Market Basket insight phase 3

Introducing :

Market Basket Analysis using a real-world dataset. Market Basket Analysis is a powerful technique that allows us to uncover patterns and associations between items that customers tend to purchase together. By analyzing these patterns, we can gain valuable insights that can drive business decisions and strategies.

In this notebook, we will work with a Market Basket dataset that captures customer transactions in a retail or e-commerce setting. The dataset provides a wealth of information about customer purchases, allowing us to dive deep into their buying behavior. By leveraging data mining techniques and association rule mining algorithms, we will unravel the relationships between items and discover interesting patterns.

Through this analysis, we can derive actionable insights to improve various aspects of business operations. We can identify frequently co-purchased items, enabling us to make targeted product recommendations and enhance cross-selling and upselling opportunities. By optimizing product placement and store layout based on association patterns, we can create more enticing shopping experiences. Furthermore, we can design effective promotional campaigns by leveraging the discovered item associations, resulting in higher customer engagement and increased sales.

In this notebook, we will take you through the entire process of Market Basket Analysis, from data preprocessing to association rule mining and visualization. By following along with the provided code and explanations, you will gain a solid understanding of how to extract valuable insights from Market Basket datasets and apply them to real-world scenarios.

So let's dive in and unlock the secrets hidden within the Market Basket dataset to gain a deeper understanding of customer behavior and optimize business strategies!

Overview of the Market Basket Analysis dataset:

This dataset contains 522,065 rows and 7 attributes that provide valuable information about customer transactions and product details. Here is a breakdown of the attributes:

BillNo: This attribute represents a 6-digit number assigned to each transaction. It serves as a unique identifier for identifying individual purchases.

Itemname: This attribute stores the name of the product purchased in each transaction. It provides nominal data representing different products.

Quantity: This attribute captures the quantity of each product purchased in a transaction. It is a numeric value that indicates the number of units of a specific item.

Date: The Date attribute records the day and time when each transaction occurred. It provides valuable information about the timing of purchases.

Price: This attribute represents the price of each product. It is a numeric value that indicates the cost of a single unit of the item.

CustomerID: Each customer is assigned a 5-digit number as their unique identifier. This attribute helps track customer-specific information and analyze individual buying patterns.

Country: The Country attribute denotes the name of the country where each customer resides. It provides nominal data representing different geographic regions.

By analyzing this dataset, we can gain insights into customer purchasing behavior, identify popular products, examine sales trends over time, and explore the impact of factors such as price and geography on customer preferences. These insights can be used to optimize marketing strategies, improve inventory management, and enhance customer satisfaction

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 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	6.0	2010-01-12 08:26:00	3,39	17850.0	United Kingdom

WHITE
HEART.

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	122.400985	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.

```python
# 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 RETROSPOT 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	1.0	2011-02-11 15:33:00	7.46	NaN	7.46

		SWEET HOME					
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 RETROSPOT 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 DOOR SIGN 6	1.0	2010-12-22 14:41:00	0.0	NaN	0.0

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:

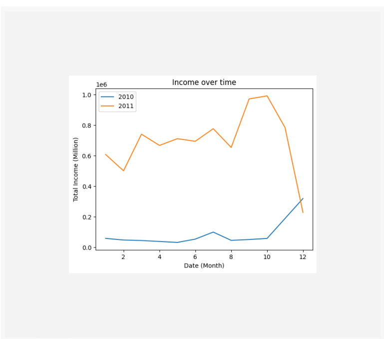
```
Exploring the Dataset: We investigate the dataset's dimensions, such as
the number of rows and columns, to gauge its size and complexity.
Additionally, we examine the data types of each column to understand
the nature of the variables.

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:

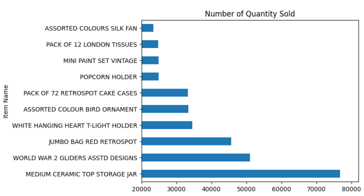
```
Timestamp('2011-12-10 17:19:00')
```

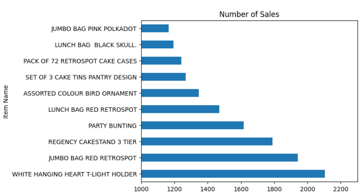
Based on the finding that the data is only available for 10 days in the last month, it becomes evident that the significant drop in sales observed during that period is likely due to the limited data rather than an actual decline in sales. The incomplete data for the last month may not provide a comprehensive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

Conclusion:

In conclusion, loading and preprocessing transaction data are crucial steps in generating market basket insights. These initial data preparation tasks help in identifying patterns and associations among items purchased by customers, which can ultimately lead to valuable insights for businesses, such as product recommendations, inventory management, and marketing strategies.