Cannabis Sales Revenue Report

Nuutti Barron - COM SCI M148
UID: 805205422

Executive Summary

For this project we analyze a large dataset of licensed cannabis sales in the state of California in order to determine factors that potentially affect sales, and to develop models meant to predict future sales. The first step is to merge together data from multiple sources, as well as combine time series data with product and brand information. Following this step we gather basic statistics, such as mean, median, standard deviation, and visualized distributions of all the features we analyze. This provides a general sense of what the data we are working with looks like. We also calculate correlations between all features in hopes of identifying factors correlated with monthly sales, as these can be adjusted to improve sales going forward. The features that we discovered to be most strongly correlated with monthly sales revenue are outlined in the following table:

Positive Correlation	Negative Correlation
Offer products with Pax filters	Mean mg THC across all products
Offer product types other than inhalables and ingestibles	Average Retail Price
Offering CBD infused products	
The number of products offered	
Price range in products offered	
Number of products with a mood effect	
Number of products that are flavored	

Note: Factors that we found to be correlated with sales, but do not provide useful information (such as revenue in the previous month), are excluded from this table

Following this step we trained Linear Regression, K-Nearest Neighbor (KNN), Random Forest, and Neural Network models in order to predict future sales. We implement hyperparameter optimization to optimize each model, and cross validation in order to build confidence in the accuracy of each model. Attempted principal component analysis yielded decreased performance metrics across each model, so we do not use this technique to alter our dataset when training and testing each model. Below we provide the mean absolute error (MAE), the average magnitude of error in predicted sales, and mean absolute percentage error (MAPE), the average percent error, of each model we built:

Model	**MAE**
Linear Regression	83673.16
KNN Regression	80769.39
Random Forest Regression	78643.26
Neural Network Regression	83010.92

Introduction and Background

In this project we analyze cannabis sales data from a wide variety of companies in the state of California collected over the last few years. Our goal is to identify potential factors that influence sales, as well as develop computational, data-driven models to accurately predict future sales. There are various reasons that make meaningful data analysis and the ability to build accurate predictive models complicated. One reason is the legal status of cannabis, and the varying regulatory status of cannabis across the United States. The legal status varies from state to state, meaning the supply chain for the industry in nonuniform, which affects the type of products sold, the demand, and other factors in the cannabis industry. This means that meaningful data analysis in one region will likely not translate to another. Our data is focused on licensed sale of cannabis in California (where Cannabis is legal for medical and recreational use), which alleviates this complexity, but it nevertheless affects our dataset. Another factor is the presence of new forms of cannabis products (other than inhalable and ingestible). The emergence of CBD based products, Pax filter products, flavored, and mood specific products is an example of this. Legal cannabis sales is an industry that is still emerging, making it generally more unpredictable.

In order to identify potential factors that influence sales, we analyze the dataset provided by Cookies. The provided dataset includes time series data for cannabis sales, meaning it provides timestamped information about total sales, total units sold, and what types of products were sold, among other information. It also contains more detailed brand information about individual brands and products. In order to analyze the dataset, the data from different sources is merged to a common data frame to be further analyzed. This step involves combining and augmenting time series data with undated product data. From the combined dataframe, we obtain various statistics, such as the mean, median, and standard deviation, for all of the variables that we consider, providing a sense of what the data looks like. For example, we extract that the average monthly sales across all brands is $409,372.90 with a standard deviation of $1,596,024 , meaning that there is a lot of variability in total sales, and that it does not resemble a normal distribution. We also visualize the distribution of the data to get a sense of what we are working with. More interestingly, we identify correlations between the variables, or features, that we consider. Most important are the features that have a strong correlation (negative or positive) with total sales, as these can be changed in hopes of boosting cannabis sales. Some factors, such as the sales from the previous month, are strongly correlated, but trivial, in that it does not provide any useful information. Variables found to have a non-negligible correlation with sales are outlined in the executive summary above.

Using the features we found to be correlated with sales, we train models intended to predict future sales. First we augment the dataset to make it suitable for model training. This includes scaling nonbinary features, encoding categorical features, and imputing the data to get rid of unknown, or null, values. We implement a pipeline for all of this, making the data augmentation easily reproducible. Following these steps, we build a simple Linear Regression, a K-Nearest Neighbor Regression, a Random Forest Regression, and a Neural Network Regression model in order to predict future monthly sales. In order to maximize the performance of each model, we implement hyperparameter optimization, and in order to validate the performance of each model, we extract cross fold validation metrics. We test the effect principal component analysis (to reduce the dimensionality of the dataset) has on our model performances, but it resulted in decreased model performance, so for our finalized models we do not do this to the dataset.

Methodology

In order to develop a dataset we can analyze and train on future models, we must merge data from multiple data files. An added layer of complexity is added since some of the data files contain time series data and some do not. In order to make one unified dataset, we remove the dates of sales made and implement a feature for previous months sales, as well as a feature for the brand's sales over a three month rolling average. We can then include the variables from the undated data. For example, if a brand indicates that they sell CBD based products, we add a feature to indicate that the brand sells CBD products. We follow this methodology for the important features in the undated data (e.g. whether the brand sells flavored products, mood directed product, Pax filters, ingestible products, etc). Although this step may not be necessary for basic data exploration and analysis, this step is crucial to build predictive models using time series data.

The dataset is further altered for a few reasons. First of all, categorical features must be encoded, so we implement our own binary encoding for categorical features that is similar to one hot encoding (in our features, more than one categorical value is possible, one hot encoding alone would not suffice). Some features also have greater variance in their data points than others, which can affect model training, and bias the importance of these features. In order to counteract this, we apply a standard scalar to features that contain nonbinary, continuous data. Finally, we must impute unknown values in the dataset. Different features are imputed in different ways. For binary features that indicate if a brand sells a certain type of product, we assume that the brand does not sell this type of product if it is not indicated, so we set the unknown values in these columns to 0. For scalar features, we assign the median value of the known data. The reasoning behind this is that there are few unknown values relative to the total amount of data in these columns, but these features are not normally distributed, so applying the mean would likely skew models trained on this data more than imputing the median. These steps are all implemented in a pipeline fashion, so these steps are easily reproducible.

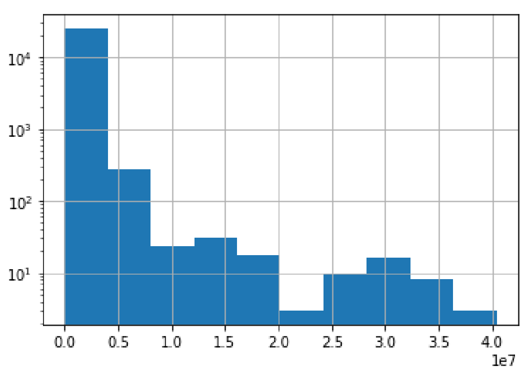
Following the implementation of a Linear Regression model, we implemented a random forest regression to test the efficacy of an ensemble regression method on our dataset. As opposed to linear regression, this model has hyperparameters that we tune in order to optimize the model. The hyperparameters that we optimized are the maximum tree depth, and the number of trees in the random forest. We actually found improved performance metrics as opposed to simple linear regression using this model.

In order to validate the results of our model, we apply K-fold cross validation. In doing this, we split the dataset into 10 train-test splits, train the random forest model on each split, retrieve performance metrics on the current split, and average these performance metrics. This provides a greater level of confidence in the validity of the models we build and train. The variance in performance metrics from our original models was essentially negligible, providing greater confidence in their validity and reproducibility.

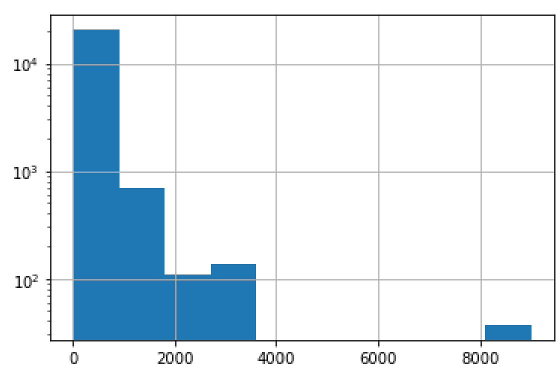
Results

After merging all of our time series and brand detail data into one dataframe, we analyze the basic statistics for the features in the dataset. The following displays distributions for a subset of the features that we consider.

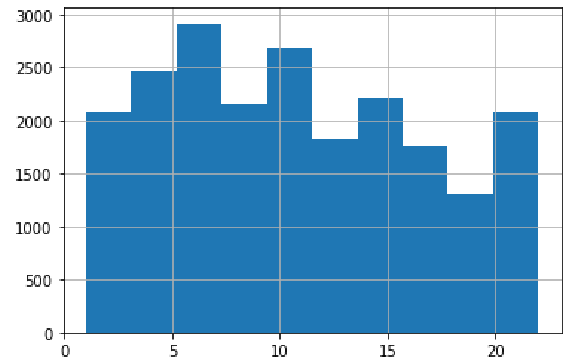
Monthly Sales ($)



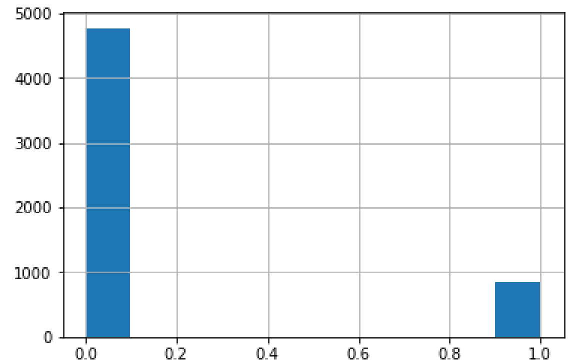
Product Count



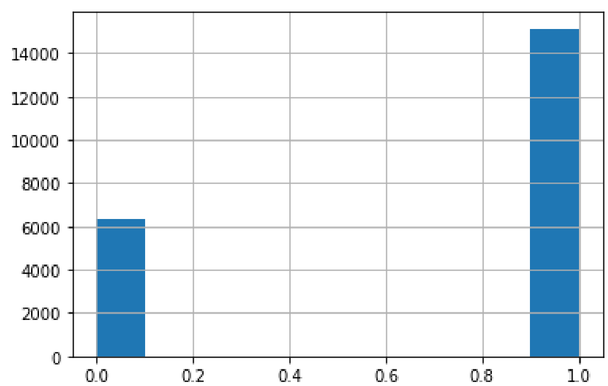
Price Range



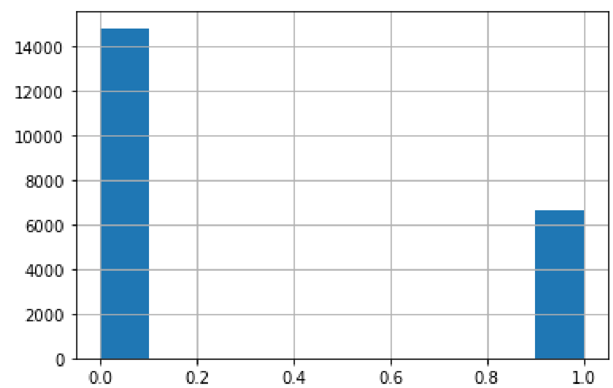
Pax Filter Product Offered



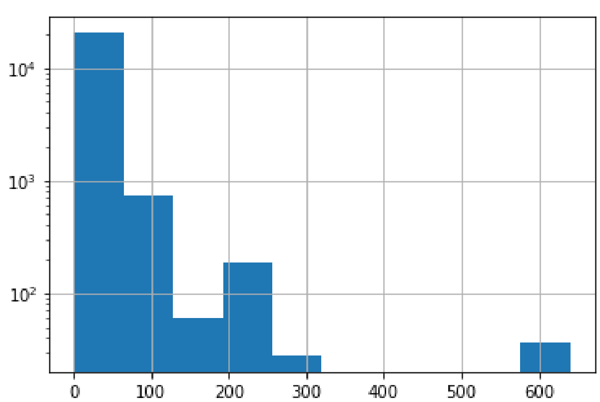
Inhalable Products Offered



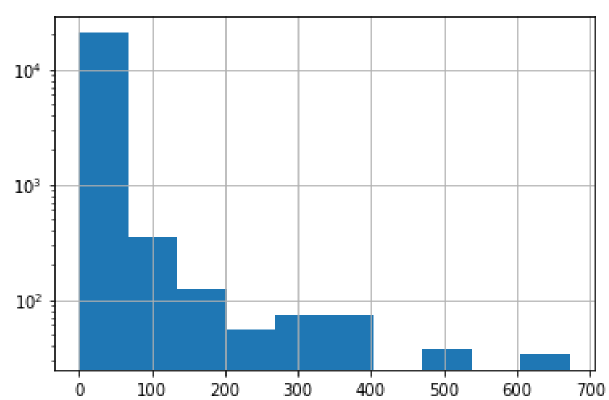
Ingestible Products Offered



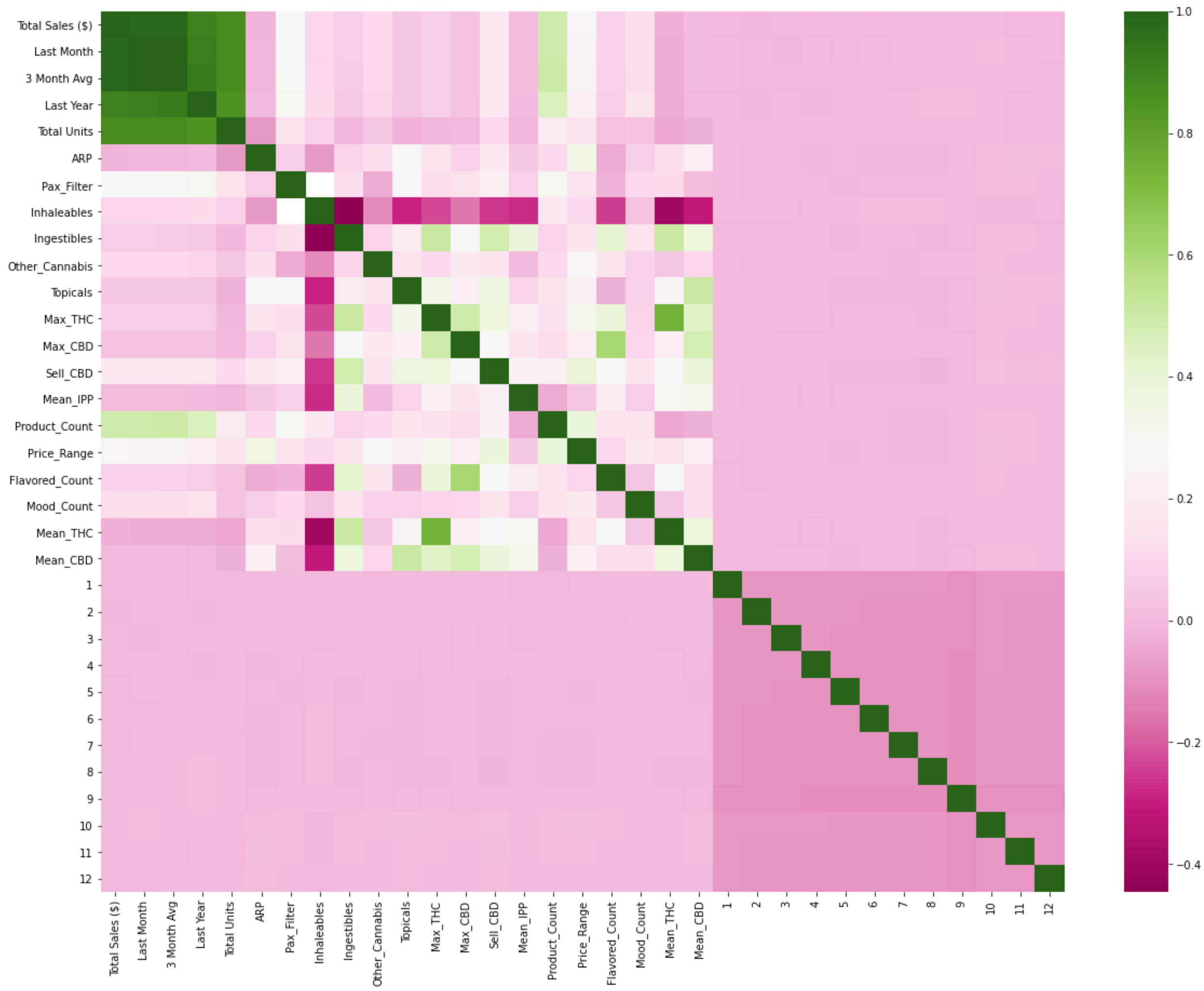
Number of Flavored Products Offered



Number of Mood Specific Products Offered



We also obtain a heatmap of correlations between important variables in our unified dataframe. The features correlated (positively or negatively) with ‘Total Sales ($)’ are the features we deem the most important, as this is the value the models we build will try to predict.



Note: Features 1-12 here indicate the month of the year of the current entry. I suspected that sales may generally increase in certain months and decrease in others, but this heat map indicates otherwise

After running basic statistical analysis on our data, we build a linear regression model with 17 features used to predict future sales. Our linear regression reported the following metrics on the testing set:

$$R^2 = 0.9776,\ MAE = 89337.48,\ MSE = 71251360719.15,\ RMSE = 266929.51$$

Coefficients for Each Feature

Feature 0 - Last_Month Score: -16726.93808

Feature 1 - 3_Month_Avg Score: -1252.40326

Feature 2 - Last_Year Score: -9613.25731

Feature 3 - ARP Score: 41837.51016

Feature 4 - Pax_Filer Score: 10005.31014

Feature 5 - Inhaleables Score: 2039.77175

Feature 6 - Ingestibles Score: 966.01786

Feature 7 - Other_Cannabis Score: -4060.54556

Feature 8 - Topicals Score: -6173.71553

Feature 9 - Max_THC Score: 13567.07529

Feature 10 - Sell_CBD Score: 12725.10978

Feature 11 - Product_Count Score: 27512.33976

Feature 12 - Price_Range Score: -10621.30227

Feature 13 - Flavored_Count Score: 18818.28732

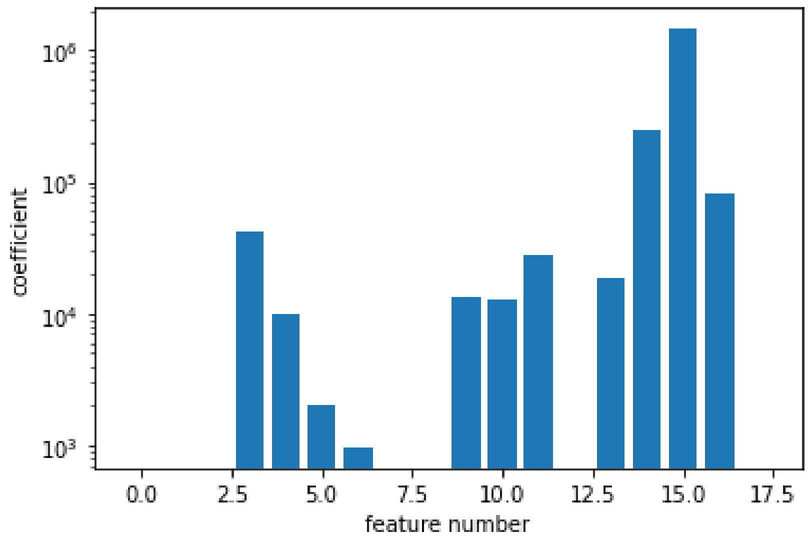
Feature 14 - Mood_Count Score: 245711.08046

Feature 15 - Mean_THC Score: 1453753.73643

Feature 16 - MTHC_Ing_Cross Score: 81676.23301

Feature 17 - New_Product Score: -149.58918

Bar graph of the Feature Coefficient



Other statistical metrics for our linear regression

feature	Coef	std err	t	P>|t|	[0.025	0.975]
const	1.616e+05	1.07e+04	15.154	0.000	1.41e+05	1.82e+05
x1	-1.673e+04	4277.383	-3.911	0.000	-2.51e+04	-8342.948
x2	-1252.4033	1.18e+04	-0.106	0.916	-2.45e+04	2.2e+04
x3	-9613.2573	2737.378	-3.512	0.000	-1.5e+04	-4247.792
x4	4.184e+04	3331.598	12.558	0.000	3.53e+04	4.84e+04
x5	1.001e+04	3166.037	3.160	0.002	3799.642	1.62e+04
x6	2039.7717	2988.370	0.683	0.495	-3817.656	7897.200
x7	966.0179	2587.289	0.373	0.709	-4105.261	6037.296
x8	-4060.5456	1.11e+04	-0.366	0.714	-2.58e+04	1.77e+04
x9	-6173.7155	1.53e+04	-0.403	0.687	-3.62e+04	2.38e+04
x10	1.357e+04	5931.178	2.287	0.022	1941.525	2.52e+04
x11	1.273e+04	8093.040	1.572	0.116	-3137.851	2.86e+04
x12	2.751e+04	1.2e+04	2.293	0.022	3995.970	5.1e+04
x13	-1.062e+04	1.15e+04	-0.923	0.356	-3.32e+04	1.19e+04
x14	1.882e+04	7134.866	2.638	0.008	4833.418	3.28e+04
x15	2.457e+05	1.03e+04	23.923	0.000	2.26e+05	2.66e+05
x16	1.454e+06	8437.734	172.292	0.000	1.44e+06	1.47e+06
x17	8.168e+04	9155.607	8.921	0.000	6.37e+04	9.96e+04
x18	-149.5892	4117.140	-0.036	0.971	-8219.491	7920.312

Using a p-value threshold of 0.05, we determine that the following variables are important in this linear regression: Previous months revenue, previous years revenue, average retail price, sells Pax filter products, Max mg THC in any product offered, number of products offered, number of flavored products offered, number of mood specific products offered, average THC across all products, and the cross feature of max THC offered and ingestible products offered.

The ensemble model we trained was a Random Forest. The following are the 10-fold cross validated metrics reported by the trained random forest on our dataset using the same features as our linear regression.

Random Forest Regression Performance Metrics

R^2	0.9484
MAE	87530.34
MSE	137768222451.09
RMSE	331004.82

In order to optimize our random forest model, we hyperparameter optimization using GridSearch. We found that setting the max depth to 15, and the number of estimators (or decision trees) to 50. With these parameters, we got the following cross validated performance metrics.

Random Forest Regression Performance Metrics After Hyperparameter Optimization

R^2	0.98
MAE	77355.78
MSE	42009717731.5782
RMSE	204962.7228

We see that the performance metrics all improved following hyperparameter tuning (R-squared increased, and MAE, MSE, and RMSE all decreased).

To test a few additional models, we implemented K-Nearest Neighbor and Neural Network regression models. We employ hyperparameter training as before, and display the optimal hyperparameters as well as predictive cross validated performance metrics for both models.

K Nearest Neighbor Regression
(neighbors = 3)

R^2	0.9493
MAE	80769.39
MSE	126561893173.77
RMSE	312018.23

Neural Network Regression
(number of hidden layers = 5, nodes in each layer = 5)

R^2	0.9521
MAE	83010.92
MSE	120426132575.03
RMSE	289853.44

Based on our data analysis, the following features likely are important in determining cannabis sales: average retail price, selling Pax filter products, maximum mg THC in any product offered, number of products offered, number of flavored products offered, number of mood specific products offered, average THC across all products.

Discussion

The subset of variables that are correlated with cannabis sales we found are listed above. These results come from an analysis of licensed cannabis sales in the state of California, so the results may not extend out of this region. Due to the average error we find in our models, it means that they may not have a lot of predictive power (the mean absolute error is around $80,000 for every model), but they can provide a good ballpark estimate for sales for a certain input.

Based on the analysis done in this project a number of things can be done to potentially increase sales. Offering CBD based products, Pax filtered products, and increasing the number of flavored and mood specific products are all indicators of increased sales. Increasing the price range of product units, and offering a wider array of products are also indicators of increased cannabis sales according to our analysis. Lowering the average retail price, and lowering the average amount of THC per product also could potentially increase monthly sales based on our data analysis.

For further analysis, cannabis sale data from outside of California should be collected and analyzed to increase the predictive power of models developed, and to allow models developed to work for states outside of California. A more detailed analysis on more detailed product descriptions could also be incorporated into the predictive models we produce.

Conclusion

In this project, we analyzed a dataset of licensed cannabis sales in the state of California over the period of a few years. We merged time series data with more detailed product information to form a dataset that we can train computational models on. We analyze the data by looking at the distribution of variables, as well as other basic statistical measures. We also calculate and extract correlations between all features in order to determine which features are useful in deciding monthly cannabis sales for a given brand or product. Following this, we further augment the dataset and prepare it for model training by scaling and imputing the data. Following this step, we train a Linear Regression, a K-Nearest Neighbor, a Random Forest, and a Neural Network regression model to predict future cannabis sales. These were found to have a strong correlation with sales (as depicted by an R-squared value close to 1), but a high MAE score. Each model is validated using K-Fold cross validation, and optimized with hyperparameter optimization. We implemented principal component analysis on our dataset in order to reduce the data's dimensionality in an attempt to prevent overfitting, but found that the model performance decreased as a result of this.

proj3

December 4, 2021

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

import os
import itertools
import random

import statistics
import seaborn as sns

%matplotlib inline
random.seed(148)
```

[2]:
```
brandAvgRetPrice = pd.read_csv('BrandAverageRetailPrice.csv')
brandDetails = pd.read_csv('BrandDetails.csv')
brandTotalSales = pd.read_csv('BrandTotalSales.csv')
brandTotalUnits = pd.read_csv('BrandTotalUnits.csv')
```

[3]:
```
brandAvgRetPrice.head(10)
```

[3]:

	Brands	Months	ARP	vs. Prior Period
0	#BlackSeries	08/2020	15.684913	NaN
1	#BlackSeries	09/2020	NaN	-1.000000
2	#BlackSeries	01/2021	13.611428	NaN
3	#BlackSeries	02/2021	11.873182	-0.127705
4	#BlackSeries	03/2021	NaN	-1.000000
5	101 Cannabis Co.	11/2019	34.066667	NaN
6	101 Cannabis Co.	12/2019	NaN	-1.000000
7	101 Cannabis Co.	01/2020	34.134929	NaN
8	101 Cannabis Co.	02/2020	29.091388	-0.147753
9	101 Cannabis Co.	03/2020	32.293498	0.110071

[4]:
```
brandAvgRetPrice.describe()
```

[4]:

	ARP	vs. Prior Period
count	25279.000000	24499.000000
mean	22.679732	-0.065028

```
std       19.802724          0.388923
min        0.000000         -1.000000
25%       10.512827         -0.088073
50%       17.033051         -0.011649
75%       31.505612          0.045232
max      700.874984         12.645741
```

[5]:
```
brandAvgRetPrice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27211 entries, 0 to 27210
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Brands            27211 non-null  object
 1   Months            27211 non-null  object
 2   ARP               25279 non-null  float64
 3   vs. Prior Period  24499 non-null  float64
dtypes: float64(2), object(2)
memory usage: 850.5+ KB
```

[6]:
```
brandDetails.head(50)
```

[6]:
```
         State   Channel  Category L1   Category L2           Category L3  \
0   California  Licensed  Inhaleables        Flower                Hybrid
1   California  Licensed  Inhaleables        Flower                Hybrid
2   California  Licensed  Inhaleables        Flower       Sativa Dominant
3   California  Licensed  Inhaleables        Flower       Sativa Dominant
4   California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
5   California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
6   California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
7   California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
8   California  Licensed  Inhaleables    Pre-Rolled     Infused Pre-Rolled
9   California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
10  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
11  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
12  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
13  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
14  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
15  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
16  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
17  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
18  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
19  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
20  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
21  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
22  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
23  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates
```

24	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
25	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
26	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
27	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
28	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
29	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
30	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
31	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
32	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
33	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
34	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
35	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
36	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
37	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
38	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
39	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
40	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
41	California	Licensed	Inhaleables	Pre-Rolled	Infused Pre-Rolled
42	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
43	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
44	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
45	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
46	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
47	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
48	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates
49	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates

	Category L4	Category L5	Brand \
0	NaN	NaN	#BlackSeries
1	NaN	NaN	#BlackSeries
2	NaN	NaN	#BlackSeries
3	NaN	NaN	#BlackSeries
4	Wax	NaN	101 Cannabis Co.
5	Wax	NaN	101 Cannabis Co.
6	Wax	NaN	101 Cannabis Co.
7	Wax	NaN	101 Cannabis Co.
8	NaN	NaN	101 Cannabis Co.
9	Wax	NaN	101 Cannabis Co.
10	Wax	NaN	101 Cannabis Co.
11	Wax	NaN	101 Cannabis Co.
12	Wax	NaN	101 Cannabis Co.
13	Wax	NaN	101 Cannabis Co.
14	Wax	NaN	101 Cannabis Co.
15	Wax	NaN	101 Cannabis Co.
16	Wax	NaN	101 Cannabis Co.
17	Wax	NaN	101 Cannabis Co.
18	Wax	NaN	101 Cannabis Co.

```
19          Wax         NaN  101 Cannabis Co.
20          Wax         NaN  101 Cannabis Co.
21          Wax         NaN  101 Cannabis Co.
22          Wax         NaN  101 Cannabis Co.
23          Wax         NaN  101 Cannabis Co.
24          Wax         NaN  101 Cannabis Co.
25          Wax         NaN  101 Cannabis Co.
26          Wax         NaN  101 Cannabis Co.
27          Wax         NaN  101 Cannabis Co.
28          Wax         NaN  101 Cannabis Co.
29          Wax         NaN  101 Cannabis Co.
30          Wax         NaN  101 Cannabis Co.
31          Wax         NaN  101 Cannabis Co.
32          Wax         NaN  101 Cannabis Co.
33          Wax         NaN  101 Cannabis Co.
34          Wax         NaN  101 Cannabis Co.
35          Wax         NaN  101 Cannabis Co.
36          Wax         NaN  101 Cannabis Co.
37          Wax         NaN  101 Cannabis Co.
38          Wax         NaN  101 Cannabis Co.
39          Wax         NaN  101 Cannabis Co.
40   Live Resin         NaN  101 Cannabis Co.
41          NaN         NaN  101 Cannabis Co.
42          Wax         NaN  101 Cannabis Co.
43          Wax         NaN  101 Cannabis Co.
44          Wax         NaN  101 Cannabis Co.
45          Wax         NaN  101 Cannabis Co.
46          Wax         NaN  101 Cannabis Co.
47          Wax         NaN  101 Cannabis Co.
48          Wax         NaN  101 Cannabis Co.
49          Wax         NaN  101 Cannabis Co.

                                   Product Description      Total Sales ($)  \
0     #BlackSeries - Vanilla Frosting - Flower (Gram)         1,103.964857
1     #BlackSeries - Vanilla Frosting - Flower (Gram)           674.645211
2     #BlackSeries - Blueberry Slushy - Flower (Gram)         2,473.699102
3     #BlackSeries - Blueberry Slushy - Flower (Gram)        14,589.916417
4                101 Cannabis Co. - Afghan Kush - Wax           145.39627
5               101 Cannabis Co. - Skywalker OG - Wax          3,261.12486
6               101 Cannabis Co. - Skywalker OG - Wax         2,062.231412
7       101 Cannabis Co. - Indica Strain Blends - Wax            62.556665
8    101 Cannabis Co. - Hybrid Strain Blends - Infu…        1,309.279796
9                101 Cannabis Co. - Kosher Kush - Wax           556.738062
10               101 Cannabis Co. - Kosher Kush - Wax         1,316.637371
11               101 Cannabis Co. - Kosher Kush - Wax   9,225.549476000000
12               101 Cannabis Co. - Kosher Kush - Wax  3,019.5250380000000
13              101 Cannabis Co. - Blood Orange - Wax           566.293122
```

```
14                     101 Cannabis Co. - 3 Kings - Wax        6,261.743324
15                     101 Cannabis Co. - 3 Kings - Wax  2,883.5042220000000
16                     101 Cannabis Co. - 3 Kings - Wax          815.067402
17                     101 Cannabis Co. - 3 Kings - Wax           407.66515
18                     101 Cannabis Co. - 3 Kings - Wax          582.488434
19                     101 Cannabis Co. - 3 Kings - Wax    826.2600400000000
20                     101 Cannabis Co. - 3 Kings - Wax          105.603128
21                  101 Cannabis Co. - Bubba Kush - Wax           145.39627
22                101 Cannabis Co. - Blood Orange - Wax           358.45266
23                 101 Cannabis Co. - Sour Diesel - Wax          434.970731
24                101 Cannabis Co. - Blood Orange - Wax           86.466443
25                    101 Cannabis Co. - MK Ultra - Wax   377.68802900000000
26                 101 Cannabis Co. - Platinum OG - Wax          105.898451
27                 101 Cannabis Co. - Platinum OG - Wax          101.972163
28            101 Cannabis Co. - Super Silver Haze - Wax   202.22070300000000
29            101 Cannabis Co. - Super Silver Haze - Wax          435.843645
30                 101 Cannabis Co. - Sour Diesel - Wax   371.63322500000000
31                101 Cannabis Co. - Skywalker OG - Wax  1,527.8381580000000
32                 101 Cannabis Co. - Sour Diesel - Wax            108.5442
33                101 Cannabis Co. - Skywalker OG - Wax   389.06757000000000
34            101 Cannabis Co. - Super Silver Haze - Wax        2,414.668334
35               101 Cannabis Co. - Durban Poison - Wax    920.0528850000000
36                    101 Cannabis Co. - Lemonade - Wax   279.18337500000000
37                    101 Cannabis Co. - Lemonade - Wax   7,766.681791000000
38               101 Cannabis Co. - Sundae Driver - Wax          292.619332
39            101 Cannabis Co. - Super Silver Haze - Wax          5,346.86861
40              101 Cannabis Co. - Zookies - Live Resin        1,196.454442
41   101 Cannabis Co. - Hybrid Strain Blends - Infu…          385.053811
42               101 Cannabis Co. - Durban Poison - Wax        2,121.358598
43                     101 Cannabis Co. - Zookies - Wax          329.280835
44                 101 Cannabis Co. - Lemon Skunk - Wax   472.14675800000000
45               101 Cannabis Co. - Durban Poison - Wax          182.056694
46               101 Cannabis Co. - Durban Poison - Wax           65.110869
47                 101 Cannabis Co. - Lemon Skunk - Wax          1,325.24142
48                 101 Cannabis Co. - Lemon Skunk - Wax        8,684.973093
49                 101 Cannabis Co. - Lemon Skunk - Wax        1,683.402394

    … Total THC  Total CBD Contains CBD  Pax Filter               Strain  \
0   …         0          0     THC Only         NaN     Vanilla Frosting
1   …         0          0     THC Only         NaN     Vanilla Frosting
2   …         0          0     THC Only         NaN     Blueberry Slushy
3   …         0          0     THC Only         NaN     Blueberry Slushy
4   …         0          0     THC Only         NaN          Afghan Kush
5   …         0          0     THC Only         NaN         Skywalker OG
6   …         0          0     THC Only         NaN         Skywalker OG
7   …         0          0     THC Only         NaN  Indica Strain Blends
8   …         0          0     THC Only         NaN                  NaN
```

```
9   …          0          0     THC Only          NaN          Kosher Kush
10  …          0          0     THC Only          NaN          Kosher Kush
11  …          0          0     THC Only          NaN          Kosher Kush
12  …          0          0     THC Only          NaN          Kosher Kush
13  …          0          0     THC Only          NaN         Blood Orange
14  …          0          0     THC Only          NaN              3 Kings
15  …          0          0     THC Only          NaN              3 Kings
16  …          0          0     THC Only          NaN              3 Kings
17  …          0          0     THC Only          NaN              3 Kings
18  …          0          0     THC Only          NaN              3 Kings
19  …          0          0     THC Only          NaN              3 Kings
20  …          0          0     THC Only          NaN              3 Kings
21  …          0          0     THC Only          NaN           Bubba Kush
22  …          0          0     THC Only          NaN         Blood Orange
23  …          0          0     THC Only          NaN          Sour Diesel
24  …          0          0     THC Only          NaN         Blood Orange
25  …          0          0     THC Only          NaN             MK Ultra
26  …          0          0     THC Only          NaN          Platinum OG
27  …          0          0     THC Only          NaN          Platinum OG
28  …          0          0     THC Only          NaN    Super Silver Haze
29  …          0          0     THC Only          NaN    Super Silver Haze
30  …          0          0     THC Only          NaN          Sour Diesel
31  …          0          0     THC Only          NaN         Skywalker OG
32  …          0          0     THC Only          NaN          Sour Diesel
33  …          0          0     THC Only          NaN         Skywalker OG
34  …          0          0     THC Only          NaN    Super Silver Haze
35  …          0          0     THC Only          NaN        Durban Poison
36  …          0          0     THC Only          NaN             Lemonade
37  …          0          0     THC Only          NaN             Lemonade
38  …          0          0     THC Only          NaN        Sundae Driver
39  …          0          0     THC Only          NaN    Super Silver Haze
40  …          0          0     THC Only          NaN              Zookies
41  …          0          0     THC Only          NaN                  NaN
42  …          0          0     THC Only          NaN        Durban Poison
43  …          0          0     THC Only          NaN              Zookies
44  …          0          0     THC Only          NaN          Lemon Skunk
45  …          0          0     THC Only          NaN        Durban Poison
46  …          0          0     THC Only          NaN        Durban Poison
47  …          0          0     THC Only          NaN          Lemon Skunk
48  …          0          0     THC Only          NaN          Lemon Skunk
49  …          0          0     THC Only          NaN          Lemon Skunk

   Is Flavored        Mood Effect       Generic Vendor      Generic Items  \
0          NaN  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
1          NaN  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
2          NaN  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
3          NaN  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
```

4	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
5	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
6	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
7	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
8	NaN	Not Mood Specific	Non-Generic Vendors	Generic Items
9	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
10	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
11	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
12	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
13	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
14	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
15	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
16	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
17	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
18	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
19	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
20	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
21	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
22	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
23	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
24	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
25	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
26	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
27	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
28	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
29	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
30	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
31	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
32	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
33	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
34	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
35	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
36	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
37	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
38	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
39	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
40	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
41	NaN	Not Mood Specific	Non-Generic Vendors	Generic Items
42	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
43	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
44	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
45	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
46	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
47	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
48	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items
49	NaN	Not Mood Specific	Non-Generic Vendors	Non-Generic Items

	$5 Price Increment
0	$10.00 to $14.99
1	$15.00 to $19.99
2	$15.00 to $19.99
3	$10.00 to $14.99
4	$35.00 to $39.99
5	$30.00 to $34.99
6	$20.00 to $24.99
7	$10.00 to $14.99
8	$25.00 to $29.99
9	$30.00 to $34.99
10	$45.00 to $49.99
11	$35.00 to $39.99
12	$40.00 to $44.99
13	$25.00 to $29.99
14	$35.00 to $39.99
15	$40.00 to $44.99
16	$30.00 to $34.99
17	$50.00 to $54.99
18	$45.00 to $49.99
19	$25.00 to $29.99
20	$20.00 to $24.99
21	$35.00 to $39.99
22	$30.00 to $34.99
23	$35.00 to $39.99
24	$20.00 to $24.99
25	$45.00 to $49.99
26	$45.00 to $49.99
27	$40.00 to $44.99
28	$20.00 to $24.99
29	$35.00 to $39.99
30	$30.00 to $34.99
31	$25.00 to $29.99
32	$25.00 to $29.99
33	$15.00 to $19.99
34	$25.00 to $29.99
35	$25.00 to $29.99
36	$30.00 to $34.99
37	$35.00 to $39.99
38	$30.00 to $34.99
39	$30.00 to $34.99
40	$25.00 to $29.99
41	$20.00 to $24.99
42	$35.00 to $39.99
43	$35.00 to $39.99
44	$15.00 to $19.99
45	$20.00 to $24.99

```
46   $15.00 to $19.99
47   $20.00 to $24.99
48   $35.00 to $39.99
49   $30.00 to $34.99

[50 rows x 25 columns]
```

[7]:
```
brandDetails.describe()
```

[7]:
```
                 ARP  Items Per Pack
count  144977.000000   144977.000000
mean       30.828439        1.938259
std        19.367580       17.294108
min         0.000000        0.000000
25%        16.407796        0.000000
50%        28.073823        0.000000
75%        41.781699        0.000000
max       874.800010     1000.000000
```

[8]:
```
brandDetails.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144977 entries, 0 to 144976
Data columns (total 25 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   State                144977 non-null  object
 1   Channel              144977 non-null  object
 2   Category L1          144977 non-null  object
 3   Category L2          144977 non-null  object
 4   Category L3          144245 non-null  object
 5   Category L4          102618 non-null  object
 6   Category L5          50135 non-null   object
 7   Brand                144977 non-null  object
 8   Product Description  144977 non-null  object
 9   Total Sales ($)      144977 non-null  object
 10  Total Units          144977 non-null  object
 11  ARP                  144977 non-null  float64
 12  Flavor               7807 non-null    object
 13  Items Per Pack       144977 non-null  int64
 14  Item Weight          64454 non-null   object
 15  Total THC            144977 non-null  object
 16  Total CBD            144977 non-null  object
 17  Contains CBD         144977 non-null  object
 18  Pax Filter           44301 non-null   object
 19  Strain               115639 non-null  object
 20  Is Flavored          11287 non-null   object
 21  Mood Effect          144977 non-null  object
```

```
 22  Generic Vendor       144977 non-null  object
 23  Generic Items        144977 non-null  object
 24  $5 Price Increment   144977 non-null  object
dtypes: float64(1), int64(1), object(23)
memory usage: 27.7+ MB
```

[9]:
```
print(len(brandDetails['Brand'].unique()))
pax_filter_options = brandDetails['Pax Filter'].unique()
print(str(pax_filter_options))
flavor_options = brandDetails['Is Flavored'].unique()
print(str(flavor_options))
mood_options = brandDetails['Mood Effect'].unique()
print(str(mood_options))
```

```
1123
[nan 'Not Pax' 'Pax']
[nan 'Not Flavored' 'Flavored']
['Not Mood Specific' 'Mood Specific']
```

[10]:
```
brandTotalSales.head(10)
```

[10]:

	Months	Brand	Total Sales ($)
0	09/2018	10x Infused	1,711.334232
1	09/2018	1964 Supply Co.	25,475.21594500000
2	09/2018	3 Bros Grow	120,153.644757
3	09/2018	3 Leaf	6,063.5297850000000
4	09/2018	350 Fire	631,510.0481550000
5	09/2018	710 Labs	2,065,970.9803990000
6	09/2018	A&A Craft Inc	5,094.305340000000
7	09/2018	AA Packaging	2,333.3399880000000
8	09/2018	Absolute Xtracts	5,747,227.563172000
9	09/2018	Aces Extracts	155,523.768684

[11]:
```
brandTotalSales.describe()
```

[11]:

	Months	Brand	Total Sales ($)
count	25279	25279	25279
unique	37	1627	25277
top	05/2021	Island Cannabis Company	0
freq	848	37	3

[12]:
```
brandTotalSales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25279 entries, 0 to 25278
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Months           25279 non-null  object
```

```
 1   Brand             25279 non-null  object
 2   Total Sales ($)  25279 non-null  object
dtypes: object(3)
memory usage: 592.6+ KB
```

[13]:
```
brandTotalUnits.head(10)
```

[13]:
```
             Brands   Months          Total Units  vs. Prior Period
0      #BlackSeries  08/2020  1,616.3390040000000               NaN
1      #BlackSeries  09/2020                  NaN         -1.000000
2      #BlackSeries  01/2021    715.5328380000000               NaN
3      #BlackSeries  02/2021           766.669135          0.071466
4      #BlackSeries  03/2021                  NaN         -1.000000
5  101 Cannabis Co.  11/2019            131.06772               NaN
6  101 Cannabis Co.  12/2019                  NaN         -1.000000
7  101 Cannabis Co.  01/2020    345.4134480000000               NaN
8  101 Cannabis Co.  02/2020    696.6584310000000          1.016883
9  101 Cannabis Co.  03/2020    943.3933280000000          0.354169
```

[14]:
```
brandTotalUnits.describe()
```

[14]:
```
       vs. Prior Period
count      24935.000000
mean           0.265306
std            3.291373
min           -1.000000
25%           -0.351822
50%           -0.055216
75%            0.240113
max          250.792020
```

[15]:
```
brandTotalUnits.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27686 entries, 0 to 27685
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Brands            27686 non-null  object
 1   Months            27686 non-null  object
 2   Total Units       25712 non-null  object
 3   vs. Prior Period  24935 non-null  float64
dtypes: float64(1), object(3)
memory usage: 865.3+ KB
```

[16]:
```
brands = brandTotalSales["Brand"].unique()
len(brands)
```

[16]: 1627

[17]:
```
brandTotalSales['Total Sales ($)'] = brandTotalSales['Total Sales ($)'].str.
↪replace(",","")[0:]
brandTotalSales['Total Sales ($)'] = pd.to_numeric(brandTotalSales['Total Sales␣
↪($)'])

brandTotalUnits['Total Units'] = brandTotalUnits['Total Units'].str.
↪replace(",","")[0:]
brandTotalUnits['Total Units'] = pd.to_numeric(brandTotalUnits['Total Units'])

brandDetails['Total THC'] = brandDetails['Total THC'].str.replace(",","")[0:]
brandDetails['Total THC'] = pd.to_numeric(brandDetails['Total THC'])

brandDetails['Total CBD'] = brandDetails['Total CBD'].str.replace(",","")[0:]
brandDetails['Total CBD'] = pd.to_numeric(brandDetails['Total CBD'])

brandTotalSales['Months'] = pd.to_datetime(brandTotalSales['Months'])
brandTotalUnits['Months'] = pd.to_datetime(brandTotalUnits['Months'])
brandAvgRetPrice['Months'] = pd.to_datetime(brandAvgRetPrice['Months'])

brandTotalSales.head(10)

filtered_data = pd.DataFrame()

for brand in brands:

    brand_sales = brandTotalSales[brandTotalSales.Brand == brand]
    brand_sales.loc[:,'Last Month'] = brand_sales.loc[:,'Total Sales ($)'].
↪shift(1)
    brand_sales.loc[:,'3 Month Avg'] = (brand_sales.loc[:,'Total Sales ($)'].
↪shift(1) + brand_sales.loc[:,'Total Sales ($)'].shift(2) + brand_sales.loc[:
↪,'Total Sales ($)'].shift(3))/3
    #brand_sales.loc[:,'12 Month Avg']= (brand_sales.loc[:,'Total Sales ($)'].
↪shift(1) + brand_sales.loc[:,'Total Sales ($)'].shift(2) + brand_sales.loc[:
↪,'Total Sales ($)'].shift(3) + brand_sales.loc[:,'Total Sales ($)'].shift(4)␣
↪+ brand_sales.loc[:,'Total Sales ($)'].shift(5) + brand_sales.loc[:,'Total␣
↪Sales ($)'].shift(6) + brand_sales.loc[:,'Total Sales ($)'].shift(7) +␣
↪brand_sales.loc[:,'Total Sales ($)'].shift(8) + brand_sales.loc[:,'Total␣
↪Sales ($)'].shift(9) + brand_sales.loc[:,'Total Sales ($)'].shift(10) +␣
↪brand_sales.loc[:,'Total Sales ($)'].shift(11) + brand_sales.loc[:,'Total␣
↪Sales ($)'].shift(12))/12
    brand_sales.loc[:,'Last Year'] = (brand_sales.loc[:,'Total Sales ($)'].
↪shift(12))

    brand_units = brandTotalUnits[brandTotalUnits.Brands == brand]
```

```
    merged_data = brand_sales.merge(brand_units, left_on='Months',␣
↪right_on='Months')
    merged_data = merged_data.drop(['Brands'], 1)

    arp_data = brandAvgRetPrice[brandAvgRetPrice.Brands == brand]
    merged_data = merged_data.merge(arp_data, left_on='Months',␣
↪right_on='Months')
    merged_data = merged_data.drop(['Brands'], 1)


    merged_data = merged_data.drop(['vs. Prior Period_x'], 1)
    merged_data = merged_data.drop(['vs. Prior Period_y'], 1)

    merged_data['Month_Num'] = merged_data['Months'].dt.month

    one_hot_encoded_months = pd.get_dummies(merged_data['Month_Num'])

    merged_data = pd.concat([merged_data,one_hot_encoded_months], axis=1)
    merged_data = merged_data.drop(['Month_Num'], 1)

    # Feature Engineering


    brand_details = brandDetails[brandDetails.Brand == brand]

    if len(brand_details) == 0:
        merged_data['Pax_Filter'] = float('NaN')
        merged_data['Inhaleables'] = float('NaN')
        merged_data['Ingestibles'] = float('NaN')
        merged_data['Other_Cannabis'] = float('NaN')
        merged_data['Topicals'] = float('NaN')
        merged_data['Max_THC'] = float('NaN')
        merged_data['Max_CBD'] = float('NaN')
        merged_data['Sell_CBD'] = float('NaN')
        merged_data['Mean_IPP'] = float('NaN')
        merged_data['Product_Count'] = float('NaN')
        merged_data['Price_Range'] = float('NaN')
        merged_data['Flavored_Count'] = float('NaN')
        merged_data['Mood_Count'] = float('NaN')


        filtered_data = filtered_data.append(merged_data)
        continue



    inhaleables_sold = 0
```

```
    ingestibles_sold = 0
    other_sold = 0
    topicals_sold = 0

    if 'Inhaleables' in brand_details['Category L1'].values:
        inhaleables_sold = 1
    if 'Ingestibles' in brand_details['Category L1'].values:
        ingestibles_sold = 1
    if 'Other Cannabis' in brand_details['Category L1'].values:
        other_sold = 1
    if 'Topicals' in brand_details['Category L1'].values:
        topicals_sold = 1
    if 'All accessories' in brand_details['Category L1'].values:
        inhaleables_sold = 1
        ingestibles_sold = 1
        other_sold = 1
        topicals_sold = 1

    pax_filter = float('NaN')
    if 'Pax' in brand_details['Pax Filter'].values:
        pax_filter = 1
    elif 'Not Pax' in brand_details['Pax Filter'].values:
        pax_filter = 0

    merged_data['Pax_Filter'] = pax_filter


    merged_data['Inhaleables'] = inhaleables_sold
    merged_data['Ingestibles'] = ingestibles_sold
    merged_data['Other_Cannabis'] = other_sold
    merged_data['Topicals'] = topicals_sold

    max_product_thc = 0
    max_product_cbd = 0

    avg_product_thc = 0
    avg_product_cbd = 0


    if brand_details['Total THC'].count() != 0:
        max_product_thc = max(brand_details['Total THC'])
        avg_product_thc = statistics.mean(brand_details['Total THC'])

    if brand_details['Total CBD'].count() != 0:
        max_product_cbd = max(brand_details['Total CBD'])
        avg_product_cbd = statistics.mean(brand_details['Total CBD'])
```

```
    contains_cbd = 0

    for x in brand_details['Contains CBD'].values:
        if x != 'THC Only':
            contains_cbd = 1

    merged_data['Max_THC'] = max_product_thc
    merged_data['Max_CBD'] = max_product_cbd
    merged_data['Mean_THC'] = avg_product_thc
    merged_data['Mean_CBD'] = avg_product_cbd

    merged_data['Sell_CBD'] = contains_cbd

    mean_ipp = 1
    if brand_details['Items Per Pack'].count() != 0:
        mean_ipp = statistics.mean(brand_details['Items Per Pack'])

    merged_data['Mean_IPP'] = mean_ipp

    product_count = len(brand_details)

    merged_data['Product_Count'] = product_count

    price_range = len(brand_details['$5 Price Increment'].unique())
    #price_range = len(price_range_set)
    merged_data['Price_Range'] = price_range

    flavored_count = 0
    mood_count = 0

    for x in brand_details['Is Flavored'].values:
        if x == 'Flavored':
            flavored_count += 1
    for x in brand_details['Mood Effect'].values:
        if x == 'Mood Specific':
            mood_count += 1

    merged_data['Flavored_Count'] = flavored_count
    merged_data['Mood_Count'] = mood_count

    filtered_data = filtered_data.append(merged_data)
    #print(str(filtered_data.shape))
```

```
/Users/nbarron/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1597: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = value
/Users/nbarron/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

[18]:
```
filtered_data.head(50)
```

[18]:

	Months	Brand	Total Sales ($)	Last Month	3 Month Avg	\
0	2018-09-01	10x Infused	1711.334232	NaN	NaN	
0	2018-09-01	1964 Supply Co.	25475.215945	NaN	NaN	
1	2018-10-01	1964 Supply Co.	13613.214128	25475.215945	NaN	
2	2018-11-01	1964 Supply Co.	5402.873064	13613.214128	NaN	
3	2018-12-01	1964 Supply Co.	11862.458357	5402.873064	14830.434379	
4	2019-01-01	1964 Supply Co.	3999.035205	11862.458357	10292.848516	
5	2019-02-01	1964 Supply Co.	2417.479974	3999.035205	7088.122209	
6	2019-03-01	1964 Supply Co.	1607.563310	2417.479974	6092.991179	
7	2019-04-01	1964 Supply Co.	292.135879	1607.563310	2674.692830	
0	2018-09-01	3 Bros Grow	120153.644757	NaN	NaN	
1	2018-10-01	3 Bros Grow	112932.164895	120153.644757	NaN	
2	2018-11-01	3 Bros Grow	109432.452831	112932.164895	NaN	
3	2018-12-01	3 Bros Grow	208424.645419	109432.452831	114172.754161	
4	2019-01-01	3 Bros Grow	214650.825825	208424.645419	143596.421048	
5	2019-02-01	3 Bros Grow	557059.818673	214650.825825	177502.641358	
6	2019-03-01	3 Bros Grow	346319.611428	557059.818673	326711.763306	
7	2019-04-01	3 Bros Grow	519579.324303	346319.611428	372676.751975	
8	2019-05-01	3 Bros Grow	278252.378245	519579.324303	474319.584801	
9	2019-06-01	3 Bros Grow	132809.898997	278252.378245	381383.771325	
10	2019-07-01	3 Bros Grow	95303.780457	132809.898997	310213.867182	
11	2019-08-01	3 Bros Grow	120435.066126	95303.780457	168788.685900	
12	2019-09-01	3 Bros Grow	444813.781709	120435.066126	116182.915193	
13	2019-10-01	3 Bros Grow	323920.510121	444813.781709	220184.209431	
14	2019-11-01	3 Bros Grow	163786.306082	323920.510121	296389.785985	
15	2019-12-01	3 Bros Grow	409535.828011	163786.306082	310840.199304	
16	2020-01-01	3 Bros Grow	466658.723817	409535.828011	299080.881405	
17	2020-02-01	3 Bros Grow	227941.631626	466658.723817	346660.285970	
18	2020-03-01	3 Bros Grow	111775.989595	227941.631626	368045.394485	
19	2020-04-01	3 Bros Grow	120002.708661	111775.989595	268792.115013	
20	2020-05-01	3 Bros Grow	211402.393646	120002.708661	153240.109961	
21	2020-06-01	3 Bros Grow	155015.603879	211402.393646	147727.030634	
22	2020-07-01	3 Bros Grow	38776.507573	155015.603879	162140.235395	

```
23 2020-08-01     3 Bros Grow    56271.446989   38776.507573  135064.835033
24 2020-09-01     3 Bros Grow   276099.258355   56271.446989   83354.519480
25 2020-10-01     3 Bros Grow    23070.759975  276099.258355  123715.737639
26 2020-11-01     3 Bros Grow     3073.929764   23070.759975  118480.488440
27 2020-12-01     3 Bros Grow     2446.050652    3073.929764  100747.982698
28 2021-01-01     3 Bros Grow    32326.493793    2446.050652    9530.246797
0  2018-09-01          3 Leaf     6063.529785            NaN            NaN
1  2018-10-01          3 Leaf    27349.643956    6063.529785            NaN
2  2018-11-01          3 Leaf    44414.218569   27349.643956            NaN
3  2018-12-01          3 Leaf    43533.767285   44414.218569   25942.464103
4  2019-01-01          3 Leaf     8971.384508   43533.767285   38432.543270
5  2019-02-01          3 Leaf    12853.649203    8971.384508   32306.456787
6  2019-03-01          3 Leaf    14397.597306   12853.649203   21786.266999
7  2019-04-01          3 Leaf    12897.159109   14397.597306   12074.210339
8  2019-05-01          3 Leaf     9865.215140   12897.159109   13382.801873
9  2019-06-01          3 Leaf    14415.335434    9865.215140   12386.657185
10 2019-07-01          3 Leaf    22075.753500   14415.335434   12392.569894
11 2019-08-01          3 Leaf    18285.969875   22075.753500   15452.101358

        Last Year   Total Units        ARP    9  Pax_Filter  …    4   10  \
0             NaN    142.839336  11.980833  1.0         NaN  …  NaN  NaN
0             NaN   2395.534726  10.634459  1.0         NaN  …  0.0  0.0
1             NaN   1910.329288   7.126109  0.0         NaN  …  0.0  1.0
2             NaN    502.815600  10.745238  0.0         NaN  …  0.0  0.0
3             NaN   2251.347983   5.269047  0.0         NaN  …  0.0  0.0
4             NaN    379.006944  10.551351  0.0         NaN  …  0.0  0.0
5             NaN    225.686193  10.711688  0.0         NaN  …  0.0  0.0
6             NaN    179.313911   8.965079  0.0         NaN  …  0.0  0.0
7             NaN     39.325986   7.428571  0.0         NaN  …  1.0  0.0
0             NaN  10018.989140  11.992592  1.0         NaN  …  0.0  0.0
1             NaN  11214.910342  10.069823  0.0         NaN  …  0.0  1.0
2             NaN  10439.708895  10.482328  0.0         NaN  …  0.0  0.0
3             NaN  24908.365990   8.367656  0.0         NaN  …  0.0  0.0
4             NaN  20666.853248  10.386237  0.0         NaN  …  0.0  0.0
5             NaN  54397.406606  10.240558  0.0         NaN  …  0.0  0.0
6             NaN  25289.238229  13.694347  0.0         NaN  …  0.0  0.0
7             NaN  45758.593710  11.354792  0.0         NaN  …  1.0  0.0
8             NaN  22375.374776  12.435652  0.0         NaN  …  0.0  0.0
9             NaN  10138.122486  13.100049  0.0         NaN  …  0.0  0.0
10            NaN   9047.681531  10.533503  0.0         NaN  …  0.0  0.0
11            NaN   9049.551241  13.308402  0.0         NaN  …  0.0  0.0
12  120153.644757  37202.683417  11.956497  1.0         NaN  …  0.0  0.0
13  112932.164895  26769.936733  12.100160  0.0         NaN  …  0.0  1.0
14  109432.452831  10463.572980  15.653000  0.0         NaN  …  0.0  0.0
15  208424.645419  33470.498126  12.235726  0.0         NaN  …  0.0  0.0
16  214650.825825  24020.626967  19.427416  0.0         NaN  …  0.0  0.0
17  557059.818673  15279.093831  14.918531  0.0         NaN  …  0.0  0.0
```

18	346319.611428	9886.948888	11.305408	0.0	NaN	…	0.0	0.0
19	519579.324303	12702.573524	9.447118	0.0	NaN	…	1.0	0.0
20	278252.378245	18189.435752	11.622262	0.0	NaN	…	0.0	0.0
21	132809.898997	13677.707625	11.333449	0.0	NaN	…	0.0	0.0
22	95303.780457	2636.133742	14.709613	0.0	NaN	…	0.0	0.0
23	120435.066126	3821.810916	14.723765	0.0	NaN	…	0.0	0.0
24	444813.781709	26713.549147	10.335551	1.0	NaN	…	0.0	0.0
25	323920.510121	1789.963356	12.888957	0.0	NaN	…	0.0	1.0
26	163786.306082	67.751596	45.370588	0.0	NaN	…	0.0	0.0
27	409535.828011	267.651935	9.138924	0.0	NaN	…	0.0	0.0
28	466658.723817	2862.131300	11.294553	0.0	NaN	…	0.0	0.0
0	NaN	1101.053215	5.507027	1.0	NaN	…	0.0	0.0
1	NaN	4081.949816	6.700142	0.0	NaN	…	0.0	1.0
2	NaN	6809.559840	6.522333	0.0	NaN	…	0.0	0.0
3	NaN	6833.258003	6.370865	0.0	NaN	…	0.0	0.0
4	NaN	1669.386544	5.374061	0.0	NaN	…	0.0	0.0
5	NaN	2022.382755	6.355696	0.0	NaN	…	0.0	0.0
6	NaN	2755.172420	5.225661	0.0	NaN	…	0.0	0.0
7	NaN	3000.010932	4.299037	0.0	NaN	…	1.0	0.0
8	NaN	2474.059833	3.987460	0.0	NaN	…	0.0	0.0
9	NaN	2571.203232	5.606455	0.0	NaN	…	0.0	0.0
10	NaN	3879.294294	5.690662	0.0	NaN	…	0.0	0.0
11	NaN	3013.540740	6.067935	0.0	NaN	…	0.0	0.0

	11	12	5	6	7	8	Mean_THC	Mean_CBD
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
0	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
1	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
2	1.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
3	0.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN
4	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
5	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
6	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
7	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
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	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	1.0	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
13   0.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
14   1.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
15   0.0  1.0  0.0  0.0  0.0  0.0        0.0        0.0
16   0.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
17   0.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
18   0.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
19   0.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
20   0.0  0.0  1.0  0.0  0.0  0.0        0.0        0.0
21   0.0  0.0  0.0  1.0  0.0  0.0        0.0        0.0
22   0.0  0.0  0.0  0.0  1.0  0.0        0.0        0.0
23   0.0  0.0  0.0  0.0  0.0  1.0        0.0        0.0
24   0.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
25   0.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
26   1.0  0.0  0.0  0.0  0.0  0.0        0.0        0.0
27   0.0  1.0  0.0  0.0  0.0  0.0        0.0        0.0
28   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  0.0        NaN        NaN
1    0.0  0.0  0.0  0.0  0.0  0.0        NaN        NaN
2    1.0  0.0  0.0  0.0  0.0  0.0        NaN        NaN
3    0.0  1.0  0.0  0.0  0.0  0.0        NaN        NaN
4    0.0  0.0  0.0  0.0  0.0  0.0        NaN        NaN
5    0.0  0.0  0.0  0.0  0.0  0.0        NaN        NaN
6    0.0  0.0  0.0  0.0  0.0  0.0        NaN        NaN
7    0.0  0.0  0.0  0.0  0.0  0.0        NaN        NaN
8    0.0  0.0  1.0  0.0  0.0  0.0        NaN        NaN
9    0.0  0.0  0.0  1.0  0.0  0.0        NaN        NaN
10   0.0  0.0  0.0  0.0  1.0  0.0        NaN        NaN
11   0.0  0.0  0.0  0.0  0.0  1.0        NaN        NaN

[50 rows x 35 columns]
```

[19]:
```
print(filtered_data.columns.tolist())
```

```
['Months', 'Brand', 'Total Sales ($)', 'Last Month', '3 Month Avg', 'Last Year',
'Total Units', 'ARP', 9, 'Pax_Filter', 'Inhaleables', 'Ingestibles',
'Other_Cannabis', 'Topicals', 'Max_THC', 'Max_CBD', 'Sell_CBD', 'Mean_IPP',
'Product_Count', 'Price_Range', 'Flavored_Count', 'Mood_Count', 1, 2, 3, 4, 10,
11, 12, 5, 6, 7, 8, 'Mean_THC', 'Mean_CBD']
```

[20]:
```
filtered_data['1'] = filtered_data[1].fillna(0)
filtered_data['2'] = filtered_data[2].fillna(0)
filtered_data['3'] = filtered_data[3].fillna(0)
filtered_data['4'] = filtered_data[4].fillna(0)
filtered_data['5'] = filtered_data[5].fillna(0)
filtered_data['6'] = filtered_data[6].fillna(0)
filtered_data['7'] = filtered_data[7].fillna(0)
filtered_data['8'] = filtered_data[8].fillna(0)
filtered_data['9'] = filtered_data[9].fillna(0)
```

```
filtered_data['10'] = filtered_data[10].fillna(0)
filtered_data['11'] = filtered_data[11].fillna(0)
filtered_data['12'] = filtered_data[12].fillna(0)

filtered_data = filtered_data.drop([1], 1)
filtered_data = filtered_data.drop([2], 1)
filtered_data = filtered_data.drop([3], 1)
filtered_data = filtered_data.drop([4], 1)
filtered_data = filtered_data.drop([5], 1)
filtered_data = filtered_data.drop([6], 1)
filtered_data = filtered_data.drop([7], 1)
filtered_data = filtered_data.drop([8], 1)
filtered_data = filtered_data.drop([9], 1)
filtered_data = filtered_data.drop([10], 1)
filtered_data = filtered_data.drop([11], 1)
filtered_data = filtered_data.drop([12], 1)
```

[21]:
```
filtered_data.head(50)
```

[21]:
```
        Months            Brand  Total Sales ($)     Last Month    3 Month Avg  \
0   2018-09-01      10x Infused      1711.334232            NaN            NaN
0   2018-09-01  1964 Supply Co.     25475.215945            NaN            NaN
1   2018-10-01  1964 Supply Co.     13613.214128   25475.215945            NaN
2   2018-11-01  1964 Supply Co.      5402.873064   13613.214128            NaN
3   2018-12-01  1964 Supply Co.     11862.458357    5402.873064   14830.434379
4   2019-01-01  1964 Supply Co.      3999.035205   11862.458357   10292.848516
5   2019-02-01  1964 Supply Co.      2417.479974    3999.035205    7088.122209
6   2019-03-01  1964 Supply Co.      1607.563310    2417.479974    6092.991179
7   2019-04-01  1964 Supply Co.       292.135879    1607.563310    2674.692830
0   2018-09-01      3 Bros Grow    120153.644757            NaN            NaN
1   2018-10-01      3 Bros Grow    112932.164895  120153.644757            NaN
2   2018-11-01      3 Bros Grow    109432.452831  112932.164895            NaN
3   2018-12-01      3 Bros Grow    208424.645419  109432.452831  114172.754161
4   2019-01-01      3 Bros Grow    214650.825825  208424.645419  143596.421048
5   2019-02-01      3 Bros Grow    557059.818673  214650.825825  177502.641358
6   2019-03-01      3 Bros Grow    346319.611428  557059.818673  326711.763306
7   2019-04-01      3 Bros Grow    519579.324303  346319.611428  372676.751975
8   2019-05-01      3 Bros Grow    278252.378245  519579.324303  474319.584801
9   2019-06-01      3 Bros Grow    132809.898997  278252.378245  381383.771325
10  2019-07-01      3 Bros Grow     95303.780457  132809.898997  310213.867182
11  2019-08-01      3 Bros Grow    120435.066126   95303.780457  168788.685900
12  2019-09-01      3 Bros Grow    444813.781709  120435.066126  116182.915193
13  2019-10-01      3 Bros Grow    323920.510121  444813.781709  220184.209431
14  2019-11-01      3 Bros Grow    163786.306082  323920.510121  296389.785985
15  2019-12-01      3 Bros Grow    409535.828011  163786.306082  310840.199304
16  2020-01-01      3 Bros Grow    466658.723817  409535.828011  299080.881405
17  2020-02-01      3 Bros Grow    227941.631626  466658.723817  346660.285970
```

```
18 2020-03-01      3 Bros Grow   111775.989595  227941.631626  368045.394485
19 2020-04-01      3 Bros Grow   120002.708661  111775.989595  268792.115013
20 2020-05-01      3 Bros Grow   211402.393646  120002.708661  153240.109961
21 2020-06-01      3 Bros Grow   155015.603879  211402.393646  147727.030634
22 2020-07-01      3 Bros Grow    38776.507573  155015.603879  162140.235395
23 2020-08-01      3 Bros Grow    56271.446989   38776.507573  135064.835033
24 2020-09-01      3 Bros Grow   276099.258355   56271.446989   83354.519480
25 2020-10-01      3 Bros Grow    23070.759975  276099.258355  123715.737639
26 2020-11-01      3 Bros Grow     3073.929764   23070.759975  118480.488440
27 2020-12-01      3 Bros Grow     2446.050652    3073.929764  100747.982698
28 2021-01-01      3 Bros Grow    32326.493793    2446.050652    9530.246797
0  2018-09-01           3 Leaf     6063.529785            NaN            NaN
1  2018-10-01           3 Leaf    27349.643956    6063.529785            NaN
2  2018-11-01           3 Leaf    44414.218569   27349.643956            NaN
3  2018-12-01           3 Leaf    43533.767285   44414.218569   25942.464103
4  2019-01-01           3 Leaf     8971.384508   43533.767285   38432.543270
5  2019-02-01           3 Leaf    12853.649203    8971.384508   32306.456787
6  2019-03-01           3 Leaf    14397.597306   12853.649203   21786.266999
7  2019-04-01           3 Leaf    12897.159109   14397.597306   12074.210339
8  2019-05-01           3 Leaf     9865.215140   12897.159109   13382.801873
9  2019-06-01           3 Leaf    14415.335434    9865.215140   12386.657185
10 2019-07-01           3 Leaf    22075.753500   14415.335434   12392.569894
11 2019-08-01           3 Leaf    18285.969875   22075.753500   15452.101358

        Last Year   Total Units        ARP  Pax_Filter  Inhaleables  …    3  \
0             NaN    142.839336  11.980833         NaN          NaN  …  0.0
0             NaN   2395.534726  10.634459         NaN          NaN  …  0.0
1             NaN   1910.329288   7.126109         NaN          NaN  …  0.0
2             NaN    502.815600  10.745238         NaN          NaN  …  0.0
3             NaN   2251.347983   5.269047         NaN          NaN  …  0.0
4             NaN    379.006944  10.551351         NaN          NaN  …  0.0
5             NaN    225.686193  10.711688         NaN          NaN  …  0.0
6             NaN    179.313911   8.965079         NaN          NaN  …  1.0
7             NaN     39.325986   7.428571         NaN          NaN  …  0.0
0             NaN  10018.989140  11.992592         NaN          1.0  …  0.0
1             NaN  11214.910342  10.069823         NaN          1.0  …  0.0
2             NaN  10439.708895  10.482328         NaN          1.0  …  0.0
3             NaN  24908.365990   8.367656         NaN          1.0  …  0.0
4             NaN  20666.853248  10.386237         NaN          1.0  …  0.0
5             NaN  54397.406606  10.240558         NaN          1.0  …  0.0
6             NaN  25289.238229  13.694347         NaN          1.0  …  1.0
7             NaN  45758.593710  11.354792         NaN          1.0  …  0.0
8             NaN  22375.374776  12.435652         NaN          1.0  …  0.0
9             NaN  10138.122486  13.100049         NaN          1.0  …  0.0
10            NaN   9047.681531  10.533503         NaN          1.0  …  0.0
11            NaN   9049.551241  13.308402         NaN          1.0  …  0.0
12  120153.644757  37202.683417  11.956497         NaN          1.0  …  0.0
```

13	112932.164895	26769.936733	12.100160	NaN	1.0	…	0.0
14	109432.452831	10463.572980	15.653000	NaN	1.0	…	0.0
15	208424.645419	33470.498126	12.235726	NaN	1.0	…	0.0
16	214650.825825	24020.626967	19.427416	NaN	1.0	…	0.0
17	557059.818673	15279.093831	14.918531	NaN	1.0	…	0.0
18	346319.611428	9886.948888	11.305408	NaN	1.0	…	1.0
19	519579.324303	12702.573524	9.447118	NaN	1.0	…	0.0
20	278252.378245	18189.435752	11.622262	NaN	1.0	…	0.0
21	132809.898997	13677.707625	11.333449	NaN	1.0	…	0.0
22	95303.780457	2636.133742	14.709613	NaN	1.0	…	0.0
23	120435.066126	3821.810916	14.723765	NaN	1.0	…	0.0
24	444813.781709	26713.549147	10.335551	NaN	1.0	…	0.0
25	323920.510121	1789.963356	12.888957	NaN	1.0	…	0.0
26	163786.306082	67.751596	45.370588	NaN	1.0	…	0.0
27	409535.828011	267.651935	9.138924	NaN	1.0	…	0.0
28	466658.723817	2862.131300	11.294553	NaN	1.0	…	0.0
0	NaN	1101.053215	5.507027	NaN	NaN	…	0.0
1	NaN	4081.949816	6.700142	NaN	NaN	…	0.0
2	NaN	6809.559840	6.522333	NaN	NaN	…	0.0
3	NaN	6833.258003	6.370865	NaN	NaN	…	0.0
4	NaN	1669.386544	5.374061	NaN	NaN	…	0.0
5	NaN	2022.382755	6.355696	NaN	NaN	…	0.0
6	NaN	2755.172420	5.225661	NaN	NaN	…	1.0
7	NaN	3000.010932	4.299037	NaN	NaN	…	0.0
8	NaN	2474.059833	3.987460	NaN	NaN	…	0.0
9	NaN	2571.203232	5.606455	NaN	NaN	…	0.0
10	NaN	3879.294294	5.690662	NaN	NaN	…	0.0
11	NaN	3013.540740	6.067935	NaN	NaN	…	0.0

	4	5	6	7	8	9	10	11	12
0	0.0	0.0	0.0	0.0	0.0	1.0	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
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	1.0	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	1.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
8    0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
9    0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
10   0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
11   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0
12   0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0
13   0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
14   0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
15   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
16   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
17   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
18   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
19   1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
20   0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
21   0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
22   0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
23   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0
24   0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0
25   0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
26   0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
27   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
28   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  0.0  1.0  0.0  0.0  0.0
1    0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
2    0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
3    0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
4    0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
5    0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
6    0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
7    1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
8    0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
9    0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
10   0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
11   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0

[50 rows x 35 columns]
```

[22]:
```
# Data Statistics

print("Average Monthly Sales: $" + str(statistics.mean(filtered_data['Total␣
↪Sales ($)'])))
print("Monthly Sales Standard Deviation: $" + str(statistics.
↪stdev(filtered_data['Total Sales ($)'])))

print("Average Monthly Units Sold: " + str(statistics.mean(filtered_data['Total␣
↪Units'])))
print("Monthly Units Sold Standard Deviation: " + str(statistics.
↪stdev(filtered_data['Total Units'])))
```

```python
print("Average Monthly ARP: " + str(statistics.mean(filtered_data['ARP'])))
print("Monthly ARP Standard Deviation: " + str(statistics.
 ↪stdev(filtered_data['ARP'])))

filtered_data.describe()
```

```
Average Monthly Sales: $409372.85619946336
Monthly Sales Standard Deviation: $1596024.283035418
Average Monthly Units Sold: 28862.10067850273
Monthly Units Sold Standard Deviation: 161715.5821856867
Average Monthly ARP: 22.679731745813
Monthly ARP Standard Deviation: 19.802723938896023
```

[22]:

	Total Sales ($)	Last Month	3 Month Avg	Last Year	\
count	2.527900e+04	2.365200e+04	2.073400e+04	1.142400e+04	
mean	4.093729e+05	4.245507e+05	4.551029e+05	5.516544e+05	
std	1.596024e+06	1.625582e+06	1.669072e+06	1.877350e+06	
min	0.000000e+00	0.000000e+00	6.011905e+01	0.000000e+00	
25%	1.390320e+04	1.608221e+04	2.249319e+04	3.295066e+04	
50%	6.210080e+04	6.905932e+04	8.316126e+04	1.227479e+05	
75%	2.473270e+05	2.627836e+05	2.920007e+05	3.909989e+05	
max	4.036351e+07	4.036351e+07	3.737876e+07	4.036351e+07	

	Total Units	ARP	Pax_Filter	Inhaleables	Ingestibles	\
count	2.527900e+04	25279.000000	5609.000000	21472.000000	21472.000000	
mean	2.886210e+04	22.679732	0.149224	0.705663	0.310404	
std	1.617156e+05	19.802724	0.356341	0.455755	0.462670	
min	3.842953e+00	0.000000	0.000000	0.000000	0.000000	
25%	7.169135e+02	10.512827	0.000000	0.000000	0.000000	
50%	3.605059e+03	17.033051	0.000000	1.000000	0.000000	
75%	1.564044e+04	31.505612	0.000000	1.000000	1.000000	
max	5.248082e+06	700.874984	1.000000	1.000000	1.000000	

	Other_Cannabis	…	3	4	5	\
count	21472.000000	…	25279.000000	25279.000000	25279.000000	
mean	0.059845	…	0.080660	0.082361	0.085802	
std	0.237206	…	0.272317	0.274919	0.280078	
min	0.000000	…	0.000000	0.000000	0.000000	
25%	0.000000	…	0.000000	0.000000	0.000000	
50%	0.000000	…	0.000000	0.000000	0.000000	
75%	0.000000	…	0.000000	0.000000	0.000000	
max	1.000000	…	1.000000	1.000000	1.000000	

	6	7	8	9	10	\
count	25279.000000	25279.000000	25279.000000	25279.000000	25279.000000	
mean	0.087147	0.086752	0.087701	0.103841	0.073856	

```
std         0.282057      0.281477      0.282866      0.305060      0.261541
min         0.000000      0.000000      0.000000      0.000000      0.000000
25%         0.000000      0.000000      0.000000      0.000000      0.000000
50%         0.000000      0.000000      0.000000      0.000000      0.000000
75%         0.000000      0.000000      0.000000      0.000000      0.000000
max         1.000000      1.000000      1.000000      1.000000      1.000000

                 11            12
count  25279.000000  25279.000000
mean       0.076981      0.077812
std        0.266566      0.267880
min        0.000000      0.000000
25%        0.000000      0.000000
50%        0.000000      0.000000
75%        0.000000      0.000000
max        1.000000      1.000000

[8 rows x 33 columns]
```

[23]:
```
half_one = filtered_data.iloc[:,[0,1,2,3,4,5,6,7,9,10,11,12,13,14,15,16]]
half_one.describe()
```

[23]:
```
       Total Sales ($)    Last Month   3 Month Avg     Last Year  \
count     2.527900e+04  2.365200e+04  2.073400e+04  1.142400e+04
mean      4.093729e+05  4.245507e+05  4.551029e+05  5.516544e+05
std       1.596024e+06  1.625582e+06  1.669072e+06  1.877350e+06
min       0.000000e+00  0.000000e+00  6.011905e+01  0.000000e+00
25%       1.390320e+04  1.608221e+04  2.249319e+04  3.295066e+04
50%       6.210080e+04  6.905932e+04  8.316126e+04  1.227479e+05
75%       2.473270e+05  2.627836e+05  2.920007e+05  3.909989e+05
max       4.036351e+07  4.036351e+07  3.737876e+07  4.036351e+07

        Total Units           ARP   Inhaleables   Ingestibles  Other_Cannabis  \
count  2.527900e+04  25279.000000  21472.000000  21472.000000    21472.000000
mean   2.886210e+04     22.679732      0.705663      0.310404        0.059845
std    1.617156e+05     19.802724      0.455755      0.462670        0.237206
min    3.842953e+00      0.000000      0.000000      0.000000        0.000000
25%    7.169135e+02     10.512827      0.000000      0.000000        0.000000
50%    3.605059e+03     17.033051      1.000000      0.000000        0.000000
75%    1.564044e+04     31.505612      1.000000      1.000000        0.000000
max    5.248082e+06    700.874984      1.000000      1.000000        1.000000

           Topicals       Max_THC       Max_CBD      Sell_CBD      Mean_IPP
count  21472.000000  21472.000000  21472.000000  21472.000000  21472.000000
mean       0.093564    110.025079     99.128748      0.379238      2.697393
std        0.291228    287.258462    483.069571      0.485209      8.084836
min        0.000000      0.000000      0.000000      0.000000      0.000000
```

```
25%         0.000000     0.000000      0.000000     0.000000     0.000000
50%         0.000000     0.000000      0.000000     0.000000     0.253623
75%         0.000000   100.000000      0.000000     1.000000     2.130435
max         1.000000  2300.000000  10000.000000     1.000000   198.931250
```

[24]:
```
half_two = filtered_data.iloc[:
↪,[17,18,19,20,21,22,23,24,25,26,8,27,28,29,30,31,32]]
half_two.describe()
```

[24]:
```
       Product_Count   Price_Range  Flavored_Count    Mood_Count  \
count   21472.000000  21472.000000    21472.000000  21472.000000
mean      210.689316     10.781762       10.832340     10.568927
std       516.966345      5.648736       39.652441     48.626711
min         1.000000      1.000000        0.000000      0.000000
25%        21.000000      6.000000        0.000000      0.000000
50%        68.000000     10.000000        0.000000      0.000000
75%       188.000000     15.000000        0.000000      0.000000
max      9004.000000     22.000000      640.000000    672.000000

           Mean_THC      Mean_CBD             1             2             3  \
count  21472.000000  21472.000000  25279.000000  25279.000000  25279.000000
mean      29.262824     13.666658      0.077495      0.079592      0.080660
std       74.689194     43.702023      0.267381      0.270665      0.272317
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%       19.977802      0.000000      0.000000      0.000000      0.000000
max     1000.000000    425.581395      1.000000      1.000000      1.000000

                  4   Pax_Filter             5             6             7  \
count  25279.000000  5609.000000  25279.000000  25279.000000  25279.000000
mean       0.082361     0.149224      0.085802      0.087147      0.086752
std        0.274919     0.356341      0.280078      0.282057      0.281477
min        0.000000     0.000000      0.000000      0.000000      0.000000
25%        0.000000     0.000000      0.000000      0.000000      0.000000
50%        0.000000     0.000000      0.000000      0.000000      0.000000
75%        0.000000     0.000000      0.000000      0.000000      0.000000
max        1.000000     1.000000      1.000000      1.000000      1.000000

                  8             9            10
count  25279.000000  25279.000000  25279.000000
mean       0.087701      0.103841      0.073856
std        0.282866      0.305060      0.261541
min        0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000
```

```
max        1.000000      1.000000      1.000000
```

[25]:
```
filtered_data.info()
```

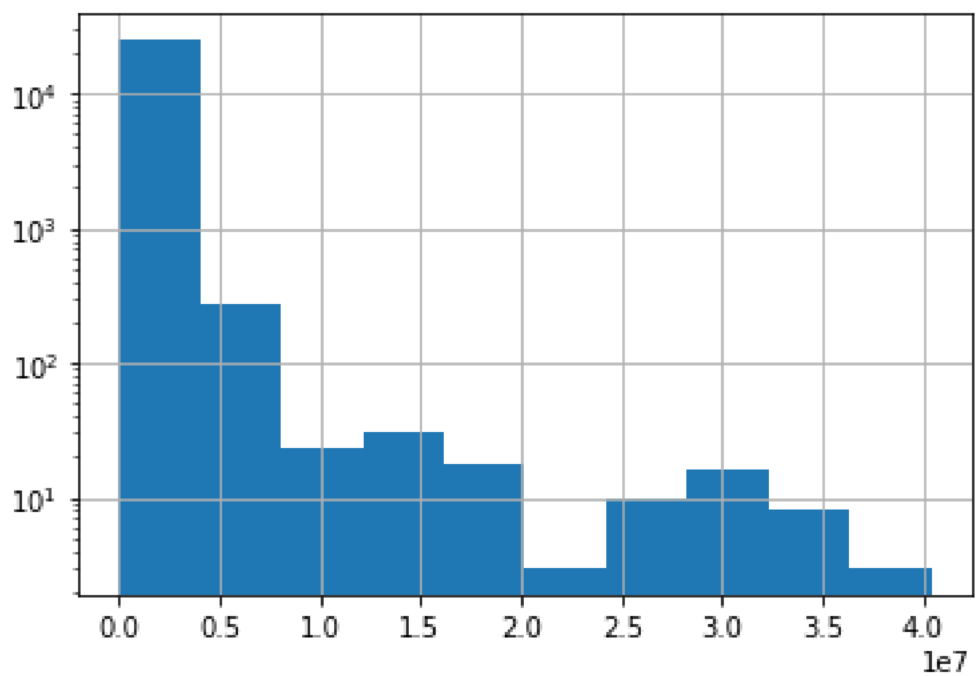
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25279 entries, 0 to 0
Data columns (total 35 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Months           25279 non-null  datetime64[ns]
 1   Brand            25279 non-null  object
 2   Total Sales ($)  25279 non-null  float64
 3   Last Month       23652 non-null  float64
 4   3 Month Avg      20734 non-null  float64
 5   Last Year        11424 non-null  float64
 6   Total Units      25279 non-null  float64
 7   ARP              25279 non-null  float64
 8   Pax_Filter       5609 non-null   float64
 9   Inhaleables      21472 non-null  float64
 10  Ingestibles      21472 non-null  float64
 11  Other_Cannabis   21472 non-null  float64
 12  Topicals         21472 non-null  float64
 13  Max_THC          21472 non-null  float64
 14  Max_CBD          21472 non-null  float64
 15  Sell_CBD         21472 non-null  float64
 16  Mean_IPP         21472 non-null  float64
 17  Product_Count    21472 non-null  float64
 18  Price_Range      21472 non-null  float64
 19  Flavored_Count   21472 non-null  float64
 20  Mood_Count       21472 non-null  float64
 21  Mean_THC         21472 non-null  float64
 22  Mean_CBD         21472 non-null  float64
 23  1                25279 non-null  float64
 24  2                25279 non-null  float64
 25  3                25279 non-null  float64
 26  4                25279 non-null  float64
 27  5                25279 non-null  float64
 28  6                25279 non-null  float64
 29  7                25279 non-null  float64
 30  8                25279 non-null  float64
 31  9                25279 non-null  float64
 32  10               25279 non-null  float64
 33  11               25279 non-null  float64
 34  12               25279 non-null  float64
dtypes: datetime64[ns](1), float64(33), object(1)
memory usage: 6.9+ MB
```

[26]:

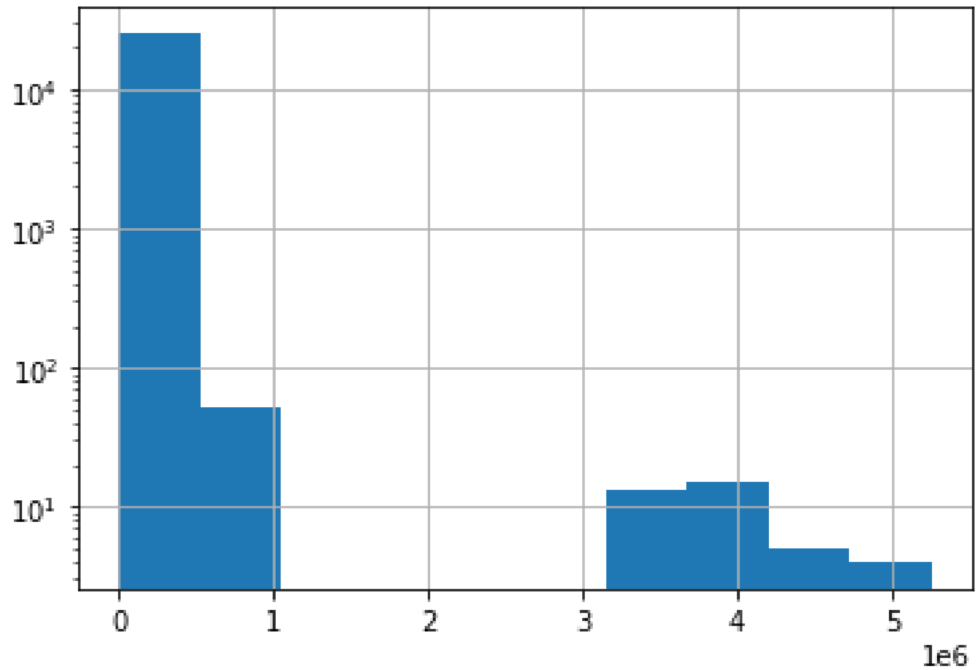
```
# Feature Histograms

print("Total Monthly Sales ($) Histogram")
filtered_data['Total Sales ($)'].hist()
plt.yscale('log')
plt.show()

print("Total Monthly Units Histogram")
filtered_data['Total Units'].hist()
plt.yscale('log')
plt.show()

print("ARP Histogram")
filtered_data['ARP'].hist()
plt.yscale('log')
plt.show()

print("Pax Filter Histogram")
filtered_data['Pax_Filter'].hist()
plt.yscale('linear')
plt.show()

print("Inhaleables Sold Histogram")
filtered_data['Inhaleables'].hist()
plt.yscale('linear')
plt.show()

print("Ingenstibles Sold Histogram")
filtered_data['Ingestibles'].hist()
plt.yscale('linear')
plt.show()

print("Other Cannabis Product Sold Histogram")
filtered_data['Other_Cannabis'].hist()
plt.yscale('linear')
plt.show()

print("Topicals Sold Histogram")
filtered_data['Topicals'].hist()
plt.yscale('linear')
plt.show()

print("Max mg THC Histogram")
filtered_data['Max_THC'].hist()
plt.yscale('log')
plt.show()
```

```
print("Max mg CBD Histogram")
filtered_data['Max_CBD'].hist()
plt.yscale('log')
plt.show()

print("Product Count Histogram")
filtered_data['Product_Count'].hist()
plt.yscale('log')
plt.show()

print("Price Range Histogram")
filtered_data['Price_Range'].hist()
plt.yscale('linear')
plt.show()

print("Flavored Count Histogram")
filtered_data['Flavored_Count'].hist()
plt.yscale('log')
plt.show()

print("Mood Count Histogram")
filtered_data['Mood_Count'].hist()
plt.yscale('log')
plt.show()
```

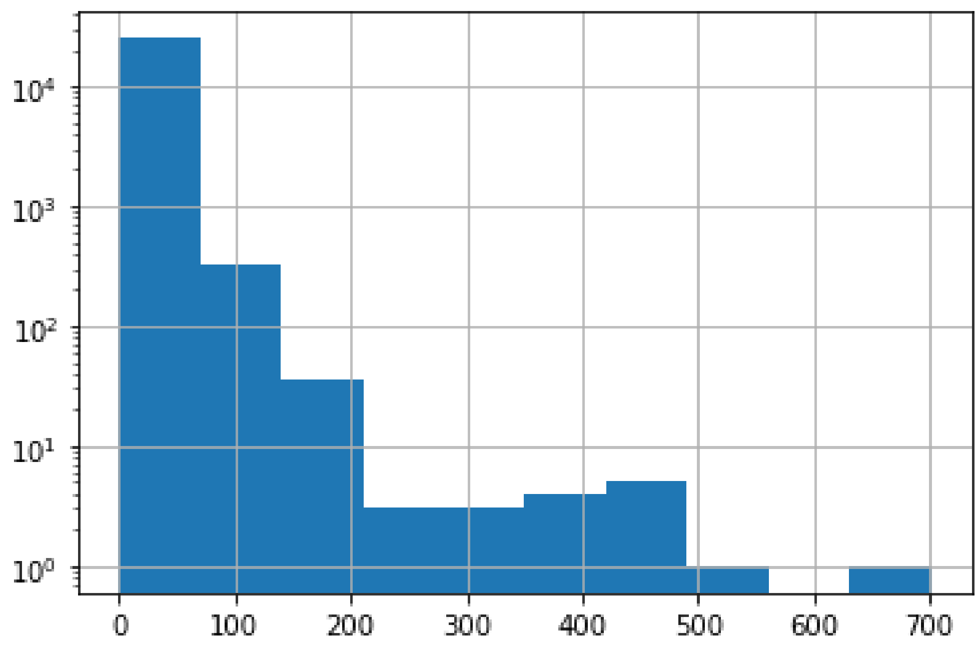
```
Total Monthly Sales ($) Histogram
```



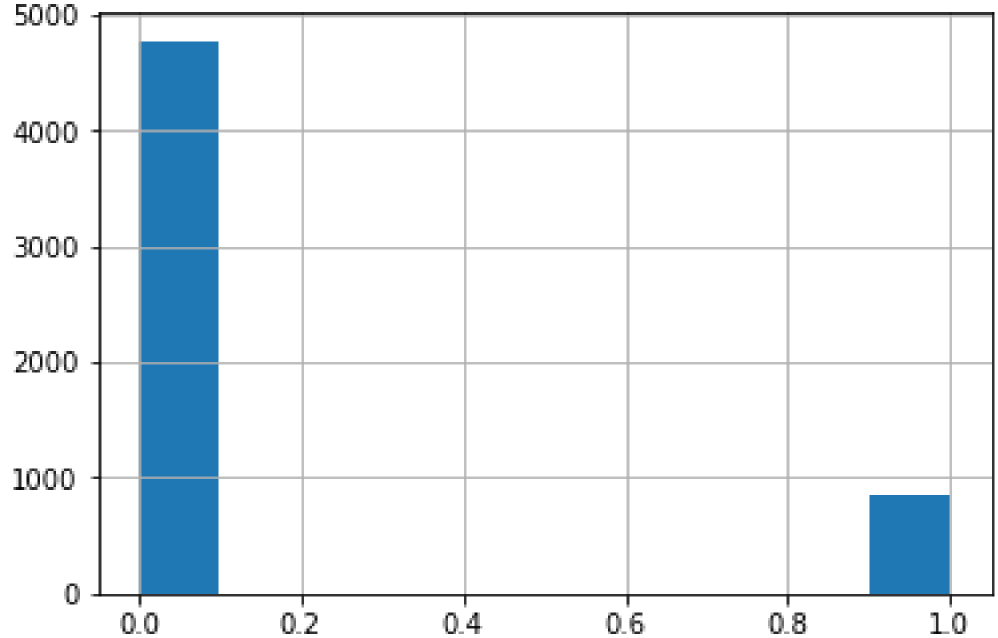
Total Monthly Units Histogram



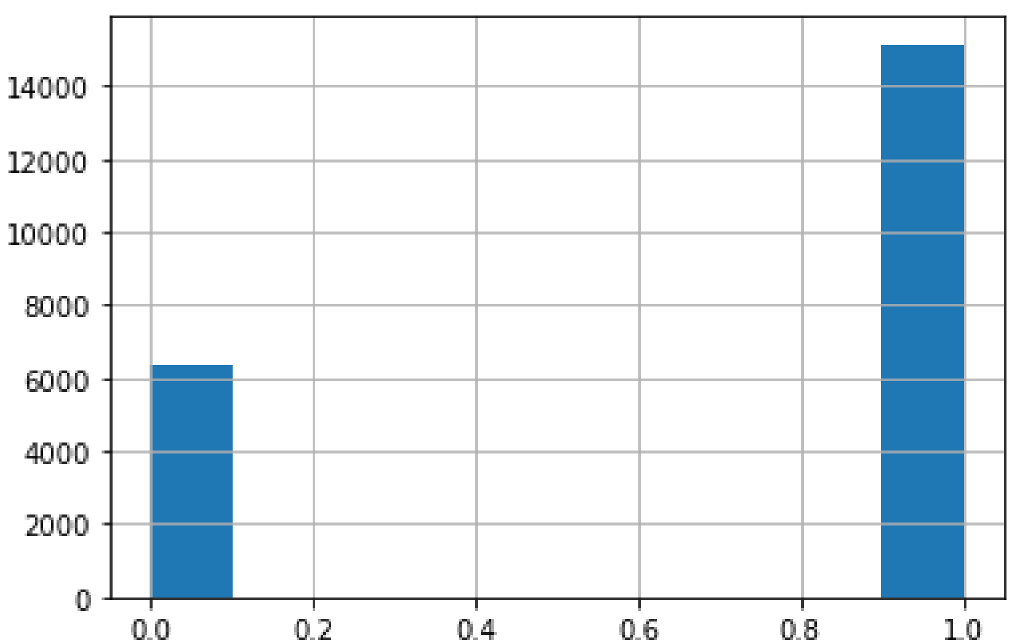
ARP Histogram



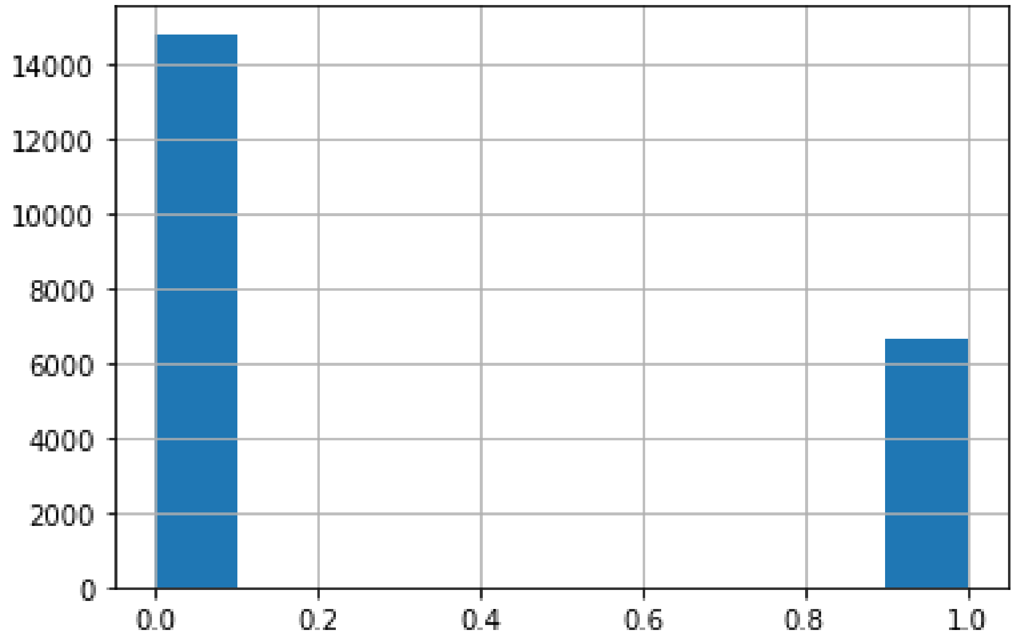
Pax Filter Histogram



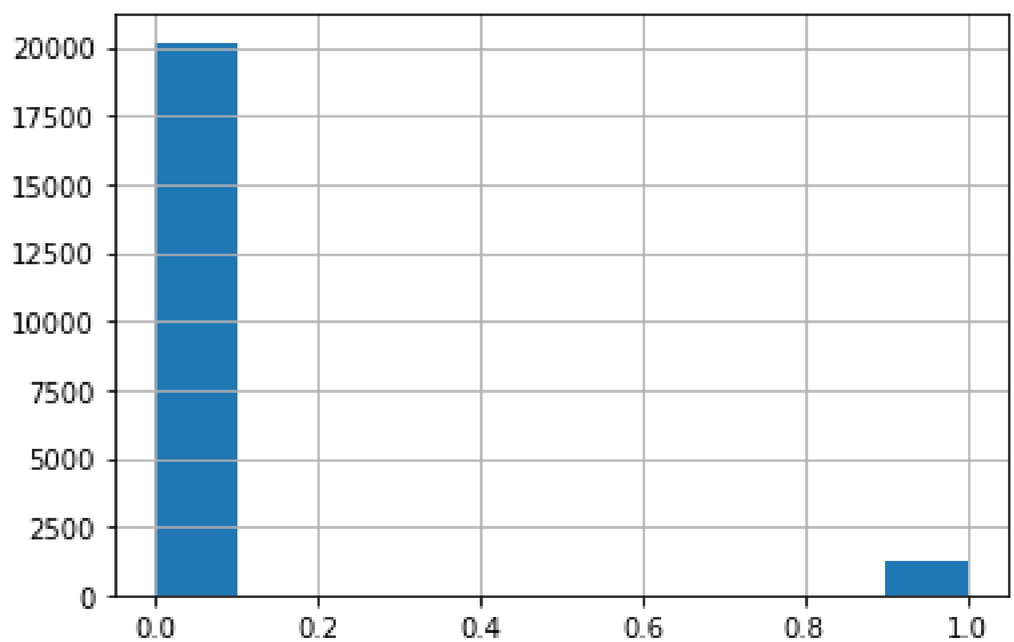
Inhaleables Sold Histogram



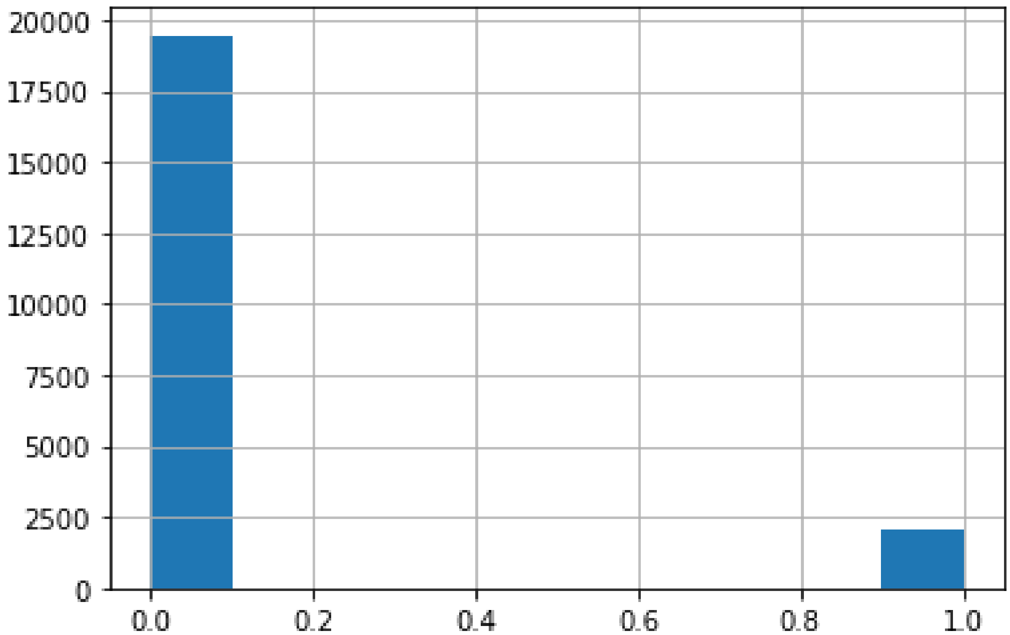
Ingenstibles Sold Histogram



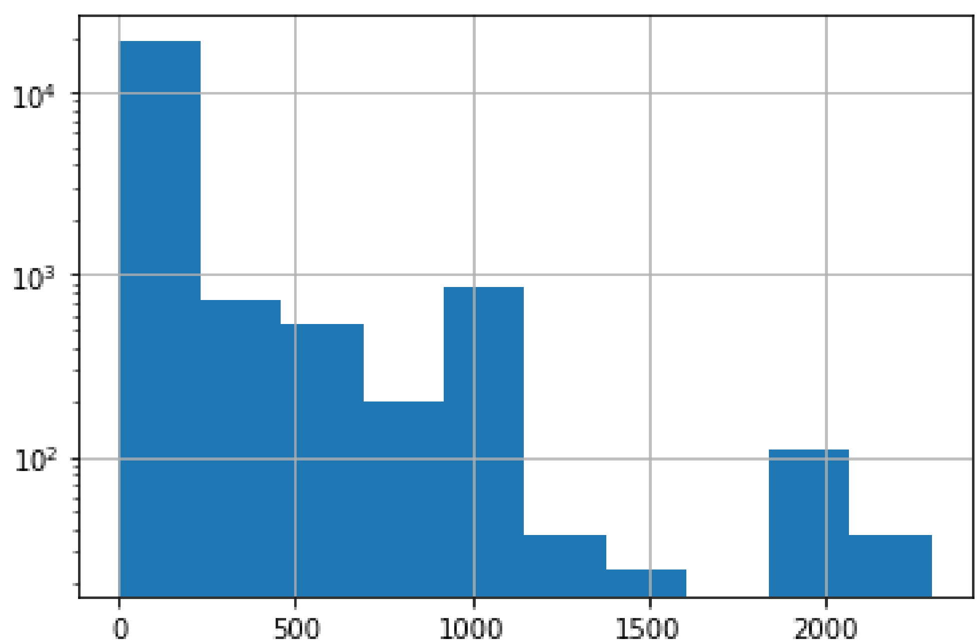
Other Cannabis Product Sold Histogram



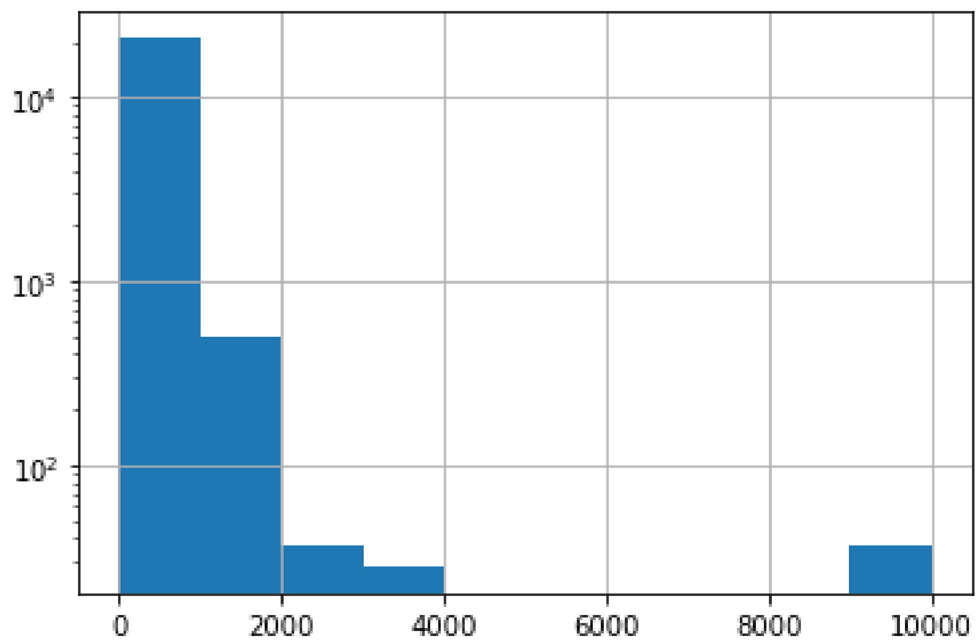
Topicals Sold Histogram



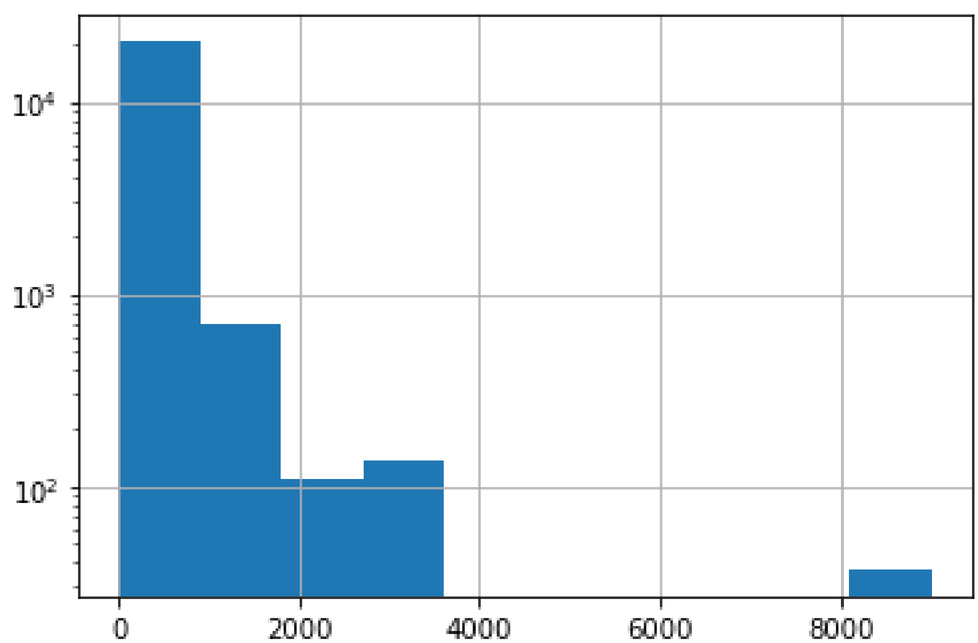
Max mg THC Histogram



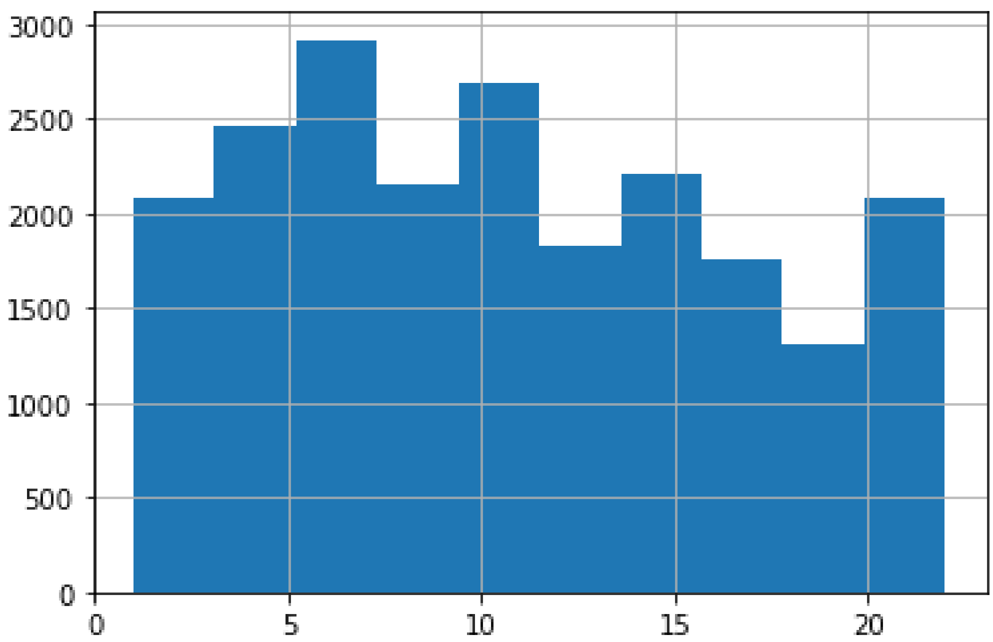
Max mg CBD Histogram



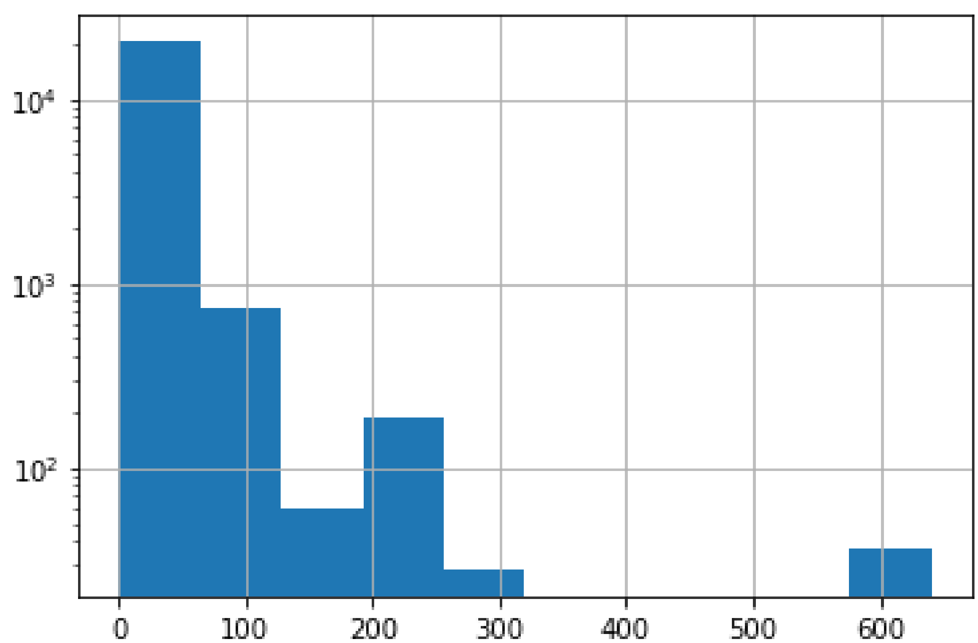
Product Count Histogram



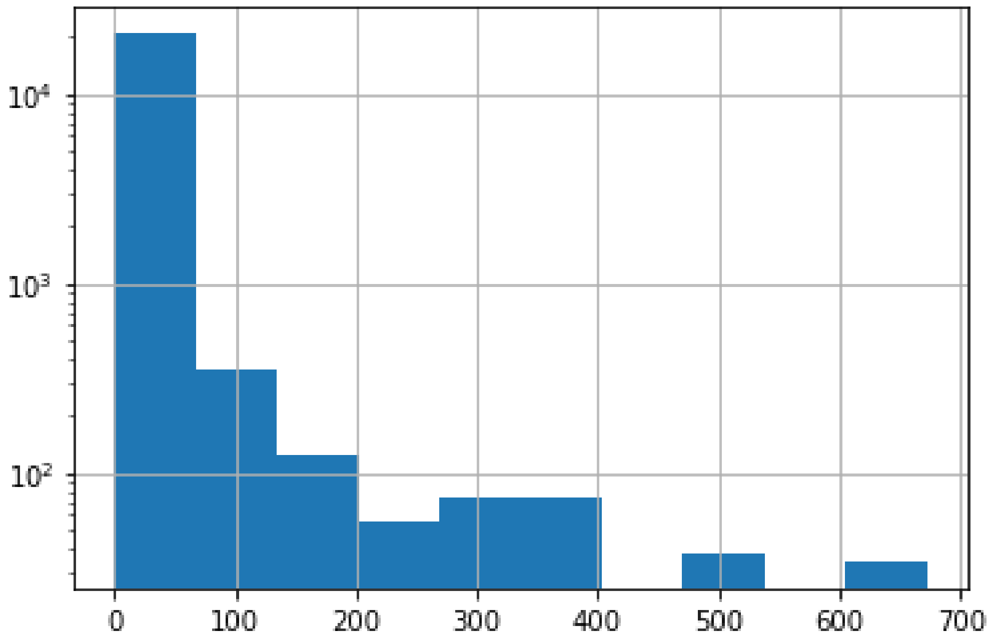
Price Range Histogram



Flavored Count Histogram

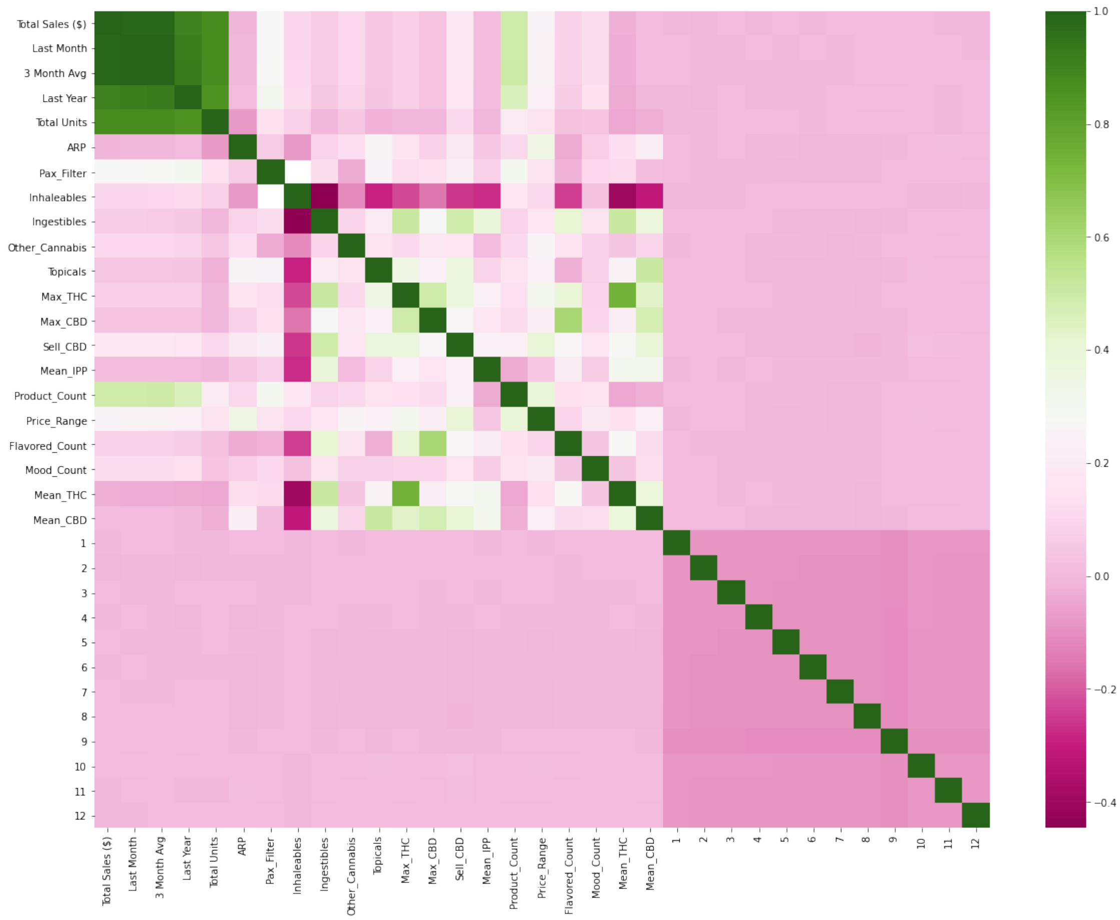


Mood Count Histogram



[27]:
```
plt.subplots(figsize=(20,15))
sns.heatmap(filtered_data.corr(), cmap="PiYG")
```

[27]: <AxesSubplot:>



Findings

I originally suspected that certain months (numbered 1-12) would have a correlation with total sales (e.g. an average of higher sales in some months and lower in others), but the heatmap above suggests that this is not true.

Variables that have a strong correlation with sales include the following: - Last Month Sales (positive correlation) - Last 3 Month Average (positive Correalation) - Last Year Sales (positive Correlation) - Total Units Sold (positive correlation) - Product Count (How many products are offered) - Pax Filter Products Sold (weak positive correlation) - Mean Product THC (weak negative correlation) - Price Range (The range of price in products sold - the number different $5 increment products availible) - Offering CBD products (weak positive correlation) - CBD Product Offered (weak positive correlation) - ARP (weak negative correaltion)

Aside from the months, most other variables (Max CBD offered, etc) have either no correaltion or a weak negative one with total sales

1 4. Additional Data Feature Extraction

Drop the following variables - month variables (1-12) - Mean_IPP - Mean_CBD - Max_CBD - Months - Brand - All other varibales not present in altered dataset that are in original datasets - labels (units sold, total sales, total units)

Maintain Scalar values as is, binary values are already set. Category L1 has effectively been One Hot Encoded during preprocessing (not exactly, as more than one is possible, but the categorical values have been encoded into binary integer ones)

Imputation Strategies For Columns with NaN values - Last Month Set to 0 (no sales last month, new brand) - new column feature made for new brands - 3 Month Avg. Set to 0 - Last Year Set to 0 - Pax_Filter Set to 0
- Inhaleables Set to 0
- Ingestibles Set to 0
- Other_Cannabis: Set to 0
- Topicals: Set to 0
- Max_THC Set to Median
- Sell_CBD Set to 0
- Product_Count Set to Median
- Price_Range Set to Median
- Flavored_Count Set to Median
- Mood_Count Set to Median
- Mean_THC Set to Mean

Create new/cross features - New Product (Binary 1 or 0, depending on if Last Month is NaN or not) - Max THC and Ingestibles (These variables are positively correlated)

Scaling strategies:

Apply Standard Scaling to the following variables - Last Month - 3 Month Avg. - Last Year - Max_THC - Product_Count - Price_Range - Flavored_Count - Mood Count - Mean THC

Pipeline for all of these alterations is implemented directly below

[28]:
```python
# 5. Create Pipeline

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin

data_y = filtered_data["Total Sales ($)"].copy()

med_impute_cols = ["Last Year", "Max_THC", "ARP", "Product_Count",␣
 ↪"Price_Range", "Flavored_Count", "Mood_Count", "MTHC_Ing_Cross"]
```

```
zero_scale_cols = []
zero_noscale_cols = ["Pax_Filter", "Inhaleables", "Ingestibles",␣
↪"Other_Cannabis", "Topicals", "Sell_CBD", "New_Product"]
mean_impute_cols = ["Last Month", "3 Month Avg","Mean_THC"]
#scale_cols = ["Last Month", "3 Month Avg", "Last Year", "Max_THC",␣
↪"Product_Count", "Price_Range", "Flavored_Count", "Mood_Count", "Mean_THC",␣
↪"MTHC_Ing_Cross"]

class AugmentFeatures(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        X = X.drop(["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11",␣
↪"12", "Mean_IPP", "Mean_CBD", "Max_CBD", "Brand", "Total Sales ($)", "Total␣
↪Units", "Months"], 1)
        X['MTHC_Ing_Cross'] = X["Max_THC"] * X["Ingestibles"]
        X['New_Product'] = (float('-inf') < X["Last Month"]) & (float('inf') >␣
↪X["Last Month"])
        X['New_Product'] = X['New_Product'].astype(int) * 1.0
        #X.info()
        return X

attr_filter = AugmentFeatures()
filtered_data_proc = attr_filter.transform(filtered_data)

med_pipeline = Pipeline([
    ('median_imputer', SimpleImputer(strategy='median')),
    ('std_scale', StandardScaler())
])

zero_scale_pipeline = Pipeline([
    ('zero_imputer', SimpleImputer(strategy='constant', fill_value=0)),
    ('std_scale', StandardScaler())
])

zero_noscale_pipeline = Pipeline([
    ('zero_imputer', SimpleImputer(strategy='constant', fill_value=0))
])

mean_pipeline = Pipeline([
    ('mean_imputer', SimpleImputer(strategy='mean')),
    ('std_scale', StandardScaler())
])

transformer = ColumnTransformer([
    ('med_imp', med_pipeline, med_impute_cols),
    ('zero_scale', zero_scale_pipeline, zero_scale_cols),
```

```
    ('zero_noscale', zero_noscale_pipeline, zero_noscale_cols),
    ('mean_imp', mean_pipeline, mean_impute_cols)
])

data_prepared = transformer.fit_transform(filtered_data_proc)
```

[29]:
```
data_prepared.shape
```

[29]: (25279, 18)

[30]:
```
# 7. Linear Regression

from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(data_prepared, data_y,␣
↪train_size=0.85, random_state=121)

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

preds_train = lin_reg.predict(X_train)
preds_test = lin_reg.predict(X_test)

def regression_results(y_true, y_pred):
    # Regression metrics
    explained_variance=metrics.explained_variance_score(y_true, y_pred)
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
    #mean_squared_log_error=metrics.mean_squared_log_error(y_true, y_pred)
    median_absolute_error=metrics.median_absolute_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    print('explained_variance: ', round(explained_variance,4))
    #print('mean_squared_log_error: ', round(mean_squared_log_error,4))
    print('r2: ', round(r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))

print("Training Set Results: ")
regression_results(y_train, preds_train)

print('\n' + "Testing Set Results: ")
regression_results(y_test, preds_test)
```

```
Training Set Results:
```

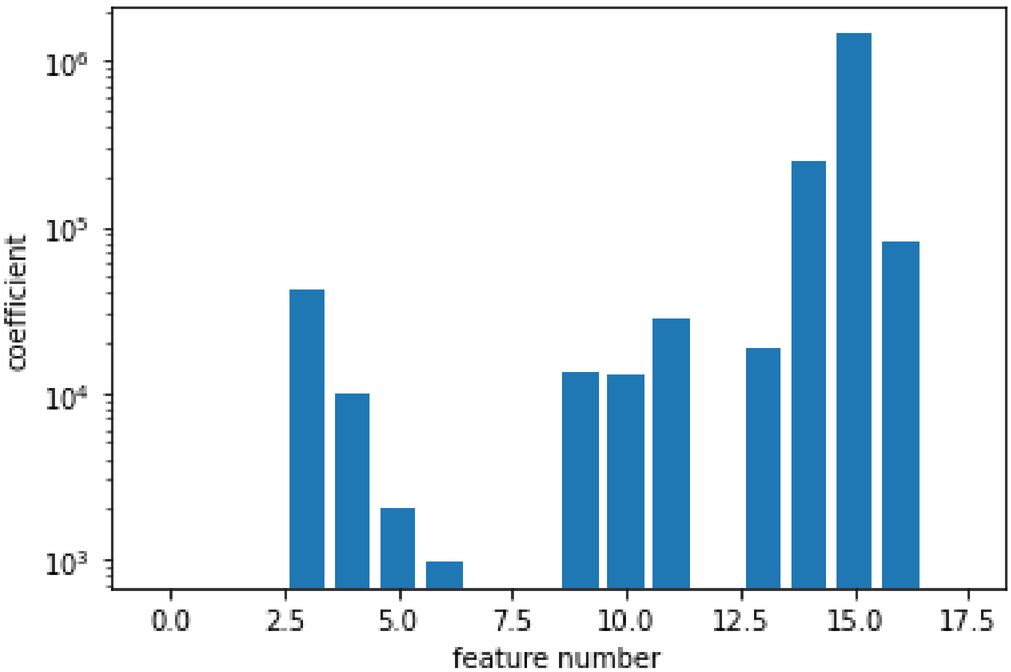
```
explained_variance:  0.946
r2:  0.946
MAE:  82824.3978
MSE:  131439722934.8976
RMSE:  362546.1666

Testing Set Results:
explained_variance:  0.9776
r2:  0.9776
MAE:  89337.4836
MSE:  71251360719.1529
RMSE:  266929.5051
```

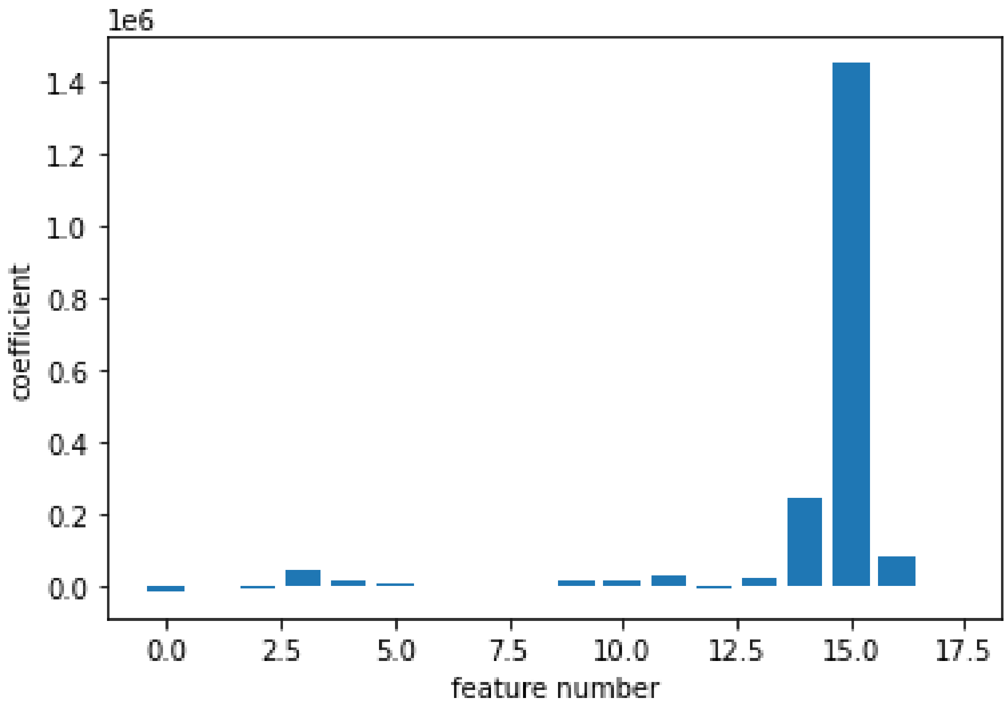
[31]:

```
# Feature Importance based on weight in model

coefficients = lin_reg.coef_

feature_ordered_list = ["Last_Month", "3_Month_Avg", "Last_Year", "ARP",␣
↪"Pax_Filer", "Inhaleables", "Ingestibles", "Other_Cannabis", "Topicals",␣
↪"Max_THC", "Sell_CBD", "Product_Count", "Price_Range", "Flavored_Count",␣
↪"Mood_Count", "Mean_THC", "MTHC_Ing_Cross", "New_Product"]

for i,v in enumerate(coefficients):
    print('Feature: %0d, %s Score: %.5f' % (i,feature_ordered_list[i],v))
# plot feature importance
plt.bar([x for x in range(len(coefficients))], coefficients)
plt.yscale('log')
plt.xlabel("feature number")
plt.ylabel("coefficient")
plt.show()

plt.bar([x for x in range(len(coefficients))], coefficients)
plt.yscale('linear')
plt.xlabel("feature number")
plt.ylabel("coefficient")
plt.show()
```

```
Feature: 0, Last_Month Score: -16726.93808
Feature: 1, 3_Month_Avg Score: -1252.40326
Feature: 2, Last_Year Score: -9613.25731
Feature: 3, ARP Score: 41837.51016
Feature: 4, Pax_Filer Score: 10005.31014
Feature: 5, Inhaleables Score: 2039.77175
Feature: 6, Ingestibles Score: 966.01786
Feature: 7, Other_Cannabis Score: -4060.54556
Feature: 8, Topicals Score: -6173.71553
Feature: 9, Max_THC Score: 13567.07529
Feature: 10, Sell_CBD Score: 12725.10978
```

```
Feature: 11, Product_Count Score: 27512.33976
Feature: 12, Price_Range Score: -10621.30227
Feature: 13, Flavored_Count Score: 18818.28732
Feature: 14, Mood_Count Score: 245711.08046
Feature: 15, Mean_THC Score: 1453753.73643
Feature: 16, MTHC_Ing_Cross Score: 81676.23301
Feature: 17, New_Product Score: -149.58918
```





[32]:
```python
import statsmodels.api as sm
from scipy import stats

X_train_2 = sm.add_constant(X_train)
est = sm.OLS(y_train, X_train_2)
lin_reg_2 = est.fit()
print(lin_reg_2.summary())
```

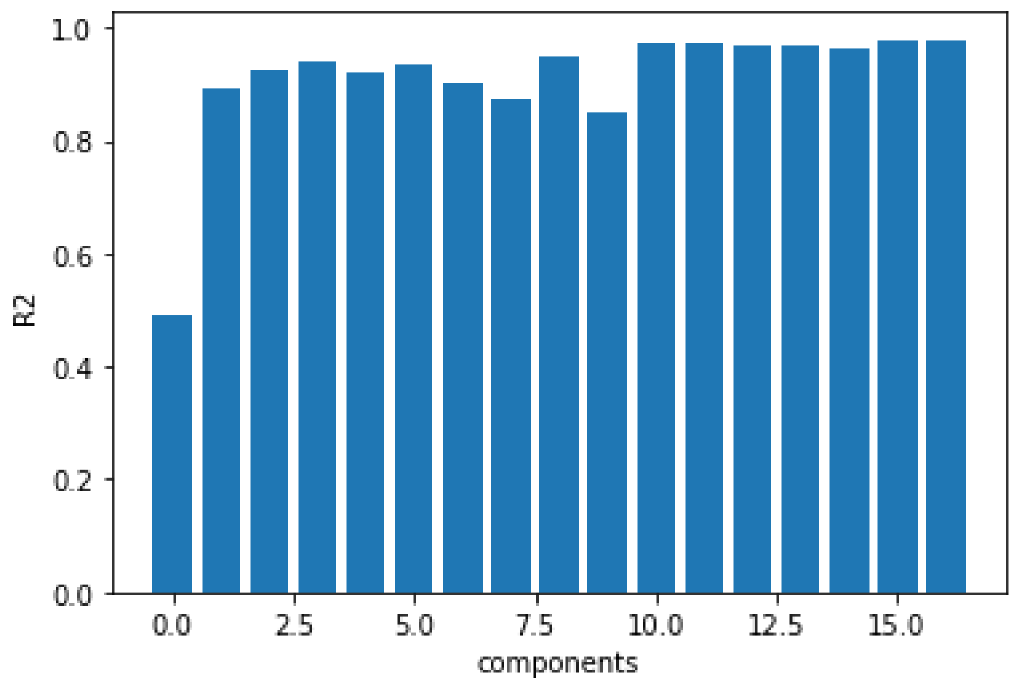
```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Total Sales ($)   R-squared:                       0.946
Model:                            OLS   Adj. R-squared:                  0.946
Method:                 Least Squares   F-statistic:                 2.089e+04
Date:                Sat, 04 Dec 2021   Prob (F-statistic):               0.00
Time:                        01:28:48   Log-Likelihood:            -3.0554e+05
No. Observations:               21487   AIC:                         6.111e+05
Df Residuals:                   21468   BIC:                         6.113e+05
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.616e+05    1.07e+04     15.154      0.000    1.41e+05    1.82e+05
x1         -1.673e+04    4277.383     -3.911      0.000   -2.51e+04   -8342.948
```

```
x2          -1252.4033   1.18e+04     -0.106      0.916   -2.45e+04     2.2e+04
x3          -9613.2573   2737.378     -3.512      0.000    -1.5e+04   -4247.792
x4           4.184e+04   3331.598     12.558      0.000    3.53e+04    4.84e+04
x5           1.001e+04   3166.037      3.160      0.002    3799.642    1.62e+04
x6           2039.7717   2988.370      0.683      0.495   -3817.656    7897.200
x7            966.0179   2587.289      0.373      0.709   -4105.261    6037.296
x8          -4060.5456   1.11e+04     -0.366      0.714   -2.58e+04    1.77e+04
x9          -6173.7155   1.53e+04     -0.403      0.687   -3.62e+04    2.38e+04
x10          1.357e+04   5931.178      2.287      0.022    1941.525    2.52e+04
x11          1.273e+04   8093.040      1.572      0.116   -3137.851    2.86e+04
x12          2.751e+04    1.2e+04      2.293      0.022    3995.970     5.1e+04
x13         -1.062e+04   1.15e+04     -0.923      0.356   -3.32e+04    1.19e+04
x14          1.882e+04   7134.866      2.638      0.008    4833.418    3.28e+04
x15          2.457e+05   1.03e+04     23.923      0.000    2.26e+05    2.66e+05
x16          1.454e+06   8437.734    172.292      0.000    1.44e+06    1.47e+06
x17          8.168e+04   9155.607      8.921      0.000    6.37e+04    9.96e+04
x18          -149.5892   4117.140     -0.036      0.971   -8219.491    7920.312
==============================================================================
Omnibus:                    75965.057   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):      46929124358.997
Skew:                          69.704   Prob(JB):                         0.00
Kurtosis:                    7241.664   Cond. No.                         12.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

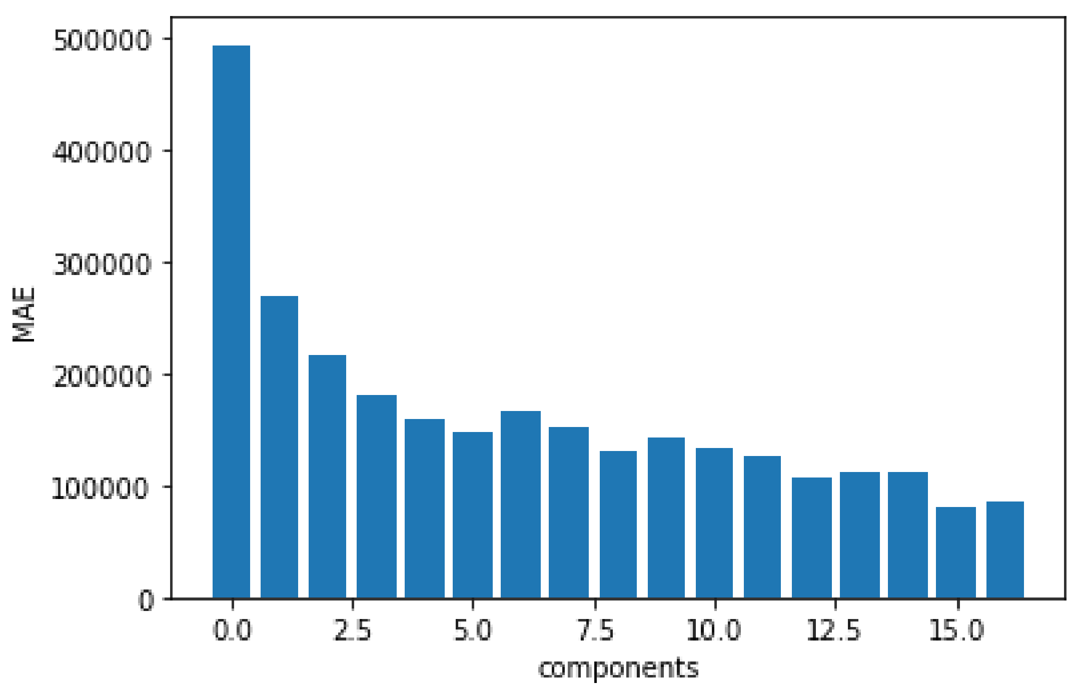
[33]:

```
# 8. PCA Analysis

from sklearn import decomposition
from sklearn.linear_model import LogisticRegression

r2_list = []
mae_list = []
mse_list = []
rmse_list = []

for i in range(1,18):

    pca = decomposition.PCA(n_components = i)
    data_pca = pca.fit_transform(data_prepared)
    X_train_pca, X_test_pca, y_train_pca, y_test_pca =␣
 ↪train_test_split(data_pca, data_y, train_size = 0.8)
    lin_reg_pca = LinearRegression()
    lin_reg_pca.fit(X_train_pca, y_train_pca)
    linreg_preds = lin_reg_pca.predict(X_test_pca)
```

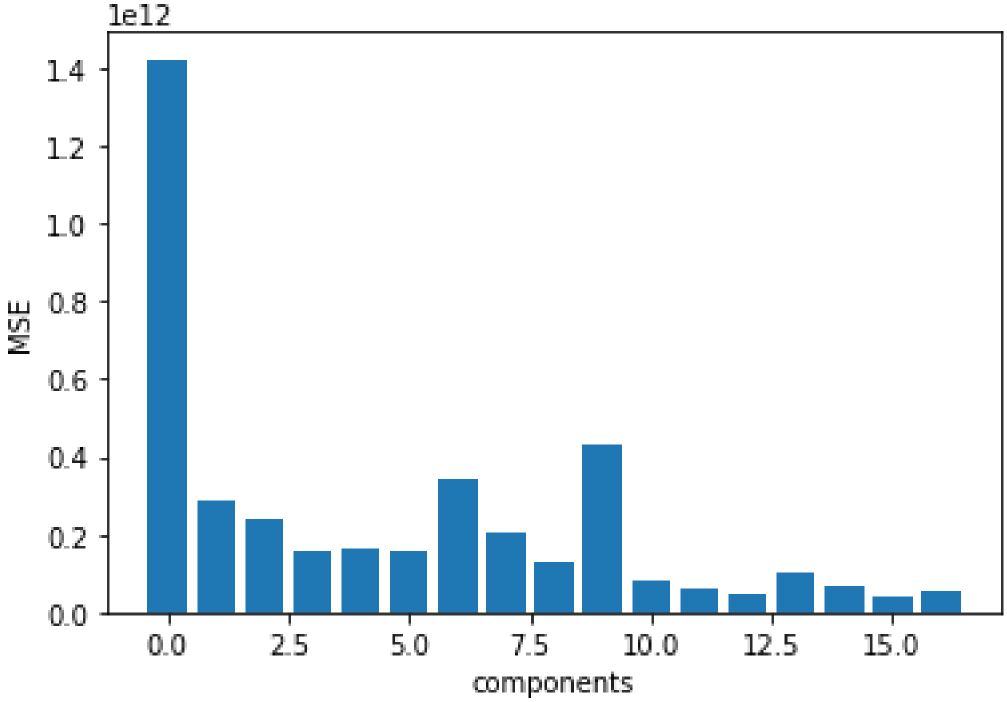
```python
    r2 = metrics.r2_score(y_test_pca, linreg_preds)
    mae = metrics.mean_absolute_error(y_test_pca, linreg_preds)
    mse = metrics.mean_squared_error(y_test_pca, linreg_preds)
    rmse = round(np.sqrt(mse),4)

    r2_list.append(r2)
    mae_list.append(mae)
    mse_list.append(mse)
    rmse_list.append(rmse)


plt.bar([x for x in range(len(r2_list))], r2_list)
plt.yscale('linear')
plt.xlabel("components")
plt.ylabel("R2")
plt.show()

plt.bar([x for x in range(len(mae_list))], mae_list)
plt.yscale('linear')
plt.xlabel("components")
plt.ylabel("MAE")
plt.show()

plt.bar([x for x in range(len(mse_list))], mse_list)
plt.yscale('linear')
plt.xlabel("components")
plt.ylabel("MSE")
plt.show()

plt.bar([x for x in range(len(rmse_list))], rmse_list)
plt.yscale('linear')
plt.xlabel("components")
plt.ylabel("RMSE")
plt.show()
```

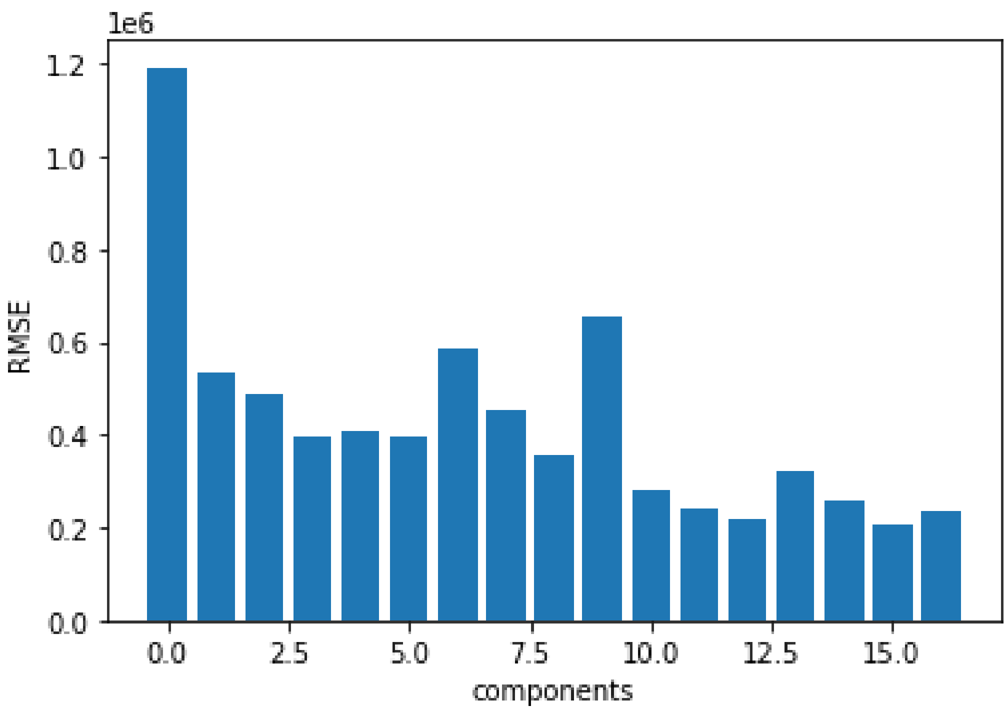


1.0
0.8
0.6
0.4
0.2
0.0
R2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
components



500000
400000
300000
200000
100000
0
MAE
0.0
2.5
5.0
7.5
10.0
12.5
15.0
components





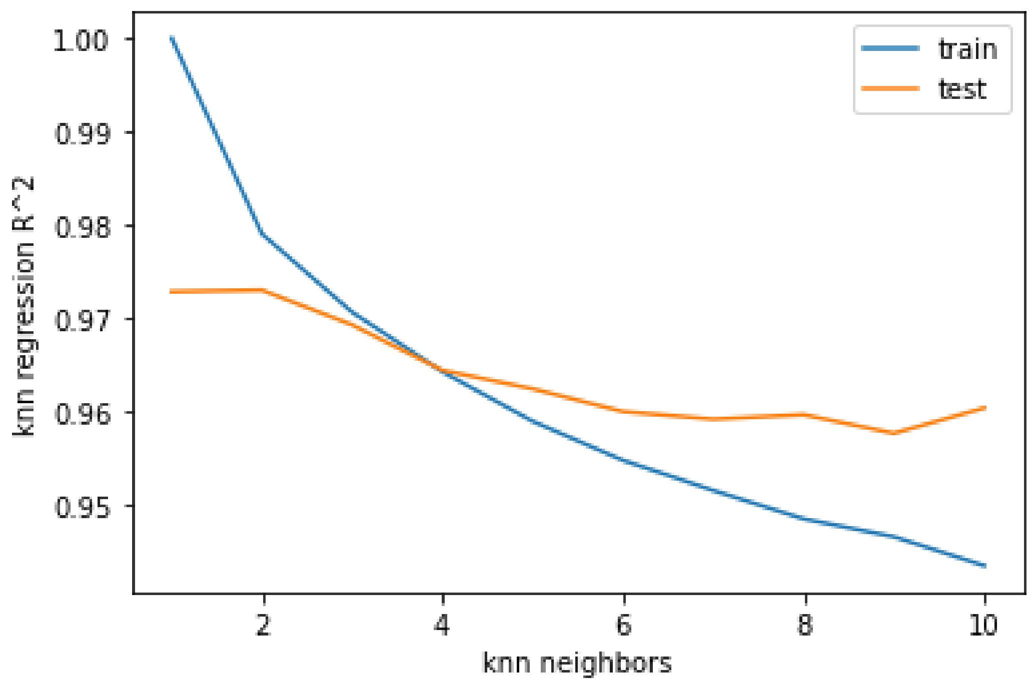
Based on these results, I choose to keep 12 components in my dataset following PCA. Adding additional components to this does not improve the performance of the model (looking at the data above tells us that increasing from 8 components does not really improve the model that much, but 12 is the closes number of components with improved performance in the model while still helping prevent overfitting).

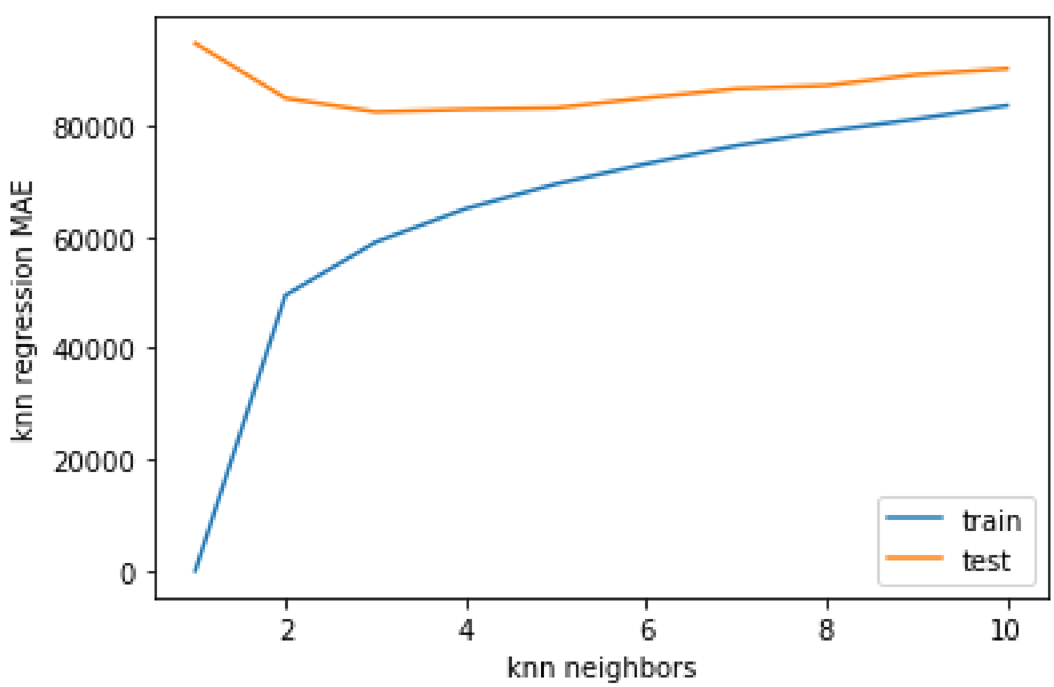
[34]:
```
pca = decomposition.PCA(n_components = 12)
data_pca = pca.fit_transform(data_prepared)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(data_pca,␣
↪data_y, train_size = 0.8)

data_pca.shape
```

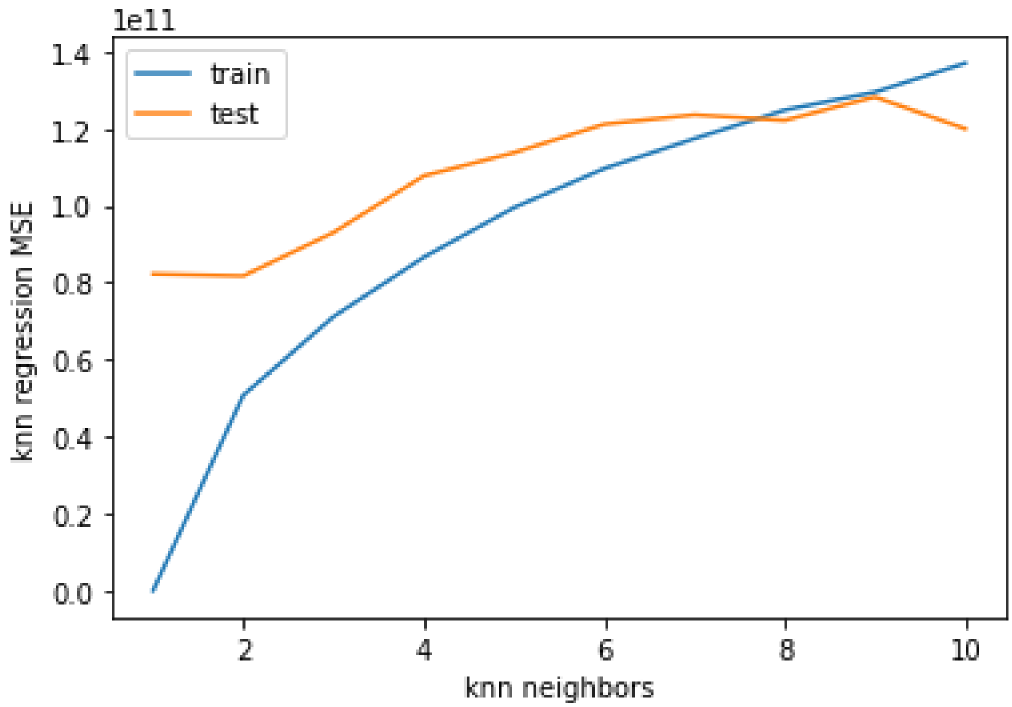
[34]:
```
(25279, 12)
```

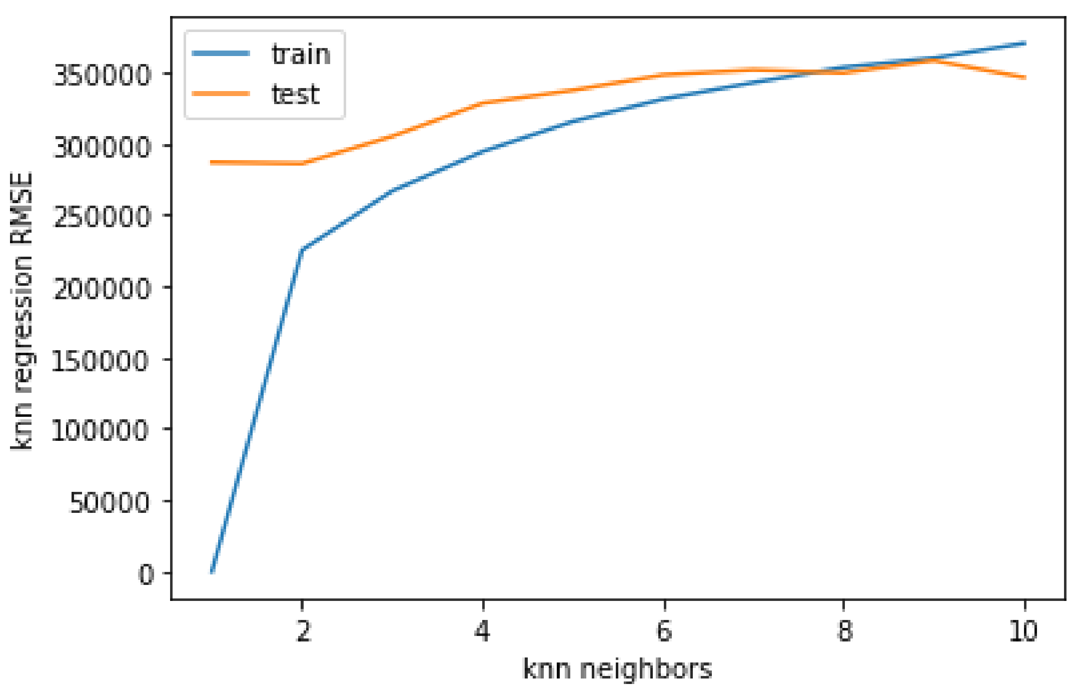
[35]:
```
# 9. Ensemble Method
#  Implement a KNN method for linear regression to have another baseline model␣
↪to compare to

from sklearn.neighbors import KNeighborsRegressor

#neighbors = [10,50,100,150,200,300,500,750,1000,1500, 2000, 3000, 5000]
neighbors = [1,2,3,4,5,6,7,8,9,10]

r2_train_list = []
r2_test_list = []
mae_train_list = []
mae_test_list = []
mse_train_list = []
mse_test_list = []
rmse_train_list = []
rmse_test_list = []

for n in neighbors:
    knn_cur = KNeighborsRegressor(n_neighbors=n)
    knn_cur.fit(X_train_pca, y_train_pca)
    preds_train_cur = knn_cur.predict(X_train_pca)
    preds_test_cur = knn_cur.predict(X_test_pca)

    r2 = metrics.r2_score(y_train_pca, preds_train_cur)
    mae = metrics.mean_absolute_error(y_train_pca, preds_train_cur)
    mse = metrics.mean_squared_error(y_train_pca,  preds_train_cur)
    rmse = round(np.sqrt(mse),4)

    r2_train_list.append(r2)
    mae_train_list.append(mae)
    mse_train_list.append(mse)
    rmse_train_list.append(rmse)
```

```python
    r2 = metrics.r2_score(y_test_pca, preds_test_cur)
    mae = metrics.mean_absolute_error(y_test_pca, preds_test_cur)
    mse = metrics.mean_squared_error(y_test_pca,  preds_test_cur)
    rmse = round(np.sqrt(mse),4)

    r2_test_list.append(r2)
    mae_test_list.append(mae)
    mse_test_list.append(mse)
    rmse_test_list.append(rmse)

plt.xlabel('knn neighbors')
plt.ylabel('knn regression R^2')
plt.plot(neighbors, r2_train_list, label='train')
plt.plot(neighbors, r2_test_list, label='test')
plt.legend()
plt.show()

plt.xlabel('knn neighbors')
plt.ylabel('knn regression MAE')
plt.plot(neighbors, mae_train_list, label='train')
plt.plot(neighbors, mae_test_list, label='test')
plt.legend()
plt.show()

plt.xlabel('knn neighbors')
plt.ylabel('knn regression MSE')
plt.plot(neighbors, mse_train_list, label='train')
plt.plot(neighbors, mse_test_list, label='test')
plt.legend()
plt.show()

plt.xlabel('knn neighbors')
plt.ylabel('knn regression RMSE')
plt.plot(neighbors, rmse_train_list, label='train')
plt.plot(neighbors, rmse_test_list, label='test')
plt.legend()
plt.show()
```


1.00
0.99
0.98
0.97
0.96
0.95
knn regression R^2
train
test
2
4
6
8
10
knn neighbors


80000
60000
40000
20000
0
knn regression MAE
train
test
2
4
6
8
10
knn neighbors





Based on checking performance accuracy with multiple numbers of neighbors considered, using either 1 or 2 yields the greatest performance. We use 2 from now on.

[36]:

```python
knn_cur = KNeighborsRegressor(n_neighbors=2)
knn_cur.fit(X_train_pca, y_train_pca)
preds_train = knn_cur.predict(X_train_pca)
preds_test = knn_cur.predict(X_test_pca)

print("KNN Regression Model (n_neighbors=2) Metric Following PCA reduction to␣
↪12 components")

print("\nTraining Set Metrics")
regression_results(y_train_pca, preds_train)
print("\nTesting Set Metrics")
regression_results(y_test_pca, preds_test)

linreg_pca = LinearRegression()
linreg_pca.fit(X_train_pca, y_train_pca)
lin_reg_pca.fit(X_train_pca, y_train_pca)
linreg_preds_train = lin_reg_pca.predict(X_train_pca)
linreg_preds_test = lin_reg_pca.predict(X_test_pca)

print("Linear Regression Model Metric Following PCA reduction to 12 components")

print("\nTraining Set Metrics")
regression_results(y_train_pca, linreg_preds_train)
print("\nTesting Set Metrics")
regression_results(y_test_pca, linreg_preds_test)
```

```
KNN Regression Model (n_neighbors=2) Metric Following PCA reduction to 12
components

Training Set Metrics
explained_variance:  0.9791
r2:  0.9791
MAE:  49494.1261
MSE:  50756344592.9696
RMSE:  225291.6878

Testing Set Metrics
explained_variance:  0.9731
r2:  0.973
MAE:  84892.3058
MSE:  81707341523.4399
RMSE:  285844.9606
Linear Regression Model Metric Following PCA reduction to 12 components

Training Set Metrics
explained_variance:  0.9298
r2:  0.9298
MAE:  131665.8164
```

```
MSE:  170356210799.2939
RMSE:  412742.3056

Testing Set Metrics
explained_variance:  0.9631
r2:  0.9631
MAE:  128519.7399
MSE:  111738089097.2553
RMSE:  334272.4773
```

By comparing the testing set metrics of the KNN regression and Linear REgression, we see that Linear Regression displays better performance (greater R^2, lower MAE, MSE, and RMSE)

[37]:
```python
# Random Forest Ensemble Method

from sklearn.ensemble import RandomForestRegressor

rforest_reg = RandomForestRegressor()
rforest_reg.fit(X_train_pca, y_train_pca)
rf_preds_train = rforest_reg.predict(X_train_pca)
rf_preds_test = rforest_reg.predict(X_test_pca)

print("Random Forest Regressor Metrics Following PCA reduction to 12␣
 ↪components")

print("\nTraining Set Metrics")
regression_results(y_train_pca, linreg_preds_train)
print("\nTesting Set Metrics")
regression_results(y_test_pca, linreg_preds_test)
```

```
Random Forest Regressor Metrics Following PCA reduction to 12 components

Training Set Metrics
explained_variance:  0.9298
r2:  0.9298
MAE:  131665.8164
MSE:  170356210799.2939
RMSE:  412742.3056

Testing Set Metrics
explained_variance:  0.9631
r2:  0.9631
MAE:  128519.7399
MSE:  111738089097.2553
RMSE:  334272.4773
```

[38]:
```python
# 10. Cross Validation (10-fold)
```

```python
from sklearn.model_selection import KFold

kfold = KFold(n_splits = 10, shuffle=True)
splits = kfold.split(data_pca, data_y)

linreg_r2 = []
linreg_mae = []
linreg_mse = []
linreg_rmse = []

knnreg_r2 = []
knnreg_mae = []
knnreg_mse = []
knnreg_rmse = []

randfor_r2 = []
randfor_mae = []
randfor_mse = []
randfor_rmse = []

for train_indeces, test_indeces in splits:

    X_train = data_pca[train_indeces]
    y_train = data_y.iloc[train_indeces]

    X_test = data_pca[test_indeces, :]
    y_test = data_y.iloc[test_indeces]

    linreg = LinearRegression()
    knnreg = KNeighborsRegressor(n_neighbors=2)
    rafreg = RandomForestRegressor()


    linreg.fit(X_train, y_train)
    knnreg.fit(X_train, y_train)
    rafreg.fit(X_train, y_train)

    linreg_preds = linreg.predict(X_test)
    knnreg_preds = knnreg.predict(X_test)
    rafreg_preds = rafreg.predict(X_test)

    linreg_r2.append(metrics.r2_score(y_test, linreg_preds))
    linreg_mae.append(metrics.mean_absolute_error(y_test, linreg_preds))
    linreg_mse.append(metrics.mean_squared_error(y_test,  linreg_preds))
    linreg_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
↪linreg_preds)),4))
```

```
    knnreg_r2.append(metrics.r2_score(y_test, knnreg_preds))
    knnreg_mae.append(metrics.mean_absolute_error(y_test, knnreg_preds))
    knnreg_mse.append(metrics.mean_squared_error(y_test,  knnreg_preds))
    knnreg_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
↪knnreg_preds)),4))

    randfor_r2.append(metrics.r2_score(y_test, rafreg_preds))
    randfor_mae.append(metrics.mean_absolute_error(y_test, rafreg_preds))
    randfor_mse.append(metrics.mean_squared_error(y_test,  rafreg_preds))
    randfor_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
↪rafreg_preds)),4))


print("Linear Regression Average R^2: " + str(np.mean(linreg_r2)))
print("Linear Regression Average MAE: " + str(np.mean(linreg_mae)))
print("Linear Regression Average MSE: " + str(np.mean(linreg_mse)))
print("Linear Regression Average RMSE: " + str(np.mean(linreg_rmse)))

print("KNN Regression Average R^2: " + str(np.mean(knnreg_r2)))
print("KNN Regression Average MAE: " + str(np.mean(knnreg_mae)))
print("KNN Regression Average MSE: " + str(np.mean(knnreg_mse)))
print("KNN Regression Average RMSE: " + str(np.mean(knnreg_rmse)))

print("Random Forest Regression Average R^2: " + str(np.mean(randfor_r2)))
print("Random Forest Average MAE: " + str(np.mean(randfor_mae)))
print("Random Forest Average MSE: " + str(np.mean(randfor_mse)))
print("Random Forest Average RMSE: " + str(np.mean(randfor_rmse)))
```

```
Linear Regression Average R^2: 0.9397111976739853
Linear Regression Average MAE: 131890.71095575058
Linear Regression Average MSE: 160319480077.21356
Linear Regression Average RMSE: 363027.2107
KNN Regression Average R^2: 0.9521348314837568
KNN Regression Average MAE: 85917.5833494429
KNN Regression Average MSE: 128593020773.09123
KNN Regression Average RMSE: 319363.48352
Random Forest Regression Average R^2: 0.948409285946294
Random Forest Average MAE: 87530.34380017841
Random Forest Average MSE: 137768222451.0931
Random Forest Average RMSE: 331004.81853000005
```

Compare this to data without PCA applied

[39]:
```
from sklearn.model_selection import KFold

kfold = KFold(n_splits = 10, shuffle=True)
splits = kfold.split(data_prepared, data_y)
```

```
linreg_r2 = []
linreg_mae = []
linreg_mse = []
linreg_rmse = []

knnreg_r2 = []
knnreg_mae = []
knnreg_mse = []
knnreg_rmse = []

randfor_r2 = []
randfor_mae = []
randfor_mse = []
randfor_rmse = []

for train_indeces, test_indeces in splits:

    X_train = data_prepared[train_indeces]
    y_train = data_y.iloc[train_indeces]

    X_test = data_prepared[test_indeces, :]
    y_test = data_y.iloc[test_indeces]

    linreg = LinearRegression()
    knnreg = KNeighborsRegressor(n_neighbors=2)
    rafreg = RandomForestRegressor()


    linreg.fit(X_train, y_train)
    knnreg.fit(X_train, y_train)
    rafreg.fit(X_train, y_train)

    linreg_preds = linreg.predict(X_test)
    knnreg_preds = knnreg.predict(X_test)
    rafreg_preds = rafreg.predict(X_test)

    linreg_r2.append(metrics.r2_score(y_test, linreg_preds))
    linreg_mae.append(metrics.mean_absolute_error(y_test, linreg_preds))
    linreg_mse.append(metrics.mean_squared_error(y_test,  linreg_preds))
    linreg_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
↪linreg_preds)),4))

    knnreg_r2.append(metrics.r2_score(y_test, knnreg_preds))
    knnreg_mae.append(metrics.mean_absolute_error(y_test, knnreg_preds))
    knnreg_mse.append(metrics.mean_squared_error(y_test,  knnreg_preds))
    knnreg_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
↪knnreg_preds)),4))
```

```
    randfor_r2.append(metrics.r2_score(y_test, rafreg_preds))
    randfor_mae.append(metrics.mean_absolute_error(y_test, rafreg_preds))
    randfor_mse.append(metrics.mean_squared_error(y_test,  rafreg_preds))
    randfor_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
↪rafreg_preds)),4))


print("Linear Regression Average R^2: " + str(np.mean(linreg_r2)))
print("Linear Regression Average MAE: " + str(np.mean(linreg_mae)))
print("Linear Regression Average MSE: " + str(np.mean(linreg_mse)))
print("Linear Regression Average RMSE: " + str(np.mean(linreg_rmse)))

print("KNN Regression Average R^2: " + str(np.mean(knnreg_r2)))
print("KNN Regression Average MAE: " + str(np.mean(knnreg_mae)))
print("KNN Regression Average MSE: " + str(np.mean(knnreg_mse)))
print("KNN Regression Average RMSE: " + str(np.mean(knnreg_rmse)))

print("Random Forest Regression Average R^2: " + str(np.mean(randfor_r2)))
print("Random Forest Average MAE: " + str(np.mean(randfor_mae)))
print("Random Forest Average MSE: " + str(np.mean(randfor_mse)))
print("Random Forest Average RMSE: " + str(np.mean(randfor_rmse)))
```

```
Linear Regression Average R^2: 0.9561843546470922
Linear Regression Average MAE: 83589.4922107453
Linear Regression Average MSE: 123247494417.44443
Linear Regression Average RMSE: 295474.7414100001
KNN Regression Average R^2: 0.9429495156388995
KNN Regression Average MAE: 83783.73564100184
KNN Regression Average MSE: 140615550422.30243
KNN Regression Average RMSE: 335589.14285
Random Forest Regression Average R^2: 0.9542348536857073
Random Forest Average MAE: 79310.09288001957
Random Forest Average MSE: 125168736383.36545
Random Forest Average RMSE: 310490.96091
```

We find that applying PCA actually hurts the performance of our models so we do not emply it going forwards

[40]:
```
# 11. Gridsearch to Optimize Parameters
#    Here we optimize the parameters on this dataset for the random forest␣
↪ensemble model we produced earlier
#    by default (which we employ above), the number of trees is 100, criterion␣
↪is "squared error", max_depth
#    is unlimited, bootstrap=True, and there are many others. The␣
↪hyperparameters were are going to experiment
#    with is the number of trees, the criterion, and max tree depth.
#
```

```
#    Note: By default, GridSearchCV employs 5-fold cross validation

X_train, X_test, y_train, y_test = train_test_split(data_prepared, data_y,␣
↪train_size=0.85, random_state=122)

from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [10, 50, 100, 200],
     'max_depth': [5, 10, 15]
    }
]

base_estimator = RandomForestRegressor()
sh = GridSearchCV(base_estimator, param_grid).fit(X_train, y_train)

sh.best_estimator_
```

[40]:

```
RandomForestRegressor(max_depth=15, n_estimators=50)
```

[41]:

```
# implement hyperparameter tuned random forest

optimal_randforest = RandomForestRegressor(max_depth=15, n_estimators=200)
optimal_randforest.fit(X_train, y_train)
opt_preds_train = optimal_randforest.predict(X_train)
opt_preds_test = optimal_randforest.predict(X_test)

print("Random Forest Metrics Following Hyperparameter Optimization")

print("\nTraining Set Metrics")
regression_results(y_train, opt_preds_train)
print("\nTesting Set Metrics")
regression_results(y_test, opt_preds_test)
```

```
Random Forest Metrics Following Hyperparameter Optimization

Training Set Metrics
explained_variance:  0.9913
r2:  0.9913
MAE:  46162.2107
MSE:  22770826332.4789
RMSE:  150900.0541

Testing Set Metrics
explained_variance:  0.98
r2:  0.98
MAE:  77355.783
MSE:  42009717731.5782
```

```
RMSE:  204962.7228
```

[42]:
```
# 12. Experiment with Custom Model and Report Findings/Metrics

# Here we will try to train a neural network regressor to predict sales

from sklearn.neural_network import MLPRegressor

param_grid = [{
    'hidden_layer_sizes': [(10,10,10,10,10), (10,10,10,10,10,10,10),
                           (15,15,15,15,15), (15,15,15,15,15,15,15),
                           (20,20,20,20,20), (20,20,20,20,20,20,20)
                          ],
    'max_iter': [2000]
}]

base_estimator = MLPRegressor()
sh = GridSearchCV(base_estimator, param_grid).fit(X_train, y_train)

sh.best_estimator_
```

[42]:
```
MLPRegressor(hidden_layer_sizes=(20, 20, 20, 20, 20), max_iter=2000)
```

[43]:
```
nn = MLPRegressor(hidden_layer_sizes=(10,10,10,10,10), max_iter=500)
nn.fit(X_train, y_train)

nn_preds_train = nn.predict(X_train)
nn_preds_test = nn.predict(X_test)

print("Neural Network Regressor Metrics")

print("\nTraining Set Metrics")
regression_results(y_train, nn_preds_train)
print("\nTesting Set Metrics")
regression_results(y_test, nn_preds_test)
```

```
Neural Network Regressor Metrics Following PCA reduction to 12 components

Training Set Metrics
explained_variance:  0.9494
r2:  0.9494
MAE:  85570.7136
MSE:  132793678641.7084
RMSE:  364408.6698

Testing Set Metrics
explained_variance:  0.9825
r2:  0.9824
```

```
MAE:  83998.5607
MSE:  36921165101.2723
RMSE:  192148.8098
```

[45]:

```python
# Finalized Models and Performance Metrics Summary

from sklearn.metrics import mean_absolute_percentage_error

kfold = KFold(n_splits = 10, shuffle=True)
splits = kfold.split(data_prepared, data_y)

linreg_r2 = []
linreg_mae = []
linreg_mse = []
linreg_rmse = []
linreg_mape = []

knnreg_r2 = []
knnreg_mae = []
knnreg_mse = []
knnreg_rmse = []
knnreg_mape = []

randfor_r2 = []
randfor_mae = []
randfor_mse = []
randfor_rmse = []
randfor_mape = []

nnreg_r2 = []
nnreg_mae = []
nnreg_mse = []
nnreg_rmse = []
nnreg_mape = []

for train_indeces, test_indeces in splits:

    X_train = data_prepared[train_indeces]
    y_train = data_y.iloc[train_indeces]

    X_test = data_prepared[test_indeces, :]
    y_test = data_y.iloc[test_indeces]

    linreg = LinearRegression()
    knnreg = KNeighborsRegressor(n_neighbors = 3)
    rafreg = RandomForestRegressor(max_depth = 15, n_estimators = 50)
    nnreg = MLPRegressor(hidden_layer_sizes=(20, 20, 20, 20, 20), max_iter=2000)
```

```
    linreg.fit(X_train, y_train)
    knnreg.fit(X_train, y_train)
    rafreg.fit(X_train, y_train)
    nnreg.fit(X_train, y_train)

    linreg_preds = linreg.predict(X_test)
    knnreg_preds = knnreg.predict(X_test)
    rafreg_preds = rafreg.predict(X_test)
    nnreg_preds = nnreg.predict(X_test)

    linreg_r2.append(metrics.r2_score(y_test, linreg_preds))
    linreg_mae.append(metrics.mean_absolute_error(y_test, linreg_preds))
    linreg_mse.append(metrics.mean_squared_error(y_test,  linreg_preds))
    linreg_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
 ↪linreg_preds)),4))
    linreg_mape.append(mean_absolute_percentage_error(y_test, linreg_preds))

    knnreg_r2.append(metrics.r2_score(y_test, knnreg_preds))
    knnreg_mae.append(metrics.mean_absolute_error(y_test, knnreg_preds))
    knnreg_mse.append(metrics.mean_squared_error(y_test,  knnreg_preds))
    knnreg_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
 ↪knnreg_preds)),4))
    knnreg_mape.append(mean_absolute_percentage_error(y_test, knnreg_preds))

    randfor_r2.append(metrics.r2_score(y_test, rafreg_preds))
    randfor_mae.append(metrics.mean_absolute_error(y_test, rafreg_preds))
    randfor_mse.append(metrics.mean_squared_error(y_test,  rafreg_preds))
    randfor_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
 ↪rafreg_preds)),4))
    randfor_mape.append(mean_absolute_percentage_error(y_test, rafreg_preds))

    nnreg_r2.append(metrics.r2_score(y_test, nnreg_preds))
    nnreg_mae.append(metrics.mean_absolute_error(y_test, nnreg_preds))
    nnreg_mse.append(metrics.mean_squared_error(y_test,  nnreg_preds))
    nnreg_rmse.append(round(np.sqrt(metrics.mean_squared_error(y_test, ␣
 ↪nnreg_preds)),4))
    nnreg_mape.append(mean_absolute_percentage_error(y_test, nnreg_preds))

print("Linear Regression")
print("R^2: " + str(np.mean(linreg_r2)))
print("MAE: " + str(np.mean(linreg_mae)))
print("MSE: " + str(np.mean(linreg_mse)))
print("RMSE: " + str(np.mean(linreg_rmse)))
print("MAPE: " + str(np.mean(linreg_mape)))
```

```python
print("\nKNN Regression")
print("R^2: " + str(np.mean(knnreg_r2)))
print("MAE: " + str(np.mean(knnreg_mae)))
print("MSE: " + str(np.mean(knnreg_mse)))
print("RMSE: " + str(np.mean(knnreg_rmse)))
print("MAPE: " + str(np.mean(knnreg_mape)))

print("\nRandom Forest Regression")
print("R^2: " + str(np.mean(randfor_r2)))
print("MAE: " + str(np.mean(randfor_mae)))
print("MSE: " + str(np.mean(randfor_mse)))
print("RMSE: " + str(np.mean(randfor_rmse)))
print("MAPE: " + str(np.mean(randfor_mape)))

print("\nNeural Network Regression")
print("R^2: " + str(np.mean(nnreg_r2)))
print("MAE: " + str(np.mean(nnreg_mae)))
print("MSE: " + str(np.mean(nnreg_mse)))
print("RMSE: " + str(np.mean(nnreg_rmse)))
print("MAPE: " + str(np.mean(nnreg_mape)))
```

```
Linear Regression
R^2: 0.9506262229145209
MAE: 83673.11524785537
MSE: 123681722060.72571
RMSE: 294337.1583
MAPE: 4.891075910234255e+16

KNN Regression
R^2: 0.9493492344416433
MAE: 80769.3875125602
MSE: 126561893173.7651
RMSE: 312018.23373000004
MAPE: 8148317432747818.0

Random Forest Regression
R^2: 0.93763077495652
MAE: 78643.25576040288
MSE: 136881793505.19601
RMSE: 321320.1806
MAPE: 2.4269400126400844e+16

Neural Network Regression
R^2: 0.9521027275545263
MAE: 83010.9176332785
MSE: 120426132575.03174
RMSE: 289853.4439
MAPE: 5.373067364072941 6e+16
```

```
[ ]:
```