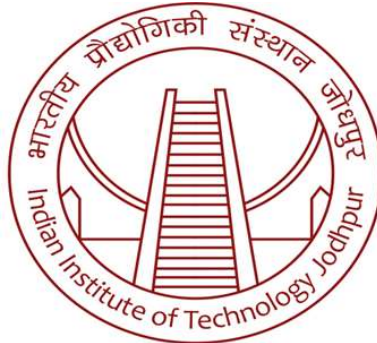


# **INDIAN INSTITUTE OF TECHNOLOGY JODHPUR**

**NH 62 Nagaur Road, Karwar, Jodhpur, Rajasthan**



॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

## **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

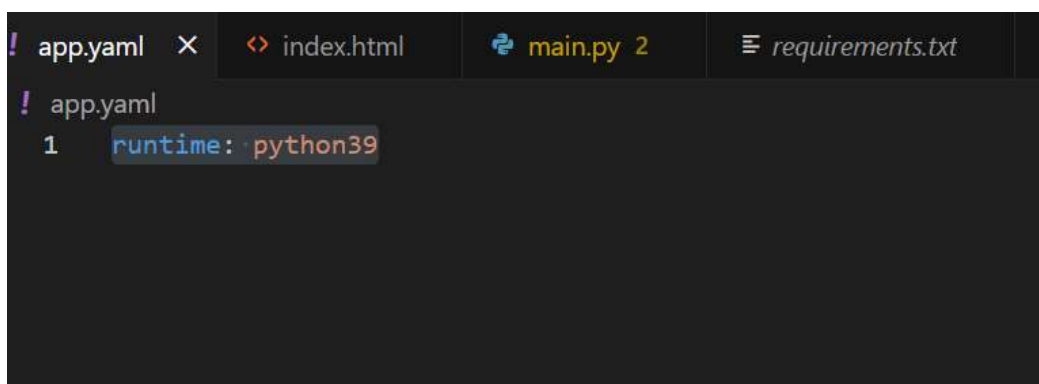
- `app.yaml` : GCP configuration file
  - `main.py` : Flask web app
  - `model.pkl` : Pre-trained K-Means model
  - `requirements.txt` : Required dependencies
  - `Mall_Customers.csv` : Input data for the model
  - `notebook.ipynb` : Jupyter notebook for data processing and model training
  - `Index.html`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



```
! app.yaml  X  <> index.html  main.py 2  requirements.txt
! app.yaml
1  runtime: python39
```

## 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.

```
! app.yaml  index.html  main.py 2 X  requirements.txt
main.py > ...
1  # app.py
2
3  from flask import Flask, request, jsonify, render_template
4  import pickle
5  import numpy as np
6
7  # Load the trained model
8  model_path = 'model.pkl'
9  with open(model_path, 'rb') as file:
10     model = pickle.load(file)
11
12  app = Flask(__name__)
13
14  @app.route('/')
15  def home():
16     return render_template('index.html')
17
18  @app.route('/predict', methods=['POST'])
19  def predict():
20     # Extract data from form
21     int_features = [int(x) for x in request.form.values()]
22     final_features = [np.array(int_features)]
23
24     # Make prediction
25     prediction = model.predict(final_features)
26
27     if prediction[0] == 0:
28         output = "High income and low expenses, prospective customer for business growth"
29     elif prediction[0] == 1:
30         output = "Moderate income and moderate expenses, no special attention required"
```

```

31     if prediction[0] == 0:
32         output = "High income and low expenses, prospective customer for business growth"
33     elif prediction[0] == 1:
34         output = "Moderate income and moderate expenses, no special attention required"
35     elif prediction[0] == 2:
36         output = "High income and high expenses, no special attention required"
37     elif prediction[0] == 3:
38         output = "Low income and high expenses, risk to business"
39     elif prediction[0] == 4:
40         output = "Low income and low expenses, low chances of business"
41
42     return render_template('index.html', prediction_text='Prediction: {}'.format(output))
43
44 if __name__ == "__main__":
45     app.run(debug=True)
```

## 5. notebook.ipynb

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

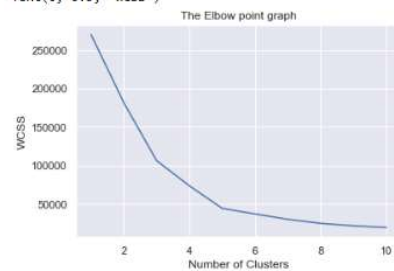
**Libraries:** numpy, pandas, matplotlib, seaborn, sklearn, pickle.

- 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)

```
#Plotting 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')
```



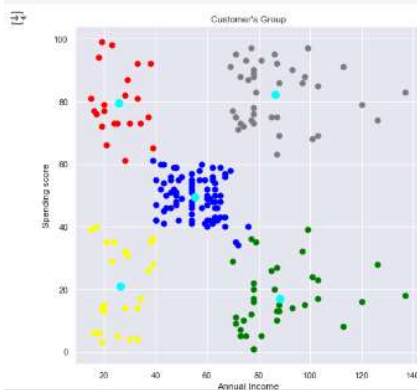
✓ Visualising the clusters

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

# plotting 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

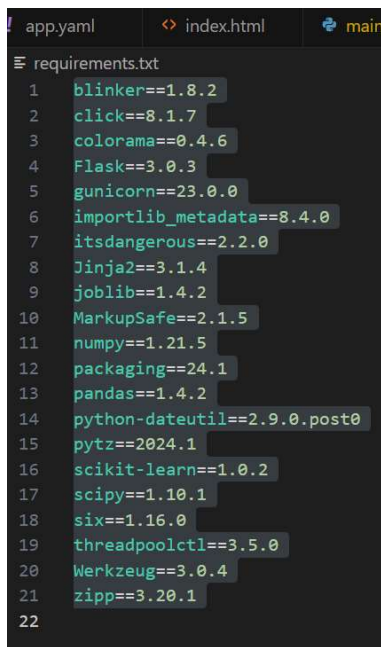
```
[ ] with open('model.pkl','wb') as file:
    pickle.dump(kmeans,file)
```

---

## 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.

A screenshot of a code editor with a dark theme. The editor shows a file named 'requirements.txt' with 22 lines of text. Each line contains a package name followed by an equals sign and a version number. The text is as follows:

```
1 blinker==1.8.2
2 click==8.1.7
3 colorama==0.4.6
4 Flask==3.0.3
5 gunicorn==23.0.0
6 importlib_metadata==8.4.0
7 itsdangerous==2.2.0
8 Jinja2==3.1.4
9 joblib==1.4.2
10 MarkupSafe==2.1.5
11 numpy==1.21.5
12 packaging==24.1
13 pandas==1.4.2
14 python-dateutil==2.9.0.post0
15 pytz==2024.1
16 scikit-learn==1.0.2
17 scipy==1.10.1
18 six==1.16.0
19 threadpoolctl==3.5.0
20 Werkzeug==3.0.4
21 zipp==3.20.1
22
```

## 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: #a9d0d9;
    }
    .form-container {
      max-width: 700px;
      margin: 50px;
      padding: 10px;
      border: 5px solid #3f2892;
      border-radius: 5px;
      background-color: #f5f5dc;
    }
    .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 environment
  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.

```
PS C:\Users\Vanshika Gupta\Downloads\DeployGCP\DeployGCP> gcloud init
Welcome! This command will take you through the configuration of gcloud.

Settings from your current configuration [default] are:
accessibility:
  screen_reader: 'False'
core:
  account: g23ai2050@iitj.ac.in
  disable_usage_reporting: 'True'
  project: vcc-group1-2050-59-75

Pick configuration to use:
[1] Re-initialize this configuration [default] with new settings
[2] Create a new configuration
Please enter your numeric choice: 1

Your current configuration has been set to: [default]

You can skip diagnostics next time by using the following flag:
  gcloud init --skip-diagnostics

Network diagnostic detects and fixes local network connection issues.
Checking network connection...done.
Reachability Check passed.
Network diagnostic passed (1/1 checks passed).
```

```
PS C:\Users\Vanshika Gupta\Downloads\DeployGCP\DeployGCP> gcloud app deploy app.yaml --project vcc-group1-2050-59-75
Services to deploy:

descriptor:      [C:\Users\Vanshika Gupta\Downloads\DeployGCP\DeployGCP\app.yaml]
source:          [C:\Users\Vanshika Gupta\Downloads\DeployGCP\DeployGCP]
target project:  [vcc-group1-2050-59-75]
target service:  [default]
target version:  [20240921151513]
target url:      [https://vcc-group1-2050-59-75.as.r.appspot.com]
target service account: [vcc-group1-2050-59-75@appspot.gserviceaccount.com]

Do you want to continue (Y/n)? Y

Beginning deployment of service [default]...
=====
# Uploading 2 files to Google Cloud Storage
=====
File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://vcc-group1-2050-59-75.as.r.appspot.com]

You can stream logs from the command line by running:
$ gcloud app logs tail -s default

To view your application in the web browser run:
$ gcloud app browse
```

---

## 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.