VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Dheemanth M (1BM22CS087)

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 **Dheemanth M (1BM22CS087),** 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.

Rashmi H Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	17-3-2025	Write a python program to import and export data using Pandas library functions	
2	17-3-2025	Demonstrate various data pre-processing techniques for a given dataset	
3	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	
4	11-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	
5	11-3-2025	Build Logistic Regression Model for a given dataset	
6	7-4-2025	Build KNN Classification model for a given dataset.	
7	7-4-2025	Build Support vector machine model for a given dataset	
8	21-4-2025	Implement Random Forest ensemble method on a given dataset.	
9	5-5-2025	Implement Boosting ensemble method on a given dataset.	
10	5-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	

Github Link: https://github.com/dheemanthm04/ML

Program 1

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

Screenshot:

ML Lab

I) Agriculture crop yield.

This dataset contains agricultural data for 1M samples aimed at predicting crop yield (in tons per hectare) based on various factors.

Attributes:

- Region
- Soil type
- Crop
- rainfall mm
- temperature_celsius
- Fertilizer used.
- weather condition.
- yield tons per hectare.

II) Mobile phone specification & prices.

This data set contains scraped and preprocessed from gadgets360 website. The data was scraped and cleaned using ~~the~~ code and it is pre processed.

Attributes:

- Name
- Brand
- Model
- Battery capacity
- Screen size
- Touchscreen.
- Resolution x, y
- Processor
- 4G/LTE
- wifi
- Bluetooth.
- Price.

III). Matches in IPL.

This dataset contains details on Indian Premier League of what all matches played and between what teams and who won.

Attributes:

- Id
- Season
- City
- Team1
- Team2
- toss winning
- Toss decision
- Result
- Winner
- win by run
- win by wicket
- Player of match
- Venue
- Umpire

IV) IMDB movies in India.

The dataset involves informational details of the Indian movie Industry from the IMDB dataset.

Attributes:

- Name
- Year
- Duration
- Genre
- Ratings
- Votes
- Director
- Actor1
- Actor2
- Actor 3.

V) Car Details Dataset.

This data is collected from 'Car Dekho'.

It includes all the basic details about the car with the following attributes.

- Car name
- Year
- Selling price
- Kms driven
- Fuel
- seller type
- Transmission
- Owner.

3/3/25

Code:

```
import pandas as pd
# ----------- Step 1: Import Data from CSV -----------
# If you already have a CSV file, replace 'data.csv' with the filename.
# For this demo, let's create a DataFrame and save it to CSV first.
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [24, 30, 22],
    'City': ['New York', 'Los Angeles', 'Chicago']
```

```
}
# Create DataFrame
df = pd.DataFrame(data)
# Exporting DataFrame to CSV
df.to_csv('people.csv', index=False)
print("Data exported to 'people.csv'.")
# ----------- Step 2: Import Data from CSV -----------
imported_df = pd.read_csv('people.csv')
print("\nData imported from 'people.csv':")
print(imported_df)
# ----------- Step 3: Export Data to Excel -----------
df.to_excel('people.xlsx', index=False)
print("\nData exported to 'people.xlsx'.")
# ----------- Step 4: Import Data from Excel -----------
imported_excel_df = pd.read_excel('people.xlsx')
print("\nData imported from 'people.xlsx':")
print(imported_excel_df)
```

OUTPUT:

```
Data exported to 'people.csv'.

Data imported from 'people.csv':
      Name  Age         City
0    Alice   24     New York
1      Bob   30  Los Angeles
2  Charlie   22      Chicago

Data exported to 'people.xlsx'.

Data imported from 'people.xlsx':
      Name  Age         City
0    Alice   24     New York
1      Bob   30  Los Angeles
2  Charlie   22      Chicago
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:

17.3.25 Lab -3

End to End Machine Learning Project.

1. Look at Big Picture → large dataset.

2) Get data → here housing.csv.

Attributes:

- longitude,
- latitude
- housing median age.
- total room
- total bed-room.
- population.
- household.
- Median income.
- median house value
- Ocean proximity

No. of rows & colums ⇒ 20615 X 10 (rows) (cols)

Creating test data set: → randomly, typically 20% of dataset.

3) Discover & visualize data:

- histogram, scatter plot, correlation matrices.

4) Prepare data for ML:

- Clean data by handling missing values & remove duplicate.
- convert categorical to numerical data format
- Scale using normalization or standardization.

5. Select and train a Model:

- Train reg → Linear regression, Random Forest
- Evaluate model on training set.
- use cross validation.

6. Fine Tune Model:

- Use Grid search.
- Randomized search.

7. Evaluate system:

Test model on unseen test set and compare with initial expectations.

8. Launch, Monitor, Maintain:

- Deploy model into production.
- Retrain as necessary.

Rajesh

Code:

```
from google.colab import files
diabetes=files.upload()

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

df1=pd.read_csv("Dataset of Diabetes .csv")
df1.head()

df2=pd.read_csv("adult.csv")
df2.head()
```

```
df1.info()
df2.info()
df1.describe()
df2.describe()

missing_values1 = df1.isnull().sum()
print(missing_values1)
missing_values2 = df2.isnull().sum()
print(missing_values2)

df1['Gender'] = df1['Gender'].replace('f', 'F')
ordinal_encoder = OrdinalEncoder(categories=[["F", M"]])
df1["Gender_Encoded"] =
ordinal_encoder.fit_transform(df1[["Gender"]]) onehot_encoder =
OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df1[["CLASS"]]) encoded_array
= encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["CLASS"])) df_encoded = pd.concat([df1,
encoded_df],  axis=1)
df1 = pd.concat([df1, encoded_df], axis=1)
df1.drop("CLASS", axis=1, inplace=True)
df1.drop("Gender", axis=1, inplace=True)
print(df2.head())
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
df_copy2 = df2
ordinal_encoder = OrdinalEncoder(categories=[["Male","Female"]])
df_copy2["Gender_Encoded"] =
ordinal_encoder.fit_transform(df_copy2[["gender"]])
print(df_copy2[["gender","Gender_Encoded"]])

onehot_encoder = OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df2[["occupation","workclass","education
","marital status","relationship","race","n ative-country","income"]])
encoded_array = encoded_data.toarray()
encoded_df =
pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["occupation","workclass","education
","marital status","relatio nship","race","native-country","income"]))
df_encoded = pd.concat([df_copy2, encoded_df], axis=1)

df_encoded.drop("gender", axis=1, inplace=True)
df_encoded.drop("occupation", axis=1, inplace=True)
df_encoded.drop("workclass", axis=1, inplace=True)
df_encoded.drop("education", axis=1, inplace=True)
df_encoded.drop("marital-status", axis=1, inplace=True)
df_encoded.drop("relationship", axis=1, inplace=True)
df_encoded.drop("race", axis=1, inplace=True)
df_encoded.drop("native-country", axis=1, nplace=True)
df_encoded.drop("income", axis=1, inplace=True)
```

```
print(df_encoded. head())

normalizer = MinMaxScaler()
df_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-week"]] =
normalizer.fit_transform(df_encoded[["fnlwgt","educational-num","capital-gain","capital-
loss","hours-per week"]
])
df_encoded.head()
normalizer = MinMaxScaler()
df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,
"Chol","TG","HDL","LDL","VLDL","BMI"]] =
normalizer.fit_transform(df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,
"Chol","TG","HDL","LDL","VLDL","BMI"]])
df1.head()
```

```
import os
import tarfile
import urllib
import pandas as pd
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"
def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    os.makedirs(housing_path, exist_ok=True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
fetch_housing_data()
def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
housing = load_housing_data()
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

housing['ocean_proximity'].value_counts()

ocean_proximity	count
<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAY	2290
ISLAND	5

dtype: int64

housing.describe()

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

```
import numpy as np
# def split_train_test(data, test_ratio):
#   np.random.seed(42)
#   shuffled_indices = np.random.permutation(len(data))
#   test_set_size = int(len(data) * test_ratio)
#   test_indices = shuffled_indices[:test_set_size]
#   train_indices = shuffled_indices[test_set_size:]
#   return data.iloc[train_indices], data.iloc[test_indices]

from sklearn.model_selection import train_test_split, StratifiedShuffleSplit

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
len(test_set)
```

```
4128
```

```
housing['income_cat'] = pd.cut(housing['median_income'],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
housing['income_cat'].hist()
```

```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing['income_cat']):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

```
strat_test_set['income_cat'].value_counts() / len(strat_test_set)
```

income_cat	count
3	0.350533
2	0.318798
4	0.176357
5	0.114341
1	0.039971

dtype: float64

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop('income_cat', axis=1, inplace=True)
housing = strat_train_set.copy()
housing.plot(kind='scatter', x='longitude', y='latitude')
```

```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.1)
```

```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,
             s=housing['population']/100, label='population', figsize=(10, 7),
             c='median_house_value', cmap=plt.get_cmap('jet'), colorbar=True)
plt.legend()
```

```
# Select only numerical features for correlation calculation
numerical_features = housing.select_dtypes(include=['number'])  # Exclude non-numeric columns
corr_matrix = numerical_features.corr()  # Calculate correlation for numerical features only
corr_matrix['median_house_value'].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.687151
total_rooms	0.135140
housing_median_age	0.114146
households	0.064590
total_bedrooms	0.047781
population	-0.026882
longitude	-0.047466
latitude	-0.142673

dtype: float64

```
from pandas.plotting import scatter_matrix
attributes = ['median_house_value', 'median_income', 'total_rooms', 'housing_median_age']
scatter_matrix(housing[attributes], figsize=(12, 8))
```

```
housing.plot(kind='scatter', x='median_income', y='median_house_value', alpha=0.1)
```

```
housing['rooms_per_household'] = housing['total_rooms']/housing['households']
housing['bedrooms_per_room'] = housing['total_bedrooms']/housing['total_rooms']
housing['population_per_household'] = housing['population']/housing['households']
# Select only numerical features for correlation calculation
numerical_features = housing.select_dtypes(include=['number'])  # Exclude non-numeric columns
```

```
corr_matrix = numerical_features.corr()  # Calculate correlation for numerical features only
corr_matrix['median_house_value'].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.687151
rooms_per_household	0.146255
total_rooms	0.135140
housing_median_age	0.114146
households	0.064590
total_bedrooms	0.047781
population_per_household	-0.021991
population	-0.026882
longitude	-0.047466
latitude	-0.142673
bedrooms_per_room	-0.259952

dtype: float64

Program 3

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

Screenshot:

24.3.25 Lab 4.

Decision Tree - ID3 classification problem.

Pseudocode:

ID3 (dataset, features, target):

1. If all instances in dataset belong to same class:
   Return that class (pure node).
2. If no features remain for splitting:
   Return the majority class in dataset.
3. Compute info. gain for each feature in dataset.
4. Select the feature with highest info. gain.
5. Create decision node with the selected feature.
6. For each unique value of the selected feature:
   a) Create a subset of the dataset where the feature has that value.
   b) Recursively call ID3 on the subset with remaining features.
   c) Assign the result to the corresponding branch of the decision node.
7. Return constructed decision tree.

$Gain(S, Attribute) = Entropy(S) - \sum \frac{|S_v|}{|S|} Entropy(S_v)$

$Entropy(S) = -P_{\oplus} \log_2 P_{\oplus} - P_{\ominus} \log_2 P_{\ominus}$

Dataset:

weather forecast.

1) Outlook
2) Temperature
3) Humidity.
4) Wind
5) Play.

(values are qualitative - categorical)

1. Outlook ∈ [Sunny, Overcast, Rainy]
2. Temperature ∈ [Hot, Mild, Cold]
3. Humidity ∈ [High, Normal]
4. Wind ∈ [Weak, Strong]

Play → classlabel → ∈ [Yes, No].

Eff.

Root → Outlook

24/3/2025

Code:

```
import pandas as pd
import numpy as np
# Load the dataset
file_path = "/content/diabetes.csv"
df = pd.read_csv(file_path)
def entropy(df, col):
    ppos = (df[col] == 1).sum()
    pneg = len(df[col]) - ppos
    total = ppos + pneg
    if ppos == 0 or pneg == 0:
        return 0
    ppos /= total
    pneg /= total
```

```
    return -ppos * np.log2(ppos) - pneg * np.log2(pneg)
def information_gain(df, feature, target):
    total_entropy = entropy(df, target)
    values, counts = np.unique(df[feature], return_counts=True)
    weighted_entropy = sum((counts[i] / sum(counts)) * entropy(df[df[feature] == values[i]], target)
                           for i in range(len(values)))
    return total_entropy - weighted_entropy

def id3(df, features, target):
    # If all target values are the same, return that class
    if len(df[target].unique()) == 1:
        return df[target].iloc[0]

    # If no features left, return the most common target value
    if not features:
        return df[target].mode()[0]

    # Compute information gain for all features
    info_gains = {feature: information_gain(df, feature, target) for feature in features}

    # Print information gain for each feature
    print("Information Gain for each feature:")
    for feature, gain in info_gains.items():
        print(f"{feature}: {gain}")

    # Select the best feature
    best_feature = max(info_gains, key=info_gains.get)
    tree = {best_feature: {}}

    # Recur for each unique value in the selected feature
    for value in df[best_feature].unique():
        sub_df = df[df[best_feature] == value].drop(columns=[best_feature])
        tree[best_feature][value] = id3(sub_df, [f for f in features if f != best_feature], target)

    return tree

def print_tree(tree, indent=""):
    if not isinstance(tree, dict):
        print(indent + "→ " + str(tree))
        return
    for key, value in tree.items():
        print(indent + str(key))
        for sub_key in value:
            print(indent + "  ├── " + str(sub_key))
            print_tree(value[sub_key], indent + "  │   ")

# Compute the decision tree
features = [col for col in df.columns if col != "Outcome"]
decision_tree = id3(df, features, "Outcome")

print("Decision Tree:")
print_tree(decision_tree)
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

10-5-24

Lab-2: Linear Regression

Supervised learning algorithm used for modelling relationship between dependent variable (Y) and one or more independent variables (x)

It is Process of estimating a functional relationship between x and Y

Linear Regression Single Variable.

$$\hat{y} = \beta_0 + \beta_1 x + \varepsilon_i$$

$\hat{\beta_1}$ (slope) $= \frac{\Delta y}{\Delta x}$

$y \rightarrow$ dependent (target)
$x \rightarrow$ independent (feature)
$\beta_0 \rightarrow$ intercept (constant term)
$\beta_1 \rightarrow$ slope (coefficient)
$\varepsilon \rightarrow$ error term (difference b/w predicted & actual value)

$\beta_1 \rightarrow$ slope is computed using:

i) Using mean based formula: $\beta_1 = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - (\bar{x})^2}$

ii) Using Ordinary Least Sq. Summation formula: $\beta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$

iii) Using std. least square formula:

$$\beta_1 = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sum x_i^2 - n\bar{x}^2}$$

compute intercept (β_0):

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

code:

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

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

df = df.iloc[:50]        # takes 50 rows.

x = df.iloc[:, 0].value
y = df.iloc[:, 1].value.

x_mean, y_mean = np.mean(x), np.mean(y)

#slope, m = np.sum(x - x_mean) * (y - y_mean)) / np.sum((x - x_mean) ** 2)

#intercept) b = y_mean - m * x_mean

y_pred = m * x + b

mse = np.mean((y - y_pred) ** 2)

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

plt.scatter(x, y, color="blue", label='Data Points')

plt.plot(x, y_pred, color='red', label='Reg line')

plt.xlabel("x values")

plt.ylabel("y values")

plt.legend()

plt.title("Linear Reg (Limited 50)")

plt.show()
```

Code:

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

```
df = pd.read_csv("/content/sample_data/linear_regression_dataset.csv")
df1 = pd.read_csv("/content/sample_data/data_for_lr.csv")
X = df[['Experience (Years)']].values.flatten()
y = df['Salary ($1000s)'].values
# X = df1['x'].values.flatten()
# y = df1['y'].values.flatten()
X, y
```

```
(array([ 7,  4, 13, 15, 11,  8, 13,  5,  7, 10,  3,  7, 11, 11,  8,  5,  4,
         8,  8,  3,  6,  5,  2,  8, 12, 14,  6,  2, 12,  5]),
 array([20184, 10459, 40385, 43021, 33300, 22747, 39904, 16632, 19474,
        29082,  9558, 22753, 33047, 34547, 24747, 13975, 11806, 22189,
        25005,  9734, 19005, 13562,  5899, 25638, 35267, 42879, 17528,
         7202, 37556, 16890]))
```

```
mean_x = np.mean(X)
mean_y = np.mean(y)

numerator = np.mean(X * y) - mean_x * mean_y
denominator = np.mean(X**2) - mean_x**2

beta1 = numerator / denominator
beta0 = mean_y - beta1 * mean_x

y_pred = beta0 + beta1 * X

error = np.sum((y - y_pred) ** 2)
mean_x, mean_y
```

```
(7.766666666666667, 23465.833333333332)
```

```
print(f"Intercept (beta0): {beta0:.2f}")
print(f"Slope (beta1): {beta1:.2f}")
print(f"Error: {error:.2f}")
```

```
Intercept (beta0): 138.77
Slope (beta1): 3003.49
Error: 43353835.99
```

```
plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('Experience (Years)')
plt.ylabel('Salary ($1000s)')
plt.title('Linear Regression: Experience vs Salary')
plt.legend()
plt.show()
```

result = f"Linear Equation: Y = {beta0:.2f} + {beta1:.2f}X + {error:.2f}"
result

```
'Linear Equation: Y = 138.77 + 3003.49X + 43353835.99'
```

Program 5

Build Logistic Regression Model for a given dataset

Code:

```
from google.colab import files
hr=files.upload()

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
from sklearn import linear_model
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df1=pd.read_csv("HR_comma_sep.csv")
df1.head()
df1.isnull().sum()
plt.figure(figsize=(12, 6))
sns.barplot(x='Department', y='left', data=df1)
plt.title('Employee Retention Rate by Department')
plt.xlabel('Department')
plt.ylabel('Proportion of Employees Left')
plt.xticks(rotation=45, ha='right')
plt.show()

ohe = OneHotEncoder(handle_unknown='ignore',
sparse_output=False)  department_encoded =
ohe.fit_transform(df1[['Department']])
department_encoded_df = pd.DataFrame(department_encoded,
columns=ohe.get_feature_names_out(['Department']))
df1 = pd.concat([df1, department_encoded_df], axis=1)
df1 = df1.drop('Department', axis=1)
ordinal_encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']],
dtype=np.int64)  salary_encoded =
ordinal_encoder.fit_transform(df1[['salary']])
df1['salary_encoded'] = salary_encoded
df1 = df1.drop('salary', axis=1)
df1.head()

correlation_matrix = df1.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
```

```
fmt=".2f")  plt.title('Correlation Matrix of Features')
plt.show()
plt.figure(figsize=(8, 6))
sns.barplot(x='salary_encoded', y='left', data=df1)
plt.title('Impact of Employee Salary on Retention')
plt.xlabel('Salary Level (Encoded)')
plt.ylabel('Proportion of Employees Left')
plt.show()



df_copy = df1[['number_project', 'average_montly_hours', 'time_spend_company',
'left','salary_encoded',  'satisfaction_level','Work_accident']]
df_copy.head()
X = df_copy.drop('left', axis=1)
y = df_copy['left']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression model: {accuracy}")

from google.colab import files
zoodata=files.upload()
zootype=files.upload()

zoo_data = pd.read_csv('zoo-data.csv')
zoo_class = pd.read_csv('zoo-class-type.csv')
merged_data = pd.merge(zoo_data, zoo_class, left_on='class_type',
right_on='Class_Number') merged_data = merged_data.drop(['Animal_Names',
'Number_Of_Animal_Species_In_Class',
'Class_Number','class_type','animal_name'], axis=1)
X = merged_data.drop('Class_Type', axis=1)
y = merged_data['Class_Type']
print(merged_data.head())
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test))  disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix")
plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot

7.4.25

KNN

Algorithm

1. Assign value of K:
   - choose suitable value of K, i.e, no. of nearest neighbours to consider.
   - A good K is usually odd and chosen through testing or cross validation.

2. calculate distance:
   - For a given new data point, calculate its distance from all training data points.

distance : $\sqrt{(x_i - x_1)^2 + (y_i - y_1)^2}$

$x_i, y_i \rightarrow$ i^{th} training point.

$x_1, y_1 \rightarrow$ new data point.

3. Find K nearest neighbors:
   - Sort all the calculated distances in ascending order.
   - Pick the first K data points with smallest distances.

4. Assign the class:
   - check the class labels of those K neighbors.
   - Use majority voting to decide the class:
     - The class that appears most frequently among neighbors is assigned to the new data point

5. Repeat 2 to 4.

Iris dataset

Accuracy v/s K value.

K=1 → lowest accuracy.

K = 3-20 → perfect accuracy.

∴ choose K other than 1.

Code:

```
import csv
import math
import random
from collections import Counter

# ---------- KNN Implementation ----------
```

```
def load_dataset(filename):
    with open(filename, 'r') as f:
        data = list(csv.reader(f))
        header = data[0]
        rows = data[1:]
        for row in rows:
            for i in range(4):
                row[i] = float(row[i])
        return rows

def compute_mean_std(dataset):
    means, stds = [], []
    for i in range(4):
        col = [row[i] for row in dataset]
        mean = sum(col) / len(col)
        std = (sum((x - mean) ** 2 for x in col) / len(col)) ** 0.5
        means.append(mean)
        stds.append(std)
    return means, stds

def normalize_dataset(dataset, means, stds):
    for row in dataset:
        for i in range(4):
            row[i] = (row[i] - means[i]) / stds[i]
    return dataset

def split_dataset(dataset, test_size=0.2):
    random.shuffle(dataset)
    split_index = int(len(dataset) * (1 - test_size))
    return dataset[:split_index], dataset[split_index:]

def euclidean_distance(row1, row2):
    return math.sqrt(sum((row1[i] - row2[i]) ** 2 for i in range(4)))

def knn_predict(train, test_row, k):
    distances = []
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    labels = [neighbor[0][4] for neighbor in neighbors]
    prediction = Counter(labels).most_common(1)[0][0]
    return prediction

def evaluate_model(train, test, k):
    correct = 0
    predictions = []
    for row in test:
        prediction = knn_predict(train, row, k)
        predictions.append(prediction)
```

```
        if prediction == row[4]:
            correct += 1
    accuracy = correct / len(test)
    return accuracy, predictions

filename = '/content/sample_data/IRIS.csv'
dataset = load_dataset(filename)

means, stds = compute_mean_std(dataset)

dataset = normalize_dataset(dataset, means, stds)

train_data, test_data = split_dataset(dataset)

k = 3
accuracy, predictions = evaluate_model(train_data, test_data, k)

print(f"Model Accuracy: {accuracy:.2f}\n\n")

print("\n--- Predict Iris Species from Your Input ---")
try:
    user_input = []
    user_input.append(float(input("Enter Sepal Length (cm): ")))
    user_input.append(float(input("Enter Sepal Width (cm): ")))
    user_input.append(float(input("Enter Petal Length (cm): ")))
    user_input.append(float(input("Enter Petal Width (cm): ")))

    for i in range(4):
        user_input[i] = (user_input[i] - means[i]) / stds[i]

    predicted_species = knn_predict(train_data, user_input, k)
    print(f"\n Predicted Iris Species: {predicted_species}")

except ValueError:
    print("Invalid input. Please enter numeric values for all measurements.")
```

```
Model Accuracy: 0.97



--- Predict Iris Species from Your Input ---
Enter Sepal Length (cm): 5.1
Enter Sepal Width (cm): 3.5
Enter Petal Length (cm): 1.4
Enter Petal Width (cm): 0.3

 Predicted Iris Species: Iris-setosa
```

Program 7

Build Support vector machine model for a given dataset

Screenshot

7.4.25

SVM (Support Vector Machine)

Algorithm

1. Input the training data

$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$

where $y_i \in \{-1, +1\}$

2. choose hyperplane:

find w and b such that

$w^T x + b = 0$

$w^T \rightarrow$ weight

$b \rightarrow$ bias vector term.

3. Set constraints:

$y_i (w^T x_i + b) \geq 1$

4. Maximize margin:

$\frac{1}{2} \|w\|^2$.

5. If data not linearly separable, allow some errors using variables:

$y_i (w^T x_i + b) \geq 1$

6. predict new data:

$f(x) = \text{sign}(w^T x + b)$

$w^T x + b > 0$ for $y_i = +1$

$w^T x + b < 0$ for $y_i = -1$.

7.4.25

Code:

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

```
def PlotData(x,y):
    pos = np.argwhere(y == 1)
    neg = np.argwhere(y == 0)

    plt.plot(x[pos, 0], x[pos, 1], linestyle='', marker='+', color='k')
    plt.plot(x[neg, 0], x[neg, 1], linestyle='', marker='o', color='y')
    plt.xlabel('Exam 1 score')
    plt.ylabel('Exam 2 score')
    plt.legend(['Admitted', 'Not admitted'], loc='upper right', numpoints=1)
    plt.figure()
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC

df = pd.read_csv("/content/sample_data/IRIS.csv")

X = df[["petal_length", "petal_width"]].values
y = df["species"].values

label_map = {"Iris-setosa": 0, "Iris-versicolor": 1, "Iris-virginica": 2}
label_map_rev = {v: k for k, v in label_map.items()}
y_numeric = np.array([label_map[label] for label in y])

svm_model = SVC(kernel="linear")
svm_model.fit(X, y_numeric)

x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300),
                     np.linspace(y_min, y_max, 300))

Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

colors = ["red", "green", "blue"]

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)

for label in np.unique(y_numeric):
    plt.scatter(X[y_numeric == label, 0], X[y_numeric == label, 1],
                label=label_map_rev[label], color=colors[label])

plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Linear SVM Decision Boundary on Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

Linear SVM Decision Boundary on Iris Dataset
Iris-setosa
Iris-versicolor
Iris-virginica
Petal Width (cm)
Petal Length (cm)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

```
import csv
import math
import random
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def load_dataset(filename):
    df = pd.read_csv("/content/sample_data/IRIS.csv")
    X  = df[["petal_length", "petal_width"]].values
    y = df["species"].values

    label_map = {"Iris-setosa": 0, "Iris-versicolor": 1, "Iris-virginica": 2}
    label_map_rev = {v: k for k, v in label_map.items()}
    y_numeric = np.array([label_map[label] for label in y])

class SimpleLinearModel:
    def __init__(self, learning_rate=0.01, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation = self._unit_step_func
        self.weights = None
        self.bias = None

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

        y_ = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation(linear_output)

                update = self.lr * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation(linear_output)
        return y_predicted

    def _unit_step_func(self, x):
        return np.where(x>=0, 1, 0)

svm_model = SimpleLinearModel()
svm_model.fit(X, y_numeric)
```

```
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300),
                     np.linspace(y_min, y_max, 300))

Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

colors = ["red", "green", "blue"]

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)

for label in np.unique(y_numeric):
    plt.scatter(X[y_numeric == label, 0], X[y_numeric == label, 1],
                label=label_map_rev[label], color=colors[label])

plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Linear Model Decision Boundary on Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

Program 8

Implement Random Forest ensemble method on a given dataset.

Screenshot

21.11.24

Random Forest

Algorithm:

```
RF (D, T, F, d-max, n-min):
    initialize forest ← [ ]
    For t ← 1 to T do:
        D-t ← BootstrapSample(D, size=N)
        Tree-t ← BuildTree (D-t, F, d-max, n-min)
        append tree-t to Forest
    Return Forest.

Function BuildTree (D-t, F, d-max, n-min):
    If stopping criteria met (depth=0):
        Return LeafNode with predicted value (majority class or mean)
    else:
        Select F random features from M total features
        For each selected feature:
            For each possible split:
                compute impurity.
        choose best feature and split with lowest impurity.
        Split D-t into D-left and D-right based on best split.

        Left subtree ← BuildTree (D-left, F, d-max-1, n-min)
        Right subtree ← BuildTree (D-right, F, d-max-1, n-min)
        Return DecisionNode (split-feature, split-value, Left subtree, Right subtree)
```

Function Predict (Forest, x):

predictions ← [ ]

For each Tree in Forest do:

pred ← TreePredict(Tree, x)

append pred to predictions.

if classification:

return MajorityVote (predictions)

else if regression:

return mean (predictions).

Function TreePredict (Tree, x):

If Tree is leafNode:

Return Tree.predicted-value

else:

if x[Tree.split-feature] ≤ Tree.split-value:

return TreePredict (Tree.left, x)

else:

return TreePredict (Tree.right, x)

Applications:

1. Finance → analyze stock & predict future performance.

2- Healthcare → Diagonise diseases based on History.

2/14/2025

Dataset used:

. Pumpkin-Seeds-Dataset.

Attributes → area, perimeter, axis length, diameter, compactness, class, extent, roundness.

O/P → Classification report.

Features → ratio, roundness etc.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Load the dataset
file_path = "/content/Pumpkin_Seeds_Dataset.xlsx"
df = pd.read_excel(file_path, sheet_name='Pumpkin_Seeds_Dataset')

# Separate features and target
X = df.drop(columns=['Class'])
y = df['Class']

# Encode the target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Train the Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test data
y_pred = rf_model.predict(X_test)

# Evaluate the model
print("✅ Accuracy:", accuracy_score(y_test, y_pred))
print("\n📊 Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# Plot the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()

# Plot top 10 feature importances
importances = rf_model.feature_importances_
features = X.columns
indices = importances.argsort()[::-1][:10]  # Top 10
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
plt.title("Top 10 Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot

5/9/25

Boosting Ensemble

Boosting is an ensemble learning technique that sequentially combines multiple weak classifiers to create a strong classifier.

types of boosting algorithm:

1. Gradient boosting
2. XGBoost
3. Adaptive Boosting (AdaBoost)

Algorithm:

Input:

- Training dataset: $D = \{(x_1, y_1) \ldots (x_n, y_n)\}$
- base learner (eg. decision stump)
- No of iterations M.

1. Initialize weights for all training samples:

$$w_i = \frac{1}{n} \quad \text{for } i = 1 \ldots n$$

2. For $m = 1$ to M (no of learners):

- Train a base classifier $h_m(x)$ using weights w_i
- compute weighted error:

$$\varepsilon_m = \frac{\sum_{i=1}^{n} w_i \cdot \mathbb{1}(h_m(x_i) \neq y_i)}{\sum_{i=1}^{n} w_i}$$

compute model weight:

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_m}{\varepsilon_m}\right)$$

update sample weights:

$$w_i \leftarrow w_i \times \exp(-\alpha_m y_i h_m(x_i))$$

then normalize weights so that $\sum w_i = 1$

Dataset used:

Liver disease

attributes (features) → age, total bilirubin, direct bilirubin, alkaline, albumin, is male, liver disease

580 rows
12 columns

Output:

ROC AUC score: 0.67
(area under curve)

Applications.

- Spam detection
- Face recognition
- Financial and banking.

Code:

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

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier as KNN

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import mean_squared_error as MSE

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingRegressor

SEED =1
# Dataset
liver = pd.read_csv('/content/sample_data/indian_liver_patient_preprocessed.csv', index_col = 0)
X = liver.drop('Liver_disease', axis = 1)
y = liver['Liver_disease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=SEED)

# Split data into 80% train and 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y,
random_state=SEED)

liver.head()
```

	Age_std	Total_Bilirubin_std	Direct_Bilirubin_std	Alkaline_Phosphotase_std	Alamine_Aminotransferase_std	Aspartate_Aminotransferase_std	Total_Protiens_std	Albumin_std	Albumin_
0	1.247403	-0.420320	-0.495414	-0.428870	-0.355832	-0.319111	0.293722	0.203446	
1	1.062306	1.218936	1.423518	1.675083	-0.093573	-0.035962	0.939655	0.077462	
2	1.062306	0.640375	0.926017	0.816243	-0.115428	-0.146459	0.478274	0.203446	
3	0.815511	-0.372106	-0.388807	-0.449416	-0.366760	-0.312205	0.293722	0.329431	
4	1.679294	0.093956	0.179766	-0.395996	-0.295731	-0.177537	0.755102	-0.930414	

```
# Import AdaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier

# Instantiate dt
dt = DecisionTreeClassifier(max_depth=2, random_state=1)

# Instantiate ada
ada = AdaBoostClassifier(estimator=dt, n_estimators=180, random_state=1)
```

```
# Fit ada to the training set
ada.fit(X_train, y_train)

# Compute the probabilities of obtaining the positive class
y_pred_proba = ada.predict_proba(X_test)[:,1]

# Import roc_auc_score
#from sklearn.metrics import roc_auc_score

# Evaluate test-set roc_auc_score
ada_roc_auc = roc_auc_score(y_test, y_pred_proba)

# Print roc_auc_score
print('ROC AUC score: {:.2f}'.format(ada_roc_auc))
```

```
ROC AUC score: 0.70
```

Program 10

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

Screenshot

5.5.25 K means clustering Algorithm

1. Data Generation:
   - Generate a 2D dataset with clear cluster structure.
   - Parameter: samples, features, centres.

2. Initialize:
   - Choose no. of clusters, K, which is user defined parameter.
   - Randomly initialize K centroids (mean points) for clusters.

3. Assign points to closest centroid:
   - For each data point in dataset, calculate distance (Euclidean distance) of K centroids.

4. Update centroid:
   - After all points assigned, recalculate centroids of clusters:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N_k} x_i$$

N_k is no. of points in cluster K.

x_i are points assigned to cluster.

5. Convergence check:
   - If centroids have not changed significantly after update step, the algm has converged & can stop. Repeat 2 & 3 steps

6. Termination:

The algorithm terminates when the centroids no longer change, or max. no. of iterations reached.

Application:
- customer segmentation
- Market basket analysis
- Image compression

Dataset:
Mall customers for segmentation

Output:

5/5/25

Code:

```
import numpy as np # linear algebra
```

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

df = pd.read_csv('/content/sample_data/Live.csv')
df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
kmeans.cluster_centers_
kmeans.inertia_
labels = kmeans.labels_
# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
```

```
Result: 4288 out of 7050 samples were correctly labeled.
```

```
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Accuracy score: 0.61
```

```
from sklearn.cluster import KMeans
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2,random_state=0)

kmeans.fit(X)

labels = kmeans.labels_

# check how many of the samples were correctly labeled

correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))

print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4288 out of 7050 samples were correctly labeled.
Accuracy score: 0.61
```

```
kmeans = KMeans(n_clusters=3, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4066 out of 7050 samples were correctly labeled.
Accuracy score: 0.58
```

```
kmeans = KMeans(n_clusters=4, random_state=0)

kmeans.fit(X)
```

```
# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4112 out of 7050 samples were correctly labeled.
Accuracy score: 0.58
```

Program 11

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

Screenshot

12.6.25 Dimensionality reduction using Principle Component Analysis (PCA)

Algorithm:

1. standardize data:
   ensure each feature has mean of 0 and sd of 1

$$X_{sd} = \frac{X-\mu}{\sigma}$$

$\mu \rightarrow$ feature mean
$\sigma \rightarrow$ std. deviation.

2. compute covariance Matrix:

$$C = \frac{1}{n-1} X_{std}^T X_{sd}$$

3. Compute eigen values and eigen vectors

$$Cv_i = \lambda_i v_i$$

$\lambda_i \rightarrow$ eigen values
$v_i \rightarrow$ corresponding eigenvector

4. Sort eigen vectors by decreasing eigenvalue magnitude.

5. Select Top-k components from projection Matrix.

6. Project Data by transforming original into new k-dimensional space

$$X_{reduced} = X_{sd} \cdot W$$

Output:

Dataset used → heart.csv

- Accuracy before PCA:

Logistic Regression: 0.9016
SVM: 0.8525
Random Forest: 0.8361

- Accuracy After PCA:

Logistic Regression: 0.8684
SVM: 0.8684
Random Forest: 0.83521

Code:

```
from google.colab import files
heart=files.upload()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, OneHotEncoder  from sklearn.model_selection import
train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score  from sklearn.preprocessing
import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
df1=pd.read_csv("heart (1).csv")
df1.head()
text_cols = df1.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()
for col in text_cols:
df1[col] =
label_encoder.fit_transform(df1[col])
print(df1.head())
X = df1.drop('HeartDisease', axis=1)
y = df1['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train) X_test =
scaler.transform(X_test)
# Support Vector Machine
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f"SVM Accuracy: {svm_accuracy}")
# Logistic Regression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train) lr_predictions = lr_model.predict(X_test) lr_accuracy = accuracy_score(y_test,
lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy}")
# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy}")
models = {
```

```
"SVM": svm_accuracy,
"Logistic Regression":
lr_accuracy, "Random Forest":
rf_accuracy
}
best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
svm_model_pca = SVC(kernel='linear', random_state=42)
svm_model_pca.fit(X_train_pca, y_train)
svm_predictions_pca = svm_model_pca.predict(X_test_pca)
svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)
print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")
lr_model_pca = LogisticRegression(random_state=42)
lr_model_pca.fit(X_train_pca, y_train)
lr_predictions_pca = lr_model_pca.predict(X_test_pca)
lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)
print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")
rf_model_pca = RandomForestClassifier(random_state=42)
rf_model_pca.fit(X_train_pca, y_train)
rf_predictions_pca = rf_model_pca.predict(X_test_pca)
rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)
print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")
models_pca = {
"SVM": svm_accuracy_pca,
"Logistic Regression": lr_accuracy_pca,
"Random Forest": rf_accuracy_pca
}
best_model_pca = max(models_pca, key=models_pca.get)
print(f"\nBest Model (with PCA): {best_model_pca} with accuracy {models_pca[best_model_pca]}")
```