Market Basket Analysis and Recency Frequency Monetary Analysis for Groceries

[60]:

```
!pip install mlxtend
!pip install apriori
import os
import plotly.express as px
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import seaborn as sns
import operator as op
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
Requirement already satisfied: mlxtend in c:\users\n347u\anaconda3\lib\site-packages (0.23.1)
```

Dataset Exploration

In [61]:
```
# Loading dataset
data.head()
```

Out[61]:

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

As we can see from the first 5 rows of the dataset, we have 3 different columns: Member_number is a number that is unique for each customer. Date represents the date of the transaction, and finally itemDescription represents the corresponding product bought for this date.

In [62]:
```python
# To check the data types of the columns, I use info() function. According to result below, Member_number and Date columns are nc
#For the following steps, I will change them into correct form.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38765 entries, 0 to 38764
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Member_number    38765 non-null  int64 
 1   Date             38765 non-null  object
 2   itemDescription  38765 non-null  object
dtypes: int64(1), object(2)
memory usage: 908.7+ KB
```

In [63]:
```python
# Renaming the columns, because it would be easier to understand what that means.
data.columns = ['memberID', 'Date', 'itemName']
data.head()
```

Out[63]:

	memberID	Date	itemName
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

In [64]:
```python
# Checking for the missing values
nan_values = data.isna().sum()
nan_values
```

Out[64]:
```
memberID    0
Date        0
itemName    0
dtype: int64
```

According to result above, there is no missing values for all columns.

In [65]:
```python
# Converting Date column into correct datatype which is datetime
data.Date = pd.to_datetime(data.Date)
data.memberID = data['memberID'].astype('str')
data.info()  # They are in correct datatype now
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38765 entries, 0 to 38764
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   memberID  38765 non-null  object
 1   Date      38765 non-null  datetime64[ns]
 2   itemName  38765 non-null  object
dtypes: datetime64[ns](1), object(2)
memory usage: 908.7+ KB
```

Data Visualization

Number of Sales Weekly

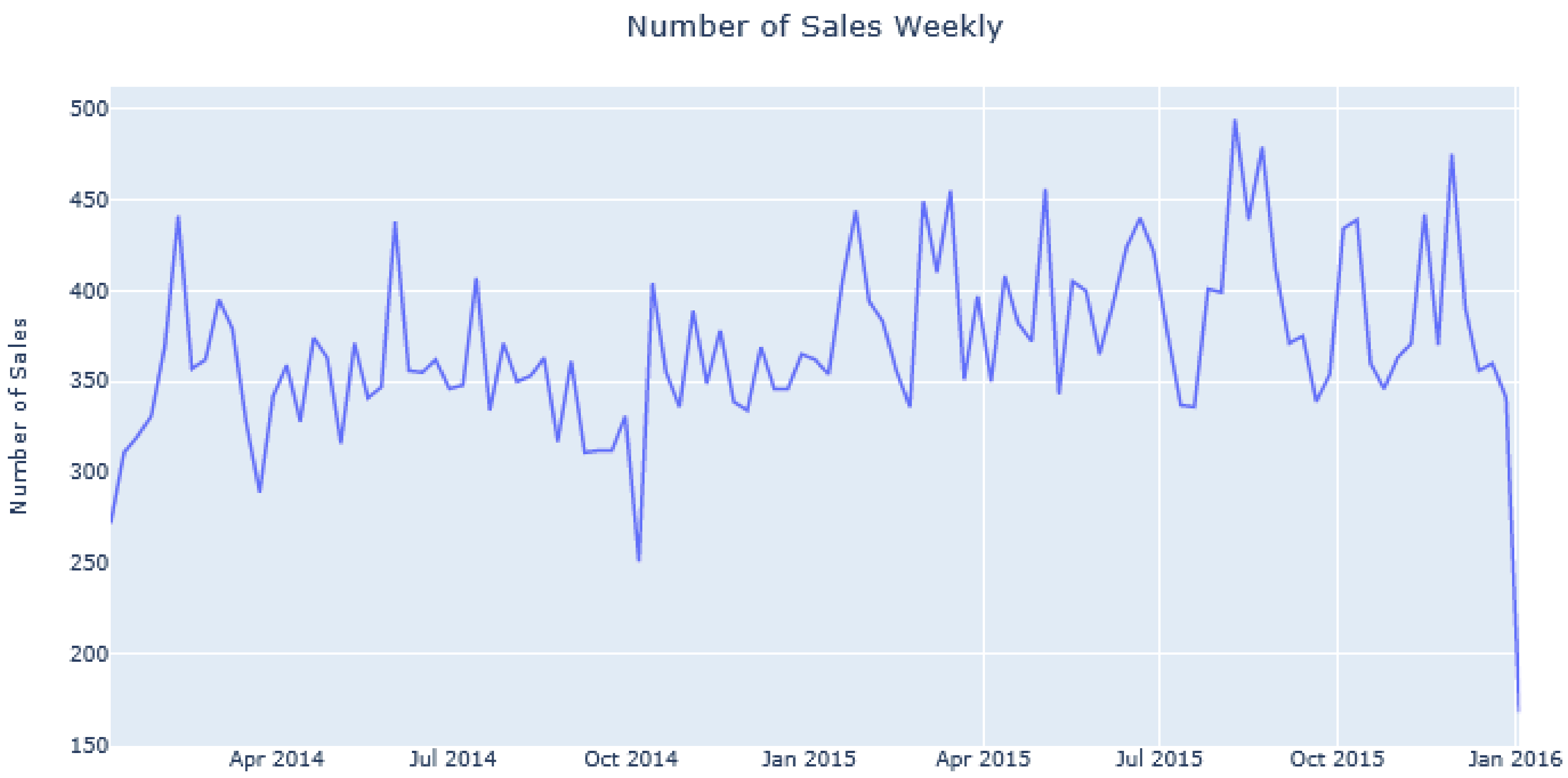
In [66]:
```
Sales_weekly = data.resample('w', on='Date').size()
fig = px.line(data, x=Sales_weekly.index, y=Sales_weekly,
              labels={'y': 'Number of Sales',
                      'x': 'Date'})
fig.update_layout(title_text='Number of Sales Weekly',
                  title_x=0.5, title_font=dict(size=18))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Number of Customers Weekly

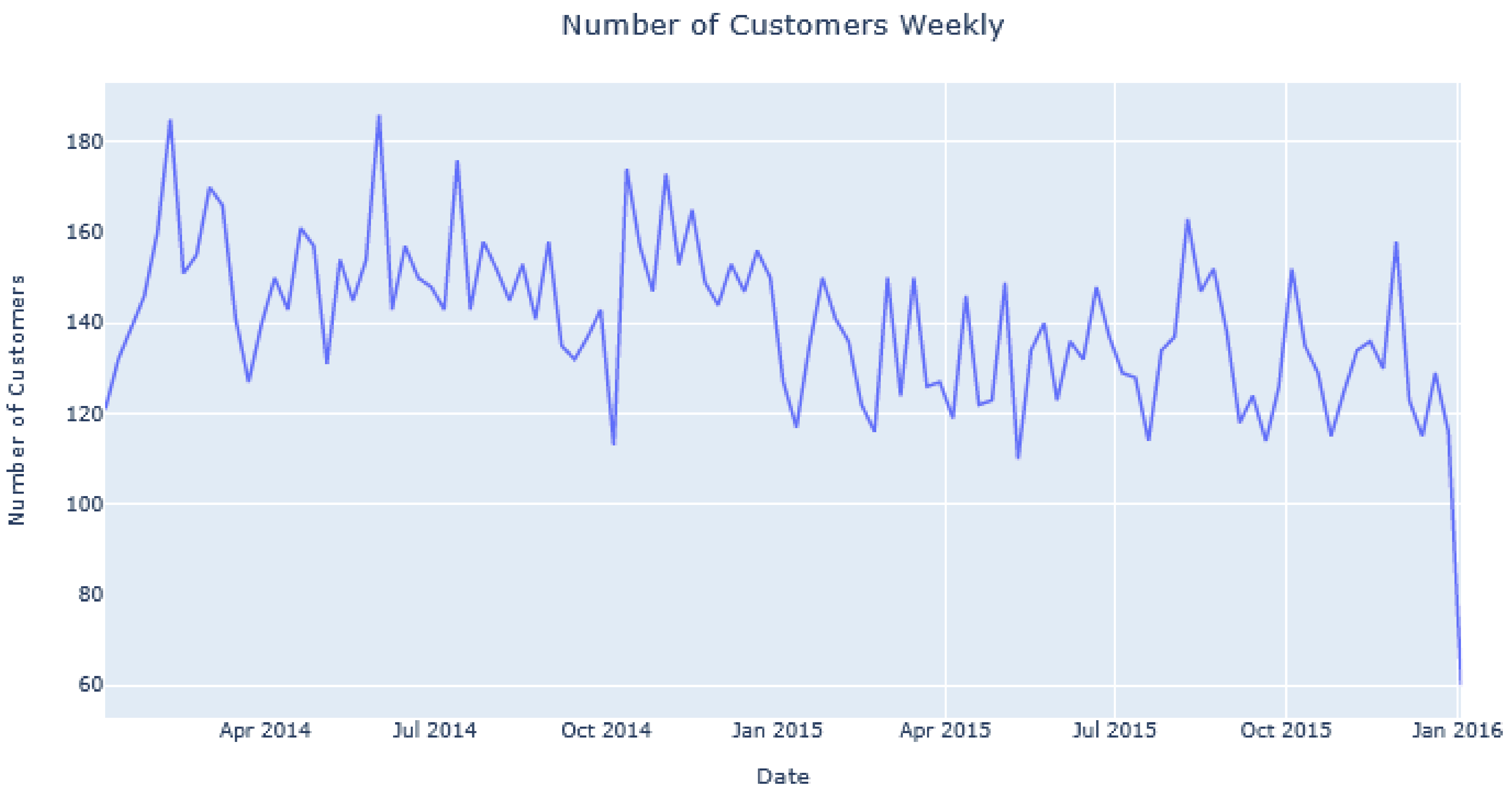
In [67]:
```
Unique_customer_weekly = data.resample('w', on='Date').memberID.nunique()
fig = px.line(Unique_customer_weekly, x=Unique_customer_weekly.index, y=Unique_customer_weekly,
              labels={'y': 'Number of Customers'})
fig.update_layout(title_text='Number of Customers Weekly',
                  title_x=0.5, title_font=dict(size=18))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Sales per Customer Weekly

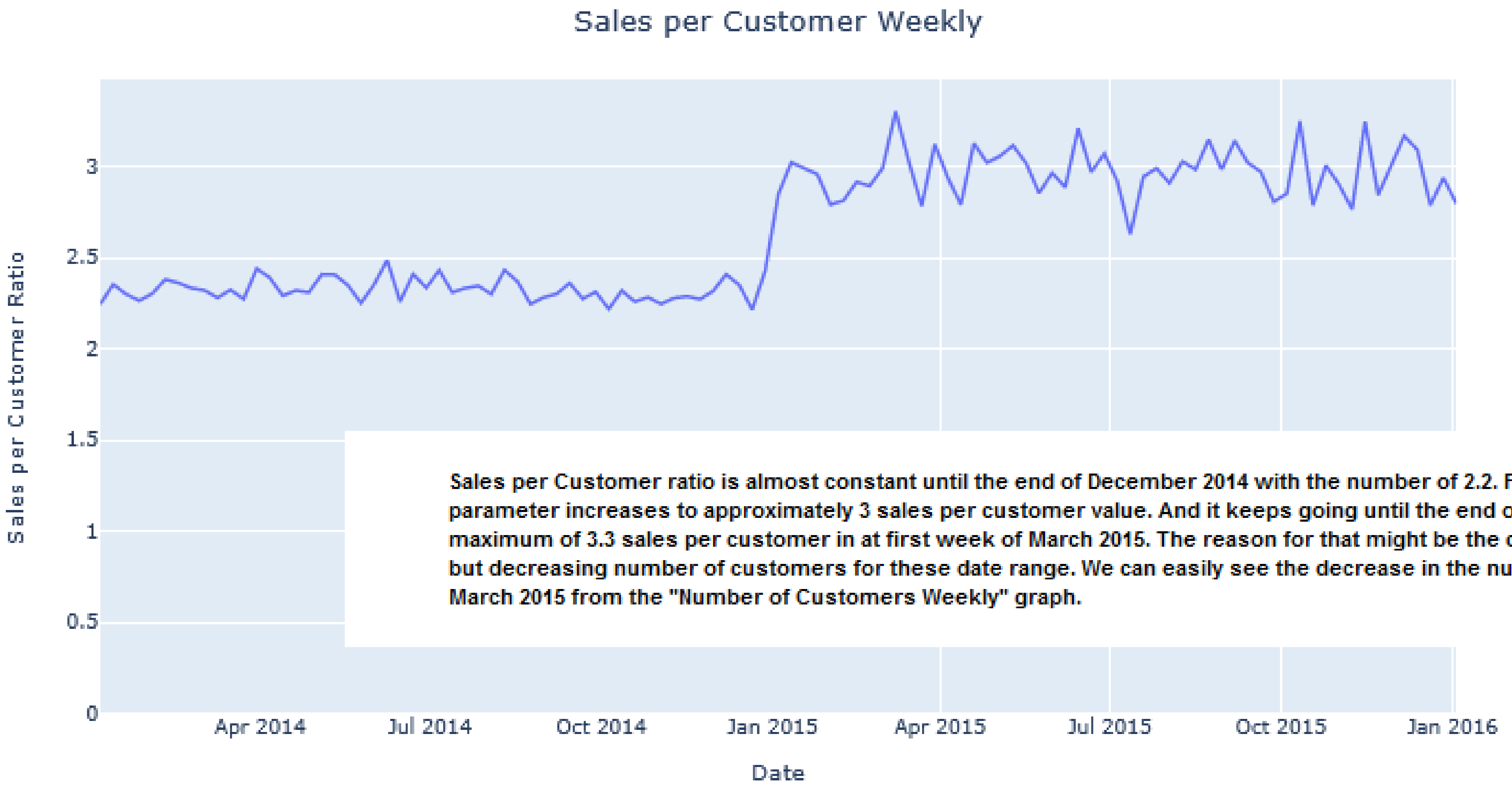
In [68]:
```
Sales_per_Customer = Sales_weekly / Unique_customer_weekly
fig = px.line(Sales_per_Customer, x=Sales_per_Customer.index, y=Sales_per_Customer,
              labels={'y': 'Sales per Customer Ratio'})
fig.update_layout(title_text='Sales per Customer Weekly',
                  title_x=0.5, title_font=dict(size=18))
fig.update_yaxes(rangemode="tozero")
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Sales per Customer ratio is almost constant until the end of December 2014 with the number of 2.2. Fortunately, after this week this parameter increases to approximately 3 sales per customer value. And it keeps going until the end of the date range. It has a maximum of 3.3 sales per customer in at first week of March 2015. The reason for that might be the constant number of sales weekly but decreasing number of customers for these date range. We can easily see the decrease in the number of customers weekly after March 2015 from the "Number of Customers Weekly" graph.

Frequency of the Items Sold

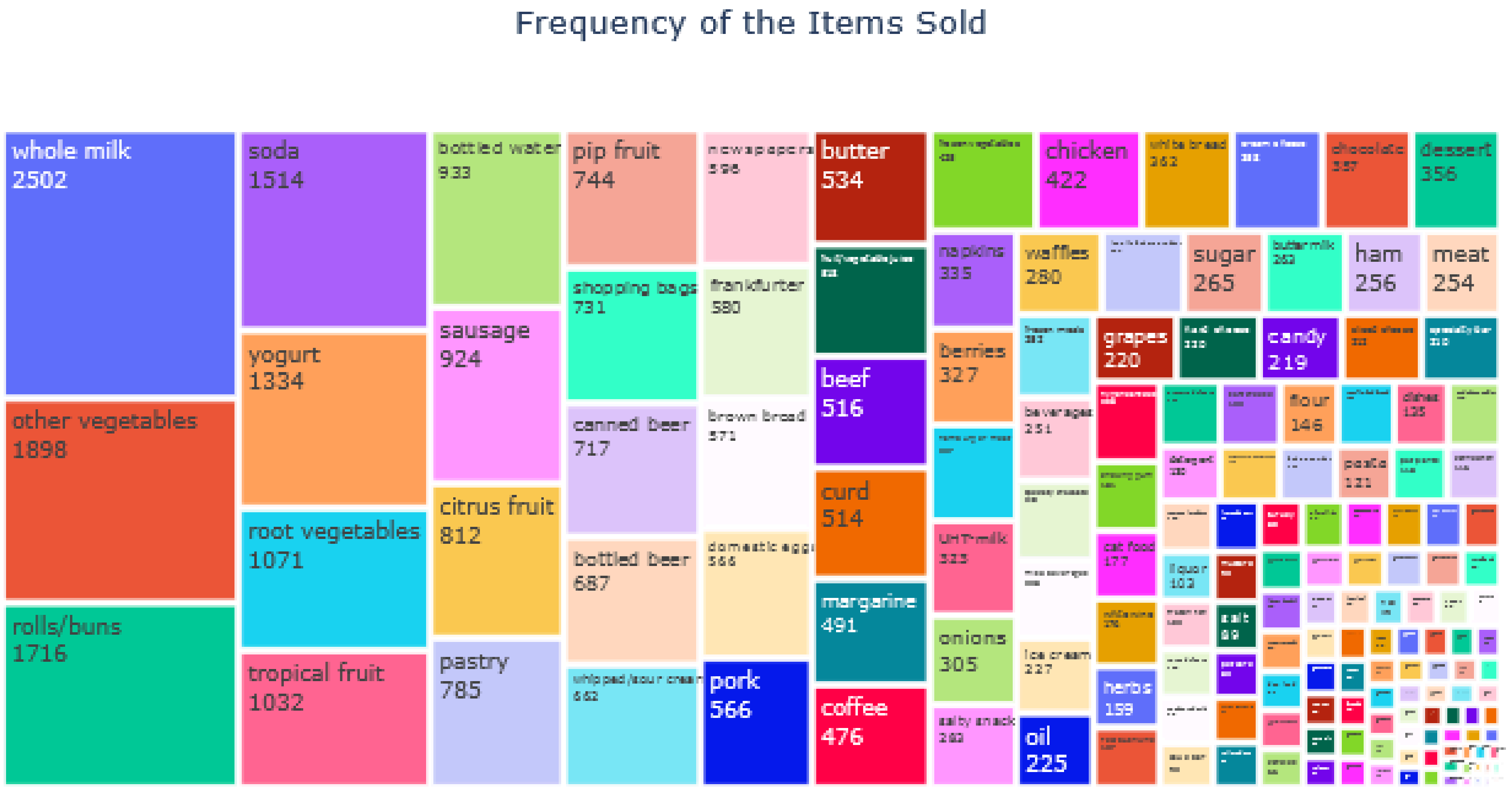
In [69]:
```
Frequency_of_items = data.groupby(pd.Grouper(key='itemName')).size().reset_index(name='count')
fig = px.treemap(Frequency_of_items, path=['itemName'], values='count')
fig.update_layout(title_text='Frequency of the Items Sold',
                  title_x=0.5, title_font=dict(size=18)
                  )
fig.update_traces(textinfo="label+value")
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Whole milk, vegetables and rolls/buns are top 3 products that sold for this groceries.

Top Customers regarding Number of Items bought

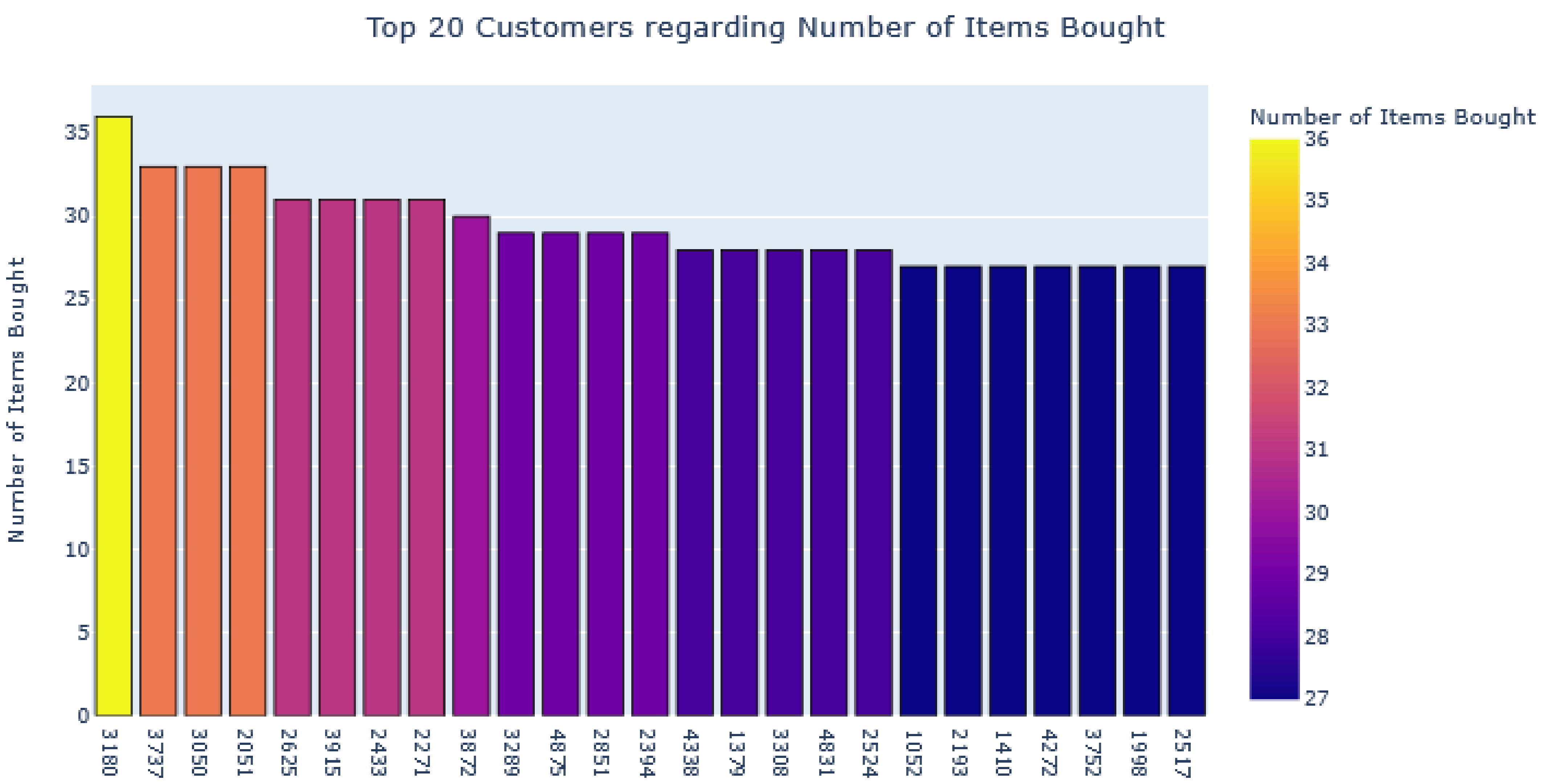
In [70]:
```
user_item = data.groupby(pd.Grouper(key='memberID')).size().reset_index(name='count') \
    .sort_values(by='count', ascending=False)
fig = px.bar(user_item.head(25), x='memberID', y='count',
             labels={'y': 'Number of Sales',
                     'count': 'Number of Items Bought'},
             color='count')
fig.update_layout(title_text='Top 20 Customers regarding Number of Items Bought',
                  title_x=0.5, title_font=dict(size=18))
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Customer 3180 looks like our the best customer :)

Number of Sales per Discrete Week Days

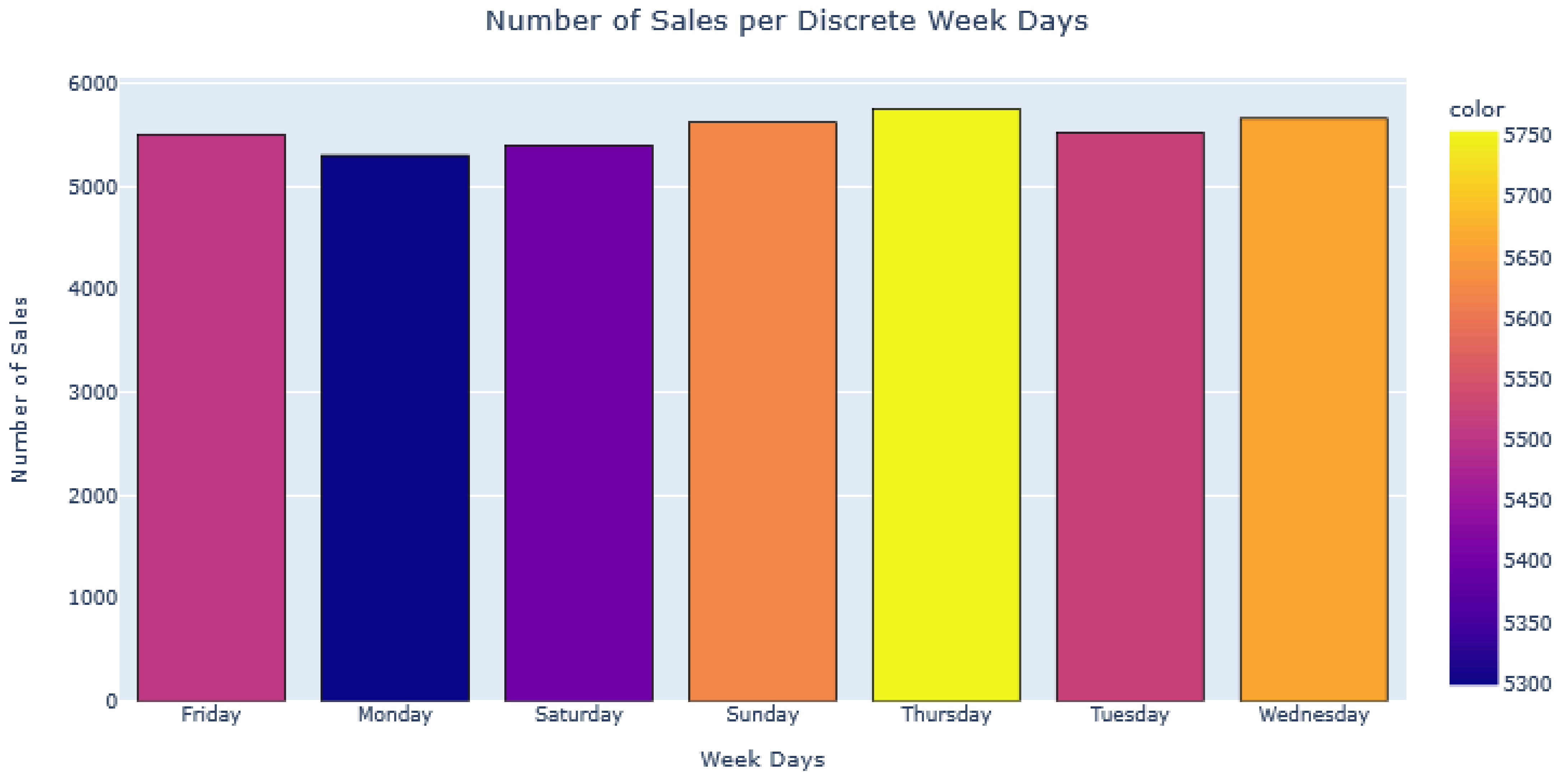
In [71]:
```
day = data.groupby(data['Date'].dt.strftime('%A'))['itemName'].count()
fig = px.bar(day, x=day.index, y=day, color=day,
             labels={'y': 'Number of Sales',
                     'Date': 'Week Days'})
fig.update_layout(title_text='Number of Sales per Discrete Week Days',
                  title_x=0.5, title_font=dict(size=18))
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



According to the graph above, there are more sales on Thursdays. Sunday and Wednesday are the following days for the most sales made.

Number of Sales per Discrete Months

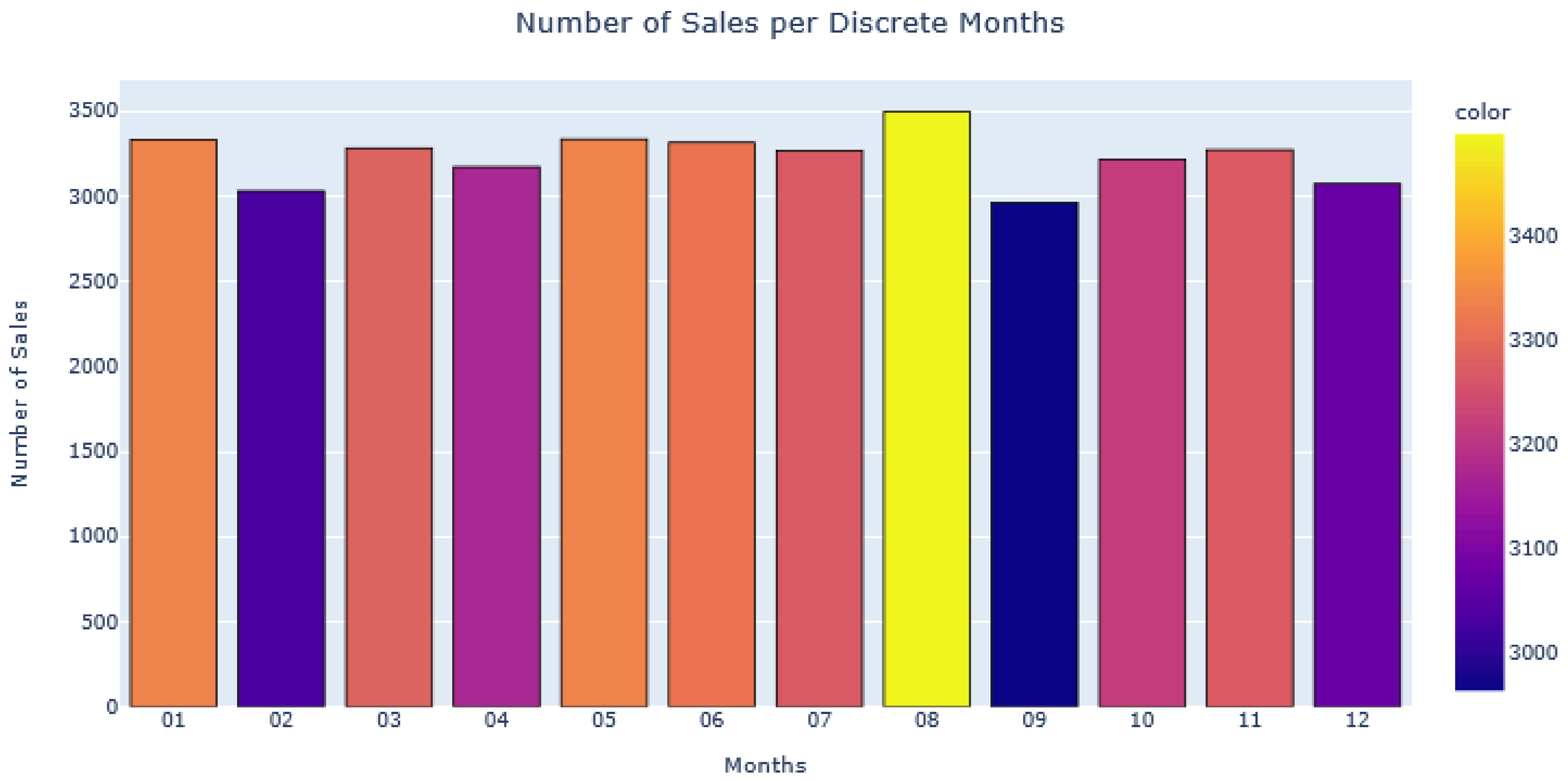
In [72]:
```
month = data.groupby(data['Date'].dt.strftime('%m'))['itemName'].count()
fig = px.bar(month, x=month.index, y=month, color=month,
             labels={'y': 'Number of Sales',
                     'Date': 'Months'})
fig.update_layout(title_text='Number of Sales per Discrete Months',
                  title_x=0.5, title_font=dict(size=18))
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



According to the graph above, there are more sales on August. January and May are the following months for the most sales made.

Number of Sales per Discrete Month Days

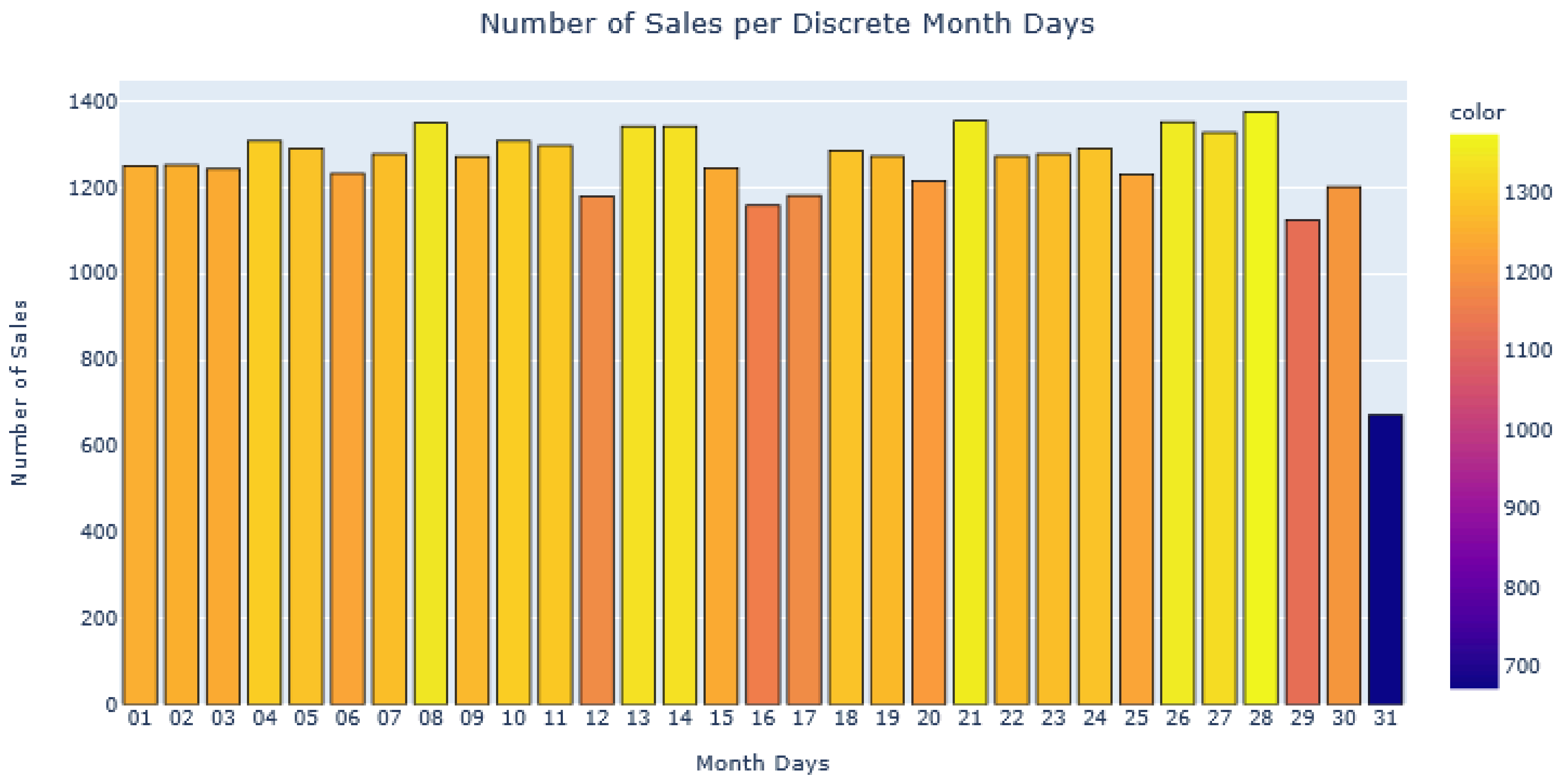
In [73]:
```
month_day = data.groupby(data['Date'].dt.strftime('%d'))['itemName'].count()
fig = px.bar(month_day, x=month_day.index, y=month_day, color=month_day,
             labels={'y': 'Number of Sales',
                     'Date': 'Month Days'})
fig.update_layout(title_text='Number of Sales per Discrete Month Days',
                  title_x=0.5, title_font=dict(size=18))
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Approximately each Month Day has equal Number of Sales.

Market Basket Analysis

Market basket analysis is a data mining technique used by retailers to increase sales by better understanding customer purchasing patterns. It involves analyzing large data sets, such as purchase history, to reveal product groupings, as well as products that are likely to be purchased together [1].

Let's Create Baskets

In [74]:
```
baskets = data.groupby(['memberID', 'itemName'])['itemName'].count().unstack().fillna(0).reset_index()
baskets.head()
```

Out[74]:

itemName	memberID	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	beef	...	turkey	vinegar	waffles	whipped/sour cream	whisky	w br
0	1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
1	1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	1.0	0.0	
2	1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
3	1003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
4	1004	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	

5 rows × 168 columns

According to the query above, we created the customers-items matrix. Each row represents the transactions of each customer, and each column represents the items bought. The numbers corresponding to the matrix represent the number of times that item is bought by the individual user.

In [75]:
```
# Let's check the most sold -item which is whole milk- has the same number of sales as we discussed above in the treemap.
baskets['whole milk'].sum()

# Yep it satisfies our results.
```

Out[75]: 2502.0

```
In [76]: # Encoding the items that sold more than 1
def one_hot_encoder(k):
    if k <= 0:
        return 0
    if k >= 1:
        return 1
```

```
In [77]: baskets_final = baskets.iloc[:, 1:baskets.shape[1]].applymap(one_hot_encoder)
baskets_final.head()
```

Out[77]:

itemName	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	beef	berries	...	turkey	vinegar	waffles	whipped/sour cream	whisky	white bread
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	...	0	0	0	1	0	1
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

5 rows × 167 columns

Final form of our customer-item matrix.

In [78]:
```python
# Finding the most frequent items sold together
frequent_itemsets = apriori(baskets_final, min_support=0.025, use_colnames=True, max_len=3).sort_values(by='support')
frequent_itemsets.head(25)
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:109: DeprecationWarning:

DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the fut
ure.Please use a DataFrame with bool type
```

Out[78]:

	support	itemsets
161	0.025141	(shopping bags, butter)
69	0.025141	(spread cheese)
405	0.025141	(whole milk, sliced cheese)
412	0.025141	(specialty bar, whole milk)
85	0.025141	(pip fruit, beef)
248	0.025141	(shopping bags, domestic eggs)
467	0.025141	(tropical fruit, whole milk, citrus fruit)
480	0.025141	(yogurt, whole milk, frankfurter)
119	0.025141	(chocolate, bottled water)
239	0.025141	(root vegetables, dessert)
540	0.025141	(yogurt, shopping bags, rolls/buns)
559	0.025141	(shopping bags, tropical fruit, whole milk)
524	0.025141	(whole milk, root vegetables, pastry)
224	0.025398	(root vegetables, cream cheese)
217	0.025398	(sausage, coffee)
204	0.025398	(pork, citrus fruit)
302	0.025398	(tropical fruit, margarine)
167	0.025398	(rolls/buns, butter milk)
473	0.025398	(yogurt, whole milk, curd)
476	0.025654	(yogurt, whole milk, domestic eggs)
462	0.025654	(rolls/buns, soda, citrus fruit)
434	0.025654	(tropical fruit, bottled water, other vegetables)
186	0.025654	(chewing gum, whole milk)
88	0.025654	(sausage, beef)
171	0.025654	(domestic eggs, canned beer)

As we can see from the results above, the most items that appeared together are butter and shopping bags, and spread cheese, and so on.

Out[79]:

	antecedents	consequents	support	confidence	lift
878	(sausage)	(yogurt, rolls/buns)	0.035659	0.173101	1.554717
875	(yogurt, rolls/buns)	(sausage)	0.035659	0.320276	1.554717
456	(root vegetables, whole milk)	(shopping bags)	0.029246	0.258503	1.536046
457	(shopping bags)	(root vegetables, whole milk)	0.029246	0.173780	1.536046
876	(sausage, rolls/buns)	(yogurt)	0.035659	0.433022	1.530298
877	(yogurt)	(sausage, rolls/buns)	0.035659	0.126020	1.530298
946	(sausage)	(yogurt, other vegetables)	0.037199	0.180573	1.500795
943	(yogurt, other vegetables)	(sausage)	0.037199	0.309168	1.500795
631	(shopping bags)	(soda, other vegetables)	0.031042	0.184451	1.485518
630	(soda, other vegetables)	(shopping bags)	0.031042	0.250000	1.485518
1194	(sausage, whole milk)	(yogurt)	0.044895	0.419664	1.483093
1195	(yogurt)	(sausage, whole milk)	0.044895	0.158658	1.483093
323	(canned beer)	(whole milk, root vegetables)	0.027707	0.167702	1.482317
318	(whole milk, root vegetables)	(canned beer)	0.027707	0.244898	1.482317
882	(rolls/buns, soda)	(sausage)	0.035916	0.299786	1.455249
883	(sausage)	(rolls/buns, soda)	0.035916	0.174346	1.455249
199	(whole milk, soda)	(curd)	0.026424	0.174873	1.447248
202	(curd)	(whole milk, soda)	0.026424	0.218684	1.447248
1193	(yogurt, whole milk)	(sausage)	0.044895	0.298126	1.447192
1196	(sausage)	(yogurt, whole milk)	0.044895	0.217933	1.447192
903	(shopping bags)	(whole milk, soda)	0.036685	0.217988	1.442643
902	(whole milk, soda)	(shopping bags)	0.036685	0.242784	1.442643
247	(yogurt)	(whole milk, butter)	0.026937	0.095195	1.438255
246	(whole milk, butter)	(yogurt)	0.026937	0.406977	1.438255
944	(sausage, other vegetables)	(yogurt)	0.037199	0.400552	1.415552

As we can see from the result above:

sausage --> yogurt, rolls/buns

root vegetables, whole milk --> shopping bags

rolls/buns, soda --> sausage

butter, whole milk --> yogurt,

and etc. have strong relationships.

RFM Analysis

Recency

In [80]:
```
# Lets Start with the calculate the Recency

# Finding last purchase date of each customer
Recency = data.groupby(by='memberID')['Date'].max().reset_index()
Recency.columns = ['memberID', 'LastDate']
Recency.head()
```

Out[80]:

	memberID	LastDate
0	1000	2015-11-25
1	1001	2015-04-14
2	1002	2015-08-30
3	1003	2015-10-02
4	1004	2015-02-12

In [81]:
```
# Finding last date for our dataset
last_date_dataset = Recency['LastDate'].max()
last_date_dataset
```

Out[81]:
```
Timestamp('2015-12-30 00:00:00')
```

In [82]:
```
# Calculating Recency by subtracting (last transaction date of dataset) and (last purchase date of each customer)
Recency['Recency'] = Recency['LastDate'].apply(lambda x: (last_date_dataset - x).days)
Recency.head()
```

Out[82]:

	memberID	LastDate	Recency
0	1000	2015-11-25	35
1	1001	2015-04-14	260
2	1002	2015-08-30	122
3	1003	2015-10-02	89
4	1004	2015-02-12	321

Recency Distribution of the Customers

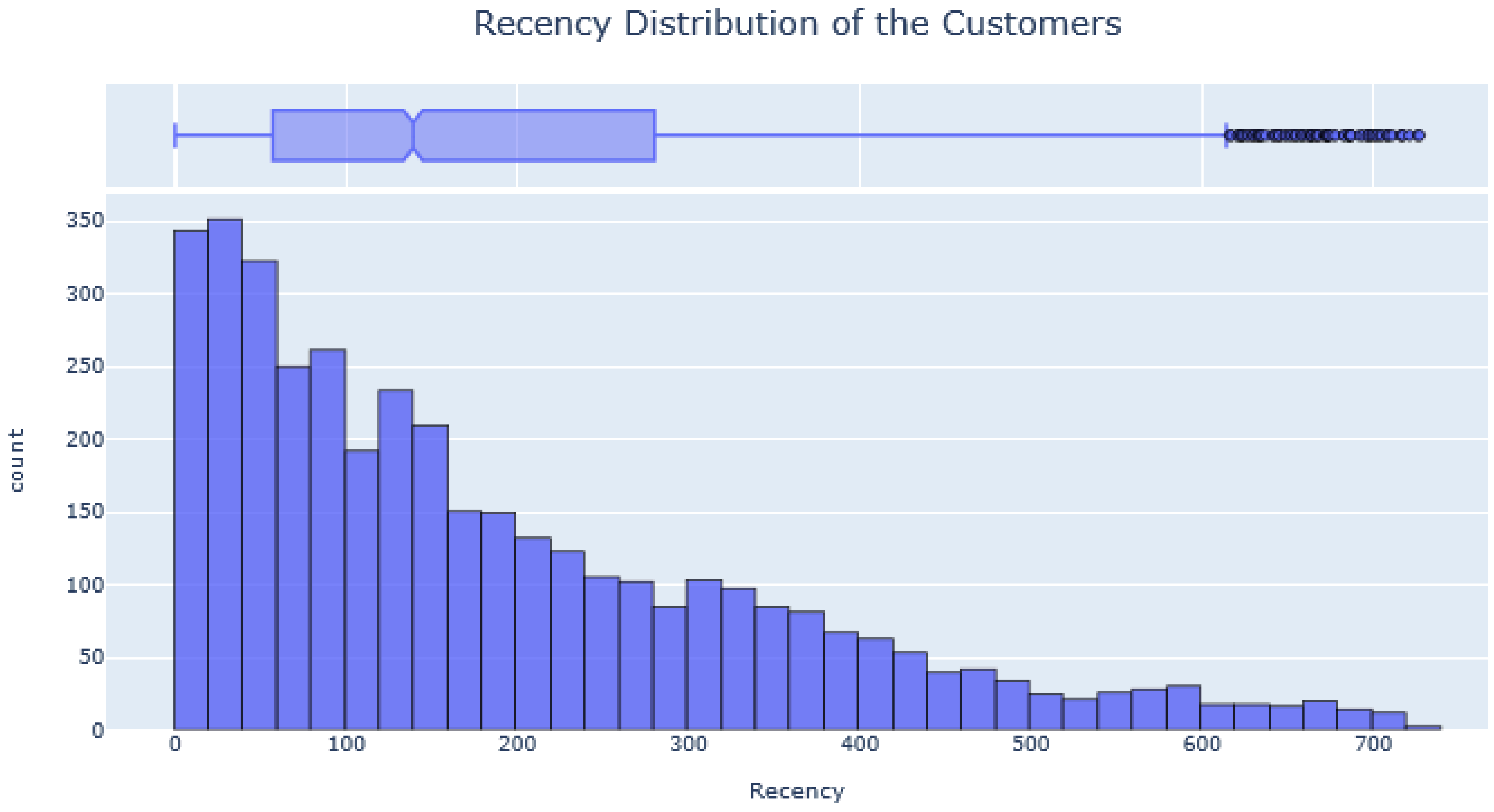
In [83]:
```
fig = px.histogram(Recency, x='Recency', opacity=0.85, marginal='box')
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)))
fig.update_layout(title_text='Recency Distribution of the Customers',
                  title_x=0.5, title_font=dict(size=20))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



According to Recency Historgram of the Customers, we can see that most of the customers are distributed between 57th-280th day of thier last purchase.

Visit Frequency

In [84]:
```
# Frequency of the customer visits
Frequency = data.drop_duplicates(['Date', 'memberID']).groupby(by=['memberID'])['Date'].count().reset_index()
Frequency.columns = ['memberID', 'Visit_Frequency']
Frequency.head()
```

Out[84]:

	memberID	Visit_Frequency
0	1000	5
1	1001	5
2	1002	4
3	1003	4
4	1004	8

Visit Frequency Distribution of the Customers

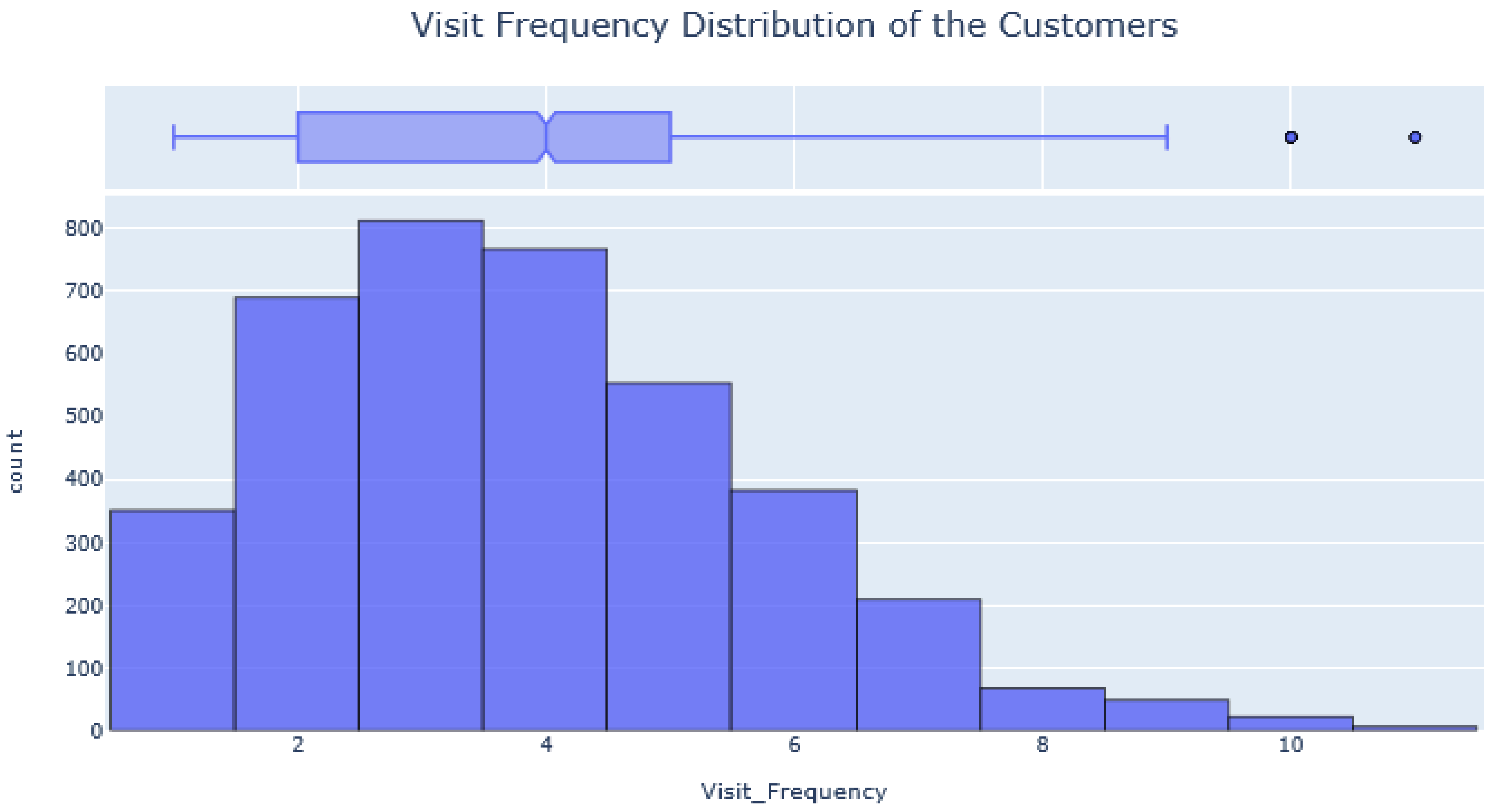
In [85]:
```
fig = px.histogram(Frequency, x='Visit_Frequency', opacity=0.85, marginal='box')
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)))
fig.update_layout(title_text='Visit Frequency Distribution of the Customers',
                  title_x=0.5, title_font=dict(size=20))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



According to Frequency Historgram of the Customers, we can see that most of the customers are distributed between their 2nd-5th visit to the Groceries Store.

Monetary

Due to our dataset, we have no data regarding the price of the products. Therefore I will consider the number of item bought per user as the Monetary value.

In [86]:
```
Monetary = data.groupby(by="memberID")['itemName'].count().reset_index()
Monetary.columns = ['memberID', 'Monetary']
Monetary.head()
```

Out[86]:

	memberID	Monetary
0	1000	13
1	1001	12
2	1002	8
3	1003	8
4	1004	21

In [87]:
```
# I assumed each item has equal price and price is 10
Monetary['Monetary'] = Monetary['Monetary'] * 10
Monetary.head()
```

Out[87]:

	memberID	Monetary
0	1000	130
1	1001	120
2	1002	80
3	1003	80
4	1004	210

Monetary Distribution of the Customers

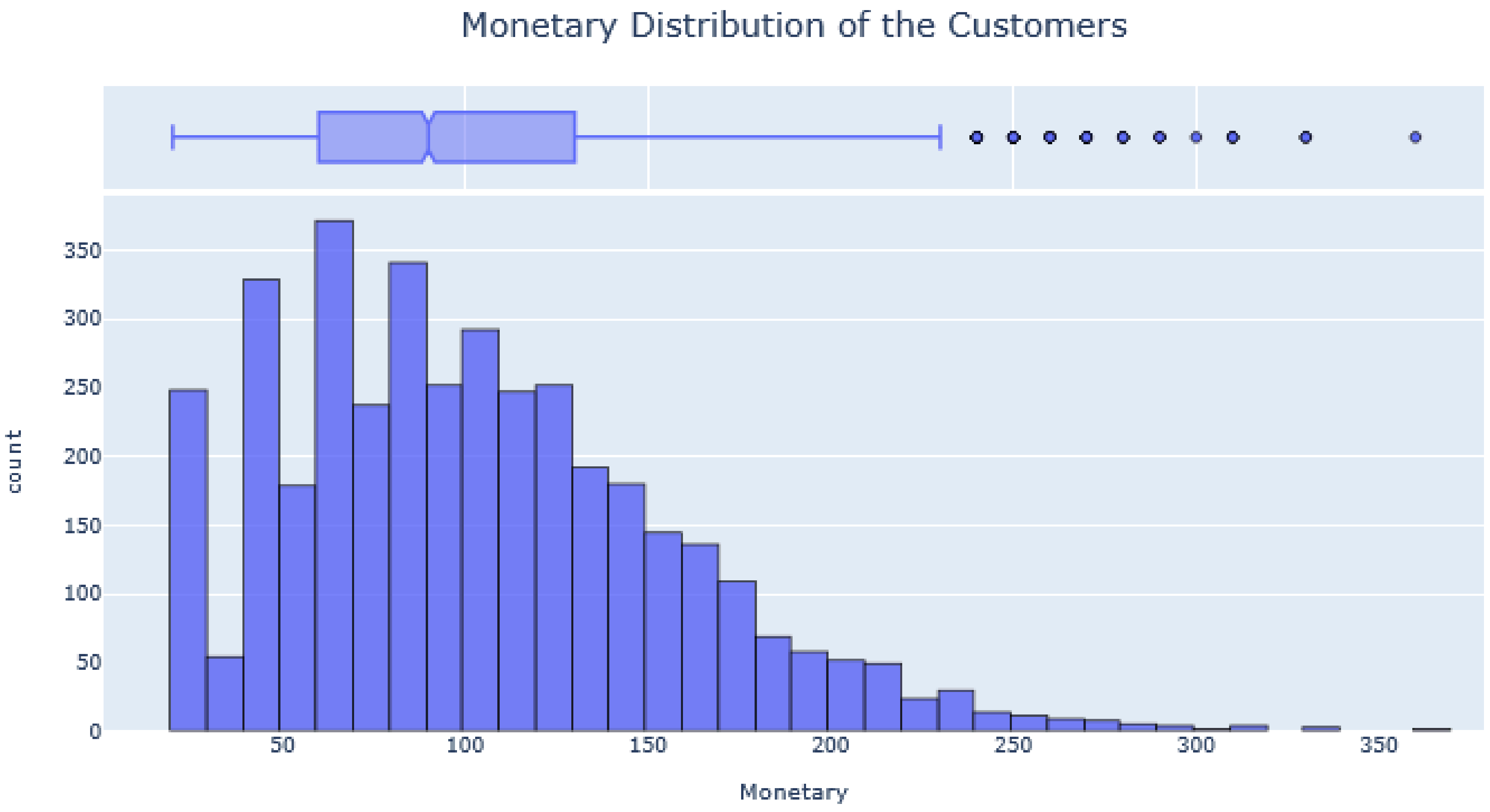
In [88]:
```
fig = px.histogram(Monetary, x='Monetary', opacity=0.85, marginal='box',
                   labels={'itemName': 'Monetary'})
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)))
fig.update_layout(title_text='Monetary Distribution of the Customers',
                  title_x=0.5, title_font=dict(size=20))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



In [89]:
```
# Combining all scores into one DataFrame
RFM = pd.concat([Recency['memberID'], Recency['Recency'], Frequency['Visit_Frequency'], Monetary['Monetary']], axis=1)
RFM.head()
```

Out[89]:

	memberID	Recency	Visit_Frequency	Monetary
0	1000	35	5	130
1	1001	260	5	120
2	1002	122	4	80
3	1003	89	4	80
4	1004	321	8	210

RFM Scores

In [90]:
```
# 5-5 score = the best customers
RFM['Recency_quartile'] = pd.qcut(RFM['Recency'], 5, [5, 4, 3, 2, 1])
RFM['Frequency_quartile'] = pd.qcut(RFM['Visit_Frequency'], 5, [1, 2, 3, 4, 5])

RFM['RF_Score'] = RFM['Recency_quartile'].astype(str) + RFM['Frequency_quartile'].astype(str)
RFM.head()
```

Out[90]:

	memberID	Recency	Visit_Frequency	Monetary	Recency_quartile	Frequency_quartile	RF_Score
0	1000	35	5	130	5	4	54
1	1001	260	5	120	2	4	24
2	1002	122	4	80	3	3	33
3	1003	89	4	80	4	3	43
4	1004	321	8	210	2	5	25

In [91]:
```

segt_map = {  # Segmentation Map [Ref]
    r'[1-2][1-2]': 'hibernating',
    r'[1-2][3-4]': 'at risk',
    r'[1-2]5': 'can\'t loose',
    r'3[1-2]': 'about to sleep',
    r'33': 'need attention',
    r'[3-4][4-5]': 'loyal customers',
    r'41': 'promising',
    r'51': 'new customers',
    r'[4-5][2-3]': 'potential loyalists',
    r'5[4-5]': 'champions'
}

RFM['RF_Segment'] = RFM['RF_Score'].replace(segt_map, regex=True)
RFM.head()
```

Out[91]:

	memberID	Recency	Visit_Frequency	Monetary	Recency_quartile	Frequency_quartile	RF_Score	RF_Segment
0	1000	35	5	130	5	4	54	champions
1	1001	260	5	120	2	4	24	at risk
2	1002	122	4	80	3	3	33	need attention
3	1003	89	4	80	4	3	43	potential loyalists
4	1004	321	8	210	2	5	25	can't loose

Distribution of the RFM Segments

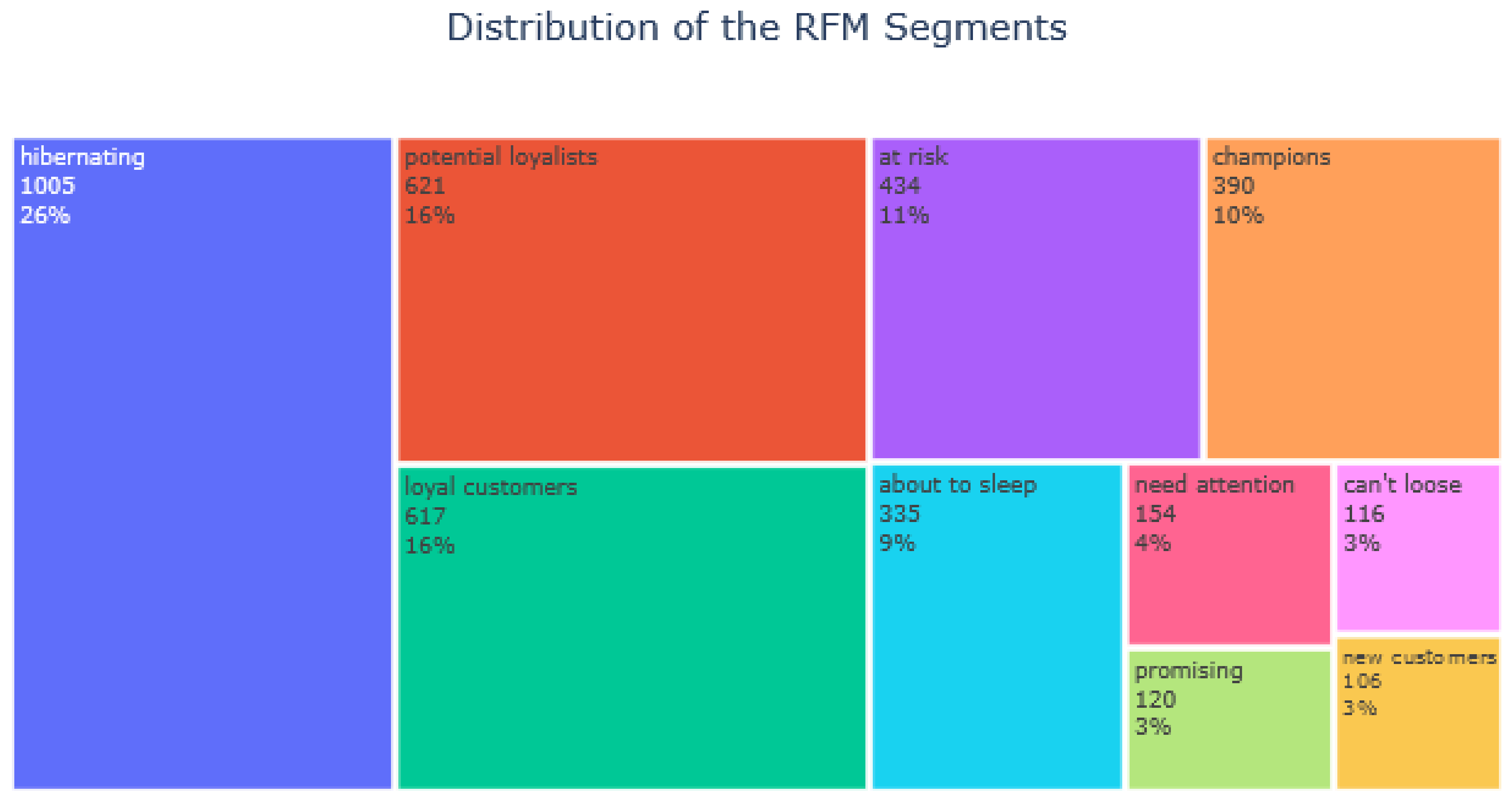
In [92]:
```
x = RFM.RF_Segment.value_counts()
fig = px.treemap(x, path=[x.index], values=x)
fig.update_layout(title_text='Distribution of the RFM Segments', title_x=0.5,
                  title_font=dict(size=20))
fig.update_traces(textinfo="label+value+percent root")
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



According to our RFM analysis, most of the customers are segmented into hibernating group which is they are not visiting our store often and it passed pretty much time after their last visit. We can find detailed information about these segments in the references part of this notebook.

Relationship between Visit_Frequency and Recency

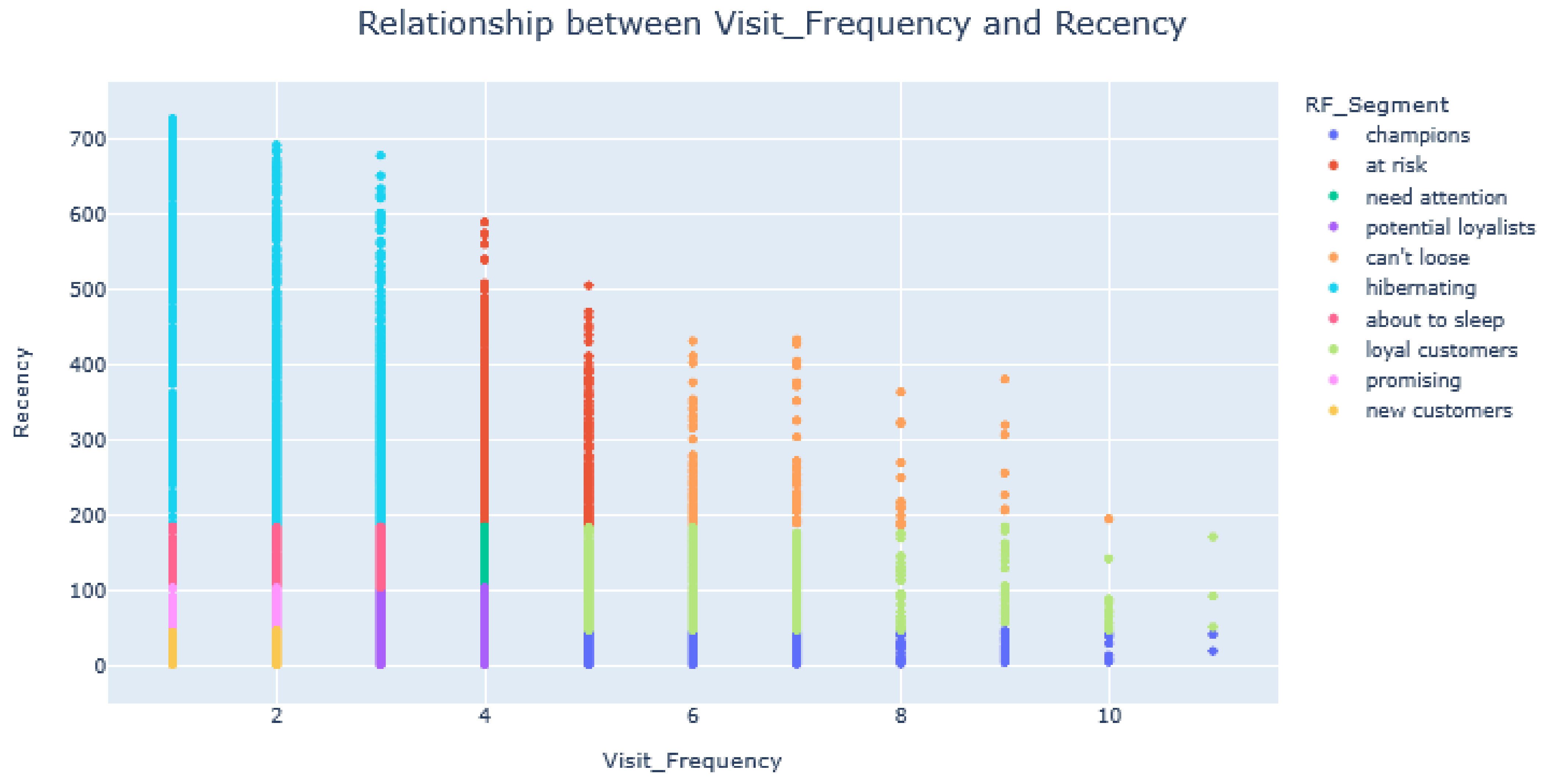
In [93]:
```
fig = px.scatter(RFM, x="Visit_Frequency", y="Recency", color='RF_Segment',
                 labels={"math score": "Math Score",
                         "writing score": "Writing Score"})
fig.update_layout(title_text='Relationship between Visit_Frequency and Recency',
                  title_x=0.5, title_font=dict(size=20))
fig.show()
```

```
C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

C:\Users\N347u\anaconda3\Lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



As we can see the graph above, when the visit frequency is low and the recency is high, customers are most likely to segmented into hibernating segment. In contrast, when they are visiting our store frequently, and their recency is low, they are most likely to segmented into champions segment which is the best segment for all of the customer segments.

Sources :

- https://www.datacamp.com/community/tutorials/introduction-customer-segmentation-python
- https://guillaume-martin.github.io/rfm-segmentation-with-python.html
- https://towardsdatascience.com/association-rules-2-aa9a77241654

Thank You