Course:ARTIFICIALINTELLIGENC
ETITLE: MARKET BASKET INSIGHTS
PHASE 5 SUBMISSION

TEAM MEMBERS:

MOHANA PRIYA.M

(au311521106061)

MOUNICKA G S

(au31152110602)

PADMA PRIYA .M

(au311521106069)

YAMINI.H

(au 311521106114)

INTRODUCTION:

In within data, ultimately leading to enhanced insights and more effective strategies.today's data-driven world, extracting valuable insights from vast datasets is a fundamental aspect of making informed decisions in various domains, such as retail,marketing, healthcare, and more. Advanced association analysis techniques and visualization tools play a pivotal role in uncovering hidden patterns, relationships, and trends

The exact tools and libraries you use for these tasks may vary depending on your preference and the size of your dataset. Common choices include Python with libraries like pandas, scikit-learn, and mlxtend for data preprocessing and association rule mining. For larger datasets, you might want to use distributed data processing frameworks like Apache Spark.

Remember that market basket analysis is a valuable technique for understanding customer behavior and making data-driven business decisions, such as optimizing product placements, running targeted promotions, and improving inventory management.

Market Basket Analysis (MBA) is a powerful data mining technique that provides invaluable
insights into customer purchase behavior. In today's competitive business landscape,
understanding what products or services customers buy together can be a game-changer forbusinesses of all sizes. This analysis uncovers hidden patterns and correlations within
transaction data, enabling businesses to make informed decisions to enhance customer
experiences and boost profitability.
By examining the items frequently purchased together, businesses can cross-sell and upsell

effectively, optimize inventory management, and personalize marketing strategies. MBA not only
helps identify popular product combinations but also reveals the impact of promotions, product
placement, and pricing strategies on customer choices. The insights gained from market basket
analysis can drive recommendations, dynamic pricing, and targeted marketing campaigns,
ultimately improving customer satisfaction and driving revenue.
In this age of data-driven decision-making, understanding customer purchase behavior through
market basket insights is a vital tool for businesses looking to stay competitive, make data-
informed choices, and better cater to the needs and preferences of their customers. This
introduction sets the stage for a deeper exploration of the methodologies and benefits associated
with market basket analysis.

for market Market Basket Analysis (MBA) is a powerful data mining technique that provides invaluable
insights into customer purchase behavior. In today's competitive business landscape,
understanding what products or services customers buy together can be a game-changer for
businesses of all sizes. This analysis uncovers hidden patterns and correlations within
transaction data, enabling businesses to make informed decisions to enhance customer
experiences and boost profitability.
By examining the items frequently purchased together, businesses can cross-sell and upsell
effectively, optimize inventory management, and personalize marketing strategies. MBA not only

helps identify popular product combinations but also reveals the impact of promotions, product
placement, and pricing strategies on customer choices. The insights gained from market basket
analysis can drive recommendations, dynamic pricing, and targeted marketing campaigns,
ultimately improving customer satisfaction and driving revenue.
In this age of data-driven decision-making, understanding customer purchase behavior through
market basket insights is a vital tool for businesses looking to stay competitive, make data-
informed choices, and better cater to the needs and preferences of their customers. This
introduction sets the stage for a deeper exploration of the methodologies and benefits associated
with market basket analysis.

Data Collection: Obtain the dataset that contains transaction data. This data typically includes information about items purchased together in different transactions.

Market basket analysis is a technique used in data mining and machine learning to discover associations between items that are frequently purchased or used together. To perform market basket analysis, you would typically need transaction data that records what items were bought together in the same transaction. Each row in the dataset represents a transaction, and the columns represent the items that were part of that transaction.

Creating your own dataset for market basket analysis may involve collecting transaction data from a retail store, an e-commerce website, or any other relevant source. However, if you're looking for a sample dataset to practice market basket analysis, here are a few options:

1.Grocery Store Transactions:

- Use a dataset of grocery store transactions where each row represents a customer's purchase, and the columns represent

items bought (e.g., milk, bread, eggs, etc.). You can find such datasets on platforms like Kaggle or from open data sources.

2.Online Retail Transactions:

- Some e-commerce websites provide transaction data for research purposes. For example, the "Online Retail" dataset from the UCI Machine Learning Repository contains e-commerce transaction data.

3. Movie Recommendations:

a. In the context of movie recommendations, you can create a dataset where each row represents a user's movie selections, and the columns represent the movies they watched or rated. This can be used for movie recommendations and association analysis.

4. Library Checkouts:

a. For a library, you could use data representing the books borrowed by library patrons. Each row could be a library checkout, and the columns would be the books borrowed.

5. Restaurant Orders:

a. For a restaurant or fast-food chain, you could use data that logs customer orders. Each row could represent a customer order, and the columns could represent the items ordered (e.g., burgers, fries, drinks).

6.Data Cleaning:

- Remove any duplicates or irrelevant records.
- Handle missing data, if any, by either removing rows with missing values or imputing them.
- Ensure data consistency and integrity.

7.Data Transformation:

- Convert the data into a suitable format, often a transaction list or a binary matrix where rows represent transactions, and columns represent items. Each cell is marked as 1 if the item is in the transaction and 0 otherwise.

8.Support Threshold: Determine a minimum support threshold, which is the minimum percentage of transactions an itemset (combination of items) must appear in to be considered frequent. This threshold helps in reducing the number of itemsets to be analyzed.

9. **Apriori Algorithm** (or other association rule mining algorithms):
    a. Apply the Apriori algorithm or a similar algorithm to find frequent itemsets. The algorithm iteratively discovers itemsets that meet the minimum support threshold.
    b. Generate association rules from frequent itemsets.
10. **Rule Pruning**: Optionally, you can prune rules based on additional criteria like confidence, lift, or interest to focus on the most meaningful associations.
11. **Loading the Processed Data**: Depending on your analysis tool, load the preprocessed data, frequent itemsets, and association rules into a data structure that can be used for insights and further analysis.
12. **Analysis and Insights**: Analyze the generated association rules to gain insights into item relationships. This can include identifying frequently co-purchased items, cross-selling opportunities, and other market basket insights.

When working with a market basket analysis dataset, it's essential to preprocess the data properly and convert it into the right format for association rule mining algorithms like Apriori or FP-growth. You may need to convert the data into a binary format (item purchased or not purchased in a transaction) or other suitable representations.

Remember that it's also important to ensure that the dataset adheres to any privacy and data protection regulations, especially if it contains customer information.

```
import os
import warnings
warnings.simplefilter(action = 'ignore', category=FutureWarnin
g)
warnings.filterwarnings('ignore')
def ignore_warn(*args, **kwargs):
    pass

warnings.warn = ignore_warn #ignore annoying warning (from skle
arn and seaborn)

import pandas as pd
import datetime
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib.cm as cm

%matplotlib inline

from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())

import seaborn as sns
sns.set(style="ticks", color_codes=True, font_scale=1.5)
color = sns.color_palette()
sns.set_style('darkgrid')

from mpl_toolkits.mplot3d import Axes3D

import plotly as py
import plotly.graph_objs as go
py.offline.init_notebook_mode()

from scipy import stats
from scipy.stats import skew, norm, probplot, boxcox
from sklearn import preprocessing
import math

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_sco
re
```

```
import Orange
from Orange.data import Domain, DiscreteVariable, ContinuousVa
riable
from orangecontrib.associate.fpgrowth import *
```

nvoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
36365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.550	17850.000	United Kingdom
36365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom
36365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.750	17850.000	United Kingdom
36365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom
36365	84029E	RED WOOLLY HOTTIE WHITE HEART	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom

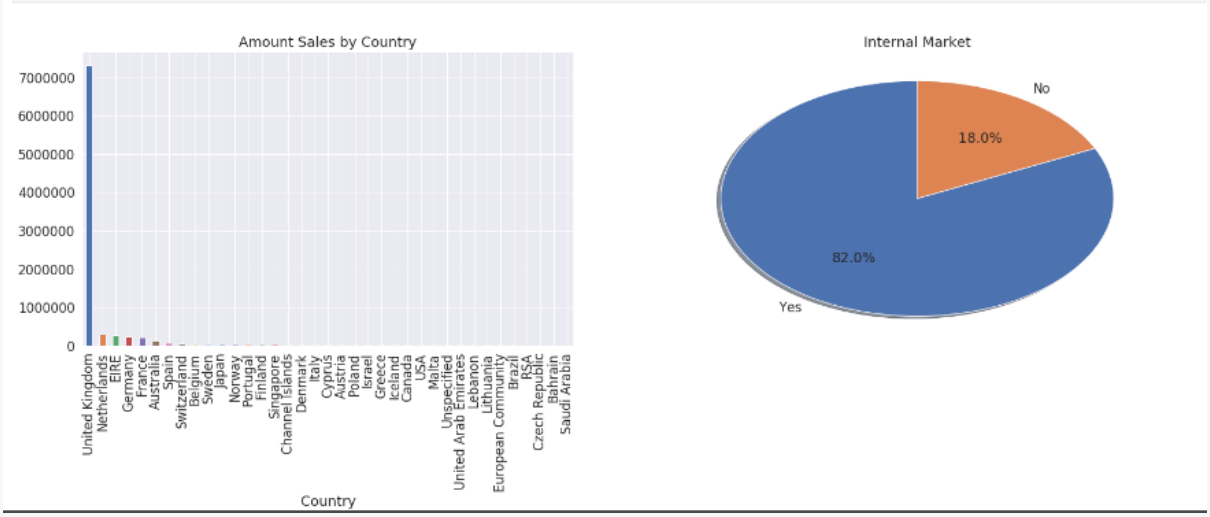
INPUT:

```
def rstr(df, pred=None):
    obs = df.shape[0]
    types = df.dtypes
    counts = df.apply(lambda x: x.count())
    uniques = df.apply(lambda x: [x.unique()])
    nulls = df.apply(lambda x: x.isnull().sum())
    distincts = df.apply(lambda x: x.unique().shape[0])
    missing_ration = (df.isnull().sum()/ obs) * 100
    skewness = df.skew()
    kurtosis = df.kurt()
    print('Data shape:', df.shape)

    if pred is None:
        cols = ['types', 'counts', 'distincts', 'nulls', 'miss
ing ration', 'uniques', 'skewness', 'kurtosis']
        str = pd.concat([types, counts, distincts, nulls, miss
ing_ration, uniques, skewness, kurtosis], axis = 1, sort=True)
```

```
    else:
        corr = df.corr()[pred]
        str = pd.concat([types, counts, distincts, nulls, miss
ing_ration, uniques, skewness, kurtosis, corr], axis = 1, sort
=True)
        corr_col = 'corr '  + pred
        cols = ['types', 'counts', 'distincts', 'nulls', 'miss
ing ration', 'uniques', 'skewness', 'kurtosis', corr_col ]

    str.columns = cols
    dtypes = str.types.value_counts()
    print('_____________________________\nData types:\n',str.typ
es.value_counts())
    print('_____________________________')
    return str

details = rstr(cs_df)
display(details.sort_values(by='missing ration', ascending=Fal
se))
```

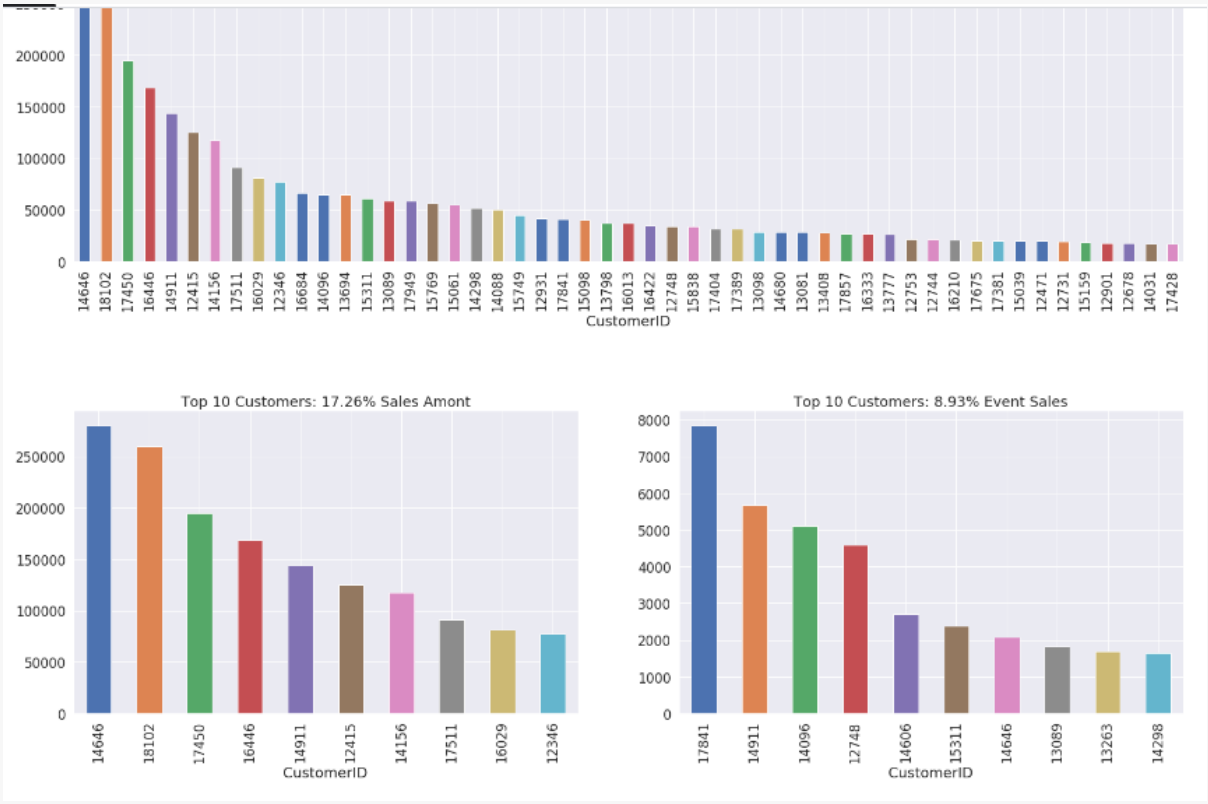
OUTPUT :

	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




```
fig = plt.figure(figsize=(25, 7))
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount
.sum().\
                         sort_values(ascending = False)[:51].
sum()/cs_df.groupby(["CustomerID"]).\
                         amount.sum().sort_values(ascending =
False).sum()) * 100, 2)
g = cs_df.groupby(["CustomerID"]).amount.sum().sort_values(asc
ending = False)[:51].\
    plot(kind='bar', title='Top Customers: {:3.2f}% Sales Amou
nt'.format(PercentSales))

fig = plt.figure(figsize=(25, 7))
f1 = fig.add_subplot(121)
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount
.sum().\
                         sort_values(ascending = False)[:10].
sum()/cs_df.groupby(["CustomerID"]).\
                         amount.sum().sort_values(ascending =
False).sum()) * 100, 2)
g = cs_df.groupby(["CustomerID"]).amount.sum().sort_values(asc
ending = False)[:10]\
    .plot(kind='bar', title='Top 10 Customers: {:3.2f}% Sales
Amont'.format(PercentSales))
f1 = fig.add_subplot(122)
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount
.count().\
```

```
                            sort_values(ascending = False)[:10].
sum()/cs_df.groupby(["CustomerID"]).\
                            amount.count().sort_values(ascending
= False).sum()) * 100, 2)
g = cs_df.groupby(["CustomerID"]).amount.count().sort_values(a
scending = False)[:10].\
    plot(kind='bar', title='Top 10 Customers: {:3.2f}% Event S
ales'.format(PercentSales))
```



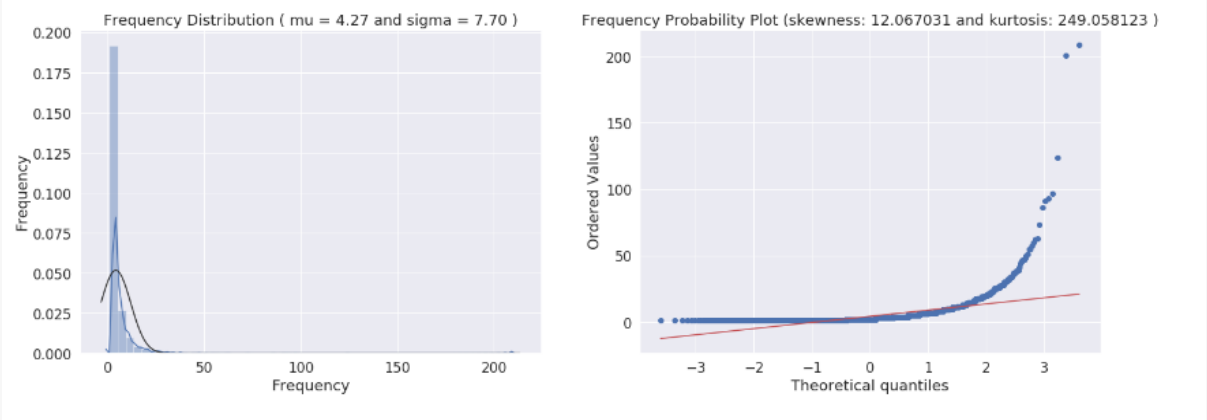
- **Higher Revenue**: This is the most obvious requirement of any customer segmentation project.
- **Customer Understanding**: One of the mostly widely accepted business paradigms is “know your customer” and a segmentation of the customer base allows for a perfect dissection of this paradigm.
- **Target Marketing**: The most visible reason for customer segmentation is the ability to focus marketing efforts effectively and efficiently. If a firm

knows the different segments of its customer base, it can devise better marketing campaigns which are tailor made for the segment. A good segmentation model allows for better understanding of customer requirements and hence increases the chances of the success of any marketing campaign developed by the organization.

- **Optimal Product Placement**: A good customer segmentation strategy can also help the firm with developing or offering new products, or a bundle of products together as a combined offering.
- **Finding Latent Customer Segments**: Finding out which segment of customers it might be missing to identifying untapped customer segments by focused on marketing campaigns or new business development.

```
def QQ_plot(data, measure):
    fig = plt.figure(figsize=(20,7))

    #Get the fitted parameters used by the function
    (mu, sigma) = norm.fit(data)

    #Kernel Density plot
    fig1 = fig.add_subplot(121)
    sns.distplot(data, fit=norm)
    fig1.set_title(measure + ' Distribution ( mu = {:.2f} and
sigma = {:.2f} )'.format(mu, sigma), loc='center')
    fig1.set_xlabel(measure)
    fig1.set_ylabel('Frequency')

    #QQ plot
    fig2 = fig.add_subplot(122)
    res = probplot(data, plot=fig2)
    fig2.set_title(measure + ' Probability Plot (skewness: {:.
6f} and kurtosis: {:.6f} )'.format(data.skew(), data.kurt()),
loc='center')

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

```
QQ_plot(customer_history_df.recency, 'Recency')
```



The whole concept of association rule-mining is based on the concept that customer purchase behavior has a pattern which can be exploited for selling more items to the customer in the future.

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness. This rule-based approach also generates

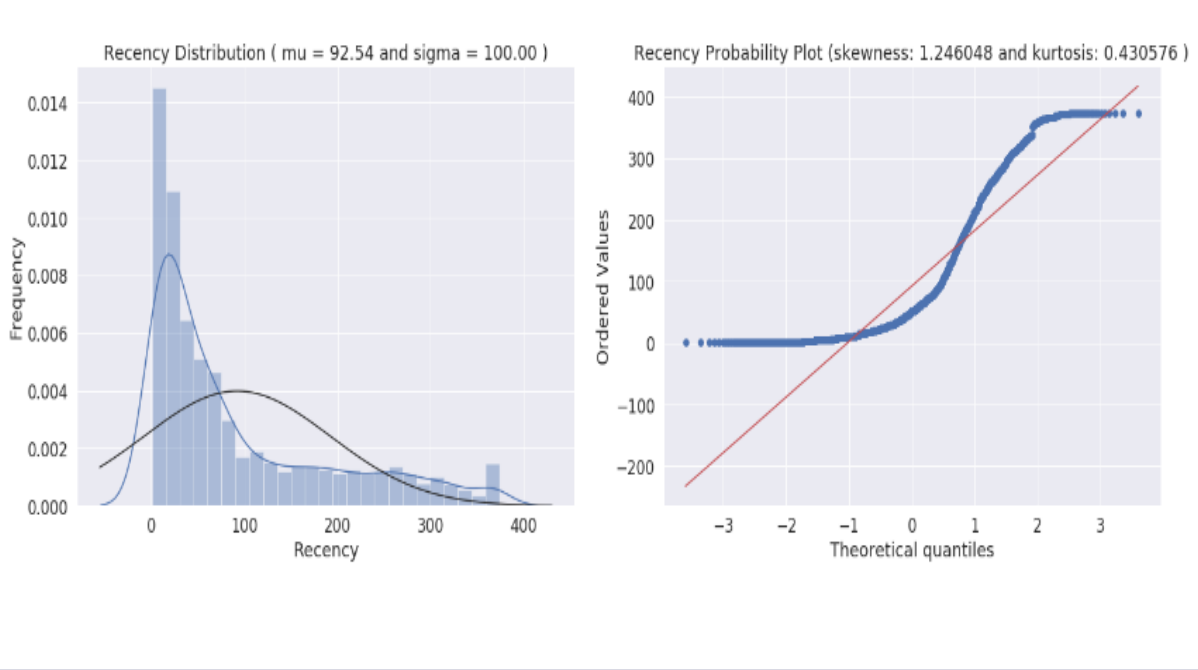
new rules as it analyzes more data. The ultimate goal, assuming a large enough dataset, is to help a machine mimic the human brain's feature extraction and abstract association capabilities from new uncategorized data

```
from mlxtend.frequent_patterns import apriori

from mlxtend.frequent_patterns import association_rules

import pandas as pd


# Sample transaction data

data = {'Transaction': [1, 2, 3, 4, 5],

        'Items': [['A', 'B', 'C'],

                  ['A', 'B'],

                  ['A', 'B', 'D'],

                  ['B', 'C'],

                  ['A', 'B', 'C', 'D']]}

df = pd.DataFrame(data)


# Convert the transaction data into a one-hot encoded DataFrame

oht = df['Items'].str.join(',').str.get_dummies(',')

frequent_itemsets = apriori(oht, min_support=0.4, use_colnames=True)


# Generate association rules

association_rules_df = association_rules(frequent_itemsets, metric='lift',
min_threshold=1.0)


# Display the frequent itemsets and association rules

print("Frequent Itemsets:")
```

```
print(frequent_itemsets)


print("\nAssociation Rules:")

print(association_rules_df)
```

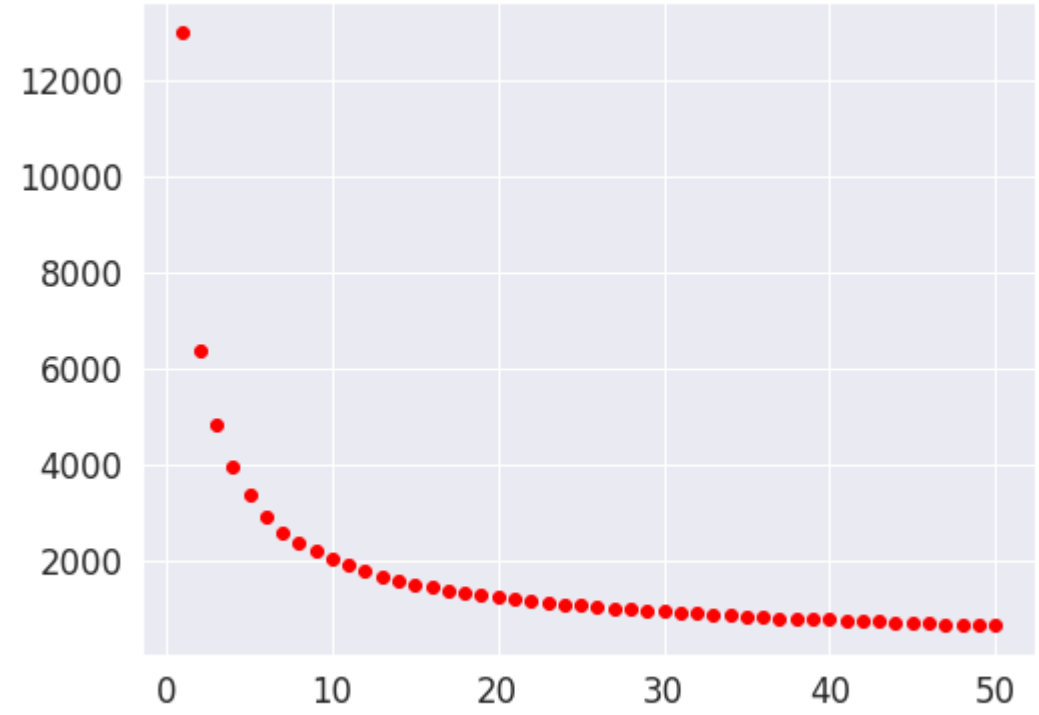
1. Replace `'data.csv'` with the path to your transaction data file. Each row should represent a transaction, and each column should represent an item.
2. The code preprocesses the data, converting it to one-hot encoded format suitable for Apriori analysis.
3. The `apriori` function is used to find frequent itemsets based on a minimum support threshold.
4. The `association_rules` function is used to generate association rules based on the frequent itemsets, specifying a minimum confidence threshold.
5. Finally, the code sorts and displays the association rules based on the lift metric. You can adjust the minimum support and confidence thresholds to tailor the analysis to your specific requirements.

Remember to install the required libraries if you haven't already. You can use `pip install pandas mlxtend` to install them.



```
# Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# Load your transaction data (replace 'data.csv' with your dataset)
data = pd.read_csv('data.csv', header=None)

# Preprocess the data
data = data.applymap(str)  # Convert data to strings
data = data.stack().str.get_dummies().groupby(level=0).sum()  # One-hot encoding

# Apply the Apriori algorithm to find frequent itemsets
min_support = 0.1  # Adjust as needed
frequent_itemsets = apriori(data, min_support=min_support, use_colnames=True)

# Generate association rules
min_confidence = 0.5  # Adjust as needed
rules = association_rules(frequent_itemsets, metric='lift',
min_threshold=min_confidence)

# Sort and display the rules
```

```
sorted_rules = rules.sort_values(by='lift', ascending=False)

print(sorted_rules)
```



Note that by the Elbow method from a K equal to 3 we already observed low rates of gain in the decay of the distortions with the decrease of K reaching the limit of 10% with the K equal to 7. With this in mind, we will begin to evaluate the options more deeply with 3, and 7, starting with the silhouette analysis.

```
refrence_date = cs_df.InvoiceDate.max() + datetime.timedelta(d
ays = 1)
print('Reference Date:', refrence_date)
cs_df['days_since_last_purchase'] = (refrence_date - cs_df.Inv
oiceDate).astype('timedelta64[D]')
customer_history_df =  cs_df[['CustomerID', 'days_since_last_p
urchase']].groupby("CustomerID").min().reset_index()
customer_history_df.rename(columns={'days_since_last_purchase'
:'recency'}, inplace=True)
customer_history_df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
CustomerID	4338.0	15300.408022	1721.808492	12346.0	13813.25	15299.5	16778.75	18287.0
recency	4338.0	92.536422	100.014169	1.0	18.00	51.0	142.00	374.0

From the first graph above we can see that sales recency distribution is ***skewed***, has a **peak** on the left and a long tail to the right. It **deviates from normal distribution** and is **positively biased**.

From the **Probability Plot**, we could see that **sales recency** also does **not align with the diagonal red line** which represent normal distribution. The form of its distribution confirm that is a skewed right.

With ***skewness positive of 1.25***, we confirm the **lack of symmetry** and indicate that sales recency are **skewed right**, as we can see too at the Sales Distribution plot, skewed right means that the right tail is **long relative to the left tail**. The skewness for a normal distribution is zero, and any symmetric data should have a skewness near zero. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point.

Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. That is, data sets with high kurtosis tend to have heavy tails, or outliers, and **positive** kurtosis indicates a **heavy-tailed distribution** and **negative** kurtosis indicates a **light tailed distribution**. So, with 0.43 of positive kurtosis **sales recency** are heavy-tailed and has some **outliers**.

```
customer_freq = (cs_df[['CustomerID', 'InvoiceNo']].groupby(["
CustomerID", 'InvoiceNo']).count().reset_index()).\
                groupby(["CustomerID"]).count().reset_index()
```

```
customer_freq.rename(columns={'InvoiceNo':'frequency'},inplace
=True)
customer_history_df = customer_history_df.merge(customer_freq)
QQ_plot(customer_history_df.frequency, 'Frequency')
```

Once we have created our customer value dataset, we will perform some preprocessing on the data. For our clustering, we will be using the K-means clustering algorithm. One of the requirements for proper functioning of the algorithm is the mean centering of the variable values. Mean centering of a variable value means that we will replace the actual value of the variable with a standardized value, so that the variable has a mean of 0 and variance of 1. This ensures that all the variables are in the same range and the difference in ranges of values doesn't cause the algorithm to not perform well. This is akin to feature scaling.

Another problem that you can investigate about is the huge range of values each variable can take. This problem is particularly noticeable for the monetary amount variable. To take care of this problem, we will transform all the variables on the log scale. This transformation, along with the standardization, will ensure that the input to our algorithm is a homogenous set of scaled and transformed values.

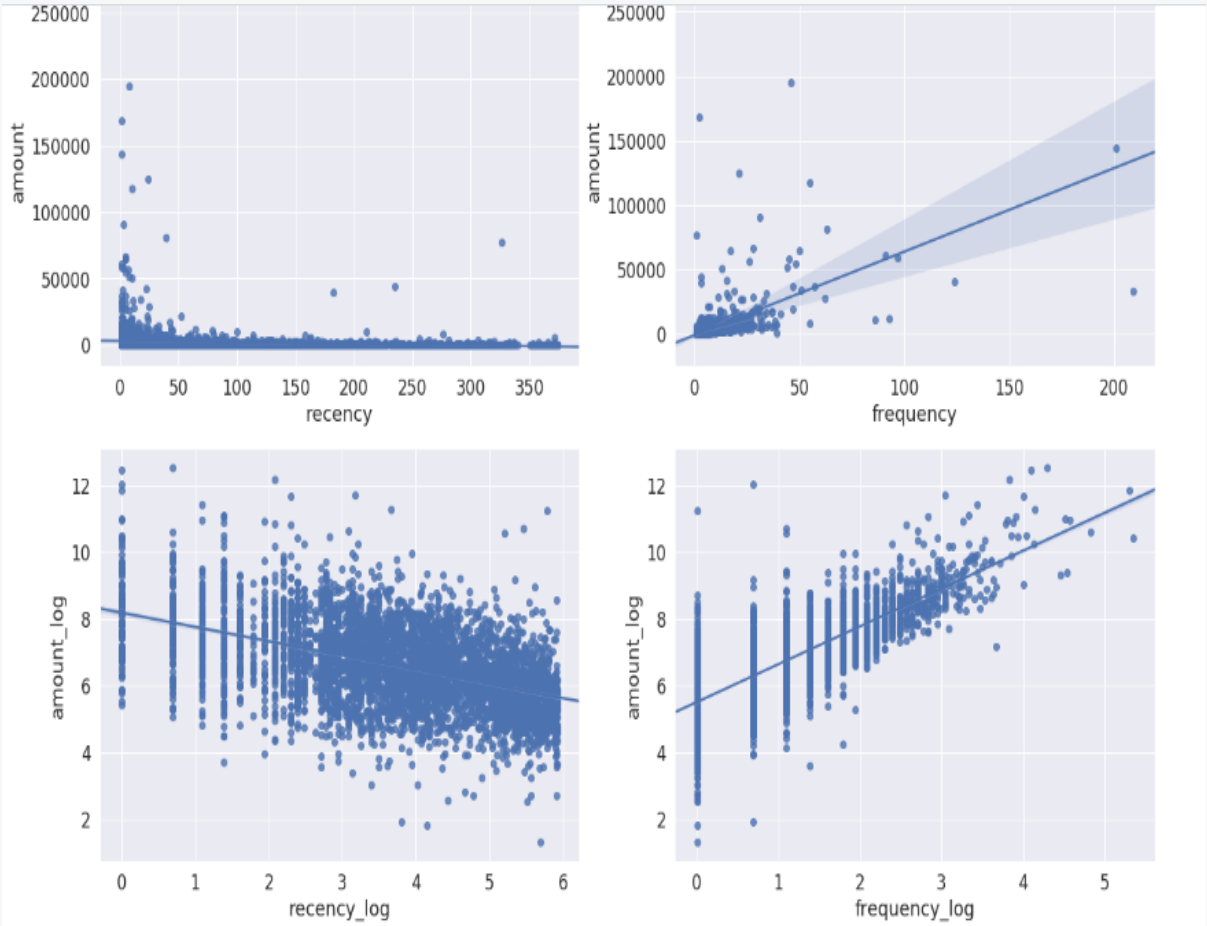
An important point about the data preprocessing step is that sometimes we need it to be reversible. In our case, we will have the clustering results in terms of the log transformed and scaled variable. But to make inferences in terms of the original data, we will need to reverse transform all the variable so that we get back the actual RFM figures. This can be done by using the preprocessing capabilities of Python.

```
linkcode
customer_history_df['recency_log'] = customer_history_df['rece
ncy'].apply(math.log)
customer_history_df['frequency_log'] = customer_history_df['fr
equency'].apply(math.log)
customer_history_df['amount_log'] = customer_history_df['amoun
t'].apply(math.log)
feature_vector = ['amount_log', 'recency_log','frequency_log']
X_subset = customer_history_df[feature_vector] #.as_matrix()
scaler = preprocessing.StandardScaler().fit(X_subset)
X_scaled = scaler.transform(X_subset)
pd.DataFrame(X_scaled, columns=X_subset.columns).describe().T
```

	count	mean	std	min	25%	50%	75%	max
amount_log	4338.0	-1.202102e-16	1.000115	-4.179280	-0.684183	-0.060942	0.654244	4.721395
recency_log	4338.0	-1.027980e-16	1.000115	-2.630445	-0.612424	0.114707	0.829652	1.505796
frequency_log	4338.0	-2.355833e-16	1.000115	-1.048610	-1.048610	-0.279044	0.738267	4.882714

```
fig = plt.figure(figsize=(20,14))
f1 = fig.add_subplot(221); sns.regplot(x='recency', y='amount'
, data=customer_history_df)
f1 = fig.add_subplot(222); sns.regplot(x='frequency', y='amoun
t', data=customer_history_df)
f1 = fig.add_subplot(223); sns.regplot(x='recency_log', y='amo
unt_log', data=customer_history_df)
f1 = fig.add_subplot(224); sns.regplot(x='frequency_log', y='a
mount_log', data=customer_history_df)

fig = plt.figure(figsize=(15, 10))
ax = fig.add_subplot(111, projection='3d')

xs =customer_history_df.recency_log
ys = customer_history_df.frequency_log
zs = customer_history_df.amount_log
ax.scatter(xs, ys, zs, s=5)

ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')
```

```
plt.show()
```



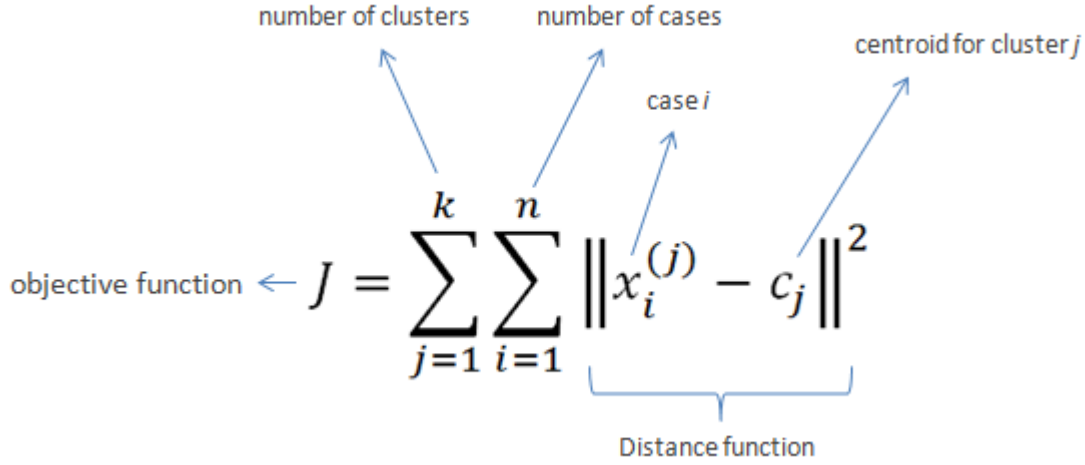
The obvious patterns we can see from the plots above is that c ostumers who buy with a higher frequency and more recency te nd to spend more based on the increasing trend in Monetary (a mount value) with a corresponding increasing and decreasing tr end for Frequency and Recency, respectively.

K-Means Clustering

The K-means clustering belongs to the partition based\centroid based hard clustering family of algorithms, a family of algorithms where each sample in a dataset is assigned to exactly one cluster.

Based on this Euclidean distance metric, we can describe the k-means algorithm as a simple optimization problem, an

iterative approach for minimizing the within-cluster sum of squared errors (SSE), which is sometimes also called cluster inertia. So, the objective of K-Means clustering is to minimize total intra-cluster variance, or, the squared error function:



The steps that happen in the K-means algorithm for partitioning the data are as given follows:

1. The algorithm starts with random point initializations of the required number of centers. The “K” in K-means stands for the number of clusters.
2. In the next step, each of the data point is assigned to the center closest to it. The distance metric used in K-means clustering is normal Euclidian distance.
3. Once the data points are assigned, the centers are recalculated by averaging the dimensions of the points belonging to the cluster.
4. The process is repeated with new centers until we reach a point where the assignments become stable. In this case, the algorithm terminates.

- Place the initial centroids far away from each other via the k-means++ algorithm, which leads to better and more consistent results than the classic k-means.

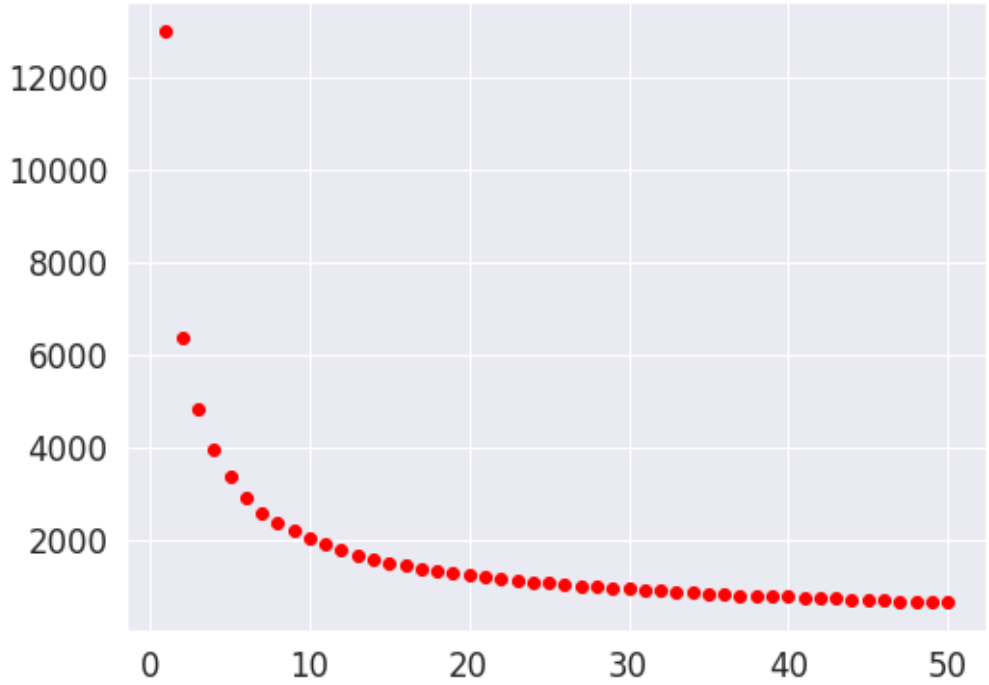
To use k-means++ with scikit-learn's KMeans object, we just need to set the init parameter to k-means++ (the default setting) instead of random.

- using the elbow method to find the optimal number of clusters. The idea behind the elbow method is to identify the value of k where the distortion begins to increase most rapidly. If k increases, the distortion will decrease, because the samples will be closer to the centroids they are assigned to.
- This method looks at the percentage of variance explained as a function of the number of clusters. More precisely, if one plots the percentage of variance explained by the clusters against the number of clusters, the first clusters will add much information (explain a lot of variance), but at some point the marginal gain will drop, giving an angle in the graph. The number of clusters is chosen at this point, hence the "elbow criterion". This "elbow" cannot always be unambiguously identified.Percentage of variance explained is the ratio of the between-group variance to the total variance, also known as an F-test. A slight variation of this method plots the curvature of the within group variance.

```
cl = 50
corte = 0.1

anterior = 100000000000000
cost = []
K_best = cl

for k in range (1, cl+1):
   # Create a kmeans model on our data, using k clusters.
random_state helps ensure that the algorithm returns the s
ame results each time.
   model = KMeans(
        n_clusters=k,
        init='k-means++', #'random',
        n_init=10,
        max_iter=300,
```

```
        tol=1e-04,
        random_state=101)

    model = model.fit(X_scaled)

    # These are our fitted labels for clusters -- the firt
cluster has label 0, and the second has label 1.
  labels = model.labels

    # Sum of distances of samples to their closest cluster
center
    interia = model.inertia_
    if (K_best == cl) and (((anterior - interia)/anterior
) < corte): K_best = k - 1
    cost.append(interia)
    anterior = interia

plt.figure(figsize=(8, 6))
plt.scatter(range (1, cl+1), cost, c='red')
plt.show()

# Create a kmeans model with the best K.
print('The best K sugest: ',K_best)
model = KMeans(n_clusters=K_best, init='k-means++', n_int
t=10,max_iter=300, tol=1e-04, random_state=101)

# Note I'm scaling the data to normalize it! Important for
good results.
model = model.fit(X_scaled)

# These are our fitted labels for clusters -- the first cl
uster has label 0, and the second has label 1.
labels = model.labels_

# And we'll visualize it:
#plt.scatter(X_scaled[:,0], X_scaled[:,1], c=model.labels_
.astype(float))
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(121)
plt.scatter(x = X_scaled[:,1], y = X_scaled[:,0], c=model
.labels_.astype(float))
ax.set_xlabel(feature_vector[1])
ax.set_ylabel(feature_vector[0])
ax = fig.add_subplot(122)
```

```
plt.scatter(x = X_scaled[:,2], y = X_scaled[:,0], c=mode.
labels_.astype(float))
ax.set_xlabel(feature_vector[2])
ax.set_ylabel(feature_vector[0])

plt.show()
```



Silhouette analysis can be used to study the separation distance between the resulting clusters, as a strategy to quantifying the quality of clustering via graphical tool to plot a measure of how tightly grouped the samples in the clusters are. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually.

It can also be applied to clustering algorithms other than k-means

Silhouette coefficients has a range of [-1, 1], it calculated by:

1. Calculate the cluster cohesion a(i)as the average distance between a sample x(i) and all other points in the same cluster.
2. Calculate the cluster separation b(i) from the next closest cluster as the average distance between the sample x(i) and all samples in the nearest cluster.
3. Calculate the silhouette s(i) as the difference between cluster cohesion and separation divided by the greater of the two, as shown here: Which can be also written as:

Where:

- If near +1, it indicate that the sample is far away from the neighboring clusters.
- a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.
- If most objects have a high value, then the clustering configuration is appropriate.
- If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.
- A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters
- Negative values indicate that those samples might have been assigned to the wrong cluster.

The silhouette plot can shows a bad K clusters pick for the given data due to the presence of clusters with below average silhouette scores and also due to wide fluctuations in the size of the silhouette plots. A good k clusters can found when all the plots are more or less of similar thickness and hence are of similar sizes.

Although we have to keep in mind that in several cases and scenarios, sometimes we may have to drop the mathematical explanation given by the algorithm and look at the business relevance of the results obtained.

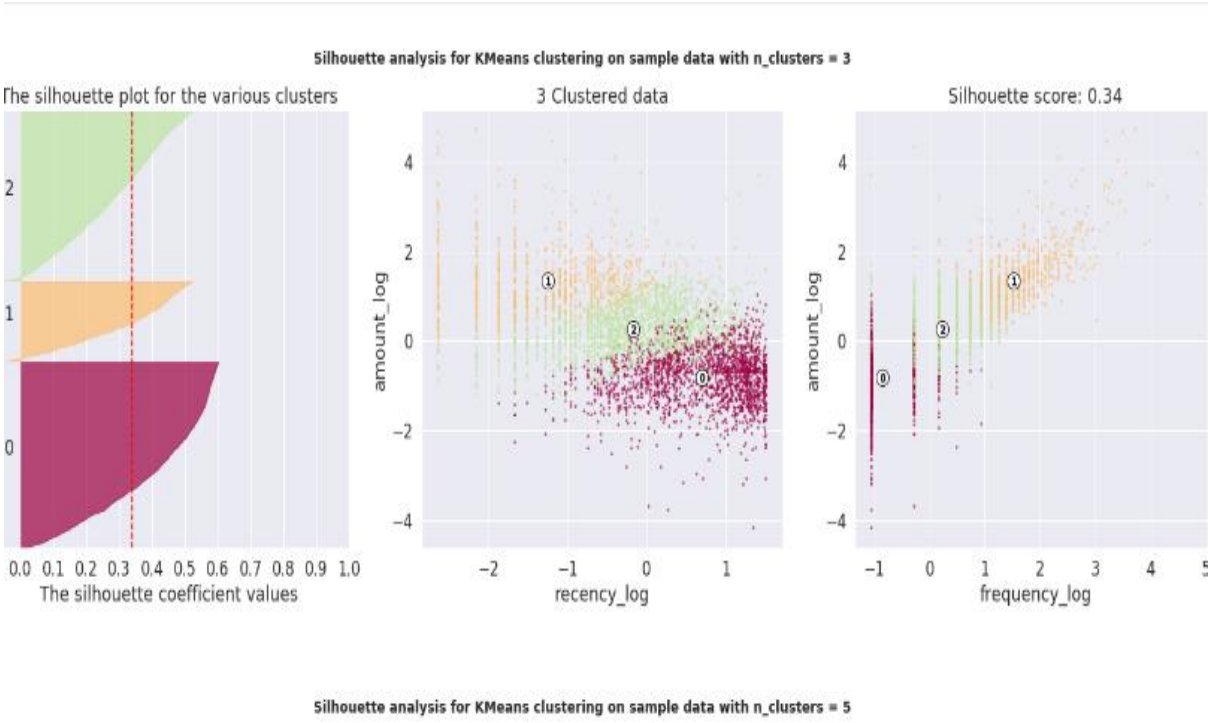
Let's see below how our data perform for each K clusters groups (3, 5 and 7) in the silhouette score of each cluster, along with the center of each of the cluster discovered in the scatter plots, by amount_log vs recency_log and vs frequency_log.

In [23]:

```
linkcode
cluster_centers = dict()

for n_clusters in range(3,K_best+1,2):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
    fig.set_size_inches(25, 7)
    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(X_scaled) + (n_clusters + 1) * 10])

    clusterer = KMeans(n_clusters=n_clusters, init='k-means++'
, n_init=10,max_iter=300, tol=1e-04, random_state=101)
    cluster_labels = clusterer.fit_predict(X_scaled)

    silhouette_avg = silhouette_score(X = X_scaled, labels = c
luster_labels)
    cluster_centers.update({n_clusters :{'cluster_center':clus
terer.cluster_centers_,
                                         'silhouette_score':si
lhouette_avg,
                                         'labels':cluster_labe
ls}
                            })

    sample_silhouette_values = silhouette_samples(X = X_scaled
, labels = cluster_labels)
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_valu
es[cluster_labels == i]

        ith_cluster_silhouette_values.sort()
```

```
        size_cluster_i = ith_cluster_silhouette_values.shape[0
]
        y_upper = y_lower + size_cluster_i

        color = cm.Spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, al
pha=0.7)

        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i)
)
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various cluster
s")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([])
    ax1.set_xticks([-0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7
, 0.8, 0.9, 1])
    colors = cm.Spectral(cluster_labels.astype(float) / n_clus
ters)

    centers = clusterer.cluster_centers_
    y = 0
    x = 1
    ax2.scatter(X_scaled[:, x], X_scaled[:, y], marker='.', s=
30, lw=0, alpha=0.7, c=colors, edgecolor='k')
    ax2.scatter(centers[:, x], centers[:, y], marker='o', c="w
hite", alpha=1, s=200, edgecolor='k')
    for i, c in enumerate(centers):
        ax2.scatter(c[x], c[y], marker='$%d$' % i, alpha=1, s=
50, edgecolor='k')
    ax2.set_title("{} Clustered data".format(n_clusters))
    ax2.set_xlabel(feature_vector[x])
    ax2.set_ylabel(feature_vector[y])

    x = 2
    ax3.scatter(X_scaled[:, x], X_scaled[:, y], marker='.', s=
30, lw=0, alpha=0.7, c=colors, edgecolor='k')
```

```
    ax3.scatter(centers[:, x], centers[:, y], marker='o', c="w
hite", alpha=1, s=200, edgecolor='k')
    for i, c in enumerate(centers):
        ax3.scatter(c[x], c[y], marker='$%d$' % i, alpha=1, s=
50, edgecolor='k')
    ax3.set_title("Silhouette score: {:1.2f}".format(cluster_c
enters[n_clusters]['silhouette_score']))
    ax3.set_xlabel(feature_vector[x])
    ax3.set_ylabel(feature_vector[y])

    plt.suptitle(("Silhouette analysis for KMeans clustering o
n sample data with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')
    plt.show()
```



- First notice that all K clusters options is valid, because they don't have presence of clusters with below average silhouette scores.

- In the other hand, all options had a some wide fluctuations in the size of the silhouette plots.

So, the best choice may lie on the option that gives us a simpler business explanation and at the same time target customers in focus groups with sizes closer to the desired

```
features = ['amount',  'recency',  'frequency']
for i in range(3,K_best+1,2):
    print("for {} clusters the silhouette score is {:1.2f}".fo
rmat(i, cluster_centers[i]['silhouette_score']))
    print("Centers of each cluster:")
    cent_transformed = scaler.inverse_transform(cluster_center
s[i]['cluster_center'])
    print(pd.DataFrame(np.exp(cent_transformed),columns=featur
es))
    print('-'*50)
```

In the three-cluster:

- The tree clusters appears have a good stark differences in the Monetary value of the customer, we will confirm this by a box plot.
- Cluster 1 is the cluster of high value customer who shops frequently and is certainly an important segment for each business.
- In the similar way we obtain customer groups with low and medium spends in clusters with labels 0 and 2, respectively.
- Frequency and Recency correlate perfectly to the Monetary value based on the trend (High Monetary-Low Recency-High Frequency).

In the five-cluster:

- Note that clusters 0 and 1 are very similar to their cluster in the configuration with only 3 clusters.

- The cluster 1 appears more robust on the affirmation of those who shop often and with high amount.
- The cluster 2 are those who have a decent spend but are not as frequent as the cluster 1
- The cluster 4 purchases medium amounts, with a relatively low frequency and not very recent
- The cluster 3 makes low-cost purchases, with a relatively low frequency, but above 1, and made their last purchase more recently. This group of customers probably response to price discounts and can be subject to loyalty promotions to try increase the medium-ticket, strategy that can be better defined when we analyzing the market basket.
- The silhouette score matrix says that the five cluster segments are less optimal then the three cluster segments.

In the five-cluster:

- Definitely cluster 6 defines those who shop often and with high amount.
- Clusters 1 and 5 show good spending and good frequency, only deferring in how recent were their last purchases, where 5 is older, which suggests an active action to sell to group 5 as soon as possible and another to 1 seeking to raise its frequency.
- Cluster 0 presents the fourth best purchase and a reasonable frequency, but this is a long time without buying. This group should be sensible to promotions and activations, so that they do not get lost and make their next purchase.
- Cluster 5 is similar to 0, but has made its purchases more recently and has a slightly better periodicity. Then actions must be taken to raise their frequency and reduce the chances of them migrating to

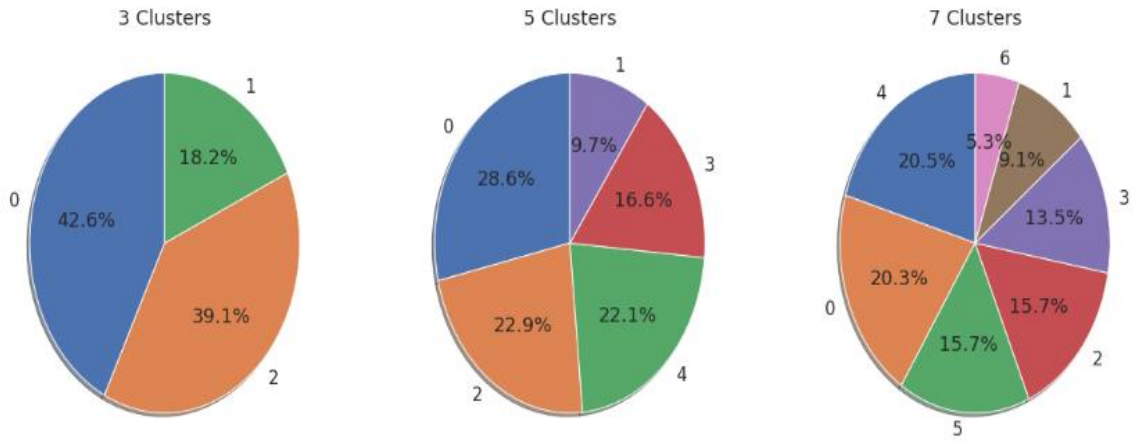
cluster 0 by staying longer without purchasing products.

Drill Down Clusters:

To further drill down on this point and find out the quality of these difference, we can label our data with the corresponding cluster label and then visualize these differences. The following code will extract the clustering label and attach it with our customer summary dataset.

linkcode

```
customer_history_df['clusters_3'] = cluster_centers[3]['labels
']
customer_history_df['clusters_5'] = cluster_centers[5]['labels
']
customer_history_df['clusters_7'] = cluster_centers[7]['labels
']
display(customer_history_df.head())

fig = plt.figure(figsize=(20,7))
f1 = fig.add_subplot(131)
market = customer_history_df.clusters_3.value_counts()
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', sh
adow=True, startangle=90)
plt.title('3 Clusters')
f1 = fig.add_subplot(132)
market = customer_history_df.clusters_5.value_counts()
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', sh
adow=True, startangle=90)
plt.title('5 Clusters')
f1 = fig.add_subplot(133)
market = customer_history_df.clusters_7.value_counts()
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', sh
adow=True, startangle=90)
plt.title('7 Clusters')
plt.show()
```



Once we have the labels assigned to each of the customers, our task is simple. Now we want to find out how the summary of customer in each group is varying. If we can visualize that information we will able to find out the differences in the clusters of customers and we can modify our strategy on the basis of those differences.

The following code leverages plotly and will take the cluster labels we got for each configurations clusters and create boxplots. Plotly enables us to interact with the plots to see the central tendency values in each boxplot in the notebook. Note that we want to avoid the extremely high outlier values of each group, as they will interfere in making a good observation around the central tendencies of each cluster. Since we have only positive values, we will restrict the data such that only data points which are less than 0.95th percentile of the cluster is used. This will give us good information about the majority of the users in that cluster segment.

I've used these charts to review my previously stated insights, but follow the same for you to explore:

```
linkcode
x_data = ['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3','Cl
uster 4', 'Cluster 5', 'Cluster 6']
colors = ['rgba(93, 164, 214, 0.5)', 'rgba(255, 144, 14, 0.5)'
, 'rgba(44, 160, 101, 0.5)', 'rgba(255, 65, 54, 0.5)',
          'rgba(22, 80, 57, 0.5)', 'rgba(127, 65, 14, 0.5)', '
rgba(207, 114, 255, 0.5)', 'rgba(127, 96, 0, 0.5)']
cutoff_quantile = 95

for n_clusters in range(3,K_best+1,2):
    cl = 'clusters_' + str(n_clusters)
    for fild in range(0, 3):
        field_to_plot = features[fild]
        y_data = list()
        ymax = 0
        for i in np.arange(0,n_clusters):
            y0 = customer_history_df[customer_history_df[cl]==
i][field_to_plot].values
            y0 = y0[y0<np.percentile(y0, cutoff_quantile)]
            if ymax < max(y0): ymax = max(y0)
            y_data.insert(i, y0)

        traces = []

        for xd, yd, cls in zip(x_data[:n_clusters], y_data, co
lors[:n_clusters]):
                traces.append(go.Box(y=yd, name=xd, boxpoints=
False, jitter=0.5, whiskerwidth=0.2, fillcolor=cls,
                    marker=dict( size=1, ),
                    line=dict(width=1),
                ))

        layout = go.Layout(
            title='Difference in {} with {} Clusters and {:1.2
f} Score'.\
            format(field_to_plot, n_clusters, cluster_centers[
n_clusters]['silhouette_score']),
            yaxis=dict( autorange=True, showgrid=True, zerolin
e=True,
                dtick = int(ymax/10),
                gridcolor='black', gridwidth=0.1, zerolinecolo
r='rgb(255, 255, 255)', zerolinewidth=2, ),
            margin=dict(l=40, r=30, b=50, t=50, ),
            paper_bgcolor='white',
```

```
        plot_bgcolor='white',
        showlegend=False
    )

    fig = go.Figure(data=traces, layout=layout)
    py.offline.iplot(fig)
```

- New indicators, such as customer relationship time, based on the date of your first purchase of the client
- whether the customer is from abroad or not
- some group or category of product to be obtained through the SKUs
- External data vendors and use it, and so on.

Another dimension to explore can be trying out different algorithms for performing the segmentation for instance hierarchical clustering, which we explored in some of the earlier chapters. A good segmentation process will encompass all these avenues to arrive at optimal segments that provide valuable insight.