In [1]:
```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
```

In [2]:
```
avgPrice = pd.read_csv('data/BrandAverageRetailPrice.csv')
avgPrice=avgPrice.rename(columns={'Brands':'Brand'})
avgPrice=avgPrice.rename(columns={"vs. Prior Period":"ARP_pp"})
avgPrice.head()
```

Out[2]:

	Brand	Months	ARP	ARP_pp
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 [3]:
```
brandDetails=pd.read_csv('data/BrandDetails.csv')
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
```

In [4]:
```
brandTotalSales=pd.read_csv('data/BrandTotalSales.csv')
brandTotalSales.head()
```

Out[4]:

	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 [5]:
```
brandTotalUnits=pd.read_csv('data/BrandTotalUnits.csv')
brandTotalUnits=brandTotalUnits.rename(columns={'Brands':'Brand'})
brandTotalUnits=brandTotalUnits.rename(columns={"vs. Prior Period":"totalUnits_pp"})
brandTotalUnits.head()
```

Out[5]:

	Brand	Months	Total Units	totalUnits_pp
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 [6]:
```
prodSales=pd.read_csv('data/Top50ProductsbyTotalSales-Timeseries.csv')
prodSales.head()
```

Out[6]:

	Products	Months	Total Sales ($)
0	Flower - Strain Blends - Flower (Gram)	07/2020	22,738,489.622206017
1	Flower - Strain Blends - Flower (Gram)	03/2021	22,648,507.64839804
2	Flower - Strain Blends - Flower (Gram)	05/2021	22,338,755.88508607
3	Flower - Strain Blends - Flower (Gram)	09/2020	21,461,950.605336975
4	Flower - Strain Blends - Flower (Gram)	06/2021	21,347,569.064233065

In [7]:

```
#convert months to datetime, convert total sales to int
prodSales['Months'] = pd.to_datetime(prodSales['Months'])
prodSales['Total Sales ($)'] = prodSales['Total Sales ($)'].str.replace(',', '').astype(float)
```

In [8]:

```
#pull out only features of interest from brand details
new=brandDetails[['Product Description','Brand','Category L1', 'Category L2', 'Category L3', 'Category L4', 'Category L5',
                  'Flavor', 'Item Weight', 'Total THC', 'Total CBD', 'Pax Filter', 'Strain', 'Mood Effect',
                   'Generic Vendor', 'Generic Items']]
newBD=new.copy()
```

In [9]:

```
#convert total THC, total CBD to int
newBD['Total THC'] = newBD['Total THC'].str.replace(',', '').astype(float)
newBD['Total CBD'] = newBD['Total CBD'].str.replace(',', '').astype(float)
```

In [10]:

```
#convert Pax Filter, Mood Effect, Generic Vendor, and Generic Items to binary
def convertToBinaryPax(s):
    if s=='Pax':
        s=1
        s=float(s)
    if s=='Not Pax':
        s=0
        s=float(s)
    return(s)

def convertToBinaryMood(s):
    if type(s)==str and s=='Mood Specific':
        s=1
        s=float(s)
    if type(s)==str and s=='Not Mood Specific':
        s=0
        s=float(s)
    return(s)

def convertToBinaryVendor(s):
    if type(s)==str and s=='Generic Vendors':
        s=1
        s=float(s)
    if type(s)==str and s=='Non-Generic Vendors':
        s=0
        s=float(s)
    return(s)

def convertToBinaryItems(s):
    if type(s)==str and s=='Generic Items':
        s=1
        s=float(s)
    if type(s)==str and s=='Non-Generic Items':
        s=0
        s=float(s)
    return(s)

def convertToIntWeight(s):
    if type(s)!=float and 'mg' in s:
        s=s.replace('mg','')
        s=float(s)
        s=s/1000
    elif type(s)!=float:
        s=float(s)
    return(s)

newBD['Pax Filter'] = newBD['Pax Filter'].apply(convertToBinaryPax)
newBD['Mood Effect'] = newBD['Mood Effect'].apply(convertToBinaryMood)
newBD['Generic Vendor'] = newBD['Generic Vendor'].apply(convertToBinaryVendor)
newBD['Generic Items'] = newBD['Generic Items'].apply(convertToBinaryItems)
newBD['Item Weight'] = newBD['Item Weight'].apply(convertToIntWeight)
```

In [11]:

```
prodSales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1671 entries, 0 to 1670
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Products         1671 non-null   object
 1   Months           1671 non-null   datetime64[ns]
 2   Total Sales ($)  1671 non-null   float64
dtypes: datetime64[ns](1), float64(1), object(1)
memory usage: 39.3+ KB
```

In [12]:

```
newBD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144977 entries, 0 to 144976
Data columns (total 16 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   Product Description  144977 non-null  object 
 1   Brand                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   Flavor               7807 non-null    object 
 8   Item Weight          64454 non-null   float64
 9   Total THC            144977 non-null  float64
 10  Total CBD            144977 non-null  float64
 11  Pax Filter           44301 non-null   float64
 12  Strain               115639 non-null  object 
 13  Mood Effect          144977 non-null  float64
 14  Generic Vendor       144977 non-null  float64
 15  Generic Items        144977 non-null  float64
dtypes: float64(7), object(9)
memory usage: 17.7+ MB
```

In [13]:
```
#merge newBD and prodSales by product
newBD= newBD.rename(columns={'Product Description': 'Products'})
```

In [14]:
```
merge= pd.merge(prodSales, newBD, on='Products', how='left')
```

In [15]:
```
#product is not a key for the brand description values and some products are present more than once because they
# have different pax filter/category L3/category L5 values. This doesn't make any sense in our merged dataset
# because we don't know which product the total sales refers too. We will drop the repeated products from our
# dataset
new_merge = merge.drop_duplicates(subset=['Products','Months'], ignore_index=True)
```

In [16]:
```
new_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1671 entries, 0 to 1670
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype         
---  ------           --------------  -----         
 0   Products         1671 non-null   object        
 1   Months           1671 non-null   datetime64[ns]
 2   Total Sales ($)  1671 non-null   float64       
 3   Brand            1671 non-null   object        
 4   Category L1      1671 non-null   object        
 5   Category L2      1671 non-null   object        
 6   Category L3      1671 non-null   object        
 7   Category L4      955 non-null    object        
 8   Category L5      360 non-null    object        
 9   Flavor           451 non-null    object        
 10  Item Weight      360 non-null    float64       
 11  Total THC        1671 non-null   float64       
 12  Total CBD        1671 non-null   float64       
 13  Pax Filter       216 non-null    float64       
 14  Strain           572 non-null    object        
 15  Mood Effect      1671 non-null   float64       
 16  Generic Vendor   1671 non-null   float64       
 17  Generic Items    1671 non-null   float64       
dtypes: datetime64[ns](1), float64(8), object(9)
memory usage: 235.1+ KB
```

In [17]:
```
#convert date column into separate month/year input
new_merge.loc[:,'year']=new_merge.loc[:,'Months'].dt.year
new_merge.loc[:,'month']=new_merge.loc[:,'Months'].dt.month
```

```
/Users/noelawheeler/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexing.py:1667: 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
```

In [18]:
```
time_merge=new_merge.sort_values(by=['Months'])
time_merge.head(100)
```

Out[18]:

	Products	Months	Total Sales ($)	Brand	Category L1	Category L2	Category L3	Category L4	Category L5	Flavor	Item Weight	Total THC	Total CBD	Pax Filter	Strain	Mood Effect	Generic Vendor
1385	Up North Humboldt - Durban Poison - Flower (Gram)	2018-10-01	232056.738077	Up North Humboldt	Inhaleables	Flower	Sativa	NaN	NaN	NaN	NaN	0.0	0.0	NaN	Durban Poison	0.0	0.0
398	Plus Products - Gummies - Indica - Unwind - Bl...	2018-10-01	717699.472242	Plus Products	Ingestibles	Edibles	Candy	Gummie Candy	NaN	Blackberry Lemon	NaN	90.0	10.0	NaN	NaN	1.0	0.0
1503	Caliva - Alien OG - Flower	2018-10-01	172868.662357	Caliva	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	NaN	0.0	0.0	NaN	Alien OG	0.0	0.0

In [19]:

```
df_time=new_merge.copy()
products=new_merge["Products"].unique()
for i in products:
    y=time_merge.index[df_time['Products']==i].tolist()
    df_time.loc[y,'Rolling Average'] = (df_time.loc[y,'Total Sales ($)'].shift(1) + df_time.loc[y,'Total Sales ($)'].shift(2) + df_time.loc[y,'Total Sale
    df_time.at[y[2],'Rolling Average'] = (df_time.iloc[y[0]]['Total Sales ($)'] + df_time.iloc[y[1]]['Total Sales ($)'])/2
    df_time.at[y[1],'Rolling Average'] = df_time.iloc[y[0]]['Total Sales ($)']
    df_time.at[y[0],'Rolling Average'] = df_time.iloc[y[0]]['Total Sales ($)']
```

In [20]:

```
df_time.head()
```

Out[20]:

	Products	Months	Total Sales ($)	Brand	Category L1	Category L2	Category L3	Category L4	Category L5	Flavor	...	Total THC	Total CBD	Pax Filter	Strain	Mood Effect	Generic Vendor	Generic Items	year	month	R Av
0	Flower - Strain Blends - Flower (Gram)	2020-07-01	2.273849e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	...	0.0	0.0	NaN	NaN	0.0	1.0	1.0	2020	7	4.406626
1	Flower - Strain Blends - Flower (Gram)	2021-03-01	2.264851e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	...	0.0	0.0	NaN	NaN	0.0	1.0	1.0	2021	3	2.918791
2	Flower - Strain Blends - Flower (Gram)	2021-05-01	2.233876e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	...	0.0	0.0	NaN	NaN	0.0	1.0	1.0	2021	5	2.468878
3	Flower - Strain Blends - Flower (Gram)	2020-09-01	2.146195e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	...	0.0	0.0	NaN	NaN	0.0	1.0	1.0	2020	9	7.910952
4	Flower - Strain Blends - Flower (Gram)	2021-06-01	2.134757e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	...	0.0	0.0	NaN	NaN	0.0	1.0	1.0	2021	6	6.606599

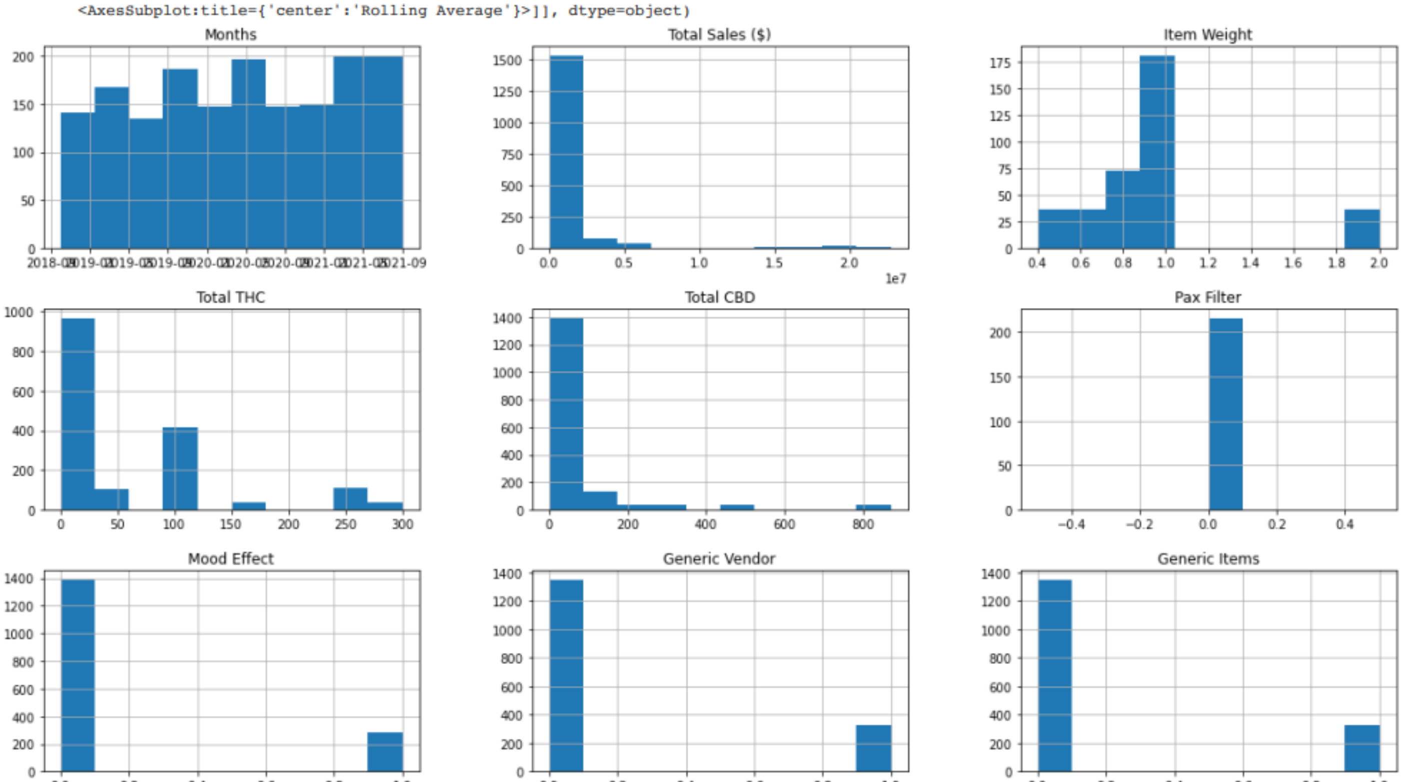
5 rows × 21 columns

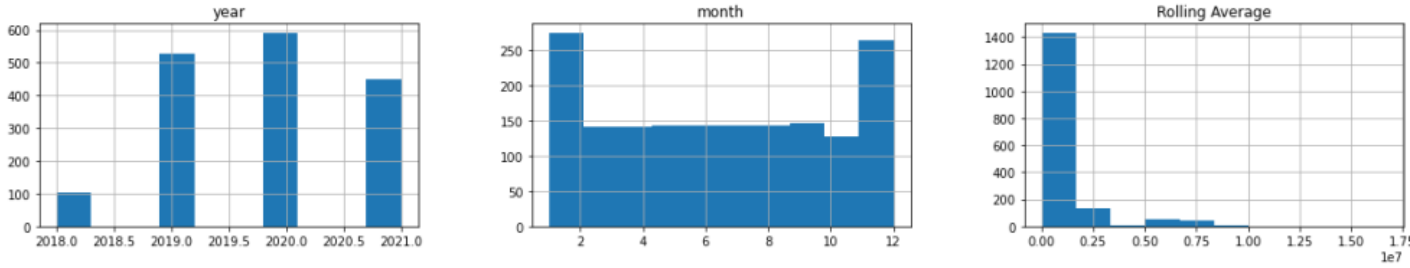
In [21]:

```
df_time.hist(figsize=(20,15))
```

Out[21]:

```
array([[<AxesSubplot:title={'center':'Months'}>,
        <AxesSubplot:title={'center':'Total Sales ($)'}>,
        <AxesSubplot:title={'center':'Item Weight'}>],
       [<AxesSubplot:title={'center':'Total THC'}>,
        <AxesSubplot:title={'center':'Total CBD'}>,
        <AxesSubplot:title={'center':'Pax Filter'}>],
       [<AxesSubplot:title={'center':'Mood Effect'}>,
        <AxesSubplot:title={'center':'Generic Vendor'}>,
        <AxesSubplot:title={'center':'Generic Items'}>],
       [<AxesSubplot:title={'center':'year'}>,
        <AxesSubplot:title={'center':'month'}>,
        <AxesSubplot:title={'center':'Rolling Average'}>]], dtype=object)
```



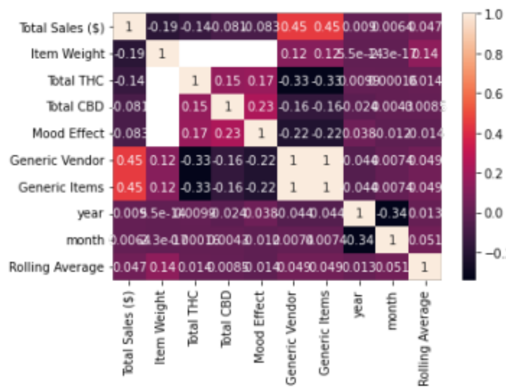


In [22]:
```
#we can see that pax filter is not very useful because it only has one value in our data set
#we can also drop generic vendor because it appear to have the same distributions as generic items
df_dropped=df_time.drop(['Pax Filter'], axis=1)
```

In [50]:
```
corr_matrix=df_dropped.corr()
corr_matrix['Total Sales ($)']
```

Out[50]:
```
Total Sales ($)     1.000000
Item Weight        -0.189561
Total THC          -0.143469
Total CBD          -0.080867
Mood Effect        -0.082969
Generic Vendor      0.453812
Generic Items       0.453812
year                0.008976
month               0.006425
Rolling Average     0.046546
Name: Total Sales ($), dtype: float64
```

In [25]:
```
heatmap=sns.heatmap(corr_matrix, annot= True)
```



In [27]:
```
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.preprocessing import FunctionTransformer
from sklearn.base import BaseEstimator, TransformerMixin
```

In [51]:
```
#augmentation: count how many products are offered by a brand
#drop Month column, we will look at year/month individually
df_dropped['Prod Counts'] = df_dropped.groupby(['Brand'])['Products'].transform('count')
df_dropped=df_dropped.drop(columns=['Months'])
df_dropped.head()
```

Out[51]:

	Products	Total Sales ($)	Brand	Category L1	Category L2	Category L3	Category L4	Category L5	Flavor	Item Weight	Total THC	Total CBD	Strain	Mood Effect	Generic Vendor	Generic Items	year	month	Rolling Average	Pr Cou
0	Flower - Strain Blends - Flower (Gram)	2.273849e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	NaN	0.0	0.0	NaN	0.0	1.0	1.0	2020	7	4.406626e+05	1
1	Flower - Strain Blends - Flower (Gram)	2.264851e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	NaN	0.0	0.0	NaN	0.0	1.0	1.0	2021	3	2.918791e+05	1
2	Flower - Strain Blends - Flower (Gram)	2.233876e+07	Flower	Inhaleables	Flower	Hybrid	NaN	NaN	NaN	NaN	0.0	0.0	NaN	0.0	1.0	1.0	2021	5	2.468878e+05	1

In [29]:

```
num_pipeline=Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('std_scaler', StandardScaler())
    ])

cat_pipeline=Pipeline([
        ('imputer', SimpleImputer(strategy="most_frequent")),
        ('OneHotEncode', OneHotEncoder(categories='auto', handle_unknown='ignore'))
    ])

numerical=["Total THC","Total CBD", "Item Weight", "year", "month"]
categorical=["Products", "Brand","Category L1", "Category L2","Category L3","Category L4","Category L5", "Flavor",
             "Strain", "Mood Effect", "Generic Items"]


full_pipeline=ColumnTransformer([
    ('cat', cat_pipeline, categorical),
    ('num', num_pipeline, numerical),
    ])
```

In [30]:

```
target_column=df_dropped['Total Sales ($)']
df=df_dropped.drop(columns=['Total Sales ($)'])
```

In [31]:

```
x=full_pipeline.fit_transform(df)
x_df = pd.DataFrame(x.toarray())
```

In [32]:

```
x_df.shape
```

Out[32]:

```
(1671, 149)
```

In [33]:

```
y=target_column
```

In [43]:

```
x_train, x_test, y_train, y_test = train_test_split(x_df, y, test_size=0.15)
```

In [35]:

```
import statsmodels.api as sm
# build the OLS model (ordinary least squares) from the training data
sales_stats = sm.OLS(target_column, x_df)

# do the fit and save regression info (parameters, etc) in results_stats
results_stats = sales_stats.fit()
print(results_stats.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Total Sales ($)   R-squared:                       0.981
Model:                            OLS   Adj. R-squared:                  0.981
Method:                 Least Squares   F-statistic:                     1656.
Date:                Sat, 04 Dec 2021   Prob (F-statistic):               0.00
Time:                        16:58:26   Log-Likelihood:                -23862.
No. Observations:                1671   AIC:                         4.783e+04
Df Residuals:                    1619   BIC:                         4.811e+04
Df Model:                          51
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
0          -7.024e+05   2.03e+04    -34.596      0.000   -7.42e+05   -6.63e+05
1           5.279e+04   3.79e+04      1.393      0.164   -2.15e+04    1.27e+05
2          -5.667e+04    3.6e+04     -1.573      0.116   -1.27e+05     1.4e+04
3          -7.049e+05   3.29e+04    -21.437      0.000   -7.69e+05    -6.4e+05
4          -2.778e+05   1.66e+04    -16.726      0.000    -3.1e+05   -2.45e+05
5          -5.175e+05   1.51e+04    -34.286      0.000   -5.47e+05   -4.88e+05
6          -4.062e+05   2.83e+04    -14.349      0.000   -4.62e+05   -3.51e+05
7          -4.105e+05   2.86e+04    -14.334      0.000   -4.67e+05   -3.54e+05
8          -4.553e+05   2.77e+04    -16.431      0.000    -5.1e+05   -4.01e+05
9          -1.579e+06   2.76e+04    -57.160      0.000   -1.63e+06   -1.52e+06
10         -2.246e+06   2.76e+04    -81.324      0.000    -2.3e+06   -2.19e+06
11          7.718e+06   3.58e+04    215.469      0.000    7.65e+06    7.79e+06
12          1.537e+05   1.89e+04      8.147      0.000    1.17e+05    1.91e+05
13          3.247e+05   2.51e+04     12.963      0.000    2.76e+05    3.74e+05
14         -3.124e+05   8138.983    -38.384      0.000   -3.28e+05   -2.96e+05
15          1.908e+05   3.84e+04      4.962      0.000    1.15e+05    2.66e+05
16         -6.409e+04   3.09e+04     -2.073      0.038   -1.25e+05   -3453.660
17         -1.183e+05   3.09e+04     -3.829      0.000   -1.79e+05   -5.77e+04
18          -5.55e+04   3.34e+04     -1.662      0.097   -1.21e+05       1e+04
19         -1.619e+04   3.09e+04     -0.524      0.601   -7.68e+04    4.45e+04
20         -1.099e+05   3.84e+04     -2.859      0.004   -1.85e+05   -3.45e+04
21          7.614e+04   2.35e+04      3.239      0.001       3e+04    1.22e+05
22          1.824e+04   2.35e+04      0.776      0.438   -2.79e+04    6.44e+04
23          1.314e+04   4.65e+04      0.283      0.777    -7.8e+04    1.04e+05
24         -5059.7078   4.65e+04     -0.109      0.913   -9.62e+04    8.61e+04
25         -5.717e+05   3.15e+04    -18.124      0.000   -6.34e+05    -5.1e+05
26          5.032e+05   2.22e+04     22.685      0.000     4.6e+05    5.47e+05
27         -5.012e+05    3.2e+04    -15.648      0.000   -5.64e+05   -4.38e+05
28         -1.577e+04   3.72e+04     -0.424      0.672   -8.87e+04    5.72e+04
29          -2.22e+04   3.87e+04     -0.573      0.566   -9.81e+04    5.37e+04
30           -2.4e+04    3.8e+04     -0.632      0.527   -9.85e+04    5.05e+04
31           3.03e+04   1.82e+04      1.662      0.097   -5458.396    6.61e+04
32          2.677e+04   2.89e+04      0.926      0.355      -3e+04    8.35e+04
33         -2.482e+04    2.6e+04     -0.955      0.339   -7.58e+04    2.61e+04
34         -1.085e+04   1.47e+04     -0.739      0.460   -3.96e+04    1.79e+04
```

35	-6.731e+05	2.4e+04	-28.020	0.000	-7.2e+05	-6.26e+05
36	-7.016e+05	2.03e+04	-34.558	0.000	-7.41e+05	-6.62e+05
37	-5.105e+05	4.1e+04	-12.467	0.000	-5.91e+05	-4.3e+05
38	5.792e+05	2.77e+04	20.944	0.000	5.25e+05	6.33e+05
39	5.922e+05	2.77e+04	21.418	0.000	5.38e+05	6.46e+05
40	4.852e+05	2.55e+04	19.038	0.000	4.35e+05	5.35e+05
41	2.04e+05	2.4e+04	8.509	0.000	1.57e+05	2.51e+05
42	2.101e+05	2.4e+04	8.763	0.000	1.63e+05	2.57e+05
43	-7.114e+05	2.17e+04	-32.802	0.000	-7.54e+05	-6.69e+05
44	3.047e+05	1.5e+04	20.303	0.000	2.75e+05	3.34e+05
45	-6.021e+04	3.34e+04	-1.801	0.072	-1.26e+05	5374.752
46	1.052e+05	5.06e+04	2.081	0.038	6024.133	2.04e+05
47	-8.374e+04	3.27e+04	-2.564	0.010	-1.48e+05	-1.97e+04
48	8.826e+04	3.27e+04	2.702	0.007	2.42e+04	1.52e+05
49	6.248e+04	3.27e+04	1.913	0.056	-1581.415	1.27e+05
50	-7.024e+05	2.03e+04	-34.596	0.000	-7.42e+05	-6.63e+05
51	-3878.8384	1.75e+04	-0.222	0.825	-3.82e+04	3.04e+04
52	-1.5e+06	2.25e+04	-66.550	0.000	-1.54e+06	-1.46e+06
53	-1.272e+06	2.17e+04	-58.506	0.000	-1.31e+06	-1.23e+06
54	3.893e+06	1.91e+04	204.072	0.000	3.86e+06	3.93e+06
55	1.537e+05	1.89e+04	8.147	0.000	1.17e+05	1.91e+05
56	3.247e+05	2.51e+04	12.963	0.000	2.76e+05	3.74e+05
57	-3.124e+05	8138.983	-38.384	0.000	-3.28e+05	-2.96e+05
58	-7.882e+04	1.73e+04	-4.558	0.000	-1.13e+05	-4.49e+04
59	8080.8052	1.2e+04	0.673	0.501	-1.55e+04	3.16e+04
60	-5.697e+05	2.38e+04	-23.938	0.000	-6.16e+05	-5.23e+05
61	-3.167e+04	1.75e+04	-1.805	0.071	-6.61e+04	2739.166
62	1951.8776	1.99e+04	0.098	0.922	-3.72e+04	4.11e+04
63	-1.085e+04	1.47e+04	-0.739	0.460	-3.96e+04	1.79e+04
64	-6.731e+05	2.4e+04	-28.020	0.000	-7.2e+05	-6.26e+05
65	-7.016e+05	2.03e+04	-34.558	0.000	-7.41e+05	-6.62e+05
66	1.146e+06	2.27e+04	50.411	0.000	1.1e+06	1.19e+06
67	4.14e+05	1.32e+04	31.457	0.000	3.88e+05	4.4e+05
68	-7.114e+05	2.17e+04	-32.802	0.000	-7.54e+05	-6.69e+05
69	3.047e+05	1.5e+04	20.303	0.000	2.75e+05	3.34e+05
70	1.12e+05	1.61e+04	6.961	0.000	8.05e+04	1.44e+05
71	-1.879e+05	8051.292	-23.332	0.000	-2.04e+05	-1.72e+05
72	6.965e+04	1.02e+04	6.859	0.000	4.97e+04	8.96e+04
73	-2.922e+04	1.32e+04	-2.219	0.027	-5.5e+04	-3389.752
74	-6.197e+04	9830.213	-6.304	0.000	-8.12e+04	-4.27e+04
75	-5.146e+05	1.07e+04	-47.991	0.000	-5.36e+05	-4.94e+05
76	-1.879e+05	8051.292	-23.332	0.000	-2.04e+05	-1.72e+05
77	4.323e+04	1.07e+04	4.042	0.000	2.23e+04	6.42e+04
78	3.946e+05	1.19e+04	33.112	0.000	3.71e+05	4.18e+05
79	9.078e+04	1.26e+04	7.207	0.000	6.61e+04	1.15e+05
80	2.642e+04	1.04e+04	2.534	0.011	5968.202	4.69e+04
81	-6.197e+04	9830.213	-6.304	0.000	-8.12e+04	-4.27e+04
82	-6.197e+04	9830.213	-6.304	0.000	-8.12e+04	-4.27e+04
83	-5.923e+04	1.03e+04	-5.750	0.000	-7.94e+04	-3.9e+04
84	9.438e+04	9243.165	10.211	0.000	7.63e+04	1.13e+05
85	-2.778e+05	1.66e+04	-16.726	0.000	-3.1e+05	-2.45e+05
86	3.258e+06	2.17e+04	149.839	0.000	3.22e+06	3.3e+06
87	6.569e+05	1.96e+04	33.593	0.000	6.19e+05	6.95e+05
88	-1.579e+06	2.76e+04	-57.160	0.000	-1.63e+06	-1.52e+06
89	1.016e+05	1.25e+04	8.159	0.000	7.72e+04	1.26e+05
90	8080.8052	1.2e+04	0.673	0.501	-1.55e+04	3.16e+04
91	-1.085e+04	1.47e+04	-0.739	0.460	-3.96e+04	1.79e+04
92	3.047e+05	1.5e+04	20.303	0.000	2.75e+05	3.34e+05
93	-2.246e+06	2.76e+04	-81.324	0.000	-2.3e+06	-2.19e+06
94	2.642e+04	1.04e+04	2.534	0.011	5968.202	4.69e+04
95	-2.369e+05	1.32e+04	-17.999	0.000	-2.63e+05	-2.11e+05
96	-1.879e+05	8051.292	-23.332	0.000	-2.04e+05	-1.72e+05
97	9.438e+04	9243.165	10.211	0.000	7.63e+04	1.13e+05
98	2.642e+04	1.04e+04	2.534	0.011	5968.202	4.69e+04
99	3.642e+05	1.03e+04	35.516	0.000	3.44e+05	3.84e+05
100	-2.778e+05	1.66e+04	-16.726	0.000	-3.1e+05	-2.45e+05
101	-1.879e+05	8051.292	-23.332	0.000	-2.04e+05	-1.72e+05
102	8080.8052	1.2e+04	0.673	0.501	-1.55e+04	3.16e+04
103	2.807e+05	1.64e+04	17.141	0.000	2.49e+05	3.13e+05
104	-5.175e+05	1.51e+04	-34.286	0.000	-5.47e+05	-4.88e+05
105	9.438e+04	9243.165	10.211	0.000	7.63e+04	1.13e+05
106	4.016e+05	1.31e+04	30.662	0.000	3.76e+05	4.27e+05
107	-5.175e+05	1.51e+04	-34.286	0.000	-5.47e+05	-4.88e+05
108	-1.879e+05	8051.292	-23.332	0.000	-2.04e+05	-1.72e+05
109	-2.178e+05	1.94e+04	-11.216	0.000	-2.56e+05	-1.8e+05
110	7.614e+04	2.35e+04	3.239	0.001	3e+04	1.22e+05
111	1.052e+05	5.06e+04	2.081	0.038	6024.133	2.04e+05
112	1.824e+04	2.35e+04	0.776	0.438	-2.79e+04	6.44e+04
113	-8.374e+04	3.27e+04	-2.564	0.010	-1.48e+05	-1.97e+04
114	8.826e+04	3.27e+04	2.702	0.007	2.42e+04	1.52e+05
115	1.908e+05	3.84e+04	4.962	0.000	1.15e+05	2.66e+05
116	-6.409e+04	3.09e+04	-2.073	0.038	-1.25e+05	-3453.660
117	-6.021e+04	3.34e+04	-1.801	0.072	-1.26e+05	5374.752
118	6.248e+04	3.27e+04	1.913	0.056	-1581.415	1.27e+05
119	-1.183e+05	3.09e+04	-3.829	0.000	-1.79e+05	-5.77e+04
120	-2.482e+04	2.6e+04	-0.955	0.339	-7.58e+04	2.61e+04
121	-5.55e+04	3.34e+04	-1.662	0.097	-1.21e+05	1e+04
122	-1.619e+04	3.09e+04	-0.524	0.601	-7.68e+04	4.45e+04
123	-1.099e+05	3.84e+04	-2.859	0.004	-1.85e+05	-3.45e+04
124	-5.717e+05	3.15e+04	-18.124	0.000	-6.34e+05	-5.1e+05
125	-7.024e+05	2.03e+04	-34.596	0.000	-7.42e+05	-6.63e+05
126	-4.062e+05	2.83e+04	-14.349	0.000	-4.62e+05	-3.51e+05
127	1.704e+06	1.9e+04	89.536	0.000	1.67e+06	1.74e+06
128	-7.114e+05	2.17e+04	-32.802	0.000	-7.54e+05	-6.69e+05
129	3.047e+05	1.5e+04	20.303	0.000	2.75e+05	3.34e+05
130	-4.105e+05	2.86e+04	-14.334	0.000	-4.67e+05	-3.54e+05
131	-4.553e+05	2.77e+04	-16.431	0.000	-5.1e+05	-4.01e+05
132	2.04e+05	2.4e+04	8.509	0.000	1.57e+05	2.51e+05
133	5.792e+05	2.77e+04	20.944	0.000	5.25e+05	6.33e+05
134	5.032e+05	2.22e+04	22.685	0.000	4.6e+05	5.47e+05
135	-7.016e+05	2.03e+04	-34.558	0.000	-7.41e+05	-6.62e+05
136	1.537e+05	1.89e+04	8.147	0.000	1.17e+05	1.91e+05
137	5.922e+05	2.77e+04	21.418	0.000	5.38e+05	6.46e+05
138	2.101e+05	2.4e+04	8.763	0.000	1.63e+05	2.57e+05
139	-5.012e+05	3.2e+04	-15.648	0.000	-5.64e+05	-4.38e+05

```
140         -6.843e+04   1.24e+04     -5.516      0.000   -9.28e+04   -4.41e+04
141         -1.409e+05   1.27e+04    -11.086      0.000   -1.66e+05   -1.16e+05
142         -1.606e+06   1.13e+04   -142.186      0.000   -1.63e+06   -1.58e+06
143          1.397e+06   1.38e+04    101.207      0.000    1.37e+06    1.42e+06
144          4455.2043    1.6e+04      0.278      0.781   -2.69e+04    3.58e+04
145          2.997e+04   1.44e+04      2.088      0.037    1814.764    5.81e+04
146         -4.563e+05   9323.573    -48.939      0.000   -4.75e+05   -4.38e+05
147          6.981e+04   1.04e+04      6.734      0.000    4.95e+04    9.01e+04
148          3.392e+04   1.02e+04      3.328      0.001    1.39e+04    5.39e+04
==============================================================================
Omnibus:                      585.560   Durbin-Watson:                   0.584
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            75278.771
Skew:                           0.559   Prob(JB):                         0.00
Kurtosis:                      35.863   Cond. No.                     8.97e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 9.63e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [58]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [36]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)
predictions = lr.predict(x_test)
```

In [37]:
```
from sklearn.metrics import mean_absolute_error

preds = lr.predict(x_test)
rmse = mean_absolute_error(y_test, preds)
rmse
```

Out[37]:
```
232942.94321512
```

In [283…
```
x_df.shape
```

Out[283…
```
(1671, 149)
```

In [38]:
```
#PCA Analysis
from sklearn import decomposition
pca = decomposition.PCA(n_components=4)


pca_df = pca.fit_transform(x_df)
```

In [39]:
```
pca_df.shape
```

Out[39]:
```
(1671, 4)
```

In [40]:
```
#new testing/training data using pca
x_train, x_test, y_train, y_test = train_test_split(pca_df, y, test_size=0.15)
```

In [44]:
```
#use random forest ensemble method
from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n_estimators=20, random_state=0)
rf.fit(x_train, y_train.ravel())
pred=rf.predict(x_test)
```


In [38]:
```
#PCA Analysis
from sklearn import decomposition
pca = decomposition.PCA(n_components=4)


pca_df = pca.fit_transform(x_df)
```

In [39]:
```
pca_df.shape
```

Out[39]:
```
(1671, 4)
```

In [40]:
```
#new testing/training data using pca
x_train, x_test, y_train, y_test = train_test_split(pca_df, y, test_size=0.15)
```

In [44]:
```
#use random forest ensemble method
from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n_estimators=20, random_state=0)
rf.fit(x_train, y_train.ravel())
pred=rf.predict(x_test)
```

In [45]:

```
rmse = mean_absolute_error(y_test, preds)
rmse
```

Out[45]: 1317184.6205840847

In [46]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
```

In [62]:

```
#Employ Kfold cross validation to training results
from sklearn.model_selection import KFold
from sklearn import model_selection

#KFold, with 10 splits where we first shuffle our data before splitting it
kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle=True)
rf_model_kfold = RandomForestRegressor(n_estimators=20, random_state=0)

lr_model_kfold = LinearRegression()

rf_results_kfold=model_selection.cross_val_score(rf_model_kfold,x_df,y.ravel(), scoring='r2', cv=kfold)
lr_results_kfold=model_selection.cross_val_score(lr_model_kfold,x_df,y.ravel(), scoring='r2', cv=kfold)

print("Random Forest Accuracy: %.2f%%" % (rf_results_kfold.mean()*100.0))

print("Linear Regression Accuracy: %.2f%%" % (lr_results_kfold.mean()*100.0))
```

```
Random Forest Accuracy: 99.08%
Linear Regression Accuracy: 79.85%
```

In [35]:

```
#gridsearch method to optimize parameters
from sklearn.model_selection import GridSearchCV
parameters = {'criterion':('squared_error', 'absolute_error', 'poisson'), 'max_depth':[5, 10, 15, 20]}
randforest=RandomForestRegressor()
clf = GridSearchCV(randforest, parameters)
clf.fit(x_df, target_column)
```

Out[35]:

```
GridSearchCV(estimator=RandomForestRegressor(),
             param_grid={'criterion': ('squared_error', 'absolute_error',
                                       'poisson'),
                         'max_depth': [5, 10, 15, 20]})
```

In [314…

```
print(clf.best_params_)
```

```
{'criterion': 'poisson', 'max_depth': 15}
```

In [48]:

```
#experiment with other models
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn import svm

gpr = GaussianProcessRegressor(random_state=0)
gpr.fit(x_train, y_train.ravel())
pred_gpr=gpr.predict(x_test)

svm_model=svm.SVR()
svm_model.fit(x_train, y_train.ravel())
pred_svm=svm_model.predict(x_test)
```

In [49]:

```
rmse_gpr = mean_absolute_error(y_test, pred_gpr)
rmse_svm = mean_absolute_error(y_test, pred_svm)
print("RMSE Gaussian Process Regressor:  " + str(rmse_gpr))
print("RMSE SVM Regressor:  " + str(rmse_svm))
```

```
RMSE Gaussian Process Regressor:  1720693.592204542
RMSE SVM Regressor:  742022.4251756461
```

In [ ]: