```
!pip install dash
!pip install dash pandas plotly

import pandas as pd
import random
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")
import dash
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd
import plotly.express as px

# Function to generate random gender
def generate_random_gender():
    return random.choice(['Male', 'Female'])

# Function to generate random marital status
def generate_random_marital_status():
    return random.choice(['Married', 'Single', 'Divorced', 'Widowed'])

# Function to generate random annual income between 30000 and 120000 USD
def generate_random_annual_income():
    return random.randint(15000, 10000000)

# Function to generate random total purchases between 5 and 50
def generate_random_total_purchases():
    return random.randint(1, 100)

# Function to generate random preferred category
def generate_random_preferred_category():
    return random.choice(['Electronics', 'Appliances', 'Fashion', 'Books'])

# Generate data for the remaining 495 customers
num_remaining_customers = 495
remaining_data = []
for customer_id in range(1006, 1006 + num_remaining_customers):
    age = random.randint(18, 65)  # Random age between 18 and 65
    gender = generate_random_gender()
```

```
    marital_status = generate_random_marital_status()
    annual_income = generate_random_annual_income()
    total_purchases = generate_random_total_purchases()
    preferred_category = generate_random_preferred_category()

    remaining_data.append([customer_id, age, gender, marital_status, annual_income,
total_purchases, preferred_category])

# Combine the data for all customers (original 5 + remaining 495)
all_customers_data = [
    [1001, 33, 'Male', 'Married', 65000, 18, 'Electronics'],
    [1002, 28, 'Female', 'Single', 45000, 15, 'Appliances'],
    [1003, 42, 'Male', 'Single', 55000, 20, 'Electronics'],
    [1004, 51, 'Female', 'Married', 80000, 12, 'Electronics'],
    [1005, 37, 'Male', 'Divorced', 58000, 10, 'Appliances'],
] + remaining_data

# Convert the data to a Pandas DataFrame
columns = ['CustomerID', 'Age', 'Gender', 'MaritalStatus', 'AnnualIncome (USD)',
'TotalPurchases', 'PreferredCategory']
df = pd.DataFrame(all_customers_data, columns=columns)

# Save the dataset to a CSV file
df.to_csv('TechElectro_Customer_Data.csv', index=False)

data = pd.read_csv("/content/TechElectro_Customer_Data.csv")

data.info()

data.isna().sum()

#check for duplicates
duplicates = data[data.duplicated(keep=False)]

if not duplicates.empty:
    print("Duplicate rows found:")
    print(duplicates)
else:
    print("No duplicates found.")

# Select the columns to scale (e.g., Age, AnnualIncome, TotalPurchases)
columns_to_scale = ['Age', 'AnnualIncome (USD)', 'TotalPurchases']

# Initialize the MinMaxScaler
```

```
scaler = MinMaxScaler()

# Apply Min-Max Scaling to selected columns
data[columns_to_scale] = scaler.fit_transform(data[columns_to_scale])

#visualise distribution of Age by Gender
plt.figure(figsize=(10, 6))
sns.boxplot(x='Gender', y='Age', data=data)
plt.title('Distribution of Age by Gender')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.show()

#visualise anuualincome
plt.figure(figsize=(10, 6))
plt.hist(data['AnnualIncome (USD)'], edgecolor='black')
plt.title('Distribution of AnnualIncome (USD)')
plt.xlabel('AnnualIncome (USD)')
plt.ylabel('Count')
plt.show()

#visualising gender distribution
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', data=df, palette='Set2')
plt.title('Gender Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()

#visualie preffered category
plt.figure(figsize=(10, 6))
sns.countplot(x='PreferredCategory', data=df, palette='pastel')
plt.title('Preferred Category Distribution')
plt.xlabel('Preferred Category')
plt.ylabel('Count')
plt.show()

# Correlation heatmap for numerical variables
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()

# Select features for clustering
```

```
features_for_clustering = ['Age', 'AnnualIncome (USD)', 'TotalPurchases']

# Standardize the features for better clustering performance
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data[features_for_clustering])

# Determine the optimal number of clusters using the Elbow Method
inertia_values = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    inertia_values.append(kmeans.inertia_)

# Plot the Elbow Method to determine the optimal number of clusters
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia_values, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (Within-Cluster Sum of Squares)')
plt.show()

# Based on the Elbow Method, let's choose K=3
k = 6

# Apply K-means clustering with K=3
kmeans = KMeans(n_clusters=k, random_state=42)
df['Cluster'] = kmeans.fit_predict(data_scaled)

# Visualize the clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(x='TotalPurchases', y='AnnualIncome (USD)', hue='Cluster', data=df,
palette='Set1', s=100)
plt.title('Customer Segmentation using K-means Clustering')
plt.xlabel('Total Purchases')
plt.ylabel('Annual Income (USD)')
plt.show()

# Create a Dash app
app = dash.Dash("TechElectro")

# Define the layout of the dashboard
app.layout = html.Div([
    html.H1('Customer Segmentation Dashboard'),
```

```
    # Dropdown to select customer segments based on PreferredCategory
    dcc.Dropdown(
        id='customer-segment-dropdown',
        options=[
            {'label': category, 'value': category} for category in data['PreferredCategory'].unique()
        ],
        value=data['PreferredCategory'].unique()[0],  # Default selected segment
    ),

    # Bar chart for PreferredCategory vs. TotalPurchases
    dcc.Graph(id='bar-preferredcategory-purchases'),

    # Bar chart for MaritalStatus vs. TotalPurchases
    dcc.Graph(id='bar-maritalstatus-purchases'),
])


# Define a callback to update the PreferredCategory vs. TotalPurchases bar chart
@app.callback(
    dash.dependencies.Output('bar-preferredcategory-purchases', 'figure'),
    [dash.dependencies.Input('customer-segment-dropdown', 'value')]
)
def update_bar_preferred_category(selected_category):
    grouped_df = data.groupby('PreferredCategory', as_index=False)['TotalPurchases'].sum()
    figure = px.bar(grouped_df, x='PreferredCategory', y='TotalPurchases', title='Preferred
Category vs. Total Purchases')
    return figure

# Define a callback to update the MaritalStatus vs. TotalPurchases bar chart
@app.callback(
    dash.dependencies.Output('bar-maritalstatus-purchases', 'figure'),
    [dash.dependencies.Input('customer-segment-dropdown', 'value')]
)
def update_bar_marital_status(selected_category):
    grouped_df = data.groupby('MaritalStatus', as_index=False)['TotalPurchases'].sum()
    figure = px.bar(grouped_df, x='MaritalStatus', y='TotalPurchases', title='Marital Status vs.
Total Purchases')
    return figure

if __name__ == '__main__':
    app.run_server(debug=True)
```