```
#Step 1 – Import Necessary Libraries and Modules
#We need pandas and matplotlib for data exploration and visualization
#KMeans class from scikit-learn’s cluster module to perform K-Means
clustering
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

#Step 2 – Load the Dataset
data = pd.read_excel(r'/content/Online Retail.xlsx')

#Step 3 – Explore and Clean the Dataset
data.head()
```

{"type":"dataframe","variable_name":"data"}

```
#calling the describe method on the dataframe to understand the
numerical features
data.describe()
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 8,\n \"fields\": [\
n   {\n   \"column\": \"Quantity\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 196412.4226608867,\n
\"min\": -80995.0,\n   \"max\": 541909.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
9.55224954743324,\n   10.0,\n   541909.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"InvoiceDate\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"1970-01-01 00:00:00.000541909\",\n   \"max\": \"2011-12-09
12:50:00\",\n   \"num_unique_values\": 7,\n   \"samples\":
[\n   \"541909\",\n   \"2011-07-04
13:34:57.156386048\",\n   \"2011-10-19 11:27:00\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"UnitPrice\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
190752.07570771928,\n   \"min\": -11062.06,\n   \"max\":
541909.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n
4.611113626088513,\n   4.13,\n   541909.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"CustomerID\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
139204.1680069419,\n   \"min\": 1713.600303321598,\n
\"max\": 406829.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   15287.690570239585,\n   16791.0,\n
406829.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
data.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  datetime64[ns]
 5   UnitPrice    541909 non-null  float64       
 6   CustomerID   406829 non-null  float64       
 7   Country      541909 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

```python
# Check for missing values in each column
missing_values = data.isnull().sum()
print(missing_values)
```

```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

```python
# Drop rows with missing CustomerID
data.dropna(subset=['CustomerID'], inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 406829 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   InvoiceNo    406829 non-null  object        
 1   StockCode    406829 non-null  object        
 2   Description  406829 non-null  object        
 3   Quantity     406829 non-null  int64         
 4   InvoiceDate  406829 non-null  datetime64[ns]
 5   UnitPrice    406829 non-null  float64       
 6   CustomerID   406829 non-null  float64       
 7   Country      406829 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 27.9+ MB
```

```
#validating if any missing values present
missing_values = data.isnull().sum()
print(missing_values)
```

```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

```
# Remove rows with negative Quantity and Price
data = data[(data['Quantity'] > 0) & (data['UnitPrice'] > 0)]
data.describe()
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 8,\n  \"fields\": [\n    {\n      \"column\": \"Quantity\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 139478.9890002285,\n        \"min\": 1.0,\n        \"max\": 397884.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          12.988237777844798,\n          12.0,\n          397884.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"InvoiceDate\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000397884\",\n        \"max\": \"2011-12-09 12:50:00\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n          \"397884\",\n          \"2011-07-10 23:41:23.511023360\",\n          \"2011-10-20 14:33:00\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"UnitPrice\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 140289.24304942065,\n        \"min\": 0.001,\n        \"max\": 397884.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          3.11648775522514,\n          3.75,\n          397884.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"CustomerID\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 136042.15806984305,\n        \"min\": 1713.141560439856,\n        \"max\": 397884.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          15294.423452564064,\n          16795.0,\n          397884.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
#converting the “CustomerID” to an integer
data['CustomerID'] = data['CustomerID'].astype(int)
```

```
# Verify the data type conversion
print(data.dtypes)
```

```
InvoiceNo              object
StockCode              object
Description            object
Quantity                int64
InvoiceDate    datetime64[ns]
UnitPrice             float64
CustomerID              int64
Country                object
dtype: object
```

```
#Step 4 — Compute Recency, Frequency, and Monetary Value

#defining reference date "snapshot_date" that’s a day later than the
most recent date in the “InvoiceDate” column

snapshot_date = max(data['InvoiceDate']) + pd.DateOffset(days=1)
print(snapshot_date)
```

```
2011-12-10 12:50:00
```

```
#Creating a “Total” column that contains Quantity*UnitPrice for all
the records
data['Total'] = data['Quantity'] * data['UnitPrice']
print(data['Total'])
```

```
0         15.30
1         20.34
2         22.00
3         20.34
4         20.34
          ...
541904    10.20
541905    12.60
541906    16.60
541907    16.60
541908    14.85
Name: Total, Length: 397884, dtype: float64
```

```
#InvoiceDate : calculated by subtracting snapshot_date from
'max(invoice_date) for each customer'
#InvoiceNo : number of unique invoiceNo for each customer
#Total : summing up the 'Total' calculated above for each customer
rfm = data.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
    'InvoiceNo': 'nunique',
    'Total': 'sum'
})
```

```python
#calculating max_invoice_date per customer
max_invoice_date_per_customer =
data.groupby('CustomerID').agg({'InvoiceDate': 'max'})
print(max_invoice_date_per_customer)
```

```
                  InvoiceDate
CustomerID
12346      2011-01-18 10:01:00
12347      2011-12-07 15:52:00
12348      2011-09-25 13:13:00
12349      2011-11-21 09:51:00
12350      2011-02-02 16:01:00
...                        ...
18280      2011-03-07 09:52:00
18281      2011-06-12 10:53:00
18282      2011-12-02 11:43:00
18283      2011-12-06 12:02:00
18287      2011-10-28 09:29:00

[4338 rows x 1 columns]
```

```python
#rename the columns for readability
rfm.rename(columns={'InvoiceDate': 'Recency', 'InvoiceNo':
'Frequency', 'Total': 'MonetaryValue'}, inplace=True)
rfm.head()
```


```
{"summary":"{\n  \"name\": \"rfm\",\n  \"rows\": 4338,\n  \"fields\":
[\n    {\n      \"column\": \"CustomerID\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1721,\n        \"min\":
12346,\n        \"max\": 18287,\n        \"num_unique_values\": 4338,\
n        \"samples\": [\n          17785,\n          14320,\n
15977\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Recency\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 100,\n        \"min\": 1,\n        \"max\": 374,\n
\"num_unique_values\": 349,\n        \"samples\": [\n          187,\n
118,\n          69\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Frequency\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7,\n        \"min\": 1,\n
\"max\": 209,\n        \"num_unique_values\": 59,\n
\"samples\": [\n          1,\n          2,\n          62\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MonetaryValue\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8989.230441338681,\n        \"min\": 3.75,\n        \"max\":
280206.02,\n        \"num_unique_values\": 4253,\n        \"samples\":
[\n          2794.51,\n          379.35,\n          954.09\
n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"rfm"}
```

```
#Step 5 – Map RFM Values onto a 1-5 Scale
rfm.describe()
```


```
{"summary":"{\n  \"name\": \"rfm\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"Recency\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1498.9229266216273,\n
\"min\": 1.0,\n        \"max\": 4338.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
92.53642231443061,\n          51.0,\n          4338.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Frequency\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1523.7988128734833,\n        \"min\": 1.0,\n        \"max\": 4338.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          4338.0,\n
4.272014753342554,\n          5.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MonetaryValue\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
98201.36857420603,\n        \"min\": 3.75,\n        \"max\":
280206.02,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          2054.2664601198708,\n          674.485,\n          4338.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


```
# Calculate custom bin edges for Recency, Frequency, and Monetary
scores
recency_bins = [rfm['Recency'].min()-1, 20, 50, 150, 250,
rfm['Recency'].max()]
frequency_bins = [rfm['Frequency'].min() - 1, 2, 3, 10, 100,
rfm['Frequency'].max()]
monetary_bins = [rfm['MonetaryValue'].min() - 3, 300, 600, 2000, 5000,
rfm['MonetaryValue'].max()]

# Calculate Recency score based on custom bins
rfm['R_Score'] = pd.cut(rfm['Recency'], bins=recency_bins,
labels=range(1, 6), include_lowest=True)

# Reverse the Recency scores so that higher values indicate more
recent purchases
rfm['R_Score'] = 5 - rfm['R_Score'].astype(int) + 1

# Calculate Frequency and Monetary scores based on custom bins
rfm['F_Score'] = pd.cut(rfm['Frequency'], bins=frequency_bins,
labels=range(1, 6), include_lowest=True).astype(int)

rfm['M_Score'] = pd.cut(rfm['MonetaryValue'], bins=monetary_bins,
labels=range(1, 6), include_lowest=True).astype(int)
```

```python
# Print the first few rows of the RFM DataFrame to verify the scores
print(rfm[['R_Score', 'F_Score', 'M_Score']].head(10))
```

```
            R_Score  F_Score  M_Score
CustomerID
12346             1        1        5
12347             5        3        4
12348             3        3        3
12349             5        1        3
12350             1        1        2
12352             4        3        4
12353             2        1        1
12354             2        1        3
12355             2        1        2
12356             4        2        4
```

```python
#extracting the R, F, and M scores to perform K-Means clustering

X = rfm[['R_Score', 'F_Score', 'M_Score']]
print(X)
```

```
            R_Score  F_Score  M_Score
CustomerID
12346             1        1        5
12347             5        3        4
12348             3        3        3
12349             5        1        3
12350             1        1        2
...             ...      ...      ...
18280             1        1        1
18281             2        1        1
18282             5        1        1
18283             5        4        4
18287             4        2        3

[4338 rows x 3 columns]
```

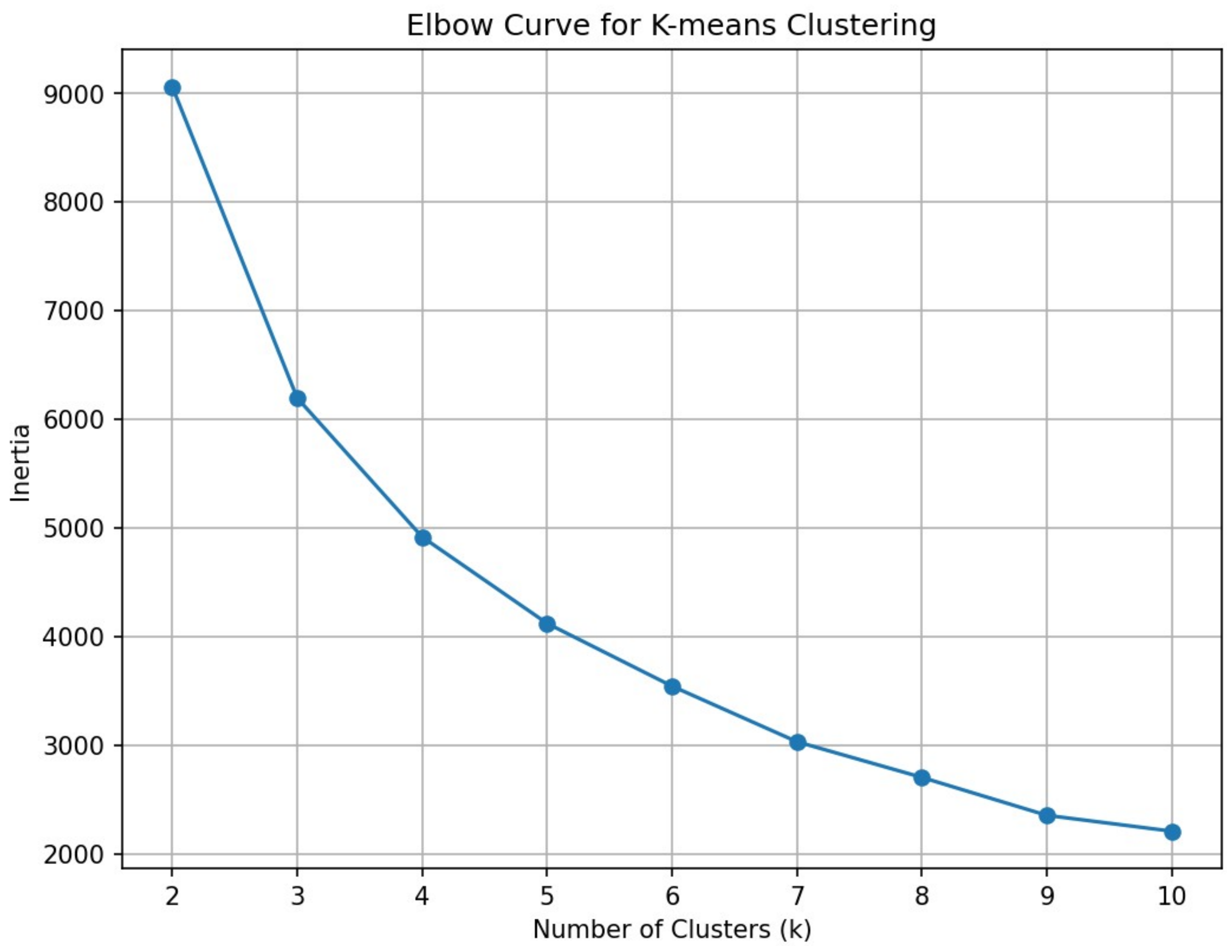
```python
# Calculate inertia (sum of squared distances) for different values of k
inertia = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, n_init= 10, random_state=42)
    a = kmeans.fit(X)
    print(a)
    inertia.append(kmeans.inertia_)

print(inertia)
```

```
KMeans(n_clusters=2, n_init=10, random_state=42)
KMeans(n_clusters=3, n_init=10, random_state=42)
KMeans(n_clusters=4, n_init=10, random_state=42)
```

```
KMeans(n_clusters=5, n_init=10, random_state=42)
KMeans(n_clusters=6, n_init=10, random_state=42)
KMeans(n_clusters=7, n_init=10, random_state=42)
KMeans(n_init=10, random_state=42)
KMeans(n_clusters=9, n_init=10, random_state=42)
KMeans(n_clusters=10, n_init=10, random_state=42)
[9060.029441570165, 6195.998861418406, 4917.61290110106,
4122.153761230396, 3544.720945367453, 3033.0935439885134,
2705.062048504891, 2355.7242555042594, 2210.1773251293666]

# Plot the elbow curve
plt.figure(figsize=(8, 6),dpi=150)
plt.plot(range(2, 11), inertia, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Curve for K-means Clustering')
plt.grid(True)
plt.show()
```



```python
# Perform K-means clustering with best K (which is 4)
best_kmeans = KMeans(n_clusters=4, n_init=10, random_state=42)
rfm['Cluster'] = best_kmeans.fit_predict(X)

print(rfm['Cluster'])
```

```
CustomerID
12346    1
12347    0
12348    1
12349    3
12350    2
        ..
18280    2
18281    2
18282    3
18283    0
18287    1
Name: Cluster, Length: 4338, dtype: int32
```

```python
#Step 7 – Interpret the Clusters to Identify Customer Segments

# Group by cluster and calculate mean values
cluster_summary = rfm.groupby('Cluster').agg({
    'R_Score': 'mean',
    'F_Score': 'mean',
    'M_Score': 'mean'
}).reset_index()

print(cluster_summary)
```

```
   Cluster   R_Score   F_Score   M_Score
0        0  4.669811  3.188679  3.764151
1        1  3.027290  1.893762  3.115984
2        2  1.442263  1.061201  1.505774
3        3  3.878194  1.083475  1.602215
```

```python
#let’s visualize the average R, F, and M scores for the clusters so
it’s easy to interpret

colors = ['#3498db', '#2ecc71', '#f39c12','#C9B1BD']

# Plot the average RFM scores for each cluster
plt.figure(figsize=(10, 8),dpi=150)

# Plot Avg Recency
plt.subplot(3, 1, 1)
bars = plt.bar(cluster_summary.index, cluster_summary['R_Score'],
color=colors)
plt.xlabel('Cluster')
plt.ylabel('Avg Recency')
```

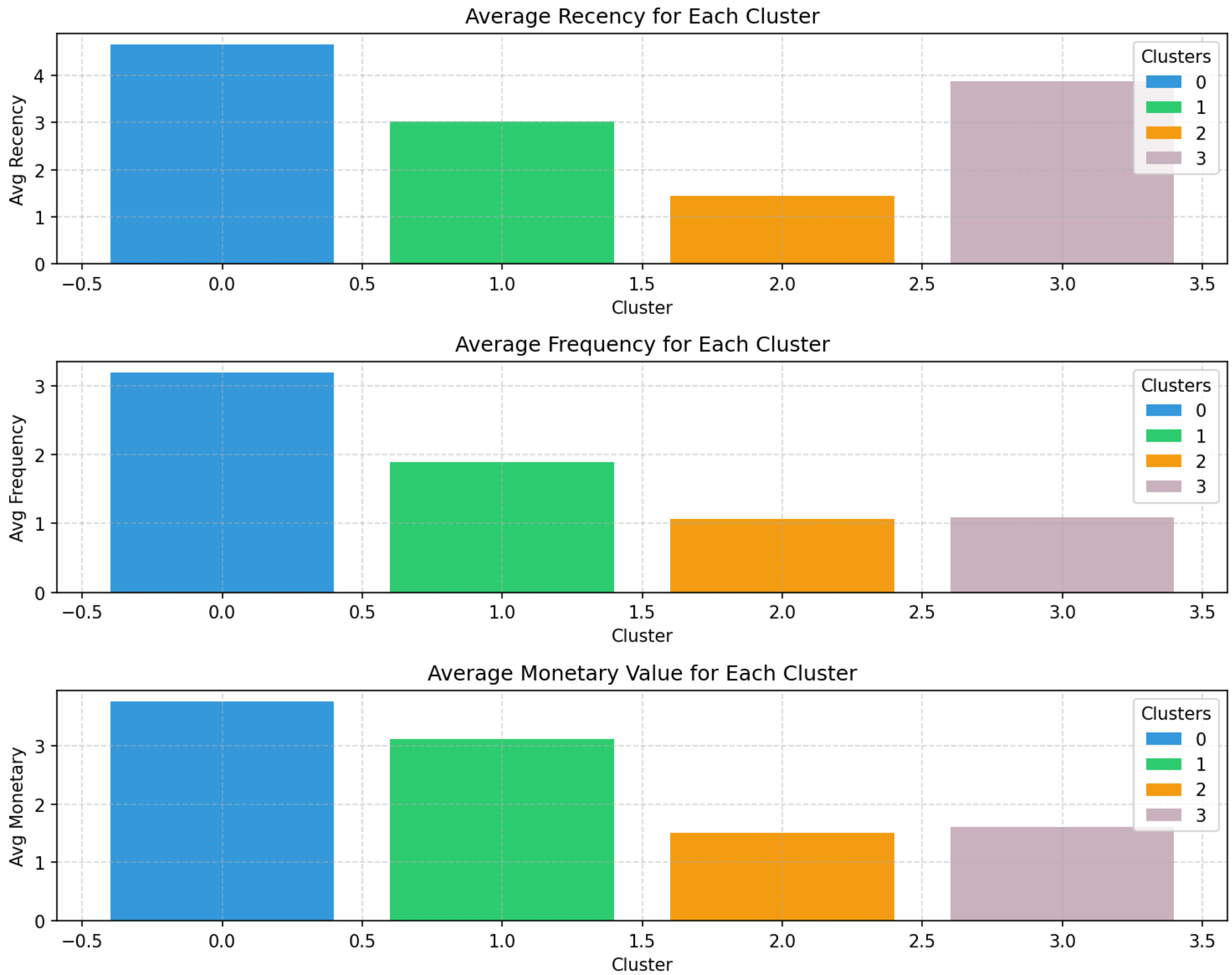
```
plt.title('Average Recency for Each Cluster')

plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(bars, cluster_summary.index, title='Clusters')

# Plot Avg Frequency
plt.subplot(3, 1, 2)
bars = plt.bar(cluster_summary.index, cluster_summary['F_Score'],
color=colors)
plt.xlabel('Cluster')
plt.ylabel('Avg Frequency')
plt.title('Average Frequency for Each Cluster')
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(bars, cluster_summary.index, title='Clusters')

# Plot Avg Monetary
plt.subplot(3, 1, 3)
bars = plt.bar(cluster_summary.index, cluster_summary['M_Score'],
color=colors)
plt.xlabel('Cluster')
plt.ylabel('Avg Monetary')
plt.title('Average Monetary Value for Each Cluster')
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(bars, cluster_summary.index, title='Clusters')

plt.tight_layout()
plt.show()
```




```
cluster_counts = rfm['Cluster'].value_counts()
print(cluster_counts)

colors = ['#3498db', '#2ecc71', '#f39c12','#C9B1BD']
# Calculate the total number of customers
total_customers = cluster_counts.sum()

# Calculate the percentage of customers in each cluster
percentage_customers = (cluster_counts / total_customers) * 100
labels = ['Champions(Power Shoppers)','Loyal Customers','At-risk
Customers','Recent Customers']

# Create a pie chart
plt.figure(figsize=(8, 8),dpi=200)
plt.pie(percentage_customers, labels=labels, autopct='%1.1f%%',
startangle=90, colors=colors)
plt.title('Percentage of Customers in Each Cluster')
plt.legend(cluster_summary['Cluster'], title='Cluster', loc='upper
left')
```

```
plt.show()

Cluster
0    1272
3    1174
1    1026
2     866
Name: count, dtype: int64
```