Market basket insight phase 5

Introduction :
Market basket analysis, also known as association rule mining, is a data mining technique used to discover interesting relationships or patterns within large sets of transactional data, such as customer purchases. It is widely applied in retail and e-commerce to understand customer behavior, improve sales, and enhance the customer shopping experience.

The fundamental concept behind market basket analysis is to uncover associations between products that are often purchased together. This information is valuable for various purposes:

1. **Cross-Selling**:** Retailers can use these insights to recommend complementary products, increasing the average transaction value.

2. **Inventory Management**:** Understanding which items are frequently bought together helps in optimizing stock levels and layout design.

3. **Customer Segmentation**:** Retailers can group customers based on their purchase patterns, allowing for targeted marketing strategies.

4. **Promotion Planning**:** Retailers can design promotions and discounts that capitalize on these associations.

The analysis typically results in "association rules" that describe the likelihood of one item being purchased when another item is bought. These rules are quantified using metrics like support, confidence, and lift.

- **Support** measures how frequently an itemset appears in the transactions.
- **Confidence** calculates the probability of one item being bought when another is purchased.
- **Lift** is a measure of how much more likely the items are to be bought together compared to if they were bought independently.

Market basket analysis provides valuable insights into consumer behavior, aiding businesses in making data-driven decisions to boost sales and enhance the overall shopping experience. It's not limited to retail and is also used in other fields like recommendation systems, healthcare, and more.

Dataset link:**https://www.kaggle.com/datasets/aslanahmedov/market-basket-analysis**

Here's list of tools and software commonly used in this process

Market basket analysis is a data mining technique used to discover associations between products or items frequently purchased together. Common tools used for market basket analysis include:

1. **Apriori Algorithm**: A popular algorithm for finding frequent itemsets and association rules in transactional data.

2. **FP-growth Algorithm:** Another algorithm for frequent itemset mining that can be more efficient than Apriori for large datasets.

3. **Python and R**: Programming languages often used for data analysis and implementing market basket analysis algorithms.

4. **Data Mining Software**: Tools like RapidMiner, KNIME, and Orange offer user-friendly interfaces for market basket analysis.

5. **SQL**: Structured Query Language is used to query and manipulate transactional data from databases.

6. **Excel**: Basic analysis can be performed using Excel, especially for smaller datasets.

7. **Tableau and Power BI**: Data visualization tools that can help in exploring and presenting the results of market basket analysis.

8. **Weka**: A data mining software that provides various machine learning and data mining tools, including association rule mining.

9. **Hadoop and Spark**: Distributed computing frameworks used for processing large-scale transactional data.

10. **Market Basket Analysis Libraries**: Various Python libraries like mlxtend and Orange provide functions for conducting market basket analysis.

11. **Visualization Tools**: Tools like Matplotlib and Seaborn can be used to create visualizations of association rules and itemsets.

12. **Online Retail Software**: E-commerce platforms often provide built-in tools for market basket analysis to optimize product recommendations.

Remember that the choice of tools depends on the specific requirements of your analysis and the scale of your data.

Problem Definition and Design Thinking

Problem Statement:

The problem statement for market basket insights typically involves analyzing customer purchase data to identify patterns and relationships between products bought together. This analysis aims to answer questions such as:

1. What are the most frequently purchased items together?
2. Can we recommend additional products to customers based on their current selections?
3. How can we optimize product placement within a store to encourage complementary purchases?
4. Are there any seasonal or time-based trends in customer shopping behavior?
5. How can we improve inventory management based on purchase patterns?

By addressing these questions, businesses can enhance their marketing strategies, optimize inventory, and improve the overall customer shopping experience.

Problem definition:

In market basket analysis, the problem definition typically revolves around identifying associations or patterns in customer purchasing behavior. Here's a concise problem definition:

"Market basket insight is the process of analyzing transaction data to uncover associations between products that are frequently purchased together. The goal is to discover patterns and relationships in customer shopping behavior to improve marketing, sales, and inventory management strategies."

Key components of this problem definition include:

1. **Transaction Data**: The analysis is based on data containing records of customer transactions, where each transaction lists the products purchased by a customer during a single shopping trip.

2. **Associations**: The primary objective is to find associations or itemsets of products that tend to be bought together in transactions. These associations are often referred to as "frequent itemsets."

3. **Insights**: The ultimate goal is to gain insights into customer behavior, which can inform various business decisions, such as product placement, cross-selling, and targeted marketing campaigns.

4. **Business Applications**: Market basket insight has practical applications in retail, e-commerce, and other industries where understanding customer preferences and optimizing product offerings is crucial.

When working on market basket analysis, you would typically use techniques like Apriori or FP-growth to identify these associations and extract meaningful insights from the data.

Design Thinking:

Design thinking can be applied to market basket analysis to help businesses gain deeper insights into customer behavior and make data-driven decisions. Here's how you can use design thinking principles in the context of market basket analysis:

1. **Empathize**:
   - Start by understanding your customers' needs, preferences, and pain points. Gather qualitative data through interviews, surveys, and observations.
   - Analyze historical transaction data to identify patterns and trends in customer purchasing behavior.

2. **Define**:
   - Clearly define the problem you want to address with market basket analysis. For example, it could be optimizing product placement, cross-selling, or improving the overall shopping experience.
   - Create a customer persona or journey map to visualize the customer's shopping experience.

3. **Ideate**:
   - Brainstorm potential solutions and insights based on the data and customer feedback.
   - Encourage cross-functional teams to come up with creative ideas, such as personalized product recommendations or store layout changes.

4. **Prototype**:
   - Develop prototypes of different strategies based on the insights from market basket analysis. This could involve rearranging store shelves, creating targeted marketing campaigns, or adjusting pricing strategies.
   - Test these prototypes in a controlled environment or through A/B testing to measure their impact.

5. **Test and Iterate**:
   - Collect data on the performance of your prototypes and analyze the results. Did the changes lead to increased sales, customer satisfaction, or other relevant KPIs?
   - Use the feedback and data to refine your strategies. Iterate on your prototypes and continue testing until you achieve the desired outcomes.

6. **Implement**:
   - Once you have a well-tested and refined solution, implement it across your business operations.
   - Monitor its performance in the long term and make necessary adjustments as market dynamics and customer preferences change.

7. **Evaluate**:
   - Continuously evaluate the effectiveness of your market basket analysis strategies.
   - Use customer feedback and new data to adapt and improve your approaches over time.

Design into Innovation

Technique used in market basket analysis
Advanced association analysis techniques are a valuable approach for gaining deeper insights into market basket data. Some advanced techniques include

Sequential Pattern Mining:
This method considers the order in which items are purchased, providing insights into the sequence of customer behavior
Code:
1)Import necessary libraries:

```
        import pandas as pd
from mlxtend.frequent_patterns import apriori
        from mlxtend.frequent_patterns import association_rules
```

2) Load your Kaggle market basket dataset:

```
# Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')
```

3)Preprocess the data:
Assuming your dataset has a column 'TransactionID' and 'Product'

```
basket = (df.groupby(['TransactionID', 'Product'])['Product']
        .count().unstack().reset_index().fillna(0)
        .set_index('TransactionID'))
```

4) Convert the dataset into a binary format(0 or1)

```
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket.applymap(encode_units)
```

5) Perform sequential pattern mining using Apriori or FP-Growth:

```
# Use Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(basket_sets, min_support=0.02, use_colnames=True)

# Mine association rules from frequent itemsets
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

# Sort rules by confidence
rules = rules.sort_values(by=['confidence'], ascending=False)
```

Time-Series Analysis:

Analyzing market basket data over time can reveal trends, seasonality, and recurring purchase patterns.

Code:

Performing time-series analysis on market basket data involves examining how purchasing patterns evolve over time. Here's a basic example of how you can do this using Python and a sample dataset. You can adapt this code to your specific Kaggle dataset.

Import the necessary libraries:

```
import pandas as pd
import matplotlib.pyplot as plt

Load your market basket dataset
Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')

Preprocess the data and convert it into a time series

Assuming your dataset has a 'Date' and 'Product' column
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# Resample the data to the desired frequency (e.g., daily, weekly)
# For example, resampling to weekly:
weekly_basket = df.groupby('Product').resample('W')['Product'].count().unstack().fillna(0)
```

Visualize the time series data:

```
# Plot the time series for a specific product
```

```
product_name = 'YourProduct'
plt.figure(figsize=(12, 6))
plt.plot(weekly_basket.index, weekly_basket[product_name])
plt.title(f'Time Series for {product_name}')
plt.xlabel('Date')
plt.ylabel('Quantity Sold')
plt.grid(True)
plt.show()
```

Hierarchical Association Rules:

These rules capture associations at multiple levels, such as department, category, and product, offering a more granular view of customer preferences.

Code:

Hierarchical Association Rules can provide insights into market basket data by capturing associations at different levels of hierarchy, such as departments, categories, and products. Below is a basic example of how to implement hierarchical association rules in Python using a sample dataset. You can adapt this code to your specific Kaggle dataset:

1. Import the necessary libraries:

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

2. Load your market basket dataset:

```python
# Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')
```

3. Preprocess the data and convert it into a binary format (0 or 1) suitable for association rule mining:

```python
# Assuming your dataset has a 'TransactionID' and 'Product' column
basket = (df.groupby(['TransactionID', 'Product'])['Product']
          .count().unstack().reset_index().fillna(0)
          .set_index('TransactionID'))

def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket.applymap(encode_units)
```

4. Define the hierarchy of your products. For example, you can have a product hierarchy with levels like 'Department', 'Category', and 'Product':

```python
# Create a new DataFrame with hierarchical product information
product_hierarchy = df[['Product', 'Category', 'Department']].drop_duplicates()
```

5. Perform hierarchical association rule mining:

```python
# Define a function to generate rules for a specific hierarchy level
def mine_hierarchy(basket_sets, level):
    frequent_itemsets = apriori(basket_sets, min_support=0.02, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

    # Filter rules for the specified level
    filtered_rules = rules[rules['antecedents'].apply(lambda x: level in x)].copy()

    return filtered_rules

# Mine association rules at the 'Department' level
department_rules = mine_hierarchy(basket_sets, 'Department')

# Mine association rules at the 'Category' level
category_rules = mine_hierarchy(basket_sets, 'Category')

# Mine association rules at the 'Product' level
product_rules = mine_hierarchy(basket_sets, 'Product')
```

6. Analyze and interpret the hierarchical association rules to gain insights into the relationships between departments, categories, and products in the market basket data

.

Apriori Variations:

Modified versions of the classic Apriori algorithm can improve efficiency and accuracy in finding associations

Code:

Apriori variations are optimized versions of the classic Apriori algorithm that improve efficiency and reduce computational complexity when mining association rules from market basket data. One such variation is the "FP-Growth" algorithm. Here's how to implement FP-Growth in Python using the `pyfpgrowth` library:

1. Install the `pyfpgrowth` library if you haven't already:

```bash
pip install pyfpgrowth
```

2. Import the necessary libraries:

```python
import pandas as pd
import pyfpgrowth
```

3. Load your market basket dataset:

```python
# Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')
```

4. Preprocess the data:

```python
# Assuming your dataset has a 'TransactionID' and 'Product' column
transactions = df.groupby('TransactionID')['Product'].apply(list).tolist()
```

5. Mine frequent itemsets and association rules using the FP-Growth algorithm:

```python
# Set the minimum support threshold (adjust as needed)
min_support = 0.02

# Mine frequent itemsets
patterns = pyfpgrowth.find_frequent_patterns(transactions, min_support)

# Mine association rules
rules = pyfpgrowth.generate_association_rules(patterns, min_support)
```

6. Analyze and interpret the generated association rules to gain insights into market basket patterns.

The `pyfpgrowth` library provides a simple and efficient way to implement the FP-Growth algorithm for market basket analysis. Make sure to replace `'your_dataset.csv'` with the actual path to your Kaggle dataset and adjust the `min_support` parameter to suit your specific dataset and analysis goals. Additionally, you can explore other Apriori variations and libraries like `Orange3` or `mlxtend` for more options in market basket insight analysis.

Frequent Pattern Growth (FP-Growth): This algorithm is efficient in handling large datasets and can discover frequent itemsets and associations effectively.
Code:
Certainly, here's how to implement the FP-Growth algorithm for market basket analysis in Python using the `mlxtend` library:

1. Import the necessary libraries:

```python
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth
```

2. Load your market basket dataset:

```python
# Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')
```

3. Preprocess the data and convert it into a binary format (0 or 1) suitable for FP-Growth:

```python
# Assuming your dataset has a 'TransactionID' and 'Product' column
basket = (df.groupby(['TransactionID', 'Product'])['Product']
          .count().unstack().reset_index().fillna(0)
          .set_index('TransactionID'))

def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket.applymap(encode_units)
```

4. Apply FP-Growth to mine frequent itemsets:

```python
# Set the minimum support threshold (adjust as needed)
min_support = 0.02

# Mine frequent itemsets using FP-Growth
frequent_itemsets = fpgrowth(basket_sets, min_support=min_support, use_colnames=True)
```

5. Analyze the frequent itemsets to gain insights into market basket patterns. You can view the results in the `frequent_itemsets` DataFrame.

This code demonstrates how to use the FP-Growth algorithm to find frequent itemsets in your market basket dataset. Make sure to replace ``'your_dataset.csv'`` with the actual path to your Kaggle dataset and adjust the `min_support` parameter to suit your specific dataset and analysis goals.

Market Basket Optimization:

Optimization techniques can help retailers improve product placement, pricing, and promotions based on association analysis insights.

Code:
Market Basket Optimization involves finding ways to improve product placement, pricing, and promotions based on insights gained from market basket analysis. While optimization techniques can vary depending on the specific goals and data, here's a basic example of how you might approach market basket optimization using Python:

1. Import the necessary libraries:

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
```

2. Load your market basket dataset:

```python
# Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')
```

3. Preprocess the data and convert it into a binary format (0 or 1) suitable for association rule mining:

```python
# Assuming your dataset has a 'TransactionID' and 'Product' column
basket = (df.groupby(['TransactionID', 'Product'])['Product']
          .count().unstack().reset_index().fillna(0)
          .set_index('TransactionID'))

def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket.applymap(encode_units)
```

4. Mine association rules to identify product associations:

```python
# Set the minimum support and confidence thresholds (adjust as needed)
min_support = 0.02
min_confidence = 0.5

# Mine frequent itemsets using Apriori
frequent_itemsets = apriori(basket_sets, min_support=min_support, use_colnames=True)

# Mine association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=min_confidence)
```

```

5. Analyze the association rules to identify opportunities for market basket optimization. Look for rules with high confidence and lift values that suggest strong associations between products.

6. Based on the insights from the association rules, you can optimize product placement, pricing, and promotions in various ways, such as bundling related products together, offering discounts on associated products, or adjusting store layouts to encourage cross-selling.

7. Implement and test your optimization strategies, and continually monitor their impact on sales and customer behavior.

This code provides a basic framework for market basket optimization using association rule mining. Adjust the `min_support` and `min_confidence` parameters to suit your specific dataset and optimization goals. Additionally, consider using more advanced optimization techniques, such as machine learning-based recommendation systems, to further enhance your market basket strategies.

**Cross-Market Basket Analysis**:

This involves analyzing associations across different store locations or customer segments to identify commonalities and differences.

Code:

Cross-Market Basket Analysis involves analyzing associations and patterns across different store locations or customer segments to identify commonalities and differences in market basket behavior. Here's a basic example of how to perform cross-market basket analysis using Python:

1. Import the necessary libraries:

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
```

2. Load your market basket dataset:

```python
# Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')
```

3. Preprocess the data and convert it into a binary format (0 or 1) suitable for association rule mining:

```python
# Assuming your dataset has a 'TransactionID', 'Product', and 'Market' column
basket = (df.groupby(['TransactionID', 'Market', 'Product'])['Product']
          .count().unstack().reset_index().fillna(0)
          .set_index(['TransactionID', 'Market']))

def encode_units(x):
    if x <= 0:
        return 0
```
```

```
    if x >= 1:
        return 1

basket_sets = basket.applymap(encode_units)
```

4. Mine association rules to identify product associations within each market:

```python
# Set the minimum support and confidence thresholds (adjust as needed)
min_support = 0.02
min_confidence = 0.5

# Mine frequent itemsets using Apriori for each market
frequent_itemsets = apriori(basket_sets, min_support=min_support, use_colnames=True)

# Mine association rules for each market
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=min_confidence)
```

5. Analyze the association rules within each market to identify common patterns and trends.

6. Perform cross-market analysis by comparing rules and patterns between different markets. You can do this by comparing rule metrics (e.g., lift, confidence) or by visualizing common and unique product associations.

7. Based on the insights from cross-market analysis, tailor your marketing and product strategies to each market's unique characteristics. For example, you might adjust product offerings, promotions, or pricing based on market-specific behavior.

This code provides a basic framework for cross-market basket analysis using association rule mining. Adjust the `min_support`, `min_confidence`, and preprocessing steps to match your specific dataset and analysis goals. Additionally, consider using more advanced statistical and machine learning techniques for a deeper cross-market analysis if needed.

Deep Learning Approaches:

Neural networks and deep learning models can be applied to market basket data for more complex pattern recognition and prediction.

Code:

Deep learning approaches can be applied to market basket analysis using neural networks or deep learning models. One common approach is to use recurrent neural networks (RNNs) or more specifically, long short-term memory networks (LSTMs) for sequence data like transaction histories. Below is a simplified example using Python and TensorFlow/Keras to demonstrate how you can apply deep learning to market basket analysis:

1. Import the necessary libraries:

```python
import pandas as pd
import numpy as np
```

```
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
```

2. Load your market basket dataset:

```python
# Replace 'your_dataset.csv' with your Kaggle dataset path
df = pd.read_csv('your_dataset.csv')
```

3. Preprocess the data:

```python
# Assuming your dataset has a 'TransactionID' and 'Product' column
transactions = df.groupby('TransactionID')['Product'].apply(list).tolist()

# Use MultiLabelBinarizer to one-hot encode products per transaction
mlb = MultiLabelBinarizer()
basket_encoded = mlb.fit_transform(transactions)

# Split the data into train and test sets
X_train, X_test = train_test_split(basket_encoded, test_size=0.2, random_state=42)
```

4. Define and train an LSTM-based deep learning model:

```python
model = keras.Sequential()
model.add(keras.layers.Embedding(input_dim=len(mlb.classes_), output_dim=128))
model.add(keras.layers.LSTM(128))
model.add(keras.layers.Dense(len(mlb.classes_), activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, X_train, epochs=10, batch_size=32)
```

5. Use the trained model for market basket analysis:

```python
# Make predictions on the test set
predictions = model.predict(X_test)

# You can analyze the predictions to identify product associations or make recommendations.
# For example, find products with the highest predicted probability for a given basket.
```

```
basket_indices = np.argmax(predictions, axis=1)
predicted_products = mlb.classes_[basket_indices]
```

6. Analyze the predictions, identify product associations, and use the model for recommendations or further market basket analysis.

This code provides a simplified example of how to use deep learning, specifically an LSTM-based model, for market basket analysis. Deep learning approaches can capture complex patterns in transaction data, but they require careful data preprocessing and tuning. You may need to adapt the model architecture, hyperparameters, and training process to your specific dataset and analysis goals.
Summery:
The choice of technique depends on the specific goals and dataset characteristics. Implementing these advanced methods can uncover valuable insights to enhance sales, customer experience, and marketing strategies in retail and e-commerce businesses.

visualization tools

Using visualization tools for market basket analysis can significantly enhance insights and make your presentations more impactful. Here's how you can leverage visualization tools:

1. ****Bar Charts****:
Create bar charts to display the most frequently occurring items in baskets. This helps identify popular products and potential cross-selling opportunities.

2. ****Heatmaps***
: Heatmaps can show item co-occurrence and association strength. Darker cells indicate stronger associations, helping you spot hidden patterns.

3. ****Scatter Plots****: Visualize the relationship between two or more items. Scatter plots can reveal interesting insights, such as which items are often purchased together.

4. ****Sankey Diagrams****: Sankey diagrams are excellent for displaying the flow of items between different categories or departments in a store. They show how items transition from one category to another.

5. ****Tree Maps****: Use tree maps to represent hierarchical structures in your data, such as categories and subcategories. This can help you identify which categories have the most significant impact on sales.

6. ****Network Graphs****: Visualize item relationships as a network. Each item is a node, and the connections between items represent associations. This can uncover complex patterns in your data.

7. ****Word Clouds****: Create word clouds to display frequently occurring item names. Larger words represent more popular items, making it easy to spot trends.

8. ****Time Series Plots****: If you have data over time, use time series plots to analyze how basket contents change seasonally or with trends.

9. ****Geospatial Maps**:** If applicable, use maps to visualize where certain items are most frequently purchased. This can be helpful for location-based insights.

10. **Interactive Dashboards**: Build interactive dashboards using tools like Tableau or Power BI. These allow users to explore the data and drill down into specific insights.

Remember to choose the visualization that best fits your data and the insights you want to convey. Also, ensure that your visuals are clear and easy to understand, as the goal is to communicate complex market basket insights effectively to your audience.

Building loading and preprocessing the data set

Loading and preprocessing a dataset for market basket analysis involves several steps. Here's a high-level overview of the process:

1. **Data Collection**: Gather transaction data that includes a list of items purchased by customers. This data is essential for market basket analysis.

2. **Data Cleaning**: Remove any inconsistencies or errors in the data, such as missing values or duplicate transactions.

3. **Transaction Identification**: Group the data into transactions, where each transaction is a set of items bought together by a customer.

4. **One-Hot Encoding**: Convert the transaction data into a binary format (0s and 1s) using one-hot encoding. Each column represents an item, and each row represents a transaction. If an item is present in a transaction, it's marked as 1; otherwise, it's marked as 0.

5. **Support Calculation**: Calculate the support for each itemset. Support measures how frequently an itemset appears in transactions. It's an essential metric for identifying frequent itemsets.

6. **Frequent Itemset Generation**: Use algorithms like Apriori or FP-growth to find frequent itemsets, which are combinations of items that appear together frequently.

7. **Association Rule Mining**: Generate association rules from frequent itemsets. These rules show relationships between items, often in the form of "if-then" statements.

8. **Filtering and Interpretation**: Filter and interpret the generated rules based on metrics like confidence and lift. This step helps in identifying actionable insights.

9. **Visualization**: Visualize the results to make them more understandable. Tools like scatter plots, network graphs, or word clouds can be used to represent the association rules and item relationships.

The choice of tools and libraries for these tasks can vary, but Python libraries like Pandas, NumPy, and scikit-learn are often used for data manipulation and analysis. For more specialized tasks like association rule mining, you might use libraries like mlxtend or specialized software like RapidMiner or Weka.

Keep in mind that the exact steps and tools you use may vary based on the specific dataset and analysis goals.

Data Preprocessing

Importing Required Libraries

```
import numpy as np  # Import numpy library for efficient array operations
import pandas as pd  # Import pandas library for data processing
import matplotlib.pyplot as plt  # Import matplotlib.pyplot for data visualization
```

Data Loading

Retrieving and Loading the Dataset

```
df = pd.read_csv('..//content/market basket .csv', sep=';',parse_dates=['Date'])
```

```
df.head()
```

Output

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Country
0	536365	WHITE HANGING HEART T-LIGHT HOLDER	6.0	2010-01-12 08:26:00	2,55	17850.0	United Kingdom
1	536365	WHITE METAL	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

		LANTERN					
2	536365	CREAM CUPID HEARTS COAT HANGER	8.0	2010-01-12 08:26:00	2,75	17850.0	United Kingdom
3	536365	KNITTED UNION FLAG HOT WATER BOTTLE	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom
4	536365	RED WOOLLY HOTTIE WHITE HEART.	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

```
# Convert the 'Price' column to float64 data type after replacing commas with dots

df['Price'] = df['Price'].str.replace(',', '.').astype('float64')

# Display the information about the DataFrame which is to provide an overview of the DataFrame's structure and column data types.

df.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>

RangeIndex: 451856 entries, 0 to 451855

Data columns (total 7 columns):

 #   Column      Non-Null Count   Dtype

---  ------      --------------   -----

 0   BillNo      451856 non-null  object

 1   Itemname    450457 non-null  object

 2   Quantity    451856 non-null  int64

 3   Date        451856 non-null  datetime64[ns]

 4   Price       451856 non-null  float64

 5   CustomerID  337788 non-null  float64

 6   Country     451855 non-null  object

dtypes: datetime64[ns](1), float64(2), int64(1), object(3)

memory usage: 24.1+ MB

# Calculate the number of missing values for each column and sort them in
descending order

df.isna().sum().sort_values(ascending=False)
```

Output;

```
CustomerID    114068
```

```
Itemname         1399
Country             1
BillNo              0
Quantity            0
Date                0
Price               0
dtype: int64
```

```
Calculate the total price by multiplying the quantity and price columns
df['Total_Price'] = df.Quantity * df.Price
df.describe(include='all')
```

Output:

<ipython-input-24-174ba9bf1a5c>:1: FutureWarning: Treating datetime data as categorical rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Specify `datetime_is_numeric=True` to silence this warning and adopt the future behavior now.

df.describe(include='all')

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Total_Price
count	451856.0	450457	451856.000000	451856	451856.000000	337788.000000	451856.000000

unique	19231.0	4131	NaN	17479	NaN	NaN	NaN
top	573585.0	WHITE HANGING HEART T-LIGHT HOLDER	NaN	2011-10-31 14:41:00	NaN	NaN	NaN
freq	1114.0	2070	NaN	1114	NaN	NaN	NaN
first	NaN	NaN	NaN	2010-01-12 08:26:00	NaN	NaN	NaN
last	NaN	NaN	NaN	2011-12-10 17:19:00	NaN	NaN	NaN
mean	NaN	NaN	10.198386	NaN	3.823006	15313.059783	19.771630
std	NaN	NaN	122.40985	NaN	43.546399	1718.808141	151.012823

min	NaN	NaN	-9600.000000	NaN	-11062.060000	181.000000	-11062.060000
25%	NaN	NaN	1.000000	NaN	1.250000	13924.000000	3.750000
50%	NaN	NaN	3.000000	NaN	2.080000	15265.000000	9.900000
75%	NaN	NaN	11.000000	NaN	4.130000	16817.000000	17.700000
max	NaN	NaN	74215.000000	NaN	13541.330000	18287.000000	77183.600000

```
# Print the number of unique countries in the 'Country' column

print("Number of unique countries:", df['Country'].nunique())


# Calculate and print the normalized value counts of the top 5 countries
in the 'Country' column

print(df['Country'].value_counts(normalize=True)[:5])
```

Output:

```
Number of unique countries: 30
```

```
United Kingdom    0.931721
Germany           0.017827
France            0.016311
Spain             0.005035
Netherlands       0.004730
Name: Country, dtype: float64
```

Considering that the majority of transactions (approximately 93%) in the dataset originate from the UK, the 'Country' column may not contribute significant diversity or variability to the analysis. Therefore, we can choose to remove the 'Country' column from the DataFrame df. we indicate that we want to drop a column, This step allows us to focus on other attributes that may provide more valuable insights for our analysis.

```
# Delete the 'Country' column from the DataFrame
df.drop('Country', axis=1, inplace=True)
```

```
# Filter the DataFrame to display rows where 'BillNo' column contains non-digit values
df[df['BillNo'].str.isdigit() == False]
```

Output:

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Total_Price
288772	A563185	Adjust bad debt	1.0	2011-12-08 14:50:00	11062.06	NaN	11062.06
288773	A563186	Adjust bad debt	1.0	2011-12-08 14:51:00	-11062.06	NaN	-11062.06
288774	A563187	Adjust bad debt	1.0	2011-12-08 14:52:00	-11062.06	NaN	-11062.06

Since the item name "Adjust bad debt" was filled accidentally and does not provide any useful information for our analysis, we can choose to remove the corresponding rows from the DataFrame. The code snippet above filters the DataFrame df to retain only the rows where the 'Itemname' column does not contain the value "Adjust bad debt". This operation effectively eliminates the rows associated with the accidental data entry, ensuring the dataset is free from this irrelevant item name.

```
# Remove rows where the 'Itemname' column contains "Adjust bad debt"

df = df[df['Itemname'] != "Adjust bad debt"]
```

```
# Here to check if all BillNo doesn't inculde letters

df['BillNo'].astype("int64")
```

Output:

```
0         536365

1         536365

2         536365

3         536365

4         536365

           ...

465135    577504

465136    577504

465137    577504

465138    577504

465139    577504

Name: BillNo, Length: 465137, dtype: int64
```

```
# Calculate the sum of 'Price' for rows where 'Itemname' is missing

df[df['Itemname'].isna()] ['Price'].sum()
```

Output:

```
0.0
```

Exploring Rows with Missing Item Names:

To investigate the data where the 'Itemname' column has missing values, we can filter the dataset to display only those rows. This subset of the data will provide insights into the records where the item names are not available.

```
# Filter the DataFrame to display rows where 'Itemname' is missing

df[df['Itemname'].isna()]
```

Output:

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Total_Price
613	536414	NaN	56.0	2010-01-12 11:52:00	0.0	NaN	0.0
1937	536545	NaN	1.0	2010-01-12 14:32:00	0.0	NaN	0.0
1938	536546	NaN	1.0	2010-01-12 14:33:00	0.0	NaN	0.0
1939	536547	NaN	1.0	2010-01-12 14:33:00	0.0	NaN	0.0

1940	536549	NaN	1.0	2010-01-12 14:34:00	0.0	NaN	0.0
...	...	...	...	...	...	...	...
461979	577264	NaN	3.0	2011-11-18 12:32:00	0.0	NaN	0.0
461980	577265	NaN	2.0	2011-11-18 12:33:00	0.0	NaN	0.0
462301	577306	NaN	12.0	2011-11-18 13:06:00	0.0	NaN	0.0
462902	577339	NaN	9.0	2011-11-18 14:57:00	0.0	NaN	0.0
462903	577340	NaN	40.0	2011-11-18 14:57:00	0.0	NaN	0.0

1410 rows × 7 columns

Upon examining the data where the 'Itemname' column has missing values, it becomes evident that these missing entries do not contribute any meaningful information. Given that the item names are

not available for these records, it suggests that these instances may not be crucial for our analysis. As a result, we can consider these missing values as non-significant and proceed with our analysis without incorporating them.

```
# Filter the DataFrame to exclude rows where 'Itemname' is missing (not NaN)
df = df[df['Itemname'].notna()]

# Print the number of unique items in the 'Itemname' column
print("Number of unique items:", df['Itemname'].nunique())

# Calculate and print the normalized value counts of the top 5 items in the 'Itemname' column
print(df['Itemname'].value_counts(normalize=True)[:5])
```

Output:

```
Number of unique items: 4144
WHITE HANGING HEART T-LIGHT HOLDER    0.004546
JUMBO BAG RED RETROSPOT               0.004194
REGENCY CAKESTAND 3 TIER              0.003864
PARTY BUNTING                         0.003489
LUNCH BAG RED RETROSPOT               0.003170
Name: Itemname, dtype: float64
```

A curious observation has caught our attention—the presence of a negative quantity in the 515,623rd row.

we are intrigued by the existence of negative quantities within the dataset. To gain a deeper understanding of this phenomenon, we focus our attention on these specific instances and aim to uncover the underlying reasons behind their occurrence. Through this exploration, we expect to

gain valuable insights into the nature of these negative quantities and their potential impact on our analysis. Our investigation aims to reveal the intriguing stories that lie within this aspect of the data.

```
# Filter the DataFrame to display rows where 'Quantity' is less than 1
df[df['Quantity'] < 1]
```

Output:

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Total_Price	
7122	537032	?	-30.0	2010-03-12 16:50:00	0.0	NaN	-0.0	
12926	537425	check	-20.0	2010-06-12 15:35:00	0.0	NaN	-0.0	
12927	537426	check	-35.0	2010-06-12 15:36:00	0.0	NaN	-0.0	
12973	537432	damages	-43.0	2010-06-12 16:10:00	0.0	NaN	-0.0	
20844	538072	faulty	-13.0	2010-09-12 14:10:00	0.0	NaN	-0.0	

...	...	...	...	...	...	...	...
460954	577123	check	-63.0	2011-11-17 18:34:00	0.0	NaN	-0.0
460955	577124	check	-327.0	2011-11-17 18:41:00	0.0	NaN	-0.0
462299	577304	check	-152.0	2011-11-18 13:04:00	0.0	NaN	-0.0
462300	577305	check	-21.0	2011-11-18 13:06:00	0.0	NaN	-0.0
462302	577307	check	-313.0	2011-11-18 13:06:00	0.0	NaN	-0.0

409 rows × 7 columns

```
# Remove rows where 'Quantity' is less than 1
df = df[df['Quantity'] >= 1]
```

Next, we turn our attention to the presence of missing values in the 'CustomerID' column. By investigating these missing values, we aim to identify any potential issues or data quality concerns associated with them. Analyzing the impact of missing 'CustomerID' values will help us assess the

completeness and reliability of the dataset, enabling us to make informed decisions on handling or imputing these missing values. Let's dive deeper into this aspect and gain a comprehensive understanding of any issues related to missing 'CustomerID' values.

```
# Select a random sample of 30 rows where 'CustomerID' is missing
df[df['CustomerID'].isna()].sample(30)
```

Output

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Total_Price	
196380	554514	PAPER POCKET TRAVELING FAN	1.0	2011-05-24 15:58:00	0.83	NaN	0.83	
67851	541975	RED RETROS POT BOWL	24.0	2011-01-24 14:24:00	1.25	NaN	30.00	
86621	543909	RED HARMONICA IN BOX	1.0	2011-02-14 12:32:00	2.46	NaN	2.46	

312028	565396	SET/6 RED SPOTTY PAPER PLATES	3.0	2011-02-09 16:39:00	0.83	NaN	2.49
78692	543097	RED HEARTS LIGHT CHAIN	1.0	2011-03-02 11:41:00	9.13	NaN	9.13
418522	574074	LETTER HOLDER HOME SWEET HOME	1.0	2011-02-11 15:33:00	7.46	NaN	7.46
386865	571508	MODERN FLORAL STATIONERY SET	2.0	2011-10-17 15:27:00	2.46	NaN	4.92
26113	538524	PORCELAIN BUTTERFLY OIL BURNER	1.0	2010-12-13 09:35:00	5.91	NaN	5.91

189643	553849	SET OF 6 SPICE TINS PANTRY DESIGN	1.0	2011-05-19 12:54:00	3.95	NaN	3.95
89919	544208	ILLUSTRATED CAT BOWL	1.0	2011-02-17 10:34:00	4.96	NaN	4.96
413702	573585	MULTICOLOUR CONFETTI IN TUBE	1.0	2011-10-31 14:41:00	1.63	NaN	1.63
60980	541508	JUMBO BAG RED RETROSPOT	1.0	2011-01-18 16:06:00	4.13	NaN	4.13
305731	564817	GIRLS ALPHABET IRON ON PATCHES	19.0	2011-08-30 12:02:00	0.42	NaN	7.98

37057	539492	GREEN REGENCY TEACUP AND SAUCER	1.0	2010-12-20 10:14:00	5.91	NaN	5.91
440011	575739	CLOTHES PEGS RETROS POT PACK 24	1.0	2011-11-11 09:05:00	3.29	NaN	3.29
203694	555336	BAG 125g SWIRLY MARBLES	2.0	2011-02-06 11:13:00	0.42	NaN	0.84
245994	559338	TRIPLE WIRE HOOK IVORY HEART	1.0	2011-07-07 16:30:00	4.13	NaN	4.13
247195	559491	JUMBO STORAGE BAG SUKI	5.0	2011-08-07 13:53:00	4.13	NaN	20.65

430039	574950	WOODLAND STICKERS	1.0	2011-08-11 09:29:00	1.63	NaN	1.63
91940	544434	BATHROOM METAL SIGN	1.0	2011-02-18 16:12:00	1.25	NaN	1.25
453296	576618	DIAMANTE HAIR GRIP PACK/2 RUBY	1.0	2011-11-15 17:00:00	1.65	NaN	1.65
245987	559338	RECYCLED ACAPULCO MAT BLUE	1.0	2011-07-07 16:30:00	16.63	NaN	16.63
167542	551718	JUMBO SHOPPER VINTAGE RED PAISLEY	2.0	2011-03-05 16:06:00	4.13	NaN	8.26

460086	577078	TOILET METAL SIGN	1.0	2011-11-17 15:17:00	1.25	NaN	1.25
242816	559055	BOYS VINTAGE TIN SEASIDE BUCKET	4.0	2011-05-07 17:09:00	2.46	NaN	9.84
247946	559515	GINGHAM HEART DOORSTOP RED	1.0	2011-08-07 15:58:00	8.29	NaN	8.29
67763	541971	PURPLE BOUDICCA LARGE BRACELET					

This sample can provide us with a glimpse into the specific instances where 'CustomerID' is missing, aiding us in further analysis or decision-making related to handling these missing values.

Upon analyzing a sample of rows where the 'CustomerID' is missing, it appears that there is no discernible pattern or specific reason behind the absence of these values. This observation suggests that the missing 'CustomerID' entries were not filled accidentally or due to a systematic issue. Instead, it is possible that these missing values occur naturally in the dataset, without any particular significance or underlying cause.

Identifying Issues in the Price Column: Ensuring Data Quality

In our analysis, we shift our focus to the 'Price' column and investigate it for any potential issues or anomalies. By thoroughly examining the data within this column, we aim to identify any irregularities, inconsistencies, or outliers that may affect the overall quality and integrity of the dataset. Analyzing the 'Price' column is crucial in ensuring accurate and reliable pricing information for our analysis. Let's dive deeper into the 'Price' column and uncover any issues that may require attention.

Counting the number of rows where the price is zero

zero_price_count = len(df[df['Price'] == 0])

print("Number of rows where price is zero:", zero_price_count)

Counting the number of rows where the price is negative

negative_price_count = len(df[df['Price'] < 0])

print("Number of rows where price is negative:", negative_price_count)

Output:

Number of rows where price is zero: 547

Number of rows where price is negative

our attention now turns to the presence of zero charges in the 'Price' column. It is important to explore instances where products were offered free of cost, as this information can provide valuable insights into promotional activities, giveaways, or other unique aspects of the dataset. By examining the data related to zero charges in the 'Price' column, we can gain a deeper understanding of these transactions and their potential impact on our analysis. Let's delve into the details of these zero-priced transactions and uncover any significant findings.

```
# Selecting a random sample of 20 rows where the price is zero

df[df['Price'] == 0].sample(20)
```

Output

	BillNo	Itemname	Quantity	Date	Price	CustomerID	Total_Price
419600	574138	BISCUIT TIN VINTAGE CHRISTMAS	216.0	2011-03-11 11:26:00	0.0	12415.0	0.0
40241	539856	RED RETROSPOT CHARLOTTE BAG	2.0	2010-12-22 14:41:00	0.0	NaN	0.0
301848	564530	OWL DOORSTOP	1.0	2011-08-25 14:57:00	0.0	NaN	0.0
233693	558340	KINGS CHOICE BISCUIT TIN	3.0	2011-06-28 14:01:00	0.0	NaN	0.0

301840	564530	FRENCH BLUE METAL DOOR SIGN 8	3.0	2011-08-25 14:57:00	0.0	NaN	0.0
40285	539856	AIRLINE BAG VINTAGE JET SET RED	3.0	2010-12-22 14:41:00	0.0	NaN	0.0
101119	545176	GLASS JAR KINGS CHOICE	2.0	2011-02-28 14:19:00	0.0	NaN	0.0
462450	577314	SET OF 2 TRAYS HOME SWEET HOME	2.0	2011-11-18 13:23:00	0.0	12444.0	0.0
412073	573495	check	184.0	2011-10-31 11:58:00	0.0	NaN	0.0

37190	539494	?	752.0	2010-12-20 10:36:00	0.0	NaN	0.0
14040	537534	BOX OF 24 COCKTAIL PARASOLS	2.0	2010-07-12 11:48:00	0.0	NaN	0.0
341902	567920	found	5.0	2011-09-22 17:21:00	0.0	NaN	0.0
20555	538071	PACK OF 6 BIRDY GIFT TAGS	1.0	2010-09-12 14:09:00	0.0	NaN	0.0
40253	539856	BISCUIT TIN VINTAGE RED	1.0	2010-12-22 14:41:00	0.0	NaN	0.0
40261	539856	FRENCH BLUE METAL	1.0	2010-12-22 14:41:00	0.0	NaN	0.0

		DOOR SIGN 6					
415931	573880	check	48.0	2011-01-11 13:21:00	0.0	NaN	0.0
387664	571633	found	100.0	2011-10-18 11:27:00	0.0	NaN	0.0
233713	558340	RECIPE BOX BLUE SKETCH BOOK DESIGN	1.0	2011-06-28 14:01:00	0.0	NaN	0.0
275167	561916	Manual	1.0	2011-01-08 11:44:00	0.0	15581.0	0.0
233696	558340	OWL DOORSTOP	1.0	2011-06-28 14:01:00	0.0	NaN	0.0

Removing Rows with Zero Price: Eliminating Misleading Data Entries

Upon reviewing the sample of rows where the price is zero, we have identified that these entries might provide misleading or inaccurate information for our analysis. Therefore, it is prudent to

proceed with removing these rows from the dataset to ensure the integrity and reliability of our analysis.

```
# Remove rows where the price is zero

df = df[df['Price'] != 0]
```

Data Understanding: Exploring and Interpreting the Dataset

In the data analysis process, data understanding plays a crucial role in gaining insights and formulating meaningful conclusions. By thoroughly examining the dataset, we aim to understand its structure, contents, and underlying patterns. This understanding empowers us to make informed decisions regarding data cleaning, feature engineering, and subsequent analysis steps.

Key aspects of data understanding include:

Exploring the Dataset: We investigate the dataset's dimensions, such as the number of rows and columns, to gauge its size and complexity. Additionally, we examine the data types of each column to understand the nature of the variables.

Assessing Data Quality: We scrutinize the data for inconsistencies, outliers, or other data quality issues that may require attention. Addressing these issues ensures the reliability and accuracy of the data.

Identifying Relationships: We analyze the relationships between variables by examining correlations, associations, or dependencies. This analysis allows us to uncover meaningful connections that can drive insights and guide our analysis.

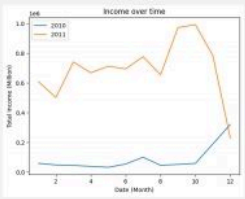
Detecting Patterns and Trends: We look for recurring patterns, trends, or distributions within the data. This step can reveal valuable information about customer behavior, market dynamics, or other relevant factors.

By thoroughly understanding the dataset, we lay the foundation for meaningful data analysis and generate insights that contribute to informed decision-making and problem-solving.

```
# Grouping the data by month and summing the total price for the year 2010

df[df["Date"].dt.year ==
2010].groupby(df["Date"].dt.month)["Total_Price"].sum().plot()


# Grouping the data by month and summing the total price for the year 2011

df[df["Date"].dt.year ==
2011].groupby(df["Date"].dt.month)["Total_Price"].sum().plot()


# Adding legend and plot labels

plt.legend(["2010", "2011"])

plt.title("Income over time")

plt.ylabel('Total Income (Million)')

plt.xlabel("Date (Month)")
```

Output:

```
Text(0.5, 0, 'Date (Month)')
```



The code snippet above creates a line plot to visualize the income over time for the years 2010 and 2011. First, the data is filtered based on the year using the dt.year attribute of the 'Date' column. The data is then grouped by month, and the 'Total_Price' column is summed. Two line plots are created, one for each year, showing the monthly total income. The legend is added to indicate the respective years, and the plot is labeled with a title, y-axis label, and x-axis label. This visualization allows us to observe the trend and compare the income between the two years.

Upon observing the line plot of income over time for the years 2010 and 2011, it becomes apparent that the sales remained relatively stable and consistent until October 2010. This suggests that the business was growing steadily during this period, as the sales continued to increase.

However, a significant drop in sales is observed in the last month of the dataset. This sudden decline indicates a notable deviation from the previously observed growth trend. Exploring the potential factors contributing to this drop becomes crucial in understanding the underlying reasons for the decline in sales during that specific period.

To verify if the data is complete for the entire last month in the dataset, we can compare the maximum date in the 'Date' column with the last day of that month. If they match, it indicates that the data is filled for the entire last month.

```
df["Date"].max()
```

Output:

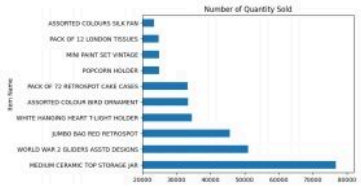
```
Timestamp('2011-12-10 17:19:00')
```

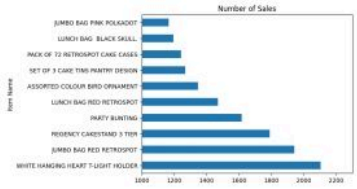
Based on the finding that the data is only available for 10 days in the last month, it becomes evident that the significant drop in sales observed during that period is likely due to the limited data rather than an actual decline in sales. The incomplete data for the last month may not provide a comprehensive representation of the sales performance during that period.

To gain a more accurate understanding of the sales trend, it is advisable to consider a broader time frame with complete data. Analyzing a more extended period that encompasses multiple months or years would provide a more reliable assessment of the sales performance and allow for more meaningful insights and conclusions.

```
# Plotting the top 10 most sold products by quantity

df.groupby('Itemname')['Quantity'].sum().sort_values(ascending=False)[:10]
.plot(kind='barh', title='Number of Quantity Sold')

plt.ylabel('Item Name')

plt.xlim(20000, 82000)

plt.show()



# Plotting the top 10 most sold products by count

df['Itemname'].value_counts(ascending=False)[:10].plot(kind='barh',
title='Number of Sales')

plt.ylabel('Item Name')

plt.xlim(1000, 2300)

plt.show()
```

Output:





The code snippet above creates two horizontal bar plots to visualize the most sold products based on quantity and count, respectively.

In the first plot, the top 10 items are determined by summing the 'Quantity' column for each unique 'Itemname' and sorting them in descending order. The plot displays the number of quantities sold for each item.

The second plot showcases the top 10 items based on the count of sales for each unique 'Itemname'. The value_counts function counts the occurrences of each item and sorts them in descending order. The plot represents the number of times each item has been sold.

Observing the plots, we can infer that there are products that are sold more frequently (higher count) compared to others, despite having relatively lower quantities sold per transaction. This indicates the presence of items that are commonly purchased in larger quantities at once. These products might include items that are frequently bought in bulk or items that are typically sold in larger packages or quantities.

This insight highlights the importance of considering both the quantity sold and the count of sales when analyzing the popularity and demand for different products. It suggests that some items may have a higher turnover rate due to frequent purchases, while others may have a higher quantity per sale, leading to different sales patterns and customer behaviors. Understanding these dynamics can be valuable for inventory management, pricing strategies, and identifying customer preferences.

Performing association analysis Generating insights

Market basket analysis is a method or technique of data analysis for retail and marketing purpose. Market basket analysis is done to understand the purchasing behavior of customers. MBA (Market Business Analysis) is used to uncover what items are frequently brought together by the customer. Market basket analysis leads to effective sales and marketing.Market basket analysis measures the co-occurrence of products and services. Market basket analysis is only considered when there is a transaction between two or more items.
Eg:if a customer is buying bread then he is likely to buy butter, jam or milk to compliment bread.

Applications of Market Basket Analysis

Market basket analysis is applied to various fields of the retail sector in order to boost sales and generate revenue by identifying the needs of the customers and make purchase suggestions to them.

Cross-selling is basically a sales technique in which seller suggests some related product to a customer after he buys a product.
Product Placement: It refers to placing the complimentary (pen and paper)and substitute goods (tea and coffee) together so that the customer addresses the goods and will buy both the goods together.
MBA has also been used in the field of healthcare for the detection of adverse drug reactions. It produces association rules that indicates what all combinations of medications and patient characteristics lead to ADRs.
Fraud Detection: Market basket analysis is also applied to fraud detection. It may be possible to identify purchase behavior that can associate with fraud on the basis of market basket analysis data that contain credit card usage

How Market Based Analysis Works

In order to make it easier to understand, think of Market Basket Analysis in terms of shopping at a supermarket. Market Basket Analysis takes data at transaction level, which lists all items bought by a customer in a single purchase. The technique determines relationships of what

products were purchased with which other product(s). These relationships are then used to build profiles containing If-Then rules of the items purchased.
The rules are written as :

if {A} then {B} i.e. {A} => {B}

The If part of the rule (the {A} above) is known as the antecedent and the THEN part of the rule is known as the consequent (the {B} above). The antecedent is the condition and the consequent is the result.

Assosciation Rules

Association Rules are widely used to analyze retail basket or transaction data, and are intended to identify strong rules discovered in transaction data using measures of interestingness, based on the concept of strong rules.

Let I={i1,i2,i3,…,in} be a set of n attributes called items and D={t1,t2,…,tn} be the set of transactions. It is called database. Every transaction, ti in D has a unique transaction ID, and it consists of a subset of itemsets in I.
Assosciation rules are produced using algorithms like :

Apriori Algorithm
Eclat Algorithm
FP-growth Algorithm

A rule can be defined as an implication, X→Y where X and Y are subsets of I(X,Y⊆I), and they have no element in common. X and Y are the antecedent and the consequent of the rule, respectively.

Eg: {Bread,Egg}=> {Milk} ItemSet={Bread,Egg,Milk}

There are various metrics in place to help us understand the strength of assosciation between antecedent and consequent:

Support

Confidence
Lift or Correlation or interest
Leverage
Conviction

Support

It gives an idea of how frequent an itemset is in all the transactions.To say in formal terms it's the fraction of total no. of transactions in which the itemset occurs.We refer to an itemset as a "frequent itemset" if you support is larger than a specified minimum-support threshold.
supp(X−>Y)
=
(TransactionscontainingbothXandY) (TotalNo.oftransactions)
Range:[0,1] Value of support helps us identifying the rules worth for future analysis.

Confidence

It defines the likelihood of occurence of consequent on the cart given that cart already has antecedent.It signifies the likelihood of item Y being purchased when item X is purchased.

confidence(X−>Y)= support(X−>Y)
support(X)

Range:[0,1]

If confidence is 0.75 then that imples that 75%of transactions containing X also contain Y .It can also be interpreted as the conditional probability P(Y|X), i.e, the probability of finding the itemset Y in transactions given the transaction already contains X.

It has a major drawback i.e. It only takes into account the popularity of the itemset X and not the popularity of Y. If Y is equally popular as X then there will be a higher probability that a transaction containing X will also contain Y thus increasing the confidence. To overcome this drawback there is another measure called lift.

Lift

Lift gives the rise in the probability of having {Y} on the cart with the knowlede of {X} being present over the probability of having {Y} on the cart without knowlede about presence of {X}.
Lift(X−>Y)=
confidence(X−>Y)
support(Y)

Range:[0,Infinity]

It can simply be considered as correlation between the antecedent and consequent.If the value of lift is greater than 1, it means that the itemset Y is likely to be bought with itemset X, while a value less than 1 implies that itemset Y is unlikely to be bought if the itemset X is bought.

Levarage or Piatetsky-Snapiro

It computes the difference between the observed frequency of X & Y appearing together and the frequency that we would expect if A and C are independent.
Leverage(X−>Y)=support(X−>Y)
−support(X)∗support(Y)

Range:[-1,1]

If X,Y are positively correlated then we get leverage>0 ,we need such type of rules. If X,Y are negatively correlated then we get leverage<0.
If X,y are independent , then we get leverage = 0.

Conviction

It can be interpreted as the ratio of the expected frequency that X occurs without Y (that is to say, the frequency that the rule makes an incorrect prediction) if X and Y were independent divided by the observed frequency of incorrect predictions.
Conviction(X−>Y)
=
support(Y) confidence(X−>Y)

Please mark in the above equation Y means it is Y bar i.e. a bar on Y Range:[0,Infinity]
A high conviction value means that the consequent is highly depending on the antecedent. For instance, in the case of a perfect confidence score, the denominator becomes 0 (due to 1 - 1) for which the conviction score is defined as 'inf'. Similar to lift, if items are independent, the conviction is 1.

Apriori Algorithm

Apriori algorithm is a classical algorithm in data mining. It is used for mining frequent itemsets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items brought by customers in a store.Association rule learning is a prominent and a well-explored method for determining relations among variables in large databases.

Rule - generation is a two step process. First is to generate frequent item set and second is to generate rules from the considered itemset.

Generating Frequent Itemset:

One approach to find the frequent itemsets is to check all possible subsets of the given item set and check the support value of each itemset and consider only those that have support values greater than the minimum threshold support value.
Here the Apriori uses the result of antimontone property of support and makes the generation of frequent Item set faster by reducing the search space.It has two principles:

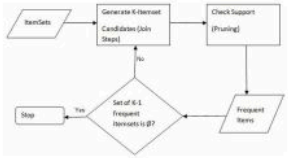
All subsets of a frequent itemset must be frequent
Similarly, for any infrequent itemset, all its supersets must be infrequent too

Apriori principle allows us to prune all supersets of an itemset which does nogt satisfy the minimum threshold condition for support.For example if {Milk,Bread} does not satify our threshold value , then the superset of {Milk,Bread} will also not cross the threshold value there by we can just prune them away i.e. do not consider the itemsets that wil be generated from the
{Milk,Bread}.

Totally 3 major steps are involved here:

Generate all frequent itemsets each satisfying the minimum threshold and having only one item let it be L1. Next use self join and generate all possible combinations of L1 and now let the result be L2.
At each step as we keep on generating candidate itemsets, for each candidate we scan
entire database so as to know its support and remove the candidates that do not satisfy minimum threshold
Here To reduce the no of comparisions, store the generated candidate items in a Hash Tree,Instead of matching each of candidate itemsets against each transaction,match each transaction with the candidates in hash tree(there by we can enhance the speed of apriori using this method)
In similar way create Lk from Lk-1 until the point where we are unable to apply selfjoin.

This approach of extending a frequent itemset one at a time is called the “bottom up” approach.



Generating all possible rules from Frequent Itemsets

If n items are in set I , no of possible assosciation rules possible are 3<supn</sup>- 2^{n+1} + 1.It becomes computationally expensive to generate all the rules and there is no meaning in genearting all that many no. of rules.

So apriori simplifies this approach by following some methodology,

Rules are formed by binary partition of each itemset.From a list of all possible candidate rules, we aim to identify rules that fall above the minimum confidence level.Just like antimontone property of support, confidence of rules generated from same itemset also follow the anti montone property. It's antimontone w.r.t no. of elements in consequent.

=> CONF(A,B,C -> D) >= CONF(B,C -> A,D) >= CONF(C -> A,B,D)

On the basis of this rules are generated.

If you want to refer further on Advanced Apriori Algorithms, Please refer to Advanced Apriori Algorithms.

Apriori uses a breadth-first search strategy to count the support of itemsets and uses a candidate generation function which exploits the downward closure property of support.

Pros of the Apriori algorithm:

- It is an easy-to-implement and easy-to-understand algorithm.
- It can be used on large itemsets.
- Cons of the Apriori Algorithm:
    - Sometimes, it may need to find a large number of candidate rules which can be computationally expensive.
    - Calculating support is also expensive because it has to go through the entire database.

Now let's work on a dataset ...

Importing Libraries

```
Data manipulation libraries
import pandas as pd
import numpy as np

#Visualizations
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("dark")
import squarify
import matplotlib

#for market basket analysis (using apriori)
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

#for preprocessing
from mlxtend.preprocessing import TransactionEncoder


#to print all the interactive output without resorting to print, not only the last result.
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Importing data

```
data = pd.read_csv('/content/market basket .csv', sep=';',parse_dates=['Date'])
data.head()
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
BillNo  Itemname       Quantity        Date    Price   CustomerID      Country
0       536365 WHITE HANGING HEART T-LIGHT HOLDER   6       2010-01-12 08:26:00     2,55
17850.0         United Kingdom
1       536365 WHITE METAL LANTERN      6       2010-01-12 08:26:00     3,39    17850.0
```

United Kingdom
2 536365 CREAM CUPID HEARTS COAT HANGER 8 2010-01-12 08:26:00 2,75
17850.0 United Kingdom
3 536365 KNITTED UNION FLAG HOT WATER BOTTLE 6 2010-01-12 08:26:00 3,39
17850.0 United Kingdom
4 536365 RED WOOLLY HOTTIE WHITE HEART. 6 2010-01-12 08:26:00 3,39
17850.0 United Kingdom

```
data.shape
```

Output
(40032, 7)

```
data.head()
```

Output
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
and should_run_async(code)
BillNo Itemname Quantity Date Price CustomerID Country
0 536365 WHITE HANGING HEART T-LIGHT HOLDER 6.0 2010-01-12 08:26:00 2,55
17850.0 United Kingdom
1 536365 WHITE METAL LANTERN 6.0 2010-01-12 08:26:00 3,39 17850.0
United Kingdom
2 536365 CREAM CUPID HEARTS COAT HANGER 8.0 2010-01-12 08:26:00 2,75
17850.0 United Kingdom
3 536365 KNITTED UNION FLAG HOT WATER BOTTLE 6.0 2010-01-12 08:26:00 3,39
17850.0 United Kingdom
4 536365 RED WOOLLY HOTTIE WHITE HEART. 6.0 2010-01-12 08:26:00 3,39
17850.0 United Kingdom.

data.tail()
Output
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
and should_run_async(code)
BillNo Itemname Quantity Date Price CustomerID Country
66914 541871 HAND TOWEL PALE BLUE W FLOWERS 1 2011-01-24 09:41:00 4,13
NaN United Kingdom
66915 541871 PINK BUTTERFLY HANDBAG W BOBBLES 1 2011-01-24 09:41:00 4,13
NaN United Kingdom
66916 541871 ANTIQUE SILVER TEA GLASS ETCHED 2 2011-01-24 09:41:00 2,46
NaN United Kingdom
66917 541871 BOX OF 6 ASSORTED COLOUR TEASPOONS 1 2011-01-24 09:41:00 3,29
NaN United Kingdom
66918 541871 SET OF 72 PINK HEART PAPER DOILIES 8 2011-01-24 09:41:00 2,92
NaN Un

Here we can find that data needs a lot of preprocessing.So let's preprocess the data.We use the TransactionEncoder() of mlxtend.preprocessing to do this work for us, if needed even we can implement the function i will provide the alternative as well.The TransactionEncoder() is an Encoder class for transaction data in Python list.It finds out what are all the different products in the transactions and will

assign each transaction a list which contains a boolean array where each index represnts the corresponding product whether purchased in the transaction or not i.e. True or False.

It needs input as a python list of lists, where the outer list stores the n transactions and the inner list stores the items.

It returns the one-hot encoded boolean array of the input transactions, where the columns represent the unique items found in the input array in alphabetic order. For further details you can refer its documentation:TransactionEncoder
converting into required format of TransactionEncoder()

```
trans=[]
for i in range(0,7501):
   trans.append([str(data.values[i,j]) for j in range(0,20)])

trans=np.array(trans)

print(trans.shape)
```

Output

(7501, 20)

Using TransactionEncoder

```
t=TransactionEncoder()
data=t.fit_transform(trans)
data=pd.DataFrame(data,columns=t.columns_,dtype=int)

data.shape
```

Output

(7501, 121)

```
data.drop('nan',axis=1,inplace=True)

#now lets check shape
data.shape

#lets verify whether nan is present in columns
'nan' in data.columns
#so its proved that nan is not in columns
```

Output

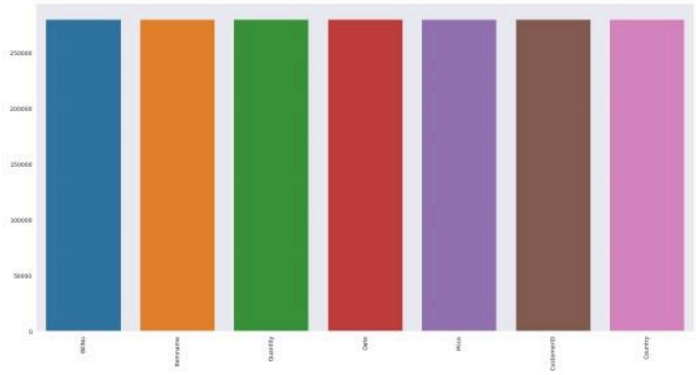
```
(133459, 7)False

data.head()

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
BillNoItemname    Quantity    Date  Price CustomerID  Country
0     1     1     1     1     1     1     1
1     1     1     1     1     1     1     1
2     1     1     1     1     1     1     1
3     1     1     1     1     1     1     1
4     1     1     1     1     1     1     1
```

Data Visualizations

```
Lets consider the top 20 items purchased freequently
r=data.sum(axis=0).sort_values(ascending=False)[:20]
#altering the figsize
plt.figure(figsize=(20,10))
s=sns.barplot(x=r.index,y=r.values)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
<Figure size 2000x1000 with 0 Axes>[Text(0, 0, 'BillNo'),
 Text(1, 0, 'Itemname'),
 Text(2, 0, 'Quantity'),
 Text(3, 0, 'Date'),
 Text(4, 0, 'Price'),
 Text(5, 0, 'CustomerID'),
 Text(6, 0, 'Country')]
```



```
We can find that mineral water is the most purchased item from the store, we
may advice that mineral water must be always in the stock not only that mostly
we can see from the above graph what 20 items are being frequently purchased.
# create a color palette, mapped to these values
my_values=r.values
cmap = matplotlib.cm.Blues
mini=min(my_values)
maxi=max(my_values)
norm = matplotlib.colors.Normalize(vmin=mini, vmax=maxi)
colors = [cmap(norm(value)) for value in my_values]


#treemap of top 20 frequent items
plt.figure(figsize=(10,10))
squarify.plot(sizes=r.values, label=r.index, alpha=.7,color=colors)
plt.title("Tree map of top 20 items")
plt.axis('off')
```

Output


```
let us return items and ietmsets with atleast 5% support:
freq_items=apriori(data,min_support=0.05,use_colnames=True)
```

```
freq_items
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

	support	itemsets
0	1.0	(BillNo)
1	1.0	(Itemname)
2	1.0	(Quantity)
3	1.0	(Date)
4	1.0	(Price)
...	...	...
122	1.0	(Price, Itemname, Country, CustomerID, Quantit...
123	1.0	(Date, Price, Itemname, Country, CustomerID, B...
124	1.0	(Date, Price, Country, CustomerID, Quantity, B...
125	1.0	(Date, Price, Country, CustomerID, Quantity, I...
126	1.0	(Date, Price, Itemname, Country, CustomerID, Q...

```
127 rows × 2 columns
```

```
res=association_rules(freq_items,metric="lift",min_threshold=1.3)
```

```
res
```

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
 and should_run_async(code)

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric

```
Selecting and Filtering the Results
frequent_itemsets = apriori(data, min_support = 0.05, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
 and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
 warnings.warn(

	support	itemsets	length

0	1.0	(BillNo)	1
1	1.0	(Itemname)	1
2	1.0	(Quantity)	1
3	1.0	(Date)	1
4	1.0	(Price)	1
...	...	...	...
122	1.0	(Price, Itemname, Country, CustomerID, Quantit...	6
123	1.0	(Date, Price, Itemname, Country, CustomerID, B...	6
124	1.0	(Date, Price, Country, CustomerID, Quantity, B...	6
125	1.0	(Date, Price, Country, CustomerID, Quantity, I...	6
126	1.0	(Date, Price, Itemname, Country, CustomerID, Q...	7

127 rows × 3 columns

```
# getting th item sets with length = 2 and support more han 10%

frequent_itemsets[ (frequent_itemsets['length'] == 2) &
```

```
                    (frequent_itemsets['support'] >= 0.01) ]
```

Output

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)

	support	itemsets	length
7	1.0	(BillNo, Itemname)	2
8	1.0	(BillNo, Quantity)	2
9	1.0	(BillNo, Date)	2
10	1.0	(BillNo, Price)	2
11	1.0	(BillNo, CustomerID)	2
12	1.0	(Country, BillNo)	2
13	1.0	(Itemname, Quantity)	2
14	1.0	(Itemname, Date)	2
15	1.0	(Itemname, Price)	2
16	1.0	(Itemname, CustomerID)	2
17	1.0	(Country, Itemname)	2
18	1.0	(Date, Quantity)	2
19	1.0	(Quantity, Price)	2
20	1.0	(CustomerID, Quantity)	2
21	1.0	(Country, Quantity)	2
22	1.0	(Date, Price)	2
23	1.0	(Date, CustomerID)	2
24	1.0	(Country, Date)	2
25	1.0	(CustomerID, Price)	2
26	1.0	(Country, Price)	2

```
# getting th item sets with length = 2 and support more han 10%

frequent_itemsets[ (frequent_itemsets['length'] == 1) &
                   (frequent_itemsets['support'] >= 0.01) ]
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
support      itemsets     length
0      1.0   (BillNo)     1
1      1.0   (Itemname)   1
2      1.0   (Quantity)   1
3      1.0   (Date) 1
4      1.0   (Price)      1
5      1.0   (CustomerID) 1
6      1.0   (Country)    1
```

ECLAT ALGORITHM

```
BottleNecks of Apriori:

Candidate generation can result in huge candidate sets
Multiple Scans of Database--- needs (n+1) scans,n is the longest pattern
To solve some of the above problems,Eclat has been introduced.

The ECLAT algorithm stands for Equivalence Class Clustering and bottom-up
Lattice Traversal. It is one of the popular methods of Association Rule mining.
It is a more efficient and scalable version of the Apriori algorithm. While the
Apriori algorithm works in a horizontal sense imitating the Breadth-First
Search of a graph, the ECLAT algorithm works in a vertical manner just like the
Depth-First Search of a graph. This vertical approach of the ECLAT algorithm
makes it a faster algorithm than the Apriori algorithm.

How it works:

The basic idea is to use Transaction Id Sets(tidsets) intersections to compute
the support value of a candidate and avoiding the generation of subsets which
do not exist in the prefix tree. In the first call of the function, all single
items are used along with their tidsets. Then the function is called
recursively and in each recursive call, each item-tidset pair is verified and
combined with other item-tidset pairs. This process is continued until no
candidate item-tidset pairs can be combined

Eg:

t1={a,b,c} t2={a,b} t3={a}

now above is horizontal layout where t1,t2,t3 are transactions a,b,c are
products.now let's make it into vertical layout....

k=1,min_support=0.5 a={t1,t2,t3},sup=1 b={t1,t2},sup=0.66 c={t1},sup=0.33
```

now we eliminate c as is supp<min_support and them generate itemsets of length k=2

{a,b}={t1,t2} supp=0.5

and we can't generate anymore sets so we end up with only {a,b}.

This method has an advantage over Apriori as it does not require scanning the database to find the support of k+1 itemsets. This is because the Transaction set will carry the count of occurrence of each item in the transaction (support). The bottleneck comes when there are many transactions taking huge memory and computational time for intersecting the sets.

If you want further reference you can visit : Eclat Algo

Advantages over Apriori algorithm:-

Memory Requirements: Since the ECLAT algorithm uses a Depth-First Search approach, it uses less memory than Apriori algorithm.
Speed: The ECLAT algorithm is typically faster than the Apriori algorithm.
Number of Computations: The ECLAT algorithm does not involve the repeated scanning of the data to compute the individual support values.
FP GROWTH(Frequent Pattern)
Shortcomings Of Apriori Algorithm

-Using Apriori needs a generation of candidate itemsets. These itemsets may be large in number if the itemset in the database is huge.

-Apriori needs multiple scans of the database to check the support of each itemset generated and this leads to high costs. These shortcomings can be overcome using the FP growth algorithm.

This algorithm is an improvement to the Apriori method. A frequent pattern is generated without the need for candidate generation. FP growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree.

This tree structure will maintain the association between the itemsets. The database is fragmented using one frequent item. This fragmented part is called “pattern fragment”. The itemsets of these fragmented patterns are analyzed. Thus with this method, the search for frequent itemsets is reduced comparatively.
FP TREE
Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset.

The root node represents null while the lower nodes represent the itemsets. The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree.

Frequent Pattern Algorithm Steps
The frequent pattern growth method lets us find the frequent pattern without candidate generation.

Let us see the steps followed to mine the frequent pattern using frequent pattern growth algorithm:

1) The first step is to scan the database to find the occurrences of the itemsets in the database. This step is the same as the first step of Apriori. The count of 1-itemsets in the database is called support count or frequency of 1-itemset.

2) The second step is to construct the FP tree. For this, create the root of the tree. The root is represented by null.

3) The next step is to scan the database again and examine the transactions. Examine the first transaction and find out the itemset in it. The itemset with the max count is taken at the top, the next itemset with lower count and so on. It means that the branch of the tree is constructed with transaction itemsets in descending order of count.

4) The next transaction in the database is examined. The itemsets are ordered in descending order of count. If any itemset of this transaction is already present in another branch (for example in the 1st transaction), then this transaction branch would share a common prefix to the root.

This means that the common itemset is linked to the new node of another itemset in this transaction.

5) Also, the count of the itemset is incremented as it occurs in the transactions. Both the common node and new node count is increased by 1 as they are created and linked according to transactions.

6) The next step is to mine the created FP Tree. For this, the lowest node is examined first along with the links of the lowest nodes. The lowest node represents the frequency pattern length 1. From this, traverse the path in the FP Tree. This path or paths are called a conditional pattern base.

Conditional pattern base is a sub-database consisting of prefix paths in the FP tree occurring with the lowest node (suffix).

7) Construct a Conditional FP Tree, which is formed by a count of itemsets in the path. The itemsets meeting the threshold support are considered in the Conditional FP Tree.

8) Frequent Patterns are generated from the Conditional FP Tree.

For example refer to link

Yeah,bored with theory now let's implement the fp growth algorithm using mlxtend

```
from mlxtend.frequent_patterns import fpgrowth
#running the fpgrowth algorithm
res=fpgrowth(data,min_support=0.05,use_colnames=True)
res.
```

Output

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
support     itemsets
```

```
0      1.0    (Country)
1      1.0    (CustomerID)
2      1.0    (Price)
3      1.0    (Date)
4      1.0    (Quantity)
...    ...    ...
122    1.0    (Price, Itemname, Country, CustomerID, Quantit...
123    1.0    (Date, Itemname, Country, CustomerID, Quantity...
124    1.0    (Date, Price, Itemname, Country, Quantity, Bil...
125    1.0    (Date, Price, Itemname, CustomerID, Quantity, ...
126    1.0    (Date, Price, Itemname, Country, CustomerID, Q...
127 rows × 2 columns
res=association_rules(res,metric="lift",min_threshold=1)
res
Output
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
antecedents consequents antecedent supportconsequent supportsupport
confidence  lift  leverage    conviction  zhangs_metric
0      (Country)    (CustomerID)1.0    1.0    1.0    1.0    1.0    0.0    inf    0.0
1      (CustomerID) (Country)    1.0    1.0    1.0    1.0    1.0    0.0    inf    0.0
2      (Country)    (Price)      1.0    1.0    1.0    1.0    1.0    0.0    inf    0.0
3      (Price)      (Country)    1.0    1.0    1.0    1.0    1.0    0.0    inf    0.0
4      (Country)    (Date)1.0    1.0    1.0    1.0    1.0    0.0    inf    0.0
...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
1927   (Itemname)   (Date, Price, Country, CustomerID, Quantity, B...     1.0
1.0    1.0    1.0    1.0    0.0    inf    0.0
1928   (Country)    (Date, Price, CustomerID, Quantity, BillNo, It...     1.0
1.0    1.0    1.0    1.0    0.0    inf    0.0
1929   (CustomerID) (Date, Price, Country, Quantity, BillNo, Itemn...     1.0
1.0    1.0    1.0    1.0    0.0    inf    0.0
1930   (Quantity)   (Date, Price, Country, CustomerID, BillNo, Ite...     1.0
1.0    1.0    1.0    1.0    0.0    inf    0.0
1931   (BillNo)     (Date, Price, Country, CustomerID, Quantity, I...     1.0
1.0    1.0    1.0    1.0    0.0    inf    0.0
1932 rows × 10 columns
Apriori Vs FP Growth¶
Since FP-Growth doesn't require creating candidate sets explicitly, it can be
magnitudes faster than the alternative Apriori algorithm. FP-Growth is about 5
times faster.Let's look at it
import time
l=[0.01,0.02,0.03,0.04,0.05]
t=[]
for i in l:
    t1=time.time()
    apriori(data,min_support=i,use_colnames=True)
    t2=time.time()
    t.append((t2-t1)*1000)
Output
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
```

```
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:
110: DeprecationWarning: DataFrames with non-bool types result in worse
computationalperformance and their support might be discontinued in the
future.Please use a DataFrame with bool type
  warnings.warn(
     support     itemsets
0        1.0    (BillNo)
1        1.0    (Itemname)
2        1.0    (Quantity)
3        1.0    (Date)
4        1.0    (Price)
...      ...    ...
122      1.0    (Price, Itemname, Country, CustomerID, Quantit...
123      1.0    (Date, Price, Itemname, Country, CustomerID, B...
124      1.0    (Date, Price, Country, CustomerID, Quantity, B...
125      1.0    (Date, Price, Country, CustomerID, Quantity, I...
126      1.0    (Date, Price, Itemname, Country, CustomerID, Q...
127 rows × 2 columns


/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:
110: DeprecationWarning: DataFrames with non-bool types result in worse
computationalperformance and their support might be discontinued in the
future.Please use a DataFrame with bool type
  warnings.warn(
     support     itemsets
0        1.0    (BillNo)
1        1.0    (Itemname)
2        1.0    (Quantity)
3        1.0    (Date)
4        1.0    (Price)
...      ...    ...
122      1.0    (Price, Itemname, Country, CustomerID, Quantit...
123      1.0    (Date, Price, Itemname, Country, CustomerID, B...
124      1.0    (Date, Price, Country, CustomerID, Quantity, B...
125      1.0    (Date, Price, Country, CustomerID, Quantity, I...
126      1.0    (Date, Price, Itemname, Country, CustomerID, Q...
127 rows × 2 columns


/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:
110: DeprecationWarning: DataFrames with non-bool types result in worse
computationalperformance and their support might be discontinued in the
future.Please use a DataFrame with bool type
  warnings.warn(
     support     itemsets
0        1.0    (BillNo)
1        1.0    (Itemname)
2        1.0    (Quantity)
3        1.0    (Date)
4        1.0    (Price)
...      ...    ...
122      1.0    (Price, Itemname, Country, CustomerID, Quantit...
```

```
123   1.0   (Date, Price, Itemname, Country, CustomerID, B...
124   1.0   (Date, Price, Country, CustomerID, Quantity, B...
125   1.0   (Date, Price, Country, CustomerID, Quantity, I...
126   1.0   (Date, Price, Itemname, Country, CustomerID, Q...
127 rows × 2 columns


/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:
110: DeprecationWarning: DataFrames with non-bool types result in worse
computationalperformance and their support might be discontinued in the
future.Please use a DataFrame with bool type
  warnings.warn(
support     itemsets
0     1.0   (BillNo)
1     1.0   (Itemname)
2     1.0   (Quantity)
3     1.0   (Date)
4     1.0   (Price)
...   ...   ...
122   1.0   (Price, Itemname, Country, CustomerID, Quantit...
123   1.0   (Date, Price, Itemname, Country, CustomerID, B...
124   1.0   (Date, Price, Country, CustomerID, Quantity, B...
125   1.0   (Date, Price, Country, CustomerID, Quantity, I...
126   1.0   (Date, Price, Itemname, Country, CustomerID, Q...
127 rows × 2 columns


/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:
110: DeprecationWarning: DataFrames with non-bool types result in worse
computationalperformance and their support might be discontinued in the
future.Please use a DataFrame with bool type
  warnings.warn(
support     itemsets
0     1.0   (BillNo)
1     1.0   (Itemname)
2     1.0   (Quantity)
3     1.0   (Date)
4     1.0   (Price)
...   ...   ...
122   1.0   (Price, Itemname, Country, CustomerID, Quantit...
123   1.0   (Date, Price, Itemname, Country, CustomerID, B...
124   1.0   (Date, Price, Country, CustomerID, Quantity, B...
125   1.0   (Date, Price, Country, CustomerID, Quantity, I...
126   1.0   (Date, Price, Itemname, Country, CustomerID, Q...
127 rows × 2 columns
l=[0.01,0.02,0.03,0.04,0.05]
f=[]
for i in l:
    t1=time.time()
    fpgrowth(data,min_support=i,use_colnames=True)
    t2=time.time()
    f.append((t2-t1)*1000)
Output
```

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to

`transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...
122	1.0	(Price, Itemname, Country, CustomerID, Quantit...
123	1.0	(Date, Itemname, Country, CustomerID, Quantity...
124	1.0	(Date, Price, Itemname, Country, Quantity, Bil...
125	1.0	(Date, Price, Itemname, CustomerID, Quantity, ...
126	1.0	(Date, Price, Itemname, Country, CustomerID, Q...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)

...	...	...
122	1.0	(Price, Itemname, Country, CustomerID, Quantit...
123	1.0	(Date, Itemname, Country, CustomerID, Quantity...
124	1.0	(Date, Price, Itemname, Country, Quantity, Bil...
125	1.0	(Date, Price, Itemname, CustomerID, Quantity, ...
126	1.0	(Date, Price, Itemname, Country, CustomerID, Q...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...
122	1.0	(Price, Itemname, Country, CustomerID, Quantit...
123	1.0	(Date, Itemname, Country, CustomerID, Quantity...
124	1.0	(Date, Price, Itemname, Country, Quantity, Bil...
125	1.0	(Date, Price, Itemname, CustomerID, Quantity, ...
126	1.0	(Date, Price, Itemname, Country, CustomerID, Q...

127 rows × 2 columns

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...
122	1.0	(Price, Itemname, Country, CustomerID, Quantit...
123	1.0	(Date, Itemname, Country, CustomerID, Quantity...
124	1.0	(Date, Price, Itemname, Country, Quantity, Bil...
125	1.0	(Date, Price, Itemname, CustomerID, Quantity, ...
126	1.0	(Date, Price, Itemname, Country, CustomerID, Q...

127 rows × 2 columns

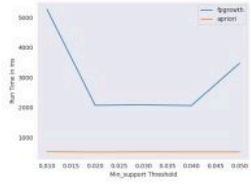
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

	support	itemsets
0	1.0	(Country)
1	1.0	(CustomerID)
2	1.0	(Price)
3	1.0	(Date)
4	1.0	(Quantity)
...	...	...

122	1.0	(Price, Itemname, Country, CustomerID, Quantit...
123	1.0	(Date, Itemname, Country, CustomerID, Quantity...
124	1.0	(Date, Price, Itemname, Country, Quantity, Bil...
125	1.0	(Date, Price, Itemname, CustomerID, Quantity, ...
126	1.0	(Date, Price, Itemname, Country, CustomerID, Q...

127 rows × 2 columns

```
sns.lineplot(x=l,y=f,label="fpgrowth")
sns.lineplot(x=l,y=t,label="apriori")
plt.xlabel("Min_support Threshold")
plt.ylabel("Run Time in ms")
Output
```



Advantage

Market basket insights provide valuable advantages for both businesses and consumers:

1. **Business Advantages**:
   a. Increased Sales: Understanding which products are commonly purchased together helps businesses cross-sell and upsell, boosting overall sales.
   b. **Inventory Management**: Helps optimize inventory levels, reducing overstocking or understocking issues.
   c. **Personalization**: Enables personalized marketing and recommendations, enhancing customer engagement and loyalty.
   d. **Pricing Strategies**: Informs pricing strategies by identifying which products are price-sensitive or complementary.
   e. **Supply Chain Optimization**: Improves supply chain efficiency by aligning procurement and production with actual demand.
   f. **Fraud Detection:** Helps identify unusual purchasing patterns, aiding in fraud detection and prevention.

2. **Consumer Advantages:**
   a. **Convenience**: Suggests complementary products, making shopping more convenient and time-efficient.
   b. **Savings**: Promotes the discovery of related discounts or bundle offers, leading to potential cost savings.

c. **Personalization**: Enhances the shopping experience through personalized product recommendations.
d. **Variety**: Encourages customers to explore a wider range of products they might not have considered.
e. **Relevant Promotions**: Results in fewer irrelevant ads and promotions, improving the overall shopping experience.

Overall, market basket insights contribute to more efficient and satisfying shopping experiences, benefiting both businesses and consumers.

Disadvantage

Market basket analysis, while a valuable tool, does have some disadvantages:

1. **Lack of Causality**: Market basket analysis identifies associations between items but doesn't explain why these associations exist. It doesn't uncover causation, which can limit its usefulness in decision-making.

2. **Data Quality**: The accuracy of insights depends on data quality. Incomplete or noisy data can lead to misleading results.

3. **Dynamic Market:** Market basket analysis assumes that item associations remain constant, which may not hold in dynamic markets where trends change quickly.

4. **Scalability**: For large datasets, the computational requirements can be significant, making it challenging to perform real-time analysis.

5. **Limited Context:** Market basket analysis often focuses on item-level associations and may not consider other contextual information, such as customer demographics or external factors.

6. **Overfitting**: It's possible to find spurious associations in the data, leading to overfit models that don't generalize well to new data.

7. **Competitive Advantage Erosion**: If everyone in a market uses market basket analysis, the competitive advantage of discovering item associations can diminish.

8. **Privacy Concerns**: Analyzing customer purchase data for market basket insights can raise privacy issues if not handled carefully.

Despite these disadvantages, market basket analysis remains a valuable tool for retail and e-commerce businesses when used in conjunction with other data analysis techniques.

Benefits

Market basket insights offer several benefits in the context of retail and e-commerce:

1. **Sales Optimization**: Retailers can use market basket analysis to optimize product placements and promotions, increasing cross-selling and upselling opportunities.

2. **Customer Segmentation**: It helps in understanding customer preferences and behaviors, allowing businesses to segment their customer base effectively.

3. **Inventory Management**: Retailers can manage their inventory more efficiently by stocking products that are frequently bought together, reducing overstock and understock situations.

4. **Price Optimization**: Market basket insights can inform pricing strategies, helping businesses set competitive prices for bundled products.

5. **Personalization**: By understanding what products are frequently purchased together, businesses can provide personalized recommendations to customers, enhancing their shopping experience.

6. **Loss Prevention**: It can identify unusual purchasing patterns, which may indicate theft or fraud.

7. **Marketing Campaigns**: Businesses can tailor their marketing campaigns based on the products commonly purchased together by specific customer segments.

8. **Assortment Planning**: It helps in deciding which products to carry and which to drop from the inventory.

9. **Forecasting**: Businesses can make more accurate sales forecasts, enabling better resource allocation.

10. **Customer Loyalty**: Understanding customer preferences and catering to them can enhance customer loyalty and retention.

These insights can ultimately lead to increased revenue, improved customer satisfaction, and more efficient operations.

Conclusion

The overall conclusion of a market basket analysis, which is a data mining technique used in retail and other industries, can be summarized as follows:

1. **Association Rules**: Market basket analysis aims to uncover associations or relationships between items that are frequently purchased together. It uses metrics like support, confidence, and lift to quantify these relationships.

2. **Item Co-Occurrences**: By analyzing transaction data, this technique identifies which items tend to co-occur in the same transactions. For example, it might reveal that customers who buy bread often also purchase butter.

3. **Cross-Selling Opportunities**: Retailers can use these insights to optimize product placements and promotions. For instance, placing butter next to the bread aisle can encourage more sales.

4. **Inventory Management**: Market basket analysis helps with inventory management. Retailers can ensure that products frequently bought together are stocked together, reducing the risk of stockouts.

5. **Customer Segmentation**: It can also help segment customers based on their purchase patterns. For example, it might identify a group of health-conscious shoppers who tend to buy organic products.

6. **Personalization**: Retailers can leverage these insights to offer personalized product recommendations to customers, enhancing the shopping experience.

7. **Limitations**: Market basket analysis has its limitations. It may not account for seasonal variations, changing customer preferences, or external factors like promotions or marketing campaigns.

8. **Ongoing Process**: Market basket analysis is not a one-time task; it's an ongoing process that requires constant data monitoring and adjustment to remain relevant.

In conclusion, market basket analysis provides valuable insights into customer behavior, product associations, and retail operations, allowing businesses to make data-driven decisions to improve sales

and customer satisfaction. However, it should be used in conjunction with other data and strategies to achieve the best results in today's dynamic marketplace.