Capstone Project 2 – Product Demand Prediction

Introduction

Accurate demand planning is one of the biggest competitive advantages a company can have in today's fast-paced global economy. Each industry and niche, however, has its own unique demand patterns. How does a company decide which statistical forecasting model to use with which demand pattern?

A 2014 study by Deloitte Consulting surveyed over 400 manufacturing and retail executives throughout the world in regard to high supply chain performance. The survey compared companies on two metrics: 1) inventory turnover and 2) the percentage of on-time and full deliveries. The study found that "supply chain leaders" had a higher percentage of both revenue growth and EBIT (earnings before interest and taxes) margin compared than "supply chain followers". Proper supply chain management can lead not only to improved profitability but also higher market share. It could also be the difference between overall business success and failure.

One of the biggest factors in a high-performing supply chain process is analyzing and planning for your future business needs. A common planning method used by businesses that engage in supply chain management is demand planning. In fact, businesses using big data analytics in demand planning experienced a 425% improvement in order-to-cycle delivery times and more than six times improvement in supply chain efficiency of 10 percent or higher. An Accenture study revealed that businesses that used big data analytics in demand planning experienced a 425% improvement in order-to-cycle delivery times and more than six times improvement in supply chain efficiency of 10 percent or higher.

The Problem

The company has not done any demand planning. Products are ordered based on periodic inventory and sales reports, or when suppliers have specials. Although individual markups are done initially, these are not pro-actively nor continually measured and evaluated. The order "system" is to more or less let store managers place ad-hoc orders for products they feel are running low. If the products are in the company warehouse, the orders get shipped. If not, the order more from the supplier.

When sales are strong, cashflow and profitability seem fine. Lately, company leaders have just noticed a drop in profitability and are wondering if demand planning could be utilized to create a reliable forecast for the business.

This project will analyze current and projected demand for 30 of the company's products in its 76 retail locations as the first step in implementing an overall supply chain management program. Specifically, the objective of this project will be to solve the problem of over-stocking and under-stocking of products. This will be completed as follows:

- Exploratory Data Analysis - Importing and exploring historical data
- Statistical data analysis
- Data Pre-Processing & Baseline Model
- Optimize Machine Learning Prediction Model

Once complete, company leadership will be able to examine company operations and vendor relationships to formalize an official company program.

Exploratory Data Analysis (EDA)

The data used in this project was acquired from the company. They provided three datasets:

1) sales.csv – contains 232,287 sales records with the UPC code of the product sold, the sales date, the store ID where the product was sold, the sales price, the base price, whether the product was on promotion for the week of the sale (1 or 0), whether the product was in the in-store circular (1 or 0) and the number of units sold for each week;
2) product_data.csv – contains the product description, manufacturer, product category and sub-category, product size and UPC for 30 products; and
3) store_data.csv – contains the store ID, store name, city, state, MSA code, market segment type of store, number of store parking spaces, store sales area square footage and average weekly baskets, for each of the 76 store locations.

We will first explore each dataset separately.

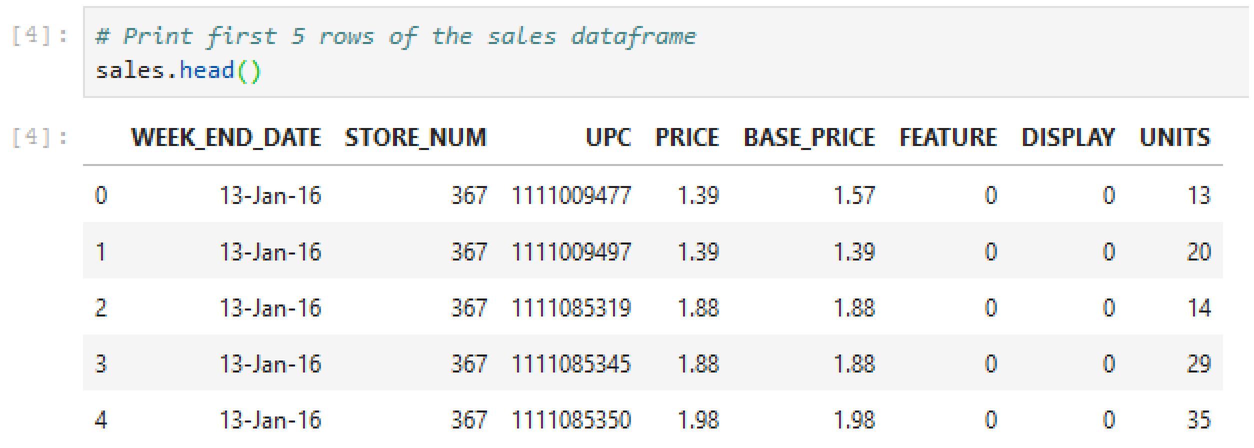
Sales Dataframe EDA

The following is a summary of the columns of the sales.csv dataset, which as mentioned earlier contains 232,287 rows of data.

Sales Data: ('sales.csv')

- **WEEK_END_DATE** - week ending date of sales report
- **STORE_NUM** - store number where sale was made
- **UPC** - (Universal Product Code) product specific identifier
- **BASE_PRICE** - base price of item
- **DISPLAY** - whether product was a part of in-store promotional display (1-Yes, 0-No)
- **FEATURE** - whether product was in in-store circular (1-Yes, 0-No)
- **UNITS** - units sold (target)

The first step in understanding the data is to take a quick look at the structure and data types.



```
[4]: # Print first 5 rows of the sales dataframe
     sales.head()
```

[4]:

	WEEK_END_DATE	STORE_NUM	UPC	PRICE	BASE_PRICE	FEATURE	DISPLAY	UNITS
0	13-Jan-16	367	1111009477	1.39	1.57	0	0	13
1	13-Jan-16	367	1111009497	1.39	1.39	0	0	20
2	13-Jan-16	367	1111085319	1.88	1.88	0	0	14
3	13-Jan-16	367	1111085345	1.88	1.88	0	0	29
4	13-Jan-16	367	1111085350	1.98	1.98	0	0	35

```
[5]: # Check datatypes of columns in sales dataframe
     sales.dtypes
```

```
[5]: WEEK_END_DATE     object
     STORE_NUM          int64
     UPC                int64
     PRICE            float64
     BASE_PRICE       float64
     FEATURE            int64
     DISPLAY            int64
     UNITS              int64
     dtype: object
```

Two issues that stand out at first are:

- WEEK_END_DATE has been imported as an object, but it is a datetime variable. This needs to be converted.
- The store number and product codes have been imported as integers, but these are categorical variables. This needs to be fixed as well.

Other issues/questions are:

- Datetime variable
    - What are the start and end dates?
    - Are these periodic intervals and are they regular?
    - Are there any missing data points?
- Numerical Variables ('PRICE', 'BASE_PRICE', and 'UNITS')
    - Need to check the distribution of numerical variables.
    - Also need to check if there are any outliers or missing values.
- Categorical Variables ('FEATURE' and 'DISPLAY')
    - Check the unique values for categorical variables
    - Are there any missing values?
    - Are there any variables with high cardinality / sparsity?

WEEK_END_DATE

- This variable was converted to a date time object.
- The sales data is for 142 weeks, based on the number of unique WEEK_END_DATE's in the sales file, starting on January 13, 2016 and ending September 26, 2018.
- No dates are missing from this period.
- All of the dates fall on Wednesday, which appears to be the date that the sales reports are generated.

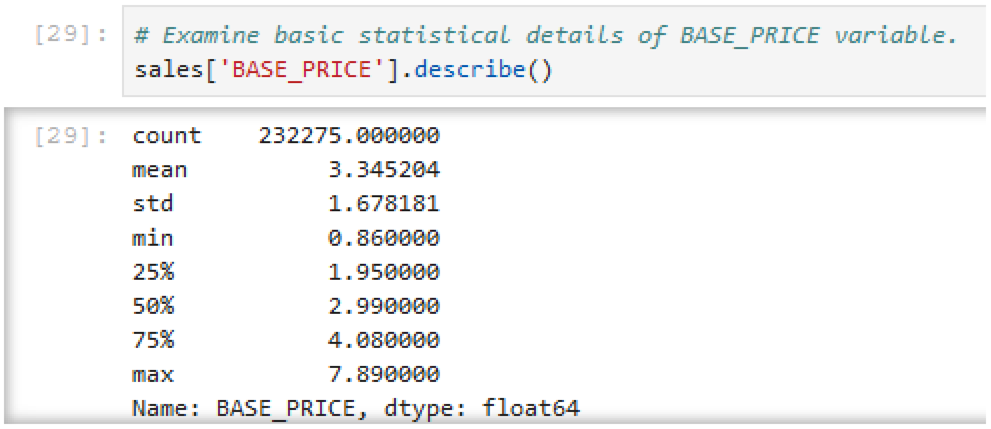
STORE_NUM and UPC

- There are no missing values in either variable.
- All 76 stores have reported sales transactions, although not the same number.

- The number of transactions reported by each store range from a low of 1,676 up to a high of 4,098.
- All stores reported selling at least one product each week (142 weeks x 76 stores = 10,792 records, which is the number of unique records in the data.
- There were 30 unique UPC codes found in the data which means that each product was sold, with the minimum number sold of 975 and a maximum of 10,790.
- With 30 products, 76 stores and 142 weeks, if every product was sold at every store at least once every week, there would be 323,760 rows of data. Since we only have 232,287 rows, not every product was sold at every store every week. This did occur 71.7% of the time.
- The average number of unique products sold each week is 22.

BASE_PRICE

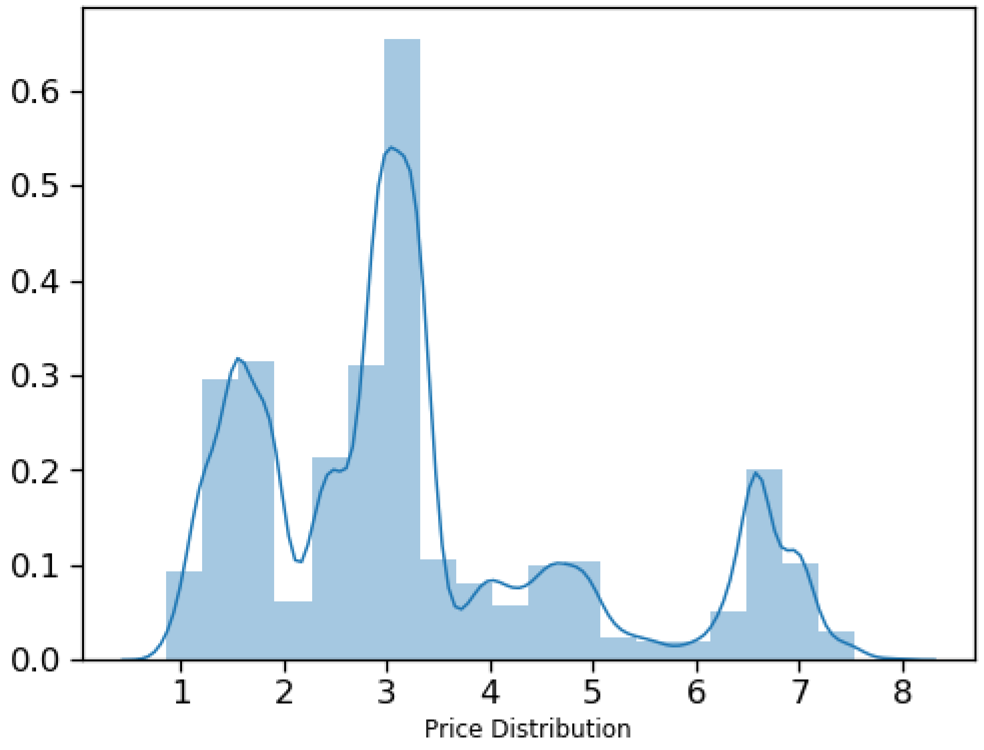
- There are no missing values.
- The basic statistics and distribution of the variable are as follows:



```
[29]: # Examine basic statistical details of BASE_PRICE variable.
      sales['BASE_PRICE'].describe()
```

```
[29]: count    232275.000000
      mean          3.345204
      std           1.678181
      min           0.860000
      25%           1.950000
      50%           2.990000
      75%           4.080000
      max           7.890000
      Name: BASE_PRICE, dtype: float64
```

```
[30]: # distribution of Base Price variable
      plt.figure(figsize=(8,6))
      sns.distplot((sales['BASE_PRICE'].values), bins=20, kde=True)
      plt.xlabel('Price Distribution', fontsize=12)
      plt.show()
```



- There are no extreme values in the BASE_PRICE variable.
- The range for base price is 0.86 to 7.89, with an average of 3.35.

FEATURE and DISPLAY

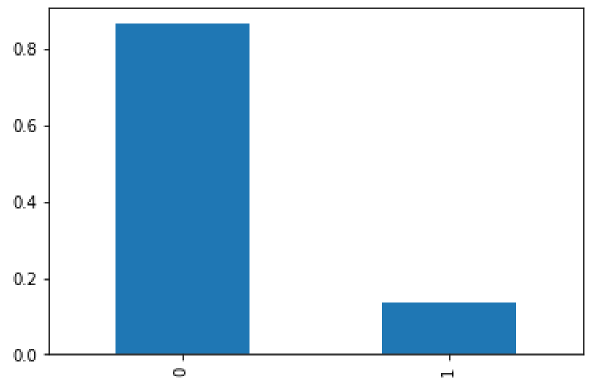
- There are no missing values in either variable.
- Both variables were imported as integer values, with both having either a ‘1’ for ‘Yes’ or ‘0’ for ‘No’.
- The value counts for each variable are shown here:

```
# Examine values for 'FEATURE'.
sales['DISPLAY'].value_counts(normalize=True)
```

```
0    0.864998
1    0.135002
Name: DISPLAY, dtype: float64
```

```
sales['DISPLAY'].value_counts(normalize=True).plot('bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2e2b402f688>
```



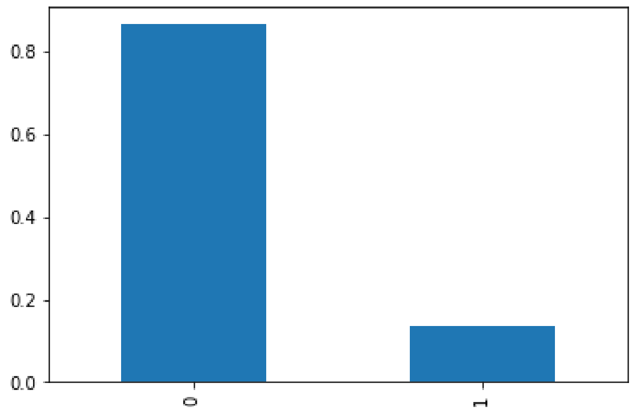
- Approximately 13.5 percent of products are on display

```
# Examine values for 'DISPLAY'.
sales['DISPLAY'].value_counts(normalize=True)
```

```
0    0.864998
1    0.135002
Name: DISPLAY, dtype: float64
```

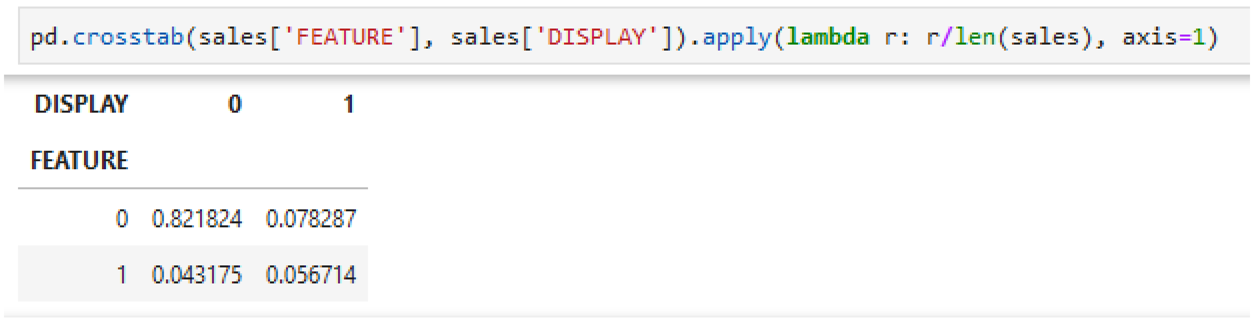
```
sales['DISPLAY'].value_counts(normalize=True).plot('bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2e2b3e4e9c8>
```



- Approximately 13.5 percent of products are on display

- The cross-tab table for FEATURE and DISPLAY is:

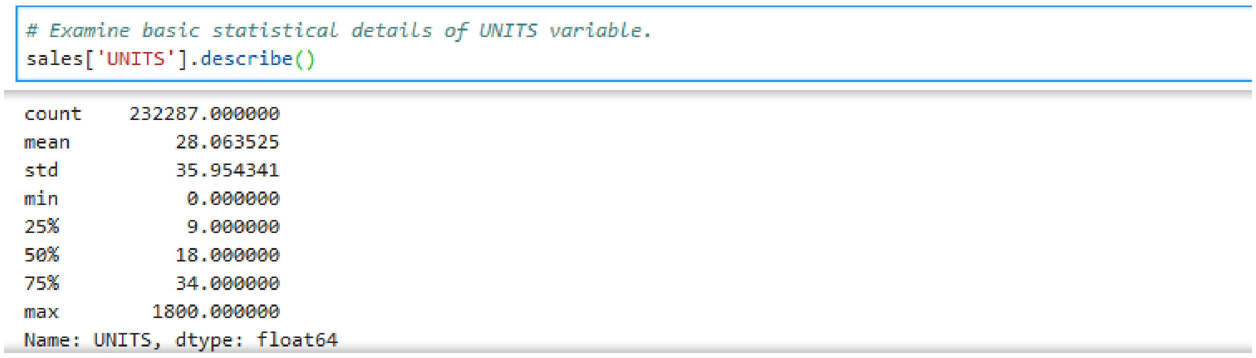


```
pd.crosstab(sales['FEATURE'], sales['DISPLAY']).apply(lambda r: r/len(sales), axis=1)
```

DISPLAY	0	1
FEATURE		
0	0.821824	0.078287
1	0.043175	0.056714

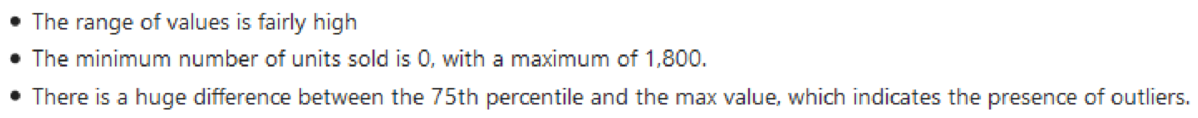
UNITS

- The basic statistical details for the UNITS variable are:



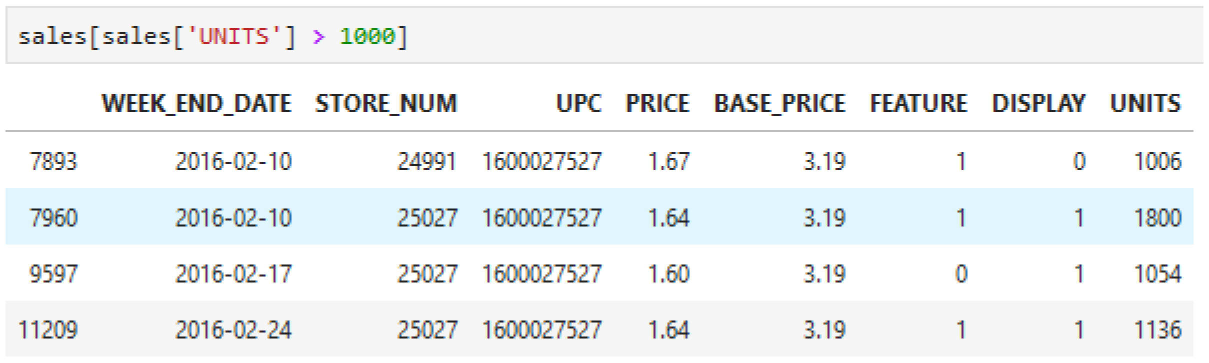
```
# Examine basic statistical details of UNITS variable.
sales['UNITS'].describe()
```

```
count    232287.000000
mean         28.063525
std          35.954341
min           0.000000
25%           9.000000
50%          18.000000
75%          34.000000
max        1800.000000
Name: UNITS, dtype: float64
```



- The range of values is fairly high
- The minimum number of units sold is 0, with a maximum of 1,800.
- There is a huge difference between the 75th percentile and the max value, which indicates the presence of outliers.

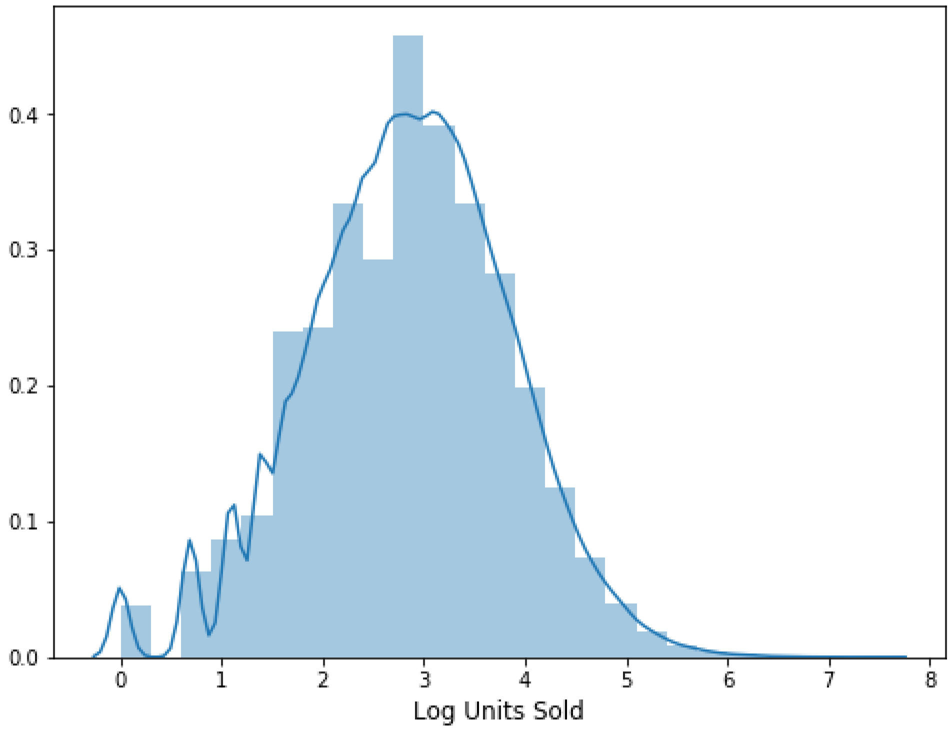
- There was only one row with 0 units sold; this row was dropped
- There were four rows of data with more than 1,000 units sold. To reduce the effect of outliers and for better visualization of the distribution, we plotted the log transformation for UNITS:



```
sales[sales['UNITS'] > 1000]
```

	WEEK_END_DATE	STORE_NUM	UPC	PRICE	BASE_PRICE	FEATURE	DISPLAY	UNITS
7893	2016-02-10	24991	1600027527	1.67	3.19	1	0	1006
7960	2016-02-10	25027	1600027527	1.64	3.19	1	1	1800
9597	2016-02-17	25027	1600027527	1.60	3.19	0	1	1054
11209	2016-02-24	25027	1600027527	1.64	3.19	1	1	1136

```
# log transformed UNITS column
plt.figure(figsize=(8,6))
sns.distplot(np.log(sales['UNITS'].values), bins=25, kde=True)
plt.xlabel('Log Units Sold', fontsize=12)
plt.show()
```



- After log transformation, the distribution looks closer to a normal distribution

Products Dataframe EDA

The following is a summary of the columns of the product_data.csv dataset, which as mentioned earlier contains 30 rows of data.

Product Data: ('product_data.csv')

- **UPC** - (Universal Product Code) product specific identifier
- **DESCRIPTION** - product description
- **MANUFACTURER** - product manufacturer/supplier
- **CATEGORY** - product category
- **SUB_CATEGORY** - product sub-category
- **PRODUCT_SIZE** - package size/quantity

```
# Print first five rows of product data
products.head()
```

	UPC	DESCRIPTION	MANUFACTURER	CATEGORY	SUB_CATEGORY	PRODUCT_SIZE
0	1111009477	PL MINI TWIST PRETZELS	PRIVATE LABEL	BAG SNACKS	PRETZELS	15 OZ
1	1111009497	PL PRETZEL STICKS	PRIVATE LABEL	BAG SNACKS	PRETZELS	15 OZ
2	1111009507	PL TWIST PRETZELS	PRIVATE LABEL	BAG SNACKS	PRETZELS	15 OZ
3	1111038078	PL BL MINT ANTSPTC RINSE	PRIVATE LABEL	ORAL HYGIENE PRODUCTS	MOUTHWASHES (ANTISEPTIC)	500 ML
4	1111038080	PL ANTSPTC SPG MNT MTHWS	PRIVATE LABEL	ORAL HYGIENE PRODUCTS	MOUTHWASHES (ANTISEPTIC)	500 ML

```
products.dtypes
```

```
UPC              int64
DESCRIPTION     object
MANUFACTURER    object
CATEGORY        object
SUB_CATEGORY    object
PRODUCT_SIZE    object
dtype: object
```

Categorical Variables

All of the variables in the Products dataframe are categorical, except for UPC code which is just the identifier which will be used to join with the Sales dataframe. As such, the following issues need to be addressed:

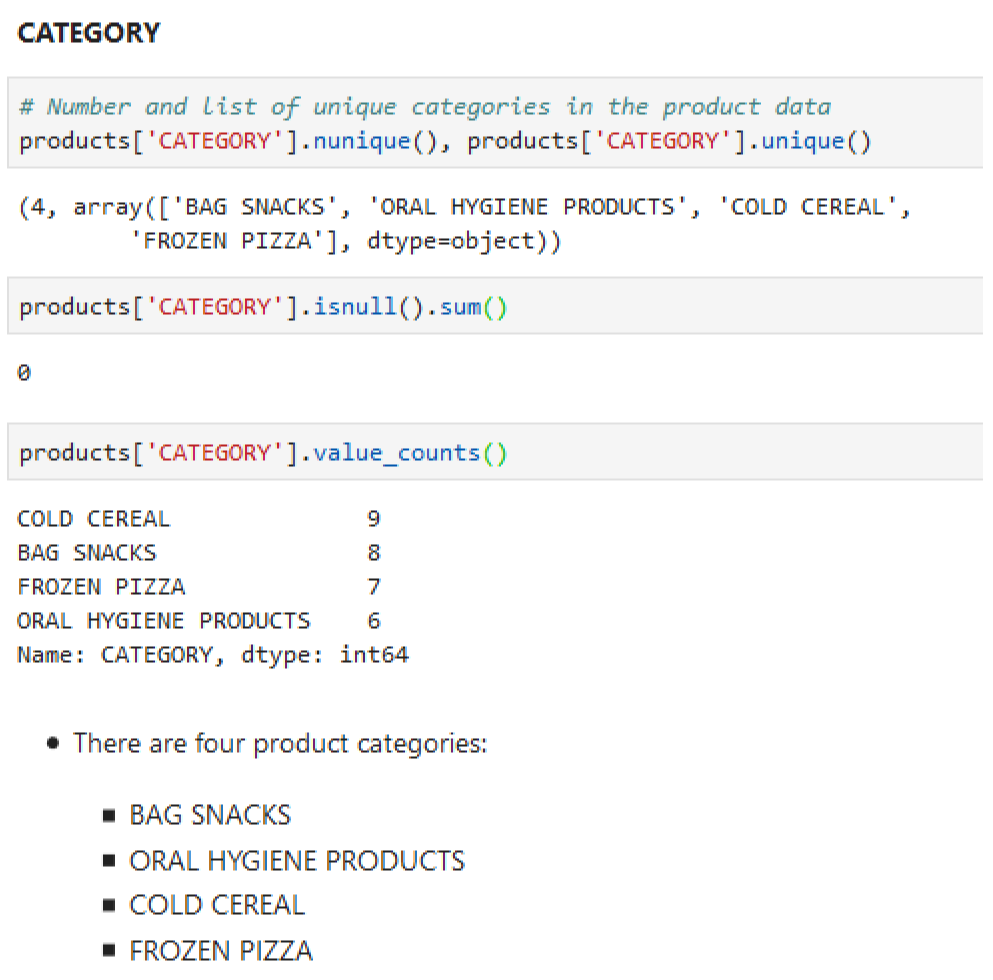
- Check unique values.
- Are there any missing values?
- Are there any variables with high cardinality or sparsity?

UPC

- In examining the 'UPC' variable, we found 30 unique values, which we validated were identical to the 'UPC' variable values in the Sales dataframe.

CATEGORY

- There are no missing values.
- The 'CATEGORY' variable has four unique values. The details for which are shown below:



CATEGORY

```
# Number and list of unique categories in the product data
products['CATEGORY'].nunique(), products['CATEGORY'].unique()
```

```
(4, array(['BAG SNACKS', 'ORAL HYGIENE PRODUCTS', 'COLD CEREAL',
        'FROZEN PIZZA'], dtype=object))
```

```
products['CATEGORY'].isnull().sum()
```

```
0
```

```
products['CATEGORY'].value_counts()
```

```
COLD CEREAL              9
BAG SNACKS               8
FROZEN PIZZA             7
ORAL HYGIENE PRODUCTS    6
Name: CATEGORY, dtype: int64
```

- There are four product categories:
    - BAG SNACKS
    - ORAL HYGIENE PRODUCTS
    - COLD CEREAL
    - FROZEN PIZZA

- There are 9 products with the category 'Cold Cereal', 8 products labeled 'Bag snacks', 7 with category 'Frozen Pizza' and 6 'Oral Hygiene' Products.

SUB_CATEGORY

- There are no missing values.
- The 'SUB_CATEGORY' variable has four unique values. The details for which are shown below:

SUB_CATEGORY

```python
# Check for null values.
products['SUB_CATEGORY'].isnull().sum()
```

```
0
```

```python
products['SUB_CATEGORY'].nunique()
```

```
7
```

```python
# Display subcategories for each category
products[['CATEGORY','SUB_CATEGORY']].drop_duplicates().sort_values(by = 'CATEGORY')
```

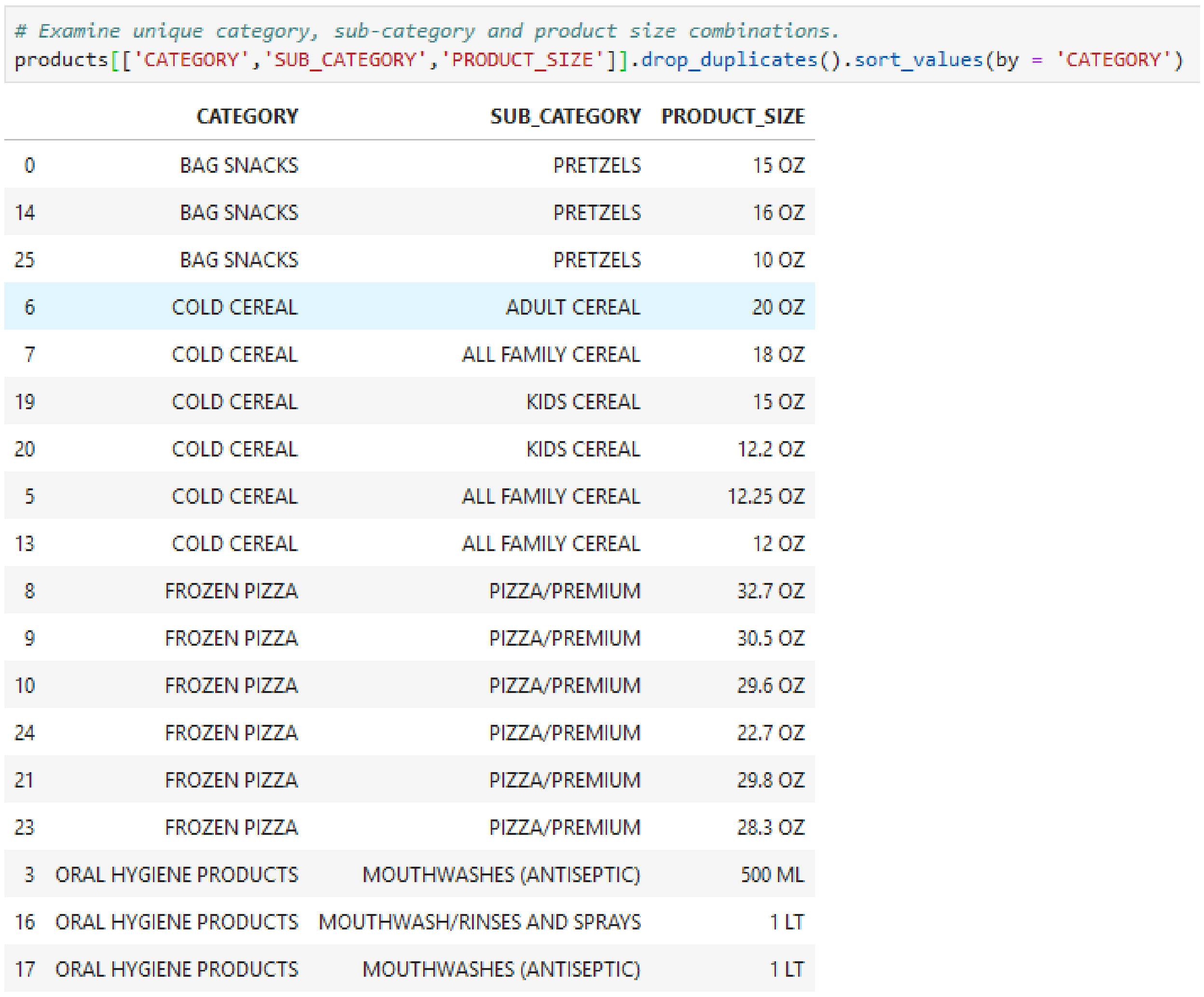
	CATEGORY	SUB_CATEGORY
0	BAG SNACKS	PRETZELS
5	COLD CEREAL	ALL FAMILY CEREAL
6	COLD CEREAL	ADULT CEREAL
19	COLD CEREAL	KIDS CEREAL
8	FROZEN PIZZA	PIZZA/PREMIUM
3	ORAL HYGIENE PRODUCTS	MOUTHWASHES (ANTISEPTIC)
16	ORAL HYGIENE PRODUCTS	MOUTHWASH/RINSES AND SPRAYS

The sub-categories give additional detail about the products.

- Cereal has 3 sub categories, differentiating on the age group.
- Oral hygiene products have 2 sub categories, antiseptic and rinse/spray.
- Bag Snacks & Frozen Pizza have just 1 sub category.

PRODUCT_SIZE

- The following is a summary of the PRODUCT_SIZE variable:



```python
# Examine unique category, sub-category and product size combinations.
products[['CATEGORY','SUB_CATEGORY','PRODUCT_SIZE']].drop_duplicates().sort_values(by = 'CATEGORY')
```

	CATEGORY	SUB_CATEGORY	PRODUCT_SIZE
0	BAG SNACKS	PRETZELS	15 OZ
14	BAG SNACKS	PRETZELS	16 OZ
25	BAG SNACKS	PRETZELS	10 OZ
6	COLD CEREAL	ADULT CEREAL	20 OZ
7	COLD CEREAL	ALL FAMILY CEREAL	18 OZ
19	COLD CEREAL	KIDS CEREAL	15 OZ
20	COLD CEREAL	KIDS CEREAL	12.2 OZ
5	COLD CEREAL	ALL FAMILY CEREAL	12.25 OZ
13	COLD CEREAL	ALL FAMILY CEREAL	12 OZ
8	FROZEN PIZZA	PIZZA/PREMIUM	32.7 OZ
9	FROZEN PIZZA	PIZZA/PREMIUM	30.5 OZ
10	FROZEN PIZZA	PIZZA/PREMIUM	29.6 OZ
24	FROZEN PIZZA	PIZZA/PREMIUM	22.7 OZ
21	FROZEN PIZZA	PIZZA/PREMIUM	29.8 OZ
23	FROZEN PIZZA	PIZZA/PREMIUM	28.3 OZ
3	ORAL HYGIENE PRODUCTS	MOUTHWASHES (ANTISEPTIC)	500 ML
16	ORAL HYGIENE PRODUCTS	MOUTHWASH/RINSES AND SPRAYS	1 LT
17	ORAL HYGIENE PRODUCTS	MOUTHWASHES (ANTISEPTIC)	1 LT

- In reviewing the SUB_CATEGORY with PRODUCT_SIZE, there are no combinations that indicate that SUB_CATEGORY is an indicator of size.

To summarize:

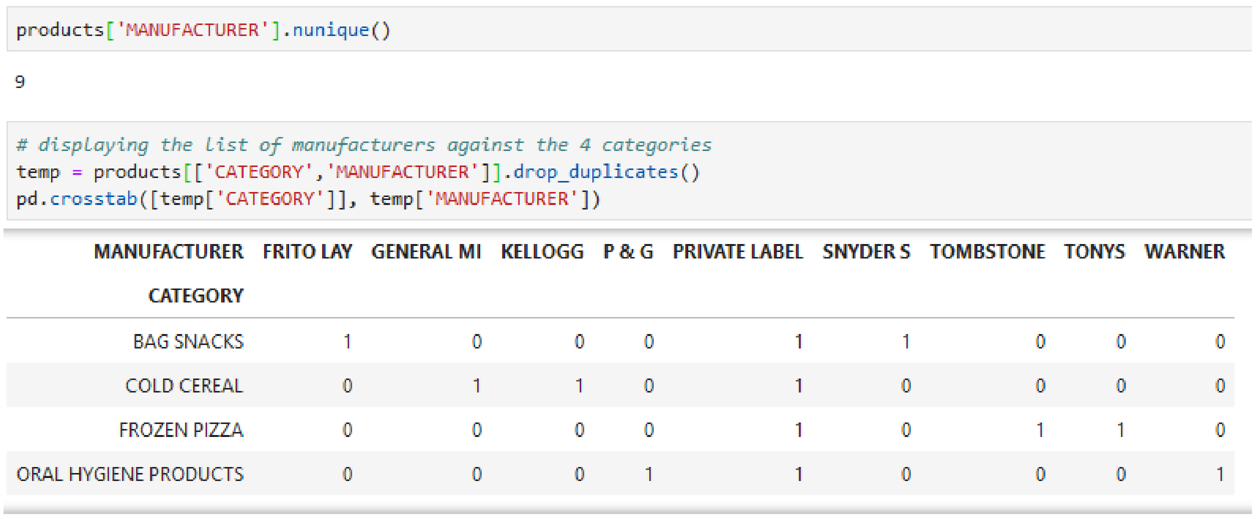
- Bag Snacks has 1 sub-category and 3 product sizes.
- Oral Hygiene has 2 sub-categories and 2 size options.
- Frozen Pizza has only 1 sub-category and 6 different package sizes.
- Cold Cereal has 3 sub-categories, and 6 size options.

DESCRIPTION

- There are no missing values.
- There are 29 unique values, with one description (GM CHEERIOS) being used twice.
- GM CHEERIOS uses the same description for two product sizes (18 OZ & 12 OZ).

MANUFACTURER

- There are no missing values.
- There are 9 unique values, broken down as follows:



```
products['MANUFACTURER'].nunique()
```

```
9
```

```
# displaying the list of manufacturers against the 4 categories
temp = products[['CATEGORY','MANUFACTURER']].drop_duplicates()
pd.crosstab([temp['CATEGORY']], temp['MANUFACTURER'])
```

MANUFACTURER	FRITO LAY	GENERAL MI	KELLOGG	P & G	PRIVATE LABEL	SNYDER S	TOMBSTONE	TONYS	WARNER
CATEGORY									
BAG SNACKS	1	0	0	0	1	1	0	0	0
COLD CEREAL	0	1	1	0	1	0	0	0	0
FROZEN PIZZA	0	0	0	0	1	0	1	1	0
ORAL HYGIENE PRODUCTS	0	0	0	1	1	0	0	0	1

- With 4 unique categories of Products, each category has three different manufacturers.
- Each category has a manufacturer identified as 'private label' along with 2 other manufacturers.

Stores Dataframe EDA

The following is a summary of the columns of the store_data.csv dataset, which as mentioned earlier contains 76 rows of data.

Store Data: ('store_data.csv')

- **STORE_ID** - store number
- **STORE_NAME** - Name of store
- **ADDRESS_CITY_NAME** - city
- **ADDRESS_STATE_PROV_CODE** - state
- **MSA_CODE** - (Metropolitan Statistical Area) Based on geographic region and population density
- **SEG_VALUE_NAME** - Store Segment Name
- **PARKING_SPACE_QTY** - number of parking spaces in the store parking lot
- **SALES_AREA_SIZE_NUM** - square footage of store
- **AVG_WEEKLY_BASKETS** - average weekly baskets sold in the store

We will examine Numerical and Categorical variables separately.

Numerical Variables

- Are there any missing values in the variables?
- What does the distribution look like?
- Are there any extreme/outlier values?

Categorical Variables

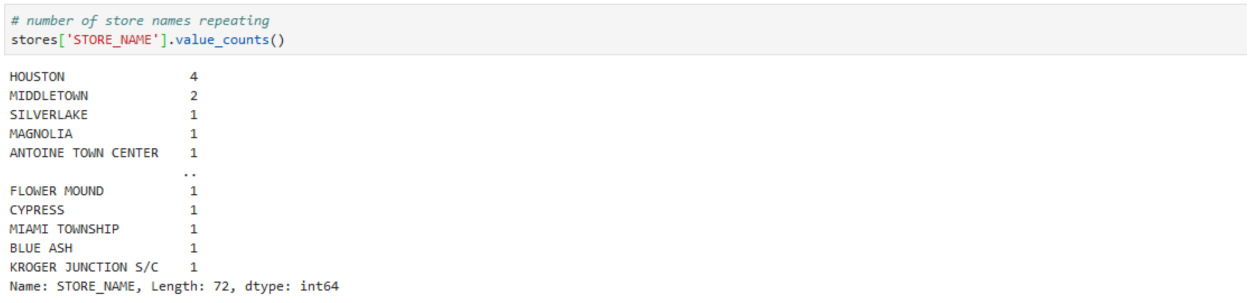
- Check the unique values for categorical variables.
- Are there any missing values in the variables?
- Are there any variables with high cardinality or sparsity?

STORE_ID & STORE_NAME

STORE_ID is a key variable and will be used to join with the Sales dataframe later.

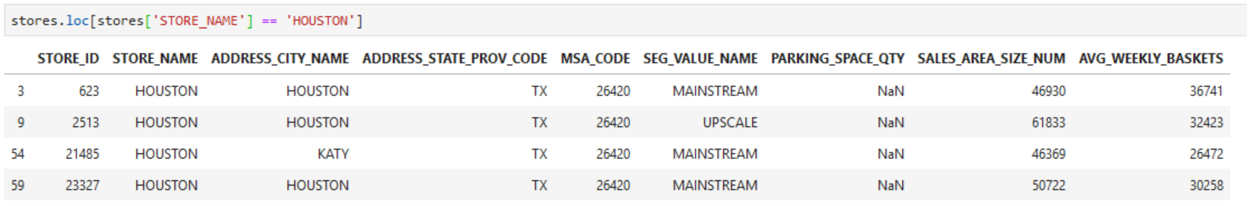
STORE_NAME is a categorical value that represents the city that the store is located in.

- There are 76 unique values in STORE_ID
- There are 72 unique values in STORE_NAME
- Some store names are being repeated, which means there are some cities with more than one store.



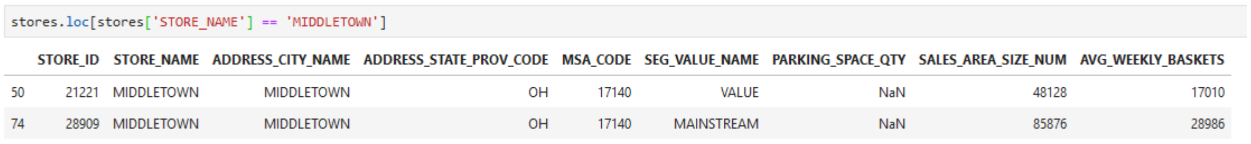
```
# number of store names repeating
stores['STORE_NAME'].value_counts()
```

```
HOUSTON                4
MIDDLETOWN             2
SILVERLAKE             1
MAGNOLIA               1
ANTOINE TOWN CENTER    1
                      ..
FLOWER MOUND           1
CYPRESS                1
MIAMI TOWNSHIP         1
BLUE ASH               1
KROGER JUNCTION S/C    1
Name: STORE_NAME, Length: 72, dtype: int64
```



```
stores.loc[stores['STORE_NAME'] == 'HOUSTON']
```

	STORE_ID	STORE_NAME	ADDRESS_CITY_NAME	ADDRESS_STATE_PROV_CODE	MSA_CODE	SEG_VALUE_NAME	PARKING_SPACE_QTY	SALES_AREA_SIZE_NUM	AVG_WEEKLY_BASKETS
3	623	HOUSTON	HOUSTON	TX	26420	MAINSTREAM	NaN	46930	36741
9	2513	HOUSTON	HOUSTON	TX	26420	UPSCALE	NaN	61833	32423
54	21485	HOUSTON	KATY	TX	26420	MAINSTREAM	NaN	46369	26472
59	23327	HOUSTON	HOUSTON	TX	26420	MAINSTREAM	NaN	50722	30258



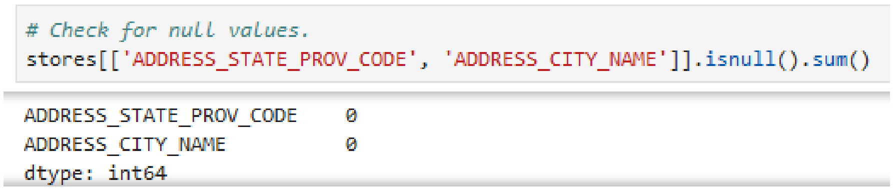
```
stores.loc[stores['STORE_NAME'] == 'MIDDLETOWN']
```

	STORE_ID	STORE_NAME	ADDRESS_CITY_NAME	ADDRESS_STATE_PROV_CODE	MSA_CODE	SEG_VALUE_NAME	PARKING_SPACE_QTY	SALES_AREA_SIZE_NUM	AVG_WEEKLY_BASKETS
50	21221	MIDDLETOWN	MIDDLETOWN	OH	17140	VALUE	NaN	48128	17010
74	28909	MIDDLETOWN	MIDDLETOWN	OH	17140	MAINSTREAM	NaN	85876	28986

- We see that four stores are named ‘Houston’ and two are named ‘Middletown’. Each store has a different segment value, location and/or sales area size, so they are in fact different stores.
- There are no missing values.

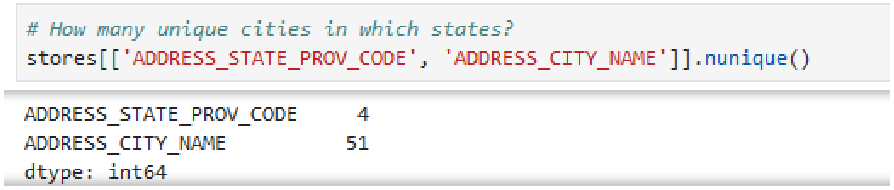
ADDRESS_CITY_NAME and ADDRESS_STATE_PROV_CODE

- There are no missing values.
- There are 51 unique city names in 4 states.



```
# Check for null values.
stores[['ADDRESS_STATE_PROV_CODE', 'ADDRESS_CITY_NAME']].isnull().sum()
```

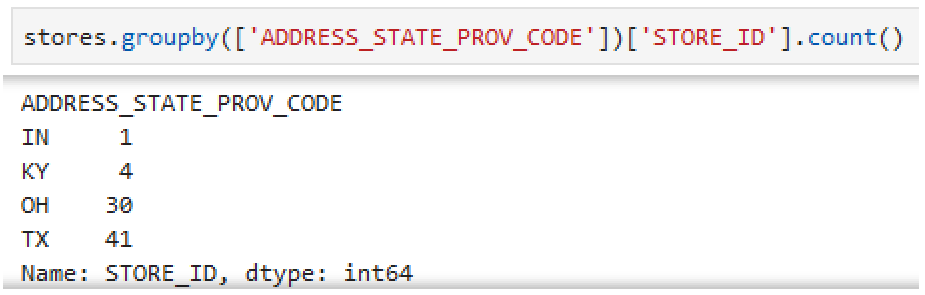
```
ADDRESS_STATE_PROV_CODE    0
ADDRESS_CITY_NAME          0
dtype: int64
```



```
# How many unique cities in which states?
stores[['ADDRESS_STATE_PROV_CODE', 'ADDRESS_CITY_NAME']].nunique()
```

```
ADDRESS_STATE_PROV_CODE     4
ADDRESS_CITY_NAME          51
dtype: int64
```

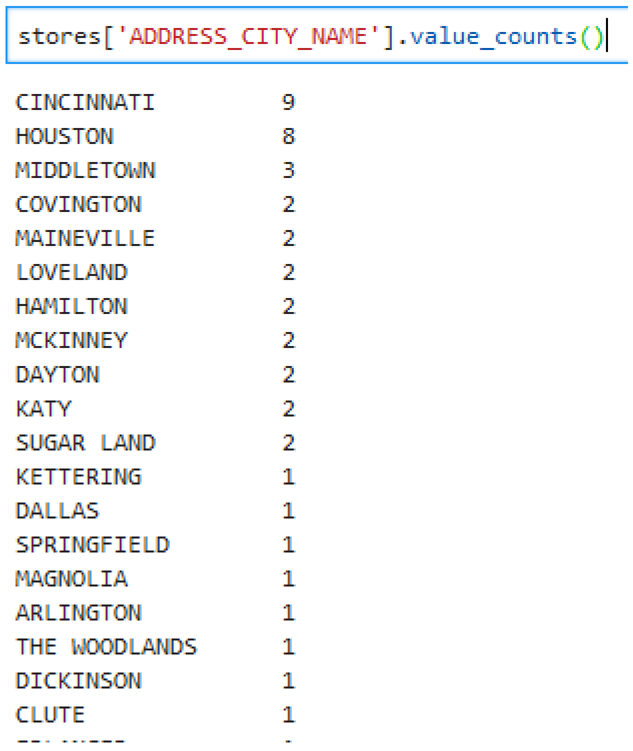
- The number of stores per state with some interesting findings:



```
stores.groupby(['ADDRESS_STATE_PROV_CODE'])['STORE_ID'].count()
```

```
ADDRESS_STATE_PROV_CODE
IN     1
KY     4
OH    30
TX    41
Name: STORE_ID, dtype: int64
```

- Each store has a unique store ID
- Most stores are from Ohio and Texas ~93%
- Few from Kentucky and Indiana ~7%



```
stores['ADDRESS_CITY_NAME'].value_counts()
```

```
CINCINNATI         9
HOUSTON            8
MIDDLETOWN         3
COVINGTON          2
MAINEVILLE         2
LOVELAND           2
HAMILTON           2
MCKINNEY           2
DAYTON             2
KATY               2
SUGAR LAND         2
KETTERING          1
DALLAS             1
SPRINGFIELD        1
MAGNOLIA           1
ARLINGTON          1
THE WOODLANDS      1
DICKINSON          1
CLUTE              1
```

- Cincinnati and Houston have the most stores (partial list, sorted from most to least).
- 11 cities have more than one store.

MSA_CODE

- There are no missing values.
- There are 9 unique MSA code values
- The top 3 MSA codes are ‘17140’ with 29, ‘26420’ with 21, and ‘19100’ with 17.

```
stores['MSA_CODE'].nunique(), stores['MSA_CODE'].unique()
```

```
(9, array([17140, 19100, 26420, 17780, 47540, 43300, 19380, 13140, 44220],
       dtype=int64))
```

```
stores['MSA_CODE'].value_counts()
```

```
17140    29
26420    21
19100    17
19380     4
13140     1
47540     1
44220     1
43300     1
17780     1
Name: MSA_CODE, dtype: int64
```

```
(stores.groupby(['MSA_CODE', 'ADDRESS_STATE_PROV_CODE'])['STORE_ID'].count())
```

```
MSA_CODE  ADDRESS_STATE_PROV_CODE
13140     TX                          1
17140     IN                          1
          KY                          4
          OH                         24
17780     TX                          1
19100     TX                         17
19380     OH                          4
26420     TX                         21
43300     TX                          1
44220     OH                          1
47540     OH                          1
Name: STORE_ID, dtype: int64
```

- These codes are assigned based on the geographical location and population density.
- 17140 is present in all three except Texas (which has a different geographical region)

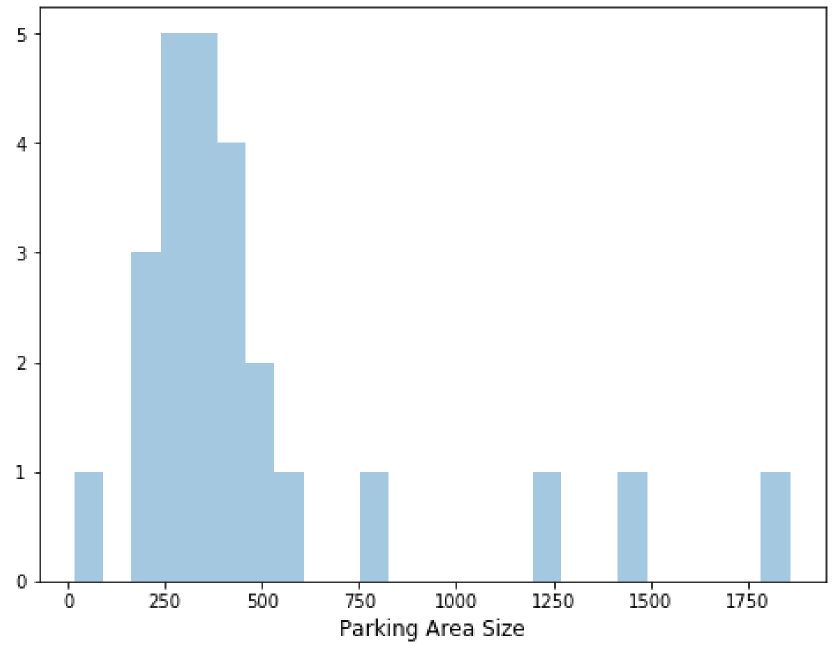
PARKING_SPACE_QTY and SALES_AREA_SIZE_NUM

- Of the 76 stores, parking area is missing for 51 of them.

```
stores[['PARKING_SPACE_QTY', 'SALES_AREA_SIZE_NUM']].isnull().sum()
```

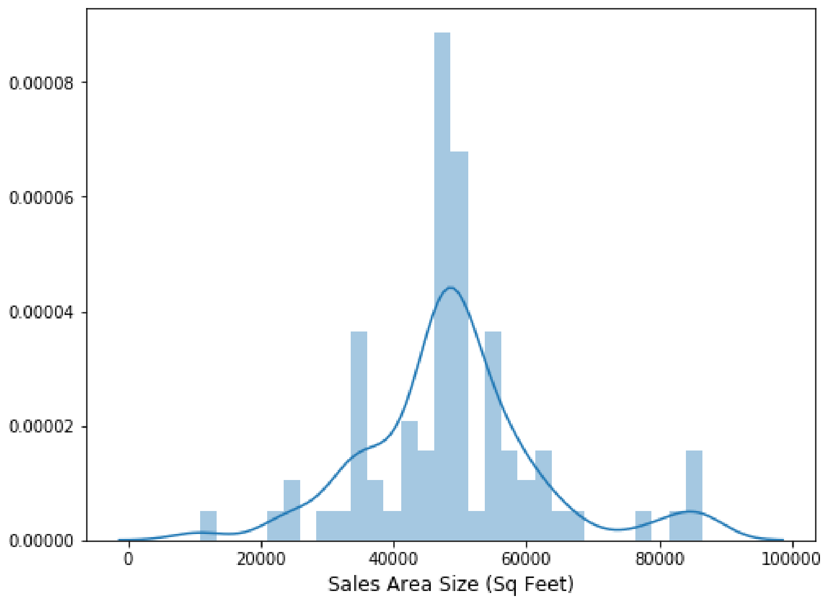
```
PARKING_SPACE_QTY      51
SALES_AREA_SIZE_NUM     0
dtype: int64
```

```
plt.figure(figsize=(8,6))
sns.distplot(stores['PARKING_SPACE_QTY'], bins=25, kde=False)
plt.xlabel('Parking Area Size', fontsize=12)
plt.show()
```



- About 15 stores have between 250-500 parking spaces.

```
plt.figure(figsize=(8,6))
sns.distplot(stores['SALES_AREA_SIZE_NUM'], bins=30, kde=True)
plt.xlabel('Sales Area Size (Sq Feet)', fontsize=12)
plt.show()
```



- Most stores have between 30,000 and 70,000 square feet of sales area.
- Only a few of the stores have less than 30,000 square feet or more than 90,000 square feet of sales area.

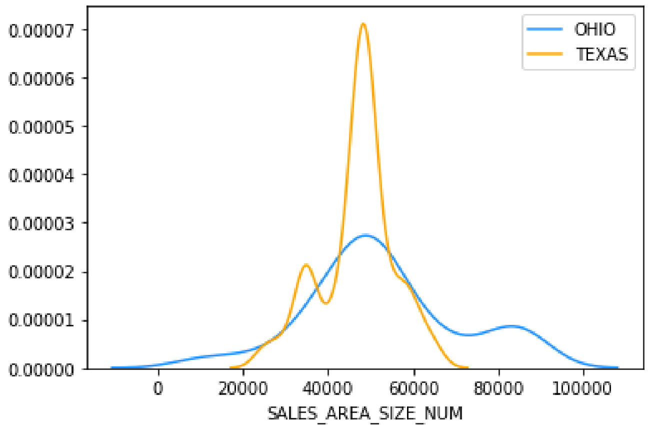
```
(stores.groupby(['ADDRESS_STATE_PROV_CODE'])['SALES_AREA_SIZE_NUM'].mean()).sort_values(ascending=False)
```

```
ADDRESS_STATE_PROV_CODE
IN    58563.000000
OH    52691.200000
TX    46920.902439
KY    39855.500000
Name: SALES_AREA_SIZE_NUM, dtype: float64
```

```
state_oh = stores.loc[stores['ADDRESS_STATE_PROV_CODE'] == 'OH']
state_tx = stores.loc[stores['ADDRESS_STATE_PROV_CODE'] == 'TX']

sns.distplot(state_oh['SALES_AREA_SIZE_NUM'], hist=False,color= 'dodgerblue', label= 'OHIO')
sns.distplot(state_tx['SALES_AREA_SIZE_NUM'], hist=False,  color= 'orange', label= 'TEXAS')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x173d0ca48c8>
```



- Indiana has the largest mean store sales area size; Kentucky has the smallest.
- Texas has some of the largest stores, but also some smaller ones too, which brings the Texas overall store sales area size down compared to Indiana and Ohio.
- Ohio's stores are more evenly distributed.

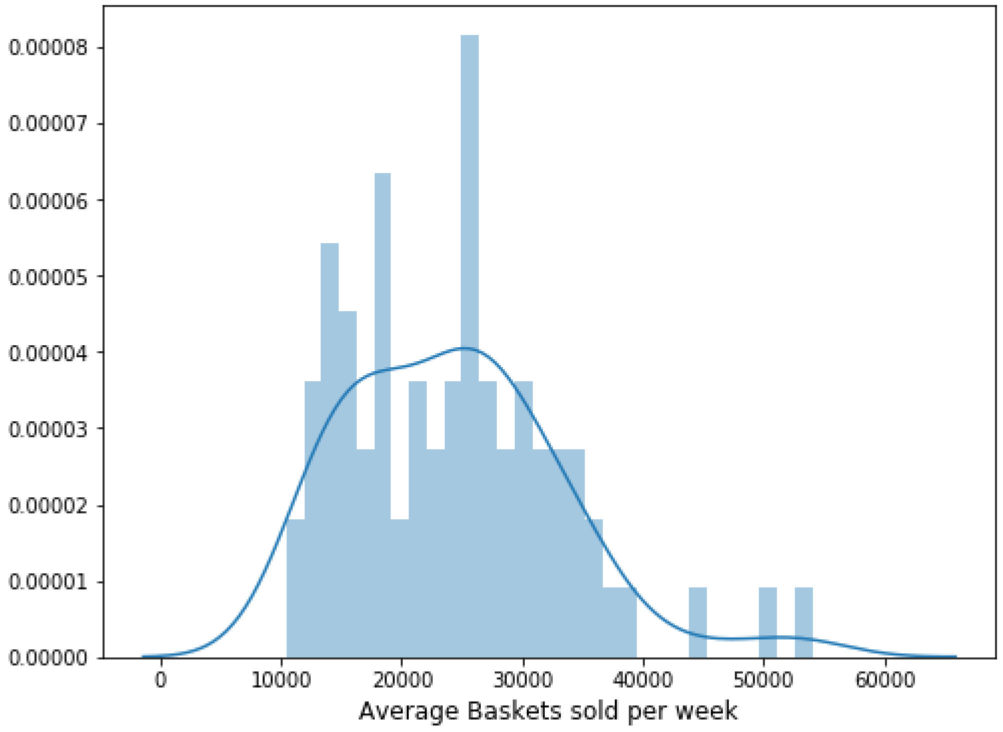
AVG_WEEKLY_BASKETS

- There are no missing values.
- The basic statistics for Average Baskets sold per week and associated distribution are as follows:

```
stores['AVG_WEEKLY_BASKETS'].describe()
```

```
count       76.000000
mean     24226.921053
std       8863.939362
min      10435.000000
25%      16983.500000
50%      24667.500000
75%      29398.500000
max      54053.000000
Name: AVG_WEEKLY_BASKETS, dtype: float64
```

```
plt.figure(figsize=(8,6))
sns.distplot(stores['AVG_WEEKLY_BASKETS'], bins=30, kde=True)
plt.xlabel('Average Baskets sold per week', fontsize=12)
plt.show()
```


Average Baskets sold per week