In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Loading the data
purchase_behaviour_data = pd.read_csv('data/QVI_purchase_behaviour.csv')  # CSV


transaction_data = pd.read_excel('data/QVI_transaction_data.xlsx')
```

In [3]:
```
# == Data understanding

# transaction data wrangling


transaction_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264836 entries, 0 to 264835
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   DATE            264836 non-null  int64  
 1   STORE_NBR       264836 non-null  int64  
 2   LYLTY_CARD_NBR  264836 non-null  int64  
 3   TXN_ID          264836 non-null  int64  
 4   PROD_NBR        264836 non-null  int64  
 5   PROD_NAME       264836 non-null  object 
 6   PROD_QTY        264836 non-null  int64  
 7   TOT_SALES       264836 non-null  float64
dtypes: float64(1), int64(6), object(1)
memory usage: 16.2+ MB
```

In [4]:
```
print("-"*30)
print("1. Col Names: ",list(transaction_data.columns))
print("-"*30)
print("2. No of rows&columns: ", transaction_data.shape)
print("-"*30)
```

```
------------------------------
1. Col Names:  ['DATE', 'STORE_NBR', 'LYLTY_CARD_NBR', 'TXN_ID', 'PROD_NBR', 'PRO
D_NAME', 'PROD_QTY', 'TOT_SALES']
------------------------------
2. No of rows&columns:  (264836, 8)
------------------------------
```

In [5]:
```
transaction_data.head()
```

Out[5]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_C
0	43390	1	1000	1	5	Natural Chip Compny SeaSalt175g	
1	43599	1	1307	348	66	CCs Nacho Cheese 175g	
2	43605	1	1343	383	61	Smiths Crinkle Cut Chips Chicken 170g	
3	43329	2	2373	974	69	Smiths Chip Thinly S/Cream&Onion 175g	
4	43330	2	2426	1038	108	Kettle Tortilla ChpsHny&Jlpno Chili 150g	

```
The value "43390" in the context of Excel and dates likely represents
a date in the Excel date system. In Excel, dates are internally stored
as serial numbers where each date is assigned a unique number. The
serial number represents the number of days since the base date, which
is January 1, 1900.
```

In [6]:
```
transaction_data.nunique() # Displayes the unique values
```

Out[6]:
```
DATE                 364
STORE_NBR            272
LYLTY_CARD_NBR     72637
TXN_ID            263127
PROD_NBR             114
PROD_NAME            114
PROD_QTY               6
TOT_SALES            112
dtype: int64
```

In [7]:
```
# Sorting the entier dataframe bY DATE column


sorted_trans_data = transaction_data.sort_values(by='DATE')
#sorted according to date
```

In [8]:
```
sorted_trans_data.head()
```

Out[8]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	P
100938	43282	19	19205	16466	26	Pringles Sweet&Spcy BBQ 134g	
65566	43282	189	189381	190189	84	GrnWves Plus Btroot & Chilli Jam 180g	
43733	43282	124	124236	127984	104	Infuzions Thai SweetChili PotatoMix 110g	
175455	43282	70	70131	68241	60	Kettle Tortilla ChpsFeta&Garlic 150g	
205813	43282	33	33140	30342	10	RRD SR Slow Rst Pork Belly 150g	

In [9]:

```
# srting value
excel_date_serial_number = 43282

# Convert Excel date serial number to Pandas datetime
date_object = pd.to_datetime(excel_date_serial_number, unit='D', origin='1900-01

# Create a range of dates for the next 364 days
next_364_days = pd.date_range(start=date_object, periods=364, freq='D')


#Creating a dict values for mapping to original data
dummy_dates = list(sorted_trans_data['DATE'].unique()) # Presnt values
mapping_func_for_date = {dum : org for dum,org in zip(dummy_dates,next_364_days)

# adding actual values of DATE column

sorted_trans_data['DATE'] = sorted_trans_data['DATE'].map(mapping_func_for_date)
```

In [10]:

```
sorted_trans_data.isnull().sum() # No null values in the transcation data
```

Out[10]:

```
DATE              0
STORE_NBR         0
LYLTY_CARD_NBR    0
TXN_ID            0
PROD_NBR          0
PROD_NAME         0
PROD_QTY          0
TOT_SALES         0
dtype: int64
```

In [11]:

```
sorted_trans_data.describe()
```

Out[11]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR
count	264836	264836.00000	2.648360e+05	2.648360e+05	264836.000000
mean	2018-12-31 12:35:23.820024064	135.08011	1.355495e+05	1.351583e+05	56.583157
min	2018-07-03 00:00:00	1.00000	1.000000e+03	1.000000e+00	1.000000
25%	2018-10-02 00:00:00	70.00000	7.002100e+04	6.760150e+04	28.000000
50%	2018-12-31 00:00:00	130.00000	1.303575e+05	1.351375e+05	56.000000
75%	2019-04-01 00:00:00	203.00000	2.030942e+05	2.027012e+05	85.000000
max	2019-07-01 00:00:00	272.00000	2.373711e+06	2.415841e+06	114.000000
std	NaN	76.78418	8.057998e+04	7.813303e+04	32.826638

Based on the above analysis, there are not many outliers in the data.

There is a outlier in the `TOT_SALES` and also `PROD_QTY`

In [12]:

```
non_chip_items = [
    "GrnWves Plus Btroot & Chilli Jam 180g",
    "RRD SR Slow Rst Pork Belly 150g",
    "Red Rock Deli Sp Salt & Truffle 150G",
    "WW Original Stacked Chips 160g",
    "Woolworths Medium Salsa 300g",
    "RRD Steak & Chimuchurri 150g",
    "Infzns Crn Crnchers Tangy Gcamole 110g",
    "Old El Paso Salsa Dip Tomato Med 300g",
    "RRD Sweet Chilli & Sour Cream 165g",
    "Old El Paso Salsa Dip Tomato Mild 300g",
    "RRD Pc Sea Salt 165g",
    "Twisties Cheese Burger 250g",
    "Tyrrells Crisps Ched & Chives 165g",
    "Tyrrells Crisps Lightly Salted 165g",
    "Grain Waves Sour Cream&Chives 210G",
    "Snbts Whlgrn Crisps Cheddr&Mstrd 90g",
    "Burger Rings 220g",
    "Cheetos Puffs 165g",
    "Cheezels Cheese 330g",
    "Sunbites Whlegrn Crisps Frch/Onin 90g",
    "WW Crinkle Cut Chicken 175g",
    "Woolworths Cheese Rings 190g",
    "Natural Chip Co Tmato Hrb&Spce 175g",
    "WW D/Style Chip Sea Salt 200g",
    "Natural ChipCo Hony Soy Chckn175g",
]
```

```
def check_chips_name(item):
    #Checks for chips names
    if item in non_chip_items: #If not returns "NON"
        return "NON"
    else: #else returns item
        return item



sorted_trans_data['PROD_NAME'] = sorted_trans_data['PROD_NAME'].apply(check_chip

# Replace the NON
index_of_NON = list(sorted_trans_data[sorted_trans_data['PROD_NAME'] == "NON"].i
sorted_trans_data.drop(index_of_NON ,inplace=True)

# BAck to normal index
sorted_trans_data.reset_index(inplace=True)
sorted_trans_data.drop(['index'],axis=1)
```

Out[12]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PI
0	2018-07-03	19	19205	16466	26	Pringles Sweet&Spcy BBQ 134g	
1	2018-07-03	124	124236	127984	104	Infuzions Thai SweetChili PotatoMix 110g	
2	2018-07-03	70	70131	68241	60	Kettle Tortilla ChpsFeta&Garlic 150g	
3	2018-07-03	33	33140	30342	10	RRD SR Slow Rst Pork Belly 150g	
4	2018-07-03	18	18221	15451	80	Natural ChipCo Sea Salt & Vinegr 175g	
...	...	...	...	...	...	...	
250995	2019-07-01	97	97085	96824	33	Cobs Popd Swt/Chlli &Sr/Cream Chips 110g	
250996	2019-07-01	148	148317	148317	112	Tyrrells Crisps Ched & Chives 165g	
250997	2019-07-01	212	212068	210874	113	Twisties Chicken270g	
250998	2019-07-01	55	55029	48630	2	Cobs Popd Sour Crm &Chives Chips 110g	
250999	2019-07-01	247	247060	248955	12	Natural Chip Co Tmato Hrb&Spce 175g	

251000 rows × 8 columns

In [13]:
```
sorted_trans_data.to_csv("data/Cleaned_data.csv",index=False)
```

Cleaning Path

- Null Values
- Unique Values
- Outliers
- Date columns
- typecasting data
- proper column names

Purchase data

In [14]:
```
purchase_behaviour_data.head()
```

Out[14]:

	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER
0	1000	YOUNG SINGLES/COUPLES	Premium
1	1002	YOUNG SINGLES/COUPLES	Mainstream
2	1003	YOUNG FAMILIES	Budget
3	1004	OLDER SINGLES/COUPLES	Mainstream
4	1005	MIDAGE SINGLES/COUPLES	Mainstream

In [15]:
```
purchase_behaviour_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72637 entries, 0 to 72636
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   LYLTY_CARD_NBR    72637 non-null  int64 
 1   LIFESTAGE         72637 non-null  object
 2   PREMIUM_CUSTOMER  72637 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.7+ MB
```

In [16]:
```
purchase_behaviour_data.isnull().sum()
```

Out[16]:
```
LYLTY_CARD_NBR      0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

In [17]:
```
purchase_behaviour_data.nunique()
```

Out[17]:
```
LYLTY_CARD_NBR      72637
LIFESTAGE               7
PREMIUM_CUSTOMER        3
dtype: int64
```

Every thing is Normal

Analysis

Certainly! Here's a step-by-step list of headlines for Exploratory Data Analysis (EDA):

1. **Load the Data:**
   - Read the dataset into a Pandas DataFrame.
2. **Understand the Structure:**

- Display basic information about the dataset (columns, data types, missing values).

3. **Explore Summary Statistics:**
   - Calculate and examine descriptive statistics (mean, median, min, max, etc.) for numeric columns.
4. **Handle Missing Data:**
   - Identify and handle missing values appropriately (impute or remove).
5. **Explore Categorical Variables:**
   - Analyze unique values, frequency distribution, and explore relationships in categorical columns.
6. **Visualize Data Distribution:**
   - Create histograms or kernel density plots to visualize the distribution of numeric variables.
7. **Analyze Relationships:**
   - Explore correlations between variables using correlation matrices or pair plots.
8. **Visualize Categorical Data:**
   - Utilize bar charts, count plots, or box plots to visualize relationships in categorical data.
9. **Identify Outliers:**
   - Use box plots or scatter plots to identify potential outliers in the dataset.
10. **Time Series Analysis (if applicable):**
    - Explore trends, seasonality, and patterns in time series data.
11. **Feature Engineering:**
    - Create new features or modify existing ones to enhance model performance.
12. **Address Skewness and Transformation:**
    - Examine and address skewed distributions; consider log transformations if needed.
13. **Explore Target Variable:**
    - Understand the distribution and characteristics of the target variable (for supervised learning tasks).
14. **Bivariate Analysis:**
    - Analyze relationships between pairs of variables using scatter plots or other appropriate visualizations.
15. **Grouping and Aggregation:**
    - Group data and perform aggregations to gain insights, especially in categorical variables.
16. **Feature Importance:**

- If applicable, explore feature importance for predictive modeling tasks.

17. **Data Visualization for Insights:**
    - Use various visualizations (line plots, bar charts, heatmaps) to derive insights and patterns.
18. **Statistical Testing:**
    - Conduct statistical tests to validate assumptions or hypotheses.
19. **Summary and Conclusions:**
    - Summarize key findings, insights, and conclusions drawn from the EDA.
20. **Documentation:**
    - Document the entire EDA process, including any decisions made, transformations performed, and insights gained.

Remember that the specific steps may vary depending on the nature of your data and the goals of your analysis.

In [18]:
```
#Merg two dataframes

merged_data  = sorted_trans_data.merge(purchase_behaviour_data ,on='LYLTY_CARD_N
```

In [19]:
```
merged_data.info()
merged_data.drop('index',axis=1,inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 251000 entries, 0 to 250999
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   index             251000 non-null  int64
 1   DATE              251000 non-null  datetime64[ns]
 2   STORE_NBR         251000 non-null  int64
 3   LYLTY_CARD_NBR    251000 non-null  int64
 4   TXN_ID            251000 non-null  int64
 5   PROD_NBR          251000 non-null  int64
 6   PROD_NAME         251000 non-null  object
 7   PROD_QTY          251000 non-null  int64
 8   TOT_SALES         251000 non-null  float64
 9   LIFESTAGE         251000 non-null  object
 10  PREMIUM_CUSTOMER  251000 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(6), object(3)
memory usage: 21.1+ MB
```

In [20]:
```
merged_data.sample(4)
```

Out[20]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR	PROD_NAME	PRO
140986	2019-02-20	271	271132	268980	65	Old El Paso Salsa Dip Chnky Tom Ht300g	
223876	2019-01-31	115	115224	118782	102	Kettle Mozzarella Basil & Pesto 175g	
50746	2019-04-26	166	166154	167833	15	Twisties Cheese 270g	
32065	2018-11-17	196	196072	195872	66	CCs Nacho Cheese 175g	

`PREMIUM_CUSTOMER: Customer segmentation used to differentiate shoppers by the price point of products they buy and the types of products they buy. It is used to identify whether customers may spend more for quality or brand or whether they will purchase the cheapest options.`

1. **Mainstream:**

   - **Definition:** Mainstream customers are typically those who prefer a balance between quality and price. They are not exclusively focused on premium products but are willing to pay for reasonably good quality.
   - **Example:** A customer who regularly buys well-known brands but is not exclusively loyal to the most expensive options. They may opt for popular and widely available products that offer a good combination of quality and affordability.

2. **Budget:**

   - **Definition:** Budget customers are price-conscious and prioritize affordability over brand names or premium quality. They are often looking for the most cost-effective options available.
   - **Example:** A customer who actively seeks discounts, buys generic or store-brand products, and is primarily motivated by getting the best possible deal. They may opt for lower-cost alternatives to save money.

3. **Premium:**

   - **Definition:** Premium customers are those who prioritize high-quality products and are willing to pay a premium price for superior features or brand prestige. They are not as concerned with cost savings and are more focused on getting top-tier products.
   - **Example:** A customer who consistently chooses luxury or high-end brands, values exclusive features, and is willing to pay extra for superior quality. They may opt for premium options in various product categories.

In [21]:
```
merged_data.select_dtypes(include='object').describe()
```

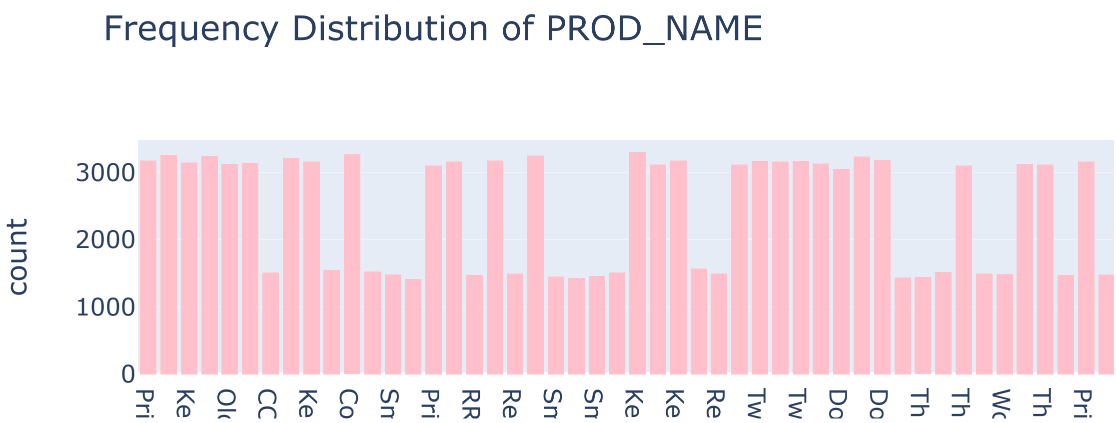
Out[21]:

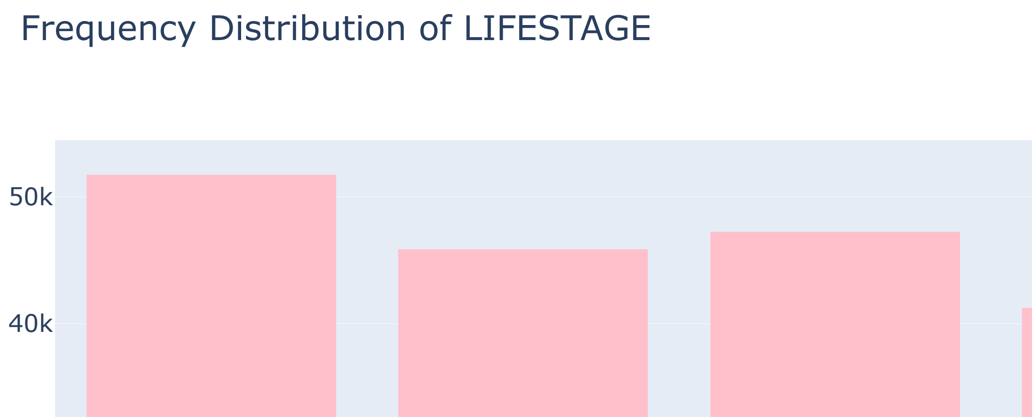
	PROD_NAME	LIFESTAGE	PREMIUM_CUSTOMER
count	251000	251000	251000
unique	107	7	3
top	Kettle Mozzarella Basil & Pesto 175g	OLDER SINGLES/COUPLES	Mainstream
freq	3304	51732	96824

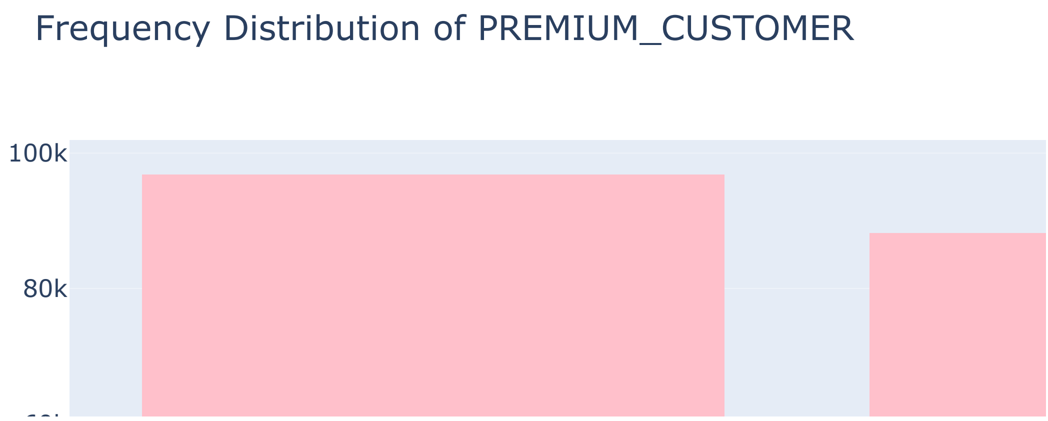
Graphical analysis

In [22]:
```
import plotly.express as px


for col in merged_data.select_dtypes(include='object').columns:
    # Assuming 'Category' is a categorical column in your DataFrame
    fig = px.histogram(merged_data, x=col, title=f'Frequency Distribution of {cc
    fig.show()
```




Frequency Distribution of LIFESTAGE
50k
40k



LIFE STAGE

`OLDER AGE` PEOPLE ARE MORE TEND TO BUY CHIPS `NEW FAMILIES` ARE LESS TENDS TO BUY CHIPS

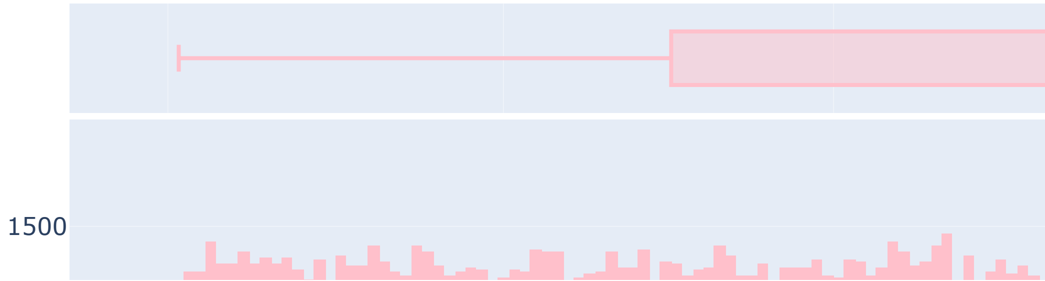
PREMIUM CUSTOMERS

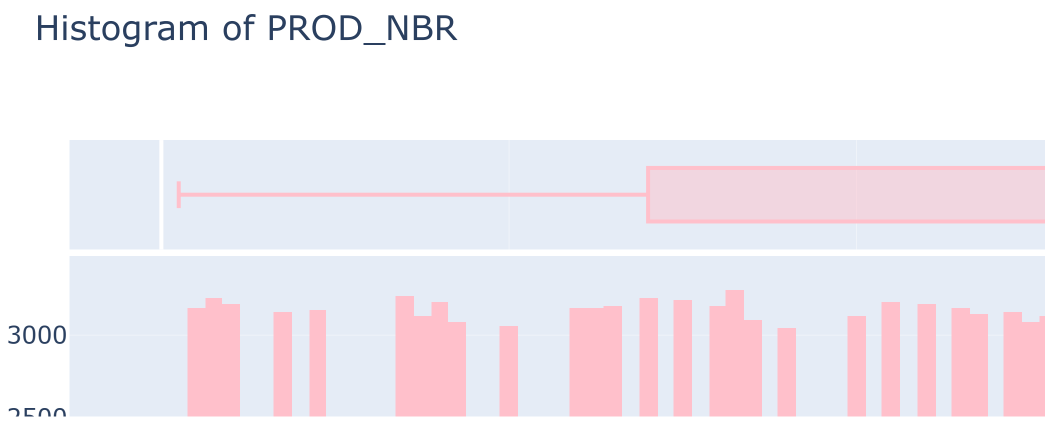
`MAINSTREAM` PEOPLES ARE MORE BUYINH THE CHIPS

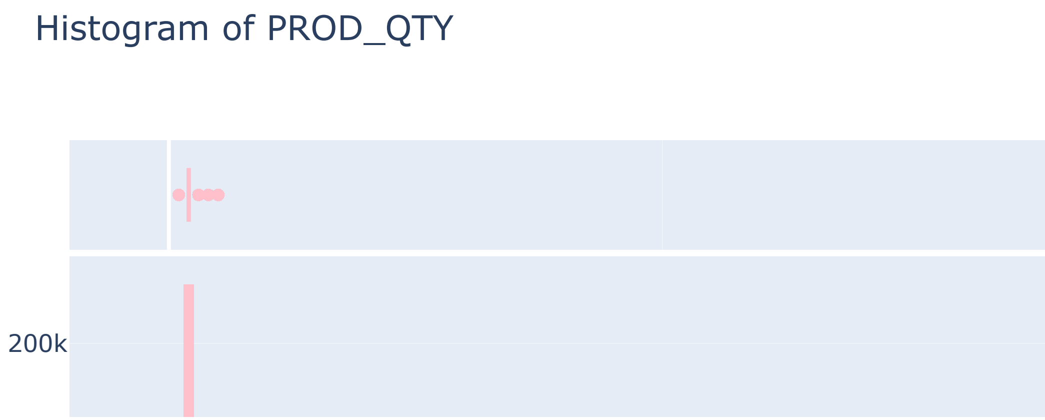
In [23]:
```
import plotly.express as px


for col in merged_data.select_dtypes(exclude='object')[1:]:
    # Assuming 'NumericVariable' is a numeric column in your DataFrame
    fig = px.histogram(merged_data, x=col, title=f'Histogram of {col}',marginal=
    fig.show()
```

Histogram of DATE




Histogram of PROD_NBR
3000



PROD QUNTITY

THERE IS LARGE PURCHASE OF `200` PACKETS WORTH `650`

In [24]:
```
merged_data['STORE_NBR'].value_counts()[:5]
```

Out[24]:
```
STORE_NBR
226    1948
88     1787
165    1739
93     1733
237    1717
Name: count, dtype: int64
```

These are the stores which have most number of transactions

In [25]:
```
print(merged_data[['PROD_NBR','PROD_NAME']].value_counts()[:5])
```

```
PROD_NBR  PROD_NAME
102       Kettle Mozzarella   Basil & Pesto 175g      3304
108       Kettle Tortilla ChpsHny&Jlpno Chili 150g    3296
33        Cobs Popd Swt/Chlli &Sr/Cream Chips 110g    3269
112       Tyrrells Crisps     Ched & Chives 165g      3268
75        Cobs Popd Sea Salt  Chips 110g              3265
Name: count, dtype: int64
```

These are the products have highest purchase history

In [26]:
```
(merged_data[['TOT_SALES','LIFESTAGE','PREMIUM_CUSTOMER']])
```

Out[26]:

	TOT_SALES	LIFESTAGE	PREMIUM_CUSTOMER
0	3.7	OLDER SINGLES/COUPLES	Mainstream
1	8.4	OLDER SINGLES/COUPLES	Mainstream
2	10.8	OLDER SINGLES/COUPLES	Mainstream
3	3.8	OLDER FAMILIES	Budget
4	5.1	OLDER FAMILIES	Budget
...	...	...	...
250995	5.4	YOUNG SINGLES/COUPLES	Mainstream
250996	7.4	MIDAGE SINGLES/COUPLES	Mainstream
250997	9.2	YOUNG SINGLES/COUPLES	Budget
250998	5.4	OLDER SINGLES/COUPLES	Mainstream
250999	8.8	YOUNG SINGLES/COUPLES	Mainstream

251000 rows × 3 columns

Summary of the Data

`The client is particularly interested in customer segments and their chip purchasing behaviour.`

In [27]:
```
merged_data[['DATE','TOT_SALES']]
```

Out[27]:

	DATE	TOT_SALES
0	2018-07-03	3.7
1	2018-08-21	8.4
2	2018-09-20	10.8
3	2018-07-03	3.8
4	2018-11-09	5.1
...	...	...
250995	2019-07-01	5.4
250996	2019-07-01	7.4
250997	2019-07-01	9.2
250998	2019-07-01	5.4
250999	2019-07-01	8.8

251000 rows × 2 columns

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: