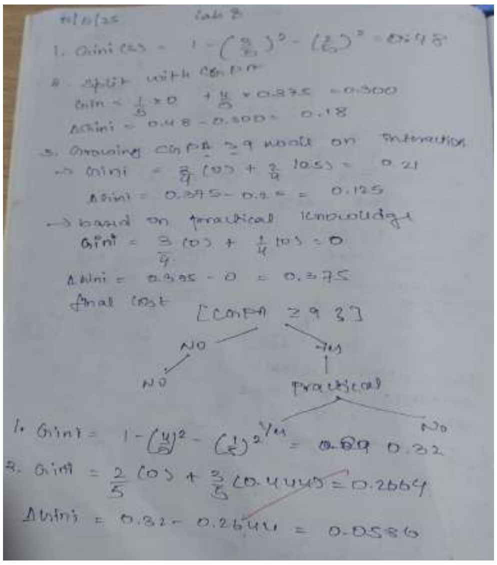
```
plt.plot(x_values, y_values, color='black', linestyle='--', label='Optimal Hyperplane (y = 0)')

plt.plot(x_values, y_values + 1, color='gray', linestyle=':', label='Margin at y = 1')
plt.plot(x_values, y_values - 1, color='gray', linestyle=':', label='Margin at y = -1')

plt.title('Optimal Hyperplane for SVM (Visual Approximation)', fontsize=14)
plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim(-1, 7)
plt.ylim(-2, 2)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.legend()

plt.grid(True)
plt.show()
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('/content/iris (1) (1).csv')

X = data.drop('species', axis=1)
y = data['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
cm_rbf = confusion_matrix(y_test, y_pred_rbf)

print("SVM with RBF Kernel:")
print("Accuracy:", accuracy_rbf)
print("Confusion Matrix:\n", cm_rbf)

plt.figure(figsize=(6, 4))
sns.heatmap(cm_rbf, annot=True, fmt='d', cmap='Blues',
        xticklabels=data['species'].unique(),
        yticklabels=data['species'].unique())
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (RBF Kernel)')
plt.show()

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
cm_linear = confusion_matrix(y_test, y_pred_linear)

print("\nSVM with Linear Kernel:")
print("Accuracy:", accuracy_linear)
print("Confusion Matrix:\n", cm_linear)

plt.figure(figsize=(6, 4))
sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Blues',
            xticklabels=data['species'].unique(),
            yticklabels=data['species'].unique())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (Linear Kernel)')
plt.show()
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc
import seaborn as sns
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier

data = pd.read_csv('/content/letter-recognition.csv')  # Replace with the correct path if necessary

X = data.drop('letter', axis=1)
y = data['letter']

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

svm_classifier = SVC(kernel='rbf', probability=True) # probability=True is needed for ROC curve
svm_classifier.fit(X_train, y_train)

y_pred = svm_classifier.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("SVM Classifier:")
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", cm)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y),
yticklabels=np.unique(y))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

classifier = OneVsRestClassifier(SVC(kernel='rbf', probability=True))
classifier.fit(X_train, y_train)
y_score = classifier.predict_proba(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure(figsize=(8, 6))
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
              ''.format(roc_auc["micro"]))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Micro-averaged ROC Curve')
plt.legend(loc="lower right")
plt.show()
print(f"Micro-averaged AUC: {roc_auc['micro']}")
```

Program 8: Implement Random forest ensemble method on a given dataset.

Observation Snapshot:



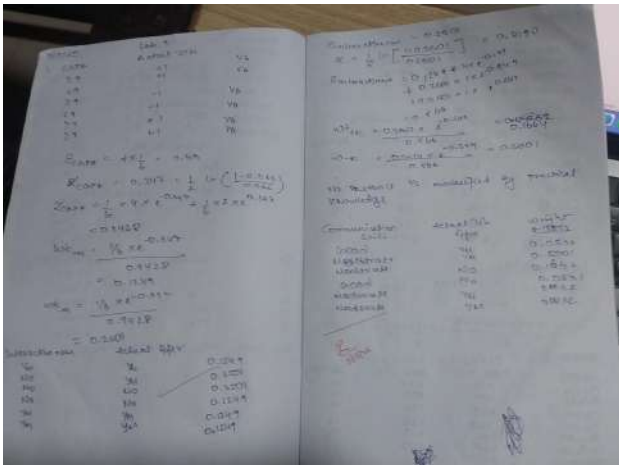
Code:

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

# Load the dataset
df = pd.read_csv('/content/iris (1).csv')

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

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

# Build Random Forest 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)

# Measure accuracy
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default RF accuracy (n_estimators=10): {default_score:.4f}")
```

```
# Fine-tune the number of trees
scores = []
n_range = range(1, 101)

for n in n_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
# Find the best score and number of trees
best_score = max(scores)
best_n = n_range[scores.index(best_score)]
print(f"Best RF accuracy: {best_score:.4f} with n_estimators={best_n}")
# Optional: Plot accuracy vs number of estimators
plt.figure(figsize=(10, 6))
plt.plot(n_range, scores, marker='o')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

Program 9: Implement Boosting ensemble method on a given dataset.

Observation Snapshot:



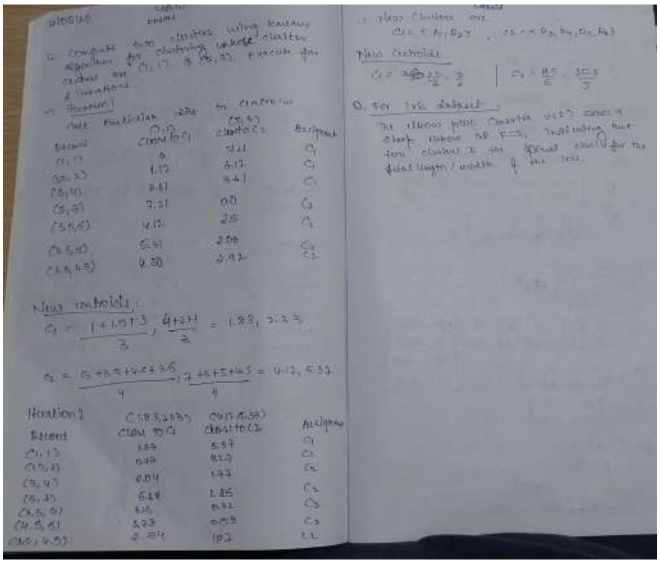
Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Load dataset
df = pd.read_csv("/content/income.csv")
# Drop rows with missing values
df.dropna(inplace=True)
# Encode categorical columns
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
# Separate features and target
X = df.drop(columns=['income_level'], errors='ignore', axis=1)
y = df['income_level']
# Split 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)
# 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)
score_10 = accuracy_score(y_test, y_pred_10)
print(f"Accuracy with 10 estimators: {score_10:.4f}")
# Fine-tune number of estimators
best_score = 0
best_n = 0
estimators_range = list(range(10, 201, 10))
scores = []
for n in estimators_range:
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
    print(f"n_estimators={n}, Accuracy={score:.4f}")
    if score > best_score:
        best_score = score
        best_n = n
print(f"\nBest Accuracy: {best_score:.4f} using {best_n} estimators")
# Plot accuracy vs number of estimators
plt.figure(figsize=(7, 4))
plt.plot(estimators_range, scores, marker='o', linestyle='-', color='blue')
plt.title("Accuracy vs Number of Estimators (AdaBoost)")
plt.xlabel("Number of Estimators (Trees)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.xticks(estimators_range)
plt.tight_layout()
plt.show()
```

Program 10: Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Observation Snapshot:



Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from scipy.stats import mode
import matplotlib.pyplot as plt

# Step 1: Generate sample data and save to CSV
np.random.seed(42)
names = [f"Person_{i}" for i in range(50)]
ages = np.random.randint(20, 60, 50)
income = np.random.randint(30000, 120000, 50)

df = pd.DataFrame({'Name': names, 'Age': ages, 'Income': income})
df.to_csv("income.csv", index=False)

# Step 2: Load the data
data = pd.read_csv("income.csv")

# Drop 'Name' and extract features
X = data[['Age', 'Income']]
```

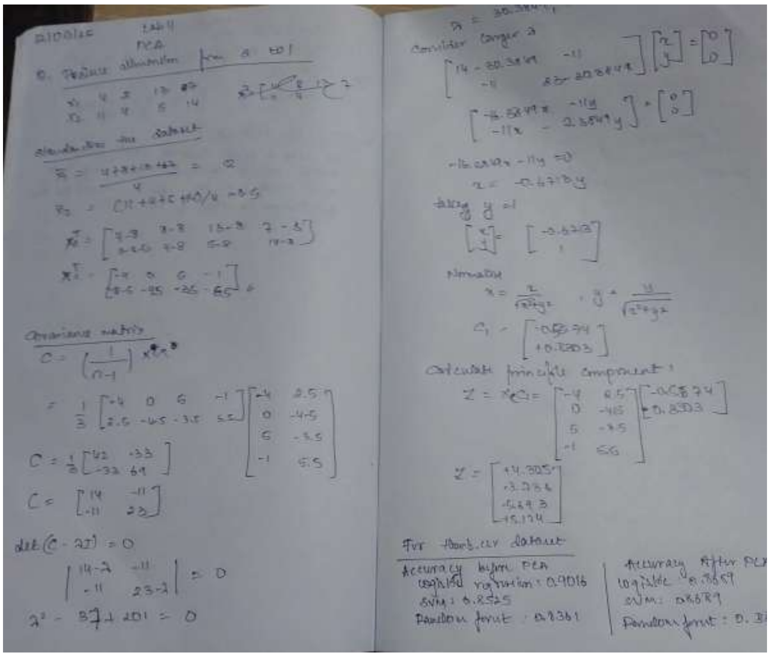
```
# Step 3: Split the data
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Step 4: Perform scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 5: Plot SSE vs number of clusters (Elbow method)
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
    sse.append(kmeans.inertia_)

plt.figure(figsize=(8, 4))
plt.plot(k_range, sse, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('SSE (Inertia)')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()

# Step 6: Choose optimal number of clusters (say 3) and fit model
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train_scaled)

# Predict on test data
predictions = kmeans.predict(X_test_scaled)

# Note: There's no ground truth labels, but for demonstration,
# we can try assigning true clusters (via KMeans on full data)
# and see if predicted clusters align

# Fit on full data to assign pseudo-labels
full_kmeans = KMeans(n_clusters=optimal_k, random_state=42)
true_clusters = full_kmeans.fit_predict(scaler.fit_transform(X))

# Align predicted clusters using majority voting (only for demonstration)
# Match predicted labels to closest true labels
def map_clusters(true_labels, pred_labels):
    labels = np.zeros_like(pred_labels)
```

```
    for i in range(optimal_k):
        mask = (pred_labels == i)
        if np.sum(mask) == 0:
            continue
        labels[mask] = mode(true_labels[mask])[0]
    return labels

mapped_preds = map_clusters(true_clusters[X_test.index], predictions)
accuracy = accuracy_score(true_clusters[X_test.index], mapped_preds)
print(f"Approximate Clustering Accuracy: {accuracy:.2f}")

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

# Step 1: Load Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Keep only petal length and petal width
X = df[['petal length (cm)', 'petal width (cm)']].values

# Step 2: Check impact of scaling
# Try without scaling
sse_unscaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    sse_unscaled.append(kmeans.inertia_)

# Now scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

sse_scaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse_scaled.append(kmeans.inertia_)
```

```
# Step 3: Plot Elbow Comparison (Scaled vs Unscaled)
plt.figure(figsize=(10, 5))

plt.plot(range(1, 11), sse_unscaled, marker='o', label='Unscaled')
plt.plot(range(1, 11), sse_scaled, marker='s', label='Scaled')
plt.title('Elbow Method (Petal Features Only)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('SSE (Inertia)')
plt.legend()
plt.grid(True)
plt.show()
```

Program 11: Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Observation Snapshot:



Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
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

# 1. Load data
df = pd.read_csv("heart.csv")

# 2. Label-encode binary text columns
le = LabelEncoder()
for col in ["Sex", "ExerciseAngina"]:
    df[col] = le.fit_transform(df[col])

# 3. Separate features and target
```

```
X = df.drop("HeartDisease", axis=1)
y = df["HeartDisease"]

# 4. Build preprocessing pipeline:
#    - One-hot for multi-category columns (using sparse_output=False)
#    - passthrough the rest
#    - then scale everything
cat_cols = ["ChestPainType", "RestingECG", "ST_Slope"]
preprocessor = Pipeline([
    ("onehot", ColumnTransformer([
        ("ohe", OneHotEncoder(sparse_output=False, drop="first"), cat_cols)
    ], remainder="passthrough")),
    ("scaler", StandardScaler())
])

# 5. Apply preprocessing
X_proc = preprocessor.fit_transform(X)

# 6. Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_proc, y, test_size=0.2, random_state=42
)

# 7. Define models
models = {
    "SVM": SVC(random_state=42),
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42)
}

# 8. Train & evaluate before PCA
print("=== Accuracies BEFORE PCA ===")
scores_before = {}
for name, clf in models.items():
    clf.fit(X_train, y_train)
    preds = clf.predict(X_test)
    acc = accuracy_score(y_test, preds)
    scores_before[name] = acc
    print(f"{name:17s}: {acc:.4f}")

# 9. Apply PCA (retain 95% variance)
pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train)
X_test_pca  = pca.transform(X_test)
```

```python
print(f"\nPCA retained {pca.n_components_} components, "
      f"explained variance = {pca.explained_variance_ratio_.sum():.4f}\n")
# 10. Train & evaluate after PCA
print("=== Accuracies AFTER PCA ===")
scores_after = {}
for name, clf in models.items():
    clf.fit(X_train_pca, y_train)
    preds = clf.predict(X_test_pca)
    acc = accuracy_score(y_test, preds)
    scores_after[name] = acc
    print(f"{name:17s}: {acc:.4f}")
```