INDIAN INSTITUTE OF TECHNOLOGY JODHPUR

NH 62 Nagaur Road, Karwar, Jodhpur, Rajasthan

Trimester 2 – Post Graduate Diploma in Data Engineering

Virtualization and Cloud Computing

Code Documentation

Submitted By:

Aneerban Chowdhury (G23AI2059)

Vanshika Gupta (G23AI2050)

Shikha Soni (G23AI2075)

Under the Guidance of

Dr. Sumit Kalra

Associate Professor, Dept. of Electrical Engg.

1. Project Overview

- **Title**: Customer Segmentation using K-Means Clustering and Flask Web App Deployed on GCP.
- **Description**: This project segments customers based on their income and expense percentage using K-Means clustering. The model is deployed via a Flask application and hosted on Google Cloud Platform (GCP) App Engine. A comparison is derived between local and cloud based on deployment.

---

2. Directory Structure

- app.yaml :GCP configuration file
- main.py : Flask web app
- model.pkl : Pre-trained K-Means model
- requirements.txt : Required dependencies
- Mall_Customers.csv : Input data for the model
- notebook.ipynb : Jupyter notebook for data processing and model training
- Index.html : HTML template for web interface

---

3. `app.yaml`

Specifies the configuration for deploying the Flask application to Google App Engine.

- `runtime: python39`: Specifies that Python 3.9 is the runtime environment.

This file ensures the correct environment setup in the cloud for running the app.

4. `main.py` (Flask App)

The main Python file for serving the web app. It loads the ML model (`model.pkl`), accepts user inputs, makes predictions, and returns results to the user.

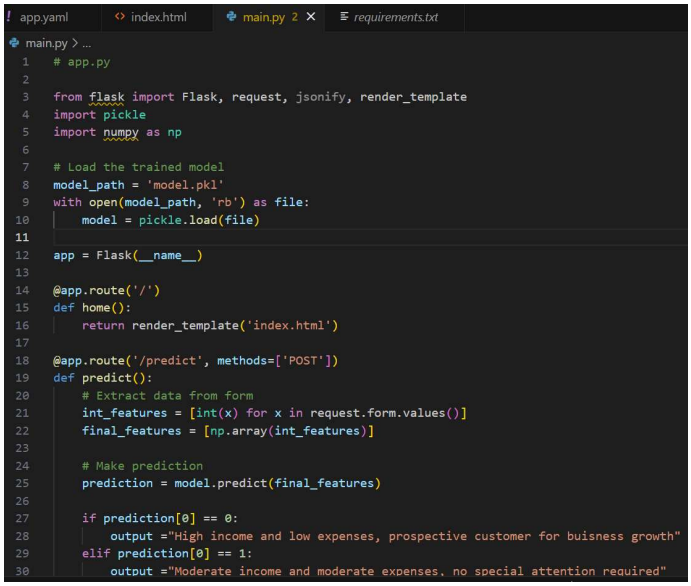
Dependencies: Flask, Numpy, Pickle, and scikit-learn.

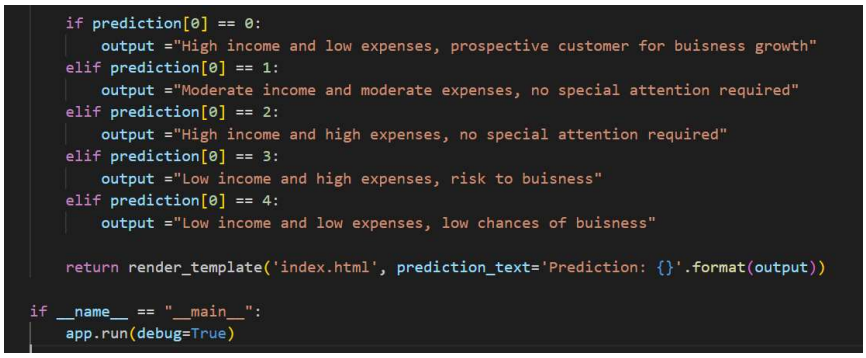
Functions:

`home()`: Renders the home page (`index.html`).

`predict()`:

- Receives form inputs from the user.
- Preprocesses inputs (converts them to an appropriate format for the model).
- Predicts the customer segmentation based on the K-Means model.
- Returns the prediction result, mapped to human-readable descriptions.





5. `notebook.ipynb`

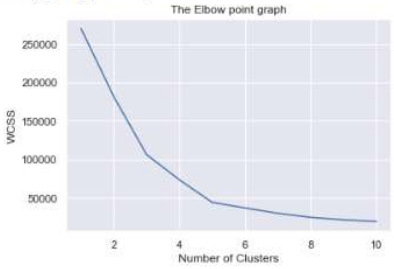
Contains code for data exploration, visualization, and model training using the K-Means clustering algorithm.

Libraries: `numpy`, `pandas`, `matplotlib`, `seaborn`, `sklearn`, `pickle`.

- Load the dataset (`Mall_Customers.csv`) using Pandas.
- Plot feature correlations using a heatmap.
- Select `Annual Income` and `Spending Score` as features for clustering.
- Use the K-Means algorithm to create clusters.
- Determine the optimal number of clusters using the elbow method (WCSS plot)

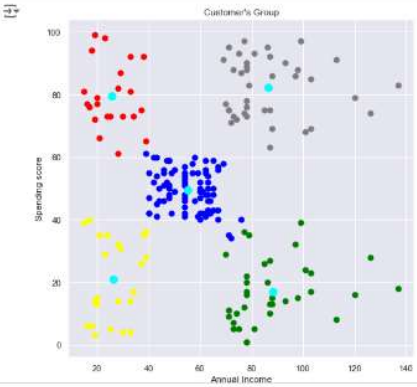
```
#Ploting elbow method

sns.set()
plt.plot(range(1,11), wcss)
plt.title('The Elbow point graph')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
```

```
Text(0, 0.5, 'WCSS')
```



Visualising the clusters

```
plt.figure(figsize=(8,8))
plt.scatter(X[Y==0,0],X[Y==0,1],s=50, c='Green', label='Cluster - 1')
plt.scatter(X[Y==1,0],X[Y==1,1],s=50, c='Blue', label='Cluster - 2')
plt.scatter(X[Y==2,0],X[Y==2,1],s=50, c='Gray', label='Cluster - 3')
plt.scatter(X[Y==3,0],X[Y==3,1],s=50, c='Red', label='Cluster - 4')
plt.scatter(X[Y==4,0],X[Y==4,1],s=50, c='Yellow', label='Cluster - 5')

# ploatting the centroid
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=100, c='cyan', label='centroids')

plt.title("Customer's Group")
plt.xlabel('Annual Income')
plt.ylabel('Spending score')

plt.show()
```



6. `model.pkl`

Stores the trained K-Means model, serialized using the `pickle` module. Ensure that the environment where the model is used has compatible library versions (especially scikit-learn).

Creating model for deployment

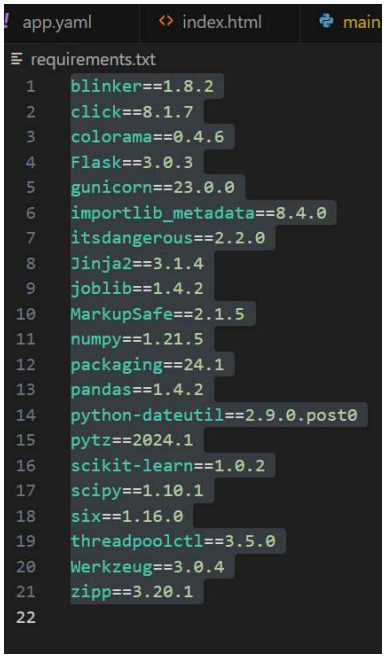
```
[ ]  with open('model.pkl','wb') as file:
         pickle.dump(kmeans,file)
```

---

7. `Requirements.txt`

Lists all the dependencies required to run the project.

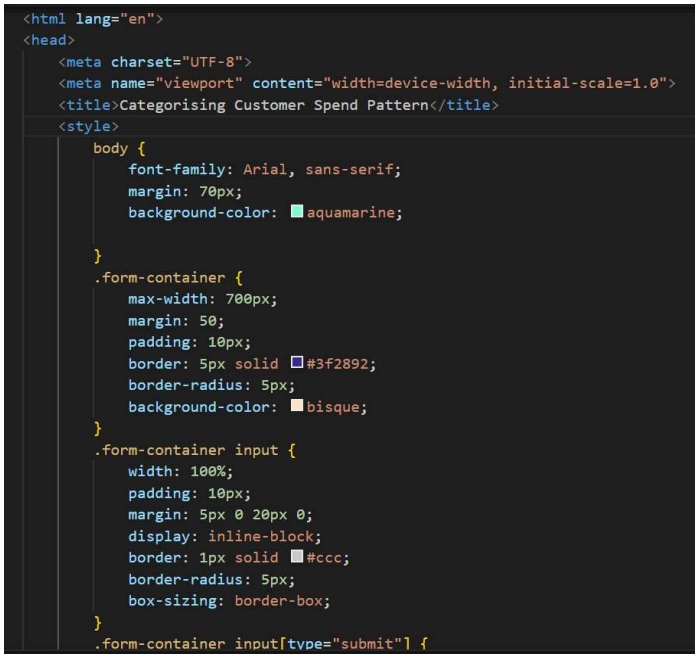
- `Flask==3.0.3`: Web framework.
- `gunicorn==23.0.0`: WSGI HTTP server for deploying the Flask app.
- `scikit-learn==1.0.2`: Machine learning library used for K-Means clustering.
- `numpy`, `pandas`, `matplotlib`, `seaborn`: For data manipulation and visualization.



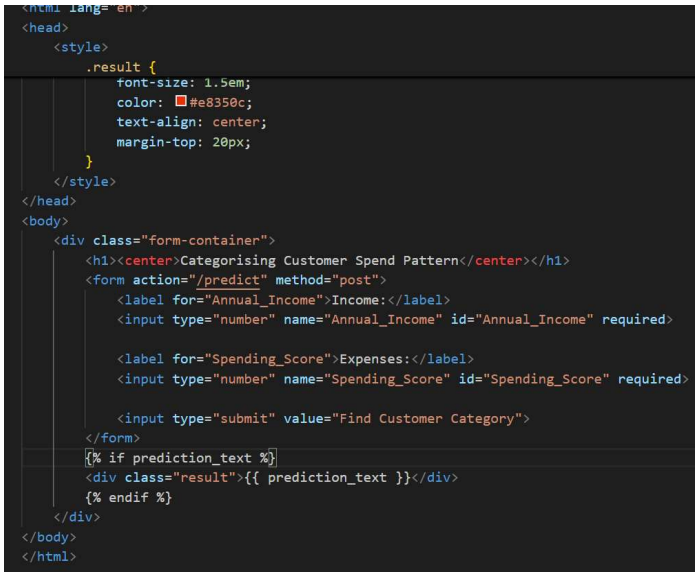
8. `index.html`

The front-end user interface for the web application. This HTML file is served by Flask and contains the input form for users to provide data (e.g., Annual Income, Spending Score). It also displays the prediction result.

- A form that sends user input via a POST request to the `/predict` route.
- A section to display the model's prediction.



```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Categorising Customer Spend Pattern</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 70px;
            background-color: aquamarine;

        }
        .form-container {
            max-width: 700px;
            margin: 50;
            padding: 10px;
            border: 5px solid #3f2892;
            border-radius: 5px;
            background-color: bisque;
        }
        .form-container input {
            width: 100%;
            padding: 10px;
            margin: 5px 0 20px 0;
            display: inline-block;
            border: 1px solid #ccc;
            border-radius: 5px;
            box-sizing: border-box;
        }
        .form-container input[type="submit"] {
```

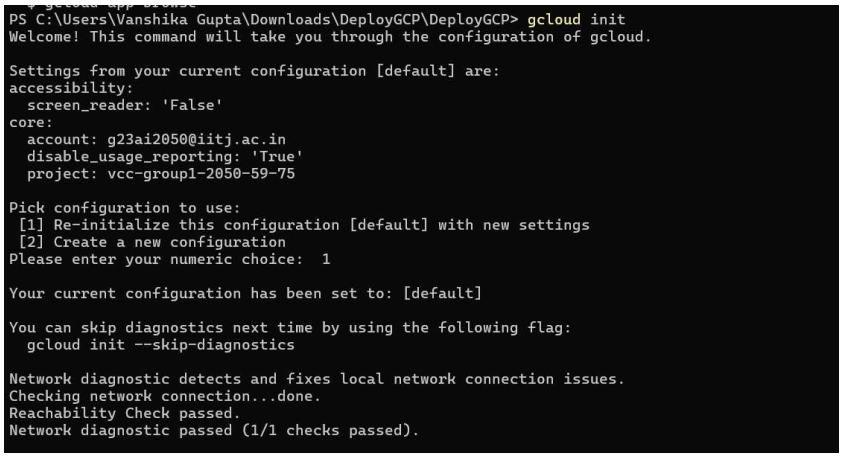


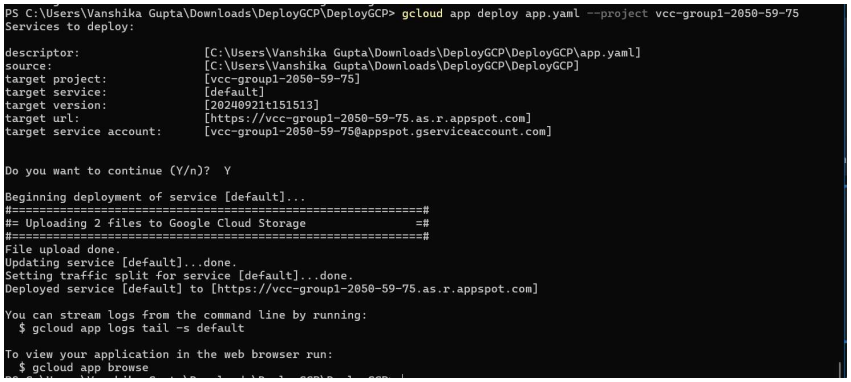
```
<html lang="en">
<head>
    <style>
        .result {
            font-size: 1.5em;
            color: #e8350c;
            text-align: center;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="form-container">
        <h1><center>Categorising Customer Spend Pattern</center></h1>
        <form action="/predict" method="post">
            <label for="Annual_Income">Income:</label>
            <input type="number" name="Annual_Income" id="Annual_Income" required>

            <label for="Spending_Score">Expenses:</label>
            <input type="number" name="Spending_Score" id="Spending_Score" required>

            <input type="submit" value="Find Customer Category">
        </form>
        {% if prediction_text %}
        <div class="result">{{ prediction_text }}</div>
        {% endif %}
    </div>
</body>
</html>
```

9. GCP Deployment

- **Steps**:

1. **Install Google Cloud SDK:** https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe
2. **Configure App Engine**: `app.yaml` ensures Python 3.9 runtime.
3. **Initialize:** Use the command **gcloud init** to initialize the enviroment
4. **Deploy**: Use the command **`gcloud app deploy`** to deploy the app to GCP.
5. **Access**: The app is accessible via the GCP App Engine URL after deployment.





---

10. Local vs GCP Deployment

- **Local Deployment**:
  - Run the app locally using `Flask`.
  - Suitable for testing and development.
- **GCP Deployment**:
  - Deploy on GCP App Engine for scalability and high availability.
  - GCP handles scaling based on user traffic.