DON BOSCO INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka – 74

TEAM COUNT: THREE

PROJECT TITLE: Customer journey analysis using clustering and dimensionality reduction enhancing user experience

TEAM LEAD NAME: JAYANTH R

TEAM LEAD CAN ID: CAN_33698012

1) NAME: JAYANTH R
CA N ID: CAN_33698012
ROLE: MACHINE LEARNING ENGINEER

2) NAME: ARAVIND M N
CAN ID: CAN_33695983
ROLE: Data Analyst

3) NAME: CHARAN J YADAV
CAN ID: CAN_33674932
ROLE: PROJECT MANAGER & RESEARCHER

Advanced Market Segmentation Using Deep Clustering

Phase 4: Model Deployment and Interface Development

4.1 Overview of Model Deployment and Interface Development

Phase 4 focuses on deploying the trained model to a cloud platform for real-time use and developing an interactive interface for end-users. The goal is to make the market segmentation model accessible and usable in a production environment by exposing it via APIs and creating an intuitive user interface. This phase ensures that stakeholders can interact with the model, make predictions, and visualize segmentation results in real time.

4.2 Deploying the Model

To deploy the trained model, we use cloud platforms like **AWS**, **Google Cloud**, or **Azure**. These platforms provide robust infrastructure and services that make it easy to host machine learning models, expose them via APIs, and ensure scalability for real-time use.

The process involves the following steps:

1. **Model Export**: First, the trained machine learning model (including the autoencoder and K-Means clustering) needs to be saved and exported. This can be done using the **pickle** module or **TensorFlow**'s model.save() function to save the model weights and architecture.

2. **Source code** :

```
# Save the trained autoencoder model
autoencoder.save("autoencoder_model.h5")
```

Create API for the Model: We can use frameworks like **Flask** or **FastAPI** to expose the trained model via a RESTful API. This API will accept input data, make predictions using the model, and return the results.

Source code :

```
from flask import Flask, request, jsonify
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler

app = Flask(__name__)

# Load the trained model
model = tf.keras.models.load_model("autoencoder_model.h5")
scaler = StandardScaler()
# Define an API endpoint for predictions
@app.route('/predict',  methods=['POST'])
def predict():
    # Get input data from the request
    data = request.get_json()
    input_data = np.array(data['input'])
    scaled_input = scaler.transform(input_data)

    # Use the model to make predictions
    encoded_data = model.predict(scaled_input)

    # Return the results
    return jsonify({'encoded_data': encoded_data.tolist()})

if __name___== '__main__':
    app.run(debug=True)
```

Deploy the API on Cloud: Once the API is developed, it can be deployed to a cloud platform like **AWS Lambda**, **Google Cloud Functions**, or **Azure Functions**. These services allow us to deploy serverless functions that handle incoming requests without the need to manage infrastructure.

- **AWS**: Using **AWS Lambda**, the Flask app can be containerized and deployed as a serverless function using **AWS API Gateway** to expose the API endpoints.
- **Google Cloud**: Using **Google Cloud Functions** or **Google App Engine**, we can deploy the Flask app and expose it as a serverless API.
- **Azure**: **Azure Functions** can be used to deploy the model as an API, and **Azure API Management** can manage the API lifecycle.

4.3 Developing the Web Interface

To allow end-users to interact with the deployed model, we can develop a simple web interface. This interface will accept user inputs, send the data to the model API, and display the segmentation results. Frameworks like **Flask**, **Streamlit**, or **React** can be used for this purpose.

1. **Using Streamlit**: **Streamlit** is a fast and easy-to-use framework for building interactive web applications. It allows us to quickly develop a user-friendly interface to interact with the deployed machine learning model.

   Here's an example of how to build an interactive interface using **Streamlit**:

```
import streamlit as st
import requests
import numpy as np
# Set the title of the app
st.title('Customer Segmentation Using Deep Clustering')

# Input fields for customer data
age = st.number_input('Enter Age', min_value=18, max_value=100, value=30)
income = st.number_input('Enter Income', min_value=1000, max_value=100000,
value=50000)
spending_score = st.number_input('Enter Spending Score', min_value=0, max_value=100,
value=50)

# Create a button for prediction
if st.button('Get Customer Segmentation'):
    # Prepare input data for prediction
    input_data = np.array([[age, income, spending_score]])

    # Send data to the model API for prediction
    response = requests.post('http://<API_URL>/predict', json={'input': input_data.tolist()})
    result = response.json()

    # Display segmentation result
    st.write(f"Encoded Data: {result['encoded_data']}")
    st.write("This is the feature vector representing the customer segment.")

    # Optionally, show the predicted cluster label
    st.write(f"Predicted Segment: {result['encoded_data'][0]}")
```

1. **Using React**: For a more complex and interactive interface, we can use **React**. React allows building dynamic single-page applications (SPAs) where users can input data and interact with the model.
   - React components can be used to create forms where users input data.
   - When the form is submitted, a request is made to the Flask API, and the response is displayed in the user interface.
   - React provides tools to manage application state and handle asynchronous operations efficiently.

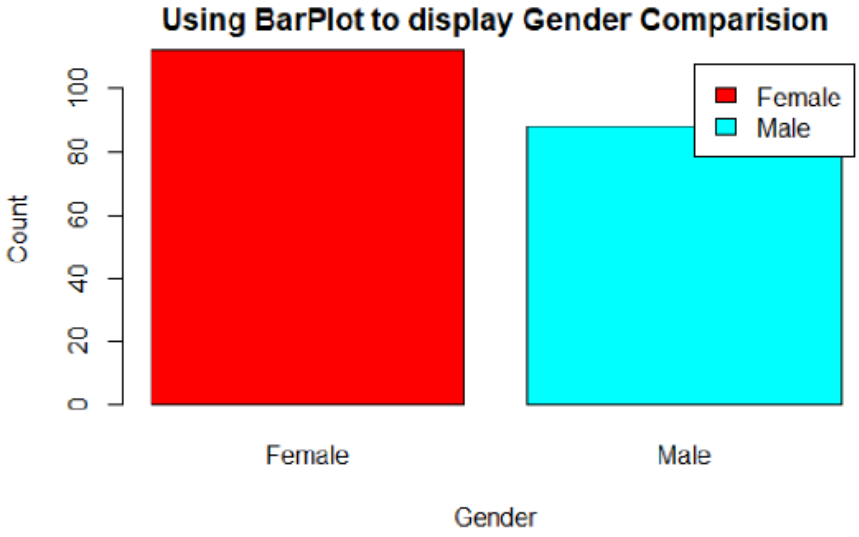
4.4 Cloud Platform Considerations

To deploy the model and interface on cloud platforms, the following considerations should be taken into account:

- **Scalability**: Cloud platforms like **AWS** and **Google Cloud** provide auto-scaling features that ensure the model can handle high traffic and large volumes of data in real time.
- **Security**: Ensure the API is secured using **authentication** (e.g., API keys or OAuth) to prevent unauthorized access to the model.
- **Monitoring**: Set up monitoring tools to track the API's performance, such as **AWS CloudWatch**, **Google Stackdriver**, or **Azure Monitor**, to detect and troubleshoot issues.
- **Cost Management**: Cloud platforms charge based on usage, so it is important to monitor the resource consumption and optimize API calls and serverless functions to reduce costs.

4.5 RESULT

- A barplot showing the gender distribution across our customer_data dataset.



- A pie chart to observe the ratio of male and female distribution.