Basic Data Preprocessing

1a. Installation of Python environment / Anaconda IDE for machine learning installing Python modules / packages like Scikit-learn, Keras and Tensorflow.

Step1: Install Anaconda

1. Download Anaconda.
2. Install Anaconda
3. Verify Installation
   - Open a terminal

```
> conda --version.
```

Step2: Set up a Virtual Environment

1. Create a new environment

```
> conda create --name ml-env python=3.9
```

2. Activate the environment

```
> conda activate ml_env
```

Step3: Install Python Modules / Packages

1. Install Machine learning libraries:

```
> conda install scikit-learn
  pip install keras tensorflow
```

(or)

```
> pip install scikit-learn keras tensorflow
```

2. Additional Useful libraries:

```
>pip install pandas numpy matplotlib seaborn
```

3. Verify Installation:

```
>python
```

Then import the packages:

```
import sklearn
import tensorflow as tf
import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

If no, errors appears, the setup is Complete.

Step 4: Start Coding.

1. Launch the Jupyter Notebook

```
>Jupyter notebook
```

(or) use an IDE like spyder or vs code.

2. Begin by importing and preprocessing datasets using the installed libraries.

16. Programs Involving Pandas, Numpy and Scipy libraries.

i. Pandas

Program:-

```
import pandas as pd
# Create sample dataset
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [25, None, 30, 35, 40],
    'Gender': ['female', 'male', None, 'male', 'female'],
    'Salary': [50000, 60000, None, 80000, 70000]
    }
df = pd.DataFrame(data)
#fill missing values
df['Age'].fillna(df['Age'].mean(), inplace=True)
df['salary'].fillna(0, inplace=True)
#Encode categorical data.
df['Gender'] = df['Gender'].fillna('Unknown') # fill missing
                                    Gender with 'Unknown'.
df['Gender'] = df['Gender'].map({'Female':0, 'Male':1, 'Unknown':2})
# map Gender values
```

```
df['salary_in_Lakhs'] = df['salary'] / 100000 # Add new column, convert salary to lakhs.

print("\nPreprocessed Dataset:")
print(df)
```

Output:-

PreProcessed Dataset:

	Name	Age	Gender	Salary	salary_in_Lakhs
0	Alice	25.0	0	50000.0	0.5
1	Bob	32.5	1	60000.0	0.6
2	Charlie	30.0	2	0.0	0.0
3	David	35.0	1	80000.0	0.8
4	Eve	40.0	0	70000.0	0.7

ii. Numpy.

Program:-

```
import numpy as np
#Example data with missing values & categorical features.
data = np.array([
    [25, 5.5, 60, 'male'],
    [30, 6.0, 75, 'female'],
    [35, 5.9, np.nan, 'male'],
    [40, 6.1, 85, 'female'],
    [45, 5.8, 95, 'male'],
])
```

```
def handle_missing_data(data):
    # Convert the data to a float array for processing.
    data_float = data[:, :-1].astype(float) # Ignore the Categorical
                                              Column.

    # Replace missing values (np.nan) with the column mean
    column_means = np.nanmean(data_float, axis=0)
    inds = np.where(np.isnan(data_float))
    data_float[inds] = np.take(column_mean, inds[1])
    return data_float.

    # Normalize the features (scale them to a range of 0-1)
    def normalize_data(data):
        min_vals = data.min(axis=0)
        max_vals = data.max(axis=0)
        return (data-min_vals)/(max_vals-min_vals)

    # Encoding Categorical data (convert 'male'/'female' to 0/1)
    def encode_categorical(data):
        gender = data[:,-1] # Extract the last column (gender)
        gender_encoded = np.where(gender='male', 0, 1)
        return gender_encoded
```

```
# Data Preprocessing Steps.
processed_data = handle_missing_data(data)
normalized_data = normalize_data(processed_data)
encoded_gender = encode_categorical(data).

# Final Processed Data.
print("Processed Data (Missing Values Handled):\n", processed_data)
print("\nNormalized Data (Scaled between 0 and 1):\n", normalized_data)
print("\nEncoded Gender Data (0:male, 1:female):\n", encoded_gender)
```

Output :-

```
Processed Data (Missing Values Handled):
[[25.   5.5   60.  ]
 [30.   6.    75.  ]
 [35.   5.9   78.75]
 [40.   6.1   85.  ]
 [45.   5.8   95.  ]]

Normalized Data (Scaled between 0 and 1):
[[0.    0.    0.    ]
 [0.25  0.8333  0.4285]
 [0.5   0.667   0.535 ]
 [0.75  1.      0.714 ]
 [1.    0.5     1.    ]]

Encoded Gender Data (0:male, 1:female): [0 1 0 1 0]
```

(iii) Scipy

Program:-

```
import numpy as np
from scipy import stats
from scipy.sparse import csr_matrix
from sklearn.preprocessing import LabelEncoder.

# Example data with missing values and Categorical features
data = np.array([
    [25, 5.5, 60, 'male'],
    [30, 6.0, 75, 'female'],
    [35, 5.9, np.nan, 'male'],
    [40, 6.1, 85, 'female'],
    [45, 5.8, 95, 'male']
])

def handle_missing_data(data):
    data_float = data[:, :-1].astype(float) # Exclude categorical column;
    col_means = np.nanmean(data_float, axis=0)

#Replace missing values(np.nan) with the column mean.
    inds = np.where(np.isnan(data_float))
    data_float[inds] = np.take(col_means, inds[1])
    return data_float
```

```
# Standardize (z-score normalization) the features.
def standardize_data(data):
    return stats.zscore(data, axis=0)

# Encoding Categorical data (convert 'male'/'female' to 0/1)
def encode_categorical(data):
    gender = data[:, -1] # Extract the last column (gender)
    label_encoder = LabelEncoder()
    gender_encoded = label_encoder.fit_transform(gender)
    return gender_encoded.

# Data preprocessing steps
processed_data = handle_missing_data(data)
standardized_data = standardize_data(processed_data)
encoded_gender = encode_categorical (data)

# Final Processed Data.
Print("Processed Data(Missing values Handled):\n", processed_data)
Print("\nStandardized Data(z-score normalization):\n", standardized_data)
Printf("\nEncoded Gender Data(0:male, 1:female):\n", encoded_gender)
```

Output:-

Processed Data (Missing values Handled):-

```
[[25.   5.5  60.  ]
 [30.   6.   75.  ]
 [35.   5.9  78.75]
 [40.   6.1  85.  ]
 [45.   5.8  95.  ]]
```

Standardized Data (z-score normalization):-

```
[[-1.414  -1.7483  -1.621 ]
 [-0.707   0.6799  -0.324 ]
 [ 0.      0.194    0.    ]
 [ 1.414  -0.291    1.405 ]]
```

Encoded Gender Data (0:Male, 1:Female): [1 0 1 0 1]

(2) Programs for classification

a. Builds models using linear regression and logistic regression and apply it to classify a new instance.

(i) Linear regression:-

Program:-

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd.

#Load the iris dataset
iris = load_iris()
X = iris.data[:, [0,1,3]]
y_continuous = iris.data[:,2] # for linearRegression, let's predict petal length.

X_train, X_test, y_train, y_test = train_test_split(X, y_continuous, test_size=0.2,
                                                    random_state=42)
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

from sklearn.preprocessing import standardScaler
scaler = scaler.fit_transform(X_train) = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

6.

```
Print("Coefficients:", lr.coef_)
Print("Intercepts:", lr.intercept_)
```

output:-

Coefficients: [0.7228 -0.6358 1.4675]

Intercepts: -0.262195

```
Print("Linear Regression (iris Dataset-Petal Length):")
Print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
```

output:-

Linear Regression (iris Dataset-Petal Length):

Mean Squared Error: 0.13001626

```
#Predict for a new instance.
new_instance = [[5.1, 3.5, 0.2]]
Predicted_Petal_Length = lr.Predict(new_instance)
Print("Predicted petal length for new_instance:", Predicted_Petal_Length)
```

Output:-

Predicted petal length for new instance: [1.49230577]

(ii) Logistic regression:-

Program:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import numpy as np
import pandas as pd

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

iris_df = pd.DataFrame(data= iris.data, columns= iris.feature_names)
iris_df['target'] = iris.target
Print(iris_df.head())
```

Output:-

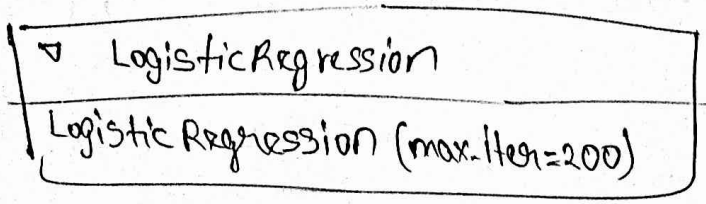
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

	target
0	0
1	0
2	0
3	0
4	0

6.

```
y_classification = iris.target
x_train, x_test, y_train, y_test = train_test_split(x, y_classification, test_size=0.2,
                                                    random_state=42)

from sklearn.preprocessing import standardscaler
scaler = Standardscaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

log_res = LogisticRegression(max_iter=200)
log_res.fit(x_train, y_train)
```

O/P:-




```
y_pred_log = log_res.predict(x_test)

print("Coefficients:", log_res.coef_)
print("Intercept:", log_res.intercept_)
```

O/P:-

```
Coefficients: [[-1.0031  1.1144 -1.81  -1.69 ]
               [0.527  -0.282  -0.3408 -0.719]
               [0.4753 -0.8616  2.153   2.411]]

Intercept: [-0.133  1.98  -1.84]
```

```
Print("LogisticRegression(IrisDataset-Species Classification):")
Print("Accuracy:", accuracy_score(y_test, y_pred_log))
Print("Classification Report:\n", classification_report(y_test, y_pred_log))
```

O/P:-

LogisticRegression(IrisDataset-Species Classification):

Accuracy: 1.0

Classification Report:

	Precision	recall	f1-Score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
new_instance = [[5.1, 3.5, 1.4, 0.2]]
Predicted_species = log_res.Predict(new_instance)
Print(Predicted_species)
Print("Predicted Species for new instance:", iris.target_names[Predicted_species[0]])
```

O/P:

[1]

Predicted Species for new instance: Versicolor.

7.

b. Write a program to demonstrate the following classifiers. Use an appropriate dataset for building the model. Apply the model to classify a new instance.

(i) Decision tree :-

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report,
                            confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt.

wine= load_wine( )
 x=wine.data
 y = wine.target

wine_df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
wine_df['target'] = wine.target
print(wine_df.head( ))
```

O/P:

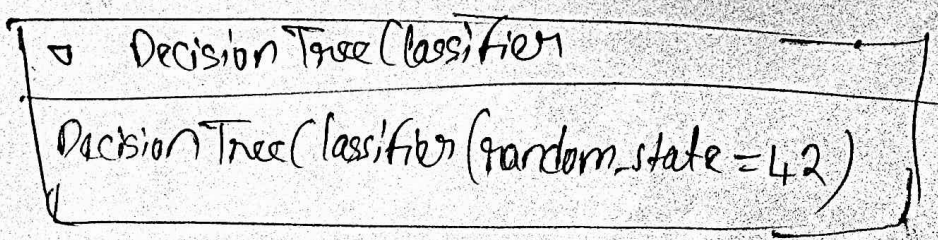
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols
0	14.23	1.71	2.43	15.6	127.0	2.80
1	13.20	1.78	2.14	11.2	100.0	2.65
2	13.16	2.36	2.67	18.6	101.0	2.80
3	14.37	1.95	2.50	16.8	113.0	3.85
4	13.24	2.59	2.87	21.0	118.0	2.80

	flavanoids	nonflavanoids_phenols	proanthocyanins	color_intensity	hue
	3.06	0.28	2.29	5.64	1.04
	2.76	0.26	1.28	4.38	1.05
	3.24	0.30	2.81	5.68	1.03
	3.49	0.24	2.18	7.80	0.86
	2.69	0.39	1.82	4.32	1.04

	od280/od315_of_diluted_wines	proline	target
	3.92	1065.0	0
	3.40	1050.0	0
	3.17	1185.0	0
	3.45	1480.0	0
	2.93	735.0	0

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

O/P:-




```
y_pred_clf = clf.predict(X_test)
```

8.

```
Print("Decision Tree Classifier(wine Dataset):");
Print("Accuracy Score:", accuracy_score(y_test, y_pred_clf))
Print("Classification_report:\n", classification_report(y_test, y_pred_clf))
```

O/P:-

Decision Tree Classifier(wine Dataset):

Accuracy Score: 0.9444

Classification_report:

	Precision	recall	f1-score	support
0	0.93	0.93	0.93	14
1	0.93	1.00	0.97	14
2	1.00	0.88	0.93	8
accuracy			0.94	36
macro avg	0.95	0.93	0.94	36
weighted avg	0.95	0.94	0.94	36

```
new_instance = [[13.4, 2.5, 2.6, 19.4, 100.0, 2.9, 2.5, 0.3, 1.3, 3.0, 1.0, 2.2, 680]]
dt_Prediction = clf.predict(new_instance)
Print("Predicted wine class with decision Tree:", Wine.target_names[dt_Prediction[0]])
```

O/P:-

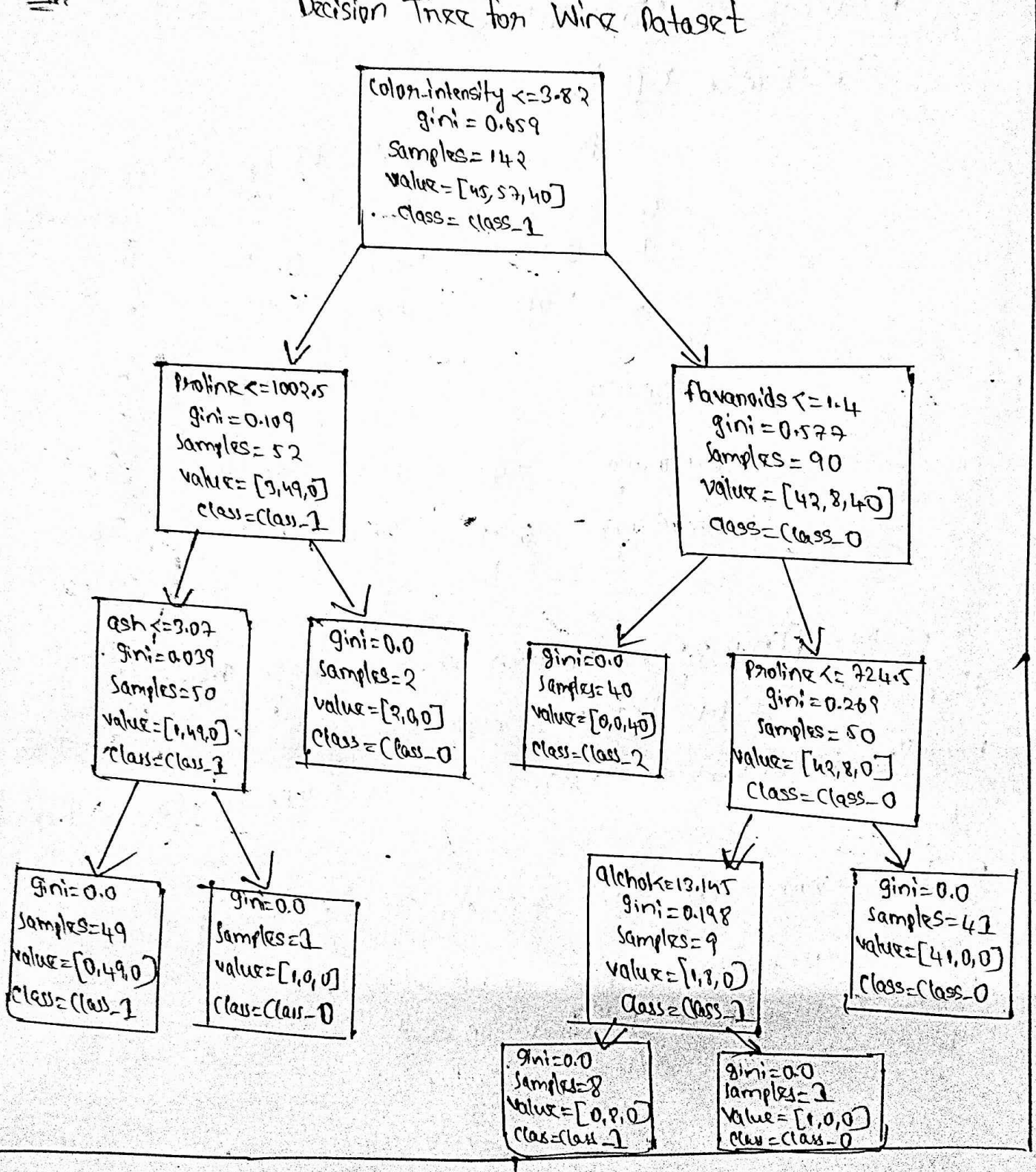
Predicted wine class with decision Tree: class_1

```
from sklearn.metrics import confusion_matrix
Print("confusion matrix:\n", confusion_matrix(y_test, y_pred_clf))
```

O/P:- confusion matrix: [[13 1 0]
[0 14 0]
[0 0 7]]

```
plt.figure(figsize=(15,10))
plot_tree(clf, feature_names=wine.feature_names, class_names=wine.
                .target_names.tolist(), filled=True)
plt.title("Decision Tree for wine Dataset")
plt.show()
```

O/P:-



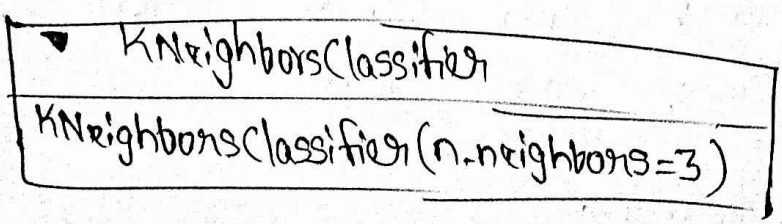
9.

(ii) K-Nearest Neighbour

Program:-

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, r2_score,
                            mean_squared_error, confusion_matrix

digits = load_digits()
x = digits.data
y = digits.target

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)
```

O/P:-




```
y_pred_knn = knn.Predict(x_test)
Printf("K-Neighbor classifier (digits Dataset):")
Print("Accuracy:", accuracy_score(y_test, y_pred_knn))
Print("Classification report:\n", classification_report(y_test, y_pred_knn))
Print("R2 score:", r2_score(y_test, y_pred_knn))
Print("Mean Squared Error:", mean_squared_error(y_test, y_pred_knn))
```

O/P:-

K_Neighbor classifier (Digits Dataset):

Accuracy: 0.98333

Classification report:-

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	33
1	0.97	1.00	0.98	28
2	1.00	1.00	1.00	33
3	0.97	1.00	0.99	34
4	0.98	1.00	0.99	46
5	0.98	0.98	0.98	47
6	0.97	1.00	0.99	35
7	1.00	0.97	0.99	34
8	1.00	0.97	0.98	40
9	0.97	0.93	0.95	40
accuracy			0.98	360
macro avg	0.98	0.98	0.98	360
weighted avg	0.98	0.98	0.98	360

R2 score : 0.9528927

Mean Squared Error: 0.363888

10.

```
new_instance = [[0,0,6,15,14,3,0,0,0,0,7,16,13,5,0,0,0,9,16,14,5,0,0,11,14,7,
0,0,3,14,15,0,0,0,9,16,10,3,0,0,0,0,7,14,13,4,0,0,1,12,13,0,0,0,13,12,0,0,
1,15,16,0,0,0]]

Knn_Prediction = Knn.Predict(new_instance)
Print("Predicted digit with KNearest Neighbor:", Knn_Prediction[0])
```

O/P:-

Predicted digit with KNearest Neighbor: 8.

```
Print("Confusion matrix:\n", Confusion_matrix(y_test, y_Pred_Knn))
```

O/P:-

Confusion matrix:

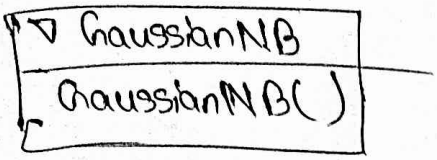
```
[[33  0  0  0  0  0  0  0  0  0]
 [ 0 28  0  0  0  0  0  0  0  0]
 [ 0  0 33  0  0  0  0  0  0  0]
 [ 0  0  0 34  0  0  0  0  0  0]
 [ 0  0  0  0 46  0  0  0  0  0]
 [ 0  0  0  0  0 46  0  0  0  0]
 [ 0  0  0  0  0  0 35  0  0  0]
 [ 0  0  0  0  0  0  0 33  0  1]
 [ 0  1  0  0  0  0  0  0 29  0]
 [ 0  0  0  1  1  1  0  0  0 37]]
```

(iii) Naive Bayes

Program:-

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, r2_score, classification_report,
                            confusion_matrix, mean_squared_error.

cancer = load_breast_cancer()
x = cancer.data
y = cancer.target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
nb = GaussianNB()
nb.fit(x_train, y_train)
```

o/p:




```
y_pred_nb = nb.predict(x_test)

print("Naive Bayes Classifier (Breast Cancer Dataset):")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("R2 score:", r2_score(y_test, y_pred_nb))
print("Mean squared Error:", mean_squared_error(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred_nb))
```

11.

O/P:-

Naive Bayes Classifier (Breast Cancer Dataset):-

Accuracy: 0.973

R2 Score: 0.88797.

Mean Squared Error: 0.02631

Classification Report:

	Precision	recall	f1-Score	Support
0	1.00	0.93	0.96	43
1	0.96	1.00	0.98	71
accuracy			0.97	114
macro avg	0.98	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114

Confusion matrix:

[[40 3]
[0 71]]

```
new_instance = [[15.0, 10.0, 110.0, 0.15, 0.1, 0.08, 0.3, 0.25, 0.7, 0.5, 0.8, 1.2, 0.5, 1.0, 0.6,
0.8, 1.0, 0.3, 1.2, 1.0, 0.4, 0.6, 0.9, 0.5, 0.7, 14.0, 1.2, 0.4, 0.1, 0.3]])

nb_predicted = nb.predict(new_instance)
print(nb_predicted)
print("Predicted class with naive bayes:", Cancer.target_names[nb_predicted[0]])
```

O/P:-

[0]

Predicted class with naive bayes: malignant.

```
prob = nb.predict_proba(new_instance)
print("Prediction probabilities for each class:", prob)
```

o/p

Prediction probabilities for each class: [[1. 0.]]

(iv) Support Vector Machine

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, r2_score, confusion_matrix,
                            classification_report, mean_squared_error
import numpy as np
import pandas as pd

wine = load_wine()
x = wine.data
y = wine.target
wine_df = pd.DataFrame(data=wine.data, column=wine.feature_names)
wine_df['target'] = wine.target
print(wine_df.head())
```

o/p

	alcohol	malic acid	ash	alcalinity of ash	magnesium	total_phenols	flavanoids	non flavanoid phenols
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39

	proanthocyanins	color_intensity	hue	od280/od315_of diluted wines	proline	target
	2.29	5.64	1.04	3.92	1065.0	0
	1.28	4.38	1.05	3.40	1050.0	0
	2.81	5.68	1.03	3.17	1185.0	0
	2.18	7.80	0.86	3.45	1480.0	0
	1.82	4.32	1.04	2.93	735.0	0

12.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
sv = SVC(kernel="linear", random_state=42)
sv.fit(X_train, y_train)
y_pred_sv = sv.predict(X_test)

print("Support Vector Machine classifier (Wine Dataset):")
print("Accuracy score:", accuracy_score(y_test, y_pred_sv))
print("R2 score:", r2_score(y_test, y_pred_sv))
print("Mean squared error:", mean_squared_error(y_test, y_pred_sv))
print("Classification Report:\n", classification_report(y_test, y_pred_sv))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred_sv))
```

O/P:-

Support Vector Machine classifier (Wine Dataset):
Accuracy score: 1.0
R2 score: 1.0
Mean squared error: 0.0
Classification Report:

	Precision	recall	f1-score	Support
0	1.00	1.00	1.00	14
1	1.00	1.00	1.00	14
2	1.00	1.00	1.00	8
accuracy			1.00	36
macro avg	1.00	1.00	1.00	36
weighted avg	1.00	1.00	1.00	36

Confusion matrix:

```
[[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]
```

```
new_instance = [[13.4, 2.5, 2.6, 19.4, 100.0, 2.9, 2.5, 0.3, 1.3, 3.0, 1.0, 2.2, 680]]
svm_prediction = sv.predict(new_instance)
print("Predicted wine class with SVM:", wine.target_names[svm_prediction[0]])
```

O/P:-

Predicted wine class with SVM: class_1.

(3Q) Demonstration of clustering algorithms using

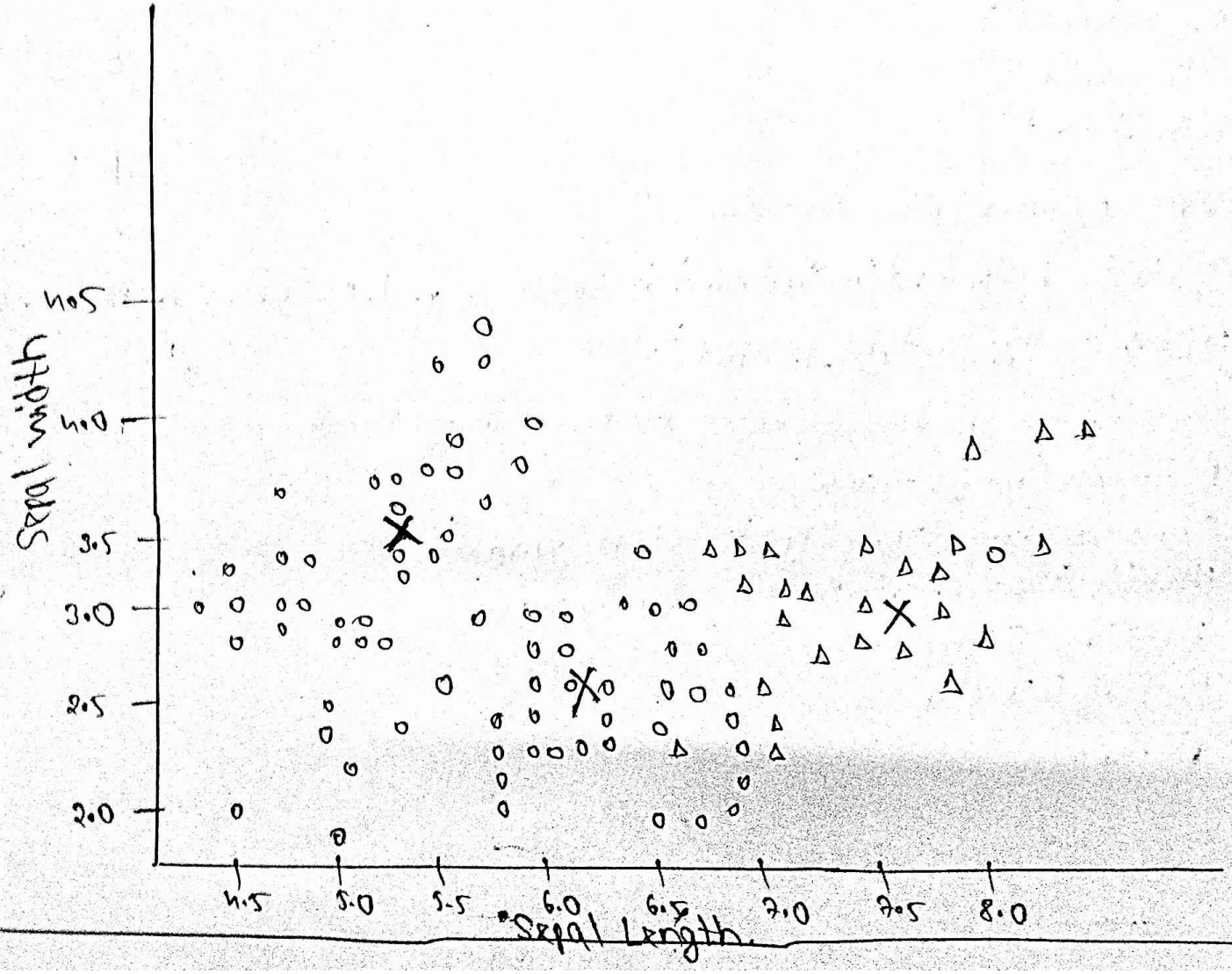
(a) K-means.

Program:-

```
from sklearn.datasets import load_iris
from sklearn.cluster import kmeans
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

iris = load_iris()
x = iris.data
y = iris.target
# Apply K-means clustering
kmeans = kmean(n_clusters=3, random_state=42) # we know 3 species.
kmeans.fit(x)
# Predict the clusters
y_kmeans = kmeans.Predict(x)
# visualizing the clusters
sns.set(style="whitegrid")
plt.figure(figsize=(10,6))
```

```
# Plotting Using the first two features for simplicity.
plt.scatter(X[:,0], centers[:,1], c=y_kmeans, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0], centers[:,1], c='red', s=200, marker='x') # centroids
plt.title('K-means clustering on Iris Dataset')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.show()

# Evaluation (comparing with actual labels)
df = pd.DataFrame({'True label': y, 'Cluster Label': y_kmeans})
print(df.head())
```

O/P:-



	True Label	Cluster Label
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

* Hierarchical algorithm - Agglomerative clustering

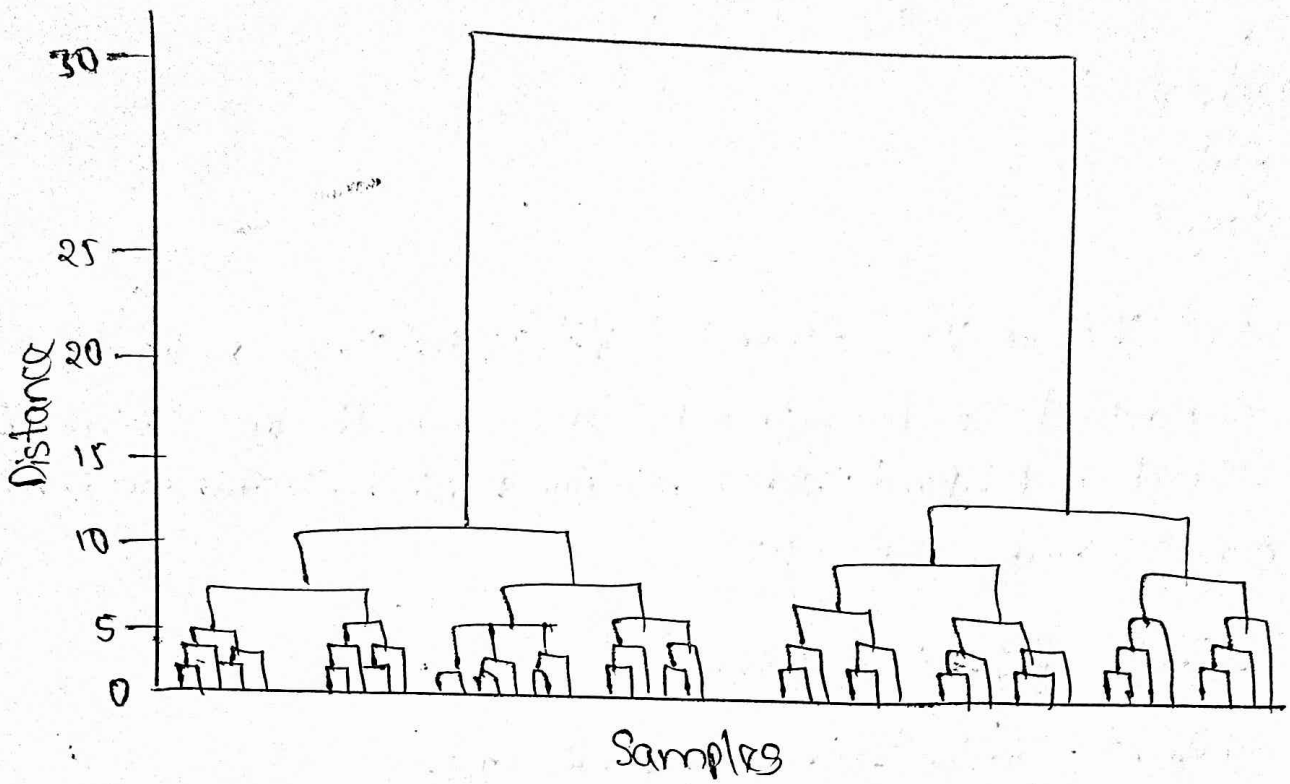
Program:-

```
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris

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

# Apply Agglomerative Clustering
agg_clust = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')
agg_labels = agg_clust.fit_predict(x)

#visualizing the Hierarchical Clustering using Dendrogram.
plt.figure(figsize=(10,6))
sch.dendrogram(sch.linkage(x, method='ward'))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('samples')
plt.ylabel('Distances')
plt.show()
```

#Evaluating the clustering results

```
df_agg = pd.Dataframe({'True Label': y, 'Agglomerative Label': agg_labels})
print(df_agg.head())
```

O/P:-



	True Labels	Agglomerative Label
0	0	1
1	0	1
2	0	1
3	0	0
4	0	1

(4a) Demonstrate ensemble techniques like boosting, bagging, random forests.

(i) Boosting:

Program:-

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
# Create an AdaBoost Classifier with a Decision Tree as the base classifier
adaboost_classifier = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
n_estimators=50, random_state=42)

adaboost_classifier.fit(X_train, y_train)
y_pred_adaboost = adaboost_classifier.predict(X_test)
# Evaluate the model
accuracy_adaboost = accuracy_score(y_test, y_pred_adaboost)
print(f'AdaBoost Classifier Accuracy: {accuracy_adaboost:.2f}')
```

O/P:

AdaBoost Classifier Accuracy: 1.00

(ii) Bagging

Program:-

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score.

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

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=42)
# Create a Bagging classifier with a Decision Tree as the base classifier
bagging_classifier = BaggingClassifier(estimator=DecisionTreeClassifier(),
    n_estimators=50, random_state=42)

bagging_classifier.fit(x_train, y_train)
y_pred = bagging_classifier.predict(x_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Bagging Classifier Accuracy: {accuracy:.2f}')
```

O/P:-

Bagging Classifier Accuracy: 1.00

(iii) Random forests

Program:-

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
#Create a Random forest classifier
rf_classifier = RandomForestClassifier(n_estimators=50, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)
#Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Random forest classifier accuracy: {accuracy_rf:.2f}')
```

O/P:-

Random forest classifier Accuracy: 1.00

Q) Build a classifier, compare its performance with an ensemble technique like random forest.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split.
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

cancer = load_breast_cancer()
x = cancer.data
y = cancer.target

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)

dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(x_train, y_train)
dt_pred = dt_clf.predict(x_test)

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(x_train, y_train)
rf_pred = rf_clf.predict(x_test)

print("Decision Tree classifier performance:")
print("Accuracy:", accuracy_score(y_test, dt_pred))
print("classification Report:\n", classification_report(y_test, dt_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, dt_pred))
```

```
print("\n Random forest Classifier Performance:4")
print("Accuracy:", accuracy_score(y_test, rf_pred))
print("classification Report:\n", classification_report(y_test, rf_pred))
print("confusion matrix:\n", confusion_matrix(y_test, rf_pred))
```

O/P:-

Decision Tree classifier Performance:

Accuracy: 0.94736

Classification Report:

	Precision	recall	f1-score	support
0	0.93	0.93	0.93	43
1	0.96	0.96	0.96	71
accuracy			0.95	114
macro avg	0.94	0.94	0.94	114
weighted avg	0.95	0.95	0.95	114

Confusion matrix:

```
[[40  3]
 [ 3 68]]
```

Random forest Classifier Performance:

Accuracy: 0.9649

Classification Report:

	Precision	recall	f1-score	support
0	0.98	0.93	0.95	43
1	0.96	0.99	0.97	71
accuracy			0.96	114
macro avg	0.97	0.96	0.96	114
weighted avg	0.97	0.96	0.96	114

Confusion matrix:

```
[[40  3]
 [ 1 70]]
```

17.

(60) Evaluate various classification algorithms performance on a dataset using various measures like True positive rate, false positive rate, Precision, recall.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report

data = load_breast_cancer()
x = data.data
y = data.target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=200),
    "Decision Tree": DecisionTreeClassifier(),
    "SVM": SVC(kernel='linear', probability=True)
}

results = { }

for name, clf in classifiers.items():
        clf.fit(x_train, y_train)
        y_pred = clf.predict(x_test)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        tpr = tp/(tp+fn) # True positive Rate
        fpr = fp/(fp+tn) # false positive Rate
```

```
precision = tp/(tp+fp) if tp+fp>0 else 0
recall = tp/(tp+fn) if tp+fn>0 else 0

results[name] = {
    "True positive Rate": tpr,
    "false positive Rate": fpr,
    "Precision": precision,
    "Recall": recall
}

for name, metrics in results.items():
    print(f"\nClassifier: {name}");
    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")
```

O/P:-

Classifier: Logistic Regression
True positive Rate: 0.9859
false positive Rate: 0.0930
Precision: 0.9459
recall: 0.9859

Classifier: SVM
True positive Rate: 0.9859
false positive Rate: 0.0930
Precision: 0.9459
recall: 0.9859.

Classifier: Decision Tree
True positive Rate: 0.9577
false positive Rate: 0.930
Precision: 0.9444
recall: 0.9577

Demonstrate GA for optimization (minimization or maximization Problem)

```
import numpy as np
import random
# objective function to minimize
    def fitness_function(x):
        return x**2 + 4*x + 4
# Generate an initial population
 def generate_population(size, lower_bound, upper_bound):
     return [random.uniform(lower_bound, upper_bound) for _ in range(size)]
#selection: choose the best individuals (based on fitness)
 def select_parents(population, fitnesses):
     sorted_population = [x for _, x in sorted(zip(fitnesses, population))]
     return sorted_population[:2] # select two best individuals
# Crossover: combine two parents to create offspring
 def crossover(parent1, parent2):
     return (parent1 + parent2)/2 #simple midpoint crossover
# Mutation: Introduce randomness to avoid local minima
 def mutate(individual, mutation_rate, lower_bound, upper_bound):
     if random.random() < mutation_rate:
         individual += random.uniform(-1, 1) #small random adjustments
         individual = np.clip(individual, lower_bound, upper_bound) # Ensure bounds
     return individual
# Main Genetic Algorithm
 def genetic_Algorithm(fitness_function, lower_bound, upper_bound,
     population_size=10, generations=50, mutation_rate=0.1):
```

```
#Step1: Initialize Population
population = generate_population(population_size, lower_bound, upper_bound)

for generation in range(generations):
    #Step2: Calculate fitness for each individual
    fitnesses = [fitness_function(x) for x in population]
    #Step3: Select parents
    parent1, parent2 = select_parents(population, fitnesses)
    #Step4: Generate offspring using Crossover
    offspring = crossover(parent1, parent2)
    #Step5: Apply mutation
    offspring = mutate(offspring, mutation_rate, lower_bound, upper_bound)
    #Step6: Replace the worst individual with the new offspring
    worst_index = fitnesses.index(max(fitnesses)) #Index of the worst
                                                   individual.
    population[worst_index] = offspring.
    #Logging for each generation
    best_fitness = min(fitnesses)
    best_individual = population[fitnesses.index(best_fitness)]
    print(f"Generation {generation+1}: Best fitness = {best_fitness},
    Best individual = {best_individual}")
#Final Result
best_fitness = min(fitnesses)
best_individual = population[fitnesses.index(best_fitness)]
return best_individual, best_fitness
```

19.

```
# Run the Genetic Algorithm
lower_bound = -10
upper_bound = 10
best_individual, best_fitness = genetic_algorithm(fitness_function, lower_bound,
                                                  upper_bound)

print("\n Optimal solution:")
print("Best individual (x):", best_individual)
print("Best fitness (f(x)):", best_fitness)
```

Op:-

Generation 1: Best fitness = 0.05657, Best individual = -2.2385

Generation 50: Best fitness: 0.0, Best individual = -2.00

Optimal solution:

Best individual (x): -2.000000139000428 4

Best fitness (f(x)): 0.0

(8) Case study on supervised/unsupervised learning algorithm:

(a) Handwritten digits classification using CNN

(b) Text classificatn using python libraries

(a) Case study: Handwritten Digits Classification using CNN.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
                                    Dense.
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist

# Load the MNIST dataset.
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data.
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32') / 255
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Define the CNN model.
model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
        MaxPooling2D((2,2)),
        Conv2D(64, (3,3), activation='relu'),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
```

20.

```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.1)
# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy}")
# Predict a Sample.
Sample = x_test[0].reshape(1,28,28,1)
Prediction = model.predict(Sample).argmax()
print(f'Predicted Digit: {Prediction}")
```

O/P:-

Epoch 1/5

844/844 — 36s 38ms/step - accuracy: 0.8863 - loss: 0.3838 - val_accuracy: 0.9803 - val_loss: 0.0689

Epochs 5/5

844/844 — 32s 38ms/step - accuracy: 0.9945 - loss: 0.0164 - val_accuracy: 0.9910 - val_loss: 0.0320

313/313 — 3s 10ms/step - accuracy: 0.9867 - loss: 0.0419

Test Accuracy: 0.99019998

1/1 — 0s 203ms/step

Predicted Digit: 7.

(b) Case study:- Text classification using python Libraries.

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.datasets import imdb

# Load the IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)
#Decode reviews into text
word_index = imdb.get_word_index()
index_word = {value:key for key, value in word_index.items()}

def decode_review(encoded_review):
    return ' '.join([index_word.get(i, '?') for i in encoded_review if i>2])

X_train_text = [decode_review(review) for review in x_train]
X_test_text = [decode_review(review) for review in x_test]

# Text preprocessing: Vectorization
vectorizer = CountVectorizer(max_features=5000, stop_words='english')
X_train_vectorized = vectorizer.fit_transform(X_train_text)
X_test_vectorized = vectorizer.transform(X_test_text)
#Train a naive Bayes classifier.
clf = MultinomialNB()
clf.fit(X_train_vectorized, y_train)
#make predictions
y_pred = clf.predict(X_test_vectorized)
```

```
# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

O/P:-

Accuracy: 0.83216

Classification Report:

	precision	recall	f1-score	support
0	0.81	0.86	0.84	12500
1	0.85	0.80	0.83	12500
accuracy			0.83	25000
macro avg	0.83	0.83	0.83	25000
weighted avg	0.83	0.83	0.83	25000