```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans

import warnings
warnings.filterwarnings('ignore')
```

```
data = pd.read_excel('/content/Online Retail.xlsx')
data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Coun
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Un Kingd
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingd
2	536365	84406B	CREAM CUPID HEARTS	8	2010-12-01 08:26:00	2.75	17850.0	Un Kingd

```
data.shape
```

```
(541909, 8)
```

```
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
```

```
data.describe().T
```

	count	mean	std	min	25%	50%	75%	
Quantity	541909.0	9.552250	218.081158	-80995.00	1.00	3.00	10.00	8
UnitPrice	541909.0	4.611114	96.759853	-11062.06	1.25	2.08	4.13	3
CustomerID	406829.0	15287.690570	1713.600303	12346.00	13953.00	15152.00	16791.00	1

```
data.describe()
```

	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

```
"""We see that the “CustomerID” column is currently a floating point value. When we clean the data, we’ll cast it into an integer:
Customer Segmentation in Python: A Practical Approach
Output of data.describe()
Also note that the dataset is quite noisy. The “Quantity” and “UnitPrice” columns contain negative values:"""
```

```
'We see that the “CustomerID” column is currently a floating point value. When we clean
the data, we’ll cast it into an integer:\nCustomer Segmentation in Python: A Practical
Approach\nOutput of data.describe()\nAlso note that the dataset is quite noisy. The “Qu
antity” and “UnitPrice” columns contain negative values:'
```

```
# 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
```

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

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

```
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
```

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

```
data['Total'] = data['Quantity'] * data['UnitPrice']
```

```
rfm = data.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
    'InvoiceNo': 'nunique',
```

```
    'Total': 'sum'
})

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

CustomerID	Recency	Frequency	MonetaryValue
12346	326	1	77183.60
12347	2	7	4310.00
12348	75	4	1797.24
12349	19	1	1757.55
12350	310	1	334.40

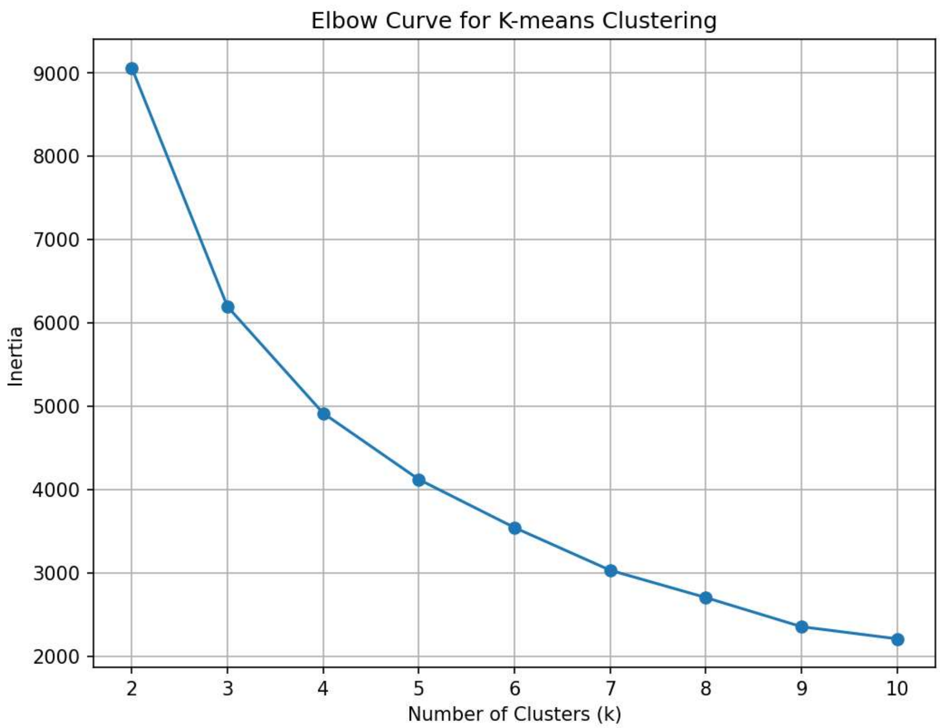
```
rfm.describe()
```

	Recency	Frequency	MonetaryValue
count	4338.000000	4338.000000	4338.000000
mean	92.536422	4.272015	2054.266460
std	100.014169	7.697998	8989.230441
min	1.000000	1.000000	3.750000
25%	18.000000	1.000000	307.415000
50%	51.000000	2.000000	674.485000
75%	142.000000	5.000000	1661.740000
max	374.000000	209.000000	280206.020000

```
# 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)


# 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
```

```
# Extract RFM scores for K-means clustering
X = rfm[['R_Score', 'F_Score', 'M_Score']]


# 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)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# 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()
```

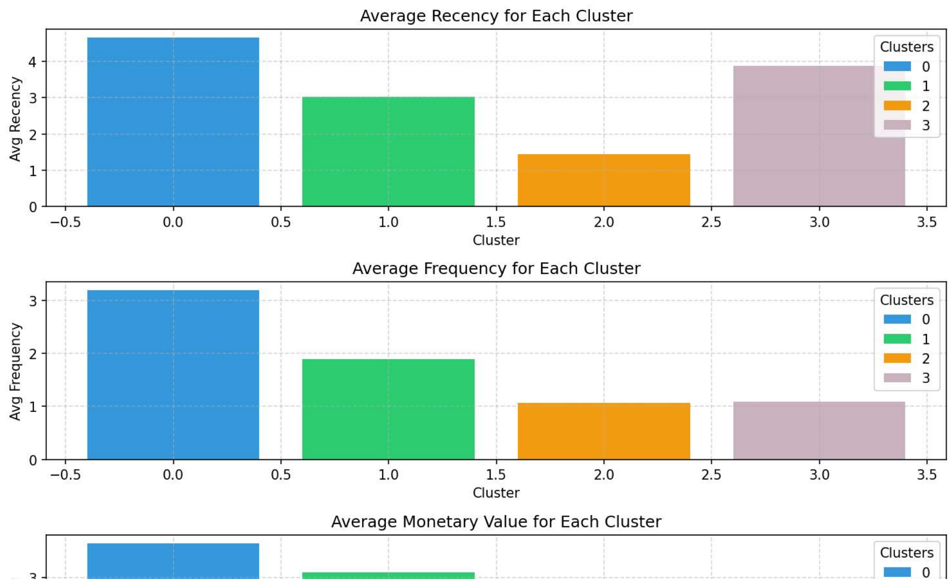



```
# Perform K-means clustering with best K
best_kmeans = KMeans(n_clusters=4, n_init=10, random_state=42)
rfm['Cluster'] = best_kmeans.fit_predict(X)


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

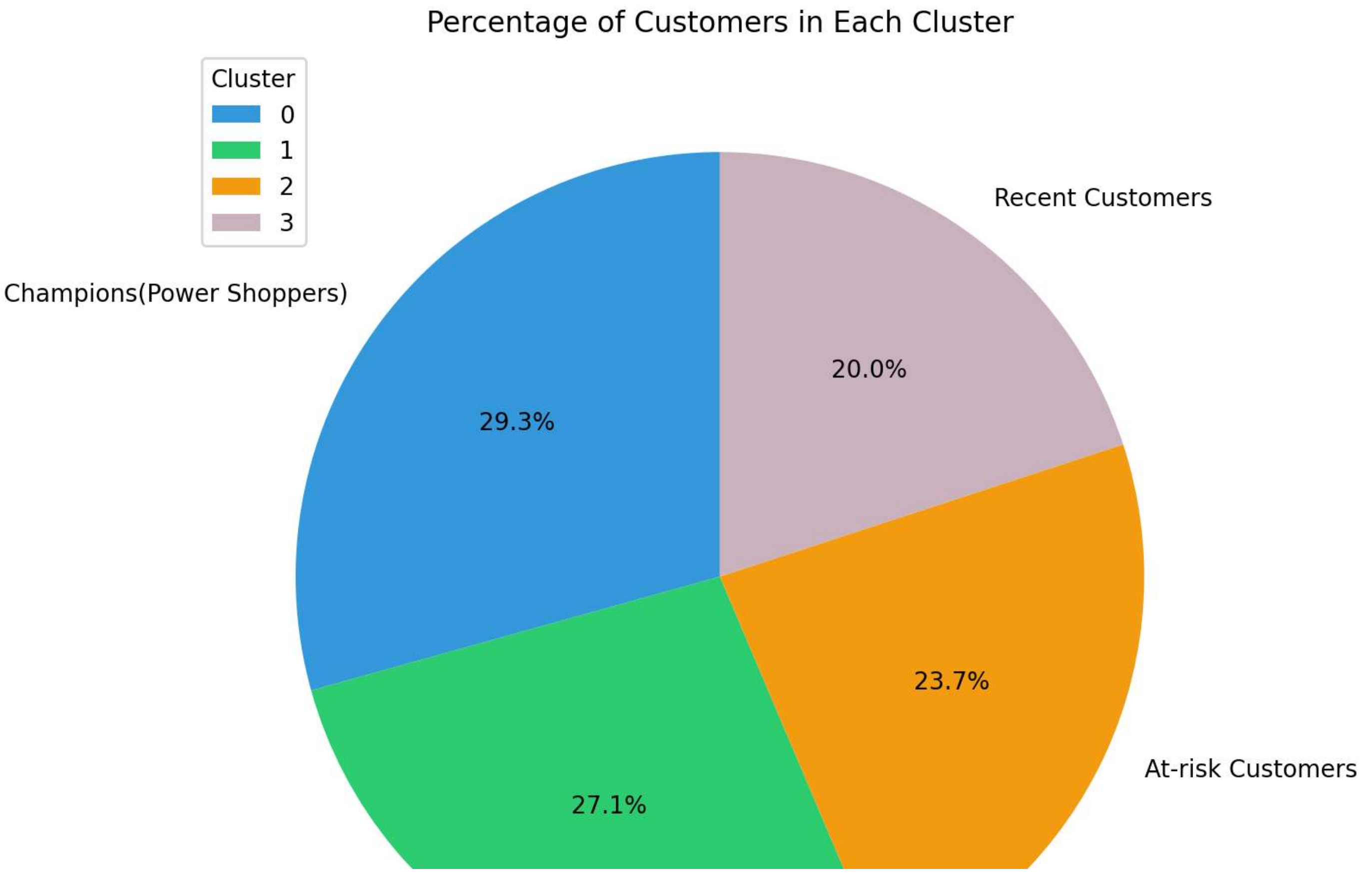
```
    'F_Score': 'mean',
    'M_Score': 'mean'
}).reset_index()


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
```

```
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()

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()
```


Percentage of Customers in Each Cluster
Cluster
0
1
2
3
Recent Customers
Champions(Power Shoppers)
20.0%
29.3%
23.7%
At-risk Customers
27.1%