Project 3

In [26]:
```
import sys
import random
import os
from pathlib import Path

%matplotlib inline
import matplotlib  # plotting library
import matplotlib.pyplot as plt  # plotting package
import numpy as np  #numerical package in python
import pandas as pd
import sklearn

random.seed(42)

ROOT_DIR = "."
DATA_PATH = os.path.join(ROOT_DIR, "data")


#pls no mark off for poor naming, ik it's not housing
#but i'm rlly low on time in my classes and my keyboard is half-broken (using on-

def load_housing_data(housing_path):
    csv_path = os.path.join(DATA_PATH, housing_path)
    return pd.read_csv(csv_path)

arp_data = load_housing_data("BrandAverageRetailPrice.csv") # we load the pandas
details_data = load_housing_data("BrandDetails.csv") # we load the pandas datafr
sales_data = load_housing_data("BrandTotalSales.csv") # we load the pandas dataf
units_data = load_housing_data("BrandTotalUnits.csv") # we load the pandas dataf
```

In [27]:
```
arp_data.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 [28]:
```
#1 combining data sets
```

In [29]:
```python
# Rename columns for ease in combining them.
units_data = units_data.rename(columns={'vs. Prior Period':'vs. Prior Period Unit
arp_data = arp_data.rename(columns={'vs. Prior Period':'vs. Prior Period Units AF
sales_data =sales_data.rename(columns={'Brand':'Brands'})

alo = pd.merge(pd.merge(units_data, arp_data, on = ["Brands", "Months"], how = 'c


alo["Months"].unique()
```

Out[29]:
```
array(['08/2020', '09/2020', '01/2021', '02/2021', '03/2021', '11/2019',
       '12/2019', '01/2020', '02/2020', '03/2020', '04/2020', '05/2020',
       '06/2020', '07/2020', '10/2020', '11/2020', '12/2020', '04/2021',
       '05/2021', '06/2021', '07/2021', '08/2021', '09/2021', '08/2018',
       '09/2018', '10/2018', '11/2018', '12/2018', '01/2019', '02/2019',
       '03/2019', '04/2019', '05/2019', '08/2019', '09/2019', '10/2019',
       '06/2019', '07/2019'], dtype=object)
```

At this point we've combined these three pretty cleanly afaik, but the problem now lies in that there doesn't seem to be any sort of notion of time especially in months for the last one which is details_file. Also that they're talking about individual products now which we didn't really deal with in the other three. so combining this with the amalgmation of the other three will be somewhat messy.

Here i'm thinking of making a separate column for each month (Sort of like onehotencoding) although this will create extraneous stuff.

Now i want to merge the details info with our alo combination. it's somewhat tricky to do this because they're mismatched in multiple ways, so for this one i'll go with the easier way and take what i want out of the details data into alo. this is because i spent too long getting stuck on the other way with python sytnax which i'm not familiar with. i will likely try the other way

In [30]:
```python
alo['Months'] = pd.to_datetime(alo['Months'])
#Total units is too large currently to convert to a float
#need to trim it first then convert to float
alo['Total Units'] = alo['Total Units'].str[:8].str.replace(",","")
alo['Total Units'] = pd.to_numeric(alo['Total Units'])

alo['Months'] = pd.to_datetime(alo['Months'])
alo['Total Sales ($)'] = alo['Total Sales ($)'].str[:8].str.replace(",","")
alo['Total Sales ($)'] = pd.to_numeric(alo['Total Sales ($)'])

alo.head(30)
```

Out[30]:

	Brands	Months	Total Units	vs. Prior Period Units	ARP	vs. Prior Period Units ARP	Total Sales ($)
0	#BlackSeries	2020-08-01	1616.3300	NaN	15.684913	NaN	25352.10
1	#BlackSeries	2020-09-01	NaN	-1.000000	NaN	-1.000000	NaN
2	#BlackSeries	2021-01-01	715.5328	NaN	13.611428	NaN	9739.42
3	#BlackSeries	2021-02-01	766.6691	0.071466	11.873182	-0.127705	9102.80
4	#BlackSeries	2021-03-01	NaN	-1.000000	NaN	-1.000000	NaN
5	101 Cannabis Co.	2019-11-01	131.0677	NaN	34.066667	NaN	4465.04
6	101 Cannabis Co.	2019-12-01	NaN	-1.000000	NaN	-1.000000	NaN
7	101 Cannabis Co.	2020-01-01	345.4134	NaN	34.134929	NaN	11790.60
8	101 Cannabis Co.	2020-02-01	696.6584	1.016883	29.091388	-0.147753	20266.70
9	101 Cannabis Co.	2020-03-01	943.3933	0.354169	32.293498	0.110071	30465.40
10	101 Cannabis Co.	2020-04-01	712.4981	-0.244750	32.934344	0.019844	23465.60
11	101 Cannabis Co.	2020-05-01	619.8410	-0.130045	34.441725	0.045769	21348.30
12	101 Cannabis Co.	2020-06-01	426.1504	-0.312484	33.114497	-0.038535	14111.70
13	101 Cannabis Co.	2020-07-01	589.7193	0.383829	32.131407	-0.029688	18948.50
14	101 Cannabis Co.	2020-08-01	1018.5700	0.727218	32.146382	0.000466	32743.40
15	101 Cannabis Co.	2020-09-01	1408.8500	0.383160	31.827140	-0.009931	44839.60
16	101 Cannabis Co.	2020-10-01	1148.9600	-0.184468	30.375108	-0.045622	34899.80

	Brands	Months	Total Units	vs. Prior Period Units	ARP	vs. Prior Period Units ARP	Total Sales ($)
17	101 Cannabis Co.	2020-11-01	447.1605	-0.610814	33.782933	0.112191	15106.30
18	101 Cannabis Co.	2020-12-01	337.9605	-0.244208	35.160945	0.040790	11883.00
19	101 Cannabis Co.	2021-01-01	250.2320	-0.259582	32.206812	-0.084017	8059.17
20	101 Cannabis Co.	2021-02-01	395.8241	0.581828	34.643599	0.075661	13712.70
21	101 Cannabis Co.	2021-03-01	686.8574	0.735259	35.448267	0.023227	24347.90
22	101 Cannabis Co.	2021-04-01	624.6255	-0.090604	33.275813	-0.061285	20784.90
23	101 Cannabis Co.	2021-05-01	345.7618	-0.446449	35.044264	0.053145	12116.90
24	101 Cannabis Co.	2021-06-01	138.1479	-0.600453	38.496970	0.098524	5318.27
25	101 Cannabis Co.	2021-07-01	291.7813	1.112093	36.685150	-0.047064	10704.00
26	101 Cannabis Co.	2021-08-01	421.2570	0.443742	24.740100	-0.325610	10421.90
27	101 Cannabis Co.	2021-09-01	483.5366	0.147842	25.389606	0.026253	12276.80
28	10x Infused	2018-08-01	855.8260	NaN	NaN	NaN	NaN
29	10x Infused	2018-09-01	142.8393	-0.833098	11.980833	NaN	1711.33

In [31]:
```
temp = pd.DataFrame()

for brand in alo.Brands.unique():
    units = alo[alo.Brands == brand]

    units.loc[:,'Previous Month T. Units'] = units.loc[:,'Total Units'].shift(1)
    # inserting another column with difference between yesterday and day before y

    units.loc[:,'Rolling Average T. Units'] = (units.loc[:,'Total Units'].shift(1


    units.loc[:,'Previous Month ARP'] = units.loc[:,"ARP"].shift(1)
    # inserting another column with difference between yesterday and day before y


    units.loc[:,'Previous Month Sales'] = units.loc[:,"Total Sales ($)"].shift(1)
    # inserting another column with difference between yesterday and day before y

    units.loc[:,'Rolling Average Sales'] = (units.loc[:,"Total Sales ($)"].shift(

    units["Average Sales for this Year"] = units.groupby(units.Months.dt.year)['T

    units["Average Sales for this Month of Year"] = units.groupby(units.Months.dt

    units["Average Units Sold for this Year"] = units.groupby(units.Months.dt.yea

    units["Average Units Sold for this Month of Year"] = units.groupby(units.Mont

    units["Cumulative Average Monthly Sales"] = units['Total Sales ($)'].mean()

    units["Cumulative Average Monthly Units"] = units['Total Units'].mean()

    #do # of unique product names and unique product types

    temp = pd.concat([temp, units])

alo = temp
alo.head(20)

#.........................
```

```
nths.dt.month)['Total Units'].transform('mean')
<ipython-input-31-0987dd8cf07f>:29: 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/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  units["Cumulative Average Monthly Sales"] = units['Total Sales ($)'].mean()
<ipython-input-31-0987dd8cf07f>:31: 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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

In [32]:
```
# need make these into columns, put into for-loop above
```

In [33]:
```
alo.columns
```

Out[33]:
```
Index(['Brands', 'Months', 'Total Units', 'vs. Prior Period Units', 'ARP',
       'vs. Prior Period Units ARP', 'Total Sales ($)',
       'Previous Month T. Units', 'Rolling Average T. Units',
       'Previous Month ARP', 'Previous Month Sales', 'Rolling Average Sales',
       'Average Sales for this Year', 'Average Sales for this Month of Year',
       'Average Units Sold for this Year',
       'Average Units Sold for this Month of Year',
       'Cumulative Average Monthly Sales', 'Cumulative Average Monthly Units'],
      dtype='object')
```

In [34]:
```python
##### details_data.info()

#This could arguably be put in the pipeline but I view it as the means in which I
#datasets, which is more #1 than #4.

#THIS CODE TOOK ME MORE THAN 22 HOURS TO WRITE..................


category_columns = ['Category L1', "Category L2", "Category L3", "Category L4", '
category_df = pd.DataFrame()
category_df['Brand'] = details_data['Brand']
#we can make truth table for all possible values depending if a brand ever sells
for cat in category_columns:
    test_df = pd.DataFrame()
    test_df['Brand'] = details_data['Brand']
    test_df2 = details_data[['Brand', cat]]
    test_df2.drop_duplicates(inplace=True)

    for column_name in details_data[cat].unique():

        # creating a new column by checking where this row's brand value exists i
        # dataframe filtered by 'cat' value
        category_df[cat[-2:] + ' ' + str(column_name)] = details_data['Brand'].is

# drop_columns = list(range(0, 25))
# drop_columns.remove(7)
# print(drop_columns)
# details_data.drop(details_data.columns[drop_columns], axis=1, inplace=True)
category_df.drop_duplicates(inplace=True)


for column in category_df.columns:
    category_df[column].replace({False:0.0, True:1.0}, inplace = True)

category_df['Number of Product Types'] = category_df[list(category_df.columns)].s


print(category_df)

alo = alo.set_index('Brands').join(category_df.set_index('Brand'))
alo.reset_index(inplace=True)
alo = alo.rename(columns={"index": "Brands"})

alo.head()
# #print(alo.head())
```

```
<ipython-input-34-db7c28540c40>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  test_df2.drop_duplicates(inplace=True)
```

```
                 Brand  L1 Inhaleables  L1 Topicals  L1 Ingestibles  \
0         #BlackSeries             1.0          0.0             0.0   
4       101 Cannabis Co.           1.0          0.0             0.0   
81               11:11             1.0          0.0             0.0   
254        19Forty LA              0.0          1.0             0.0   
257              1Lyfe             1.0          0.0             0.0   
...                ...             ...          ...             ...   
144811     Zendo Edibles           0.0          0.0             1.0   
144872           Zenleaf           0.0          0.0             0.0   
144898           Zig Zag           0.0          0.0             0.0   
144909     Zips Weed Co.           1.0          0.0             0.0   
144932          Zkittlez           1.0          0.0             0.0   

        L1 All Accessories  L1 Other Cannabis  L2 Flower  L2 Concentrates  \
0                      0.0                0.0        1.0              0.0   
4                      0.0                0.0        0.0              1.0   
81                     0.0                0.0        0.0              1.0   
254                    0.0                0.0        0.0              0.0   
257                    0.0                0.0        1.0              0.0   
...                    ...                ...        ...              ...   
144811                 0.0                1.0        0.0              0.0   
144872                 0.0                1.0        0.0              0.0   
144898                 1.0                0.0        0.0              0.0   
144909                 0.0                0.0        0.0              1.0   
144932                 0.0                0.0        0.0              1.0   

        L2 Pre-Rolled  L2 Topicals  ...  L5 Brownies  L5 Other Chocolates  \
0                 0.0          0.0  ...          0.0                  0.0   
4                 1.0          0.0  ...          0.0                  0.0   
81                1.0          0.0  ...          0.0                  0.0   
254               0.0          1.0  ...          0.0                  0.0   
257               1.0          0.0  ...          0.0                  0.0   
...               ...          ...  ...          ...                  ...   
144811            0.0          0.0  ...          0.0                  0.0   
144872            0.0          0.0  ...          0.0                  0.0   
144898            0.0          0.0  ...          0.0                  0.0   
144909            0.0          0.0  ...          0.0                  0.0   
144932            1.0          0.0  ...          0.0                  0.0   

        L5 Lollipop  L5 Coffee Drink  L5 Flower  L5 Flower and Concentrate  \
0               0.0              0.0        0.0                        0.0   
4               0.0              0.0        0.0                        0.0   
81              0.0              0.0        0.0                        0.0   
254             0.0              0.0        0.0                        0.0   
257             0.0              0.0        0.0                        0.0   
...             ...              ...        ...                        ...   
144811          0.0              0.0        0.0                        0.0   
144872          0.0              0.0        0.0                        0.0   
144898          0.0              0.0        0.0                        0.0   
144909          0.0              0.0        0.0                        0.0   
144932          0.0              0.0        0.0                        0.0   

        L5 Dab Rig  L5 Concentrate  L5 Tools  Number of Product Types  
0              0.0             0.0       0.0                      4.0  
4              0.0             0.0       0.0                      7.0  
81             0.0             0.0       0.0                     11.0  
254            0.0             0.0       0.0                      3.0  
```

```
257           0.0          0.0        0.0                      8.0
...           ...          ...        ...                      ...
144811        0.0          0.0        0.0                     15.0
144872        0.0          0.0        0.0                      4.0
144898        0.0          0.0        0.0                      6.0
144909        0.0          0.0        0.0                      5.0
144932        0.0          0.0        0.0                      6.0

[1123 rows x 185 columns]
```

Out[34]:

	Brands	Months	Total Units	vs. Prior Period Units	ARP	vs. Prior Period Units ARP	Total Sales ($)	Previous Month T. Units	Rolling Average T. Units
0	#BlackSeries	2020-08-01	1616.3300	NaN	15.684913	NaN	25352.10	NaN	NaN
1	#BlackSeries	2020-09-01	NaN	-1.000000	NaN	-1.000000	NaN	1616.3300	NaN
2	#BlackSeries	2021-01-01	715.5328	NaN	13.611428	NaN	9739.42	NaN	NaN
3	#BlackSeries	2021-02-01	766.6691	0.071466	11.873182	-0.127705	9102.80	715.5328	NaN
4	#BlackSeries	2021-03-01	NaN	-1.000000	NaN	-1.000000	NaN	766.6691	NaN

5 rows × 202 columns

In [ ]:

In [ ]:

In [ ]:

In [35]:

```
import sklearn.metrics as metrics
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)
    median_absolute_error=metrics.median_absolute_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    print(y_true)
    print(y_pred)
    print('explained_variance: ', round(explained_variance,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))
```

In [36]:
```
alo.corr()
```

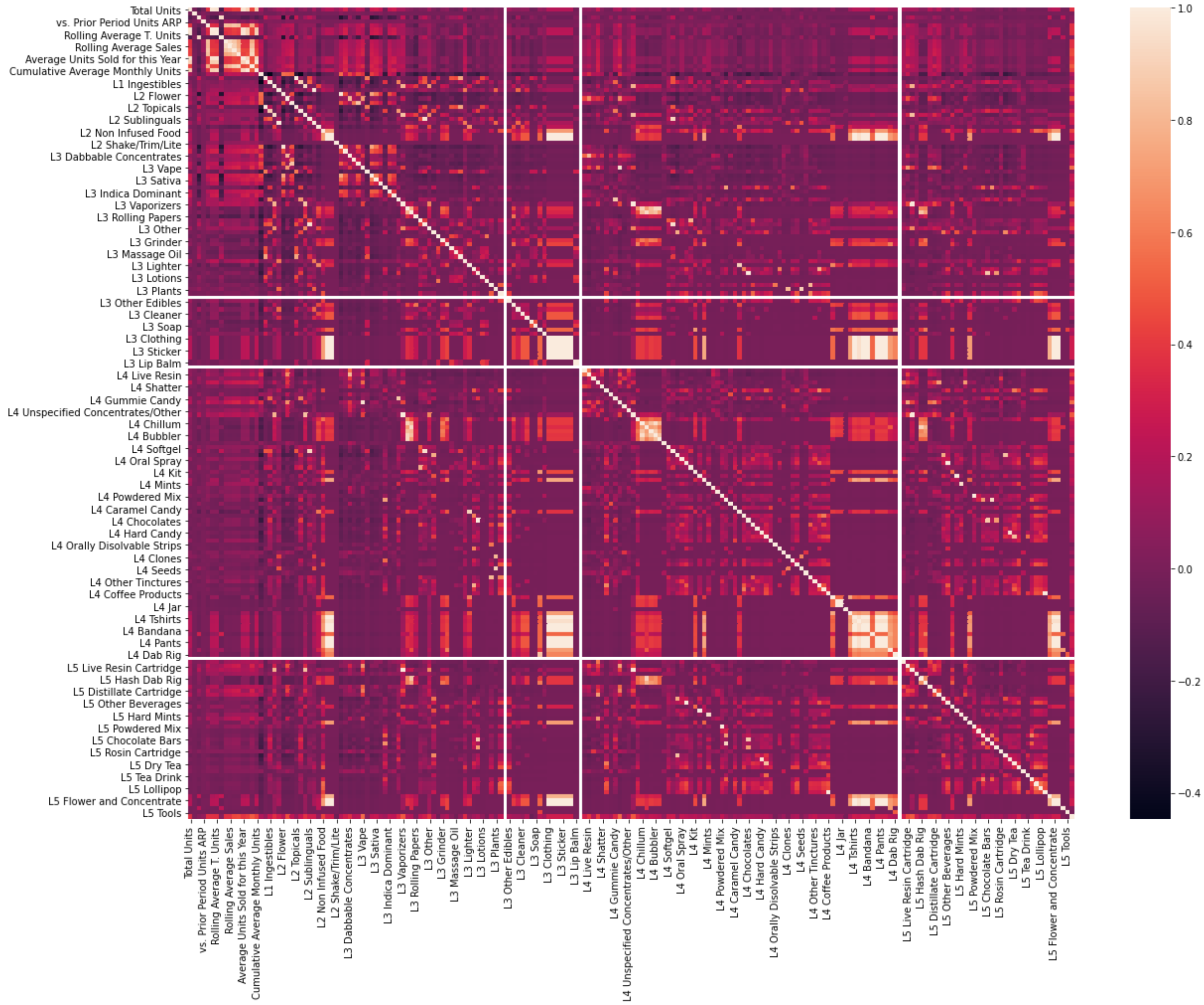
Out[36]:

	Total Units	vs. Prior Period Units	ARP	vs. Prior Period Units ARP	Total Sales ($)	Previous Month T. Units	Rolling Average T. Units	Previous Month ARP
Total Units	1.000000	-0.011140	-0.135363	-0.024310	0.381529	0.973898	0.966086	-0.153154
vs. Prior Period Units	-0.011140	1.000000	-0.040921	0.027300	-0.002142	-0.030178	-0.018677	0.016791
ARP	-0.135363	-0.040921	1.000000	0.095541	-0.057710	-0.146764	-0.172942	0.946568
vs. Prior Period Units ARP	-0.024310	0.027300	0.095541	1.000000	-0.027629	0.074888	0.057231	-0.064552
Total Sales ($)	0.381529	-0.002142	-0.057710	-0.027629	1.000000	0.357571	0.339802	-0.062656
...	...	...	...	...	...	...	...	...
L5 Flower and Concentrate	0.203100	-0.003475	-0.036552	0.004451	0.002925	0.204330	0.209075	-0.036662
L5 Dab Rig	0.203100	-0.003475	-0.036552	0.004451	0.002925	0.204330	0.209075	-0.036662
L5 Concentrate	-0.016147	-0.002776	0.224632	0.005263	-0.003549	-0.016292	-0.018740	0.223690
L5 Tools	-0.012195	-0.002695	-0.015018	-0.010152	-0.023452	-0.012151	-0.011833	-0.014800
Number of Product Types	0.371263	0.004982	0.046585	0.084542	0.371223	0.370932	0.366635	0.045859

200 rows × 200 columns

In [37]:
```python
import seaborn as sns

plt.subplots(figsize=(20,15))
sns.heatmap(alo.corr())
```

Out[37]: <AxesSubplot:>



Basic Correlation stuff above. I decided later that because a lot of these categories are useless (and some other columns as well), to remove many columns based on that. As indicated in the report, the creation of these columns for categories was largely a result of my original goal of trying to predict a certain product's sale, rather than a brand's... in which case the category of the product is much more useful as certain products sell for more.

In [ ]:

In [38]:
```
pd.set_option('display.max_rows', 200)
pd.set_option('display.max_columns', 200)
alo.head(200)
```

12	101 Cannabis Co.	2020-06-01	426.15040	-0.312484	33.114497	-0.038535	14111.700000	619.
13	101 Cannabis Co.	2020-07-01	589.71930	0.383829	32.131407	-0.029688	18948.500000	426.
14	101 Cannabis Co.	2020-08-01	1018.57000	0.727218	32.146382	0.000466	32743.400000	589.
15	101 Cannabis Co.	2020-09-01	1408.85000	0.383160	31.827140	-0.009931	44839.600000	1018.
16	101 Cannabis Co.	2020-10-01	1148.96000	-0.184468	30.375108	-0.045622	34899.800000	1408.
17	101 Cannabis	2020-	447.16050	0.610814	33.782933	0.112191	15106.300000	1148.

In [39]:
```
alo = alo[alo["Total Sales ($)"].notna()]
```

In [40]:
```
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
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

#drop all rows with NaN for total sales -> of no use to us.
alo = alo[alo["Total Sales ($)"].notna()]

# drop columns with very weak correlation to Total Sales
corr_matrix = alo.corr()
alo.drop(columns = corr_matrix["Total Sales ($)"].sort_values(ascending=False).lo


cookies = alo[alo["Brands"] == "Cookies"]
aloha = alo
alo = alo.drop("Brands", axis = 1) #at this point, everything I need to know abou
#in the other columns because everything is brand-specific i.e. L1 Inhalables is
#so I don't even need it anymore.

# get label
sales = alo["Total Sales ($)"]

categorical_features = ["Months"]

# remove Month basically lol
aloX = alo.drop("Total Sales ($)", axis=1)
alo_nums = aloX.drop(columns=categorical_features)

# list the numerical features
numerical_features = list(alo_nums)

# list of categorical features
class AugmentFeatures(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        ARP_Annual = X["Average Sales for this Year"] / X["Average Units Sold for
        ARP_Month_of_Year = X["Average Sales for this Month of Year"] / X["Averag
        ARP_Cumu = X["Cumulative Average Monthly Sales"] /  X["Cumulative Average
        return np.c_[X, ARP_Annual, ARP_Month_of_Year, ARP_Cumu]


num_pipeline = Pipeline([
    ('attribs_adder', AugmentFeatures()),
    ('imputer', SimpleImputer(strategy="constant", fill_value=0)),
    ('std_scaler', StandardScaler()),
])

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, numerical_features),
```

```
    ("cat", OneHotEncoder(), categorical_features),
])

aloe = full_pipeline.fit_transform(alo)
```

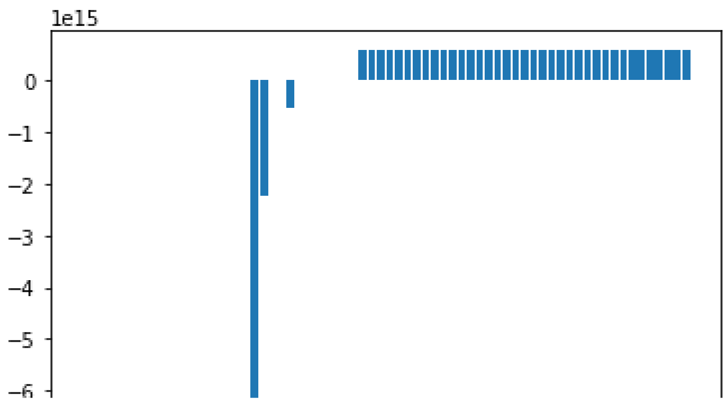
In [ ]:

In [41]:
```
# a, b, validation_train, validation_test = train_test_split(aloe, sales, test_si
# X_train, X_test, y_train, y_test = train_test_split(a, b, test_size=0.15)

X_train, X_test, y_train, y_test = train_test_split(aloe, sales, test_size=0.2)

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

predicted = lin_reg.predict(X_test)
regression_results(y_test, predicted)
```

```
8768      10878.90
6187      54190.00
19976    559891.00
20391      4444.26
17886    448174.00
           ...
17148    146393.00
8518     301295.00
25735      1264.50
5592     847062.00
7825       5838.59
Name: Total Sales ($), Length: 5056, dtype: float64
[ 12796.75   68337.625 562675.75  ...   -857.375 596582.625   8352.875]
explained_variance:  0.8428
r2:  0.8428
MAE:  37925.2627
MSE:  5956123666.7128
RMSE:  77175.9267
```

In [55]:
```python
from matplotlib import pyplot

importance = lin_reg.coef_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
    # plot feature importance
    pyplot.bar([x for x in range(len(importance))], importance)
    pyplot.show()
```


Feature: 56, Score: 570011614257578.62500



In [42]:
```python
aloe.shape

#That's a lot of columns... so let's apply PCA
```

Out[42]: (25279, 68)

In [43]:
```python
from sklearn import decomposition

# First we create a PCA object with the 5 components as a parameter
pca = decomposition.PCA(n_components=5)

# Now we run the fit operation to convert our
# data to a PCA transformmed data
my_pca = pca.fit_transform(aloe)
```

In [44]:
```python
my_pca.shape
```

Out[44]: (25279, 5)

In [45]:
```
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(my_pca, sales

new_lin_reg = LinearRegression()
new_lin_reg.fit(new_X_train, new_y_train)

new_predicted = new_lin_reg.predict(new_X_test)
regression_results(new_y_test, new_predicted)
```

```
10184      8949.5
14044    227501.0
214      630737.0
25747     11611.1
14940    172428.0
           ...
4443      56186.4
23516    128196.0
2777     169458.0
413       14502.5
11257    200517.0
Name: Total Sales ($), Length: 5056, dtype: float64
[ 13614.81823535 256161.52089473 437345.39548284 ... 185771.28848582
   6131.388766   198203.19404728]
explained_variance:  0.7667
r2:  0.7666
MAE:  48333.9412
MSE:  8643912260.9985
RMSE:  92972.6425
```

In [46]:
```python
X_train, X_test, y_train, y_test = train_test_split(aloe, sales, test_size=0.2)

from sklearn.datasets import make_friedman1
from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(
     n_estimators=100, learning_rate=0.1, max_depth=1, random_state=0).fit(X_trai
#but they were arbitrarily chosen. may need cross validation to optimize these pr
#and etc.

gbr_results = gbr.predict(X_test)
regression_results(y_test, gbr_results)
```

```
13796      5848.92
12397     31712.00
7799      97719.30
23641    216149.00
16337     77426.50
           ...
16798     50182.60
553       69510.40
20942      1163.68
24066    325654.00
22365     39825.00
Name: Total Sales ($), Length: 5056, dtype: float64
[  8379.51235308  26540.738697   114593.78148329 ...  14571.2276721
 428767.14953926 156749.64277901]
explained_variance:  0.8167
r2:  0.8166
MAE:  42201.9331
MSE:  6533034862.128
RMSE:  80827.1914
```

In [47]:
```python
from sklearn.model_selection import KFold
from sklearn import model_selection

# First we define our cross-validation model parameters. In this case we're going
# where we first shuffle our data before splitting it, and use a random seed to e
kfold = model_selection.KFold(n_splits=7, random_state=42, shuffle=True)

# Next we define the classifier we will be using for our model (we simply reuse t
model_kfold = LinearRegression()

# Finally we pull it all together. We call cross val score to generate an accurac
# we define our learning model, data, labels, and cross-val splitting strategy (c
results_kfold = model_selection.cross_val_score(model_kfold, aloe, sales, cv=kfol

# Because we're collecting results from all runs, we take the mean value
print("Accuracy: %.2f%%" % (results_kfold.mean()*100.0))
```

```
Accuracy: 84.26%
```

In [48]:
```
# First we define our cross-validation model parameters. In this case we're going
# where we first shuffle our data before splitting it, and use a random seed to e
kfold2 = model_selection.KFold(n_splits=7, random_state=42, shuffle=True)

# Next we define the classifier we will be using for our model (we simply reuse t
model_kfold2 =  GradientBoostingRegressor(
     n_estimators=100, learning_rate=0.1, max_depth=1, random_state=0).fit(X_trai

# Finally we pull it all together. We call cross val score to generate an accurac
# we define our learning model, data, labels, and cross-val splitting strategy (c
results_kfold2 = model_selection.cross_val_score(model_kfold2, aloe, sales, cv=kf

# Because we're collecting results from all runs, we take the mean value
print("Accuracy: %.2f%%" % (results_kfold2.mean()*100.0))
```

```
Accuracy: 82.14%
```

In [49]:
```
from sklearn import datasets
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor

parameters = {'learning_rate': [0.1,0.2],
              'n_estimators' : [50, 100],
              'max_depth'    : [1,3]
             }

grid_GBR = GridSearchCV(estimator=gbr, param_grid = parameters, cv = 2, n_jobs=-1
abc = grid_GBR.fit(X_train, y_train)

print(" Results from Grid Search " )
print("\n The best estimator across ALL searched params:\n",grid_GBR.best_estimat
print("\n The best score across ALL searched params:\n",grid_GBR.best_score_)
print("\n The best parameters across ALL searched params:\n",grid_GBR.best_params

gbr_results = grid_GBR.best_estimator_.predict(X_test)
regression_results(y_test, gbr_results)

#THIS IS SO SLOW, couldn't add more parameters because of lag
```

```
 Results from Grid Search 

 The best estimator across ALL searched params:
 GradientBoostingRegressor(learning_rate=0.2, random_state=0)

 The best score across ALL searched params:
 0.8862534812902025

 The best parameters across ALL searched params:
 {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100}
13796      5848.92
12397     31712.00
7799      97719.30
23641    216149.00
16337     77426.50
           ...    
16798     50182.60
553       69510.40
20942      1163.68
24066    325654.00
22365     39825.00
Name: Total Sales ($), Length: 5056, dtype: float64
[  4539.55702373  31462.87872194 130201.71232353 ...   1229.37474065
 481628.08653585  76113.29808946]
explained_variance:  0.8955
r2:  0.8955
MAE:  27369.0355
MSE:  3723519682.4119
RMSE:  61020.6496
```

In [50]:
```python
from sklearn import linear_model

clf = linear_model.Lasso(alpha=0.1)

parameters = {'alpha': [0.1,0.3,0.5,0.7,1]}

grid_clf = GridSearchCV(estimator=clf, param_grid = parameters, cv = 2, n_jobs=-1
res = grid_clf.fit(X_train, y_train)

print(" Results from Grid Search " )
print("\n The best estimator across ALL searched params:\n",res.best_estimator_)
print("\n The best score across ALL searched params:\n",res.best_score_)
print("\n The best parameters across ALL searched params:\n",res.best_params_)

clf_results = res.best_estimator_.predict(X_test)
regression_results(y_test, clf_results)
```

```
 Results from Grid Search

 The best estimator across ALL searched params:
 Lasso(alpha=1)

 The best score across ALL searched params:
 0.8429486899832297

 The best parameters across ALL searched params:
 {'alpha': 1}
13796      5848.92
12397     31712.00
7799      97719.30
23641    216149.00
16337     77426.50
           ...
16798     50182.60
553       69510.40
20942      1163.68
24066    325654.00
22365     39825.00
Name: Total Sales ($), Length: 5056, dtype: float64
[ -1062.66971513  41587.895227   120492.44156441 ...   2885.41870269
 494200.7976688  224766.32610326]
explained_variance:  0.8401
r2:  0.8401
MAE:  38595.5978
MSE:  5698107791.7346
RMSE:  75485.8119

C:\Users\cxcha\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_des
cent.py:530: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 85627102758.67188, tolerance: 7
5095079834.48248
  model = cd_fast.enet_coordinate_descent(
```

In [ ]:
```
def me_predictor(brand, month, units, arp) #<- takes string, time, int, int)
    if(aloha[aloha["Brand"] == brand].empty or aloha[aloha["Months"] == month].em
        print("Error... Brand or month does not exist")
    else:
        aloha = aloha[aloha["Brand"] == brand]
        if (units <= 0):
            aloha["Total Units"] = aloha["Cumulative Average Monthly Sales"]
        if (arp <= 0):
            aloha["ARP"] = aloha["Average Sales for this Year"] / aloha["Average

    aloha = aloha.drop("Brands", axis = 1)
    grid_gbr.predict(full_pipeline.fit_transform(aloha.iloc[0]))

#CURRENTLY DOESNT DO STUFF OUTSIDE OF OUR TIME FRAME. HOWEVER: we can use a predi
#and use the latest month instead, and then multiply it with an expected growth r
#in that marijuana sales is rapidly increasing and expected to grow, for example,
#we can then say if it's 6 months ahead, then we can guesstimate our prediction o
#in these cases.
#I will assume that retroactive prediction is not useful for us in some weird hyp
#goes before marijuana was even legalized.
```