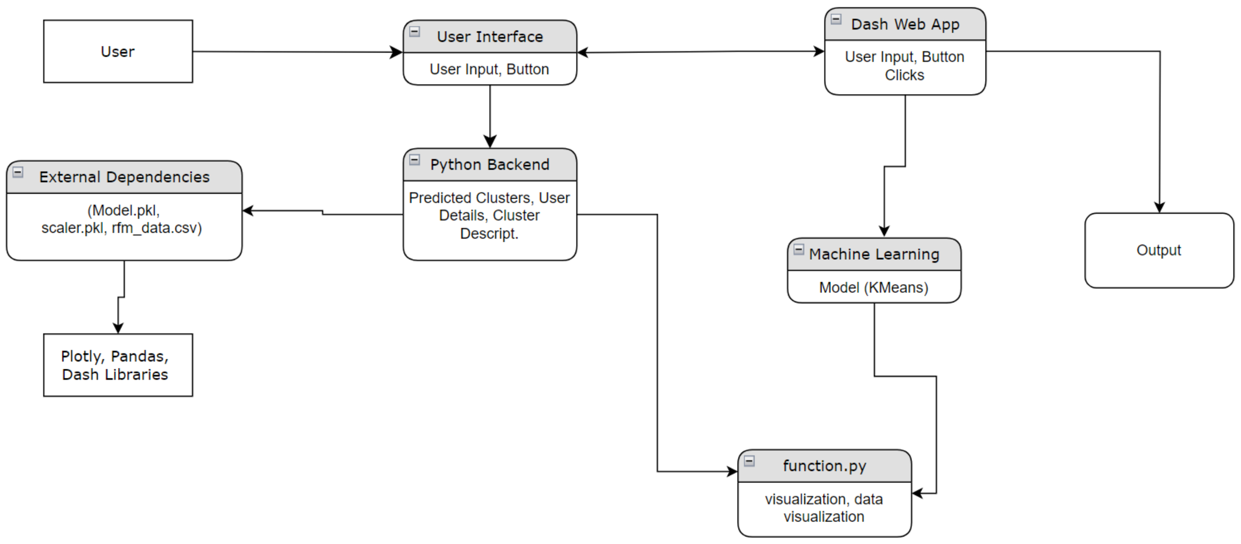
Customer Segmentation

Project Description:

This transformative project endeavors to reshape marketing paradigms by employing a cutting-edge machine learning methodology for customer segmentation, specifically utilizing RFM (Recency, Frequency, Monetary) analysis. The system dynamically categorizes customers based on their transaction recency, frequency, and monetary value, fostering hyper-personalized marketing strategies. The solution architecture encompasses key components such as data collection, RFM analysis, and machine learning integration. Its uniqueness lies in the adaptive nature of the model, ensuring it evolves with shifting customer behaviors. The impact is profound, as businesses can tailor marketing efforts with unprecedented precision, reducing noise and enhancing customer satisfaction. The revenue model involves offering the solution as a subscription-based service with potential consulting services. Highly scalable, the project accommodates diverse industries, promising a dynamic and responsive tool for businesses seeking a competitive edge in understanding and engaging their audience.

Technical Architecture:


User
User Interface
User Input, Button
Dash Web App
User Input, Button Clicks
External Dependencies
(Model.pkl, scaler.pkl, rfm_data.csv)
Python Backend
Predicted Clusters, User Details, Cluster Descript.
Machine Learning
Model (KMeans)
Output
Plotly, Pandas, Dash Libraries
function.py
visualization, data visualization

Pre requisites:

To complete this project, you must require following software's , concepts and packages

- **Programming Language:**
  Proficiency in a programming language such as Python for implementing machine learning algorithms and data analysis.

- **Data Analysis Libraries:**
  Familiarity with data analysis libraries like Pandas and NumPy for processing and manipulating data efficiently.

- **Machine Learning Framework:**
  Understanding of a machine learning framework, such as Scikit-Learn or TensorFlow, for implementing and training machine learning models.

- **Database Knowledge:**
  Basic knowledge of databases, as the project may involve storing and retrieving customer data. Familiarity with databases like MySQL or MongoDB is beneficial.

- **Data Visualization:**
  Knowledge of data visualization tools like Matplotlib or Seaborn to interpret and present findings effectively.

- **RFM Analysis Understanding:**
  Understanding of RFM (Recency, Frequency, Monetary) analysis concepts and its relevance in customer segmentation.

- **Solution Architecture:**
  Familiarity with solution architecture concepts to design and document the system's structure and components.

- **Diagramming Tools:**
  Proficiency in using diagramming tools like Lucidchart, draw.io, or Microsoft Visio for creating solution architecture diagrams.

- **Version Control:**
  Basic understanding of version control systems, such as Git, for collaborative development and project management.

- **Data Privacy and Compliance:**
  Awareness of data privacy regulations and compliance standards relevant to handling customer data.

- **Business Understanding:**
  A fundamental understanding of business and marketing concepts to align the technical solution with organizational goals.

Project Objectives:

- Enhance marketing precision through machine learning-driven RFM analysis.
- Foster hyper-personalized interactions by dynamically categorizing customers.
- Streamline marketing efforts, tailoring campaigns for maximum efficiency.
- Develop a scalable solution adaptable to diverse industries and business sizes.
- Create a dynamic model that evolves with shifting customer behaviors.
- Increase customer satisfaction via targeted and personalized marketing.
- Enable businesses to derive actionable insights from customer data.
- Establish a competitive edge by understanding and engaging the audience effectively.

Project Flow:

- **Data Collection:**
  - Gather relevant customer data from various sources.
  - Ensure data quality and accuracy for robust analysis.

- **RFM Analysis:**
  - Implement Recency, Frequency, and Monetary analysis techniques.
  - Generate individual customer RFM scores for segmentation.

- **Machine Learning Integration:**
  - Utilize machine learning frameworks (e.g., Scikit-Learn) for model development.
  - Train the model using historical customer data for predictive segmentation.

- **Solution Architecture:**
  - Design a scalable architecture incorporating data collection, RFM analysis, and machine learning components.
  - Ensure seamless integration with existing systems.

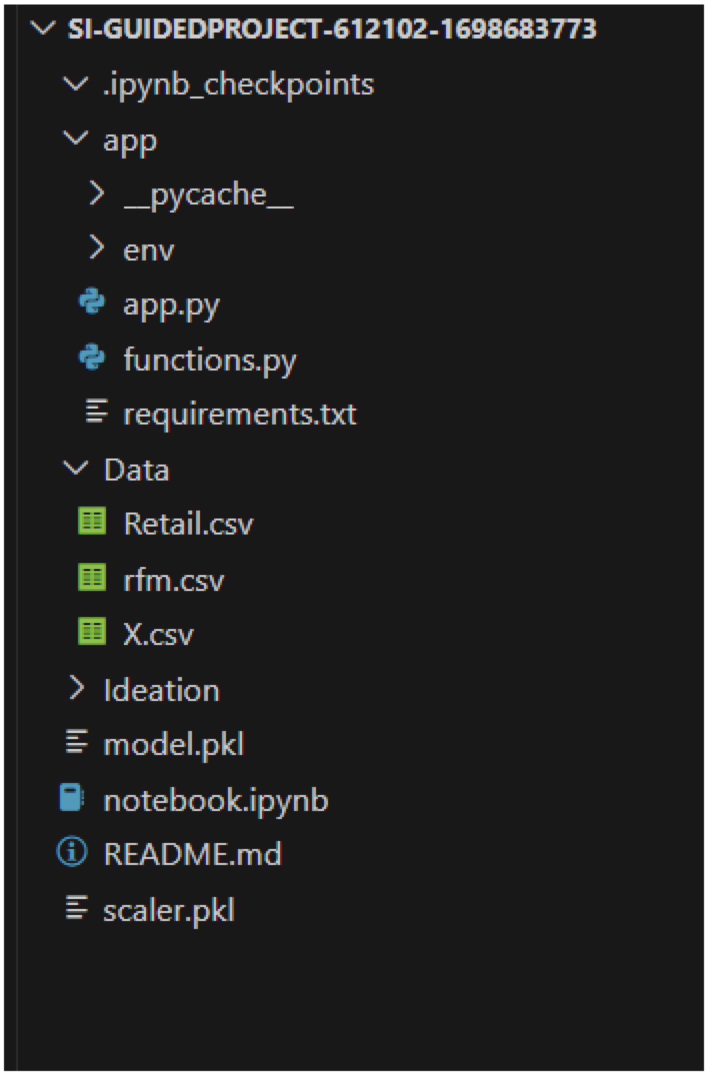
- **Dynamic Adaptability:**
  - Develop the model to adapt to changing customer behaviors.
  - Implement regular updates to maintain relevance.

- **Marketing Strategy Optimization:**
  - Tailor marketing campaigns based on segmented customer groups.
  - Monitor and analyze campaign effectiveness for continuous improvement.

- **User Training and Support:**
  - Provide comprehensive training materials for end-users.
  - Offer ongoing support to address user queries and concerns.

- **Performance Monitoring:**
    - Implement real-time monitoring of system performance.
    - Set up alerts for anomalies and potential issues.

Project Structure



```
SI-GUIDEDPROJECT-612102-1698683773
  .ipynb_checkpoints
  app
    __pycache__
    env
    app.py
    functions.py
    requirements.txt
  Data
    Retail.csv
    rfm.csv
    X.csv
  Ideation
  model.pkl
  notebook.ipynb
  README.md
  scaler.pkl
```

A python file called app.py for server side scripting.

- We need the model which is saved and the saved model in this content is model.pkl
- Scaler.pkl for scaling

Milestone 1: Data Collection: ML depends heavily on data, without data, it is impossible for an “AI” to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

Activity1: Download the dataset

You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository etc. Please refer to the link given below to download the data set and to know about the dataset

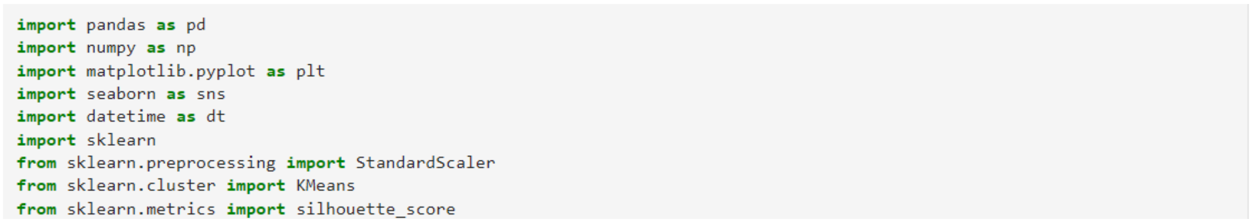
https://www.kaggle.com/code/sarahm/customer-segmentation-using-rfm-analysis

Milestone 2: Data Pre-processing

Data Pre-processing includes the following main tasks

o Import the Libraries.
o Importing the dataset.
o Checking for Null Values.
o Data Visualization.
o Taking care of Missing Data.
o Feature Scaling.

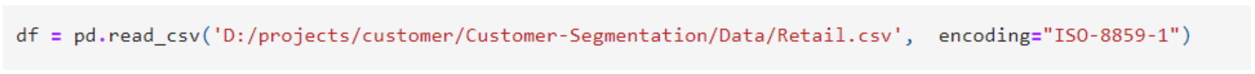
Activity 1: Import Necessary Libraries



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

This activity involves importing the essential libraries for data manipulation and analysis. Key libraries like pandas for data manipulation, numpy for numerical operations, matplotlib and seaborn for visualization, datetime for handling date-related operations, sklearn for machine learning, and pickle for saving models are imported.

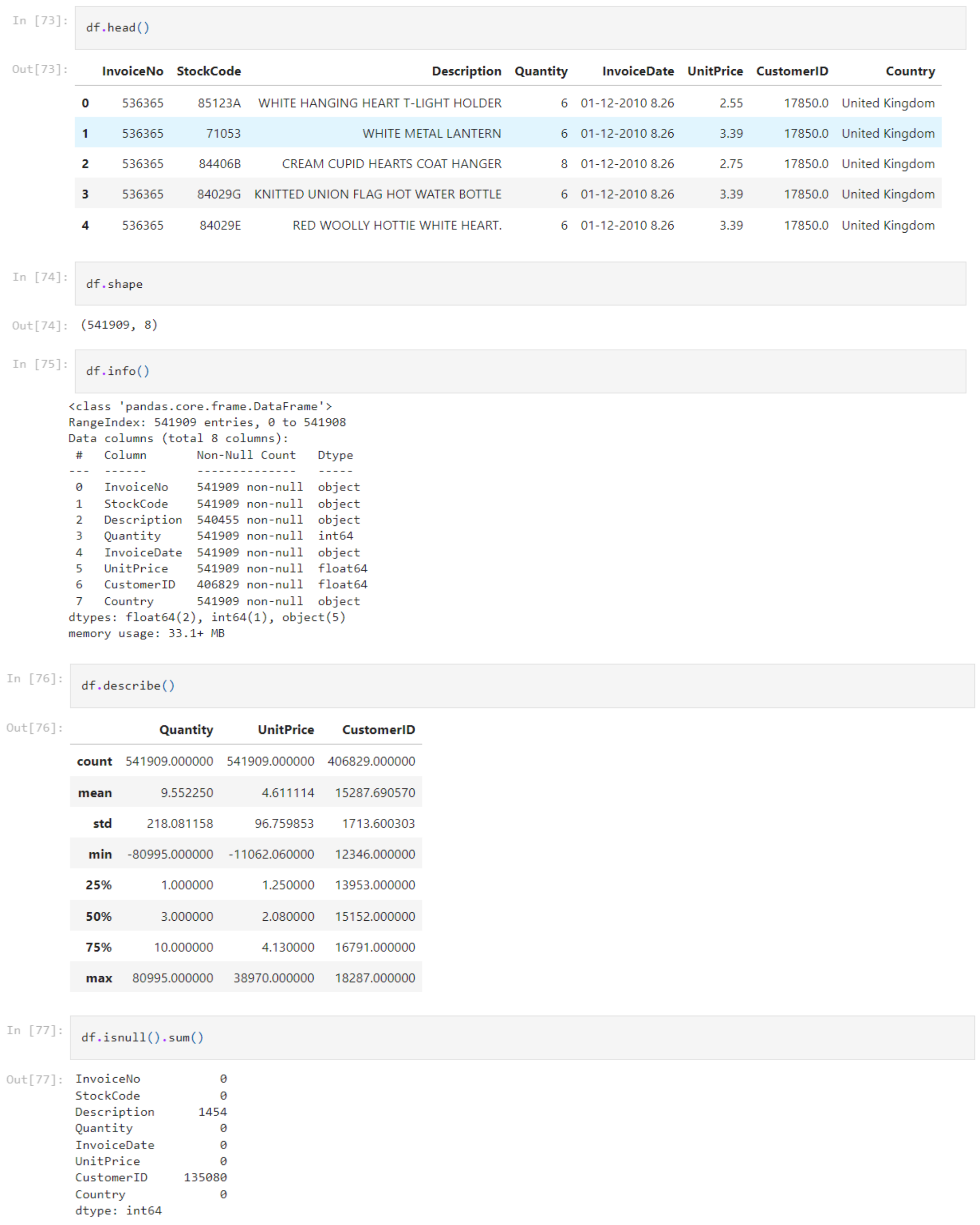
Activity 2: Importing the Dataset



```
df = pd.read_csv('D:/projects/customer/Customer-Segmentation/Data/Retail.csv',  encoding="ISO-8859-1")
```

This code reads the dataset from a CSV file into a Pandas DataFrame. The encoding parameter is used to handle any special characters in the dataset.

Activity 3: Analyze the data



```
In [73]: df.head()
```

Out[73]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 8.26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	01-12-2010 8.26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 8.26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01-12-2010 8.26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01-12-2010 8.26	3.39	17850.0	United Kingdom

```
In [74]: df.shape

Out[74]: (541909, 8)

In [75]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    541909 non-null  object 
 1   StockCode    541909 non-null  object 
 2   Description  540455 non-null  object 
 3   Quantity     541909 non-null  int64  
 4   InvoiceDate  541909 non-null  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB

In [76]: df.describe()
```

Out[76]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

```
In [77]: df.isnull().sum()

Out[77]: InvoiceNo           0
         StockCode           0
         Description      1454
         Quantity            0
         InvoiceDate         0
         UnitPrice           0
         CustomerID     135080
         Country             0
         dtype: int64
```

These commands provide an initial analysis of the dataset. head() shows the first few rows, shape gives the dimensions, info() provides information about data types and missing values, describe() gives statistical summary, and isnull().sum() shows the count of missing values in each column.

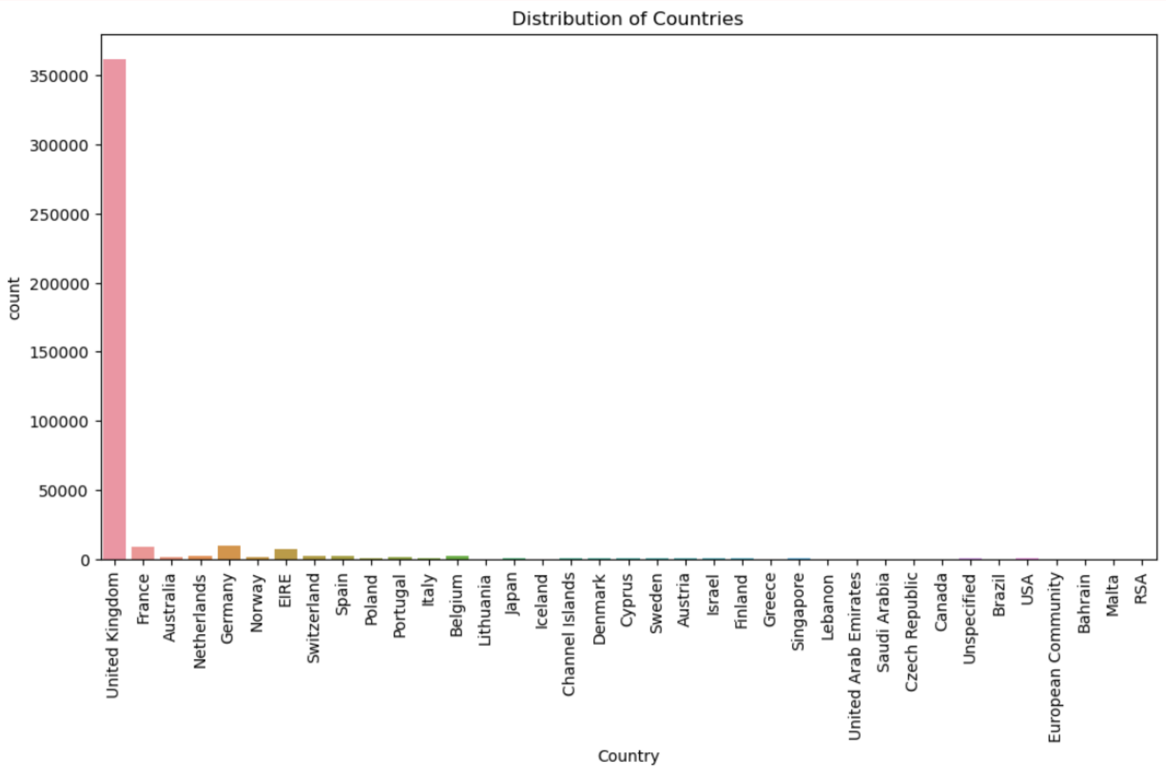
Activity 4: Handling Missing Values

```
In [78]:  df = df.dropna()
          df.shape

Out[78]:  (406829, 8)
```

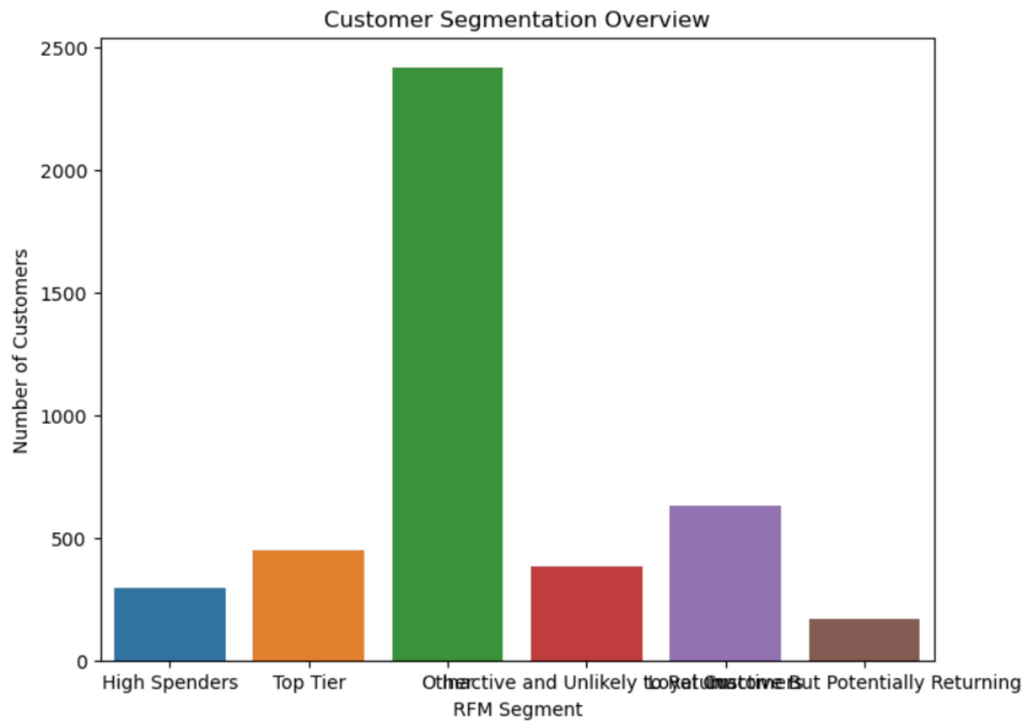
This line removes rows with any missing values. It's a straightforward approach, and in practice, you might want to explore other methods like imputation depending on the context.

Activity 5: Data Visualization

```
In [82]:  plt.figure(figsize=(12, 6))
          sns.countplot(x='Country', data=df)
          plt.title('Distribution of Countries')
          plt.xticks(rotation=90)
          plt.show()
```



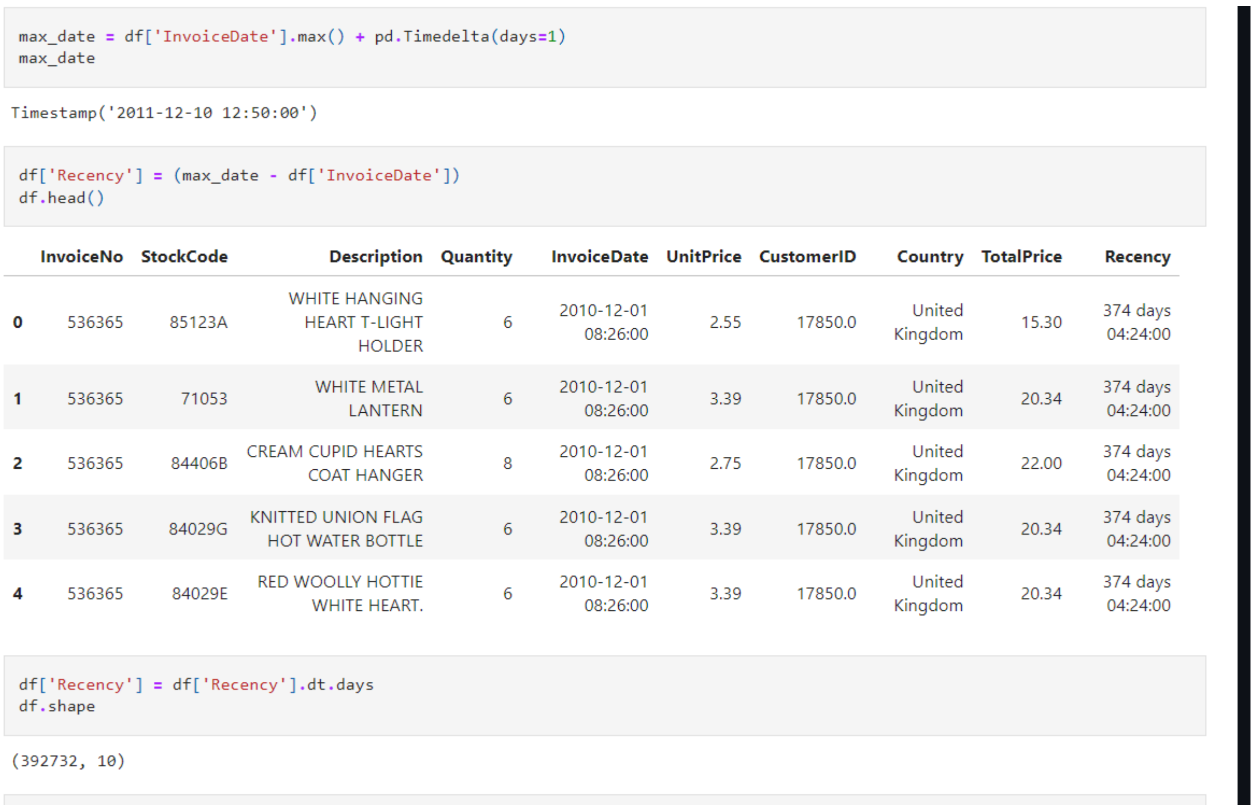
```
plt.figure(figsize=(8, 6))
sns.countplot(x='RFM_Segment', data=segmentation_data)
plt.title('Customer Segmentation Overview')
plt.xlabel('RFM Segment')
plt.ylabel('Number of Customers')
plt.show()
```



This code creates a count plot to visualize the distribution of data across different countries. It uses matplotlib and seaborn for plotting.

Activity 6: Feature Engineering

```
#df['InvoiceDay'] = df['InvoiceDate'].dt.date
df['TotalPrice'] = df['Quantity'] * df['UnitPrice']
```



```python
max_date = df['InvoiceDate'].max() + pd.Timedelta(days=1)
max_date
```

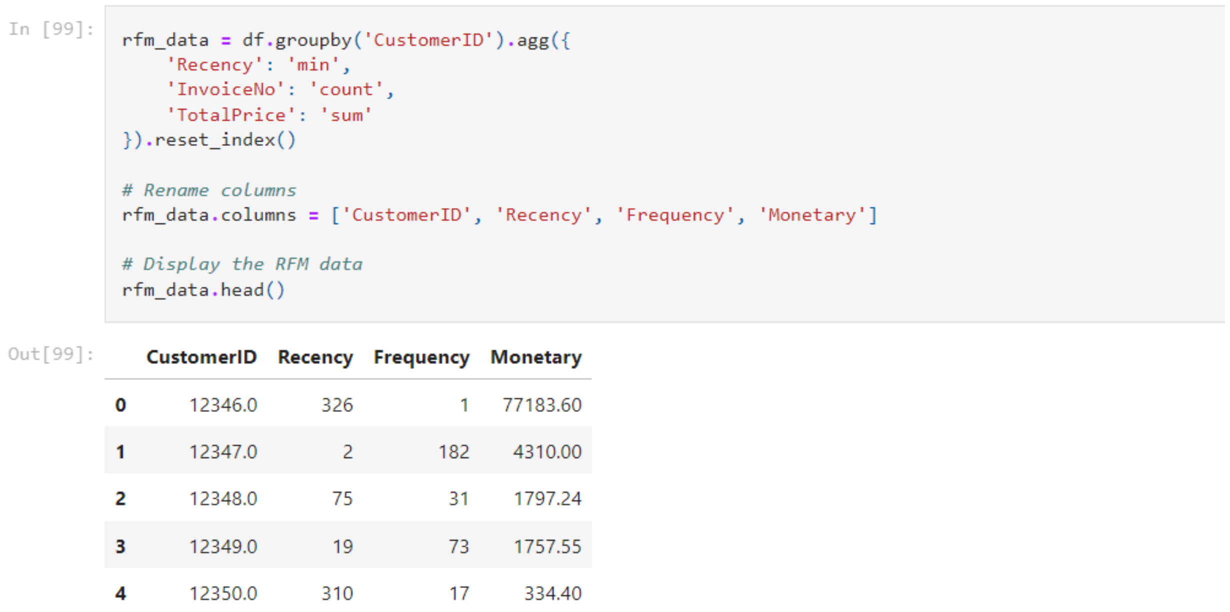
```
Timestamp('2011-12-10 12:50:00')
```

```python
df['Recency'] = (max_date - df['InvoiceDate'])
df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalPrice	Recency
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	15.30	374 days 04:24:00
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	374 days 04:24:00
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	22.00	374 days 04:24:00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	374 days 04:24:00
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	20.34	374 days 04:24:00

```python
df['Recency'] = df['Recency'].dt.days
df.shape
```

```
(392732, 10)
```



In [99]:

```python
rfm_data = df.groupby('CustomerID').agg({
    'Recency': 'min',
    'InvoiceNo': 'count',
    'TotalPrice': 'sum'
}).reset_index()

# Rename columns
rfm_data.columns = ['CustomerID', 'Recency', 'Frequency', 'Monetary']

# Display the RFM data
rfm_data.head()
```

Out[99]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	326	1	77183.60
1	12347.0	2	182	4310.00
2	12348.0	75	31	1797.24
3	12349.0	19	73	1757.55
4	12350.0	310	17	334.40

In [102…

```
quantiles = rfm_data.quantile(q=[0.25,0.5,0.75])
quantiles = quantiles.to_dict()
rfm = rfm_data
```

In [103…

```
def RClass(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4

def FMClass(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
```

In [104…

```
rfm['R_Quartile'] = rfm['Recency'].apply(RClass, args=('Recency',quantiles,))
rfm['F_Quartile'] = rfm['Frequency'].apply(FMClass, args=('Frequency',quantiles,))
rfm['M_Quartile'] = rfm['Monetary'].apply(FMClass, args=('Monetary',quantiles,))
```

In [105…

```
rfm['RFMClass'] = rfm.R_Quartile.map(str) \
                            + rfm.F_Quartile.map(str) \
                            + rfm.M_Quartile.map(str)

rfm
```

Out[105…

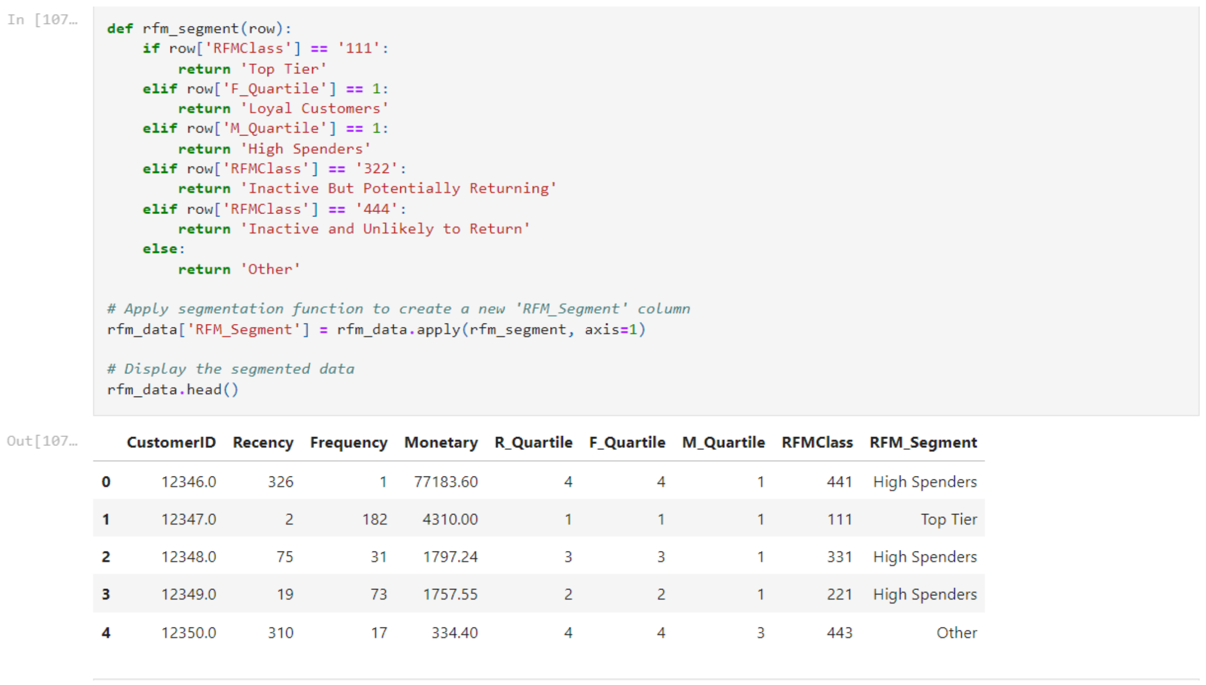
	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMClass
0	12346.0	326	1	77183.60	4	4	1	441
1	12347.0	2	182	4310.00	1	1	1	111
2	12348.0	75	31	1797.24	3	3	1	331
3	12349.0	19	73	1757.55	2	2	1	221
4	12350.0	310	17	334.40	4	4	3	443
...	...	...	...	...	...	...	...	...
4334	18280.0	278	10	180.60	4	4	4	444
4335	18281.0	181	7	80.82	4	4	4	444
4336	18282.0	8	12	178.05	1	4	4	144
4337	18283.0	4	721	2045.53	1	1	1	111
4338	18287.0	43	70	1837.28	2	2	1	221

4339 rows × 8 columns

In [106…

```
print("Top tier: ",len(rfm[rfm['RFMClass']=='111']), "(", round(len(rfm[rfm['RFMClass']=='111'])/len(rfm)*100,2), "%)")
print('Loyal Customers: ',len(rfm[rfm['F_Quartile']==1]), "(", round(len(rfm[rfm['F_Quartile']==1])/len(rfm)*100,2), "%)")
print("high Spenders: ",len(rfm[rfm['M_Quartile']==1]), "(", round(len(rfm[rfm['M_Quartile']==1])/len(rfm)*100,2), "%)")
print('Inactive But Potentially Returning: ', len(rfm[rfm['RFMClass']=='322']), "(", round(len(rfm[rfm['RFMClass']=='322'])/
print('Inactive and Unlikely to Return: ',len(rfm[rfm['RFMClass']=='444']), "(", round(len(rfm[rfm['RFMClass']=='444'])/len
```

```
Top tier:  450 ( 10.37 %)
Loyal Customers:  1080 ( 24.89 %)
high Spenders:  1085 ( 25.01 %)
Inactive But Potentially Returning:  168 ( 3.87 %)
Inactive and Unlikely to Return:  381 ( 8.78 %)
```



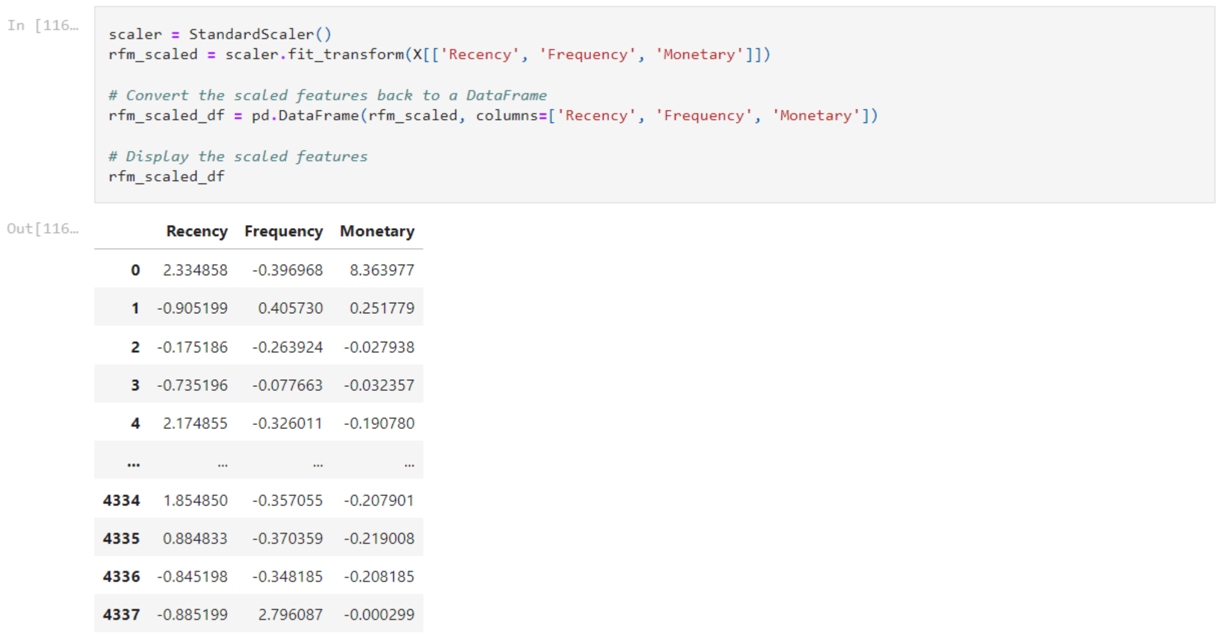
```
In [107…
def rfm_segment(row):
    if row['RFMClass'] == '111':
        return 'Top Tier'
    elif row['F_Quartile'] == 1:
        return 'Loyal Customers'
    elif row['M_Quartile'] == 1:
        return 'High Spenders'
    elif row['RFMClass'] == '322':
        return 'Inactive But Potentially Returning'
    elif row['RFMClass'] == '444':
        return 'Inactive and Unlikely to Return'
    else:
        return 'Other'

# Apply segmentation function to create a new 'RFM_Segment' column
rfm_data['RFM_Segment'] = rfm_data.apply(rfm_segment, axis=1)

# Display the segmented data
rfm_data.head()
```

Out[107…

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMClass	RFM_Segment
0	12346.0	326	1	77183.60	4	4	1	441	High Spenders
1	12347.0	2	182	4310.00	1	1	1	111	Top Tier
2	12348.0	75	31	1797.24	3	3	1	331	High Spenders
3	12349.0	19	73	1757.55	2	2	1	221	High Spenders
4	12350.0	310	17	334.40	4	4	3	443	Other

Features added for RFM analysis

Activity 7: Feature Scaling



```
In [116…
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(X[['Recency', 'Frequency', 'Monetary']])

# Convert the scaled features back to a DataFrame
rfm_scaled_df = pd.DataFrame(rfm_scaled, columns=['Recency', 'Frequency', 'Monetary'])

# Display the scaled features
rfm_scaled_df
```

Out[116…

	Recency	Frequency	Monetary
0	2.334858	-0.396968	8.363977
1	-0.905199	0.405730	0.251779
2	-0.175186	-0.263924	-0.027938
3	-0.735196	-0.077663	-0.032357
4	2.174855	-0.326011	-0.190780
...	...	...	...
4334	1.854850	-0.357055	-0.207901
4335	0.884833	-0.370359	-0.219008
4336	-0.845198	-0.348185	-0.208185
4337	-0.885199	2.796087	-0.000299

Feature scaling is a method used to normalize the range of independent variables or features of data.

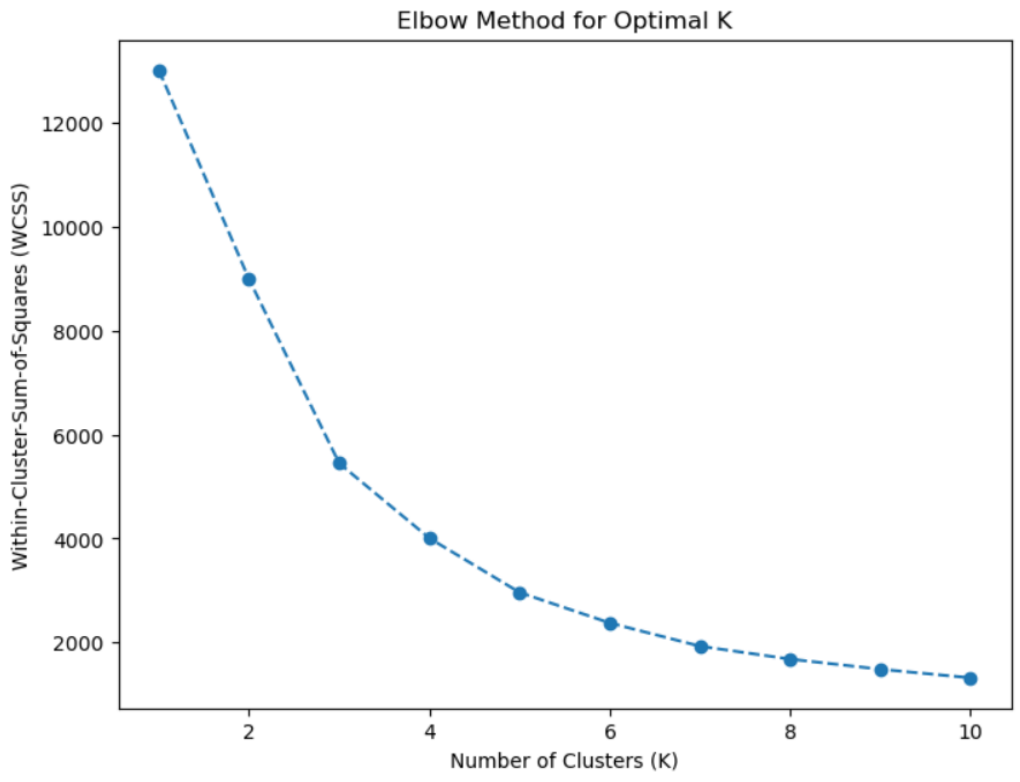
- After scaling the data will be converted into array form
- Loading the feature names before scaling and converting them back to data frame after standard scaling is applied

Milestone : Clustering

Activity 1 : Model and Clustering Explanation:

```
sse = []  # Within-Cluster-Sum-of-Squares

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(rfm_scaled_df)
    sse.append(kmeans.inertia_)

# Plot the Elbow Method
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), sse, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Within-Cluster-Sum-of-Squares (WCSS)')
plt.show()
```

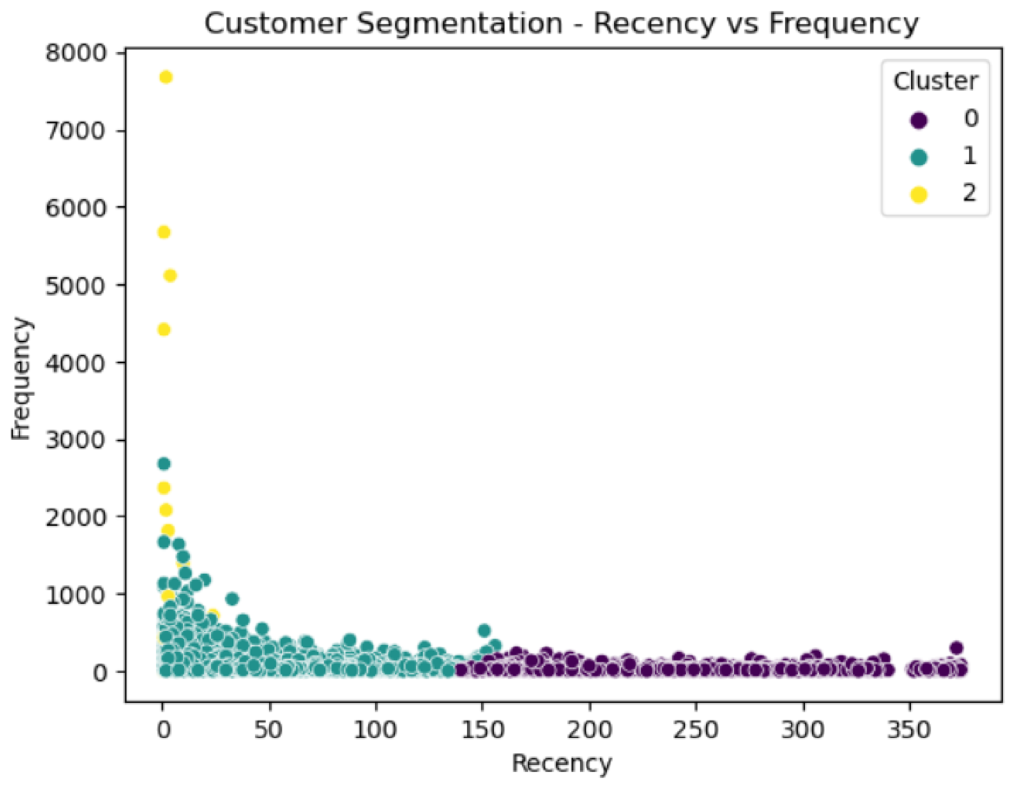
In this part, you are using the KMeans algorithm to perform clustering. The Elbow Method is employed to determine the optimal number of clusters (K). You fit the KMeans model for different values of K and calculate the Within-Cluster-Sum-of-Squares (WCSS), storing the results in the sse list.



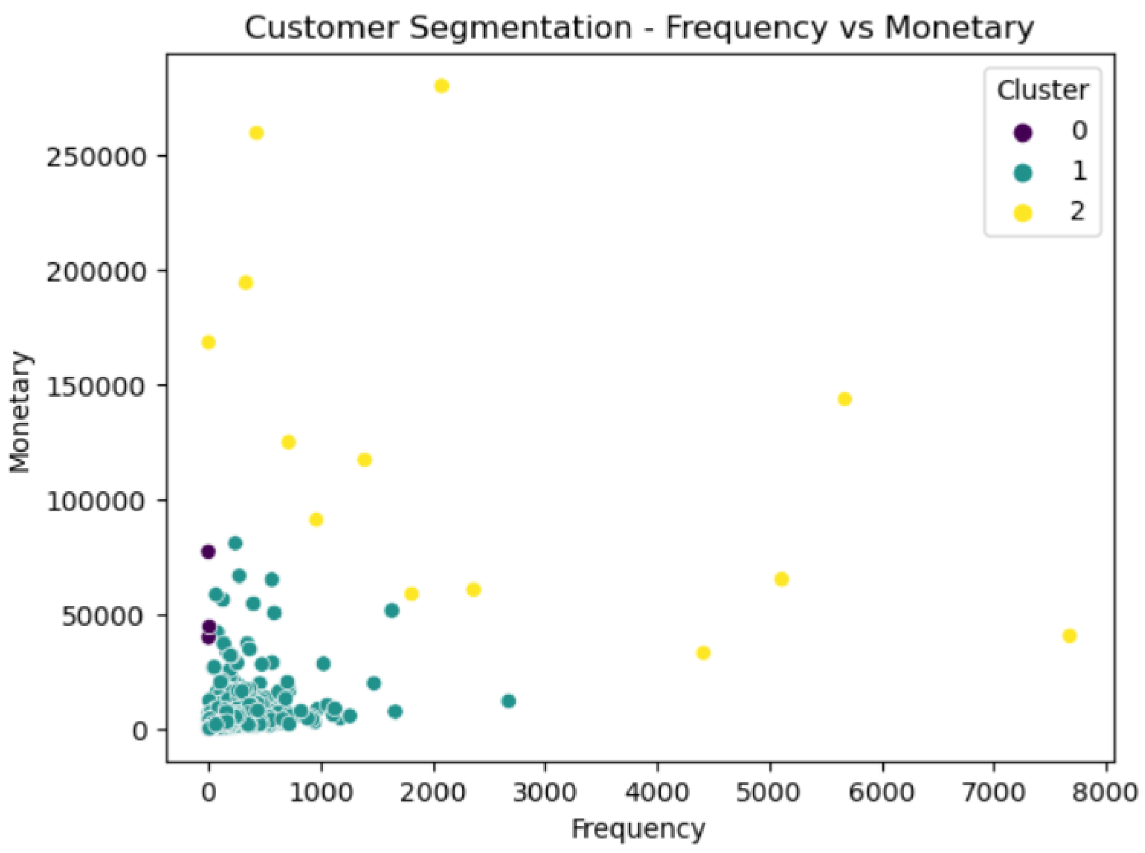
This code then plots the WCSS against the number of clusters. The "elbow" in the graph is often the point where the rate of decrease in WCSS slows down, suggesting the optimal number of clusters.

```python
optimal_k = 3  # Update this based on the visual inspection of the elbow curve

# Applying KMeans clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
X['Cluster'] = kmeans.fit_predict(rfm_scaled)

# Visualizing the clusters
sns.scatterplot(x='Recency', y='Frequency', hue='Cluster', data=X, palette='viridis')
plt.title('Customer Segmentation - Recency vs Frequency')
plt.show()

sns.scatterplot(x='Frequency', y='Monetary', hue='Cluster', data=X, palette='viridis')
plt.title('Customer Segmentation - Frequency vs Monetary')
plt.show()

# Assessing cluster sizes
cluster_sizes = X['Cluster'].value_counts().sort_index()
print(cluster_sizes)

# Assessing silhouette score for clustering quality
silhouette_avg = silhouette_score(rfm_scaled, X['Cluster'])
print(f'Silhouette Score: {silhouette_avg}')
```

Based on the elbow method analysis, you choose an optimal value for K (in this case, 3). You then fit the KMeans model again with this optimal K and assign each data point to a cluster. The cluster labels are stored in the 'Cluster' column of your DataFrame.



These visualizations show the clusters formed by the KMeans algorithm. Each point represents a customer, and their position on the plot is determined by their recency, frequency, and monetary values. The coloring represents the cluster they belong to.



```
Cluster
0    1080
1    3246
2      13
Name: count, dtype: int64
Silhouette Score: 0.6023126726335443
```

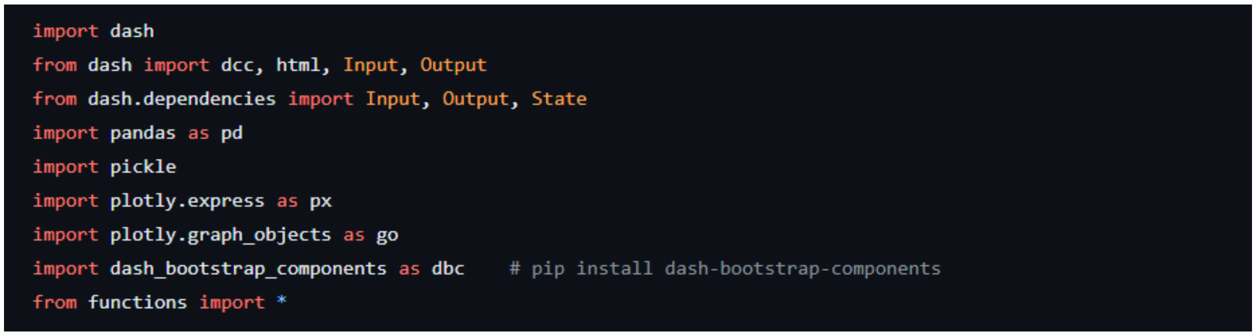
Here, evaluate the sizes of each cluster and calculate the silhouette score, which is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). A higher silhouette score indicates better-defined clusters.

Milestone : Saving Model

```
# Save the KMeans model
#with open('model.pkl', 'wb') as model_file:
#    pickle.dump(kmeans, model_file)

# Save the scaler
#with open('scaler.pkl', 'wb') as scaler_file:
#    pickle.dump(scaler, scaler_file)
```

Milestone : Application

Activity 1: Import Necessary Libraries

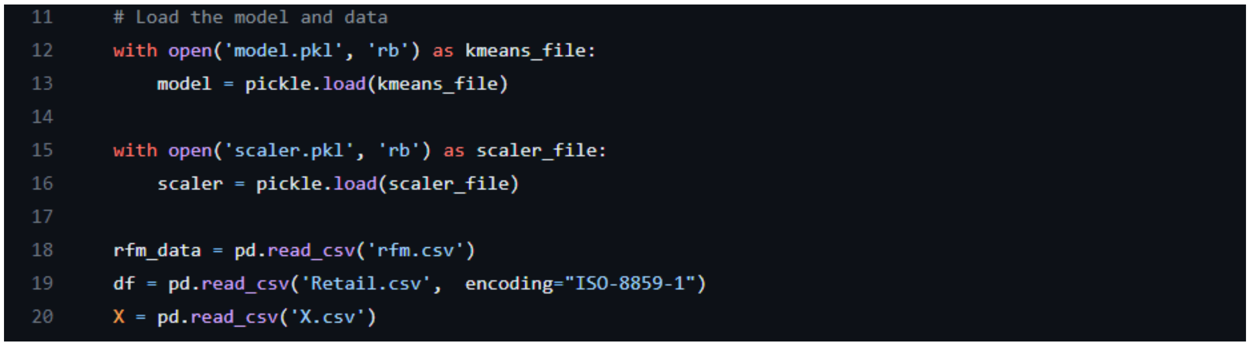


```python
import dash
from dash import dcc, html, Input, Output
from dash.dependencies import Input, Output, State
import pandas as pd
import pickle
import plotly.express as px
import plotly.graph_objects as go
import dash_bootstrap_components as dbc    # pip install dash-bootstrap-components
from functions import *
```

This activity involves importing the necessary libraries for building a Dash web application. Dash is used for creating interactive web-based data visualizations, and other libraries like plotly, dash-bootstrap-components, and pandas are used to enhance functionality.

Activity 2: Load Model and Data

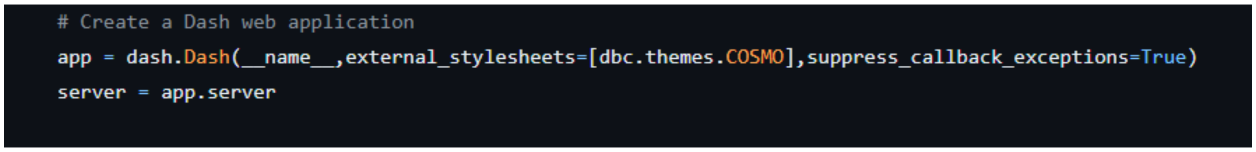
This activity involves loading the pre-trained clustering model (KMeans) and the scaler used for standardization. The rfm_data, df, and X DataFrames are loaded from CSV files.



```python
# Load the model and data
with open('model.pkl', 'rb') as kmeans_file:
    model = pickle.load(kmeans_file)

with open('scaler.pkl', 'rb') as scaler_file:
    scaler = pickle.load(scaler_file)

rfm_data = pd.read_csv('rfm.csv')
df = pd.read_csv('Retail.csv',  encoding="ISO-8859-1")
X = pd.read_csv('X.csv')
```

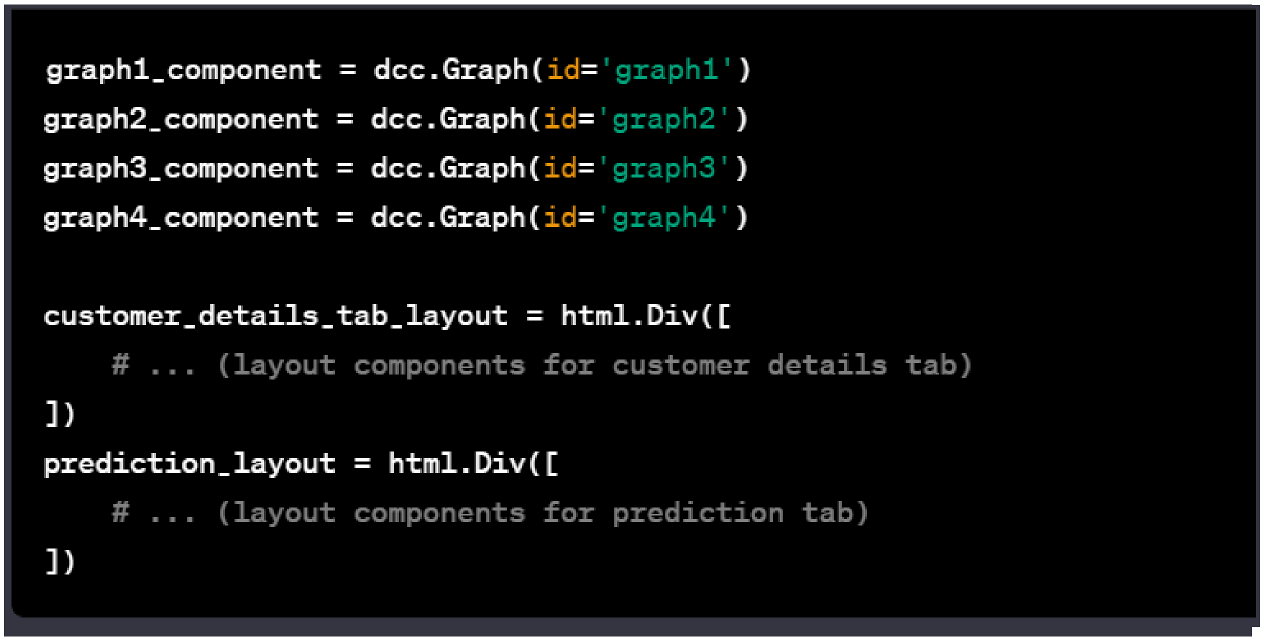
Activity 3: Create Dash Application

This activity creates a Dash web application with a Bootstrap theme.



```python
# Create a Dash web application
app = dash.Dash(__name__,external_stylesheets=[dbc.themes.COSMO],suppress_callback_exceptions=True)
server = app.server
```

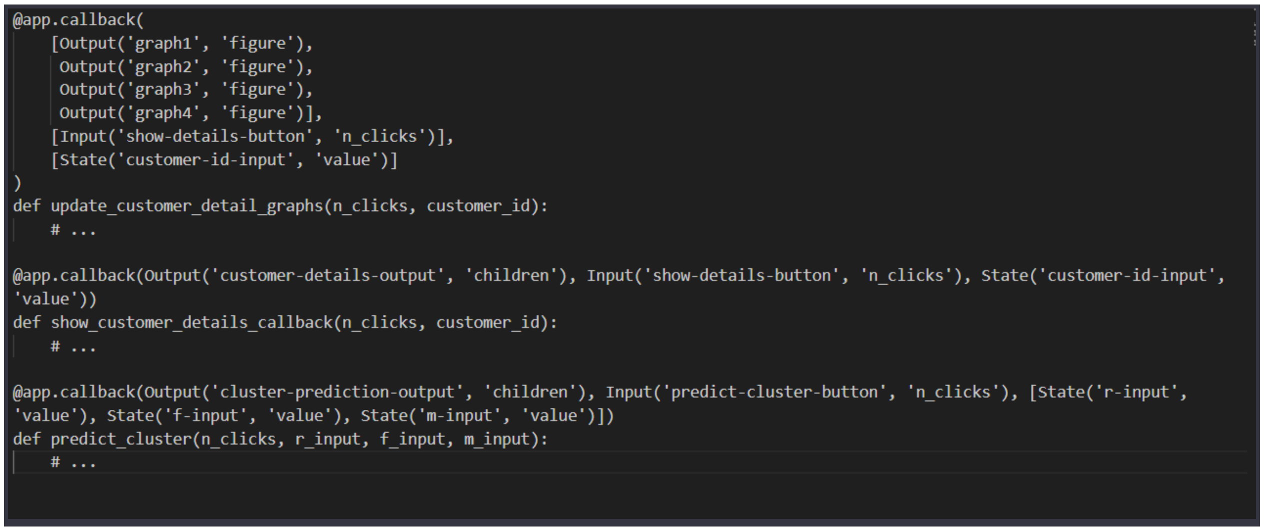
Activity 4: Layout Definition



```python
graph1_component = dcc.Graph(id='graph1')
graph2_component = dcc.Graph(id='graph2')
graph3_component = dcc.Graph(id='graph3')
graph4_component = dcc.Graph(id='graph4')

customer_details_tab_layout = html.Div([
    # ... (layout components for customer details tab)
])
prediction_layout = html.Div([
    # ... (layout components for prediction tab)
])
```

Activity 5: Callback Functions

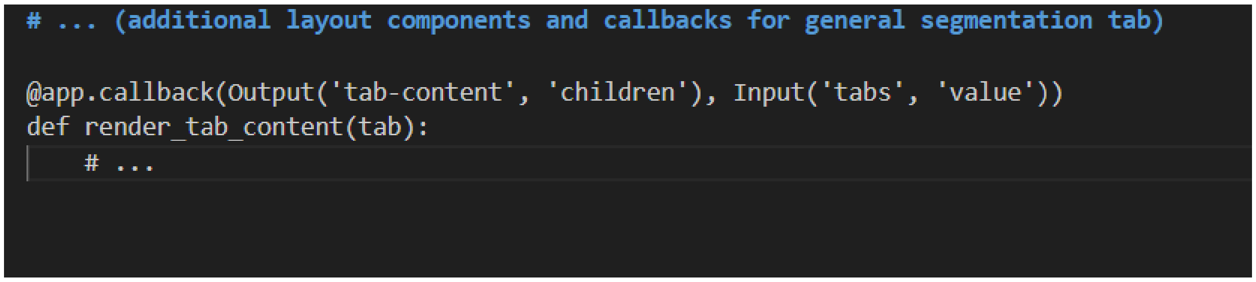
These callback functions define the behavior of your app in response to user interactions. For example, updating graphs when a button is clicked or displaying customer details based on user input.



```python
@app.callback(
    [Output('graph1', 'figure'),
     Output('graph2', 'figure'),
     Output('graph3', 'figure'),
     Output('graph4', 'figure')],
    [Input('show-details-button', 'n_clicks')],
    [State('customer-id-input', 'value')]
)
def update_customer_detail_graphs(n_clicks, customer_id):
    # ...

@app.callback(Output('customer-details-output', 'children'), Input('show-details-button', 'n_clicks'), State('customer-id-input',
'value'))
def show_customer_details_callback(n_clicks, customer_id):
    # ...

@app.callback(Output('cluster-prediction-output', 'children'), Input('predict-cluster-button', 'n_clicks'), [State('r-input',
'value'), State('f-input', 'value'), State('m-input', 'value')])
def predict_cluster(n_clicks, r_input, f_input, m_input):
    # ...
```

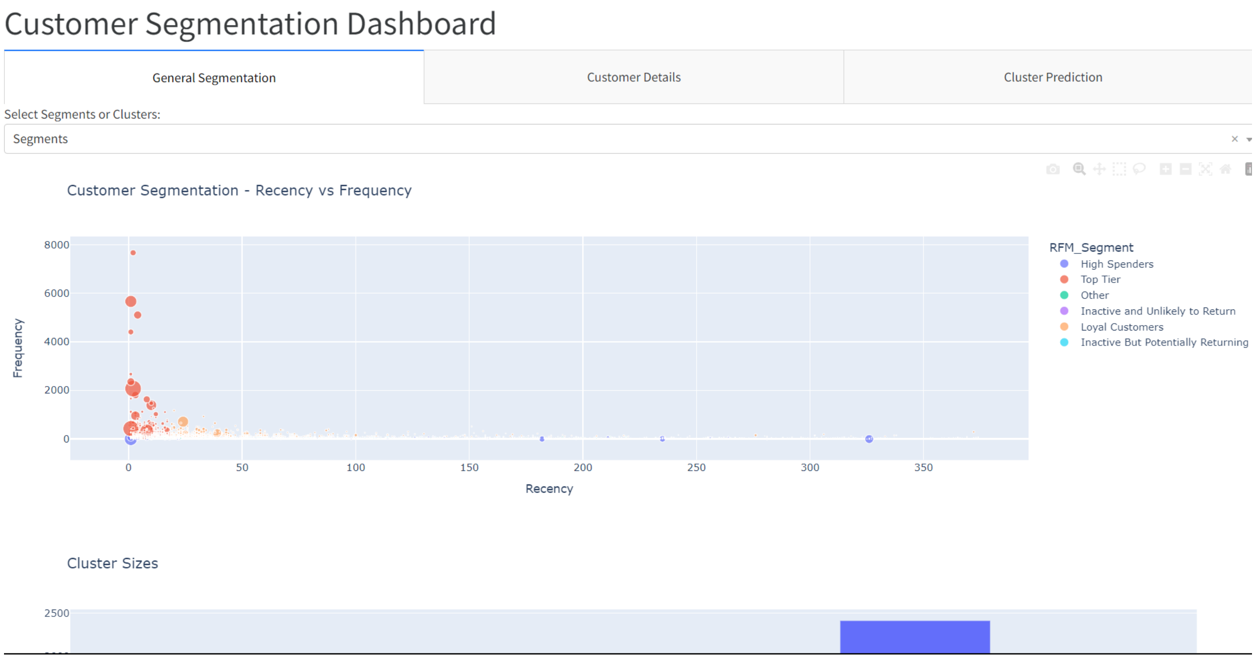
Activity 6: Additional Layout and Callbacks

These components and callbacks handle the general segmentation tab, including a dropdown to select segments or clusters.



```python
# ... (additional layout components and callbacks for general segmentation tab)

@app.callback(Output('tab-content', 'children'), Input('tabs', 'value'))
def render_tab_content(tab):
    # ...
```

Output



Customer Segmentation Dashboard

General Segmentation | Customer Details | Cluster Prediction

Select Segments or Clusters:

Segments