**APPENDICES I**

**SAMPLE CODING**

**MACHINE LEARNING CODE:**

**Importing the Dependencies:**

import numpy as np # type: ignore import pandas as pd # type: ignore

import matplotlib.pyplot as plt # type: ignore import seaborn as sns # type: ignore

from sklearn.cluster import KMeans # type: ignore

from sklearn.preprocessing import LabelEncoder # type: ignore

**Data Collection & Analysis:**

**# loading the data from csv file to a Pandas DataFrame**

customer\_data = pd.read\_csv('customers.csv')

**# Missing Value Handling** customer\_data.fillna(customer\_data.mean(), inplace=True) print("Missing values after handling:") print(customer\_data.isnull().sum())

X = customer\_data[['Age', 'Average Amount Spent', 'Spending Score']].values print("Any NaN values in X:", np.isnan(X).any())

**# first 5 rows in the dataframe**

customer\_data.tail()

**# finding the number of rows and columns**

customer\_data.shape

**# getting some informations about the dataset**

Customer data.info()

**# checking for missing values** customer\_data.isnull().sum() print(X)

**Choosing the number of clusters:**

customer\_data.fillna(customer\_data.mean(), inplace=True)

**WCSS -> Within Clusters Sum of Squares:**

X = customer\_data[['Age', 'Average Amount Spent', 'Spending Score']].values

**# finding wcss value for different number of clusters**

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42) kmeans.fit(X)

wcss.append(kmeans.inertia\_)

**# plot an elbow graph**

sns.set()

plt.plot(range(1, 11), wcss) plt.title('The Elbow Point Graph') plt.xlabel('Number of Clusters') plt.ylabel('WCSS')

plt.show() **Optimum Number of Clusters = 5 Training the k-Means Clustering Model**

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state=42) Y = kmeans.fit\_predict(X)

**5 Clusters - 0, 1, 2, 3, 4 Visualizing all the Clusters:**

**# plotting all the clusters and their Centroids**

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='red', label='Cluster 2')

plt.scatter(X[Y == 2, 0], X[Y == 2, 1], s=50, c='yellow', label='Cluster 3')

plt.scatter(X[Y == 3, 0], X[Y == 3, 1], s=50, c='violet', label='Cluster 4')

plt.scatter(X[Y == 4, 0], X[Y == 4, 1], s=50, c='blue', label='Cluster 5')

**# Plot the centroids**

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=100, c='cyan', label='Centroids')

plt.title('CustomerGroups') plt.xlabel('Age') plt.ylabel('Spending Score') plt.legend()

plt.show()

**HTML CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Customer Segmentation Form</title>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font- awesome/5.15.4/css/all.min.css">

<style> body {

font-family: 'Arial', sans-serif;

background: linear-gradient(135deg, #f5f7fa, #c3cfe2); color: #333;

margin: 0;

padding: 0; display:

flex;

align-items: center; justify-content: center; height: 100vh;

}

.container { background: white; padding: 30px; border-radius: 10px;

box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);

width: 90%;

max-width: 500px; text-align: center;

transition: transform 0.3s, box-shadow 0.3s;

}

.container:hover { transform: translateY(-5px);

box-shadow: 0 12px 24px rgba(0, 0, 0, 0.3);

}

h1 {

color: #007bff; margin- bottom: 20px;

}

label {

display: block; margin: 15px 0 5px; font- weight: bold;

}

input[type="number"], select {

width: 100%; padding: 12px; margin-bottom: 15px;

border: 1px solid #007bff; border-radius: 5px; transition: border-color 0.3s;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

}

input[type="number"]:focus, select:focus {

border-color: #0056b3; outline: none;

}

input[type="submit"] { background-color: #007bff; color: white;

border: none; padding: 15px; cursor: pointer; border-radius: 5px;

transition: background-color 0.3s ease, transform 0.2s; width: 100%;

font-size: 16px;

}

input[type="submit"]:hover { background-color: #0056b3; transform: translateY(-2px);

}

.footer {

margin-top: 20px; font-size: 14px; color: #666;

}

.error {

color: red; margin-top: 10px;

}

.input-container { position: relative; margin-bottom: 20px;

}

.input-container i { position: absolute; top: 12px;

left: 10px; color:

#007bff;

}

.success-message { background-color:#dff0d8; color: #3c763d;

padding: 10px; border-radius: 5px; margin-top: 15px;

display: none; /\* Hidden by default \*/

}

</style>

<script>

function validateForm() {

const age = document.getElementById("age").value; if (age > 100) {

alert("Age must be less than or equal to 100."); return false;

}

return true;

}

</script>

</head>

<body>

<div class="container">

<h1><i class="fas fa-users"></i> Customer Segmentation Form</h1>

<form action="/submit" method="post" onsubmit="return validateForm()">

<div class="input-container">

<i class="fas fa-calendar-alt"></i>

<label for="age">Age (must be ≤ 100):</label>

<input type="number" id="age" name="age" required min="0" max="100" placeholder="Enter your age">

</div>

<div class="input-container">

<i class="fas fa-shopping-cart"></i>

<label for="purchase\_frequency">Purchase Frequency (times per month):</label>

<input type="number" id="purchase\_frequency" name="purchase\_frequency" required min="1" placeholder="Enter frequency">

</div>

<div class="input-container">

<i class="fas fa-money-bill-wave"></i>

<label for="average\_spent">Average Amount Spent (in INR):</label>

<input type="number" id="average\_spent" name="average\_spent" required min="0" placeholder="Enter amount in INR">

</div>

<input type="submit" value="Submit">

</form>

<div class="success-message" id="successMessage"> Data Submitted Successfully! Thank you.

</div>

<div class="footer">

<p>&copy; 2024 Customer Segmentation Project</p>

</div>

</div>

<script>

const form = document.querySelector('form');

const successMessage = document.getElementById('successMessage');

form.addEventListener('submit', function(e) {

// Perform form validation if (validateForm()) {

// Let the form be submitted to the server successMessage.style.display = 'block'; setTimeout(() => {

successMessage.style.display = 'none'; // Hide message after 3 seconds

}, 3000);

} else {

e.preventDefault(); // Prevent form submission only if validation fails

}

});

</script>

</body>

</html>

<!—This is used as a dummy to clone further dialogs

<div id=”modalsGarbage”>

<div class=”modal fade animate animated” id=”dummy-dialog-modal”

tabindex=”-1” role=”dialog” aria-labelledby=”” Aria-hidden=”true”>

<div class=”modal-dialog modal-dialog-centered modal-dialog-scrollable”>

<div class=”modal-content blur” style=”box-shadow: rgba(3, 102, 214, 0.3) 0px 0px 0px 3px”>

<div class=”modal-header”>

<h4 class=”modal-title”></h4>

</div>

<div class=”modal-body”> </div>

<div class=”modal-footer”>

</div>

</div>

</div>

</div>

</div>

<script

Src=”<?=get\_config(‘base\_path’)?>assets/dist/js/bootstrap.bundle.min.js”>

</script>

<script src=https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js></script>

<script src=https://cdn.jsdelivr.net/npm/masonrylayout@4.2.2/dist/masonry.pkgd.min.js

Integrity=”sha384- GNFwBvfVxBkLMJpYMOABq3c+d3KnQxudP/mGPkzpZSTYykLBNsZEnG2D9 G

/X/+7D” crossorigin=”anonymous”>

</script>

<script src=https://unpkg.com/imagesloaded@5/imagesloaded.pkgd.min.js></script>

<script src=”/js/app.min.js”></script>

<script src=”/js/dialog.js”></script>

<script src=”/js/toast.js”></script>

<script>

// Initialize the agent at application startup.

Const fpPromise = import(‘https://openfpcdn.io/fingerprintjs/v3’)

.then(FingerprintJS => FingerprintJS.load()) // Get the visitor identifier when you need it. fpPromise

.then(fp => fp.get())

.then(result => {

// This is the visitor identifier: Const visitorId = result.visitorId Console.log(visitorId)

$(‘#fingerprint’).val(visitorId);

})

</script>

</body>

</html>

**CSS FILES:**

#login.css

.bd-placeholder-img { Font-size: 1.125rem; Text-anchor: middle;

-webkit-user-select: none;

-moz-user-select: none; User-select: none;

}

.form-signin { Width: 100%; Max-width: 330px; Padding: 15px; Margin: auto;

}

.form-signin .checkbox { Font-weight: 400;

}

.form-signin .form-floating:focus-within { z-index: 2;}

.form-signin input[type=”email”] { Margin-bottom: -1px;

Border-bottom-right-radius: 0;

Border-bottom-left-radius: 0;

}

.form-signin input[type=”password”] { Margin-bottom: 10px;

Border-top-left-radius: 0;

Border-top-right-radius: 0;

}

#signup.css

.bd-placeholder-img { Font-size: 1.125rem; Text-anchor: middle;

-webkit-user-select: none;

-moz-user-select: none; User-select: non

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=devi ce-width, initial- scale=1.0">

<title>Segmentation Result</title>

</head>

<body>

<h1>Customer Segmentation Result</h1>

<img src="{{ plot\_url }}" alt="Segmentation Plot" style="max- width:100%; height:auto;">

<br>

<a href="/">Go Back</a>

</body>

</html>

.form-signup { Width: 100%; Max-width: 330px;Padding: 15px; Margin: auto;

}

.form-signup .checkbox { Font-weight: 400;

}

.form-signup .form-floating:focus-within { z-index: 2;

}

.form-signup input[type=”email”] { Margin-bottom: -1px; Border-bottom-right-radius: 0;

Border-bottom-left-radius: 0;

Border-top-left-radius: 0;

Border-top-right-radius: 0;

}

.form-signup input[type=”password”] { Margin-bottom: 10px;

Border-top-left-radius: 0;

Border-top-right-radius: 0;

}

.form-signup input[name=”phone”] {

Margin-bottom: -1px; Border-bottom-right-radius: 0; Border-bottom-left-radius: 0;

Border-top-left-radius: 0;

Border-top-right-radius: 0;

}

.form-signup input[name=”username”] { Margin-bottom: -1px;

Border-bottom-right-radius: 0;

Border-bottom-left-radius: 0;

}

**JAVA SCRIPT**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Customer Segmentation with K-Means</title>

<!-- Include ml5.js for machine learning -->

<script src="https://cdn.jsdelivr.net/npm/ml5@2.3.1/dist/ml5.min.js"></script>

<!-- Include Chart.js for data visualization -->

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<style>

body {

font-family: Arial, sans-serif;

}

canvas {

max-width: 100%;

}

</style>

</head>

<body>

<h1>Customer Segmentation Using K-Means Clustering</h1>

<canvas id="segmentationChart"></canvas>

<script>

// Sample customer data: [Age, Spending, Frequency] const customers = [

[25, 200, 10], // Customer 1

[45, 600, 15], // Customer 2

[35, 400, 12], // Customer 3

[50, 900, 20], // Customer 4

[28, 250, 8], // Customer 5

[55, 800, 25], // Customer 6

[33, 500, 18], // Customer 7

[40, 700, 10], // Customer 8

[30, 450, 13], // Customer 9

[60, 1100, 30] // Customer 10

];

// Number of segments (clusters) we want to create const numClusters = 3;

let kmeans;

// Initialize K-means model with ml5.js function setup() {

// Initialize the K-means model with the specified number of clusters and the customer data kmeans = ml5.kmeans(numClusters, customers, modelReady);

}

// Callback when the model is ready function modelReady() {

// Perform clustering and handle the result kmeans.cluster(customers, (err, result) => {

if (err) {

console.error('Error in clustering:', err); return;

}

// Visualize the segmentation result visualizeSegmentation(result);

});

}

// Function to visualize the segmentation result using Chart.js function visualizeSegmentation(result) {

// Prepare data for visualization

const segmentColors = ['#FF5733', '#33FF57', '#3357FF']; // Colors for different segments let datasets = [];

result.forEach((segment, index) => { let segmentData = {

label: `Segment ${index + 1}`, data: segment.map(customer => ({

x: customer[0], // Age

y: customer[1], // Spending r: customer[2] // Frequency

})),

backgroundColor: segmentColors[index], borderColor: segmentColors[index], borderWidth: 1

};

datasets.push(segmentData);

});

// Create a bubble chart using Chart.js

const ctx = document.getElementById('segmentationChart').getContext('2d'); new Chart(ctx, {

type: 'bubble', data: {

datasets: datasets

},

options: { responsive: true, scales: {

x: {

title: {

display: true, text: 'Age'

}

},

y: {

title: {

display: true, text: 'Spending'

}

},

r: {

title: {

display: true, text: 'Frequency'

}

}

},

plugins: { legend: {

display: true, position: 'top'

}

}

});

}

// Call setup to initialize the K-means algorithm setup();

</script>

</body>

</html>

**DATA PROCESSING CODE:**

import pandas as pd import numpy as np

from sklearn.preprocessing import StandardScaler from sklearn.impute import SimpleImputer

from sklearn.cluster import KMeans

from flask import Flask, jsonify, request, render\_template app = Flask( name )

# Sample customer data with missing values

customer\_data = [

[25, 200, 10], # Customer 1 (Complete)

[45, 600, 15], # Customer 2 (Complete)

[35, np.nan, 12], # Customer 3 (Missing Spending)

[50, 900, 20], # Customer 4 (Complete)

[28, 250, 8], # Customer 5 (Complete)

[np.nan, 800, 25], # Customer 6 (Missing Age)

[33, 500, 18], # Customer 7 (Complete)

[40, np.nan, 10], # Customer 8 (Missing Spending)

[30, 450, 13], # Customer 9 (Complete)

[60, 1100, 30] # Customer 10 (Complete)

]

# Create a DataFrame from the customer data

df = pd.DataFrame(customer\_data, columns=["Age", "Spending", "Frequency"])

# Data Processing: Handling missing values and scaling the data # 1. Handle missing values (Imputation)

imputer = SimpleImputer(strategy='mean') # Impute with the mean value df\_imputed = pd.DataFrame(imputer.fit\_transform(df), columns=df.columns)

# 2. Normalize/scale the data (Standardization)

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df\_imputed), columns=df.columns)

# Print the processed data

print("Processed Data (Imputed and Scaled):") print(df\_scaled)

# Function to perform K-means clustering

def perform\_clustering(data, num\_clusters=3): kmeans = KMeans(n\_clusters=num\_clusters) data['Segment'] = kmeans.fit\_predict(data) return data, kmeans

@app.route('/') def index():

return render\_template('index.html')

@app.route('/segment', methods=['POST']) def segment():

# Get customer data from the request (assuming JSON format)

input\_data = request.get\_json()

# Convert input data to a DataFrame

input\_df = pd.DataFrame(input\_data, columns=["Age", "Spending", "Frequency"])

# Process the input data (Impute and Scale)

input\_imputed = pd.DataFrame(imputer.transform(input\_df), columns=input\_df.columns) input\_scaled = pd.DataFrame(scaler.transform(input\_imputed),

columns=input\_df.columns)

# Perform K-means clustering

segmented\_data, kmeans\_model = perform\_clustering(input\_scaled)

# Convert the result to a JSON-friendly format result = segmented\_data.to\_dict(orient='records') return jsonify(result)

if name == ' main ': app.run(debug=True)

imputer = SimpleImputer(strategy='mean') # Impute missing data with the mean

df\_imputed = pd.DataFrame(imputer.fit\_transform(df), columns=df.columns)

def perform\_clustering(data, num\_clusters=3): kmeans = KMeans(n\_clusters=num\_clusters) data['Segment'] = kmeans.fit\_predict(data) return data, kmeans

@app.route('/segment', methods=['POST']) def segment():

input\_data = request.get\_json() # Get customer data as JSON

input\_df = pd.DataFrame(input\_data, columns=["Age", "Spending", "Frequency"])

# Process input data (Impute missing values and Scale)

input\_imputed = pd.DataFrame(imputer.transform(input\_df), columns=input\_df.columns) input\_scaled = pd.DataFrame(scaler.transform(input\_imputed),

columns=input\_df.columns)

# Perform clustering and return the result

segmented\_data, kmeans\_model = perform\_clustering(input\_scaled) result = segmented\_data.to\_dict(orient='records')

return jsonify(result)

pip install flask scikit-learn pandas numpy

**VISUALIZATION CODE:**

import matplotlib.pyplot as plt import seaborn as sns

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Sample customer data

customers = [

[25, 200, 10], # Customer 1

[45, 600, 15], # Customer 2

[35, 400, 12], # Customer 3

[50, 900, 20], # Customer 4

[28, 250, 8], # Customer 5

[55, 800, 25], # Customer 6

[33, 500, 18], # Customer 7

[40, 700, 10], # Customer 8

[30, 450, 13], # Customer 9

[60, 1100, 30] # Customer 10

]

# Convert the customer data into a DataFrame

df = pd.DataFrame(customers, columns=["Age", "Spending", "Frequency"])

# Data Preprocessing: Scaling the data

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df[['Age', 'Spending']]), columns=["Age", "Spending"])

# Apply K-means clustering

kmeans = KMeans(n\_clusters=3) df\_scaled['Segment'] = kmeans.fit\_predict(df\_scaled)

# Plotting the 2D scatter plot

plt.figure(figsize=(8, 6))

sns.scatterplot(data=df\_scaled, x="Age", y="Spending", hue="Segment", palette="Set1", s=100)

plt.title("Customer Segmentation (K-means)", fontsize=16) plt.xlabel("Age")

plt.ylabel("Spending ($)") plt.legend(title="Segment", loc='upper right') plt.show()