CONTENT

EX.No : 1	**Write a python program to compute the Central Tendency Measures: Mean, Median, Mode, Measure of dispersion: Variance, Standard deviation**
DATE :	

AIM:

To create a python program to compute the Central Tendency Measures: Mean, Median, Mode, and Measure of dispersion: Variance, Standard deviation.

ALGORITHM:

Step 1: Start the program.

Step 2: Initialize the list of numbers.

Step 3: Calculate the Mean and Meadian.

Step 4: Calculate the Variance and display the variance.

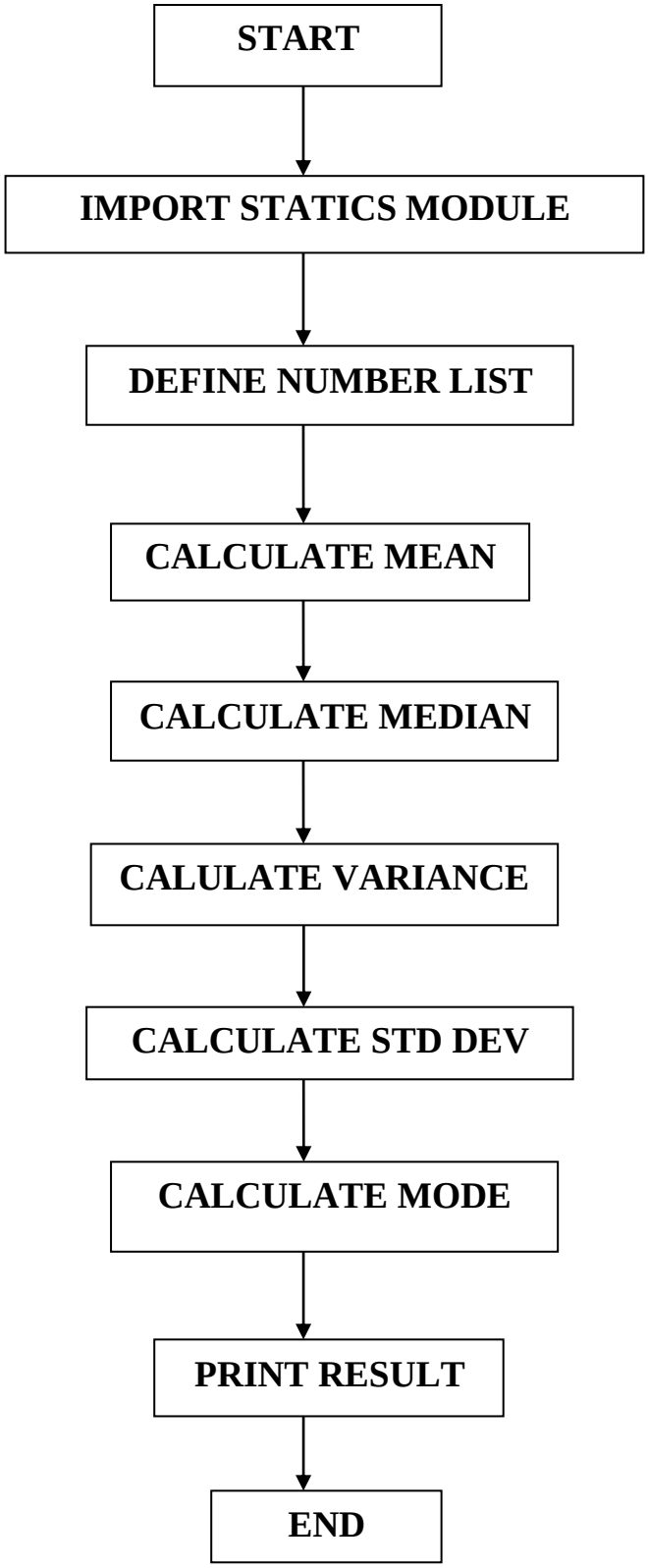
Step 5: And calculating the Standard Deviation using Stdev function.

Step 6: Calculate the Mode values.

Step 7: Finally print the whole values.

Step 8: End the Program.

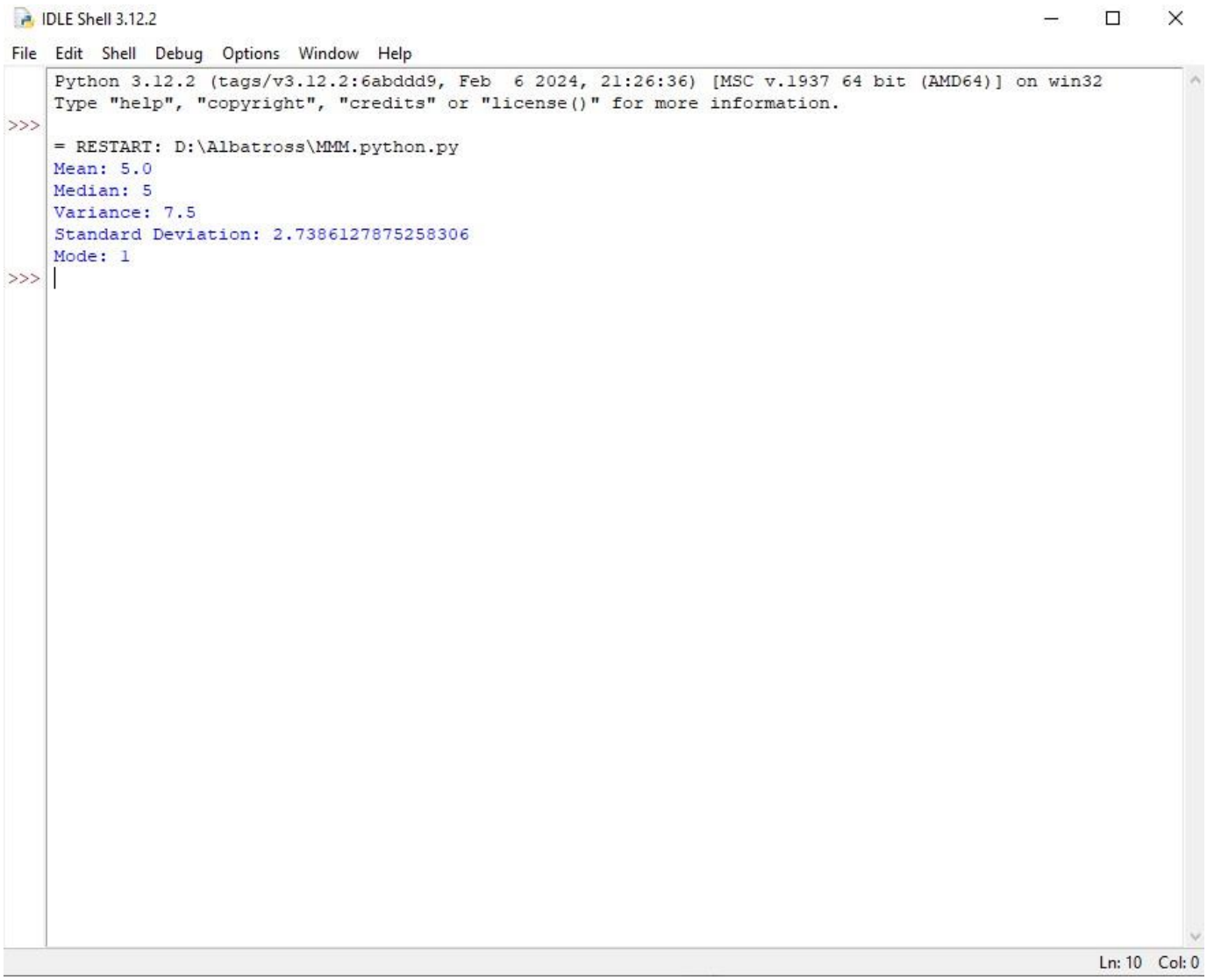
FLOWCHART:


START
IMPORT STATICS MODULE
DEFINE NUMBER LIST
CALCULATE MEAN
CALCULATE MEDIAN
CALULATE VARIANCE
CALCULATE STD DEV
CALCULATE MODE
PRINT RESULT
END

SOURCE CODE:

```
import statistics
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
mean = sum(numbers) / len(numbers)
print("Mean:", mean)
median = statistics.median(numbers)
print("Median:", median)
variance = statistics.variance(numbers)
print("Variance:", variance)
std_dev = statistics.stdev(numbers)
print("Standard Deviation:", std_dev)
mode = statistics.mode(numbers)
print("Mode:", mode)
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 2	**Implement a Linear Regression and Multiple Linear Regression with a Real dataset.**
DATE :	

AIM:

To Implement a Linear Regression and multiple linear regression with a real dataset using python.

ALGORITHM:

Step 1: Ensure that Numpy and Matplotlib are installed.
You can install it using pip in command prompt.

Step 2**:** Import libraries like Numpy and matplotlib.

Step 3: Define Dataset generation function.

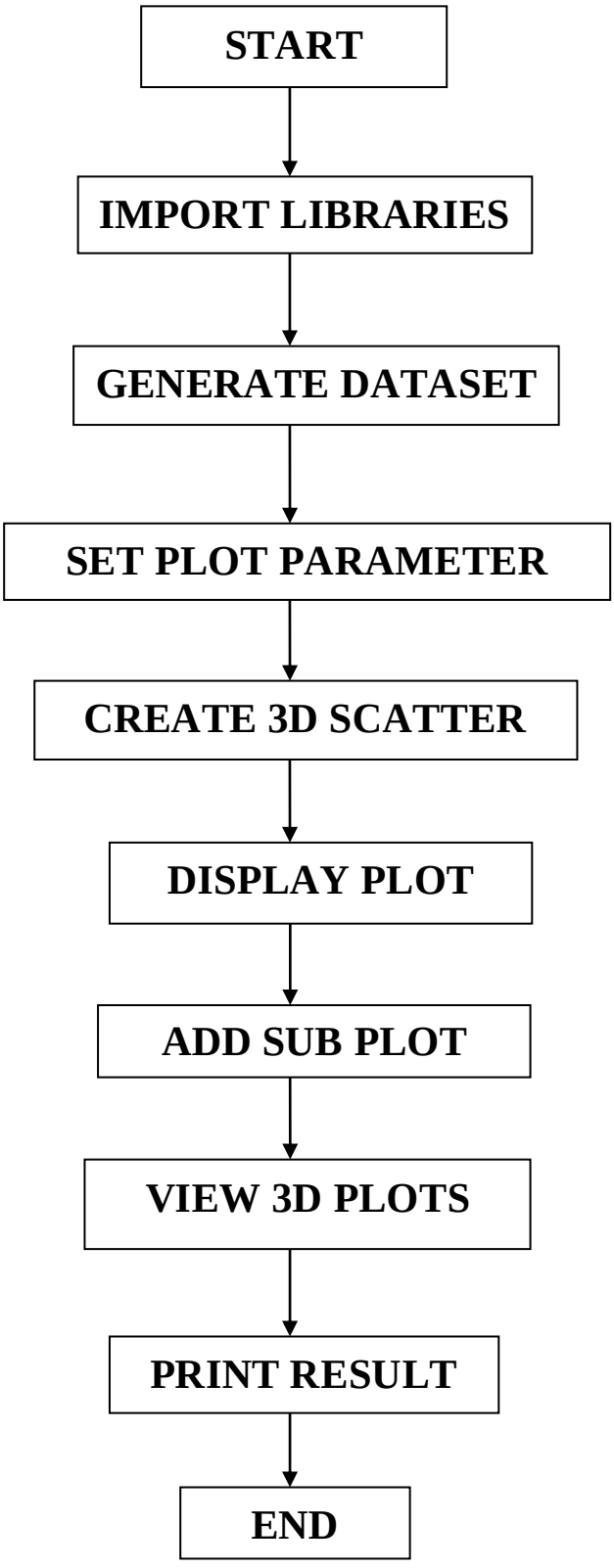
- Using generate_dataset(n).

Step 4: Prepare the plotting parametes and set the dataplots.

Step 5: We have to create 3D scatter plot and add a 3D subplots.

Step 6: End the program gracefully, possible with a user-friendly message.

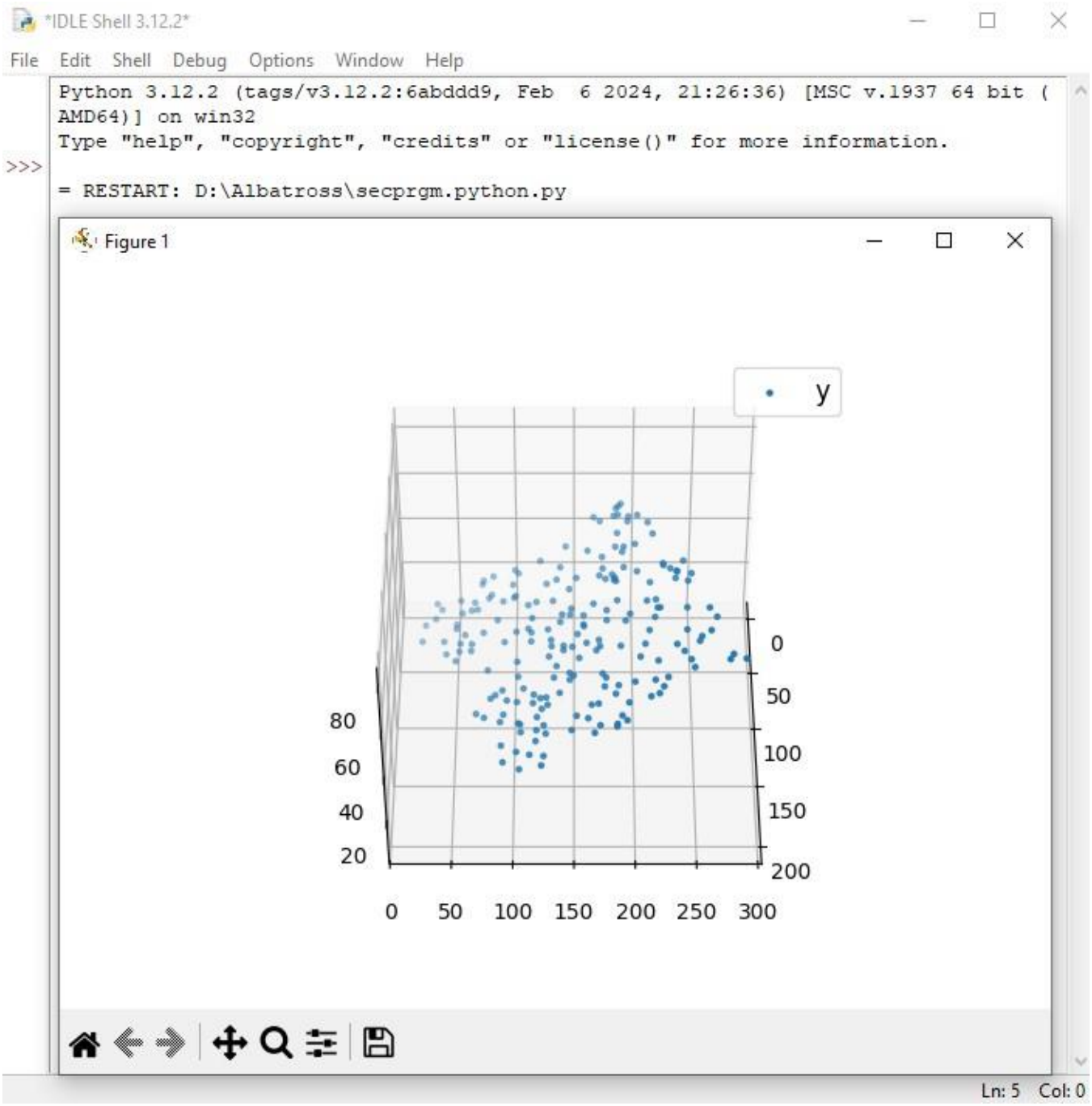
FLOW CHART:


START
IMPORT LIBRARIES
GENERATE DATASET
SET PLOT PARAMETER
CREATE 3D SCATTER
DISPLAY PLOT
ADD SUB PLOT
VIEW 3D PLOTS
PRINT RESULT
END

SOURCE CODE:

```
import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
def generate_dataset(n):
x = []
y = []
random_x1 = np.random.rand()
random_x2 = np.random.rand()
for i in range(n):
x1 = i
x2 = i/2 + np.random.rand()*n
x.append([1, x1, x2])
y.append(random_x1 * x1 + random_x2 * x2 + 1)
return np.array(x), np.array(y)
x, y = generate_dataset(200)
mpl.rcParams['legend.fontsize'] = 12
fig = plt.figure()
ax = fig.add_subplot(projection ='3d')
ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)
ax.legend()
ax.view_init(45, 0)
plt.show()
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 3	**Implement of Logistic Regression using Sklearn.**
DATE :	

AIM:

To create and Implement of logistic regression using sklearn.

ALGORITHM:

Step 1: Ensure that Numpy and Matplotlib are installed.

You can install it using pip in command prompt.

Step 2: Import libraries like, numpy, pandas, sklearn using

scikit-learn command.

Step 3: Load the Dataset from sklearn, prepare the binary classification data with features of X and Y labels.

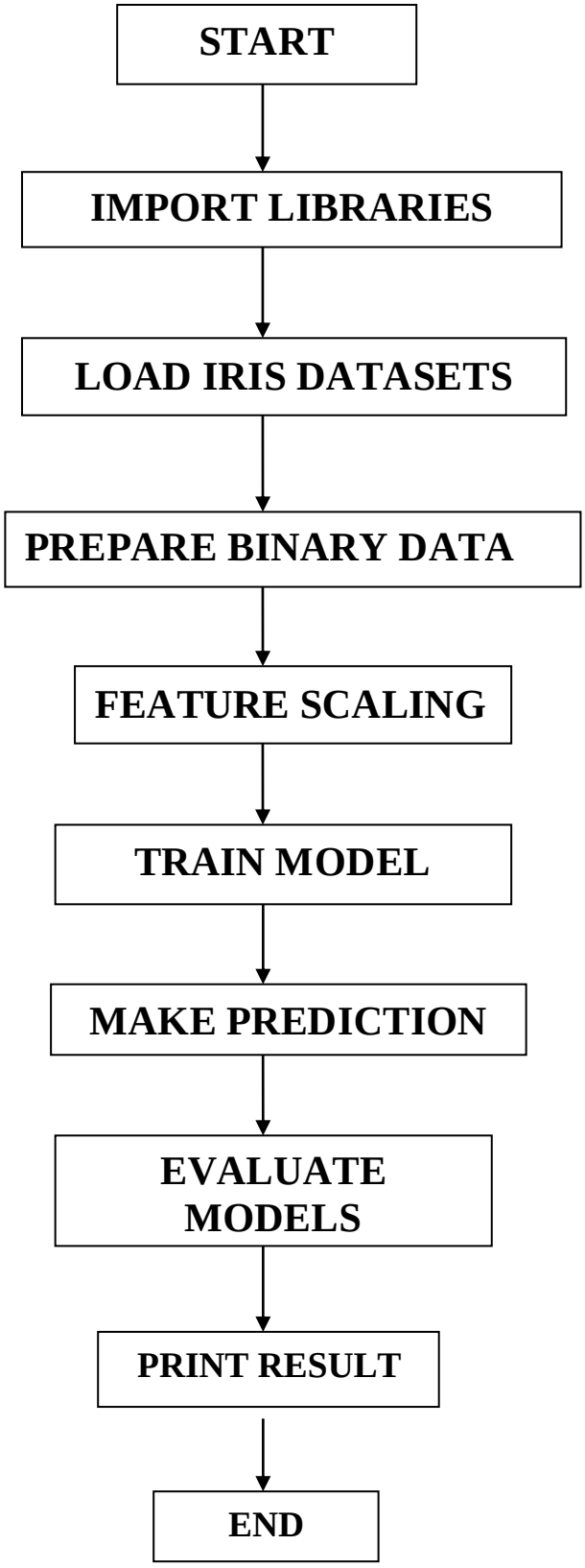
Step 4**:** Split the overall data's using train_test_split.

Step 5: Next we have to feature scaling and initialize Standard scaler values. Initialize the trained mode with Logistic Regression.

Step 6: Finally predict the labels, testing data and evaluate model.

Step 7: Get the results and End the program.

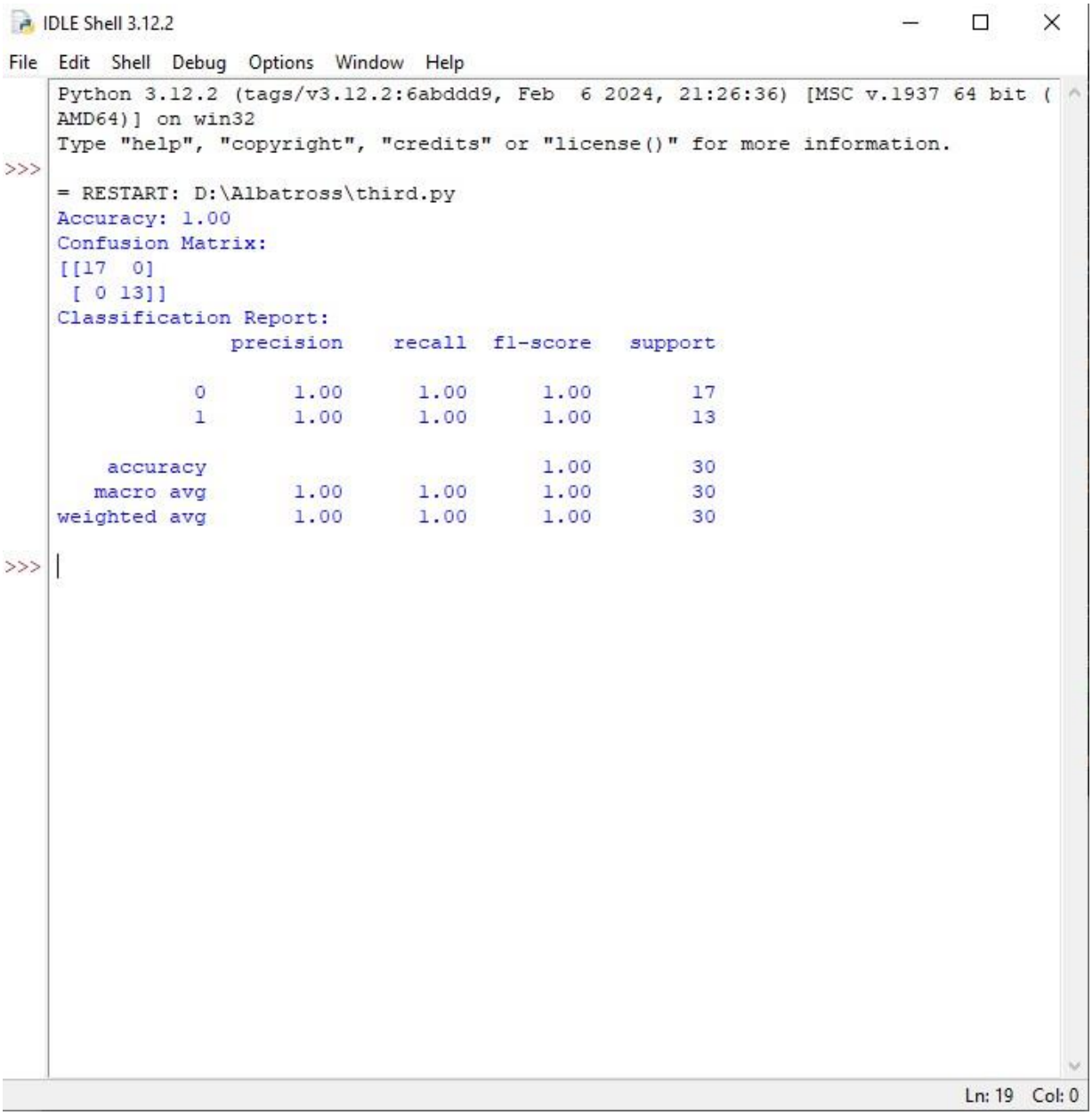
FLOWCHART:


START
IMPORT LIBRARIES
LOAD IRIS DATASETS
PREPARE BINARY DATA
FEATURE SCALING
TRAIN MODEL
MAKE PREDICTION
EVALUATE MODELS
PRINT RESULT
END

SOURCE CODE:

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
iris = datasets.load_iris()
X = iris.data
y = iris.target
binary_class_indices = np.where(y != 2)
X, y = X[binary_class_indices], y[binary_class_indices]
y = np.where(y == 0, 0, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 4	Implement a Binary classification model.
DATE :	

AIM:

To create a machine learning program using python and Implement a Binary Classification model.

ALGORITHM:

Step 1: Ensure that Numpy and Matplotlib are installed.
You can install it using pip in command prompt.

Step 2: Import libraries like, numpy, pandas, sklearn using scikit-learn command.

Step 3: Prepare the Binary classification data and split data.

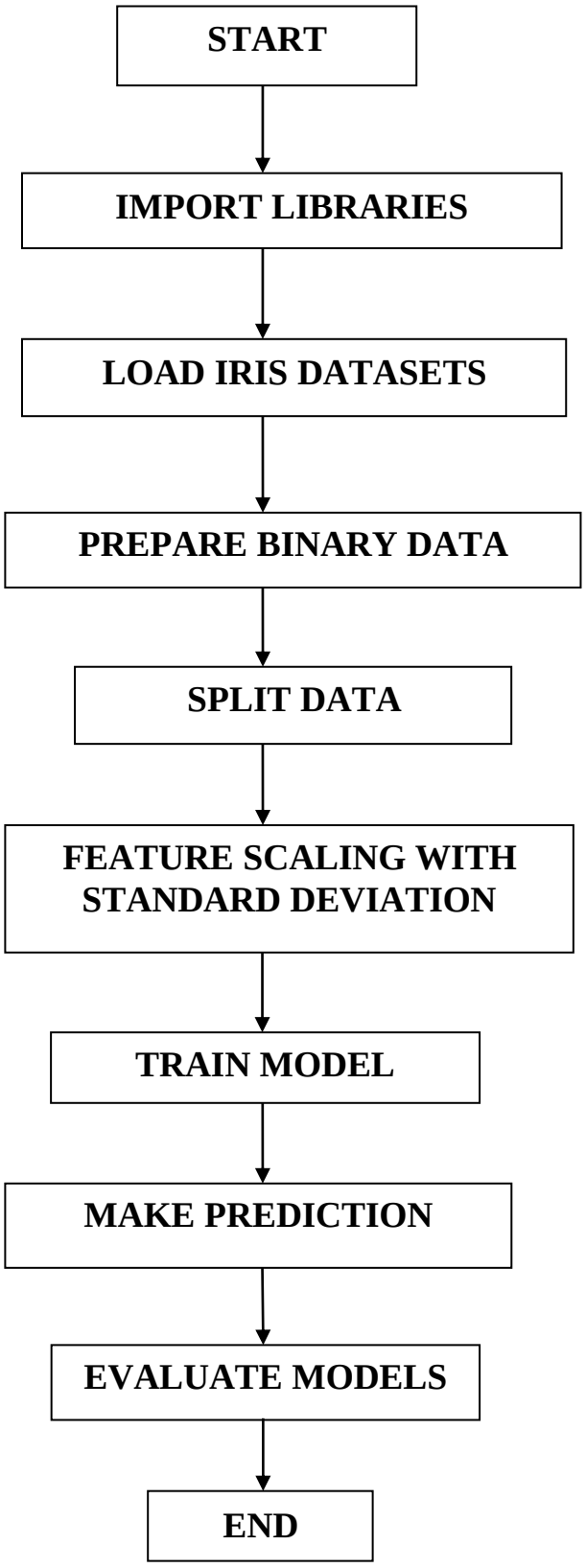
Step 4**:** Split the overall data's using train_test_split.

Step 5: Adding a binary values and establishing the performance.

Step 6: Finally predict the labels, testing data and evaluate model.

Step 7: Get the results and End the program.

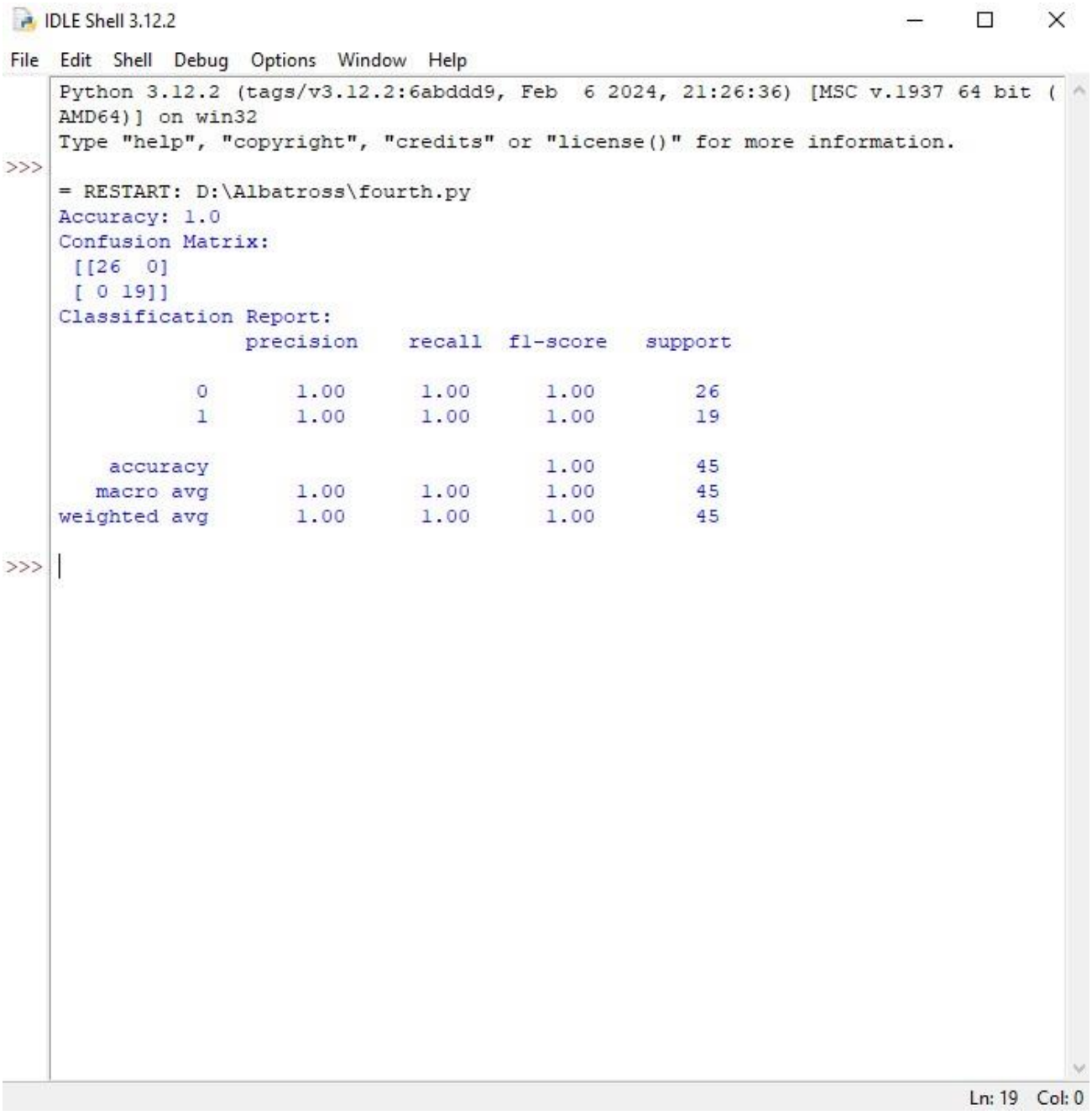
FLOWCHART:


START
IMPORT LIBRARIES
LOAD IRIS DATASETS
PREPARE BINARY DATA
SPLIT DATA
FEATURE SCALING WITH STANDARD DEVIATION
TRAIN MODEL
MAKE PREDICTION
EVALUATE MODELS
END

SOURCE CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
y_binary = (y == 0).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y_binary,
test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 5	Classification with Nearest Neighbours and NavieBaye Algorithm
DATE :	

AIM:

To create a program and classified with Nearest Neighbours and Naviebaye Algorithm.

ALGORITHM:

Step 1: Install the needed libraries using pip install in command prompt.

Step 2: Load the Dataset with the use of load_urus() from sklearn datasets.

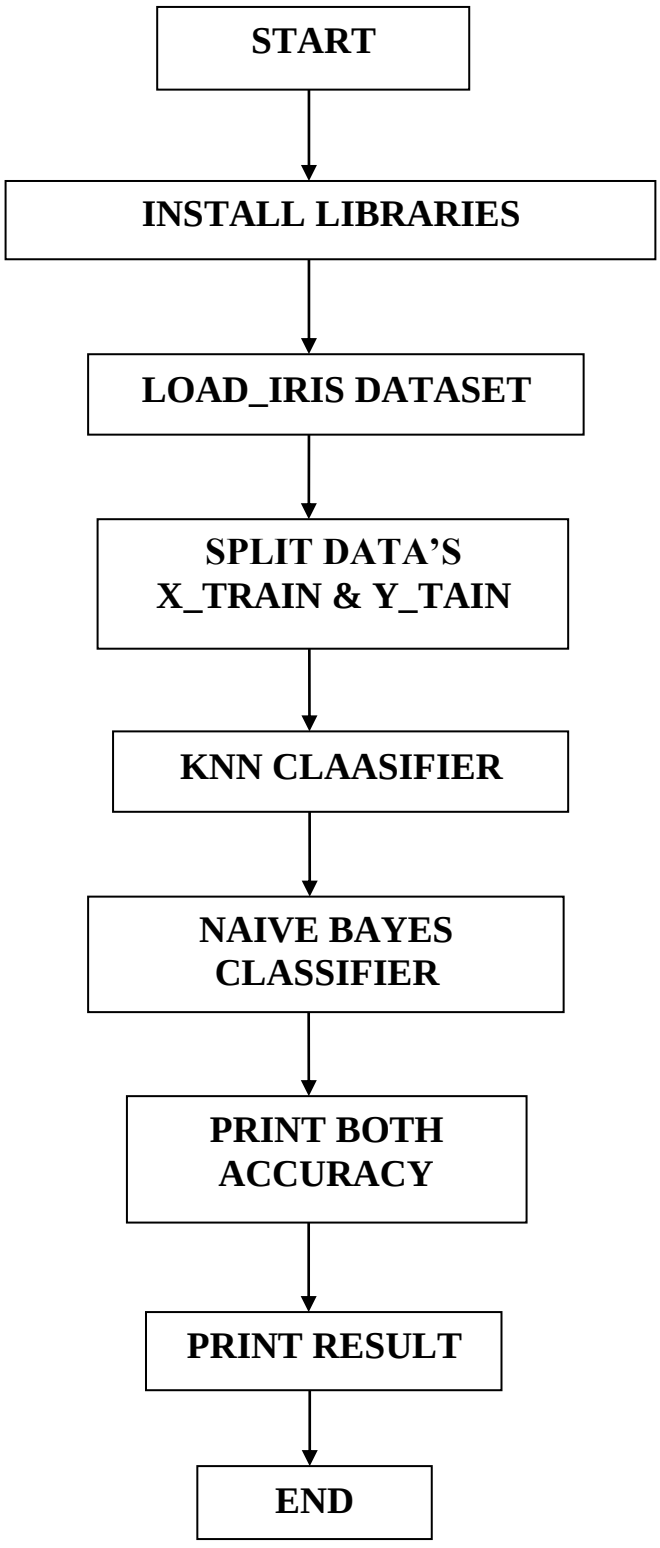
Step 3: We can split the data's using train_test-split() and set the random values using random_state = 42.

Step 4: Initialize the K-NN classifier with k=5 neighbors KNeighbors classifier and print the accuracy.

Step 5: And also we have to initialize the Naïve bayes classifier with (GaussianNBG) and print the values.

Step 6: Confirm that the KNN and Naïve Bayes initialized or not, get the Results and End the program.

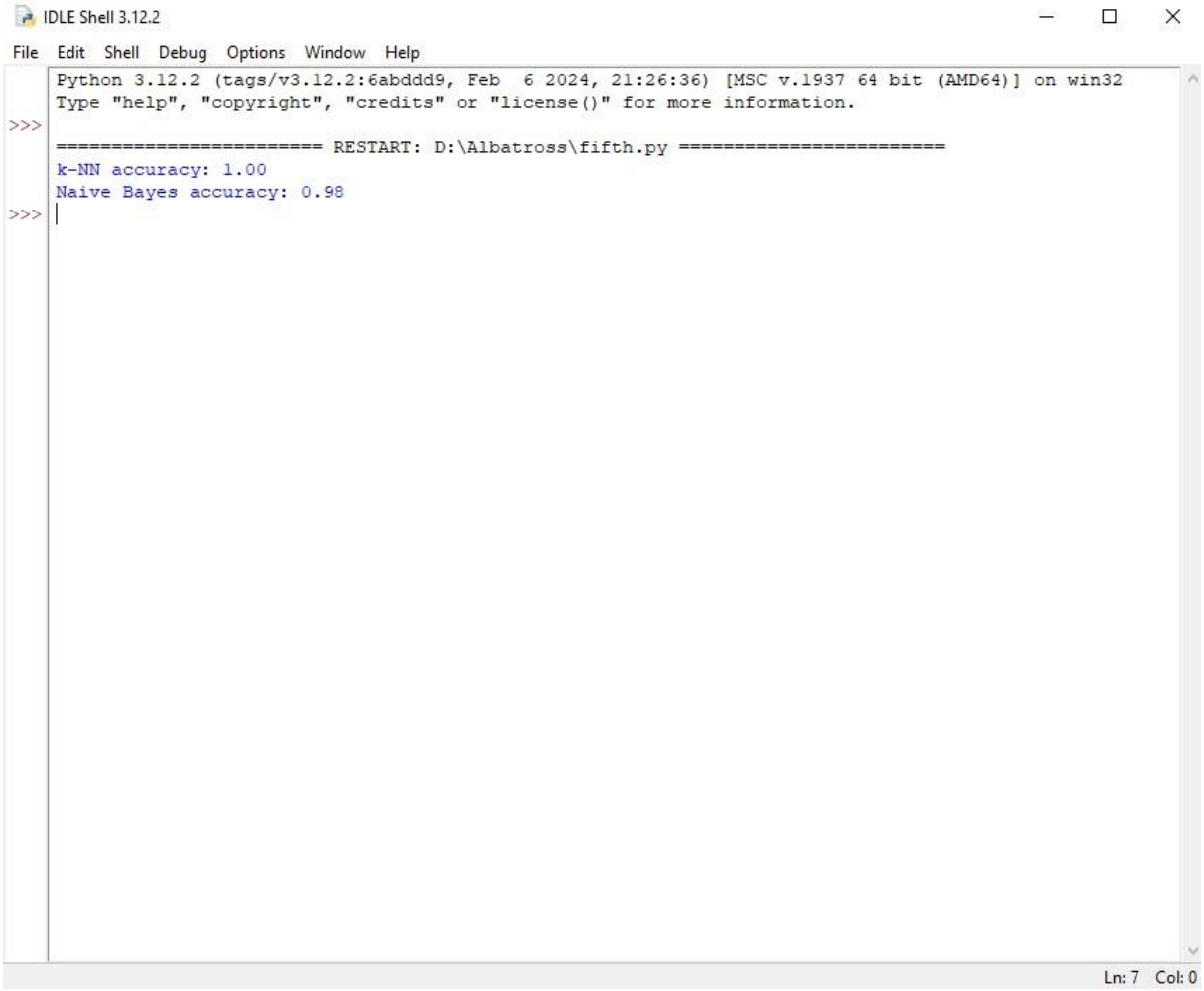
FLOW CHART:


START
INSTALL LIBRARIES
LOAD_IRIS DATASET
SPLIT DATA'S
X_TRAIN & Y_TAIN
KNN CLAASIFIER
NAIVE BAYES
CLASSIFIER
PRINT BOTH
ACCURACY
PRINT RESULT
END

SOURCE CODE:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'k-NN accuracy: {accuracy:.2f}')
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Naive Bayes accuracy: {accuracy:.2f}')
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 6 DATE :	**Implement Decision tree for classification using sklearn and its parameter tuning.**

AIM:

To create a Decision tree for classification using sklearn and its parameter tuning.

ALGORITHM:

Step 1: Import libraries for numerical operation, dataset handling, model training and evaluation.

Step 2: Load the dataset using load_iris() function with data's.

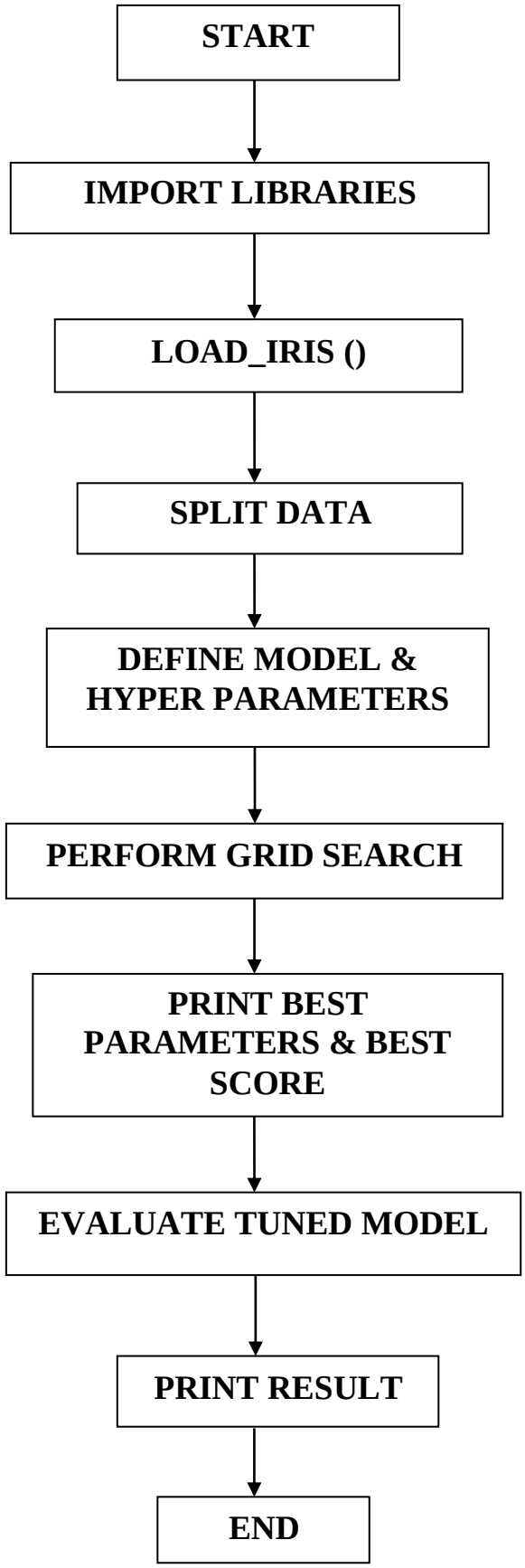
Step 3: Split the data using this command sklearn and model_selection.

Step 4: Define model and hyper parameters and also a decision tree classification.

Step 5: Initialize the Grid search with decision tree model and evaluate tuned models.

Step 6: Finally get the results and end the program.

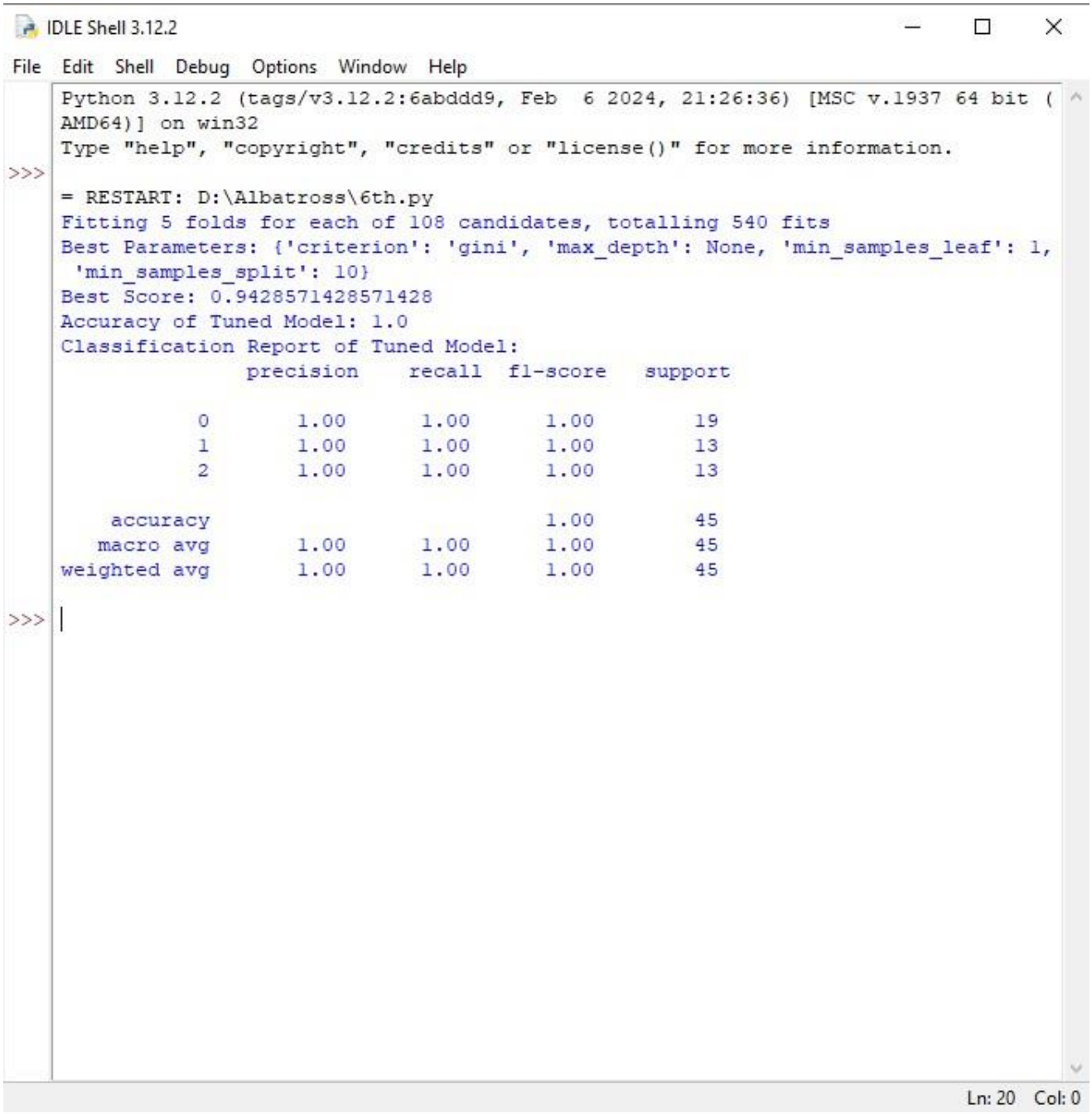
FLOW CHART:


START
IMPORT LIBRARIES
LOAD_IRIS ()
SPLIT DATA
DEFINE MODEL & HYPER PARAMETERS
PERFORM GRID SEARCH
PRINT BEST PARAMETERS & BEST SCORE
EVALUATE TUNED MODEL
PRINT RESULT
END

SOURCE CODE:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
dt = DecisionTreeClassifier()
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5,
n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
best_dt = grid_search.best_estimator_
y_pred_tuned = best_dt.predict(X_test)
print("Accuracy of Tuned Model:", accuracy_score(y_test, y_pred_tuned))
print("Classification Report of Tuned Model:\n", classification_report(y_test,
y_pred_tuned))
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 7	Implement the K-means algorithm.
DATE :	

AIM:

To create a program and implement using the K-means algorithm.

ALGORITHM:

Step 1: Import necessary library for numerical operation and data visualization and clustering.

Step 2: Generate synthetic data, use make_blobs () from sklearn in datasets.

Step 3: Initialize the K-means object with number of cluster sets and random_state.

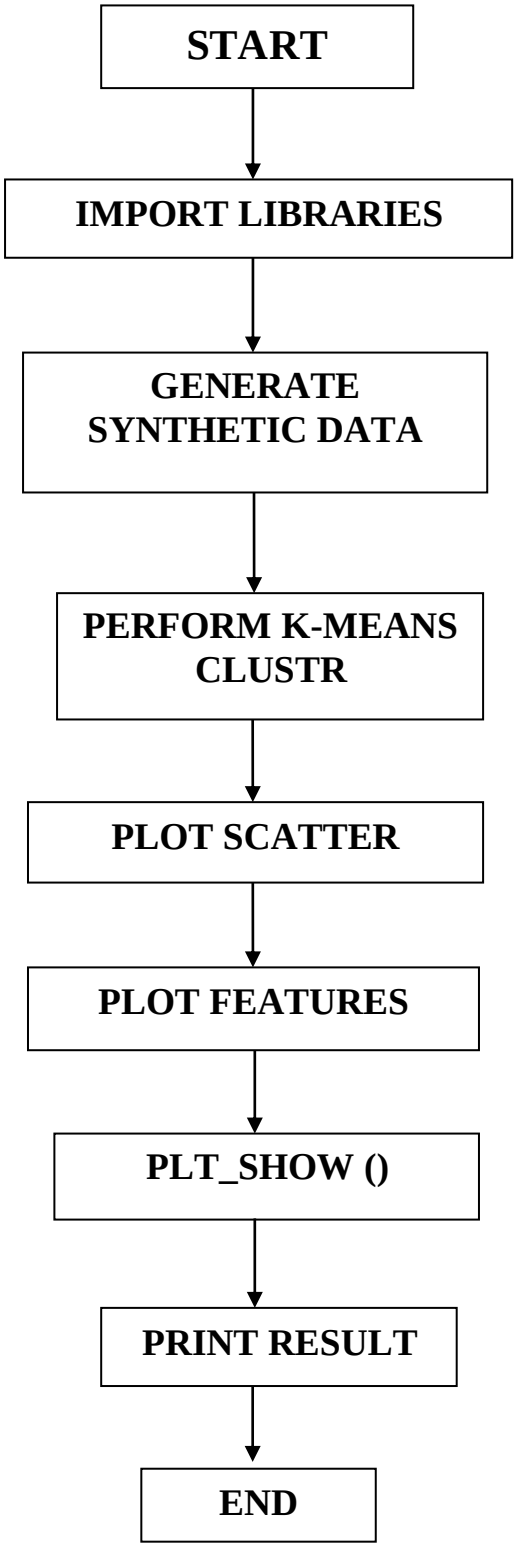
Step 4: Visualize clustering results, use color map like “viridis” to differentiate clusters.

Step 5: And also to overlay the cluster centers on plots use distint color red and mark X.

Step 6: Display the plotting point.

Step 7: End the program.

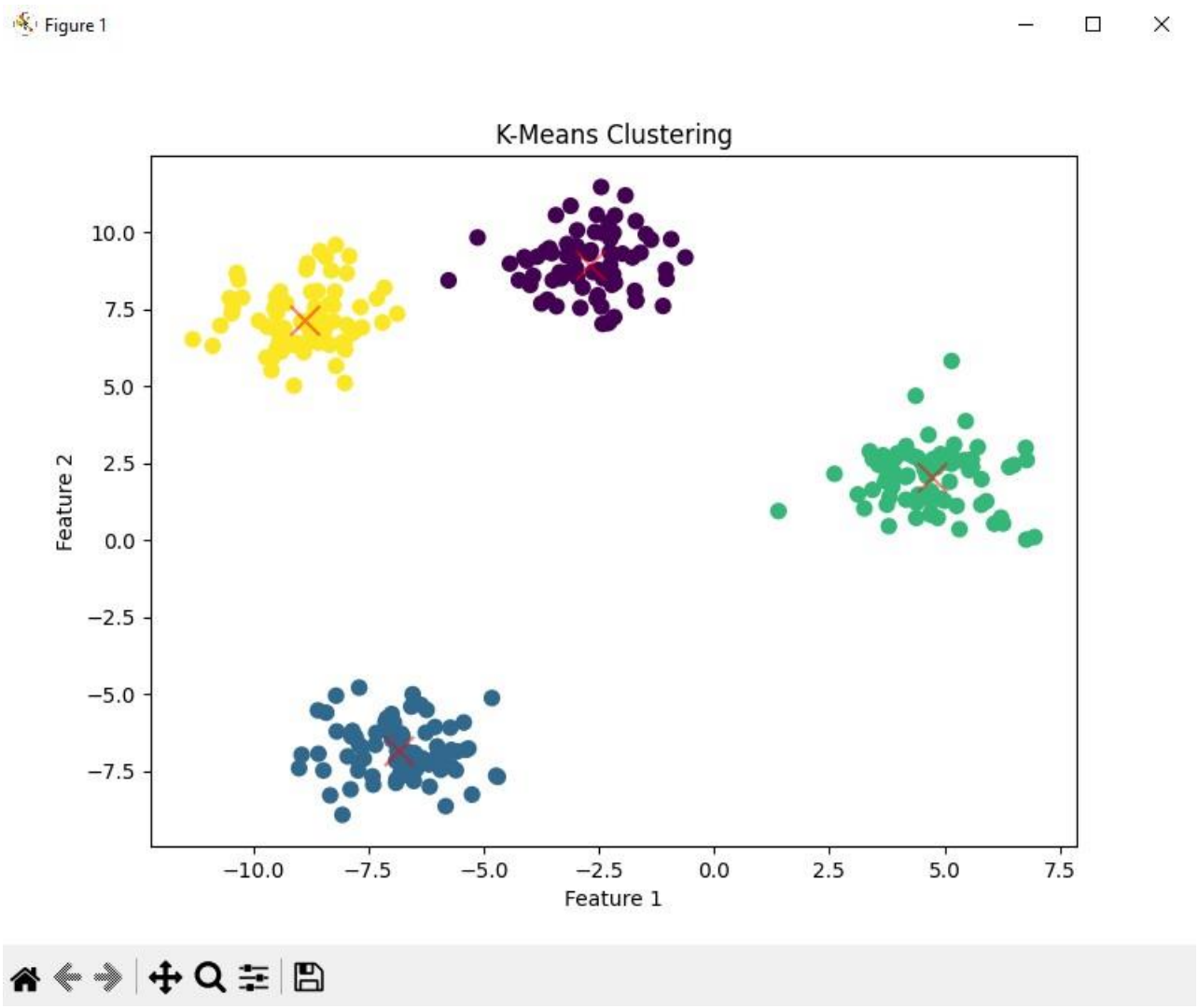
FLOW CHART:


START
IMPORT LIBRARIES
GENERATE SYNTHETIC DATA
PERFORM K-MEANS CLUSTR
PLOT SCATTER
PLOT FEATURES
PLT_SHOW ()
PRINT RESULT
END

SOURCE CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
n_samples = 300
n_clusters = 4
x, y = make_blobs(n_samples=n_samples, centers=n_clusters,
random_state=42)
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(x)
y_kmeans = kmeans.predict(x)
plt.figure(figsize=(8, 6))
plt.scatter(x[:, 0], x[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.5, marker='x')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 8	**Implement an Image classifier using CNN in TensorFlow/Keras.**
DATE :	

AIM:

To implement an image classifier using CNN in tensor flow/keras.

ALGORITHM:

Step 1: Import tensor flow and required modules from keras and import matplotlib for plotting.

Step 2: Load the dataset using cifar10.load_data ().

Step 3: Preprocess the data and normalize the image data to pixel.

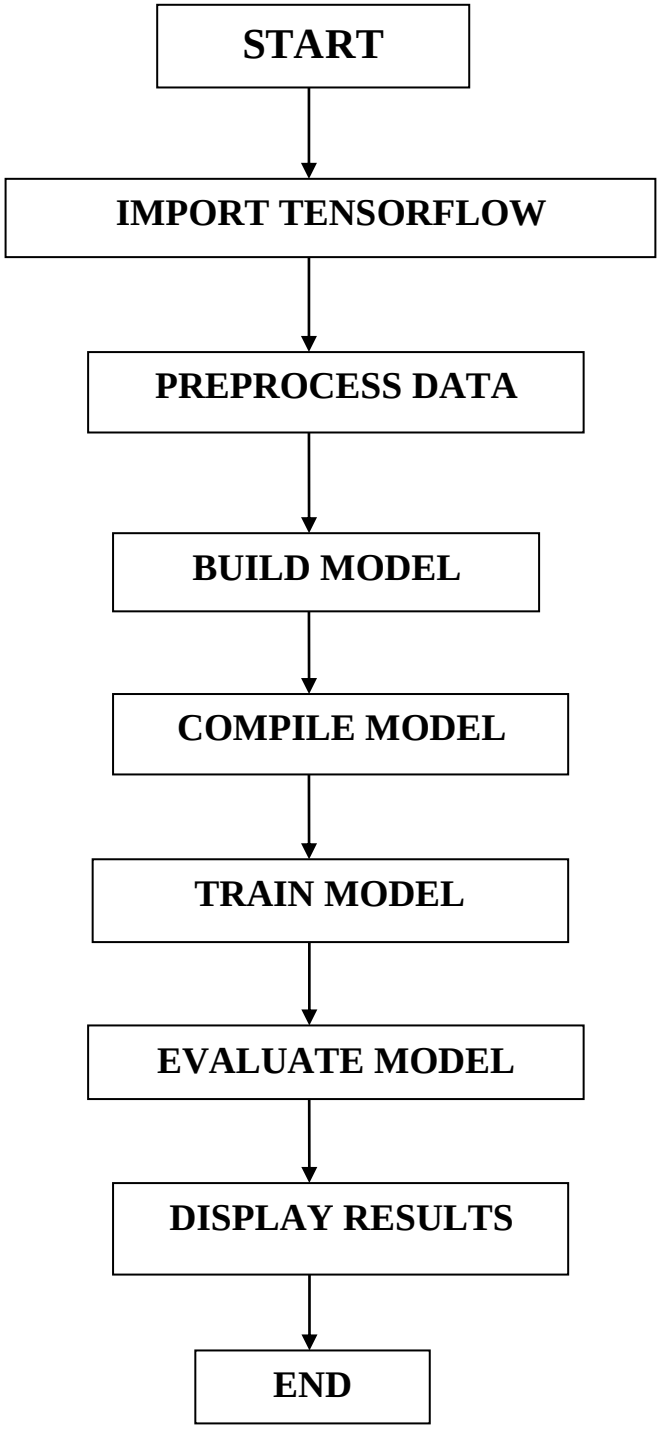
Step 4: Build the model and initialize a sequential models.

Step 5: Compile and evaluate the trained model.

Step 6: Print the test accuracy and plotted training results.

Step 7: End the program.

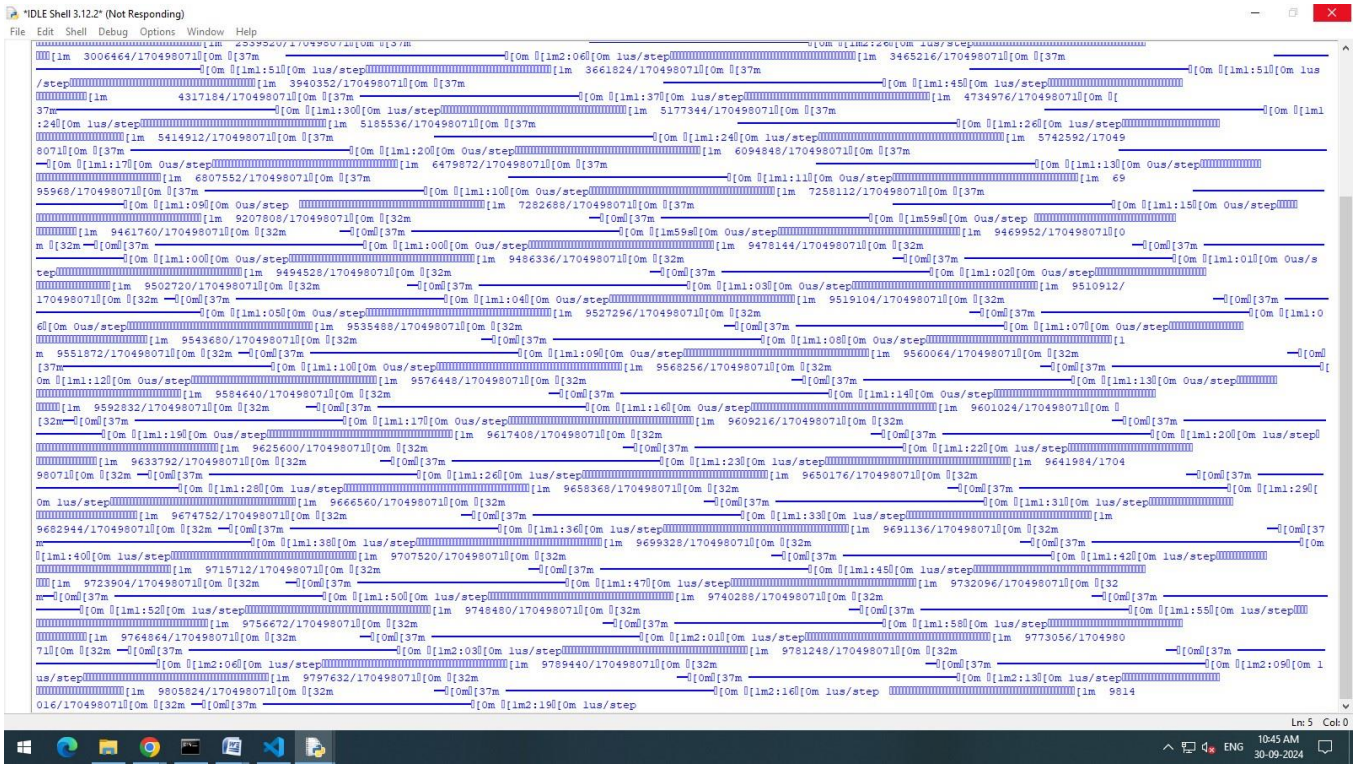
FLOW CHART:


START
IMPORT TENSORFLOW
PREPROCESS DATA
BUILD MODEL
COMPILE MODEL
TRAIN MODEL
EVALUATE MODEL
DISPLAY RESULTS
END

SOURCE CODE:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = tf.keras.utils.to_categoricaly(y_train,10)
y_test = tf.keras.utils.to_categoricaly(y_test,10)
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=
(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
loss='categorical_crossentropy',
metrics=['accuracy']
history = model.fit(x_train, y_train, epochs=10, batch_size=64,
validation_split=0.2)
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy:.3f}')
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 9	**Implement an Autoencoder in TensorFlow/Keras.**
DATE :	

AIM:

To implement an Autoencoder in tensor flow/ keras.

ALGORITHM:

Step 1: Import necessary libraries: Numpy, Matplotlib and keras modules.

Step 2: Load the MNIST dataset using mnist.load_data().

Step 3: Preprocess the data and normalize the image data to pixel.

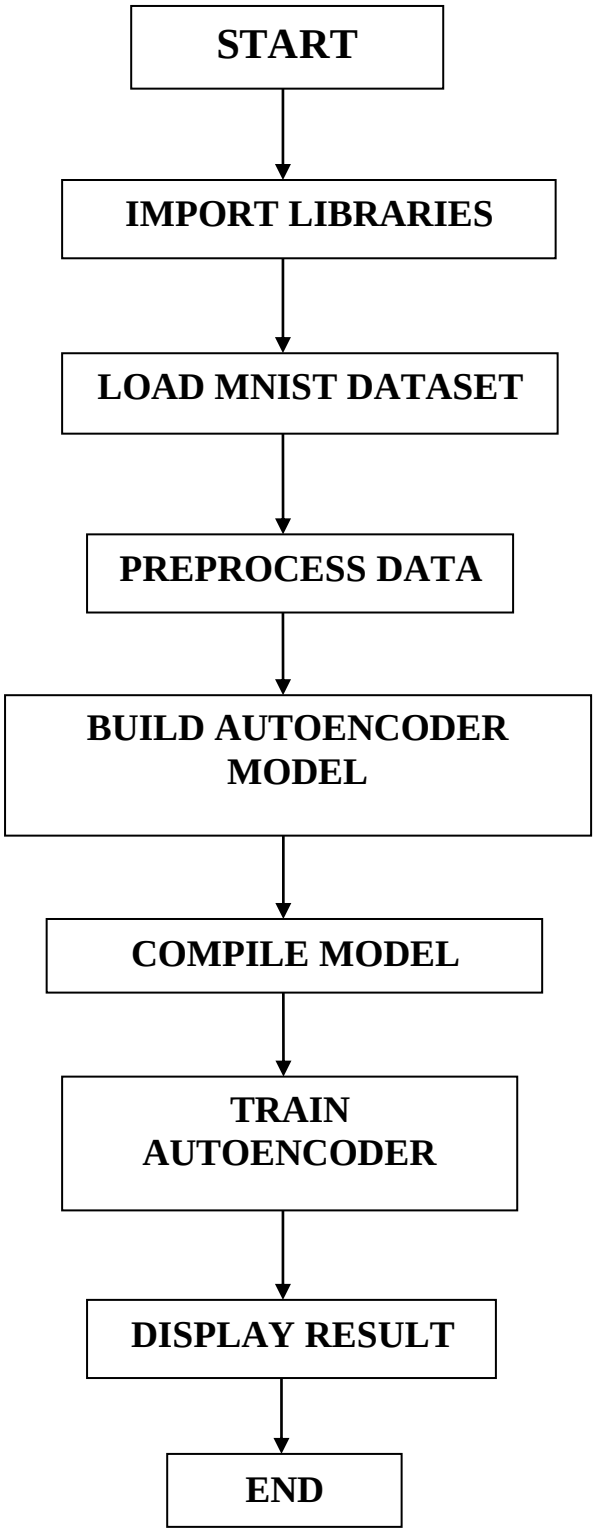
Step 4: Build the Auto encoder models.

Step 5: Compile and evaluate the trained model.

Step 6: Display the results.

Step 7: End the program.

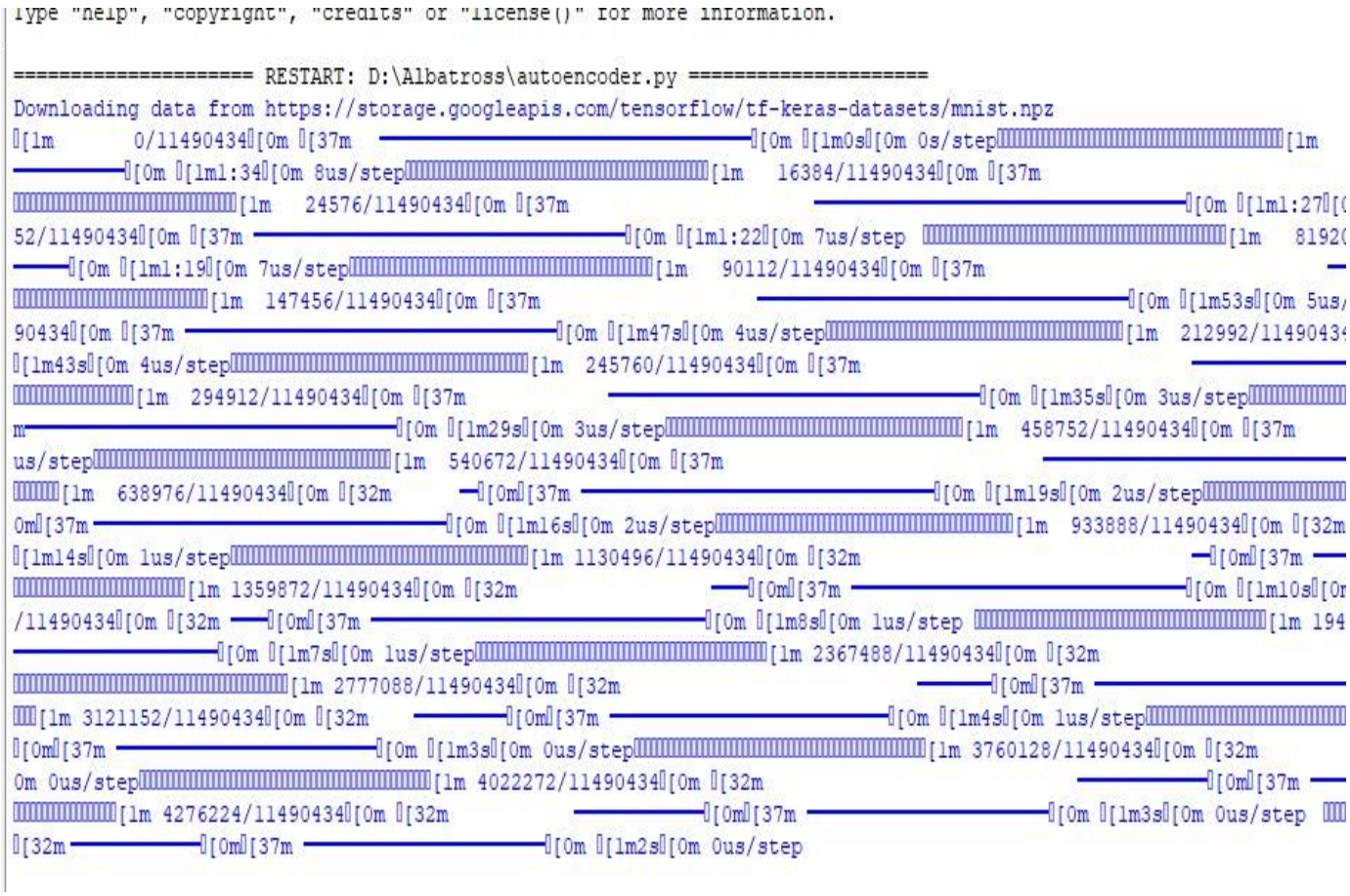
FLOW CHART:


START
IMPORT LIBRARIES
LOAD MNIST DATASET
PREPROCESS DATA
BUILD AUTOENCODER MODEL
COMPILE MODEL
TRAIN AUTOENCODER
DISPLAY RESULT
END

SOURCE CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
input_img = layers.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
autoencoder = models.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
decoded_imgs = autoencoder.predict(x_test)
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis('off')
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Reconstructed")
    plt.axis('off')
    plt.show()
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.

EX.No : 10	Implement a Simple STM using TensorFlow/Keras.
DATE :	

AIM:

To create and implement a simple STM using tensorflow/keras.

ALGORITHM:

Step 1: Import the Tensor flow and Matplotlib using command prompt.

Step 2: Generate the data's in np.linespace with X and Y axis.

Step 3: Prepare Training Data: Use prepare_data() to format the data into input-output pairs based on a specified time step.

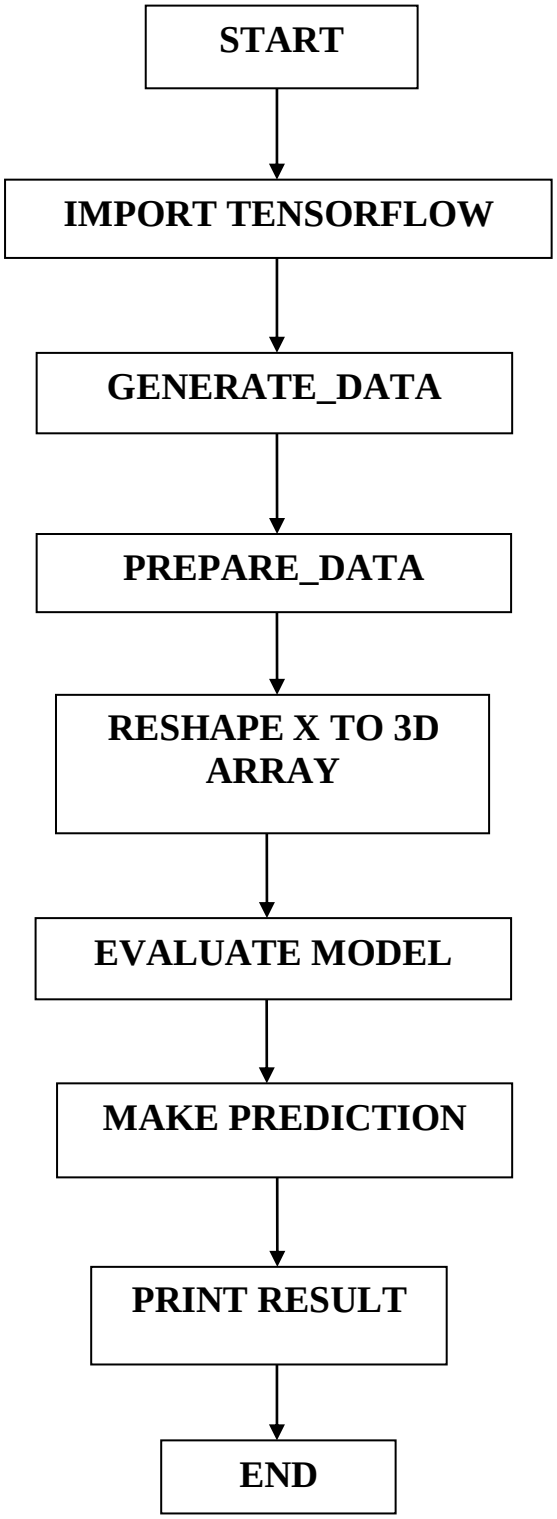
Step 4: After we have to reshape input data X to have 3D Array.

Step 5: Calculate the validation loss using the evaluate () method.

Step 6: Predict the output for validation set using predict ().

Step 7: Get the output and end the program.

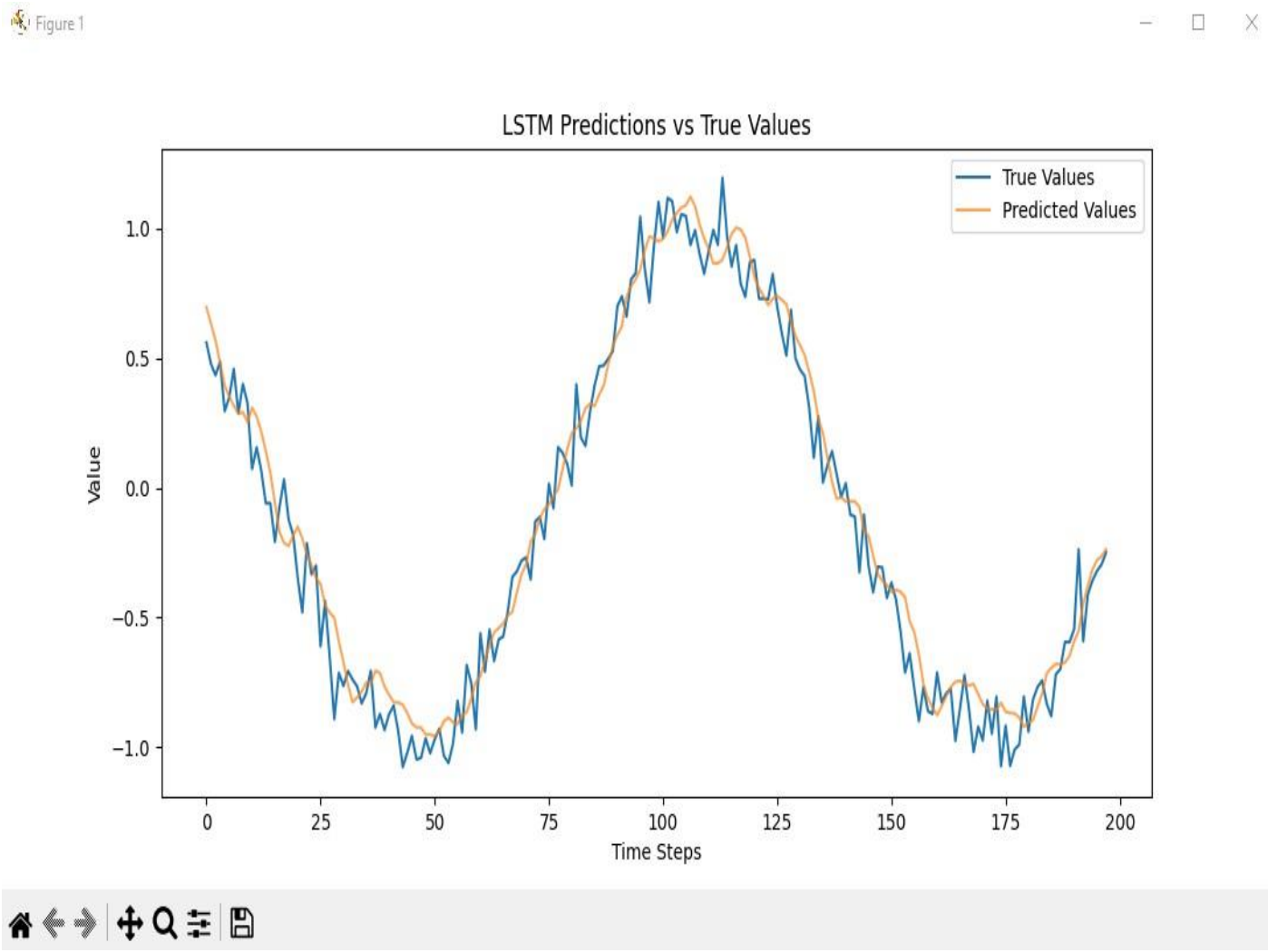
FLOW CHART:


START
IMPORT TENSORFLOW
GENERATE_DATA
PREPARE_DATA
RESHAPE X TO 3D ARRAY
EVALUATE MODEL
MAKE PREDICTION
PRINT RESULT
END

SOURCE CODE:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
def generate_data(timesteps=1000):
    x = np.linspace(0, 50, timesteps)
    y = np.sin(x) + np.random.normal(scale=0.1, size=timesteps)
    return y
def prepare_data(data, time_step=10):
    X, y = [], []
    for i in range(len(data) - time_step):
        X.append(data[i:(i + time_step)])
        y.append(data[i + time_step])
    return np.array(X), np.array(y)
timesteps = 1000
time_step = 10
batch_size = 32
epochs = 20
data = generate_data(timesteps)
X, y = prepare_data(data, time_step)
X = X.reshape((X.shape[0], X.shape[1], 1))
split = int(0.8 * len(X))
X_train, X_val = X[:split], X[split:]
y_train, y_val = y[:split], y[split:]
model = models.Sequential([
    layers.LSTM(50, return_sequences=True, input_shape=(time_step, 1)),
    layers.LSTM(50),
    layers.Dense(1)
])  model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=epochs,
                    batch_size=batch_size)
loss = model.evaluate(X_val, y_val)
print(f'Validation Loss: {loss}')
predictions = model.predict(X_val)
plt.figure(figsize=(14, 5))
plt.plot(range(len(y_val)), y_val, label='True Values')
plt.plot(range(len(predictions)), predictions, label='Predicted Values', alpha=0.7)
plt.title('LSTM Predictions vs True Values')
plt.xlabel('Time Steps')
plt.ylabel('Value')
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Thus the above program is verified and executed successfully.