project3

December 2, 2021

[1]:
```
import numpy as np
import pandas as pd
import datetime
```

[2]:
```
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)
    mean_squared_log_error=metrics.mean_squared_log_error(y_true, y_pred)
    median_absolute_error=metrics.median_absolute_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    print('explained_variance: ', round(explained_variance,4))
    print('mean_squared_log_error: ', round(mean_squared_log_error,4))
    print('r2: ', round(r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))
```

[3]:
```
# Read the files that we need

dataunits = pd.read_csv('data/BrandTotalUnits.csv')
datasales = pd.read_csv('data/BrandTotalSales.csv')
dataapr = pd.read_csv('data/BrandAverageRetailPrice.csv')
branddetails = pd.read_csv('data/BrandDetails.csv')
```

[4]:
```
# change data types

dataunits['Months'] = pd.to_datetime(dataunits['Months'])
dataunits['Total Units'] = dataunits['Total Units'].str.replace(',', '')
dataunits['Total Units'] = dataunits['Total Units'].str[:8]
dataunits['Total Units'] = pd.to_numeric(dataunits['Total Units'])
#dataunits['Total Units'] = dataunits['Total Units'].fillna(0)
```

[5]:
```
dataunits.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
```

[6]:
```
# change data types

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

[7]:
```
datasales.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
```

[8]:
```
# change data types

dataapr['Months'] = pd.to_datetime(dataunits['Months'])
```

[9]:
```
dataapr.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  datetime64[ns]
 2   ARP               25279 non-null  float64
 3   vs. Prior Period  24499 non-null  float64
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 850.5+ KB
```

[10]:
```
dataunits = dataunits.merge(datasales, how='outer',␣
 ↪left_on=["Brands","Months"], right_on=["Brand","Months"])
dataunits = dataunits.drop(['Brand'], 1)

# Fill the missing value with mean of each brand
dataunits['Total Units'] = dataunits.groupby("Brands")["Total Units"].
 ↪transform(lambda x: x.fillna(x.mean()))
dataunits['Total Sales ($)'] = dataunits.groupby("Brands")["Total Sales ($)"].
 ↪transform(lambda x: x.fillna(x.mean()))
dataunits["ARP"] = dataunits["Total Sales ($)"]/dataunits["Total Units"]
dataunits['Year'] = pd.DatetimeIndex(dataunits['Months']).year
dataunits['Month'] = pd.DatetimeIndex(dataunits['Months']).month
dataunits["Product Count"] = 0
dataunits['Rolling Average'] = 0
dataunits['Aggregate Sales'] = 0

# Add L1 features
dataunits["Ingestibles"] = 0
dataunits["Inhaleables"] = 0
dataunits["All accessories"] = 0
dataunits["Other Cannabis"] = 0
dataunits["Topicals"] = 0

# Add L2 features
dataunits['Concentrates'] = 0
dataunits["Pre-Rolled"] = 0
dataunits["Flower"] = 0
dataunits["Edibles"] = 0
dataunits["Sublinguals"] = 0
dataunits["Other Cannabis_L2"] = 0
dataunits['Topicals_L2'] = 0
dataunits["Devices"] = 0
dataunits["Shake/Trim/Lite"] = 0
dataunits["Accessories"] = 0
dataunits["Apparel"] = 0
dataunits["Non Infused Food"] = 0
dataunits["Grow Supplies"] = 0

value = 0

# Length in market of the brand
dataunits['Length_'] = dataunits.groupby('Brands')['Months'].diff().dt.days.
 ↪fillna(0).astype(int)
dataunits['Length'] = dataunits.groupby(['Brands'])['Length_'].cumsum()
dataunits['Length'] = dataunits['Length'] // 30
dataunits.drop('Length_', axis=1, inplace=True)
```

```
# for-loop to calculate new features that we will add
brands = dataunits["Brands"].unique()
for brand in brands:
    dataunits['Rolling Average'].loc[dataunits["Brands"] == brand] =␣
↪(dataunits[dataunits.Brands == brand].loc[:,'Total Units'].shift(1) +␣
↪dataunits[dataunits.Brands == brand].loc[:,'Total Units'].shift(2) +␣
↪dataunits[dataunits.Brands == brand].loc[:,'Total Units'])/3
    dataunits['Aggregate Sales'].loc[dataunits["Brands"] == brand] =␣
↪dataunits[dataunits.Brands == brand].loc[:,'Total Sales ($)'].shift(1) +␣
↪dataunits[dataunits.Brands == brand].loc[:,'Total Sales ($)'].shift(2) +␣
↪dataunits[dataunits.Brands == brand].loc[:,'Total Sales ($)']
    dataunits["Product Count"].loc[dataunits["Brands"] == brand] =␣
↪branddetails[branddetails.Brand == brand].Brand.count()
    dataunits['Total Units'].loc[dataunits["Brands"] == brand] =␣
↪dataunits['Total Units'].loc[dataunits["Brands"] == brand].fillna(value =␣
↪dataunits['Total Units'].loc[dataunits["Brands"] == brand].mean())

    # check L1 features
    if 'Inhaleables' in branddetails[branddetails.Brand == brand]['Category␣
↪L1'].values:
        value = 1
    dataunits["Inhaleables"].loc[dataunits["Brands"] == brand] = value
    value = 0
    if 'Ingestibles' in branddetails[branddetails.Brand == brand]['Category␣
↪L1'].values:
        value = 1
    dataunits["Ingestibles"].loc[dataunits["Brands"] == brand] = value
    value = 0
    if 'All accessories' in branddetails[branddetails.Brand == brand]['Category␣
↪L1'].values:
        value = 1
    dataunits["All accessories"].loc[dataunits["Brands"] == brand] = value
    value = 0
    if 'Other Cannabis' in branddetails[branddetails.Brand == brand]['Category␣
↪L1'].values:
        value = 1
    dataunits["Other Cannabis"].loc[dataunits["Brands"] == brand] = value
    value = 0
    if 'Topicals' in branddetails[branddetails.Brand == brand]['Category L1'].
↪values:
        value = 1
    dataunits["Topicals"].loc[dataunits["Brands"] == brand] = value
    value = 0


    # check L2 features
```

```
   if 'Concentrates' in branddetails[branddetails.Brand == brand]['Category␣
↪L2'].values:
       value = 1
   dataunits["Concentrates"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Pre-Rolled' in branddetails[branddetails.Brand == brand]['Category L2'].
↪values:
       value = 1
   dataunits["Pre-Rolled"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Flower' in branddetails[branddetails.Brand == brand]['Category L2'].
↪values:
       value = 1
   dataunits["Flower"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Edibles' in branddetails[branddetails.Brand == brand]['Category L2'].
↪values:
       value = 1
   dataunits["Edibles"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Sublinguals' in branddetails[branddetails.Brand == brand]['Category␣
↪L2'].values:
       value = 1
   dataunits["Sublinguals"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Other Cannabis' in branddetails[branddetails.Brand == brand]['Category␣
↪L2'].values:
       value = 1
   dataunits["Other Cannabis_L2"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Topicals' in branddetails[branddetails.Brand == brand]['Category L2'].
↪values:
       value = 1
   dataunits["Topicals_L2"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Devices' in branddetails[branddetails.Brand == brand]['Category L2'].
↪values:
       value = 1
   dataunits["Devices"].loc[dataunits["Brands"] == brand] = value
   value = 0
   if 'Shake/Trim/Lite' in branddetails[branddetails.Brand == brand]['Category␣
↪L2'].values:
       value = 1
   dataunits["Shake/Trim/Lite"].loc[dataunits["Brands"] == brand] = value
   value = 0
```

```
    if 'Accessories' in branddetails[branddetails.Brand == brand]['Category␣
↪L2'].values:
        value = 1
    dataunits["Accessories"].loc[dataunits["Brands"] == brand] = value
    value = 0
    if 'Apparel' in branddetails[branddetails.Brand == brand]['Category L2'].
↪values:
        value = 1
    dataunits["Apparel"].loc[dataunits["Brands"] == brand] = value
    value = 0
    if 'Non Infused Food' in branddetails[branddetails.Brand ==␣
↪brand]['Category L2'].values:
        value = 1
    dataunits["Non Infused Food"].loc[dataunits["Brands"] == brand] = value
    value = 0
    if 'Grow Supplies' in branddetails[branddetails.Brand == brand]['Category␣
↪L2'].values:
        value = 1
    dataunits["Grow Supplies"].loc[dataunits["Brands"] == brand] = value
    value = 0




#dataunits = dataunits.drop(["Months"],axis=1)
#dataunits = dataunits.drop(["vs. Prior Period_y"],axis=1)
```

```
/var/folders/tj/xyv4wj516tq0rhtzfz2b5d2h0000gn/T/ipykernel_7749/2114397665.py:2:
FutureWarning: In a future version of pandas all arguments of DataFrame.drop
except for the argument 'labels' will be keyword-only
  dataunits = dataunits.drop(['Brand'], 1)
/Users/qingyuanpan/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1732: 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/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

[11]:
```
# Fill null with the mean value, grouby the brands

dataunits['Rolling Average'] = dataunits.groupby("Brands")["Rolling Average"].
↪transform(lambda x: x.fillna(x.mean()))
dataunits['Aggregate Sales'] = dataunits.groupby("Brands")["Aggregate Sales"].
↪transform(lambda x: x.fillna(x.mean()))
```

[12]:
```
dataunits.head(20)
```

[12]:

	Brands	Months	Total Units	vs. Prior Period	\
0	#BlackSeries	2020-08-01	1616.339000	NaN	
1	#BlackSeries	2020-09-01	1032.846967	-1.000000	
2	#BlackSeries	2021-01-01	715.532800	NaN	
3	#BlackSeries	2021-02-01	766.669100	0.071466	
4	#BlackSeries	2021-03-01	1032.846967	-1.000000	
5	101 Cannabis Co.	2019-11-01	131.067700	NaN	
6	101 Cannabis Co.	2019-12-01	566.557827	-1.000000	
7	101 Cannabis Co.	2020-01-01	345.413400	NaN	
8	101 Cannabis Co.	2020-02-01	696.658400	1.016883	
9	101 Cannabis Co.	2020-03-01	943.393300	0.354169	
10	101 Cannabis Co.	2020-04-01	712.498100	-0.244750	
11	101 Cannabis Co.	2020-05-01	619.841000	-0.130045	
12	101 Cannabis Co.	2020-06-01	426.150400	-0.312484	
13	101 Cannabis Co.	2020-07-01	589.719300	0.383829	
14	101 Cannabis Co.	2020-08-01	1018.574000	0.727218	
15	101 Cannabis Co.	2020-09-01	1408.850000	0.383160	
16	101 Cannabis Co.	2020-10-01	1148.962000	-0.184468	
17	101 Cannabis Co.	2020-11-01	447.160500	-0.610814	
18	101 Cannabis Co.	2020-12-01	337.960500	-0.244208	
19	101 Cannabis Co.	2021-01-01	250.232000	-0.259582	

	Total Sales ($)	ARP	Year	Month	Product Count	Rolling Average	\
0	25352.130000	15.684909	2020	8	4	932.757389	
1	14731.451667	14.262957	2020	9	4	932.757389	
2	9739.423000	13.611428	2021	1	4	1121.572922	
3	9102.802000	11.873182	2021	2	4	838.349622	
4	14731.451667	14.262957	2021	3	4	838.349622	
5	4465.040000	34.066669	2019	11	77	585.324868	
6	18276.247000	32.258396	2019	12	77	585.324868	
7	11790.660000	34.134924	2020	1	77	347.679642	
8	20266.760000	29.091388	2020	2	77	536.209876	
9	30465.470000	32.293498	2020	3	77	661.821700	
10	23465.650000	32.934333	2020	4	77	784.183267	
11	21348.390000	34.441720	2020	5	77	758.577467	
12	14111.750000	33.114483	2020	6	77	586.163167	
13	18948.510000	32.131406	2020	7	77	545.236900	
14	32743.470000	32.146383	2020	8	77	678.147900	
15	44839.680000	31.827150	2020	9	77	1005.714433	
16	34899.870000	30.375130	2020	10	77	1192.128667	
17	15106.390000	33.782926	2020	11	77	1001.657500	
18	11883.010000	35.160943	2020	12	77	644.694333	
19	8059.176000	32.206816	2021	1	77	345.117667	

	…	Sublinguals	Other Cannabis_L2	Topicals_L2	Devices	\
0	…	0	0	0	0	
1	…	0	0	0	0	

```
2   …              0                  0            0        0
3   …              0                  0            0        0
4   …              0                  0            0        0
5   …              0                  0            0        0
6   …              0                  0            0        0
7   …              0                  0            0        0
8   …              0                  0            0        0
9   …              0                  0            0        0
10  …              0                  0            0        0
11  …              0                  0            0        0
12  …              0                  0            0        0
13  …              0                  0            0        0
14  …              0                  0            0        0
15  …              0                  0            0        0
16  …              0                  0            0        0
17  …              0                  0            0        0
18  …              0                  0            0        0
19  …              0                  0            0        0

    Shake/Trim/Lite  Accessories  Apparel  Non Infused Food  Grow Supplies  \
0                 0            0        0                 0              0
1                 0            0        0                 0              0
2                 0            0        0                 0              0
3                 0            0        0                 0              0
4                 0            0        0                 0              0
5                 0            0        0                 0              0
6                 0            0        0                 0              0
7                 0            0        0                 0              0
8                 0            0        0                 0              0
9                 0            0        0                 0              0
10                0            0        0                 0              0
11                0            0        0                 0              0
12                0            0        0                 0              0
13                0            0        0                 0              0
14                0            0        0                 0              0
15                0            0        0                 0              0
16                0            0        0                 0              0
17                0            0        0                 0              0
18                0            0        0                 0              0
19                0            0        0                 0              0

    Length
0        0
1        1
2        5
3        6
4        7
```

```
5        0
6        1
7        2
8        3
9        4
10       5
11       6
12       7
13       8
14       9
15      10
16      11
17      12
18      13
19      14

[20 rows x 30 columns]
```

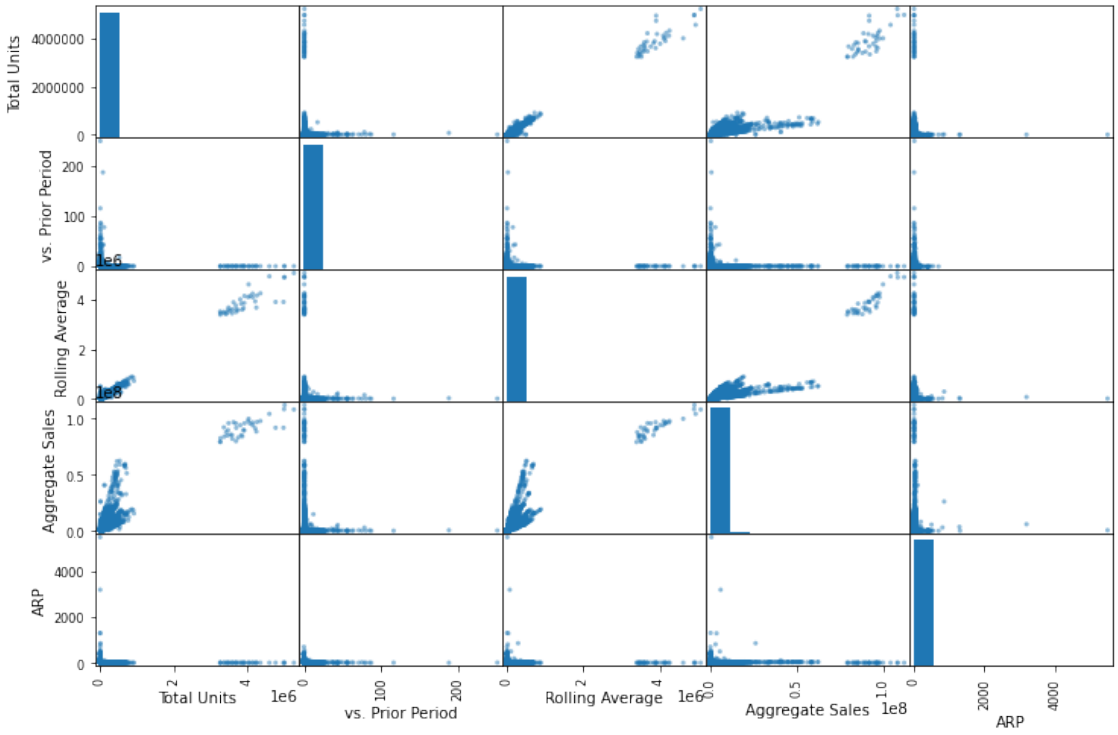
[13]:
```
# Correlation with features and label

from pandas.plotting import scatter_matrix
corr_matrix = dataunits.corr()
print(corr_matrix["Total Sales ($)"].sort_values(ascending=False))
attributes = ["Total Units", "vs. Prior Period", "Rolling Average",
              "Aggregate Sales", "ARP"]
scatter_matrix(dataunits[attributes], figsize=(12, 8))
#save_fig("correlation_matrix")
```

```
Total Sales ($)      1.000000
Aggregate Sales      0.991939
Total Units          0.873737
Rolling Average      0.870348
Product Count        0.504025
Devices              0.161291
Inhaleables          0.133795
Concentrates         0.128036
Other Cannabis_L2    0.114537
Other Cannabis       0.114537
Length               0.112535
Edibles              0.098954
Ingestibles          0.090269
Flower               0.089698
Pre-Rolled           0.078196
Sublinguals          0.074772
Topicals             0.062194
Topicals_L2          0.062194
Apparel              0.032085
```

```
Grow Supplies         0.032085
Non Infused Food      0.023848
Shake/Trim/Lite       0.005336
Month                 0.003241
ARP                   0.002643
vs. Prior Period     -0.009732
Year                 -0.010916
Accessories          -0.023692
All accessories            NaN
Name: Total Sales ($), dtype: float64
```

[13]:
```
array([[<AxesSubplot:xlabel='Total Units', ylabel='Total Units'>,
        <AxesSubplot:xlabel='vs. Prior Period', ylabel='Total Units'>,
        <AxesSubplot:xlabel='Rolling Average', ylabel='Total Units'>,
        <AxesSubplot:xlabel='Aggregate Sales', ylabel='Total Units'>,
        <AxesSubplot:xlabel='ARP', ylabel='Total Units'>],
       [<AxesSubplot:xlabel='Total Units', ylabel='vs. Prior Period'>,
        <AxesSubplot:xlabel='vs. Prior Period', ylabel='vs. Prior Period'>,
        <AxesSubplot:xlabel='Rolling Average', ylabel='vs. Prior Period'>,
        <AxesSubplot:xlabel='Aggregate Sales', ylabel='vs. Prior Period'>,
        <AxesSubplot:xlabel='ARP', ylabel='vs. Prior Period'>],
       [<AxesSubplot:xlabel='Total Units', ylabel='Rolling Average'>,
        <AxesSubplot:xlabel='vs. Prior Period', ylabel='Rolling Average'>,
        <AxesSubplot:xlabel='Rolling Average', ylabel='Rolling Average'>,
        <AxesSubplot:xlabel='Aggregate Sales', ylabel='Rolling Average'>,
        <AxesSubplot:xlabel='ARP', ylabel='Rolling Average'>],
       [<AxesSubplot:xlabel='Total Units', ylabel='Aggregate Sales'>,
        <AxesSubplot:xlabel='vs. Prior Period', ylabel='Aggregate Sales'>,
        <AxesSubplot:xlabel='Rolling Average', ylabel='Aggregate Sales'>,
        <AxesSubplot:xlabel='Aggregate Sales', ylabel='Aggregate Sales'>,
        <AxesSubplot:xlabel='ARP', ylabel='Aggregate Sales'>],
       [<AxesSubplot:xlabel='Total Units', ylabel='ARP'>,
        <AxesSubplot:xlabel='vs. Prior Period', ylabel='ARP'>,
        <AxesSubplot:xlabel='Rolling Average', ylabel='ARP'>,
        <AxesSubplot:xlabel='Aggregate Sales', ylabel='ARP'>,
        <AxesSubplot:xlabel='ARP', ylabel='ARP'>]], dtype=object)
```



[14]:
```
#data = dataunits.loc[dataunits["Total Sales ($)"] < 100000].copy()
#data = data.loc[data["Total Sales ($)"] > 50000].copy()

# save a copy to use
data = dataunits.copy()
```

[15]:
```
data.head()
```

[15]:

	Brands	Months	Total Units	vs. Prior Period	Total Sales ($)	\
0	#BlackSeries	2020-08-01	1616.339000	NaN	25352.130000	
1	#BlackSeries	2020-09-01	1032.846967	-1.000000	14731.451667	
2	#BlackSeries	2021-01-01	715.532800	NaN	9739.423000	
3	#BlackSeries	2021-02-01	766.669100	0.071466	9102.802000	
4	#BlackSeries	2021-03-01	1032.846967	-1.000000	14731.451667	

	ARP	Year	Month	Product Count	Rolling Average	…	Sublinguals	\
0	15.684909	2020	8	4	932.757389	…	0	
1	14.262957	2020	9	4	932.757389	…	0	
2	13.611428	2021	1	4	1121.572922	…	0	
3	11.873182	2021	2	4	838.349622	…	0	
4	14.262957	2021	3	4	838.349622	…	0	

	Other	Cannabis_L2	Topicals_L2	Devices	Shake/Trim/Lite	Accessories	\

```
0                  0            0        0                 0            0
1                  0            0        0                 0            0
2                  0            0        0                 0            0
3                  0            0        0                 0            0
4                  0            0        0                 0            0

   Apparel  Non Infused Food  Grow Supplies  Length
0        0                 0              0       0
1        0                 0              0       1
2        0                 0              0       5
3        0                 0              0       6
4        0                 0              0       7

[5 rows x 30 columns]
```

[16]:
```
# drop label and non-necessary features

data_label = data["Total Sales ($)"].copy()
data = data.drop("Total Sales ($)", axis=1)
data = data.drop("Months",axis=1)
data = data.drop("Brands",axis=1)
```

[17]:
```
data.head(5)
```

[17]:
```
   Total Units  vs. Prior Period        ARP  Year  Month  Product Count  \
0  1616.339000               NaN  15.684909  2020      8              4
1  1032.846967         -1.000000  14.262957  2020      9              4
2   715.532800               NaN  13.611428  2021      1              4
3   766.669100          0.071466  11.873182  2021      2              4
4  1032.846967         -1.000000  14.262957  2021      3              4

   Rolling Average  Aggregate Sales  Ingestibles  Inhaleables  …  \
0       932.757389     38990.119333            0            1  …
1       932.757389     38990.119333            0            1  …
2      1121.572922     49823.004667            0            1  …
3       838.349622     33573.676667            0            1  …
4       838.349622     33573.676667            0            1  …

   Sublinguals  Other Cannabis_L2  Topicals_L2  Devices  Shake/Trim/Lite  \
0            0                  0            0        0                0
1            0                  0            0        0                0
2            0                  0            0        0                0
3            0                  0            0        0                0
4            0                  0            0        0                0

   Accessories  Apparel  Non Infused Food  Grow Supplies  Length
0            0        0                 0              0       0
```

1	0	0	0	0	1
2	0	0	0	0	5
3	0	0	0	0	6
4	0	0	0	0	7

[5 rows x 27 columns]

[18]:
```
# label encoder

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data_label = le.fit_transform(data_label)
data_label
```

[18]:
```
array([9167, 6907, 5402, …, 6159, 3450, 5860])
```

[19]:
```
# The pipeline

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

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

from sklearn.base import BaseEstimator, TransformerMixin


imputer = SimpleImputer(strategy="median")
data_num = data.drop(["Ingestibles", "Inhaleables", "All accessories", "Other␣
↪Cannabis", "Topicals","Year", "Month",
                      "Flower", 'Concentrates', 'Pre-Rolled', 'Topicals_L2',␣
↪'Edibles', 'Devices', 'Sublinguals',
                      'Other Cannabis_L2', 'Accessories', 'Non Infused Food',␣
↪'Apparel', 'Grow Supplies', 'Shake/Trim/Lite'], axis=1)

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

data_num_tr = num_pipeline.fit_transform(data_num)
numerical_features = list(data_num)
categorical_features = ["Ingestibles", "Inhaleables", "All accessories", "Other␣
↪Cannabis", "Topicals","Year", "Month",
```

```
                    "Flower", 'Concentrates', 'Pre-Rolled', 'Topicals_L2',␣
 ↪'Edibles', 'Devices', 'Sublinguals',
                    'Other Cannabis_L2', 'Accessories', 'Non Infused Food',␣
 ↪'Apparel', 'Grow Supplies', 'Shake/Trim/Lite']

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(sparse=False, categories='auto'),␣
 ↪categorical_features),
    ])

data_prepared = full_pipeline.fit_transform(data)

print(data_prepared)
```

```
[[-0.16241267 -0.09237606 -0.1664302  … 0.          1.
  0.        ]
 [-0.16611303 -0.39470865 -0.19734558 … 0.          1.
  0.        ]
 [-0.16812535 -0.09237606 -0.21151081 … 0.          1.
  0.        ]
 …
 [-0.17045579 -0.15482569  0.24364159 … 0.          1.
  0.        ]
 [-0.1718011  -0.2697429   0.29373115 … 0.          1.
  0.        ]
 [-0.17066416 -0.39470865  0.2596952  … 0.          1.
  0.        ]]
```

[66]:
```
# data split

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_prepared, data_label,␣
 ↪test_size = 0.15, random_state=200)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(23533, 58)
(4153, 58)
(23533,)
(4153,)
```

[67]:
```
# Fit into Linear Regression
from sklearn.linear_model import LinearRegression
```

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


y_pred = lin_reg.predict(X_test)
```

[69]:
```
regression_results(y_test, y_pred)
```

```
explained_variance:  0.4653
mean_squared_log_error:  0.8463
r2:  0.4651
MAE:  4644.065
MSE:  31760404.5061
RMSE:  5635.637
```

[70]:
```
# check p-value, F stats and T stats

import statsmodels.api as sm
stats = sm.OLS(data_label, data_prepared)
results_stats = stats.fit()

print(results_stats.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.449
Model:                            OLS   Adj. R-squared:                  0.448
Method:                 Least Squares   F-statistic:                     643.5
Date:                Tue, 30 Nov 2021   Prob (F-statistic):               0.00
Time:                        22:23:28   Log-Likelihood:            -2.7849e+05
No. Observations:               27686   AIC:                         5.570e+05
Df Residuals:                   27650   BIC:                         5.573e+05
Df Model:                          35
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1          3838.4300    295.733     12.979      0.000    3258.780    4418.081
x2           178.8917     34.102      5.246      0.000     112.049     245.734
x3           -88.2774     34.819     -2.535      0.011    -156.525     -20.030
x4           506.3485     53.428      9.477      0.000     401.627     611.070
x5         -4723.7903    308.007    -15.337      0.000   -5327.500   -4120.080
x6          2272.7147    103.759     21.904      0.000    2069.341    2476.088
x7          2354.4982     45.842     51.361      0.000    2264.645    2444.351
x8          1934.3692    134.237     14.410      0.000    1671.258    2197.481
x9          1011.0193    129.327      7.818      0.000     757.532    1264.507
```

x10	309.8788	80.190	3.864	0.000	152.703	467.055
x11	2635.5097	73.349	35.931	0.000	2491.743	2779.277
const	2945.3886	69.159	42.589	0.000	2809.833	3080.944
x12	1218.2132	55.783	21.838	0.000	1108.875	1327.551
x13	1727.1753	53.460	32.308	0.000	1622.392	1831.959
x14	1517.4078	49.029	30.949	0.000	1421.309	1613.507
x15	1427.9808	53.567	26.658	0.000	1322.988	1532.974
x16	5554.4669	111.220	49.941	0.000	5336.471	5772.463
x17	2028.0256	64.094	31.642	0.000	1902.399	2153.652
x18	-781.8398	61.966	-12.617	0.000	-903.295	-660.384
x19	-3855.2642	86.549	-44.545	0.000	-4024.904	-3685.625
x20	1418.7079	119.147	11.907	0.000	1185.173	1652.243
x21	1057.9701	117.981	8.967	0.000	826.722	1289.218
x22	1319.9285	117.061	11.276	0.000	1090.484	1549.373
x23	988.0658	115.866	8.528	0.000	760.962	1215.170
x24	778.0286	113.797	6.837	0.000	554.981	1001.076
x25	380.3079	112.544	3.379	0.001	159.717	600.899
x26	299.3993	112.218	2.668	0.008	79.446	519.353
x27	114.7767	103.650	1.107	0.268	-88.383	317.937
x28	-273.4448	103.296	-2.647	0.008	-475.909	-70.980
x29	-629.5580	123.583	-5.094	0.000	-871.787	-387.329
x30	-1177.3216	122.771	-9.590	0.000	-1417.958	-936.685
x31	-1331.4719	122.618	-10.859	0.000	-1571.810	-1091.134
x32	185.4704	57.490	3.226	0.001	72.786	298.154
x33	2759.9182	64.386	42.865	0.000	2633.717	2886.119
x34	567.2162	58.377	9.716	0.000	452.795	681.638
x35	2378.1723	65.679	36.209	0.000	2249.438	2506.907
x36	291.0588	56.199	5.179	0.000	180.906	401.211
x37	2654.3298	61.354	43.262	0.000	2534.072	2774.587
x38	1517.4078	49.029	30.949	0.000	1421.309	1613.507
x39	1427.9808	53.567	26.658	0.000	1322.988	1532.974
x40	-449.5245	120.175	-3.741	0.000	-685.074	-213.975
x41	3394.9131	125.795	26.988	0.000	3148.349	3641.477
x42	1589.0442	68.027	23.359	0.000	1455.708	1722.380
x43	1356.3443	68.463	19.811	0.000	1222.152	1490.536
x44	323.5075	102.558	3.154	0.002	122.489	524.526
x45	2621.8811	107.543	24.380	0.000	2411.091	2832.671
x46	1218.2132	55.783	21.838	0.000	1108.875	1327.551
x47	1727.1753	53.460	32.308	0.000	1622.392	1831.959
x48	3083.3767	117.852	26.163	0.000	2852.382	3314.372
x49	-137.9881	118.752	-1.162	0.245	-370.748	94.772
x50	-429.0535	481.336	-0.891	0.373	-1372.496	514.389
x51	3374.4420	483.015	6.986	0.000	2427.708	4321.176
x52	-1797.6967	319.419	-5.628	0.000	-2423.774	-1171.619
x53	4743.0853	352.046	13.473	0.000	4053.058	5433.112
x54	-1797.6967	319.419	-5.628	0.000	-2423.774	-1171.619
x55	4743.0853	352.046	13.473	0.000	4053.058	5433.112
x56	-3531.7992	444.627	-7.943	0.000	-4403.290	-2660.308

```
x57         6477.1877    493.826     13.116      0.000    5509.264    7445.112
==============================================================================
Omnibus:                      202.179   Durbin-Watson:                   0.377
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              182.364
Skew:                          -0.156   Prob(JB):                     2.51e-40
Kurtosis:                       2.754   Cond. No.                     1.21e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 2.83e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

[71]:
```
data_prepared.shape
```

[71]:
```
(27686, 58)
```

[108]:
```
# try to use PCA to reduce dimension

from sklearn import decomposition

pca = decomposition.PCA(n_components=5)

data_pca = pca.fit_transform(data_prepared)
```

[109]:
```
data_pca.shape
```

[109]:
```
(27686, 5)
```

[110]:
```
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(data_pca,␣
↪data_label, test_size=0.15)
```

[111]:
```
lin_reg = LinearRegression()
lin_reg.fit(new_X_train, new_y_train)

new_y_pred = lin_reg.predict(new_X_test)
```

[112]:
```
regression_results(new_y_test, new_y_pred)
```

```
explained_variance:  0.2841
mean_squared_log_error:  0.9671
r2:  0.2839
MAE:  5433.6358
MSE:  41921615.8216
RMSE:  6474.6904
```

[189]:
```
# Use Gradient Boosting as the ensemble method

import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.datasets import make_friedman1
from sklearn.ensemble import GradientBoostingRegressor


est = GradientBoostingRegressor(
    n_estimators=100, learning_rate=1, max_depth=1, random_state=100).
 ↪fit(X_train, y_train)

y_pred = est.predict(X_test)


explained_variance=metrics.explained_variance_score(y_test, y_pred)
mean_absolute_error=metrics.mean_absolute_error(y_test, y_pred)
mse=metrics.mean_squared_error(y_test, y_pred)
median_absolute_error=metrics.median_absolute_error(y_test, y_pred)
r2=metrics.r2_score(y_test, 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))
```

```
explained_variance:  0.9761
r2:  0.9761
MAE:  749.8252
MSE:  1418480.9144
RMSE:  1191.0
```

[213]:
```
# Grid Search(Gradient Boosting) and 10-fold CV

param_grid={'learning_rate': [0.1,1,10]}

estimator = GradientBoostingRegressor(n_estimators=100, random_state=100)
grid = GridSearchCV(estimator=estimator, cv=10,␣
 ↪scoring='neg_mean_absolute_error', param_grid=param_grid)
grid_result = grid.fit(X_train, y_train)
```

[215]:
```
print('Best Score:', grid_result.best_score_)
```

```
Best Score: -389.59106260950824
```

[216]:
```
print('Best Params: ', grid_result.best_params_)
```

```
Best Params:  {'learning_rate': 0.1}
```

[199]:
```
# Grid Search(Lasso) and 10-fold CV

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
# Train model with default alpha=1
lasso = Lasso(alpha=1).fit(X_train, y_train)
# find optimal alpha with grid search
alpha = [0.001, 0.01, 0.1, 1, 10, 100, 1000,10000, 1000000]
param_grid = dict(alpha=alpha)
grid = GridSearchCV(estimator=lasso, param_grid=param_grid, cv=10,␣
 ↪scoring='neg_mean_absolute_error', verbose=1, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)
print('Best Score: ', grid_result.best_score_)
print('Best Params: ', grid_result.best_params_)
```

```
Fitting 10 folds for each of 9 candidates, totalling 90 fits
Best Score:  -4662.264280553139
Best Params:  {'alpha': 0.001}
```

```
/Users/qingyuanpan/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:530: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 13857784097.46399, tolerance: 135841389.92386362
  model = cd_fast.enet_coordinate_descent(
```

[198]:
```
# Grid Search(Ridge) and 10-fold CV

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
# Train model with default alpha=1
ridge = Ridge(alpha=1).fit(X_train, y_train)
# find optimal alpha with grid search
alpha = [0.001, 0.01, 0.1, 1, 10, 100, 1000,10000, 1000000]
param_grid = dict(alpha=alpha)
grid = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=10,␣
 ↪scoring='neg_mean_absolute_error', verbose=1, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)
print('Best Score: ', grid_result.best_score_)
print('Best Params: ', grid_result.best_params_)
```

```
Fitting 10 folds for each of 9 candidates, totalling 90 fits
Best Score:  -4662.263069821733
Best Params:  {'alpha': 0.001}
```

[186]:
```
# fit data into Multi-layer Regression

from sklearn.neural_network import MLPRegressor
```

```
regr = MLPRegressor(solver='adam', alpha=1, max_iter=500, random_state=200).
 ↪fit(X_train, y_train)
MLP_y_pred = regr.predict(X_test)
```

```
/Users/qingyuanpan/opt/anaconda3/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[187]:
```
regression_results(y_test, MLP_y_pred)
```

```
explained_variance:  0.8533
mean_squared_log_error:  0.4839
r2:  0.8533
MAE:  2182.2451
MSE:  8710332.3339
RMSE:  2951.3272
```

[ ]:
```

```

Project 3 Report

1. Executive Summary:

In this project, we use several raw data containing cannabis information from California to predict the **Total Sales** of different cannabis brands in different months. The basic steps are simply as follows: we preprocessed the data at the initial stage and extracted Total Sales as our label for prediction, selected some applicable models, trained and tested the cannabis data after it was manually processed, evaluated the performance of these models, and finally came up with some good fits and compared them to the original data. Our main goal is to use some applicable models to predict the month's sales of different cannabis brands.

The given datas are unprocessed, unrepresentative and containing missing values. First thing to do is to **preprocess** the data. We collect most of the useful data and merge them into one complete dataframe without missing value. This step is important since missing data may cause bias in the estimation of parameters and it may reduce the representativeness of the datas.

After preprocessing the data, it is time to select some appropriate models. Since we already have the total sales data from the original data, we remove that column and divide the remaining completed data into 85% and 15%, which are the train set and test set. Because we are predicting a numerical value, our initial selection of the model is the linear regression. In the process of constantly improving the model, we found that Linear Regression is not the optimal model. In order to improve the performance, we also develop the PCA method, implement K-fold Cross-Validation, employ a GridSearch method and so on to test and optimize our models and parameters.

The final step of coding is to compare the predicted sales result with the actual sales value through the model. We tested a few potential models and compared each regression results such as MAE and R square value. Different statistical methods and values reflect different meanings and results. As our final model, we tried the Multi-layer Perceptron Regressor to predict the sales and the result was better than what we got from the Linear Regression model.

There are several key findings: First of all, since we are predicting a numerical feature, which is the monthly sales, we applied Linear Regression as our first model. The R2 result for Linear Regression is 46.51% and MAE is around 4644, which is **not terrible but not ideal either**. We then started to implement Principle Component Analysis (PCA) to reduce dimensionality. But again, the results after PCA are **even worse**, with r2 around 28.39% and MAE around 5433. The correlation matrix showed that features are highly independent to each other and unlikely to be highly linear, thus we believed that PCA is an **inappropriate** strategy to improve the result.

Fortunately, we tested several other models such as Multi-layer Regression, Grid Search(Lasso), Grid Search(Ridge) and Gradient Boosting as the ensemble method and all the results were significantly improved. The highest result can be up to **r2: 97.61%** and **MAE: 749.8252.**

2. Background/Introduction:

Cookies is one of the most respected and top-selling cannabis brands in California and is globally recognized, amassing a stable of over 50 cannabis varieties and product lines including indoor and outdoor. Cookies was founded in 2012. It is one of the first companies to establish an identity and streetwear company that has come to dominate the industry.

We are hired as a Cookies consultant to develop and evaluate the performance and the predictive power of a model trained and tested on data collected from California cannabis brands. Once we get a good fit, we will use the model to predict the monthly sale price of not only Cookies, but some other brands with valuable information along. A model like this would be very valuable for agents like us who could make use of the information provided on a daily basis.

Based on our research, we found out along with Cookies, Tempo,NUG (CA), and ROVE Brands are some of the other high sales cannabis brands. Initially, we are very curious why some brands have higher sales while others are not, and the possible reasons behind it. Apparently, we cannot observe all these 1,123 brand data and thousands of related data information with the naked eye, but we can utilize the model to analyze the data for us and bring out the results we want. If we are able to predict the total sales accurately, we may figure out the reason. Unfortunately, based on finite time and current understanding of Data Science, we are unable to intuitively make judgment of the reason behind high total sales. After we extracted the information and employed the models several times, we still did not achieve the perfect fit, but some highly accurate ones. In fact, our models somewhat still did predict the monthly sales for each brand to some extent.

3. Methodology:

We select 4 given databases in this project, which are BrandTotalUnits.csv, BrandTotalSales.csv, BrandAverageRetailPrice.csv and BrandDetails.csv, respectively. 4 datas are raw and unprocessed datas, and all files contain some missing values. We also refer to DataDictionary.csv to help us understand the data better.

- BrandTotalUnits.csv:

    - The datafile contains 4 columns Brands, Months, Total Units, and vs. Prior Period. It indicates the total unit number of each brand in different months and the comparison data with the prior period.
- BrandTotalSales.csv:
    - The datafile contains 3 columns Months,Brand and Total Sales ($). It shows the monthly total sales of each brand in different months.
- BrandAverageRetailPrice.csv:
    - The datafile contains 4 columns Brands, Months, ARP, and vs. Prior Period. Again, It shows each brand in different months, and the ARP(AverageRetailPrice) value besides the compassion data with the prior period respectively.
- BrandDetails.csv:
    - The datafile contains 25 columns which are: State,Channel,Category L1,Category L2,Category L3,Category L4,Category L5,Brand,Product Description,Total Sales ($),Total Units,ARP,Flavor,Items Per Pack,Item Weight,Total THC,Total CBD,Contains CBD,Pax Filter,Strain,Is Flavored,Mood Effect,Generic Vendor,Generic Items,$5 Price Increment.
    - This data introduces the specific features of each cannabis brand. These data roughly correspond to the data of each cannabis brand in the Sales and Units data above in order. However, after importing and comparing the data into dataframe, we found that some rows are missing.
- DataDictionary.csv:
    - This data introduces the specific meaning of each label in the Details data above. We refer to it to decide which features should be utilized in the following data processing.

We select the above 4 data files as the main source of the database. Since all the data are unprocessed and containing some missing values and needed to be merged to one single dataframe, we decide to first preprocess the data. Some key packages in this project are as usual: **Pandas, Numpy and scikit-learn.** We imported the given csv files into dataframe. We slightly modified the time format of the data, and then merged the Sales and Units data into a new dataframe by brand name and time, named dataunits. For the null values in total units and total sales, we fill the mean that is grouped by the brand so that we will not have so many outliers in our data. Since we have noticed that time might be a key effect to the sales, we figured out and expanded some **Time Series features** such as rolling averages, aggregate sales(for nearest 3 months), and length of time in the market for each brand. We also found the rolling average and aggregate sales may contain null values, so we still fill them with mean of these values that are grouped by brand. The complete Time Series Features augmentation include: Rolling averages, Percent changes and month-to-month fluctuations, Year, Season/period, Aggregate sales and trends for previous timeframe and Length of time in market.

Besides Time Series features, we also added some Brand features. We pick some features from branddetails.csv and we simply **added all Category L1 and L2 features** to the data we use. We believe that they are the top 2 factors to predict the sales because Category L1 explains the highest-level category for each product and Category L2 explains the multiple product category of the next layer of categorization. In total, we added 18 categorical features based on Category L1 and L2.

We noticed that there are over 1000 brands in the market, so One Hot Encoding may blow up the feature space. Also if we keep the 'Brands' as a feature, the performance of our model is really bad, thus we decide to **drop the feature 'Brands**' and get better results. We also dropped 'Months' because we have already taken the year and month as features from it, and the type of Months is datetime which we need to do some process in order to use this feature, so we just simply dropped it. We also normalize the label column, which is the total sale, by Label Encoder to reduce redundancy and complexity.

After that, we applied **Pipeline** to the processed dataframe. Pipeline combines most of the data processing steps together. A full pipeline runs the processing steps for both numerical and categorical features and sticks them together. Firstly, we seperated numerical and categorical features from our dataframe. Categorical features are:

["Ingestibles", "Inhaleables", "All accessories", "Other Cannabis", "Topicals","Year", "Month", "Flower", 'Concentrates', 'Pre-Rolled', 'Topicals_L2', 'Edibles', 'Devices', 'Sublinguals', 'Other Cannabis_L2', 'Accessories', 'Non Infused Food', 'Apparel', 'Grow Supplies', 'Shake/Trim/Lite']

Which simply come from Category L1 and Category L2 and numerical features are the remaining from the data. We applied **One hot Encoding (OHE)** to change each unique categorical value into its own binary column. This process is necessary to fit the categorical features into the ML models for better prediction. We also defined our strategy for imputing numerical values. With the remaining NULL values in the 'vs. Prior Period' column, we decided to impute them with median values in order to preserve all cases of data. We skipped the process of augmenting features, which have already been done outside of Pipeline. We defined a standard scaling algorithm as a strategy for scaling features and the final step is to nestle numeric pipelines and categorical features. Pipeline is very useful in dealing with trials and errors in continuous model testing.

After finishing the pipeline, we have completed all the steps of data preprocessing and cleaning. we **splitted** the prepared data into 85% of the training set and 15% of the test set. The remaining steps are to test and predict in different models using training sets and test sets.

The next step is to apply several models to test our data and compare each performance. Obviously, there are many existing models from the package, and we cannot choose them arbitrarily. We applied some methodologies to refine and improve the selected model. In step 9, we employed boosting as an ensemble method because we have so far a pretty low score for the prediction after doing Linear Regression. We simply applied **Gradient Boosting** as the boosting method. The Gradient Boosting Model combines a set of weak learners into a strong learner to minimize training errors and improve variance.

Besides, we **cross-validate** our training results, using models from both Gradient Boosting model and Linear Regression model. The purpose of Cross Validation is to estimate the skill of a particular model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

4. Results:

We used correlation_matrix with label Total Sales ($), and we found that aggregate sales, total units, rolling average have very high positive correlation with the total sales. They are easy to understand, because the aggregate sales and rolling average are calculated by the past 3 months of this brand. Also the product count is also a high correlated feature, the reason might be the customer wants more choices.

The other features that we added from Category L1 and L2 are also useful according to the value of correlation matrix. We found that for some features, there are **some correlations between Category L1 and L2** as well. For example, if a product is Topicals in L1, it is also Topicals in L2. We also use OLS regression results for feature selection after the pipeline, and the p-value of our features are almost all very small, so we did not drop them after we dropped Brands and Months. We looked up the F-stats and T-stats in this table as well, and the results of them showed our features are good to use.

Then we use the data that we prepared by the pipeline to fit in our Linear Regression model. **The MAE is 5433.6358 and RMSE is 6474.6904**. As a big market and high sales data, these two results are acceptable. RMSE is greater than MAE because RMSE penalizes large gaps more harshly than MAE. But we also get a very high MSLE(mean squared log error) and low r_square results. A high MSLE might indicate that our model performs really bad with small sales. A low r_square means that the data may not fit our linear model. Overall, we thought linear regression may not be a good model for our data, and that’s why we don’t get good performance.

We also tried to implement PCA to reduce dimensions of our dataframe. But no matter what n_components we choose, we're not going to get anything close to the r_squrae value that we got before. Also the MAE, RMSE all increased after PCA. The reason might be that our features are **not correlated and our model is not linear.** As PCA is designed to tease out statistically independent feature vectors, a covariant matrix may not capture highest variances here.

We also employ Gradient Boosting Regressor as our ensemble method. GB regressor is used for training multiple high-bias and low-variance models and each model tries to improve on previous model's mistakes. We select our train and test set before implementing PCA, since the result is getting lower after PCA. And we set the parameter n_estimators=100, learning_rate=1, max_depth=1. n_estimators=100 is the default value of the number of boosting stages to perform, learning_rate=1 is the shrinking percentage and max_depth=1 is the maximum depth of each regression estimator. The result is much better than any other model we selected so far. **explained_variance: 0.9761, r2: 0.9761, MAE: 749.8474, RMSE: 1191.3052**. Both MAE and MSE are ideal and r^2 is nearly perfect which is unexpected initially. Since each feature is highly independent, Gradient Boosting might be a good fit for the model selection.

We merged the gridsearch and cross-validation steps together, we use 10-fold cross validation here. We used both Lasso and Ridge as the estimator. Lasso is a modification of linear regression, where the model is penalized for the sum of absolute values of the weights. Thus, the absolute values of weight will be (in general) reduced, and many will tend to be zeros. Ridge takes a step further and penalizes the model for the sum of squared value of the weights. Thus, the weights not only tend to have smaller absolute values, but also really tend to penalize the extremes of the weights, resulting in a group of weights that are more evenly distributed. In order to get the lowest MAE, our results showed that the best learning rate for both Lasso and Ridge are alpha=0.001. We also use the ensemble model as the estimator, the best learning rate we got is 0.1 and the lowest **MAE is 389.59**, which is **too perfect but weird.** We believe that the value of MAE is too small and too 'perfect', because by observing the total sales data, the expected MAE value should be around 10 thousand, and a little bit lower should be in the thousands. But our MAE came to 300. The basic idea of gradient boosting is to start with a weak learner and at each step add another weak learner until you reach an excellent one. Therefore, one assumption for the result is that the model has almost reached the excellent learner and training errors have been minimized to a very small number.

At the end, we also try to use MLP Regressor to predict the sales. The performance of it is better than the result of linear regression. We got **MSLE=0.4839, r_square=0.8533, MAE=2182.2451, and RMSE=2951.3272.** These results showed that the multi-layer perceptron model can be used for our data to predict sales, as the data fit the model, and the errors are acceptable for this large dataframe. But we got the warning that the optimization can not converge. Even if we tried to add more iterations or change the learning rate, they won't help with this problem. So the

possible solutions might be to add more and more iterations and try more different learning rates. But my laptop cannot run a very large number of iterations fast enough so I just leave this result here, but I think we can get an even better result by using MLP Regression.

The highest performing model is Gradient Boosting Regressor after we compared all results by different models.

5. Discussion:

As a result, we noticed that the data may not fit into the linear model because the results of linear regression are the worst. As we mentioned above, PCA is also not applicable in this dataset. The reason might be that our features are not correlated and the dataset is not linear. However, we can use other models to get better performance such as Gradient Boosting regression and Multi-layer regression.

Cookies can make more different products for customers to choose, this is an important feature to increase sales. Also if the product has good sales and units sold in the past 3 months, Cookies should still keep selling that product. Cookies can make more cannabis that can be used on cannabis devices because devices is the most correlated feature in category L2, also we noticed that if the product can be used on devices, it is also belongs to all accessories in category L1, so maybe Cookies can make this kind of product in the future.

For the next steps, we will add more features into the dataframe from branddetail.csv, but we are lacking in computing performance of our laptops so we could not at this time. Also we need more current data to analyse, the data we have now is a little bit outdated for some brands. For example, the brand 10x Infused only has data around 2018. At the end, we also want to try many more iterations in our Multi-layer regression model because it has not converged yet. If we have more time, we will also try to predict sales from the product aspect, then we can use more features from the product details.

6. Conclusion:

In this project, we started a data science project with unclean data for the first time. We preprocessed the given data at the very beginning. We merged the datasets with linked information and then found some time series features from these datasets. Then we added more features based on brand details and ran some statistics for feature selection. After we dropped some features we implemented a pipeline to process our data. Then we used a linear regression predictive model but we did not get a good result from it. Even if we tried to implement the

PCA, we found that it could not provide good results after we reduced the dimension. Then we used Gradient Boosting regression as an ensemble method and we got the best result from it. The next step we merged the gridsearch and 10-fold cross validation together for the models that we used, and gridsearch found the optimal parameter of our models with lowest MAE. At the end we implemented Multi-layer regression to predict sales and the performance was also better than the linear regression.

In the end, we thought it was a very special experience. It is our first time to deal with data under very vague rules, and the project was very challenging. We tried a lot of different data processing methods and a lot of different machine learning algorithms, and it helped us understand more about data science and the tasks that might come up at work in the future.