Data Transformations and EDA - Retail Data Analysis

Prerequisites:

Install all the libraries listed below in order to perform EDA for this project.

- pip install numpy
- pip install pandas
- pip install pandas_profiling
- pip install seaborn
- pip install jupyter_scheduler
- pip install jupyterlab-scheduler
- pip install snowflake-connector-python
- pip install snowflake-sqlalchemy
- pip install "snowflake-connector-python[pandas]"

Import all the required Libraries:

In [1]:
```
import numpy as np              # For numerical operations and array manipulatio
import pandas as pd             # For data manipulation and analysis
import ydata_profiling          # For creating detailed data reports
import matplotlib.pyplot as plt # For data visualization
import getpass                  # For securely handling user input (like passwor
import seaborn as sns           # For statistical data visualization
import snowflake.connector      # For connecting to Snowflake
```

In [2]:
```
# Ensures that plots appear within the notebook
%matplotlib inline
```

Establishing a Connection Between Snowflake and Python:

- Ensure that the user, password, and other connection details are accurate and have the necessary permissions to establish a successful connection.

In [3]:
```
conn = snowflake.connector.connect(
        user = 'JSANTU',
        # password = getpass.getpass('Your Snowflake Password: '),
        password = '**********',
        account = 'KMMGNKB-XC40527',
        database='RETAILS',
        schema='PUBLIC',
        warehouse='COMPUTE_WH'
        )
```

```
print('Connection established successfully with the Snowflake')
```

```
Connection established successfully with the Snowflake
```

- Now, create a cursor object to interact with the Snowflake database. This will help us to run queries and fetch data from the Snowflake database.

In [4]:
```
cur = conn.cursor()
```

- Creating variables for each table to hold the query, which can be used to fetch all records from the respective table in Snowflake.

In [5]:
```
select_CAMPAIGN_DESC_RAW = 'SELECT * FROM CAMPAIGN_DESC_RAW'
select_CAMPAIGN_RAW = 'SELECT * FROM CAMPAIGN_RAW'
select_COUPON_RAW = 'SELECT * FROM COUPON_RAW'
select_COUPON_REDEMPT_RAW = 'SELECT * FROM COUPON_REDEMPT_RAW'
select_DEMOGRAPHIC_RAW = 'SELECT * FROM DEMOGRAPHIC_RAW'
select_PRODUCT_RAW = 'SELECT * FROM PRODUCT_RAW'
select_TRANSACTION_RAW = 'SELECT * FROM TRANSACTION_RAW'
```

- Executing the queries using the above variables to fetch all tables and their records from the `RETAILS` database in Snowflake.
- The `fetch_pandas_all()` method is a convenient way to directly fetch query results into a pandas DataFrame. This will retrieve all the rows from each table and store them in the respective `pandas DataFrames`. We can then manipulate and analyze this data using pandas functions.

In [6]:
```
cur.execute(select_CAMPAIGN_DESC_RAW)
CAMPAIGN_DESC_RAW = cur.fetch_pandas_all()
```

In [7]:
```
cur.execute(select_CAMPAIGN_RAW)
CAMPAIGN_RAW = cur.fetch_pandas_all()
```

In [8]:
```
cur.execute(select_COUPON_RAW)
COUPON_RAW = cur.fetch_pandas_all()
```

In [9]:
```
cur.execute(select_COUPON_REDEMPT_RAW)
COUPON_REDEMPT_RAW = cur.fetch_pandas_all()
```

In [10]:
```
cur.execute(select_DEMOGRAPHIC_RAW)
DEMOGRAPHIC_RAW = cur.fetch_pandas_all()
```

In [11]:
```
cur.execute(select_PRODUCT_RAW)
PRODUCT_RAW = cur.fetch_pandas_all()
```

In [12]:
```
cur.execute(select_TRANSACTION_RAW)
TRANSACTION_RAW = cur.fetch_pandas_all()
```

- Once the connection is successfully established and data is retrieved, always close the cursor and connection.

- Closing the cursor and connection is a good practice to ensure the efficient and secure use of resources.
- Even after closing the cursor and connection, we can still access the data because it is already copied into the DataFrame.

In [13]:
```
cur.close()
conn.close()
```

In [14]:
```
if cur.is_closed():
    print("The cursor is closed.")
else:
    print("The cursor is still open.")

if conn.is_closed():
    print("The connection is closed.")
else:
    print("The connection is still open.")
```

```
The cursor is closed.
The connection is closed.
```

In [15]:
```
CAMPAIGN_DESC_RAW.head(5)
```

Out[15]:

	DESCRIPTION	CAMPAIGN	START_DAY	END_DAY
0	TypeB	24	659	719
1	TypeC	15	547	708
2	TypeB	25	659	691
3	TypeC	20	615	685
4	TypeB	23	646	684

In [16]:
```
CAMPAIGN_RAW.head(5)
```

Out[16]:

	DESCRIPTION	HOUSEHOLD_KEY	CAMPAIGN
0	TypeA	17	26
1	TypeA	27	26
2	TypeA	212	26
3	TypeA	208	26
4	TypeA	192	26

In [17]:
```
COUPON_RAW.head(5)
```

Out[17]:

	COUPON_UPC	PRODUCT_ID	CAMPAIGN
0	10000089061	27160	4
1	10000089064	27754	9
2	10000089073	28897	12
3	51800009050	28919	28
4	52100000076	28929	25

In [18]:
```
COUPON_REDEMPT_RAW.head(5)
```

Out[18]:

	HOUSEHOLD_KEY	DAY	COUPON_UPC	CAMPAIGN
0	1	421	10000085364	8
1	1	421	51700010076	8
2	1	427	54200000033	8
3	1	597	10000085476	18
4	1	597	54200029176	18

In [19]:
```
DEMOGRAPHIC_RAW.head(5)
```

Out[19]:

	AGE_DESC	MARITAL_STATUS_CODE	INCOME_DESC	HOMEOWNER_DESC	HH_COMP_I
0	65+	A	35-49K	Homeowner	2 Adults No
1	45-54	A	50-74K	Homeowner	2 Adults No
2	25-34	U	25-34K	Unknown	2 Adults
3	25-34	U	75-99K	Homeowner	2 Adults
4	45-54	B	50-74K	Homeowner	Single Fe

In [20]:
```
PRODUCT_RAW.head(5)
```

Out[20]:

	PRODUCT_ID	MANUFACTURER	DEPARTMENT	BRAND	COMMODITY_DESC	SUB_CC
0	25671	2	GROCERY	National	FRZN ICE	ICE -
1	26081	2	MISC. TRANS.	National	NO COMMODITY DESCRIPTION	NO
2	26093	69	PASTRY	Private	BREAD	BREAD
3	26190	69	GROCERY	Private	FRUIT - SHELF STABLE	
4	26355	69	GROCERY	Private	COOKIES/CONES	SF

In [21]:
```
TRANSACTION_RAW.head(5)
```

Out[21]:

	HOUSEHOLD_KEY	BASKET_ID	DAY	PRODUCT_ID	QUANTITY	SALES_VALUE	STOR
0	2056	29330027026	177	1106523	1	2.32	
1	2056	29330027026	177	5567325	1	2.69	
2	2056	29330027027	177	1106523	1	2.32	
3	1873	29330097522	177	1135681	1	1.29	
4	1602	29339301484	177	828427	1	0.22	

Examining Data:

`CAMPAIGN_DESC_RAW`

In [22]:
```
# checking total rows and columns
CAMPAIGN_DESC_RAW.shape
```

Out[22]: (30, 4)

In [23]:
```
# statistical description of numerical columns
CAMPAIGN_DESC_RAW.describe()
```

Out[23]:

	CAMPAIGN	START_DAY	END_DAY
count	30.000000	30.000000	30.000000
mean	15.500000	463.866667	510.466667
std	8.803408	134.488490	137.730555
min	1.000000	224.000000	264.000000
25%	8.250000	360.000000	405.750000
50%	15.500000	470.000000	502.000000
75%	22.750000	584.000000	640.250000
max	30.000000	659.000000	719.000000

In [24]:
```
# checking the count of missing/null values for each columns
CAMPAIGN_DESC_RAW.isnull().sum()
```

Out[24]:
```
DESCRIPTION    0
CAMPAIGN       0
START_DAY      0
END_DAY        0
dtype: int64
```

`CAMPAIGN_RAW`

In [25]:
```
CAMPAIGN_RAW.shape
```

Out[25]: (7208, 3)

In [26]:
```
CAMPAIGN_RAW.describe()
```

Out[26]:

	HOUSEHOLD_KEY	CAMPAIGN
count	7208.000000	7208.000000
mean	1268.696726	15.659545
std	731.936391	6.949705
min	1.000000	1.000000
25%	644.000000	9.000000
50%	1263.000000	14.000000
75%	1924.000000	20.000000
max	2500.000000	30.000000

In [27]:
```
CAMPAIGN_RAW.isnull().sum()
```

Out[27]:
```
DESCRIPTION      0
HOUSEHOLD_KEY    0
CAMPAIGN         0
dtype: int64
```

COUPON_RAW

In [28]:
```
COUPON_RAW.shape
```

Out[28]:
```
(124548, 3)
```

In [29]:
```
COUPON_RAW.describe()
```

Out[29]:

	COUPON_UPC	PRODUCT_ID	CAMPAIGN
count	1.245480e+05	1.245480e+05	124548.000000
mean	2.198225e+10	4.651277e+06	15.855935
std	1.961641e+10	4.843450e+06	6.015524
min	1.000009e+10	2.567100e+04	1.000000
25%	1.000009e+10	9.593030e+05	13.000000
50%	1.000009e+10	1.411451e+06	14.000000
75%	5.111108e+10	8.357538e+06	18.000000
max	5.998660e+10	1.814854e+07	30.000000

In [30]:
```
COUPON_RAW.isnull().sum()
```

Out[30]:
```
COUPON_UPC    0
PRODUCT_ID    0
CAMPAIGN      0
dtype: int64
```

`COUPON_REDEMPT_RAW`

In [31]:
```
COUPON_REDEMPT_RAW.shape
```

Out[31]:
```
(2318, 4)
```

In [32]:
```
COUPON_REDEMPT_RAW.describe()
```

Out[32]:

	HOUSEHOLD_KEY	DAY	COUPON_UPC	CAMPAIGN
count	2318.000000	2318.000000	2.318000e+03	2318.000000
mean	1302.816652	528.217429	4.123049e+10	15.551337
std	783.002545	100.360985	1.986068e+10	5.716636
min	1.000000	225.000000	1.000009e+10	1.000000
25%	588.000000	458.250000	1.000009e+10	13.000000
50%	1396.500000	532.000000	5.234003e+10	14.000000
75%	2004.000000	605.000000	5.430002e+10	18.000000
max	2500.000000	704.000000	5.897850e+10	30.000000

In [33]:
```
COUPON_REDEMPT_RAW.isnull().sum()
```

Out[33]:
```
HOUSEHOLD_KEY    0
DAY              0
COUPON_UPC       0
CAMPAIGN         0
dtype: int64
```

`DEMOGRAPHIC_RAW`

In [34]:
```
DEMOGRAPHIC_RAW.shape
```

Out[34]:
```
(2500, 8)
```

In [35]:
```
DEMOGRAPHIC_RAW.describe()
```

Out[35]:

	HOUSEHOLD_KEY
count	2500.00000
mean	1250.50000
std	721.83216
min	1.00000
25%	625.75000
50%	1250.50000
75%	1875.25000
max	2500.00000

In [36]:
```
DEMOGRAPHIC_RAW.isnull().sum()
```

Out[36]:
```
AGE_DESC               0
MARITAL_STATUS_CODE    0
INCOME_DESC            0
HOMEOWNER_DESC         0
HH_COMP_DESC           0
HOUSEHOLD_SIZE_DESC    0
KID_CATEGORY_DESC      0
HOUSEHOLD_KEY          0
dtype: int64
```

`PRODUCT_RAW`

In [37]:
```
PRODUCT_RAW.shape
```

Out[37]:
```
(92353, 7)
```

In [38]:
```
PRODUCT_RAW.describe()
```

Out[38]:

	PRODUCT_ID	MANUFACTURER
count	9.235300e+04	92353.00000
mean	5.328353e+06	1739.22833
std	5.359937e+06	1818.26957
min	2.567100e+04	1.00000
25%	9.706280e+05	328.00000
50%	1.621091e+06	1094.00000
75%	9.704770e+06	2264.00000
max	1.831630e+07	6477.00000

In [39]:
```
PRODUCT_RAW.isnull().sum()
```

Out[39]:
```
PRODUCT_ID              0
MANUFACTURER            0
DEPARTMENT              0
BRAND                   0
COMMODITY_DESC          0
SUB_COMMODITY_DESC      0
CURR_SIZE_OF_PRODUCT    0
dtype: int64
```

`TRANSACTION_RAW`

In [40]:
```
TRANSACTION_RAW.shape
```

Out[40]:
```
(2595732, 12)
```

In [41]:
```
TRANSACTION_RAW.describe()
```

Out[41]:

	HOUSEHOLD_KEY	BASKET_ID	DAY	PRODUCT_ID	QUANTITY	SAL
count	2.595732e+06	2.595732e+06	2.595732e+06	2.595732e+06	2.595732e+06	2.59
mean	1.271953e+03	3.402620e+10	3.887562e+02	2.891435e+06	1.004286e+02	3.10
std	7.260660e+02	4.711649e+09	1.897210e+02	3.837404e+06	1.153436e+03	4.18
min	1.000000e+00	2.698485e+10	1.000000e+00	2.567100e+04	0.000000e+00	0.00
25%	6.560000e+02	3.040805e+10	2.290000e+02	9.174590e+05	1.000000e+00	1.29
50%	1.272000e+03	3.276081e+10	3.900000e+02	1.028816e+06	1.000000e+00	2.00
75%	1.913000e+03	4.012685e+10	5.530000e+02	1.133018e+06	1.000000e+00	3.49
max	2.500000e+03	4.230536e+10	7.110000e+02	1.831630e+07	8.963800e+04	8.40

In [42]:
```
TRANSACTION_RAW.isnull().sum()
```

Out[42]:
```
HOUSEHOLD_KEY        0
BASKET_ID            0
DAY                  0
PRODUCT_ID           0
QUANTITY             0
SALES_VALUE          0
STORE_ID             0
RETAIL_DISC          0
TRANS_TIME           0
WEEK_NO              0
COUPON_DISC          0
COUPON_MATCH_DISC    0
dtype: int64
```

Data Transformation:

- Upon examining the data in various tables such as `CAMPAIGN_DESC_RAW`, `COUPON_REDEMPT_RAW`, and `TRANSACTION_RAW`, it is observed that they contain date-related information in the 'DAY' format. To facilitate easier and more comprehensible analysis, it is necessary to transform this information into actual 'DATE' format.
- The client has provided a reference date of **'2023-01-01'** as the start_date. Utilizing this reference date, we can convert all the relevant columns into an actual date format accordingly.
- To accomplish this task, we are importing two classes from the `datetime` module: `datetime`, which allows working with specific dates and times, and `timedelta`, which helps to calculate the difference or add/subtract time intervals.
- We then assign a variable **start_date** to the given reference date and convert the string '2023-01-01' into a datetime object using the pandas library.

In [43]:
```
from datetime import datetime,timedelta
```

In [44]:
```
start_date = pd.to_datetime('2023-01-01')
```

In [45]:
```
start_date
```

Out[45]:
```
Timestamp('2023-01-01 00:00:00')
```

- Firstly, we are transforming the date-related information from the `CAMPAIGN_DESC_RAW` table. This table contains two attributes, 'START_DAY' and 'END_DAY,' which represent date information. Using these attributes, we will create two new attributes, **'Start_Date'** and **'End_Date.**'

In [46]:
```
CAMPAIGN_DESC_RAW.head()
```

Out[46]:

	DESCRIPTION	CAMPAIGN	START_DAY	END_DAY
0	TypeB	24	659	719
1	TypeC	15	547	708
2	TypeB	25	659	691
3	TypeC	20	615	685
4	TypeB	23	646	684

In [47]:
```
CAMPAIGN_DESC_RAW['Start_Date']= start_date + pd.to_timedelta(CAMPAIGN_DESC_RAW[
```

In [48]:
```
CAMPAIGN_DESC_RAW['End_Date']= start_date + pd.to_timedelta(CAMPAIGN_DESC_RAW['E
```

- In the above code, the `pd.to_timedelta` function converts the 'DAY' column values from the CAMPAIGN_DESC_RAW DataFrame into timedelta objects. The `unit='D'` argument specifies that the values in the 'DAY' column represent the number of days.

In [49]:
```
CAMPAIGN_DESC_RAW.head()
```

Out[49]:

	DESCRIPTION	CAMPAIGN	START_DAY	END_DAY	Start_Date	End_Date
0	TypeB	24	659	719	2024-10-21	2024-12-20
1	TypeC	15	547	708	2024-07-01	2024-12-09
2	TypeB	25	659	691	2024-10-21	2024-11-22
3	TypeC	20	615	685	2024-09-07	2024-11-16
4	TypeB	23	646	684	2024-10-08	2024-11-15

- Here, we can also add one more new attribute, i.e., **'Campaign_Duration'**, which represents the total duration of the specific campaign.

In [50]:
```
CAMPAIGN_DESC_RAW['Campaign_Duration'] = CAMPAIGN_DESC_RAW['END_DAY'] - CAMPAIGN
```

In [51]:
```
CAMPAIGN_DESC_RAW.head()
```

Out[51]:

	DESCRIPTION	CAMPAIGN	START_DAY	END_DAY	Start_Date	End_Date	Campaign_I
0	TypeB	24	659	719	2024-10-21	2024-12-20	
1	TypeC	15	547	708	2024-07-01	2024-12-09	
2	TypeB	25	659	691	2024-10-21	2024-11-22	
3	TypeC	20	615	685	2024-09-07	2024-11-16	
4	TypeB	23	646	684	2024-10-08	2024-11-15	

- Also, we can extract the month and year parts from the date columns into separate columns.
- In the code below, the `dt` accessor is used to access the datetime properties of the 'Start_date' column. The methods `strftime('%m')` and `strftime('%Y')` convert the datetime objects in the column to their corresponding month and year in a two-digit string format, respectively.

In [52]:
```
CAMPAIGN_DESC_RAW['Start_Month'] = CAMPAIGN_DESC_RAW['Start_Date'].dt.strftime('
```

In [53]:
```
CAMPAIGN_DESC_RAW['End_Month'] = CAMPAIGN_DESC_RAW['End_Date'].dt.strftime('%m')
```

In [54]:
```
CAMPAIGN_DESC_RAW['Start_Year'] = CAMPAIGN_DESC_RAW['Start_Date'].dt.strftime('%
```

In [55]:
```
CAMPAIGN_DESC_RAW['End_Year'] = CAMPAIGN_DESC_RAW['End_Date'].dt.strftime('%Y')
```

In [56]:
```
CAMPAIGN_DESC_RAW.head()
```

Out[56]:

	DESCRIPTION	CAMPAIGN	START_DAY	END_DAY	Start_Date	End_Date	Campaign_I
0	TypeB	24	659	719	2024-10-21	2024-12-20	
1	TypeC	15	547	708	2024-07-01	2024-12-09	
2	TypeB	25	659	691	2024-10-21	2024-11-22	
3	TypeC	20	615	685	2024-09-07	2024-11-16	
4	TypeB	23	646	684	2024-10-08	2024-11-15	

- Now, we can drop the unneeded columns such as **START_DAY** and **END_DAY** from the table.
- In the code below, the `axis` parameter specifies whether to drop labels from the `rows (axis=0)` or `columns (axis=1)`. The `inplace` parameter, when set to True, means that the operation will be performed directly on the original DataFrame without returning a new DataFrame.

In [57]:
```
CAMPAIGN_DESC_RAW.drop(['START_DAY','END_DAY'], axis=1, inplace=True)
```

In [58]:
```
CAMPAIGN_DESC_RAW.head()
```

Out[58]:

	DESCRIPTION	CAMPAIGN	Start_Date	End_Date	Campaign_Duration	Start_Month
0	TypeB	24	2024-10-21	2024-12-20	60	10
1	TypeC	15	2024-07-01	2024-12-09	161	07
2	TypeB	25	2024-10-21	2024-11-22	32	10
3	TypeC	20	2024-09-07	2024-11-16	70	09
4	TypeB	23	2024-10-08	2024-11-15	38	10

- The table above has now been transformed into a cleaned file. Upon checking the data types of each column, the date columns are shown as `datetime64[ns]`.
- If we directly push this table back to Snowflake, this date type will throw an error. Therefore, we need to change it to a suitable format.

In [59]:
```
CAMPAIGN_DESC_RAW.dtypes
```

Out[59]:
```
DESCRIPTION                  object
CAMPAIGN                      int8
Start_Date           datetime64[ns]
End_Date             datetime64[ns]
Campaign_Duration             int16
Start_Month                  object
End_Month                    object
Start_Year                   object
End_Year                     object
dtype: object
```

- In order to achieve this, we use the code below. This code converts the `Start_date` and `End_Date` columns in the `CAMPAIGN_DESC_RAW` DataFrame to **date objects**, effectively removing the time component from each value in these columns.

- In the code, the `pd.to_datetime()` function from the Pandas library converts the values in the 'Start_date' and 'End_Date' columns to **datetime objects**, ensuring they are treated as dates and times. The `apply()` method is then used to apply a function along the axis of the DataFrame. In this case, a lambda function extracts just the date part from each datetime object (referred to as `x`) using the `date()` method. This essentially converts each datetime object to a **date object**, removing the time component.

In [60]:
```
CAMPAIGN_DESC_RAW['Start_Date'] = pd.to_datetime(CAMPAIGN_DESC_RAW['Start_Date']
```

In [61]:
```
CAMPAIGN_DESC_RAW['End_Date'] = pd.to_datetime(CAMPAIGN_DESC_RAW['End_Date']).ap
```

In [62]:
```
CAMPAIGN_DESC_RAW.dtypes
```

Out[62]:
```
DESCRIPTION          object
CAMPAIGN               int8
Start_Date           object
End_Date             object
Campaign_Duration     int16
Start_Month          object
End_Month            object
Start_Year           object
End_Year             object
dtype: object
```

In the same way, we are transforming date-related information from both the `COUPON_REDEMPT_RAW` and `TRANSACTION_RAW` tables.

In [63]:
```
COUPON_REDEMPT_RAW.head()
```

Out[63]:

	HOUSEHOLD_KEY	DAY	COUPON_UPC	CAMPAIGN
0	1	421	10000085364	8
1	1	421	51700010076	8
2	1	427	54200000033	8
3	1	597	10000085476	18
4	1	597	54200029176	18

In [64]:
```
COUPON_REDEMPT_RAW['Date']= start_date + pd.to_timedelta(COUPON_REDEMPT_RAW['DAY
```

In [65]:
```
COUPON_REDEMPT_RAW['Month'] = COUPON_REDEMPT_RAW['Date'].dt.strftime('%m')
```

In [66]:
```
COUPON_REDEMPT_RAW['Year'] = COUPON_REDEMPT_RAW['Date'].dt.strftime('%Y')
```

In [67]:
```
COUPON_REDEMPT_RAW.drop(['DAY'], axis=1, inplace=True)
```

In [68]:
```
COUPON_REDEMPT_RAW.head()
```

Out[68]:

	HOUSEHOLD_KEY	COUPON_UPC	CAMPAIGN	Date	Month	Year
0	1	10000085364	8	2024-02-26	02	2024
1	1	51700010076	8	2024-02-26	02	2024
2	1	54200000033	8	2024-03-03	03	2024
3	1	10000085476	18	2024-08-20	08	2024
4	1	54200029176	18	2024-08-20	08	2024

In [69]:
```
COUPON_REDEMPT_RAW.dtypes
```

Out[69]:
```
HOUSEHOLD_KEY             int16
COUPON_UPC                int64
CAMPAIGN                   int8
Date             datetime64[ns]
Month                    object
Year                     object
dtype: object
```

In [70]:
```
COUPON_REDEMPT_RAW['Date'] = pd.to_datetime(COUPON_REDEMPT_RAW['Date']).apply(la
```

In [71]:
```
COUPON_REDEMPT_RAW.dtypes
```

Out[71]:
```
HOUSEHOLD_KEY     int16
COUPON_UPC        int64
CAMPAIGN           int8
Date             object
Month            object
Year             object
dtype: object
```

In [72]:
```
TRANSACTION_RAW.head()
```

Out[72]:

	HOUSEHOLD_KEY	BASKET_ID	DAY	PRODUCT_ID	QUANTITY	SALES_VALUE	STOR
0	2056	29330027026	177	1106523	1	2.32	
1	2056	29330027026	177	5567325	1	2.69	
2	2056	29330027027	177	1106523	1	2.32	
3	1873	29330097522	177	1135681	1	1.29	
4	1602	29339301484	177	828427	1	0.22	

In [73]:
```
TRANSACTION_RAW['Date'] = start_date + pd.to_timedelta(TRANSACTION_RAW['DAY'], u
```

In [74]:
```
TRANSACTION_RAW['Month'] = TRANSACTION_RAW['Date'].dt.strftime('%m')
```

In [75]:
```
TRANSACTION_RAW['Year'] = TRANSACTION_RAW['Date'].dt.strftime('%Y')
```

In [76]:
```
TRANSACTION_RAW.drop(['DAY', 'WEEK_NO'], axis=1, inplace=True)
```

In [77]:
```
TRANSACTION_RAW.head()
```

Out[77]:

	HOUSEHOLD_KEY	BASKET_ID	PRODUCT_ID	QUANTITY	SALES_VALUE	STORE_ID
0	2056	29330027026	1106523	1	2.32	341
1	2056	29330027026	5567325	1	2.69	341
2	2056	29330027027	1106523	1	2.32	341
3	1873	29330097522	1135681	1	1.29	320
4	1602	29339301484	828427	1	0.22	334

In [78]:

```
# Converting TRANS_TIME to HH:MM format

TRANSACTION_RAW['HOURS'] = TRANSACTION_RAW['TRANS_TIME'] // 100  # Extract hours
TRANSACTION_RAW['MINUTES'] = TRANSACTION_RAW['TRANS_TIME'] % 100  # Extract minu

# Validating and formatting the TRANS_TIME column as TIME_FORMAT

TRANSACTION_RAW['TIME_FORMAT'] = TRANSACTION_RAW.apply(
    lambda row: f"{row['HOURS']:02d}:{row['MINUTES']:02d}"
    if (0 <= row['TRANS_TIME'] <= 2359 and row['MINUTES'] < 60)
    else "INVALID_TIME",
    axis = 1)
```

- This code creates a new column, **TIME_FORMAT**, in the DataFrame `TRANSACTION_RAW`. It uses the apply() function with a lambda function to format the **HOURS** and **MINUTES** columns into a time string (HH:MM) or marks the row as INVALID_TIME if the conditions are not met.
- The `apply()` function processes each row of the DataFrame. By using **axis = 1**, it ensures that the function applies row-wise instead of column-wise.
- The `lambda` function takes one row of the DataFrame at a time as input (referred to as row).
- `f"{row['HOURS']:02d}:{row['MINUTES']:02d}"`: In this formatted string,
  - `row['HOURS']:02d` converts the **HOURS** column value into a two-digit number, adding a leading zero if necessary.
  - Similarly, `row['MINUTES']:02d` ensures that MINUTES is displayed as two digits.
  - The two values are combined into a string with a colon `(:)` in between, creating the **HH:MM** format.
- `if (0 <= row['TRANS_TIME'] <= 2359 and row['MINUTES'] < 60)`: This condition checks if **TRANS_TIME** is within the valid range for a 24-hour clock (0 to 2359). The minutes part (**row['MINUTES']**) is less than 60 (to ensure valid time). If both conditions are true, the time is formatted as HH:MM.
- If the conditions fail (e.g., TRANS_TIME is out of range or MINUTES is 60 or more), the lambda function returns "INVALID_TIME" for that row.

In [79]:
```
TRANSACTION_RAW.head()
```

Out[79]:

	HOUSEHOLD_KEY	BASKET_ID	PRODUCT_ID	QUANTITY	SALES_VALUE	STORE_ID
0	2056	29330027026	1106523	1	2.32	341
1	2056	29330027026	5567325	1	2.69	341
2	2056	29330027027	1106523	1	2.32	341
3	1873	29330097522	1135681	1	1.29	320
4	1602	29339301484	828427	1	0.22	334

In [80]:
```
TRANSACTION_RAW.drop(['HOURS', 'MINUTES'], axis=1, inplace=True)
```

In [81]:
```
TRANSACTION_RAW.head()
```

Out[81]:

	HOUSEHOLD_KEY	BASKET_ID	PRODUCT_ID	QUANTITY	SALES_VALUE	STORE_ID
0	2056	29330027026	1106523	1	2.32	341
1	2056	29330027026	5567325	1	2.69	341
2	2056	29330027027	1106523	1	2.32	341
3	1873	29330097522	1135681	1	1.29	320
4	1602	29339301484	828427	1	0.22	334

In [82]:
```
TRANSACTION_RAW.dtypes
```

Out[82]:
```
HOUSEHOLD_KEY                int16
BASKET_ID                    int64
PRODUCT_ID                   int32
QUANTITY                     int32
SALES_VALUE                float64
STORE_ID                     int32
RETAIL_DISC                float64
TRANS_TIME                   int16
COUPON_DISC                   int8
COUPON_MATCH_DISC             int8
Date                 datetime64[ns]
Month                        object
Year                         object
TIME_FORMAT                  object
dtype: object
```

In [83]:
```
TRANSACTION_RAW['Date'] = pd.to_datetime(TRANSACTION_RAW['Date']).apply(lambda x
```

In [84]:
```
# Convert TIME_FORMAT column to valid time objects

TRANSACTION_RAW['TRANS_TIME_FMTD'] = TRANSACTION_RAW['TIME_FORMAT'].apply(
    lambda x: datetime.strptime(x, "%H:%M").time() if x != "INVALID_TIME" else N
```

In [85]:
```
TRANSACTION_RAW.head()
```

Out[85]:

	HOUSEHOLD_KEY	BASKET_ID	PRODUCT_ID	QUANTITY	SALES_VALUE	STORE_ID
0	2056	29330027026	1106523	1	2.32	341
1	2056	29330027026	5567325	1	2.69	341
2	2056	29330027027	1106523	1	2.32	341
3	1873	29330097522	1135681	1	1.29	320
4	1602	29339301484	828427	1	0.22	334

In [86]:
```
TRANSACTION_RAW.drop(['TRANS_TIME', 'TIME_FORMAT'], axis=1, inplace=True)
```

In [87]:
```
TRANSACTION_RAW.head()
```

Out[87]:

	HOUSEHOLD_KEY	BASKET_ID	PRODUCT_ID	QUANTITY	SALES_VALUE	STORE_ID
0	2056	29330027026	1106523	1	2.32	341
1	2056	29330027026	5567325	1	2.69	341
2	2056	29330027027	1106523	1	2.32	341
3	1873	29330097522	1135681	1	1.29	320
4	1602	29339301484	828427	1	0.22	334

In [88]:
```
TRANSACTION_RAW.dtypes
```

Out[88]:
```
HOUSEHOLD_KEY          int16
BASKET_ID              int64
PRODUCT_ID             int32
QUANTITY               int32
SALES_VALUE          float64
STORE_ID               int32
RETAIL_DISC          float64
COUPON_DISC             int8
COUPON_MATCH_DISC       int8
Date                  object
Month                 object
Year                  object
TRANS_TIME_FMTD       object
dtype: object
```

Loading the cleaned files back to the Snowflake database:

- This below code sets up a connection to the Snowflake instance using SQLAlchemy and the Snowflake connector. It then reads a DataFrame from a CSV file and writes it back to a specified table in Snowflake.

In [89]:
```
# Importing all the required Libraries

from sqlalchemy import create_engine # For creating a connection to the Snowflak
from sqlalchemy.engine import URL # For defining the URL of the Snowflake connec
import snowflake.connector as snowCtx # For connecting to Snowflake using Snowfl
from snowflake.connector.pandas_tools import write_pandas # For writing pandas D
import pandas as pd # For working with data in DataFrame format
import getpass # For securely handling password input
```

In [90]:
```
#Establishing connection to Snowflake

conn = snowflake.connector.connect(
        user = 'JSANTU',
        # password = getpass.getpass('Your Snowflake Password: '),
```

```
        password = '**********',
        account = 'KMMGNKB-XC40527',
        database='RETAILS',
        schema='PUBLIC',
        warehouse='COMPUTE_WH'
        )

print('Connection established successfully with the Snowflake')
```

```
Connection established successfully with the Snowflake
```

In [91]:
```
# Creating a cursor object
cur = conn.cursor()
```

- Executing SQL statements to create tables in Snowflake using a cursor to load cleaned data.

In [92]:
```
# Creating a new table in Snowflake named `CAMPAIGN_DESC_NEW`

cur.execute(''' CREATE OR REPLACE TABLE CAMPAIGN_DESC_NEW (
DESCRIPTION VARCHAR(10),
CAMPAIGN INT,
Start_Date DATE,
End_Date  DATE,
Campaign_Duration INT,
Start_Month INT,
End_Month INT,
Start_Year INT,
End_Year INT,
PRIMARY KEY (DESCRIPTION),
UNIQUE (CAMPAIGN)
) ''')
```

Out[92]:
```
<snowflake.connector.cursor.SnowflakeCursor at 0x13caa32d410>
```

In [93]:
```
# Creating a new table in Snowflake named `COUPON_REDEMPT_NEW`

cur.execute(''' CREATE OR REPLACE TABLE COUPON_REDEMPT_NEW (
HOUSEHOLD_KEY INT,
COUPON_UPC INT,
CAMPAIGN INT,
Date DATE,
Month INT,
Year INT,
FOREIGN KEY (HOUSEHOLD_KEY) REFERENCES DEMOGRAPHIC_RAW(household_key),
FOREIGN KEY (CAMPAIGN) REFERENCES CAMPAIGN_DESC_NEW(CAMPAIGN)
) ''')
```

Out[93]:
```
<snowflake.connector.cursor.SnowflakeCursor at 0x13caa32d410>
```

In [94]:
```
# Creating a new table in Snowflake named `TRANSACTION_NEW`

cur.execute('''CREATE OR REPLACE TABLE TRANSACTION_NEW (
HOUSEHOLD_KEY INT,
BASKET_ID INT,
PRODUCT_ID INT,
QUANTITY INT,
SALES_VALUE FLOAT,
```

```
STORE_ID INT,
RETAIL_DISC FLOAT,
COUPON_DISC INT,
COUPON_MATCH_DISC INT,
Date DATE,
Month INT,
Year INT,
TRANS_TIME_FMTD TIME,
FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_RAW(PRODUCT_ID),
FOREIGN KEY (HOUSEHOLD_KEY) REFERENCES DEMOGRAPHIC_RAW(household_key)
) ''')
```

Out[94]: `<snowflake.connector.cursor.SnowflakeCursor at 0x13caa32d410>`

Now, using the `write_pandas` function, we can write a pandas DataFrame to a table in Snowflake. The function includes the following parameters and output variables:

- Parameters:
  - `conn` : The Snowflake connection object.
  - `CAMPAIGN_DESC_RAW` : The pandas DataFrame containing the data to be written to Snowflake.
  - `'CAMPAIGN_DESC_NEW'` : The name of the target table in Snowflake where the data will be written.
  - `quote_identifiers = False` : This parameter specifies whether to quote the identifiers (e.g., table names). Setting it to False means identifiers will not be quoted.
- Output Variables:
  - `success` : A boolean value indicating whether the operation was successful.
  - `nchunks` : The number of chunks that were written to Snowflake (the data might be divided into chunks for writing).
  - `nrows` : The number of rows that were written to the Snowflake table.
  - `_` : An unused variable (often represented by an underscore) to ignore the fourth return value of the function.

In [95]:
```
success, nchunks, nrows, _ = write_pandas(conn, CAMPAIGN_DESC_RAW, 'CAMPAIGN_DES

print(str(success)+','+str(nchunks)+','+str(nrows))
```

```
True,1,30
```

In [96]:
```
success, nchunks, nrows, _ = write_pandas(conn, COUPON_REDEMPT_RAW, 'COUPON_REDE

print(str(success)+','+str(nchunks)+','+str(nrows))
```

```
True,1,2318
```

In [97]:
```
success, nchunks, nrows, _ = write_pandas(conn, TRANSACTION_RAW , 'TRANSACTION_N

print(str(success)+','+str(nchunks)+','+str(nrows))
```

```
True,1,2595732
```

In [98]:
```
# Closing 'cursor' & 'connection'
cur.close()
```

```
conn.close()
```

In [99]:
```
if cur.is_closed():
    print("The cursor is closed.")
else:
    print("The cursor is still open.")

if conn.is_closed():
    print("The connection is closed.")
else:
    print("The connection is still open.")
```

```
The cursor is closed.
The connection is closed.
```

Now, the cleaned data is successfully loaded back into Snowflake database.

EDA:

DEMOGRAPHIC_RAW

In [100…
```
DEMOGRAPHIC_RAW.head(10)
```

Out[100…

	AGE_DESC	MARITAL_STATUS_CODE	INCOME_DESC	HOMEOWNER_DESC	HH_COMP_I
0	65+	A	35-49K	Homeowner	2 Adults No
1	45-54	A	50-74K	Homeowner	2 Adults No
2	25-34	U	25-34K	Unknown	2 Adults
3	25-34	U	75-99K	Homeowner	2 Adults
4	45-54	B	50-74K	Homeowner	Single Fe
5	65+	B	Under 15K	Homeowner	2 Adults No
6	45-54	A	100-124K	Homeowner	2 Adults No
7	35-44	B	15-24K	Unknown	Single Fe
8	25-34	A	75-99K	Renter	2 Adults No
9	45-54	A	75-99K	Homeowner	2 Adults No

In [101…
```
DEMOGRAPHIC_RAW.shape
```

Out[101…
```
(2500, 8)
```

In [102…
```
DEMOGRAPHIC_RAW.dtypes
```

Out[102…

```
AGE_DESC               object
MARITAL_STATUS_CODE    object
INCOME_DESC            object
HOMEOWNER_DESC         object
HH_COMP_DESC           object
HOUSEHOLD_SIZE_DESC    object
KID_CATEGORY_DESC      object
HOUSEHOLD_KEY           int16
dtype: object
```

In [103…

```
DEMOGRAPHIC_RAW['AGE_DESC'].value_counts()
```

Out[103…

```
AGE_DESC
45-54    901
35-44    594
25-34    445
65+      230
55-64    187
19-24    143
Name: count, dtype: int64
```

In [104…

```
DEMOGRAPHIC_RAW['MARITAL_STATUS_CODE'].value_counts()
```

Out[104…

```
MARITAL_STATUS_CODE
U    1087
A    1052
B     361
Name: count, dtype: int64
```

In [105…

```
DEMOGRAPHIC_RAW['INCOME_DESC'].value_counts()
```

Out[105…

```
INCOME_DESC
50-74K       588
35-49K       540
75-99K       305
15-24K       238
25-34K       237
Under 15K    193
125-149K     118
100-124K     109
150-174K      96
175-199K      33
250K+         29
200-249K      14
Name: count, dtype: int64
```

In [106…

```
DEMOGRAPHIC_RAW['HOMEOWNER_DESC'].value_counts()
```

Out[106…

```
HOMEOWNER_DESC
Homeowner          1573
Unknown             728
Renter              132
Probable Renter      34
Probable Owner       33
Name: count, dtype: int64
```

In [107…

```
DEMOGRAPHIC_RAW['HH_COMP_DESC'].value_counts()
```

Out[107…

```
HH_COMP_DESC
2 Adults No Kids    800
2 Adults Kids       584
Single Female       446
Single Male         292
Unknown             229
1 Adult Kids        149
Name: count, dtype: int64
```

In [108…

```
DEMOGRAPHIC_RAW['HOUSEHOLD_SIZE_DESC'].value_counts()
```

Out[108…

```
HOUSEHOLD_SIZE_DESC
2     1004
1      783
3      338
5+     206
4      169
Name: count, dtype: int64
```

In [109…

```
DEMOGRAPHIC_RAW['KID_CATEGORY_DESC'].value_counts()
```

Out[109…

```
KID_CATEGORY_DESC
None/Unknown    1737
1                357
3+               216
2                190
Name: count, dtype: int64
```

In [110…

```
DEMOGRAPHIC_RAW.groupby('AGE_DESC')['MARITAL_STATUS_CODE'].value_counts().sort_v
```

Out[110…

```
AGE_DESC  MARITAL_STATUS_CODE
45-54     U                     470
          A                     348
35-44     A                     290
          U                     210
25-34     U                     203
          A                     159
55-64     A                     113
65+       A                     111
35-44     B                      94
45-54     B                      83
25-34     B                      83
19-24     U                      81
65+       U                      72
55-64     U                      51
65+       B                      47
19-24     B                      31
          A                      31
55-64     B                      23
Name: count, dtype: int64
```

In [111…

```
pd.crosstab([DEMOGRAPHIC_RAW.HH_COMP_DESC, DEMOGRAPHIC_RAW.HOUSEHOLD_SIZE_DESC,
            [DEMOGRAPHIC_RAW.HOMEOWNER_DESC], margins=True)
```

Out[111…

HH_COMP_DESC	HOUSEHOLD_SIZE_DESC	KID_CATEGORY_DESC	HOMEOWNER_DESC Homeowner	Probable Owner
1 Adult Kids	2	1	6	0
	3	1	14	3
		2	13	0
	4	2	6	0
		3+	7	0
	5+	3+	30	0
2 Adults Kids	3	1	194	3
	4	2	112	3
	5+	3+	141	0
2 Adults No Kids	2	None/Unknown	631	12
Single Female	1	None/Unknown	113	9
	2	None/Unknown	73	3
Single Male	1	None/Unknown	89	0
	2	None/Unknown	26	0
Unknown	1	None/Unknown	94	0
	2	1	0	0
		None/Unknown	7	0
	3	1	13	0
	5+	3+	4	0
All			1573	33

In [112…

```
PRODUCT_RAW.head()
```

Out[112…

	PRODUCT_ID	MANUFACTURER	DEPARTMENT	BRAND	COMMODITY_DESC	SUB_CC
0	25671	2	GROCERY	National	FRZN ICE	ICE -
1	26081	2	MISC. TRANS.	National	NO COMMODITY DESCRIPTION	NO
2	26093	69	PASTRY	Private	BREAD	BREAD
3	26190	69	GROCERY	Private	FRUIT - SHELF STABLE	
4	26355	69	GROCERY	Private	COOKIES/CONES	SF

In [113…

```
PRODUCT_RAW['DEPARTMENT'].unique()
```

Out[113…

```
array(['GROCERY', 'MISC. TRANS.', 'PASTRY', 'DRUG GM', 'MEAT-PCKGD',
       'SEAFOOD-PCKGD', 'PRODUCE', 'NUTRITION', 'DELI', 'COSMETICS',
       'MEAT', 'FLORAL', 'TRAVEL & LEISUR', 'SEAFOOD', 'MISC SALES TRAN',
       'SALAD BAR', 'KIOSK-GAS', 'ELECT &PLUMBING', 'GRO BAKERY',
       'GM MERCH EXP', 'FROZEN GROCERY', 'COUP/STR & MFG', 'SPIRITS',
       'GARDEN CENTER', 'TOYS', 'CHARITABLE CONT', 'RESTAURANT', 'RX',
       'PROD-WHS SALES', 'MEAT-WHSE', 'DAIRY DELI', 'CHEF SHOPPE', 'HBC',
       'DELI/SNACK BAR', 'PORK', 'AUTOMOTIVE', 'VIDEO RENTAL', ' ',
       'CNTRL/STORE SUP', 'HOUSEWARES', 'POSTAL CENTER', 'PHOTO', 'VIDEO',
       'PHARMACY SUPPLY'], dtype=object)
```

In [114…

```
PRODUCT_RAW['COMMODITY_DESC'].unique()
```

Out[114…

```
array(['FRZN ICE', 'NO COMMODITY DESCRIPTION', 'BREAD',
       'FRUIT - SHELF STABLE', 'COOKIES/CONES', 'SPICES & EXTRACTS',
       'VITAMINS', 'BREAKFAST SWEETS', 'PNT BTR/JELLY/JAMS',
       'ICE CREAM/MILK/SHERBTS', 'MAGAZINE', 'AIR CARE', 'CHEESE',
       'SHORTENING/OIL', 'COFFEE', 'DIETARY AID PRODUCTS',
       'PAPER HOUSEWARES', 'BAKED BREAD/BUNS/ROLLS',
       'VEGETABLES - SHELF STABLE', 'HISPANIC', 'DINNER MXS:DRY',
       'CONDIMENTS/SAUCES', 'FRZN VEGETABLE/VEG DSH', 'BAKING NEEDS',
       'DINNER SAUSAGE', 'FRZN FRUITS', 'SEAFOOD - FROZEN',
       'HOUSEHOLD CLEANG NEEDS', 'FD WRAPS/BAGS/TRSH BG',
       'DRY MIX DESSERTS', 'PICKLE/RELISH/PKLD VEG', 'CAKES',
       'BAKING MIXES', 'POTATOES', 'FLUID MILK PRODUCTS', 'SOUP',
       'BAKED SWEET GOODS', 'COOKIES', 'DRY BN/VEG/POTATO/RICE',
       'FACIAL TISS/DNR NAPKIN', 'FROZEN PIZZA', 'EGGS',
       'REFRGRATD DOUGH PRODUCTS', 'HOT CEREAL', 'COLD CEREAL',
       'SUGARS/SWEETNERS', 'SEAFOOD - SHELF STABLE', 'POPCORN',
       'CANNED JUICES', 'STATIONERY & SCHOOL SUPPLIES', 'COLD AND FLU',
       'BABY HBC', 'BAG SNACKS', 'BEANS - CANNED GLASS & MW',
       'FROZEN MEAT', 'SOAP - LIQUID & BAR', 'ROLLS',
       'NON-DAIRY BEVERAGES', 'KITCHEN GADGETS', 'CRACKERS/MISC BKD FD',
       'CONVENIENT BRKFST/WHLSM SNACKS', 'CHIPS&SNACKS',
       'CANDY - PACKAGED', 'SOFT DRINKS', 'HAIR CARE PRODUCTS',
       'CANDY - CHECKLANE', 'BUTTER', 'FRZN MEAT/MEAT DINNERS',
       'SHAVING CARE PRODUCTS', 'WATER - CARBONATED/FLVRD DRINK',
       'FRZN BREAKFAST FOODS', 'MILK BY-PRODUCTS', 'FIRST AID PRODUCTS',
       'NEWSPAPER', 'SALADS/DIPS', 'INSECTICIDES',
       'ORGANICS FRUIT & VEGETABLES', 'ELECTRICAL SUPPPLIES',
       'LAUNDRY DETERGENTS', 'JUICE', 'ISOTONIC DRINKS',
       'FRZN JCE CONC/DRNKS', 'LAUNDRY ADDITIVES',
       'IRONING AND CHEMICALS', 'TEAS', 'DRY NOODLES/PASTA',
       'PASTA SAUCE', 'PROCESSED', 'MAKEUP AND TREATMENT', 'ANALGESICS',
       'HOSIERY/SOCKS', 'CAT FOOD', 'BATTERIES',
       'MOLASSES/SYRUP/PANCAKE MIXS', 'BOOKSTORE', 'BATH TISSUES',
       'FROZEN PIE/DESSERTS', 'MEAT - SHELF STABLE', 'MEAT - MISC',
       'SPRING/SUMMER SEASONAL', 'LUNCHMEAT',
       'BREAKFAST SAUSAGE/SANDWICHES', 'DRIED FRUIT',
       'CHARCOAL AND LIGHTER FLUID', 'SALD DRSNG/SNDWCH SPRD', 'LIQUOR',
       'FROZEN BREAD/DOUGH', 'HAND/BODY/FACIAL PRODUCTS', 'SNACK NUTS',
       'ORAL HYGIENE PRODUCTS', 'BEERS/ALES', 'INFANT FORMULA',
       'REFRGRATD JUICES/DRNKS', 'YOGURT', 'DEODORANTS', 'BACON',
       'DOG FOODS', 'FRZN NOVELTIES/WTR ICE', 'FEMININE HYGIENE',
       'COFFEE FILTERS', 'BROOMS AND MOPS',
       'GREETING CARDS/WRAP/PARTY SPLY', 'WAREHOUSE SNACKS',
       'DRY SAUCES/GRAVY', 'MARGARINES', 'PWDR/CRYSTL DRNK MX', 'OLIVES',
       'MISC. DAIRY', 'COCOA MIXES', 'SALAD MIX', 'HARDWARE SUPPLIES',
       'HOT DOGS', 'FLOUR & MEALS', 'SYRUPS/TOPPINGS', 'ANTACIDS',
       'BLEACH', 'PAPER TOWELS', 'STONE FRUIT', 'MELONS', 'BERRIES',
       'CHICKEN', 'SANDWICHES', 'GRAPES', 'CARROTS', 'APPLES', 'HERBS',
       'CIGARETTES', 'HEAT/SERVE', 'BABY FOODS', 'SQUASH',
       'VEGETABLES - ALL OTHERS', 'DELI MEATS', 'BEEF', 'FRZN POTATOES',
       'CORN', 'CHEESES', 'MISCELLANEOUS', 'CITRUS', 'FROZEN',
       'DISHWASH DETERGENTS', 'SWEET GOODS & SNACKS', 'NUTS',
       'TROPICAL FRUIT', 'CHICKEN/POULTRY', 'BROCCOLI/CAULIFLOWER',
       'SINUS AND ALLERGY', 'MUSHROOMS', 'CANNED MILK', 'ONIONS',
       'INFANT CARE PRODUCTS', 'TURKEY', 'WATER', 'CAT LITTER',
       'SEAFOOD-FRESH', 'PIES', 'TOBACCO OTHER', 'SEAFOOD - MISC',
       'FILM AND CAMERA PRODUCTS', 'PEPPERS-ALL', 'PLASTIC HOUSEWARES',
       'FOOT CARE PRODUCTS', 'SHOE CARE', 'FIREWORKS',
       'FLORAL-FLOWERING PLANTS', 'SUNTAN', 'CEREAL/BREAKFAST',
       'CANDLES/ACCESSORIES', 'COOKWARE & BAKEWARE',
```

```
       'DISPOSIBLE FOILWARE', 'FLORAL-FRESH CUT', 'VEGETABLES SALAD',
       'AUDIO/VIDEO PRODUCTS', 'REFRIGERATED', 'FROZEN CHICKEN',
       'SMOKED MEATS', 'VALUE ADDED VEGETABLES',
       'EYE AND EAR CARE PRODUCTS', 'SNKS/CKYS/CRKR/CNDY',
       'FLORAL BALLOONS', 'AUTOMOTIVE PRODUCTS', 'LAXATIVES',
       'DOMESTIC WINE', 'MISC WINE', 'OVERNIGHT PHOTOFINISHING',
       'TOMATOES', 'PEARS', 'FITNESS&DIET', 'BATH', 'COUPON/MISC ITEMS',
       'PORK', 'PERSONAL CARE APPLIANCES', 'CONDIMENTS', 'BEVERAGE',
       'SNACKS', 'VALUE ADDED FRUIT', 'SALAD BAR',
       'HAIR CARE ACCESSORIES', 'DIAPERS & DISPOSABLES',
       'SMOKING CESSATIONS', 'FUEL', 'PREPARED FOOD', 'PET CARE SUPPLIES',
       'COFFEE SHOP', 'COUPON', '(CORP USE ONLY)', 'ROSES',
       'PROD SUPPLIES', 'J-HOOKS', 'RICE CAKES', 'LAMB', 'IMPORTED WINE',
       'UNKNOWN', 'DELI SPECIALTIES (RETAIL PK)', 'PREPARED/PKGD FOODS',
       'FAMILY PLANNING', 'EASTER', 'CIGARS', 'PACKAGED NATURAL SNACKS',
       'FLORAL-FOLIAGE PLANTS', 'MISCELLANEOUS CROUTONS',
       'COUPONS/STORE & MFG', 'APPAREL', 'PREPAID WIRELESS&ACCESSORIES',
       'HOME FREEZING & CANNING SUPPLY', 'IN-STORE PHOTOFINISHING',
       'ADULT INCONTINENCE', 'PARTY TRAYS', 'DOMESTIC GOODS',
       'FALL AND WINTER SEASONAL', 'TOYS AND GAMES',
       'LAWN AND GARDEN SHOP', 'GLASSWARE & DINNERWARE', 'BAKING',
       'RESTRICTED DIET', 'FRAGRANCES', 'HALLOWEEN', 'BAKERY PARTY TRAYS',
       'SEWING', 'NATURAL HBC', 'BIRD SEED', 'GARDEN CENTER',
       'COSMETIC ACCESSORIES', 'ETHNIC PERSONAL CARE',
       'GLASSES/VISION AIDS', 'FLORAL- HARD GOODS', 'BABYFOOD', 'TOYS',
       'CHRISTMAS  SEASONAL', 'COFFEE SHOP SWEET GOODS&RETAIL',
       'DRY TEA/COFFEE/COCO MIX', 'VEAL', 'CONTINUITIES',
       'PORTABLE ELECTRIC APPLIANCES', 'VALENTINE',
       'FROZEN - BOXED(GROCERY)', 'SERVICE BEVERAGE',
       'GIFT & FRUIT BASKETS', 'SUSHI', 'PROPANE', 'FRZN SEAFOOD',
       'HOME FURNISHINGS', 'PHARMACY', 'NATURAL VITAMINS',
       'DELI SUPPLIES', 'WATCHES/CALCULATORS/LOBBY', 'SEASONAL',
       'BOUQUET (NON ROSE)', 'HOME HEALTH CARE',
       'RW FRESH PROCESSED MEAT', 'EXOTIC GAME/FOWL',
       'NDAIRY/TEAS/JUICE/SOD', 'TICKETS', 'MEAT SUPPLIES', 'NEW AGE',
       'FLORAL-ACCESSORIES', 'EASTER LILY', 'MISCELLANEOUS HBC',
       'BULK FOODS', 'NON EDIBLE PRODUCTS', 'QUICK SERVICE',
       'BOTTLE DEPOSITS', ' ', 'DOLLAR VALUE PRODUCTS',
       'MISCELLANEOUS(CORP USE ONLY)', 'SPORTS MEMORABLILIA',
       'LONG DISTANCE CALLING CARDS', 'PKG.SEAFOOD MISC',
       'FROZEN PACKAGE MEAT'], dtype=object)
```

In [115…

```
PRODUCT_RAW['SUB_COMMODITY_DESC'].unique()
```

Out[115…

```
array(['ICE - CRUSHED/CUBED', 'NO SUBCOMMODITY DESCRIPTION',
       'BREAD:ITALIAN/FRENCH', ..., 'SEMI MOIST CAT FOOD: BOX',
       'BERRIES OTHER', 'ROSES OTHER'], dtype=object)
```

In [116…

```
pd.crosstab([PRODUCT_RAW.DEPARTMENT, PRODUCT_RAW.COMMODITY_DESC] ,
            [PRODUCT_RAW.SUB_COMMODITY_DESC], margins=True)
```

Out[116…

	SUB_COMMODITY_DESC	*ATH ACCES:TOWEL BARS/SOAP D	*ATTERIES:CAMERA/FLASH/W
DEPARTMENT	**COMMODITY_DESC**		
		15	0
AUTOMOTIVE	**COUPON**	0	0
	NEWSPAPER	0	0
CHARITABLE CONT	**UNKNOWN**	0	0
CHEF SHOPPE	**UNKNOWN**	0	0
...	**...**	...	...
TRAVEL & LEISUR	**MELONS**	0	0
	TROPICAL FRUIT	0	0
VIDEO	**BEERS/ALES**	0	0
VIDEO RENTAL	**UNKNOWN**	0	0
All		15	1

361 rows × 2384 columns

In [117…

```
PRODUCT_RAW.groupby('DEPARTMENT')['COMMODITY_DESC'].value_counts().sort_values(a
```

Out[117…

```
DEPARTMENT  COMMODITY_DESC
DRUG GM     GREETING CARDS/WRAP/PARTY SPLY    2785
            CANDY - PACKAGED                  2473
COSMETICS   MAKEUP AND TREATMENT              2467
DRUG GM     HAIR CARE PRODUCTS                1744
GROCERY     SOFT DRINKS                       1704
                                              ...
HBC         INFANT CARE PRODUCTS                 1
HOUSEWARES  UNKNOWN                              1
PRODUCE     MISCELLANEOUS(CORP USE ONLY)         1
MEAT-PCKGD  FROZEN PACKAGE MEAT                  1
MEAT        LUNCHMEAT                            1
Name: count, Length: 360, dtype: int64
```

In [ ]: