CS 148 - Project 3
Brian Chang and William Randall

Executive Summary

The cannabis industry is an extremely fast growing industry both in California and around the world. Global annual sales are projected to breach 30.6 billion USD in 2021, and are expected to double within the next 5 years. In order to remain competitive and innovative it is imperative that we utilise data to generate novel insights on our consumers and their purchase decisions.

In order to leverage data to build a predictive model of sales performance, we used sales data from 2018-2021 to train our models. From the dataset, we focused on the the following features: Method of Ingestion (ie. inhalables, ingestibles, etc.), Total Monthly Units Sold, and the change in sales volume or units sold from month to month. These features were chosen as they had the highest correlation with our target variable.

We then iteratively created 5 different models, LinReg, LinReg-PCA, SVM, Random Forest, and Bagging, in order to find the best fit for our data. Using GridSearchCV, we optimised the model parameters, and conducted 10-fold cross validation on their results. Finally, we observed that our Linear Regression model produced the most accurate sales predictions, and recommend that Cookies use this model for future forecasts.

Finally, for the follow up steps pertaining to analytic work, we recommend that Cookies continue aggregating sales data, both state-wide and nationally, as well as, to attempt to explore creating new models using different methods. While we have tested 5 different models in this report, it is by no means exhaustive and though we are confident in our results, we cannot rule out that there may exist a better performing model given this dataset.

Background/Introduction:

The cannabis industry is one of the fastest growing industries in the world, with global forecasted sales of 30.6 billion dollars in 2021 alone. This upward trend has persisted despite the COVID-19 pandemic as global cannabis sales grew by 45% between 2019 and 2020, while domestic sales grew by 48% in the same span. (BSDA, 2021) Furthermore, due to the emergent nature of this industry, with each passing year, the market is only growing bigger.

Cultural adoption is on the rise as well. Currently, 73% of 21+ adults in fully-legal states either consume cannabis or are open to consuming cannabis, demonstrating widespread adoption following positive legislation. Furthermore, 87% of adults in the U.S. agree that some form of cannabis use should be legal. (BSDA, 2021) This collective mindset shift has led the way for full legalisation of adult cannabis use in 18 states and Washington D.C., and with each new state, the market size only increases and new opportunities are presented.

In order to fully capitalise on these new opportunities, it is imperative that we deeply understand consumer behavior, in order to identify what motivates their purchase decisions.

Methodology:

<u>2) Develop basic Time Series Feature Extraction Plan - develop a series of standard timeseries features to augment your dataset and enable timeseries predictive models.</u>

For the Time Series Feature Extraction Plan we decided to add the "previous month's total units", "change in units", "rolling average of units", "sales from previous months", "change in sales per month", and a "rolling average of sales". These new features would hold information from the previous time periods in the current row. Then from there we would remove all rows/brands which did not have at least 6 months of data, or if they did not have any products, or if they had all NaN values. From here we were able to begin to try to predict the next month's total sales. By including these averages we were able to have more substantial information about the current month and brand. It allowed our models to be able to look backwards without just using the previous month's sales.

<u>6) Document your data strategy in your report. Provide an explanation or justification for why you chose the data you did, and also detail any experiments you ran and the results.</u>

In order to determine our data strategy we first created a correlation matrix to better visualise the correlations between features within our dataset. First we determined the percentage of each feature that was NaN. From this we could determine which features we should keep and which we should drop. Then we created features based on the time series data such as the “difference” features and the “rolling average” features because those would give us more information when we try to predict the next month's sales. Then we created features such as “Carries Inhalables,” “Carries Topicals,” and “Carries Ingestibles.” These binary features would allow us to know which brand sold different types of cannabis and types of ingestion they had in their brand. We also removed any features that 100% NaN values as we deemed them unusable. We also dropped any brand which had less than or equal to 6 months of data because we concluded that these brands would not have enough data to be usable. To test our hypothesis we tried different ways of imputing the data and settled on median imputation and we also tested dropping different features and running that new feature set through our models.

9) Employ an ensemble method to your predictive model exercise - Leverage an ensemble learning method to generate an optimized prediction model.

For our ensemble method, we decided to choose the random forest method. This is because we wanted a method that would be able to reduce the variance of the dataset. Random forest allows us to aggregate/average the predictions made by multiple models, allowing the final variance to be lower than the variance of any individual learner.

Furthermore, we wanted an ensemble technique that would be able to handle the high-dimensionality of our dataset(s). The Random Forest method is generally quite robust (less influenced by outliers in the data), which is especially useful for our dataset as the ranges for the total sales, APR, and total units features are very wide. Random Forest can handle binary features, categorical features, and numerical features which are all present within our dataset.

Lastly, the data does not need to be standardised or transformed in order to be used by the model which makes our data pipeline simpler/easier to implement.

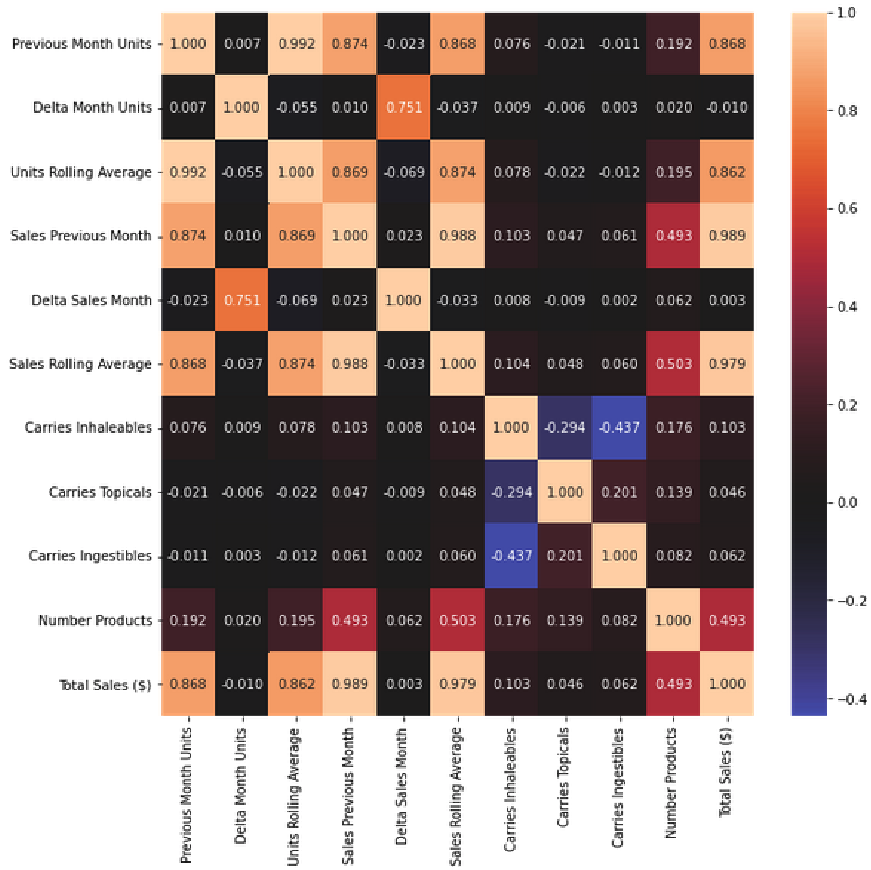
10) Cross-Validate your training results - Employ K-Fold Cross-validation to your training regimen for both ensemble and single regression models.

In order to cross-validate our data, we used the GridSearchCV function from sklearn. This allowed us simultaneously conduct k-fold cross validation on our dataset while optimising our hyperparameters.

Results:

<u>3) Run some basic statistics on your variables including correlations with labels and report findings.</u>

______Correlation Heatmap



The image above is a heatmap of our findings after checking for correlations between the features of our feature engineered dataframe. We observed strong correlations between our Units and Sales features, which is well within our expectations. Similarly, we observed a moderately strong correlation between “Delta Month Units” and “Delta Sales Month”.

Interestingly enough, we were also able to find moderate negative correlations between a brand carrying inhalables and the same brand carrying topicals/ingestibles. This suggests that there is a tendency for brands to specialise in either inhalables or topicals/ingestibles. It is also worth noting that there is a small positive correlation between a brand carrying topicals and ingestibles, which supports our earlier assessment.

<u>7) Implement a basic Linear Regression predictive model.</u>

We implemented a simple linear regression model in order to get a preliminary look at the predictive power of our data. This required us to develop a simple pipeline that scaled our numerical features and one hot encoded our categorical features. This model produced the following results:

- Train MAE Linear Regression = 85274.44529647274
- Test MAE Linear Regression = 93401.31958423468
- Test r^2 = 0.9726778732765677
- Test variance = 0.9726789176934861

<u>9) Employ an ensemble method to your predictive model exercise</u>

For our ensemble method, we implemented a random forest method for the reasons outlined in the methodology section. Our random forest model produced the following results:

- Train mae = 68495.38013224969
- Test mae = 91873.989375508

Our random forest method was able to decrease the variance of our final model by increasing its bias. This resulted in a 2% decrease in MAE score from our linear regression model.

<u>11) Employ a GridSearch method to optimise your parameters.</u>

In order to carry out cross validation and hyperparameter optimisation we utilised the GridSearchCV function from sklearn’s model_selection package. Employing GridSearch yielded the following results for our Random Forest Model:

- Best Negative MAE -102066.13128689004

For our parameter selection, we chose to do set cv=10 to half 10-fold cross validation and we scored our models based on negative MAE. After implementing GridSearchCV on our Random Forest model, we observed that the GridSearchCV function outputted the following optimised hyperparameters:

- Optimal Hyperparameters {'max_depth': 10, 'n_estimators': 200}

<u>12) Experiment with your own custom models and report your highest performing model.</u>

In order to assess the performance of our LinReg and Random Forest models, we decided to create 3 more “custom” models, namely, PCA LinReg, SVM, and Bagging. After building our models, we used GridSearchCV to cross validate our data and optimise the hyperparameters of our models. The following results were observed:

- LinReg - Best Negative MAE: -87737.11689503728
- PCA LinReg - Best Negative MAE: -90316.56950190946
- Random Forest - Best Negative MAE: -102066.13128689004

- SVM - Best Negative MAE: -412120.8287665685
- Bagging - Best Negative MAE: -406754.90988141636

From these results, we can see that our best performing model is LinReg as it has the lowest absolute MAE.

Key Indicators of the likely success of a new product launch in the current market

From our analysis, we were unable to find any hard correlations between specific indicators and predicted sales. We did however notice a trend that larger and more established companies tended to continue to perform well with much less volatility in their sales metrics.

Discussion:

From our data analysis, we have found Linear Regression (post GridSearch optimisation) to be our best performing model. While this was a surprising result, we hypothesize that the features we created during feature creation as well as the features that we retained after data cleaning resulted in linearly separable data. This would explain why Linear Regression was our best performing model.

In order to better leverage our findings, we recommend that Cookies continue to collect sales data from the cannabis industry within and outside of California. We believe that our model is generalisable to most datasets within the US, so continuing to feed it more (relevant) data should be able to further maximise its performance. We also believe that having more data points for each single brand (having a longer time series) would increase the predictive power of our model.

For next steps pertaining to analytic work, we recommend that Cookies attempt to create new models using different methods. While we have tested 5 different models in this report, it is by no means exhaustive and though we are confident in our results, we cannot rule out that there may exist a better performing model given this dataset.

Conclusion:

To accurately predict future sales of brands, our group went through many different iterations of models and parameters. Initially, we merged the data from the 4 datasets to better visualise our problem space. Then we began to create time series features by computing and inserting categorical features from the brand details dataset, based on method of ingestion.

After feature creation, we imputed our missing or NaN values through median imputation. This allowed us to prevent issues associated with our incomplete dataset and also to impute values that would not be affected by outliers. Next, we visualized the data with a covariance heatmap to get an understanding of which features were heavily related.

From here we consolidated all of our findings and clean data and passed it to many different models, and with each model we gained a better insight into how the data was structured, and we created cross features. After running through a linear regression we also observed the p and t values in the data. Finally we ran our dataset through 5 different industry-standard models and conducted GridSearchCV to cross-validate and optimize our hyper parameters.

Overall, we found that our Linear Regression model was the best performing model given our dataset and our own feature creation. We observed a trend that established Cannabis companies, such as Cookies, tend to have more consistent sales and outperform smaller businesses. We recommend that Cookies continue to aggregate sales data both within California and throughout the U.S., and we believe that other model methods should still be explored.

project 3

william randall & brian chang

In [1]:
```
from itertools import combinations
from sklearn import metrics
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.cluster import KMeans
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import Lasso, LinearRegression, LogisticRegression
from sklearn.metrics import confusion_matrix, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score, GridSearc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.svm import SVR
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import sklearn.metrics.cluster as smc
```

In [2]:
```
brandTotalSales_df = pd.read_csv(os.path.join('data','BrandTotalSales.csv'))
brandTotalUnits_df = pd.read_csv(os.path.join('data','BrandTotalUnits.csv'))
brandAverageRetailPrice_df = pd.read_csv(os.path.join('data','BrandAverageRetail
brandDetails_df = pd.read_csv(os.path.join('data','BrandDetails.csv'))
```

brand total sales

In [3]:
```
brandTotalSales_df.head(5)
```

Out[3]:

	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

In [4]:
```
brandTotalSales_df.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
```

In [5]:

```
brandTotalSales_df.describe()
```

Out[5]:

	Months	Brand	Total Sales ($)
count	25279	25279	25279
unique	37	1627	25277
top	05/2021	Lift Ticket Laboratories	0
freq	848	37	3

brand total units

In [6]:

```
brandTotalUnits_df.head(5)
```

Out[6]:

	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

In [7]:

```
brandTotalUnits_df.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
```

In [8]:

```
brandTotalUnits_df.describe()
```

Out[8]:

	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

brand average retail price

In [9]:

```
brandAverageRetailPrice_df.head(5)
```

Out[9]:

	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

In [10]:

```
brandAverageRetailPrice_df.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
```

In [11]:

```
brandAverageRetailPrice_df.describe()
```

Out[11]:

	ARP	vs. Prior Period
count	25279.000000	24499.000000
mean	22.679732	-0.065028
std	19.802724	0.388923

	ARP	vs. Prior Period
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

brand details

In [12]:

```
brandDetails_df.head(5)
```

Out[12]:

	State	Channel	Category L1	Category L2	Category L3	Category L4	Category L5	Brand
0	California	Licensed	Inhaleables	Flower	Hybrid	NaN	NaN	#BlackSeries
1	California	Licensed	Inhaleables	Flower	Hybrid	NaN	NaN	#BlackSeries
2	California	Licensed	Inhaleables	Flower	Sativa Dominant	NaN	NaN	#BlackSeries
3	California	Licensed	Inhaleables	Flower	Sativa Dominant	NaN	NaN	#BlackSeries
4	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates	Wax	NaN	101 Cannabis Co.

5 rows × 25 columns

In [13]:

```
brandDetails_df.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
```

In [14]:

```
brandDetails_df.describe()
```

Out[14]:

	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

data cleaning

fix months

In [15]:

```
# make months a date time
brandTotalSales_df['Months'] = pd.to_datetime(brandTotalSales_df['Months'])
brandTotalUnits_df['Months'] = pd.to_datetime(brandTotalUnits_df['Months'])
brandAverageRetailPrice_df['Months'] = pd.to_datetime(brandAverageRetailPrice_df
```

fix total sales

In [16]:
```
# make total sales floats
brandTotalUnits_df['Total Units'] = pd.to_numeric(brandTotalUnits_df['Total Unit
```

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

Out[17]:

	Brands	Months	Total Units	vs. Prior Period
0	#BlackSeries	2020-08-01	1616.3390	NaN
1	#BlackSeries	2020-09-01	NaN	-1.000000
2	#BlackSeries	2021-01-01	715.5328	NaN
3	#BlackSeries	2021-02-01	766.6691	0.071466
4	#BlackSeries	2021-03-01	NaN	-1.000000

In [18]:
```
brandTotalUnits_df.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  datetime64[ns]
 2   Total Units       25712 non-null  float64
 3   vs. Prior Period  24935 non-null  float64
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 865.3+ KB
```

In [19]:
```
# make Total Sales ($) floats
brandTotalSales_df['Total Sales ($)'] = pd.to_numeric(brandTotalSales_df['Total
```

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

Out[20]:

	Months	Brand	Total Sales ($)
0	2018-09-01	10x Infused	1711.334
1	2018-09-01	1964 Supply Co.	25475.210
2	2018-09-01	3 Bros Grow	120153.600
3	2018-09-01	3 Leaf	6063.529
4	2018-09-01	350 Fire	631510.000

In [21]:
```
brandTotalSales_df.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  datetime64[ns]
 1   Brand            25279 non-null  object
 2   Total Sales ($)  25279 non-null  float64
dtypes: datetime64[ns](1), float64(1), object(1)
memory usage: 592.6+ KB
```

brand details categorical features

In [22]:

```
print('brand details\n')
for i, column in enumerate(brandDetails_df.columns):
    print(column, '-'*5 + '>', len(list(brandDetails_df[column].unique())))
```

```
brand details

State -----> 1
Channel -----> 1
Category L1 -----> 5
Category L2 -----> 13
Category L3 -----> 54
Category L4 -----> 72
Category L5 -----> 39
Brand -----> 1123
Product Description -----> 32608
Total Sales ($) -----> 133144
Total Units -----> 96910
ARP -----> 131319
Flavor -----> 496
Items Per Pack -----> 33
Item Weight -----> 70
Total THC -----> 136
Total CBD -----> 149
Contains CBD -----> 2
Pax Filter -----> 3
Strain -----> 5825
Is Flavored -----> 3
Mood Effect -----> 2
Generic Vendor -----> 2
Generic Items -----> 2
$5 Price Increment -----> 22
```

In [23]:

```
print('brand details\n')
for i, column in enumerate(brandDetails_df.columns):
    uniq = list(brandDetails_df[column].unique())
    if len(uniq) > 100:
        uniq = "too big: " + str(len(uniq))
    print(column, '\n', uniq, '\n')
```

```
brand details

State
 ['California']

Channel
 ['Licensed']
```

```
Category L1
 ['Inhaleables', 'Topicals', 'Ingestibles', 'All Accessories', 'Other Cannabis']

Category L2
 ['Flower', 'Concentrates', 'Pre-Rolled', 'Topicals', 'Edibles', 'Devices', 'Sub
linguals', 'Other Cannabis', 'Accessories', 'Non Infused Food', 'Apparel', 'Grow
Supplies', 'Shake/Trim/Lite']

Category L3
 ['Hybrid', 'Sativa Dominant', 'Dabbable Concentrates', 'Infused Pre-Rolled', 'P
re-Rolled', 'Vape', 'Other Topicals', 'Indica', 'Sativa', 'Sativa Leaning', 'Inf
used Foods', 'Indica Dominant', 'Indica Leaning', 'Candy', 'Vaporizers', 'Pipe',
'Water Pipe', 'Rolling Papers', 'Pills', 'Tinctures', 'Other', 'Beverages', 'Spr
ay', 'Grinder', 'Storage Device', 'Creams', 'Massage Oil', 'Balms/Salves', 'Bund
les/Collections', 'Lighter', 'Culinary', 'Chocolates', 'Lotions', 'Suppositorie
s', 'Dissolvable', 'Plants', 'Gum', nan, 'Other Edibles', 'Rolling Machine', 'Pe
t Products', 'Cleaner', 'Pre-Loaded', 'Lubricants', 'Soap', 'Bowl', 'Patches',
'Clothing', 'Accessories', 'Pins', 'Sticker', 'Candles', 'Jewlery', 'Lip Balm']

Category L4
 [nan, 'Wax', 'Live Resin', 'Rosin', 'Vape Disposable', 'Shatter', 'Spreads', 'R
SO', 'Gummie Candy', 'Vape Cartridge', 'Bubble Hash', 'Unspecified Concentrates/
Other', 'Pen/ Hand Held', 'Other', 'Chillum', 'Oil/ Hash Dab Rig', 'Bong', 'Bubb
ler', 'Spoon', 'Caviar', 'Softgel', 'Dropper', 'Drinks', 'Oral Spray', 'Other Be
verages', 'Capsule', 'Kit', 'Shots', 'Humidor', 'Mints', 'Crystalline', 'Distill
ate', 'Powdered Mix', 'Baked Goods', 'Oils', 'Caramel Candy', 'Disposable', 'Oth
er Culinary', 'Chocolates', 'Tablet', 'Syrup', 'Hard Candy', 'Tea', 'Taffy', 'Or
ally Disolvable Strips', 'Butter', 'Kief', 'Clones', 'Other Infused Foods', 'Oth
er Candy', 'Seeds', 'Hash', 'Tablets', 'Other Tinctures', 'Oil', 'Honey', 'Coffe
e Products', 'Torch', 'Flower', 'Jar', 'Inhaler', 'Butane', 'Tshirts', 'Accessor
y', 'Hat', 'Bandana', 'Table Top', 'Hoodie', 'Pants', 'Lockable', 'Soft', 'Dab R
ig']

Category L5
 [nan, 'Oil Disposable', 'Live Resin Cartridge', 'Rechargeable Battery', 'Kit',
'Hash Dab Rig', 'Accessory', 'Oil Cartridge', 'Distillate Cartridge', 'Distillat
e Disposable', 'Carbonated Drinks', 'Other Beverages', 'Replacement Parts and Ac
cessories', 'Shots', 'Hard Mints', 'Live Resin Disposable', 'Disposable Batter
y', 'Powdered Mix', 'Granola', 'Other Baked Goods', 'Chocolate Bars', 'Cookies',
'Chocolate Pieces', 'Rosin Cartridge', 'Noncarbonated Drinks', 'Lozenges', 'Dry
Tea', 'Chewable Mints', 'Unspecified Cartridge', 'Tea Drink', 'Brownies', 'Other
Chocolates', 'Lollipop', 'Coffee Drink', 'Flower', 'Flower and Concentrate', 'Da
b Rig', 'Concentrate', 'Tools']

Brand
 too big: 1123

Product Description
 too big: 32608

Total Sales ($)
 too big: 133144

Total Units
 too big: 96910

ARP
 too big: 131319
```

```
Flavor
 too big: 496

Items Per Pack
 [0, 1, 5, 10, 20, 30, 2, 4, 1000, 12, 6, 3, 8, 7, 15, 120, 200, 50, 100, 40, 6
0, 24, 18, 17, 25, 33, 74, 14, 9, 160, 16, 48, 47]

Item Weight
 [nan, '1.00', '2.00', '500mg', '0.70', '3.00', '0.30', '1000mg', '300mg', '0.7
5', '1.75', '0.85', '0.50', '550mg', '1050mg', '1.30', '6.00', '2.50', '2.75',
'0.80', '1.25', '1.20', '1.50', '0.35', '1.40', '10000mg', '1.70', '600mg', '0.6
0', '3.50', '3.30', '9.00', '400mg', '750mg', '1100mg', '1500mg', '900mg', '250m
g', '200mg', '185mg', '125mg', '0.65', '2.40', '1.60', '0.38', '2.20', '0.66',
'0.37', '2200mg', '0.25', '0.33', '0.90', '8.00', '2.30', '3.20', '5.00', '7.0
0', '1.33', '1.10', '0.40', '3.85', '2.80', '2.15', '1.16', '0.20', '0.42', '0.5
8', '360mg', '330mg', '350mg']

Total THC
 too big: 136

Total CBD
 too big: 149

Contains CBD
 ['THC Only', 'Contains CBD']

Pax Filter
 [nan, 'Not Pax', 'Pax']

Strain
 too big: 5825

Is Flavored
 [nan, 'Not Flavored', 'Flavored']

Mood Effect
 ['Not Mood Specific', 'Mood Specific']

Generic Vendor
 ['Non-Generic Vendors', 'Generic Vendors']

Generic Items
 ['Non-Generic Items', 'Generic Items']

$5 Price Increment
 ['$10.00 to $14.99', '$15.00 to $19.99', '$35.00 to $39.99', '$30.00 to $34.9
9', '$20.00 to $24.99', '$25.00 to $29.99', '$45.00 to $49.99', '$40.00 to $44.9
9', '$50.00 to $54.99', '$60.00 to $64.99', 'Over $100', '$00.00 to $4.99', '$0
5.00 to $9.99', '$70.00 to $74.99', '$65.00 to $69.99', '$55.00 to $59.99', '$7
5.00 to $79.99', '$80.00 to $84.99', '$85.00 to $89.99', '$90.00 to $94.99', '$9
5.00 to $99.99', 'Zero Value']
```

null values

In [24]:

```
for i, column in enumerate(brandDetails_df.columns):
    notNullL = len(brandDetails_df.loc[brandDetails_df[column].notnull()])
```

```
    l = len(brandDetails_df)
    percent = notNullL/l
    s = percent
    if percent == 1:
        s = "no null vals"
    print(column, '-'*20+'>', s)
```

```
State --------------------> no null vals
Channel --------------------> no null vals
Category L1 --------------------> no null vals
Category L2 --------------------> no null vals
Category L3 --------------------> 0.9949509232498948
Category L4 --------------------> 0.7078226201397463
Category L5 --------------------> 0.3458134738613711
Brand --------------------> no null vals
Product Description --------------------> no null vals
Total Sales ($) --------------------> no null vals
Total Units --------------------> no null vals
ARP --------------------> no null vals
Flavor --------------------> 0.053849921021955204
Items Per Pack --------------------> no null vals
Item Weight --------------------> 0.44458086455092877
Total THC --------------------> no null vals
Total CBD --------------------> no null vals
Contains CBD --------------------> no null vals
Pax Filter --------------------> 0.3055726080688661
Strain --------------------> 0.7976368665374508
Is Flavored --------------------> 0.07785372852245528
Mood Effect --------------------> no null vals
Generic Vendor --------------------> no null vals
Generic Items --------------------> no null vals
$5 Price Increment --------------------> no null vals
```

time series feature engineering

In [25]:

```
brands = brandTotalUnits_df['Brands'].unique()
```

In [26]:

```
brands
```

Out[26]:

```
array(['#BlackSeries', '101 Cannabis Co.', '10x Infused', ..., 'Zlixir',
       'Zoma', 'Zuma Topicals'], dtype=object)
```

create timeseries features

1. Previous Month Units
2. Delta Month Units (diff between last month and month before that)
3. Units Rolling Average (past 3 months)
4. Sales Previous Month
5. Delta Sales Month
6. Sales Rolling Average
7. Carries Inhaleables
8. Carries Topicals

9. Carries Ingestibles
10. Number Products

In [27]:

```
l = len(brands)
tempDf = []
featureEngineeredDf = pd.DataFrame()
for i, brand in enumerate(brands):
    print(f'{i}/{l-1}',end='\r')
    newDf = brandTotalUnits_df.where(brandTotalUnits_df.Brands==brand)
    # get units for last month
    newDf.loc[:,'Previous Month Units'] = newDf.loc[:,'Total Units'].shift(1)
    # get delta units for last month vs this month
    newDf.loc[:,'Delta Month Units'] = newDf.loc[:,'Total Units'].shift(1) - new
    # get rolling average of units
    newDf.loc[:,'Units Rolling Average'] = (newDf.loc[:,'Total Units'].shift(1)
                                           newDf.loc[:,'Total Units'].shift(2) +
                                           newDf.loc[:,'Total Units'].shift(3))
    # get sales
    newDf = newDf.merge( \
                        brandTotalSales_df[brandTotalSales_df.Brand == brand], \
                        left_on='Months',right_on='Months')
    # get price
    newDf = newDf.merge( \
                        brandAverageRetailPrice_df[brandAverageRetailPrice_df.Br
                        left_on='Months',right_on='Months')
    # drop extra
    newDf = newDf.drop(['Brands_x'], 1)
    newDf = newDf.drop(['Brands_y'], 1)
    # get last month's average retail price
    newDf.loc[:,'Sales Previous Month'] = newDf.loc[:,'Total Sales ($)'].shift(1
    # get delta month's average retial price
    newDf.loc[:,'Delta Sales Month'] = newDf.loc[:,'Total Sales ($)'].shift(1) -
    # get rolling average of sales
    newDf.loc[:,'Sales Rolling Average'] = (newDf.loc[:,'Total Sales ($)'].shift
                                            newDf.loc[:,'Total Sales ($)'].shift
                                            newDf.loc[:,'Total Sales ($)'].shift

    # make bools
    carriesInhaleables = 0
    carriesTopicals = 0
    carriesIngestibles = 0
    if 'Inhaleables' in brandDetails_df[brandDetails_df.Brand == brand]['Categor
        carriesInhaleables = 1
    if 'Topicals' in brandDetails_df[brandDetails_df.Brand == brand]['Category L
        carriesTopicals = 1
    if 'Ingestibles' in brandDetails_df[brandDetails_df.Brand == brand]['Categor
        carriesIngestibles = 1
    newDf['Carries Inhaleables'] = carriesInhaleables
    newDf['Carries Topicals'] = carriesTopicals
    newDf['Carries Ingestibles'] = carriesIngestibles

    # how many products the brand has
    newDf['Number Products'] = len(brandDetails_df.loc[brandDetails_df['Brand']

    # append to temp dataframe
    tempDf.append(newDf)

featureEngineeredDf = pd.concat(tempDf)
```

```
3/1639
<ipython-input-27-20d9ff8bb8ba>:24: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  newDf = newDf.drop(['Brands_x'], 1)
<ipython-input-27-20d9ff8bb8ba>:25: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  newDf = newDf.drop(['Brands_y'], 1)
1639/1639
```

In [28]:

```
featureEngineeredDf.head(20)
```

Out[28]:

	Months	Total Units	vs. Prior Period_x	Previous Month Units	Delta Month Units	Units Rolling Average	Brand	Total Sales ($)
0	2020-08-01	1616.3390	NaN	NaN	NaN	NaN	#BlackSeries	25352.130
1	2021-01-01	715.5328	NaN	NaN	NaN	NaN	#BlackSeries	9739.423
2	2021-02-01	766.6691	0.071466	715.5328	NaN	NaN	#BlackSeries	9102.802
0	2019-11-01	131.0677	NaN	NaN	NaN	NaN	101 Cannabis Co.	4465.040
1	2020-01-01	345.4134	NaN	NaN	NaN	NaN	101 Cannabis Co.	11790.660
2	2020-02-01	696.6584	1.016883	345.4134	NaN	NaN	101 Cannabis Co.	20266.760
3	2020-03-01	943.3933	0.354169	696.6584	351.2450	NaN	101 Cannabis Co.	30465.470
4	2020-04-01	712.4981	-0.244750	943.3933	246.7349	661.821700	101 Cannabis Co.	23465.650
5	2020-05-01	619.8410	-0.130045	712.4981	-230.8952	784.183267	101 Cannabis Co.	21348.390
6	2020-06-01	426.1504	-0.312484	619.8410	-92.6571	758.577467	101 Cannabis Co.	14111.750
7	2020-07-01	589.7193	0.383829	426.1504	-193.6906	586.163167	101 Cannabis Co.	18948.510
8	2020-08-01	1018.5740	0.727218	589.7193	163.5689	545.236900	101 Cannabis Co.	32743.470
9	2020-09-01	1408.8500	0.383160	1018.5740	428.8547	678.147900	101 Cannabis Co.	44839.680

	Months	Total Units	vs. Prior Period_x	Previous Month Units	Delta Month Units	Units Rolling Average	Brand	Total Sales ($)
10	2020-10-01	1148.9620	-0.184468	1408.8500	390.2760	1005.714433	101 Cannabis Co.	34899.870
11	2020-11-01	447.1605	-0.610814	1148.9620	-259.8880	1192.128667	101 Cannabis Co.	15106.390
12	2020-12-01	337.9605	-0.244208	447.1605	-701.8015	1001.657500	101 Cannabis Co.	11883.010
13	2021-01-01	250.2320	-0.259582	337.9605	-109.2000	644.694333	101 Cannabis Co.	8059.176
14	2021-02-01	395.8241	0.581828	250.2320	-87.7285	345.117667	101 Cannabis Co.	13712.770
15	2021-03-01	686.8574	0.735259	395.8241	145.5921	328.005533	101 Cannabis Co.	24347.900
16	2021-04-01	624.6255	-0.090604	686.8574	291.0333	444.304500	101 Cannabis Co.	20784.920

In [29]:

```
featureEngineeredDf.columns
```

Out[29]:

```
Index(['Months', 'Total Units', 'vs. Prior Period_x', 'Previous Month Units',
       'Delta Month Units', 'Units Rolling Average', 'Brand',
       'Total Sales ($)', 'ARP', 'vs. Prior Period_y', 'Sales Previous Month',
       'Delta Sales Month', 'Sales Rolling Average', 'Carries Inhaleables',
       'Carries Topicals', 'Carries Ingestibles', 'Number Products'],
      dtype='object')
```

create month column

In [30]:

```
# make a month column
getMonth = lambda val: str(val).split('-')[1]
featureEngineeredDf.insert(2, "Month", featureEngineeredDf["Months"].apply(getMo
```

In [31]:

```
featureEngineeredDf.columns
```

Out[31]:

```
Index(['Months', 'Total Units', 'Month', 'vs. Prior Period_x',
       'Previous Month Units', 'Delta Month Units', 'Units Rolling Average',
       'Brand', 'Total Sales ($)', 'ARP', 'vs. Prior Period_y',
       'Sales Previous Month', 'Delta Sales Month', 'Sales Rolling Average',
       'Carries Inhaleables', 'Carries Topicals', 'Carries Ingestibles',
       'Number Products'],
      dtype='object')
```

data cleaning

drop columns which are not useful

In [32]:
```
# drop columns which are not useful
columnsGettingDropped = ['Months','vs. Prior Period_x', 'vs. Prior Period_y', 'A
for column in columnsGettingDropped:
    featureEngineeredDf = featureEngineeredDf.drop(column, axis=1)
```

find nans

In [33]:
```
# find columns with nans
print(featureEngineeredDf.columns[featureEngineeredDf.isna().any()])
```

```
Index(['Previous Month Units', 'Delta Month Units', 'Units Rolling Average',
       'Sales Previous Month', 'Delta Sales Month', 'Sales Rolling Average'],
      dtype='object')
```

In [34]:
```
brands = list(featureEngineeredDf['Brand'].unique())
```

In [35]:
```
len(brands)
```

Out[35]:
```
1627
```

drop certain brands

1. if they have less than 6 months of data
2. if they have all nans
3. if they have 0 products

In [36]:
```
l = len(brands)
# drop certain brands
for i,brand in enumerate(brands):
    print(f'{i}/{l-1}',end='\r')
    tempDf = featureEngineeredDf[featureEngineeredDf.Brand == brand]
    lTempDf = len(tempDf)
    if lTempDf <= 6:
        # drop it if it has less than 6 months of data
        featureEngineeredDf = featureEngineeredDf[featureEngineeredDf.Brand != b
        continue
    # if they have all NaN
    for column in featureEngineeredDf.columns[featureEngineeredDf.isna().any()].
        if len(list(tempDf[column].unique())) == 1:
            featureEngineeredDf = featureEngineeredDf.loc[featureEngineeredDf.Br

# if they have 0 products
featureEngineeredDf = featureEngineeredDf.loc[featureEngineeredDf['Number Produc
```

```
1626/1626
```

data impution

median impution

In [37]:

```
# impute the nans with median values
brands = list(featureEngineeredDf['Brand'].unique())
l = len(brands)
for i, brand in enumerate(brands):
    print(f'{i}/{l-1}',end='\r')
    for column in featureEngineeredDf.columns[featureEngineeredDf.isna().any()].
        median = featureEngineeredDf.loc[featureEngineeredDf.Brand==brand,column
        featureEngineeredDf.loc[featureEngineeredDf['Brand'] == brand, column] =
```

```
857/857
```

In [38]:

```
# make sure there are no nulls
featureEngineeredDf[featureEngineeredDf.isna().any(axis=1)].head()
```

Out[38]:

Month	Previous Month Units	Delta Month Units	Units Rolling Average	Brand	Total Sales ($)	Sales Previous Month	Delta Sales Month	Sales Rolling Average	Carries Inhaleables	Carri Topica

In [39]:

```
featureEngineeredDf.info()
```

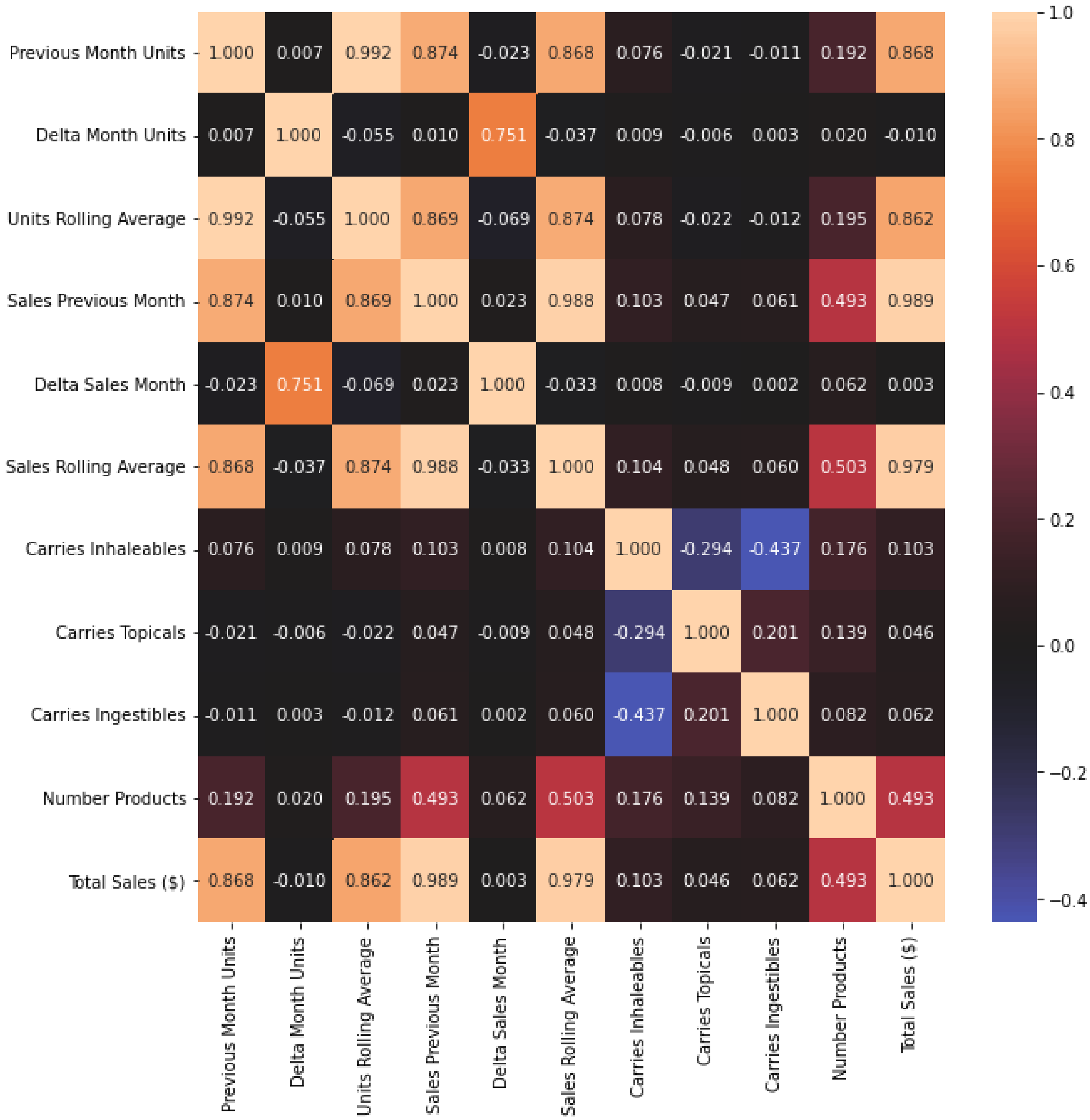
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20549 entries, 0 to 26
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Month                  20549 non-null  object
 1   Previous Month Units   20549 non-null  float64
 2   Delta Month Units      20549 non-null  float64
 3   Units Rolling Average  20549 non-null  float64
 4   Brand                  20549 non-null  object
 5   Total Sales ($)        20549 non-null  float64
 6   Sales Previous Month   20549 non-null  float64
 7   Delta Sales Month      20549 non-null  float64
 8   Sales Rolling Average  20549 non-null  float64
 9   Carries Inhaleables    20549 non-null  int64
 10  Carries Topicals       20549 non-null  int64
 11  Carries Ingestibles    20549 non-null  int64
 12  Number Products        20549 non-null  int64
dtypes: float64(7), int64(4), object(2)
memory usage: 2.2+ MB
```

correlation matrix

In [40]:

```
numericalFeatures = ['Previous Month Units',
                     'Delta Month Units',
                     'Units Rolling Average',
                     'Sales Previous Month',
```

```
                     'Delta Sales Month',
                     'Sales Rolling Average',
                     'Carries Inhaleables',
                     'Carries Topicals',
                     'Carries Ingestibles',
                     'Number Products',
                     'Total Sales ($)']
correlationMatrix = featureEngineeredDf[numericalFeatures].corr()
plt.figure(figsize=(10,10))
ax = sns.heatmap(correlationMatrix, center=0, annot=True, fmt='.3f')
```



linear regression

In [41]:

```
numericalFeatures = ['Previous Month Units',
                     'Delta Month Units',
                     'Units Rolling Average',
```

```
                      'Sales Previous Month',
                      'Delta Sales Month',
                      'Sales Rolling Average',
                      'Carries Inhaleables',
                      'Carries Topicals',
                      'Carries Ingestibles',
                      'Number Products']
categoricalFeatures = ['Month']

allPermutationsOfFeatures = sum([list(map(list,combinations(numericalFeatures, i
l = len(allPermutationsOfFeatures)
bestFeatures = []
bestTrainMae = float('inf')
bestTestMae = float('inf')
for i,features in enumerate(allPermutationsOfFeatures):
    print(f'{i}/{l-1}',end='\r')
    # if it is less than 3 features just skip
    if len(features) <= 3:
        continue

    featuresToDrop = list(set(list(featureEngineeredDf.columns)) - set(features)
    tempDf = featureEngineeredDf.drop(featuresToDrop, axis=1).copy(deep=True)

    # data pipeline for numerical data
    num_pipeline = Pipeline([
        ('std_scaler', StandardScaler()),
    ])

    # full data pipeline
    full_pipeline = ColumnTransformer([
        ('num', num_pipeline, features),
        ('cat', OneHotEncoder(), categoricalFeatures),
    ])

    preparedData = full_pipeline.fit_transform(tempDf)

    # train test split
    labels = featureEngineeredDf['Total Sales ($)'].copy()
    train_set, test_set, train_label, test_label = train_test_split(preparedData

    # perform linear regression
    linearRegression = LinearRegression()
    linearRegression.fit(train_set,train_label)
    trainPrediction = linearRegression.predict(train_set)
    testPrediction = linearRegression.predict(test_set)

    # get mean absolute error
    trainMae = mean_absolute_error(train_label, trainPrediction)
    testMae = mean_absolute_error(test_label, testPrediction)

    # append to list of best features
    if testMae < bestTestMae:
        bestFeatures = features
        bestTrainMae = trainMae
        bestTestMae = testMae
    elif testMae == bestTestMae:
        bestFeatures.append(features)
```

```
1023/1023
```

In [42]:
```
print('best features', bestFeatures)
print('best train mae', bestTrainMae)
print('best test mae', bestTestMae)
```

```
best features ['Sales Previous Month', 'Sales Rolling Average', 'Carries Inhalea
bles', 'Carries Topicals']
best train mae 85099.48444282636
best test mae 93150.0571973241
```

feature cross

In [43]:
```
# feature cross
featureEngineeredDf['Units Average Per Product'] = featureEngineeredDf['Units Ro
```

Linear Regression

In [44]:
```
featureEngineeredDf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20549 entries, 0 to 26
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Month                      20549 non-null  object
 1   Previous Month Units       20549 non-null  float64
 2   Delta Month Units          20549 non-null  float64
 3   Units Rolling Average      20549 non-null  float64
 4   Brand                      20549 non-null  object
 5   Total Sales ($)            20549 non-null  float64
 6   Sales Previous Month       20549 non-null  float64
 7   Delta Sales Month          20549 non-null  float64
 8   Sales Rolling Average      20549 non-null  float64
 9   Carries Inhaleables        20549 non-null  int64
 10  Carries Topicals           20549 non-null  int64
 11  Carries Ingestibles        20549 non-null  int64
 12  Number Products            20549 non-null  int64
 13  Units Average Per Product  20549 non-null  float64
dtypes: float64(8), int64(4), object(2)
memory usage: 2.4+ MB
```

In [45]:
```
numericalFeatures = ['Delta Month Units',
                     'Sales Previous Month',
                     'Sales Rolling Average',
                     'Carries Inhaleables',
                     'Carries Ingestibles']
augmentedFeature = ['Units Average Per Product']
categoricalFeatures = ['Month']

# drop features
featuresToDrop = list(set(featureEngineeredDf.columns) - set(augmentedFeature) -
tempDf = featureEngineeredDf.drop(featuresToDrop, axis=1).copy(deep=True)

# pipeline
num_pipeline = Pipeline([
```

```
    ('std_scaler', StandardScaler()),
])

full_pipeline = ColumnTransformer([
    ('num', num_pipeline, numericalFeatures),
    ('cat', OneHotEncoder(), categoricalFeatures),
])

preparedData = full_pipeline.fit_transform(tempDf)

# train test split
labels = featureEngineeredDf['Total Sales ($)'].copy()
train_set, test_set, train_label, test_label = train_test_split(preparedData,lab

# linear regresssion
linearRegression = LinearRegression()
linearRegression.fit(train_set, train_label)

trainPrediction = linearRegression.predict(train_set)
testPrediction = linearRegression.predict(test_set)

# get mean absolute error
trainMae = mean_absolute_error(train_label, trainPrediction)
testMae = mean_absolute_error(test_label, testPrediction)

print('train mae Linear Regression', trainMae)
print('test mae Linear Regression', testMae)
print('test r^2', metrics.r2_score(test_label, testPrediction))
print('test variance',metrics.explained_variance_score(test_label, testPredictio
```

```
train mae Linear Regression 85274.44529647274
test mae Linear Regression 93401.31958423468
test r^2 0.9726778732765677
test variance 0.9726789176934861
```

In [47]:

```
import statsmodels.api as sm
stats = sm.OLS(labels, preparedData)
resultStats = stats.fit()
print(resultStats.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Total Sales ($)   R-squared:                       0.978
Model:                            OLS   Adj. R-squared:                  0.978
Method:                 Least Squares   F-statistic:                 5.729e+04
Date:                Sun, 05 Dec 2021   Prob (F-statistic):               0.00
Time:                        10:46:37   Log-Likelihood:            -2.8540e+05
No. Observations:               20549   AIC:                         5.708e+05
Df Residuals:                   20532   BIC:                         5.710e+05
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1         -2.677e+04   1910.811    -14.008      0.000   -3.05e+04    -2.3e+04
x2          1.591e+06   1.23e+04    129.289      0.000    1.57e+06    1.62e+06
x3          1.496e+05   1.23e+04     12.148      0.000    1.26e+05    1.74e+05
x4          3210.9605   2042.907      1.572      0.116    -793.300    7215.220
x5          3779.1611   2035.132      1.857      0.063    -209.859    7768.181
```

```
x6          4.785e+05   6475.478    73.899      0.000     4.66e+05    4.91e+05
x7          4.674e+05   6387.615    73.179      0.000     4.55e+05     4.8e+05
x8          5.512e+05   6339.055    86.957      0.000     5.39e+05    5.64e+05
x9          4.849e+05   6301.777    76.943      0.000     4.73e+05    4.97e+05
x10         5.119e+05   6190.895    82.681      0.000        5e+05    5.24e+05
x11         4.799e+05   6147.577    78.065      0.000     4.68e+05    4.92e+05
x12         5.224e+05   6158.565    84.820      0.000      5.1e+05    5.34e+05
x13         4.936e+05   6124.489    80.602      0.000     4.82e+05    5.06e+05
x14         4.641e+05   5705.697    81.336      0.000     4.53e+05    4.75e+05
x15         4.678e+05   6789.384    68.896      0.000     4.54e+05    4.81e+05
x16         4.704e+05   6625.322    71.006      0.000     4.57e+05    4.83e+05
x17         5.067e+05   6538.853    77.486      0.000     4.94e+05    5.19e+05
==============================================================================
Omnibus:                    18116.401   Durbin-Watson:                   2.004
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        270602261.814
Skew:                           2.520   Prob(JB):                         0.00
Kurtosis:                     565.158   Cond. No.                         13.6
==============================================================================

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

linear regression with cross validation and grid search

In [48]:

```
labels = np.array(labels)
fit_intercept = ['True', 'False']
positive = ['True', 'False']

parameter_grid = dict (
    fit_intercept = fit_intercept,
    positive = positive
)
linearRegression = LinearRegression()

gridSearchCV = GridSearchCV(estimator=linearRegression,
                            param_grid=parameter_grid,
                            scoring='neg_mean_absolute_error',
                            cv=10)

gridResult = gridSearchCV.fit(preparedData, labels)
```

In [49]:

```
print('best negative mae', gridResult.best_score_)
print('best hyperparameters', gridResult.best_params_)
```

```
best negative mae -87737.11689503728
best hyperparameters {'fit_intercept': 'True', 'positive': 'True'}
```

PCA

In [50]:

```
components = ['a','b','c','d']
pca = PCA(n_components=len(components))
```

```
principleComponents = pca.fit_transform(preparedData)
pcaDf = pd.DataFrame(data=principleComponents,columns=components)

# train test split
labels = featureEngineeredDf['Total Sales ($)'].copy()
train_set, test_set, train_label, test_label = train_test_split(principleCompone

#linear regression
linearRegression.fit(train_set, train_label)

trainPrediction = linearRegression.predict(train_set)
testPrediction = linearRegression.predict(test_set)

trainMae = mean_absolute_error(train_label,trainPrediction)
testMae = mean_absolute_error(test_label, testPrediction)
```

In [51]:

```
print('train mae', trainMae)
print('test mae', testMae)
print('test r^2', metrics.r2_score(test_label, testPrediction))
print('test variance',metrics.explained_variance_score(test_label, testPredictio
```

```
train mae 86956.30569385413
test mae 96055.03777373991
test r^2 0.9685975104308667
test variance 0.9685977508341262
```

pca with cross validation through grid search

In [52]:

```
pca = PCA()

linearRegression = LinearRegression()
pipeline = Pipeline(
    steps=[('pca',pca),('lin_reg',linearRegression)]
)
param_grid = {
    'pca__n_components': [2,3,4,5,6,7,8], # confused about this
    'lin_reg__fit_intercept': ['True','False'],
    'lin_reg__positive': ['True','False']
}

gridSearchCV = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring='neg_mean_absolute_error',
    cv=10
)
gridResult = gridSearchCV.fit(preparedData, labels)
```

In [53]:

```
print('best neg mae', gridResult.best_score_)
print('best hyper params', gridResult.best_params_)
```

```
best neg mae -90316.55756935169
best hyper params {'lin_reg__fit_intercept': 'False', 'lin_reg__positive': 'Fals
e', 'pca__n_components': 4}
```

random forrest

In [54]:

```
labels = featureEngineeredDf['Total Sales ($)'].copy()
train_set,test_set,train_label,test_label = train_test_split(preparedData,labels

randomForrest = RandomForestRegressor(max_depth=8, random_state=42)
randomForrest.fit(train_set,train_label)

trainPrediction = randomForrest.predict(train_set)
testPrediction = randomForrest.predict(test_set)

trainMae = mean_absolute_error(train_label,trainPrediction)
testMae = mean_absolute_error(test_label,testPrediction)
```

In [55]:

```
print('train mae', trainMae)
print('test mae', testMae)
```

```
train mae 68495.38013224969
test mae 91873.989375508
```

random forrest with grid search

In [56]:

```
max_depth=[10]
n_estimators = [50,75, 100,125, 150,200]
n_jobs = [-1]
param_grid = dict(
    max_depth=max_depth,
    n_estimators=n_estimators,
    n_jobs=n_jobs
)

randomForrest = RandomForestRegressor()
gridSearchCV = GridSearchCV(
    estimator=randomForrest,
    param_grid=param_grid,
    scoring='neg_mean_absolute_error',
    cv=10
)

gridResult = gridSearchCV.fit(preparedData,labels)
```

In [57]:

```
print('best negative mae', gridResult.best_score_)
print('best hyperparameters', gridResult.best_params_)
```

```
best negative mae -102503.55839251845
best hyperparameters {'max_depth': 10, 'n_estimators': 200, 'n_jobs': -1}
```

cross validate svm using grid search

In [58]:

```
kernel = ['linear', 'poly']
param_grid = dict (
```

```
    kernel=kernel
)

svr = SVR()
gridSearch = GridSearchCV(
    estimator=svr,
    param_grid=param_grid,
    scoring='neg_mean_absolute_error',
    cv=10,
    n_jobs=-1)

gridResult = gridSearch.fit(preparedData,labels)
```

In [59]:

```
print('best negative mae', gridResult.best_score_)
print('best hyperparameters', gridResult.best_params_)
```

```
best negative mae -412120.8287665685
best hyperparameters {'kernel': 'poly'}
```

cross validation bagging using grid search

In [60]:

```
n_estimators = [10,15,20]
max_features = [1,2,3]

param_grid = dict(
    n_estimators=n_estimators,
    max_features=max_features
)

bagging = BaggingRegressor()
gridSearch = GridSearchCV(
    estimator=bagging,
    param_grid=param_grid,
    scoring='neg_mean_absolute_error',
    cv=10,
    n_jobs=-1)

gridResult = gridSearch.fit(preparedData,labels)
```

In [61]:

```
print('best negative mae', gridResult.best_score_)
print('best hyperparameters', gridResult.best_params_)
```

```
best negative mae -383134.07567148004
best hyperparameters {'max_features': 3, 'n_estimators': 20}
```