Data Science Project: Customer Segmentation

Introduction

When marketing to customers, different groups have different unique messaging needs. One form of segmentation is RFM which stands for Recency, Frequency, Monetary. This means breaking up customers into clusters based on how recent their purchases took place, how many purchases they have made, and the monetary level of their purchases. With these clusters, you can customize the messages these customers receive when logged into their account, the promotions they receive via text or e-mail, and the effectiveness of types of messaging across subgroups i.e. A/B testing a promotion.

Methodology

The Data

The data provided contained the entirety of purchases and returns for an online retailer over a period of a little more than 1 year. Included are the Invoice Number, StockCode, Description, Invoice Date, Unit Price, Customer ID, and Country. The data was received in a CSV form, allowing for segregation and manipulation.

Elementry Method:

The data can be grouped via an RFM score by assigning a value via each customer's recency quartile, frequency quartile, and monetary quartile. In theory, this creates a ranking system of 3-12 (where 3 is the least ideal customer and 12 is the most ideal customer); however, this method does not account for the variances within a quartile and, therefore, isn't as precise as the clustering method detailed later. Recency is broken down by date from the last transaction, where the most recent transaction is in the fourth quartile and the first transaction is in the first quartile. Frequency is broken down by the sum of purchases by a given customer, where the most purchases are in the fourth quartile and the least purchases are in the first quartile. Finally, monetary value is assessed by separating the sum of total money spent by any given customer, where the highest paying customer is in the fourth quartile and the lowest paying is in the first quartile. With this in mind, independently each customer is assigned 1-4 in each category, creating 81 possible unique groupings, for 9 possible sums. This means you can target each of these groups with different possible promotions, deals, and marketing materials to target certain variables and objectives. Also, since this is formulaic, it can be updated real-time via an unsupervised algorithm, creating an updated list of RFM orders as new customers are added and as old customers make purchases and time passes. Further, the progress of certain objectives for a subgroup and the movement of a customer from one subgroup to another can be analyzed. This includes automation such as when a customer who has a 4 in recency drops to a 3, an email is triggered sending a promotion to urge a new purchase.

Advanced Method:

A more precise method for creating RFM customer segments with less variability is to use a K-Means Clustering or using Agglomerative Clustering (situation dependent).

Quantifying the Data

The data is quantified into values for RFM. The recency is quantified by time from the last transaction; frequency is quantified by the sum of invoices by a customer in the past year, and monetary value is quantified by the average sum of money spent per purchase by a customer. This allows for each category to be accurately interpreted and acted upon.

Data Cleansing

Firstly, with this method, it becomes important for the data to be more accurate and to clean it for outliers, incomplete data, and extraneous data (returns in this case). This was done by removing all negative quantities and any transactions missing values. The outliers in this case are still possible repeat customers and are therefore extremely important given their anomalously high RFM scores. Their data is a reason for the skewed distribution and therefore a log transformation was performed leading to a pseudo-normal distribution of the data which is more easily manipulated without adversely affecting the integrity of the data. Due to this reduced skewness and the approximately normal distribution, a log transformation was indeed the correct method (which is not necessarily guaranteed).

Elbow Method and Sillhouette Method for Number of Clusters

When performing a cluster analysis using k-means, you need to choose how many groups or segments you want to have in your analysis. Again, it is assessing 3 variables (RFM) with n possible values (1-4 was used in the elementary method). Two of the primary ways to analyze the best number of clusters are the elbow method and the silhouette method.

Note

The distance is calculated in 3D as a centroid so $\langle a,b,c \rangle$ is any given point and its distance from the center is assessed.

Elbow Method

The elbow method looks at the inertia and distortion of a distribution based on the number of clusters used. Distortion is the average Euclidean distance of the cluster data from the cluster center. Inertia is the sum of squared distances of the cluster data from the cluster center. The elbow method is used to find where the data changes from sharp decreases in inertia and distortion to a more linear decrease or even logarithmic. In other words, it is where the second derivative is increasing and therefore the first derivative is approaching 0 and the function is decreasing at an increasingly slow rate. This means that any additional cluster would be less meaningful because the precision is similar.

Results

No clear substantial change from sharp decrease to gradual decrease, so, not the best method. See graphs in the appendix.

Silhouette Method

The silhouette method for finding the appropriate number of clusters uses the distance within a cluster and compares the distance within the cluster to outside clusters. The distance of the cluster to its cluster center is its cohesion. The distance of the cluster to other clusters is its separation. This method gets a coefficient that is better quantified than the mostly heuristic elbow method {coefficeint = (cluster to cluster center mean - an average of cluster distance to other clusters) / the max of both}. Furthermore, a high silhouette coefficient means that the data is close together within its cluster and adequately separate from other clusters to make it unique and mostly homogenous, ideal for segmentation.

Results

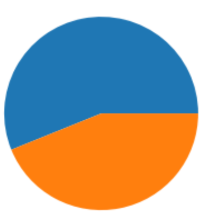
Clusters 2,3,4 have the highest silhouette coefficient (good) and after 4 the remaining coefficients approach approximately .27 compared to the .3 of 4. Further, it is highest at 2, then decreases slightly to 3, and increases again to 4 before more or less converging after 4. Therefore, we'll separate the customers into 2, 3, and 4 clusters– giving each cluster a label and specific goals. If/Then/Else statements can be used to decide which promotions a customer should receive depending on their group, creating a hierarchical relationship amongst the clusters allowing it to be more versatile and more precise.

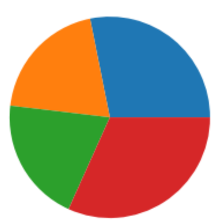
K-Means vs. Agglomerative Clustering

K-Means

The clusters for K at 2/3/4 are as follows– for 2, cluster 0: not recent, low frequency, low monetary value, and cluster 1: recent, high frequency, high monetary value; for 3, cluster 0: recent, high frequency, high monetary value, and cluster 1: middle recency, middle frequency, medium monetary value, and cluster 2: not recent, low frequency, low monetary value; for 4, cluster 0: less recent than average, more frequent than average, more monetary value than average, and cluster 1: most recent, most frequent, most monetary value, and cluster 2: more recent than average, lower frequency than average, lower monetary value than average, and cluster 3: least recent, least frequency, least monetary value.

Segments by Percentage





2 cluster pie chart:

1. Orange {2,435 customers, 56.12%, low RFM customers}
2. Blue {1,904 customers, 43.88%, high RFM customers}

3 cluster pie chart:

1. Blue {980 customers, 22.58%, high RFM customers}
2. Orange {1,844 customers, 42.5%, middle RFM customers}
3. Green {1,515 customers, 34.92%, low RFM customers}

4 cluster pie chart:

1. Blue {1,222 customers, 28.16%, less recent but higher frequency and value}
2. Green {867 customers, 19.98%, high RFM customers}
3. Orange {872 customers, 20.09%, more recent but lower frequency and value}
4. Red {1,378 customers, 31.76%, low RFM customers}

Cluster Comparison

In the appendix, Agglomerative Cluster Distributions and K-Mean Cluster Distribution, it is seen the K-Means groupings are both more distinct and tighter than the Agglomerative Groupings. Therefore K-Means groupings were used for further analysis.

Actionable Insights

These groupings should be utilized and kept under an unsupervised learning database that regularly updates records and can stratify customers based on the cluster but also other trigger events associated with their cluster.

2 K-Means

Goal: Convert Orange Customers to Blue Customers
Insights:

1. Offer rewards program to retain Blue Customers.
2. Use promotion to reengage Orange Customers.
3. After reengaged purchases from Orange Customers, funnel them into Blue through rewards/other incentives.

Notes: Use A/B testing of different funnel systems successes for conversion from re-engagement to maximize results. Use A/B testing for reengagement success from a given promotion.

3 K-Means

Goal: Convert Green to Orange to Blue (Funnel)
Insights:

1. Offer Rewards to retain Blue Customers
2. Send Rewards program to Orange Customers
3. Send Promotion to reengage Green Customers
4. After reengaged purchases by Green Customer, funnel them into Green and then Blue.

Notes: The biggest group present is Green which means that most customers purchase at a middle RFM. This means that the more customers you can funnel up, the more orders placed and revenue made. Furthermore, metrics include Green to Orange conversion, Orange to Blue conversion, and Blue retention.

4 K-Means

Goal: Convert Orange to Green, Retain Green, Reengage Blue, Reengage and Convert Red
Insights:

1. Offer Rewards to retain Green Customers
2. Offer Promotion to reengage Blue Customers who used to have higher frequency/monetary
3. Offer Rewards to make recent Orange Customers purchase more frequently/monetary
4. Offer Promotion to Red to reengage them and then offer ongoing rewards

Notes: This method caters to long-term customer retention and re-engagement of past high RFM customers as well as acquisition and conversion of new customers to high RFM customers.

RFM Scores

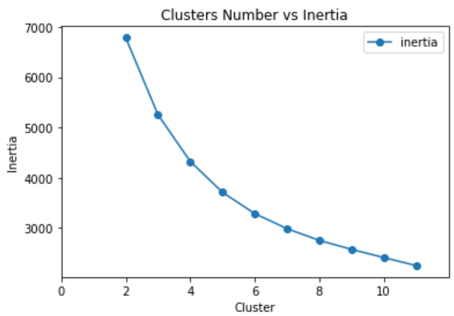
Provides insight into how customer's rank relative to other customers. Not useful as a KPI. However, changes in the box-and-whisker plot for raw RFM distributions does provide a useful KPI as purchases increase on average, monetary levels increase, and purchases become more recent. Therefore, RFM scores can augment the K-Means analysis.

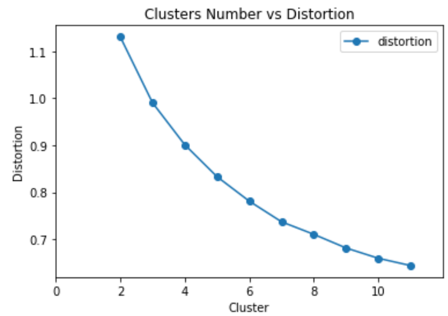
Unsupervised Machine Learning

The code can be run with respect to new purchases, updating the data set. In an unsupervised fashion, it can run through a K-Means cluster analysis, looking at silhouette scores to autonomously choose how many clusters should be made, and providing ongoing insights from the associated snake-plot values of each cluster.

Appendix

Elbow Graphs:





Silhouette Coefficients:

For n_clusters = 2 The average silhouette_score is : 0.39936610104262726

For n_clusters = 3 The average silhouette_score is : 0.30396507969419945

For n_clusters = 4 The average silhouette_score is : 0.3082848091892372

For n_clusters = 5 The average silhouette_score is : 0.27763953851542233

For n_clusters = 6 The average silhouette_score is : 0.2732791046313118

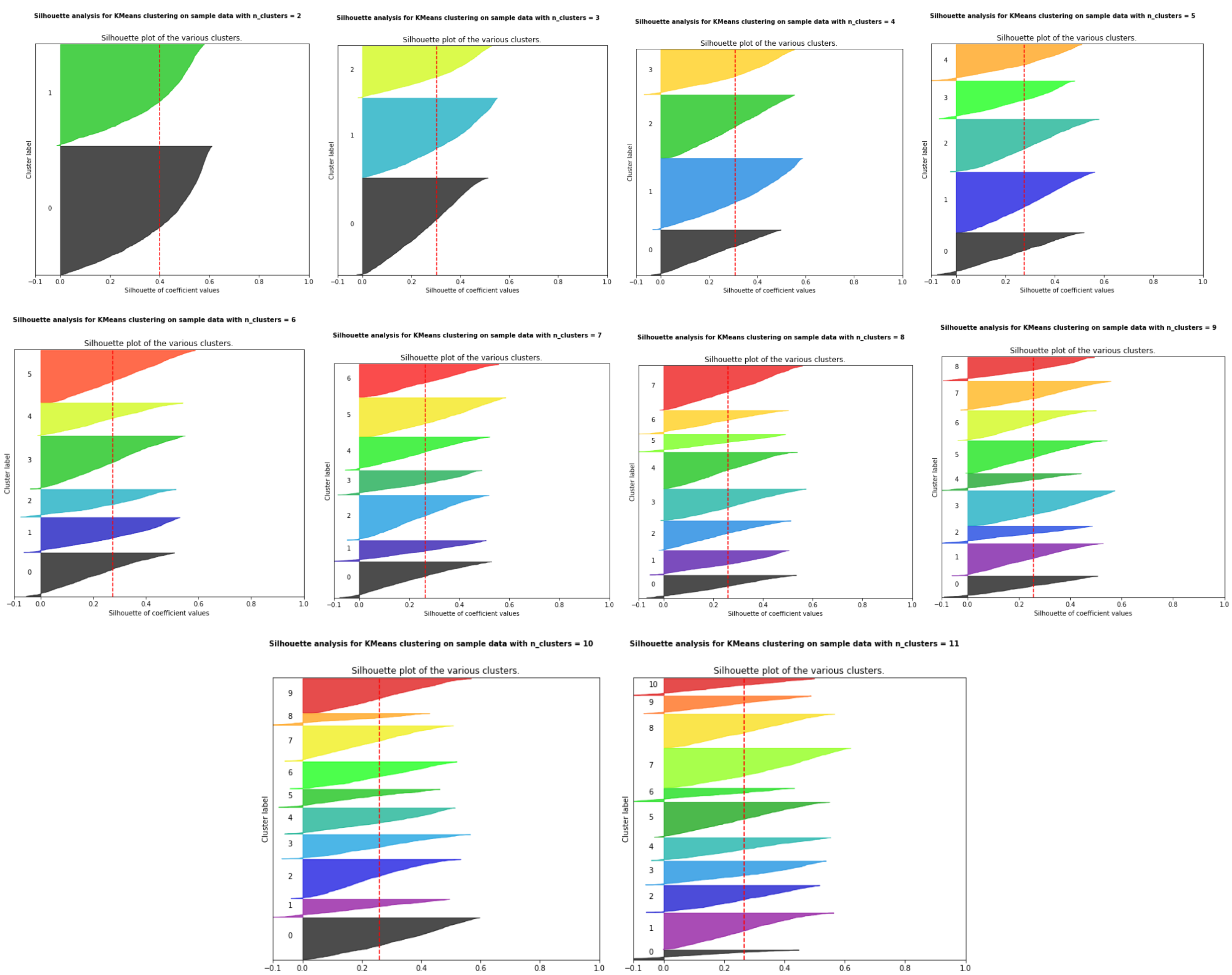
For n_clusters = 7 The average silhouette_score is : 0.2632621891834308

For n_clusters = 8 The average silhouette_score is : 0.25935201362686205

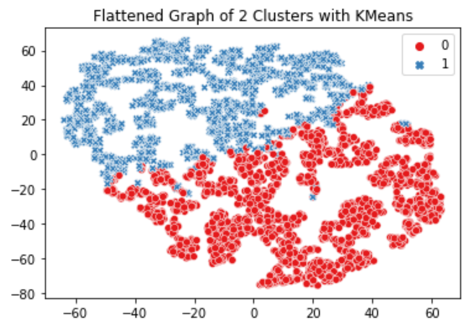
For n_clusters = 9 The average silhouette_score is : 0.2580392989523528

For n_clusters = 10 The average silhouette_score is : 0.2586789187842584

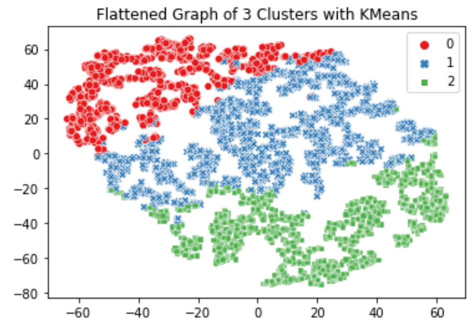
For n_clusters = 11 The average silhouette_score is : 0.26756216800799204



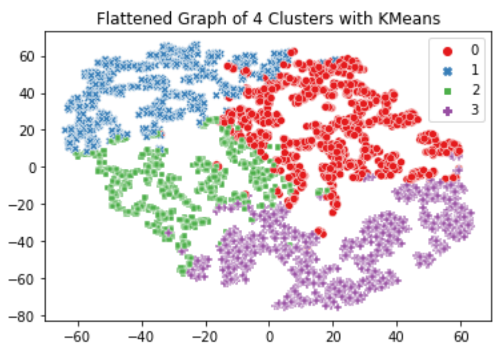
K-Mean Clusters Distributions:




```
Wall time: 22.3 s
```

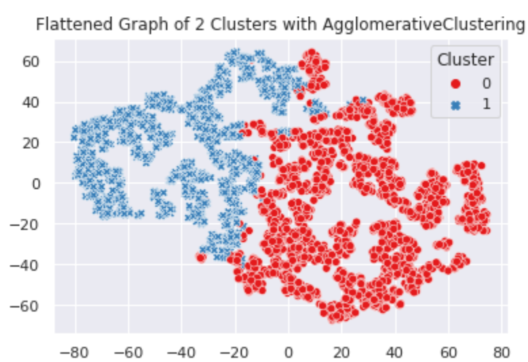


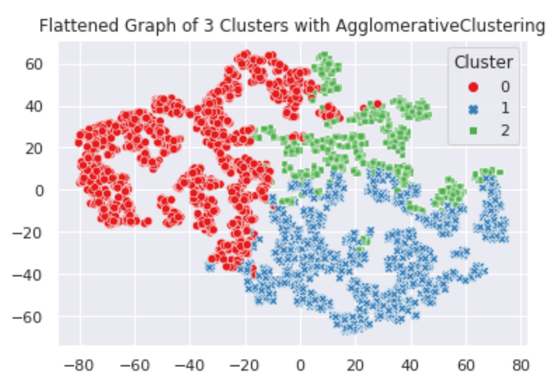

```
Wall time: 21.1 s
```

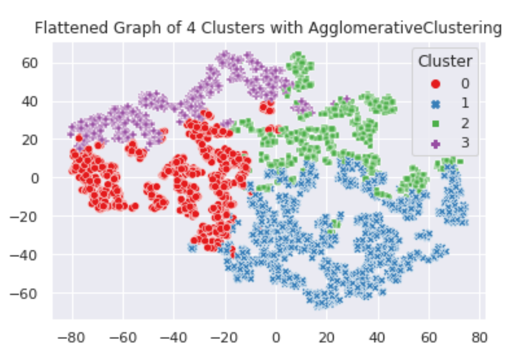



```
Wall time: 21.1 s
```

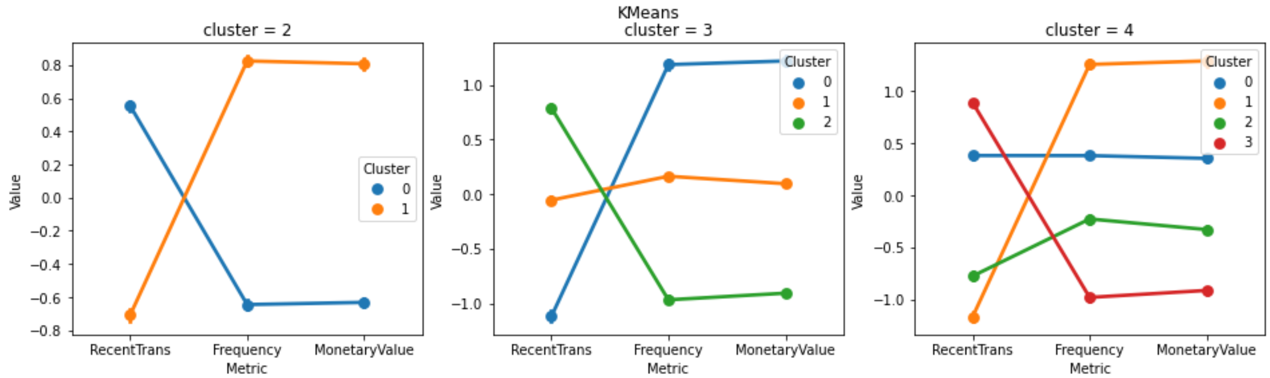
Agglomerative Clusters Distributions:







Snake Plots with Insights:




```
Wall time: 1.47 s
```

Descriptive Data:

COUNTUNIQUE of Description	COUNTUNIQUE of Country	COUNTUNIQUE of StockCode	COUNTUNIQUE of CustomerID
4223	38	4070	4372

Out[58]:

	RecentTrans	Frequency	MonetaryValue
count	4339.0000	4339.0000	4339.0000
mean	-0.0000	-0.0000	0.0000
std	1.0001	1.0001	1.0001
min	-2.3412	-2.4328	-5.2266
25%	-0.6612	-0.6764	-0.6828
50%	0.0901	0.0009	-0.0608
75%	0.8450	0.7023	0.6534
max	1.5644	4.1818	4.7184

Out[2]:

	Quantity	UnitPrice
count	541909.000000	541909.000000
mean	9.552250	4.611114
std	218.081158	96.759853
min	-80995.000000	-11062.060000
25%	1.000000	1.250000
50%	3.000000	2.080000
75%	10.000000	4.130000
max	80995.000000	38970.000000

Code:

In [19]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import datetime as dt
```

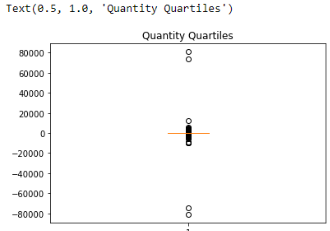
In [2]:
```python
customers.describe()
```

Out[2]:

	Quantity	UnitPrice
count	541909.000000	541909.000000
mean	9.552250	4.611114
std	218.081158	96.759853
min	-80995.000000	-11062.060000
25%	1.000000	1.250000
50%	3.000000	2.080000
75%	10.000000	4.130000
max	80995.000000	38970.000000

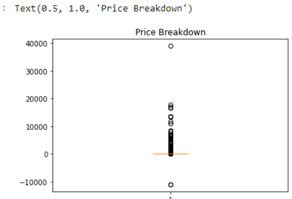
In [7]:
```python
plt.boxplot(data["Quantity"])
plt.title("Quantity Quartiles")
```

Out[7]: `Text(0.5, 1.0, 'Quantity Quartiles')`



In [20]:
```python
plt.boxplot(data["UnitPrice"])
plt.title("Price Breakdown")
```

Out[20]: `Text(0.5, 1.0, 'Price Breakdown')`



In [25]:
```
data= data[pd.notnull(data['InvoiceNo'])]
data= data[pd.notnull(data['StockCode'])]
data= data[pd.notnull(data['Description'])]
data= data[pd.notnull(data['Quantity'])]
data= data[pd.notnull(data['InvoiceDate'])]
data= data[pd.notnull(data['UnitPrice'])]
data= data[pd.notnull(data['CustomerID'])]
data= data[pd.notnull(data['Country'])]
```

In [26]:
```
data['CustomerID'] = data['CustomerID'].astype(int)
data = data[(data['Quantity']>0)]
```

In [27]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 397924 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    397924 non-null  object
 1   StockCode    397924 non-null  object
 2   Description  397924 non-null  object
 3   Quantity     397924 non-null  int64
 4   InvoiceDate  397924 non-null  object
 5   UnitPrice    397924 non-null  float64
 6   CustomerID   397924 non-null  int32
 7   Country      397924 non-null  object
dtypes: float64(1), int32(1), int64(1), object(5)
memory usage: 25.8+ MB
```

In [28]:
```
data=data[['CustomerID','InvoiceDate','InvoiceNo','Quantity','UnitPrice']]
```

In [29]:
```
data['TotalPrice'] = data['Quantity'] * data['UnitPrice']
```

In [31]:
```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
data['InvoiceDate'].min(),data['InvoiceDate'].max()
```

Out[31]:
```
(Timestamp('2010-12-01 08:26:00'), Timestamp('2011-12-09 12:50:00'))
```

In [32]:
```
PRESENT = dt.datetime(2011,12,10)
```

In [33]:
```
rfm= data.groupby('CustomerID').agg({'InvoiceDate': lambda date: (PRESENT - date.max()).days,
                                        'InvoiceNo': lambda num: len(num),
                                        'TotalPrice': lambda price: price.sum()})
```

In [34]:
```
rfm.columns
```

Out[34]:
```
Index(['InvoiceDate', 'InvoiceNo', 'TotalPrice'], dtype='object')
```

In [35]:
```
rfm.columns=['recency','frequency','monetary']
```

In [36]:
```
rfm['recency'] = rfm['recency'].astype(int)
```

In [37]:
```
rfm.head()
```

Out[37]:

CustomerID	recency	frequency	monetary
12346	325	1	77183.60
12347	2	182	4310.00
12348	75	31	1797.24
12349	18	73	1757.55
12350	310	17	334.40

In [38]:
```
rfm['r_quartile'] = pd.qcut(rfm['recency'], 4, ['1','2','3','4'])
rfm['f_quartile'] = pd.qcut(rfm['frequency'], 4, ['4','3','2','1'])
rfm['m_quartile'] = pd.qcut(rfm['monetary'], 4, ['4','3','2','1'])
```

In [39]:
```
rfm.head()
```

Out[39]:

CustomerID	recency	frequency	monetary	r_quartile	f_quartile	m_quartile
12346	325	1	77183.60	4	4	1
12347	2	182	4310.00	1	1	1
12348	75	31	1797.24	3	3	1
12349	18	73	1757.55	2	2	1
12350	310	17	334.40	4	4	3

In [40]:
```
rfm['RFM_Segment_Concat'] = rfm.r_quartile.astype(str)+ rfm.f_quartile.astype(str) + rfm.m_quartile.astype(str)
rfm['RFM_Score'] = rfm.r_quartile.astype(int)+rfm.f_quartile.astype(int)+rfm.m_quartile.astype(int)
```

In [41]:
```
rfm
```

Out[41]:

CustomerID	recency	frequency	monetary	r_quartile	f_quartile	m_quartile	RFM_Segment_Concat	RFM_Score
12346	325	1	77183.60	4	4	1	441	9
12347	2	182	4310.00	1	1	1	111	3
12348	75	31	1797.24	3	3	1	331	7
12349	18	73	1757.55	2	2	1	221	5
12350	310	17	334.40	4	4	3	443	11
...	...	...	...	...	...	...	...	...
18280	277	10	180.60	4	4	4	444	12
18281	180	7	80.82	4	4	4	444	12
18282	7	12	178.05	1	4	4	144	9
18283	3	756	2094.88	1	1	1	111	3
18287	42	70	1837.28	2	2	1	221	5

4339 rows × 8 columns

In [42]:
```
rfm[rfm['RFM_Score']==3].sort_values('monetary', ascending=False).head()
```

Out[42]:

CustomerID	recency	frequency	monetary	r_quartile	f_quartile	m_quartile	RFM_Segment_Concat	RFM_Score
14646	1	2080	280206.02	1	1	1	111	3
18102	0	431	259657.30	1	1	1	111	3
17450	8	337	194550.79	1	1	1	111	3
14911	1	5677	143825.06	1	1	1	111	3
14156	9	1400	117379.63	1	1	1	111	3

In [43]:
```
rfm.sort_values('RFM_Score', ascending=True)
```

Out[43]:

CustomerID	recency	frequency	monetary	r_quartile	f_quartile	m_quartile	RFM_Segment_Concat	RFM_Score
16592	4	216	4113.68	1	1	1	111	3
14110	3	156	5683.15	1	1	1	111	3
14121	3	159	2780.15	1	1	1	111	3
14125	10	167	2740.43	1	1	1	111	3
14132	2	200	3586.03	1	1	1	111	3
...	...	...	...	...	...	...	...	...
12837	173	12	134.10	4	4	4	444	12
16498	161	15	100.97	4	4	4	444	12
17245	204	9	171.45	4	4	4	444	12
15083	256	5	88.20	4	4	4	444	12
16849	186	8	124.57	4	4	4	444	12

4339 rows × 8 columns

In [46]:
```
from datetime import timedelta
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, silhouette_samples
from scipy.spatial.distance import cdist
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
```

In [47]:
```
def check_values(df):
    col_desc = []
    data = {
        'features': [col for col in df.columns],
        'data_type': [df[col].dtype for col in df.columns],
        'nan_total': [df[col].isna().sum() for col in df.columns],
        'nan_pct': [round(df[col].isna().sum()/len(df)*100,2) for col in df.columns],
        'unique': [df[col].nunique() for col in df.columns],
        'values_ex': [df[col].drop_duplicates().sample(df[col].nunique()).values if df[col].nunique() <= 5 else df[col].drop_dupl
    }
    return pd.DataFrame(data)
```

In [48]:
```
%%time
check_values(data)
```

```
Wall time: 290 ms
```

Out[48]:

	features	data_type	nan_total	nan_pct	unique	values_ex
0	CustomerID	int32	0	0.0	4339	[14522, 16913]
1	InvoiceDate	datetime64[ns]	0	0.0	17286	[2011-11-24T10:39:00.000000000, 2011-03-28T12:...
2	InvoiceNo	object	0	0.0	18536	[559153, 580137]
3	Quantity	int64	0	0.0	302	[35, 3]
4	UnitPrice	float64	0	0.0	441	[5.55, 0.62]
5	TotalPrice	float64	0	0.0	2940	[7.03, 537.6]

In [49]:
```
data[data.InvoiceNo.str.startswith('C')]
```

Out[49]:

CustomerID	InvoiceDate	InvoiceNo	Quantity	UnitPrice	TotalPrice

In [50]:
```
%%time
df_clean = data.dropna(subset=['CustomerID'])
cancelled = df_clean[df_clean.InvoiceNo.str.startswith('C')].index
df_clean = df_clean.drop(index=cancelled)
df_clean.loc[:,'Date'] = pd.to_datetime(df_clean['InvoiceDate'])
df_clean.loc[:,'TotalSum'] = df_clean['Quantity'] * df_clean['UnitPrice']
df_clean[['Quantity', 'UnitPrice', 'TotalSum']].head()
```

```
Wall time: 360 ms
```

Out[50]:

	Quantity	UnitPrice	TotalSum
0	6	2.55	15.30
1	6	3.39	20.34
2	8	2.75	22.00
3	6	3.39	20.34
4	6	3.39	20.34

In [51]:
```
print('df_clean length:',len(df_clean))
```

```
df_clean length: 397924
```

In [52]:
```
%%time
check_values(df_clean)
```

Wall time: 228 ms

Out[52]:

	features	data_type	nan_total	nan_pct	unique	values_ex
0	CustomerID	int32	0	0.0	4339	[12864, 12610]
1	InvoiceDate	datetime64[ns]	0	0.0	17286	[2011-02-07T16:21:00.000000000, 2011-09-23T11:...
2	InvoiceNo	object	0	0.0	18536	[572932, 561858]
3	Quantity	int64	0	0.0	302	[388, 670]
4	UnitPrice	float64	0	0.0	441	[0.85, 5.32]
5	TotalPrice	float64	0	0.0	2940	[21.189999999999998, 9.5]
6	Date	datetime64[ns]	0	0.0	17286	[2011-06-08T08:12:00.000000000, 2011-11-14T13:...
7	TotalSum	float64	0	0.0	2940	[2748.0, 809.76]

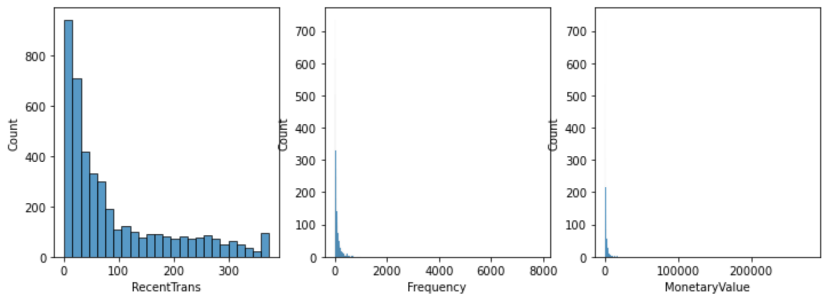
In [53]:
```
record_date = df_clean.Date.max() + timedelta(days=1)

# Calculate Recency, Frequency and Monetary value for each customer
df_rfm = df_clean.groupby(['CustomerID']).agg({
    'Date': lambda x: (record_date - x.max()).days,
    'InvoiceNo': 'count',
    'TotalSum': 'sum'})

df_rfm.rename(columns={
    'Date':'RecentTrans',
    'InvoiceNo':'Frequency',
    'TotalSum':'MonetaryValue'}, inplace=True)

df_rfm
```

Out[53]:

CustomerID	RecentTrans	Frequency	MonetaryValue
12346	326	1	77183.60
12347	2	182	4310.00
12348	75	31	1797.24
12349	19	73	1757.55
12350	310	17	334.40
...	...	...	...
18280	278	10	180.60
18281	181	7	80.82
18282	8	12	178.05
18283	4	756	2094.88
18287	43	70	1837.28

4339 rows × 3 columns

In [54]:
```python
%%time
fig, ax = plt.subplots(1,3, figsize=(12,4))
for i, col in enumerate(df_rfm.columns):
    sns.histplot(df_rfm[col], ax=ax[i])
plt.show()
```

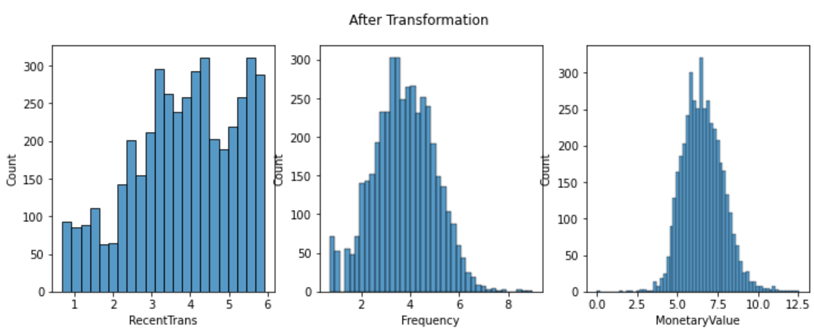



```
Wall time: 3.55 s
```

In [55]:
```python
skew_columns = (df_rfm.skew().sort_values(ascending=False))

skew_columns = skew_columns.loc[skew_columns > 0.75]
skew_columns
```

Out[55]:
```
MonetaryValue    19.326985
Frequency        18.106243
RecentTrans       1.246357
dtype: float64
```

In [56]:
```python
%%time
df_transf = df_rfm.copy()
for col in skew_columns.index.tolist():
    df_transf[col] = np.log1p(df_transf[col])


fig, ax = plt.subplots(1,3, figsize=(12,4))
for i, col in enumerate(df_transf.columns):
    sns.histplot(df_transf[col], ax=ax[i])
plt.suptitle('After Transformation')
plt.show()
```



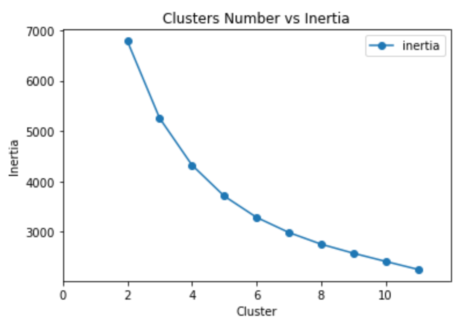

```
Wall time: 464 ms
```

In [58]:
```python
scaler = StandardScaler()
df_train = df_transf.copy()
for col in df_transf.columns:
    df_train[col] = scaler.fit_transform(df_train[[col]])
df_train.describe().round(4)
```

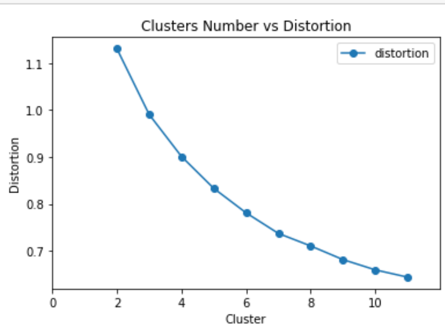
Out[58]:

	RecentTrans	Frequency	MonetaryValue
count	4339.0000	4339.0000	4339.0000
mean	-0.0000	-0.0000	0.0000
std	1.0001	1.0001	1.0001
min	-2.3412	-2.4328	-5.2266
25%	-0.6612	-0.6764	-0.6828
50%	0.0901	0.0009	-0.0608
75%	0.8450	0.7023	0.6534
max	1.5644	4.1818	4.7184

In [59]:
```
### BEGIN SOLUTION
# Create and fit a range of models
km_list = []

for clust in range(2,12):
    km = KMeans(n_clusters=clust, random_state=26)
    km = km.fit(df_train)

    km_list.append(pd.Series({'clusters': clust,
                              'inertia': km.inertia_,
                              'model': km}))

plot_data = (pd.concat(km_list, axis=1)
             .T
             [['clusters','inertia']]
             .set_index('clusters'))

ax = plot_data.plot(marker='o',ls='-')
ax.set_xticks(range(0,12,2))
ax.set_xlim(0,12)
ax.set_title('Clusters Number vs Inertia')
ax.set(xlabel='Cluster', ylabel='Inertia');
### END SOLUTION
```



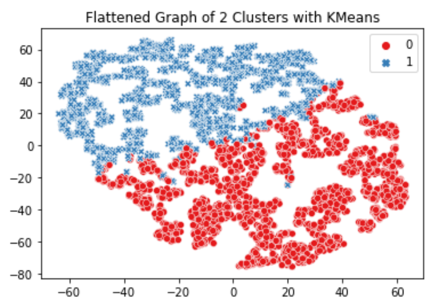
In [60]:
```
### BEGIN SOLUTION
# Create and fit a range of models
km_list = []

for clust in range(2,12):
    km = KMeans(n_clusters=clust, random_state=26)
    km = km.fit(df_train)

    km_list.append(pd.Series({'clusters': clust,
                              'distortion': sum(np.min(cdist(df_train, km.cluster_centers_,
                                          'euclidean'), axis=1)) / df_train.shape[0],
                              'model': km}))

plot_data = (pd.concat(km_list, axis=1)
             .T
             [['clusters','distortion']]
             .set_index('clusters'))

ax = plot_data.plot(marker='o',ls='-')
ax.set_xticks(range(0,12,2))
ax.set_xlim(0,12)
ax.set_title('Clusters Number vs Distortion')
ax.set(xlabel='Cluster', ylabel='Distortion');
### END SOLUTION
```



In [110]:
```
def kmeans(df, clusters_number):
    '''
    Implement k-means clustering on dataset

    INPUT:
        dataset : dataframe. Dataset for k-means to fit.
        clusters_number : int. Number of clusters to form.
        end : int. Ending range of kmeans to test.
    OUTPUT:
        Cluster results and t-SNE visualisation of clusters.
    '''

    kmeans = KMeans(n_clusters = clusters_number, random_state = 1)
    kmeans.fit(df)

    cluster_labels = kmeans.labels_

    df_new = df.assign(Cluster = cluster_labels)

    model = TSNE(random_state=1)
    transformed = model.fit_transform(df)

    plt.title('Flattened Graph of {} Clusters'.format(clusters_number))
    sns.scatterplot(x=transformed[:,0], y=transformed[:,1], hue=cluster_labels, style=cluster_labels, palette="Set1")

    return df_new, cluster_labels
```

In [112]:
```
def plot_ag(df, clusters_number, df_agg):

    cluster_labels = df_agg['Cluster']

    model = TSNE(random_state=1)
    transformed = model.fit_transform(df)

    plt.title('Flattened Graph of {} Clusters'.format(clusters_number))
    sns.scatterplot(x=transformed[:,0], y=transformed[:,1], hue=cluster_labels, style=cluster)
```

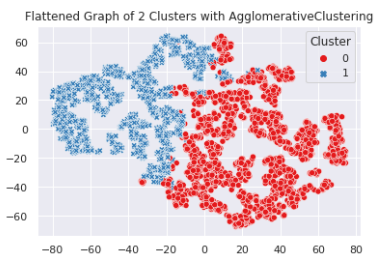
In [113]:
```
%%time
df_km_2, labels_2 = kmeans(df_train, 2)
df_label_2 = df_rfm.assign(Cluster = labels_2)
plt.title('Flattened Graph of 2 Clusters with KMeans')
plt.show()
```



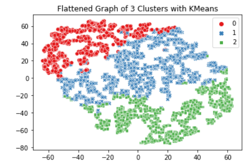

```
Wall time: 22.3 s
```

In [125]:
```
ag = AgglomerativeClustering(n_clusters=2, linkage='ward')
ag = ag.fit(df_train)
cluster_labels = ag.fit_predict(df_train)
df_agg_2 = df_train.assign(Cluster = cluster_labels)
```

In [126]:
```
plot_ag(df_train, clusters_number=2, df_agg=df_agg_2)
plt.title('Flattened Graph of 2 Clusters with AgglomerativeClustering')
plt.show()
```



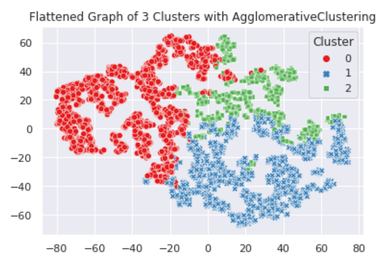

```
In [127]: %%time
df_km_3, labels_3 = kmeans(df_train, 3)
df_label_3 = df_rfm.assign(Cluster = labels_3)
plt.title('Flattened Graph of 3 Clusters with KMeans')
plt.show()
```



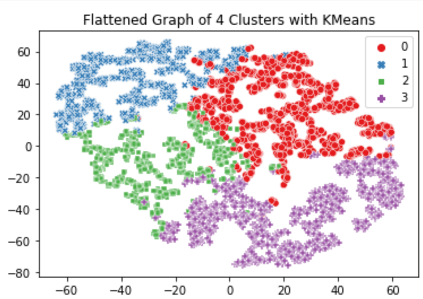

```
Wall time: 21.1 s
```

```
In [128]: ag = AgglomerativeClustering(n_clusters=3, linkage='ward')
ag = ag.fit(df_train)
cluster_labels = ag.fit_predict(df_train)
df_agg_3 = df_train.assign(Cluster = cluster_labels)
```

```
In [129]: plot_ag(df_train, clusters_number=3, df_agg=df_agg_3)
plt.title('Flattened Graph of 3 Clusters with AgglomerativeClustering')
plt.show()
```



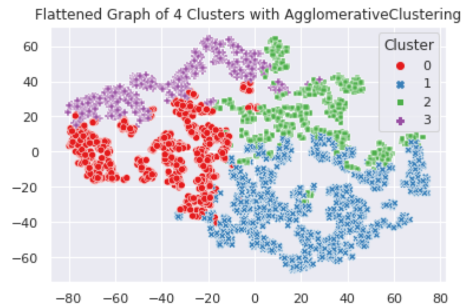

```
In [130]: %%time
df_km_4, labels_4 = kmeans(df_train, 4)
df_label_4 = df_rfm.assign(Cluster = labels_4)
plt.title('Flattened Graph of 4 Clusters with KMeans')
plt.show()
```




```
Wall time: 21.1 s
```

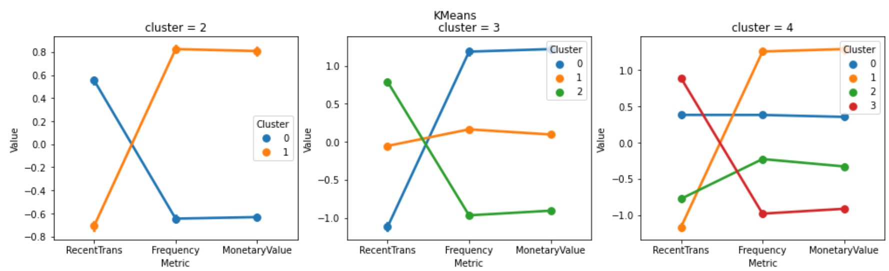
```
In [131]: ag = AgglomerativeClustering(n_clusters=4, linkage='ward')
ag = ag.fit(df_train)
cluster_labels = ag.fit_predict(df_train)
df_agg_4 = df_train.assign(Cluster = cluster_labels)
```

```
In [132]: %%time
plot_ag(df_train, clusters_number=4, df_agg=df_agg_4)
plt.title('Flattened Graph of 4 Clusters with AgglomerativeClustering')
plt.show()
```



In [133]:
```
def snake_plot(normalised_df_rfm, df_rfm_kmeans, df_rfm_original=df_rfm):
    '''
    Transform dataframe and plot snakeplot
    '''
    normalised_df_rfm = pd.DataFrame(normalised_df_rfm,
                                      index=df_rfm_original.index,
                                      columns=df_rfm_original.columns)
    normalised_df_rfm['Cluster'] = df_rfm_kmeans['Cluster']

    df_melt = pd.melt(normalised_df_rfm.reset_index(),
                      id_vars=['CustomerID', 'Cluster'],
                      value_vars=['RecentTrans', 'Frequency', 'MonetaryValue'],
                      var_name='Metric',
                      value_name='Value')

    plt.xlabel('Metric')
    plt.ylabel('Value')

    return sns.pointplot(data=df_melt, x='Metric', y='Value', hue='Cluster')
```

In [134]:
```
%%time
fig, ax = plt.subplots(1,3, figsize=(16,4))
plt.subplot(1,3,1)
ax[0]=snake_plot(df_train, df_km_2)
ax[0].set_title('cluster = 2')
plt.subplot(1,3,2)
ax[1]=snake_plot(df_train, df_km_3)
ax[1].set_title('cluster = 3')
plt.subplot(1,3,3)
ax[2]=snake_plot(df_train, df_km_4)
ax[2].set_title('cluster = 4')
plt.suptitle('KMeans')
plt.show()
```



Wall time: 1.47 s

In [135]:
```
def rfm_values(df):
    '''
    Calcualte average RFM values and size for each cluster

    '''
    df_new = df.groupby(['Cluster']).agg({
        'RecentTrans': 'mean',
        'Frequency': 'mean',
        'MonetaryValue': ['mean', 'count']
    }).round(0)

    return df_new
```

In [136]:
```
rfm_values(df_label_3)
```

Out[136]:

	RecentTrans	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	15.0	265.0	6502.0	980
1	65.0	62.0	1129.0	1844
2	176.0	15.0	302.0	1515

In [65]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([980, 1844, 1515])

plt.pie(y)
plt.show()
```

In [66]:
```
rfm_values(df_label_2)
```

Out[66]:

	RecentTrans	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	141.0	25.0	479.0	2435
1	31.0	177.0	4067.0	1904

In [67]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([2435,1904])

plt.pie(y)
plt.show()
```

In [68]:
```
rfm_values(df_label_4)
```

Out[68]:

	RecentTrans	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	96.0	80.0	1522.0	1222
1	13.0	283.0	7043.0	867
2	20.0	39.0	612.0	872
3	185.0	15.0	298.0	1378

In [69]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([1222,867,872,1378])

plt.pie(y)
plt.show()
```