In [1]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold
from sklearn import model_selection
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
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 KFold, GridSearchCV
from sklearn.feature_selection import RFE
from sklearn import decomposition
from sklearn.decomposition import TruncatedSVD
from sklearn.svm import SVC
from sklearn.metrics import plot_confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

import os
import random
```

Prep

In [2]:
```
#read the datasets
avg_retail = pd.read_csv('data/BrandAverageRetailPrice.csv')
details = pd.read_csv('data/BrandDetails.csv')
total_sales = pd.read_csv('data/BrandTotalSales.csv')
total_units = pd.read_csv('data/BrandTotalUnits.csv')
customers = pd.read_csv('data/cust.csv')
data_dict = pd.read_csv('data/DataDictionary.csv')
top_50 = pd.read_csv('data/Top50ProductsbyTotalSales-TimeSeries.csv')
```

In [3]:
```
#set names for dataframes
avg_retail.name = 'Average Retail'
details.name = 'Brand Details'
total_sales.name = 'Brand Total Sales'
total_units.name = 'Brand_Total Units'
customers.name = 'Customers'
data_dict.name = 'Data Dictionary'
top_50.name = 'Top 50 Products by Total Sales'
```

In [4]:
```
#container to keep track of all datasets
my_data = [avg_retail, details, total_sales, total_units, customers, data_dict, top_50]
```

Look at Data

In [5]:
```
#look at info about dataframes using head and info functions
for df in my_data:
    print(df.name)
    print(df.head())
    print(df.info(), '\n')
```

```
Average Retail
         Brands   Months        ARP  vs. Prior Period
0  #BlackSeries  08/2020  15.684913               NaN
1  #BlackSeries  09/2020        NaN         -1.000000
2  #BlackSeries  01/2021  13.611428               NaN
3  #BlackSeries  02/2021  11.873182         -0.127705
4  #BlackSeries  03/2021        NaN         -1.000000
<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
```

```
None

Brand Details
        State   Channel  Category L1   Category L2            Category L3  \
0  California  Licensed  Inhaleables        Flower                 Hybrid
1  California  Licensed  Inhaleables        Flower                 Hybrid
2  California  Licensed  Inhaleables        Flower        Sativa Dominant
3  California  Licensed  Inhaleables        Flower        Sativa Dominant
4  California  Licensed  Inhaleables  Concentrates  Dabbable Concentrates

  Category L4 Category L5             Brand  \
0         NaN         NaN      #BlackSeries
1         NaN         NaN      #BlackSeries
2         NaN         NaN      #BlackSeries
3         NaN         NaN      #BlackSeries
4         Wax         NaN  101 Cannabis Co.

                             Product Description Total Sales ($)  ...  \
0  #BlackSeries - Vanilla Frosting - Flower (Gram)    1,103.964857  ...
1  #BlackSeries - Vanilla Frosting - Flower (Gram)      674.645211  ...
2  #BlackSeries - Blueberry Slushy - Flower (Gram)    2,473.699102  ...
3  #BlackSeries - Blueberry Slushy - Flower (Gram)   14,589.916417  ...
4             101 Cannabis Co. - Afghan Kush - Wax       145.39627  ...

  Total THC  Total CBD Contains CBD  Pax Filter            Strain Is Flavored  \
0         0          0     THC Only         NaN  Vanilla Frosting         NaN
1         0          0     THC Only         NaN  Vanilla Frosting         NaN
2         0          0     THC Only         NaN  Blueberry Slushy         NaN
3         0          0     THC Only         NaN  Blueberry Slushy         NaN
4         0          0     THC Only         NaN       Afghan Kush         NaN

         Mood Effect       Generic Vendor      Generic Items  \
0  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
1  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
2  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
3  Not Mood Specific  Non-Generic Vendors  Non-Generic Items
4  Not Mood Specific  Non-Generic Vendors  Non-Generic Items

  $5 Price Increment
0   $10.00 to $14.99
1   $15.00 to $19.99
2   $15.00 to $19.99
3   $10.00 to $14.99
4   $35.00 to $39.99

[5 rows x 25 columns]
<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
None

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

```
 1   Brand            25279 non-null  object
 2   Total Sales ($)  25279 non-null  object
dtypes: object(3)
memory usage: 592.6+ KB
None

Brand_Total Units
         Brands   Months          Total Units  vs. Prior Period
0  #BlackSeries  08/2020  1,616.3390040000000               NaN
1  #BlackSeries  09/2020                  NaN         -1.000000
2  #BlackSeries  01/2021    715.5328380000000               NaN
3  #BlackSeries  02/2021           766.669135          0.071466
4  #BlackSeries  03/2021                  NaN         -1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27686 entries, 0 to 27685
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Brands            27686 non-null  object
 1   Months            27686 non-null  object
 2   Total Units       25712 non-null  object
 3   vs. Prior Period  24935 non-null  float64
dtypes: float64(1), object(3)
memory usage: 865.3+ KB
None

0    Customers
1    Customers
2    Customers
Name: name, dtype: object
   customerid       name  years  spent  Unnamed: 4  Unnamed: 5  Unnamed: 6
0           1  Customers      2   2500         NaN         NaN         NaN
1           2  Customers      4   1300         NaN         NaN         NaN
2           3  Customers      5   2400         NaN         NaN         NaN
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   customerid  3 non-null      int64
 1   name        3 non-null      object
 2   years       3 non-null      int64
 3   spent       3 non-null      int64
 4   Unnamed: 4  0 non-null      float64
 5   Unnamed: 5  0 non-null      float64
 6   Unnamed: 6  0 non-null      float64
dtypes: float64(3), int64(3), object(1)
memory usage: 296.0+ bytes
None

Data Dictionary
  Feature Name                                        Description  Unnamed: 2  \
0        State  State where the sales occurred (for our datase...         NaN
1      Channel  Options between legal (licensed) and grey mark...         NaN
2  Category L1  Highest-level category for product. Options in...         NaN
3  Category L2  Next layer of categorization. Multiple product...         NaN
4  Category L3                    Further detail on product-type         NaN

   Unnamed: 3  Unnamed: 4  Unnamed: 5  Unnamed: 6  Unnamed: 7  Unnamed: 8  \
0         NaN         NaN         NaN         NaN         NaN         NaN
1         NaN         NaN         NaN         NaN         NaN         NaN
2         NaN         NaN         NaN         NaN         NaN         NaN
3         NaN         NaN         NaN         NaN         NaN         NaN
4         NaN         NaN         NaN         NaN         NaN         NaN

   Unnamed: 9  ...  Unnamed: 12  Unnamed: 13  Unnamed: 14  Unnamed: 15  \
0         NaN  ...          NaN          NaN          NaN          NaN
1         NaN  ...          NaN          NaN          NaN          NaN
2         NaN  ...          NaN          NaN          NaN          NaN
3         NaN  ...          NaN          NaN          NaN          NaN
4         NaN  ...          NaN          NaN          NaN          NaN

   Unnamed: 16  Unnamed: 17  Unnamed: 18  Unnamed: 19  Unnamed: 20  \
0          NaN          NaN          NaN          NaN          NaN
1          NaN          NaN          NaN          NaN          NaN
2          NaN          NaN          NaN          NaN          NaN
3          NaN          NaN          NaN          NaN          NaN
4          NaN          NaN          NaN          NaN          NaN

   Unnamed: 21
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN

[5 rows x 22 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 22 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Feature Name  25 non-null     object
```

```
 1   Description   25 non-null     object
 2   Unnamed: 2    0 non-null      float64
 3   Unnamed: 3    0 non-null      float64
 4   Unnamed: 4    0 non-null      float64
 5   Unnamed: 5    0 non-null      float64
 6   Unnamed: 6    0 non-null      float64
 7   Unnamed: 7    0 non-null      float64
 8   Unnamed: 8    0 non-null      float64
 9   Unnamed: 9    0 non-null      float64
 10  Unnamed: 10   0 non-null      float64
 11  Unnamed: 11   0 non-null      float64
 12  Unnamed: 12   0 non-null      float64
 13  Unnamed: 13   0 non-null      float64
 14  Unnamed: 14   0 non-null      float64
 15  Unnamed: 15   0 non-null      float64
 16  Unnamed: 16   0 non-null      float64
 17  Unnamed: 17   0 non-null      float64
 18  Unnamed: 18   0 non-null      float64
 19  Unnamed: 19   0 non-null      float64
 20  Unnamed: 20   0 non-null      float64
 21  Unnamed: 21   0 non-null      float64
dtypes: float64(20), object(2)
memory usage: 4.4+ KB
None

Top 50 Products by Total Sales
                                 Products   Months       Total Sales ($)
0  Flower - Strain Blends - Flower (Gram)  07/2020  22,738,489.622206017
1  Flower - Strain Blends - Flower (Gram)  03/2021   22,648,507.64839804
2  Flower - Strain Blends - Flower (Gram)  05/2021   22,338,755.88508607
3  Flower - Strain Blends - Flower (Gram)  09/2020  21,461,950.605336975
4  Flower - Strain Blends - Flower (Gram)  06/2021  21,347,569.064233065
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1671 entries, 0 to 1670
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Products         1671 non-null   object
 1   Months           1671 non-null   object
 2   Total Sales ($)  1671 non-null   object
dtypes: object(3)
memory usage: 39.3+ KB
None
```

Clean up some of the dataframes

Some of the dataframes have columns we cannot use with just unnamed and missing columns

Dataframes in question:

- data_dict
- customers

In [6]:
```
data_dict.head()
```

Out[6]:

	Feature Name	Description	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 12	Unnamed: 13
0	State	State where the sales occurred (for our datase...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
1	Channel	Options between legal (licensed) and grey mark...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
2	Category L1	Highest-level category for product. Options in...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
3	Category L2	Next layer of categorization. Multiple product...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
4	Category L3	Further detail on product-type	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

5 rows × 22 columns

In [7]:
```
customers.head()
```

Out[7]:

	customerid	name	years	spent	Unnamed: 4	Unnamed: 5	Unnamed: 6
0	1	Customers	2	2500	NaN	NaN	NaN
1	2	Customers	4	1300	NaN	NaN	NaN
2	3	Customers	5	2400	NaN	NaN	NaN

In [8]:
```
#trimming to only have useful columns
data_dict = data_dict[['Feature Name', 'Description']]
customers = customers[['customerid', 'name', 'years', 'spent']]
```

In [9]:
```
data_dict.head()
```

Out[9]:

	Feature Name	Description
0	State	State where the sales occurred (for our datase...
1	Channel	Options between legal (licensed) and grey mark...
2	Category L1	Highest-level category for product. Options in...
3	Category L2	Next layer of categorization. Multiple product...
4	Category L3	Further detail on product-type

In [10]:
```
customers.head()
```

Out[10]:

	customerid	name	years	spent
0	1	Customers	2	2500
1	2	Customers	4	1300
2	3	Customers	5	2400

Some dataframes need their information to be formatted in a way we can use it. Examples of how we can do this includes:

- converting months to date time
- changing numbers from string to float form

In [11]:
```
#cleaning avg_retail
avg_retail['Months'] = pd.to_datetime(avg_retail['Months'])

avg_retail.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
```

In [12]:
```
#clean total_sales
#convert months to date time
total_sales['Months'] = pd.to_datetime(total_sales['Months'])
#change sales to float
total_sales['Total Sales ($)'] = total_sales['Total Sales ($)'].str.replace(',', '').astype(float)
total_sales['Total Sales ($)'] = pd.to_numeric(total_sales['Total Sales ($)'])

total_sales.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
```

In [13]:
```
#clean total_units
#convert months to date time
total_units['Months'] = pd.to_datetime(total_units['Months'])
#change units to float
total_units['Total Units'] = total_units['Total Units'].str.replace(',', '').astype(float)
total_units['Total Units'] = pd.to_numeric(total_units['Total Units'])

total_sales.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
```

In [14]:
```
#clean up details
#change total sales to float
details['Total Sales ($)'] = details['Total Sales ($)'].str.replace(',', '').astype(float)
details['Total Sales ($)'] = pd.to_numeric(details['Total Sales ($)'])

#changes units to float
details['Total Units'] = details['Total Units'].str.replace(',', '').astype(float)
details['Total Units'] = pd.to_numeric(details['Total Units'])

#change total thc to float
details['Total THC'] = details['Total THC'].str.replace(',', '').astype(float)
details['Total THC'] = pd.to_numeric(details['Total THC'])

#change total cbd to float
details['Total CBD'] = details['Total CBD'].str.replace(',', '').astype(float)
details['Total CBD'] = pd.to_numeric(details['Total CBD'])
```

Time Series Feature Engineering, Merge Data, Develop New Features

- break up data by brand
- use data from other data frames to make new features
- merge with other brands for overall dataset

In [15]:
```
brands = total_units['Brands'].unique()
print(len(brands))
print(brands)
```

```
1640
['#BlackSeries' '101 Cannabis Co.' '10x Infused' ... 'Zlixir' 'Zoma'
 'Zuma Topicals']
```

In [16]:
```
details.head()
```

Out[16]:

	State	Channel	Category L1	Category L2	Category L3	Category L4	Category L5	Brand	Product Description	Total Sales ($)	...	Total THC	Total CBD	Conta C
0	California	Licensed	Inhaleables	Flower	Hybrid	NaN	NaN	#BlackSeries	#BlackSeries - Vanilla Frosting - Flower (Gram)	1103.964857	...	0.0	0.0	T C
1	California	Licensed	Inhaleables	Flower	Hybrid	NaN	NaN	#BlackSeries	#BlackSeries - Vanilla Frosting - Flower (Gram)	674.645211	...	0.0	0.0	T C
2	California	Licensed	Inhaleables	Flower	Sativa Dominant	NaN	NaN	#BlackSeries	#BlackSeries - Blueberry Slushy - Flower (Gram)	2473.699102	...	0.0	0.0	T C
3	California	Licensed	Inhaleables	Flower	Sativa Dominant	NaN	NaN	#BlackSeries	#BlackSeries - Blueberry Slushy - Flower (Gram)	14589.916417	...	0.0	0.0	T C
4	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates	Wax	NaN	101 Cannabis Co.	101 Cannabis Co. - Afghan Kush - Wax	145.396270	...	0.0	0.0	T C

5 rows × 25 columns

In [17]:
```
df = pd.DataFrame()
for brand in brands:
    units = total_units[total_units.Brands == str(brand)]
    units.loc[:,'Previous Month'] = units.loc[:,'Total Units'].shift(1)
    units.loc[:,'Rolling Average'] = (units.loc[:,'Total Units'].shift(1) + units.loc[:,'Total Units'].shift(2) + units.l

    sales = total_sales[total_sales.Brand == str(brand)]
    units = units.merge(sales, left_on='Months', right_on='Months')
    units = units.drop(['Brand'], 1)
```

```
    brand_details = details[details.Brand == str(brand)]

    #new features using brand details
    #number of products
    units['Num Products'] = len(brand_details)

    #number strains
    num_strain = len(brand_details['Strain'].unique())
    units['Num Strains'] = num_strain

    #num thc only products
    temp = brand_details['Contains CBD']
    counter = 0
    for i in temp:
        if i == 'THC Only':
            counter += 1
    units['Number THC Only Products'] = counter

    #percentage thc only products
    if len(brand_details) == 0:
        units['Percent THC Only'] = float('Nan')
    else:
        units['Percent THC Only'] = counter / len(brand_details)


    #add to dataframe with other brands
    df = df.append(units)
```

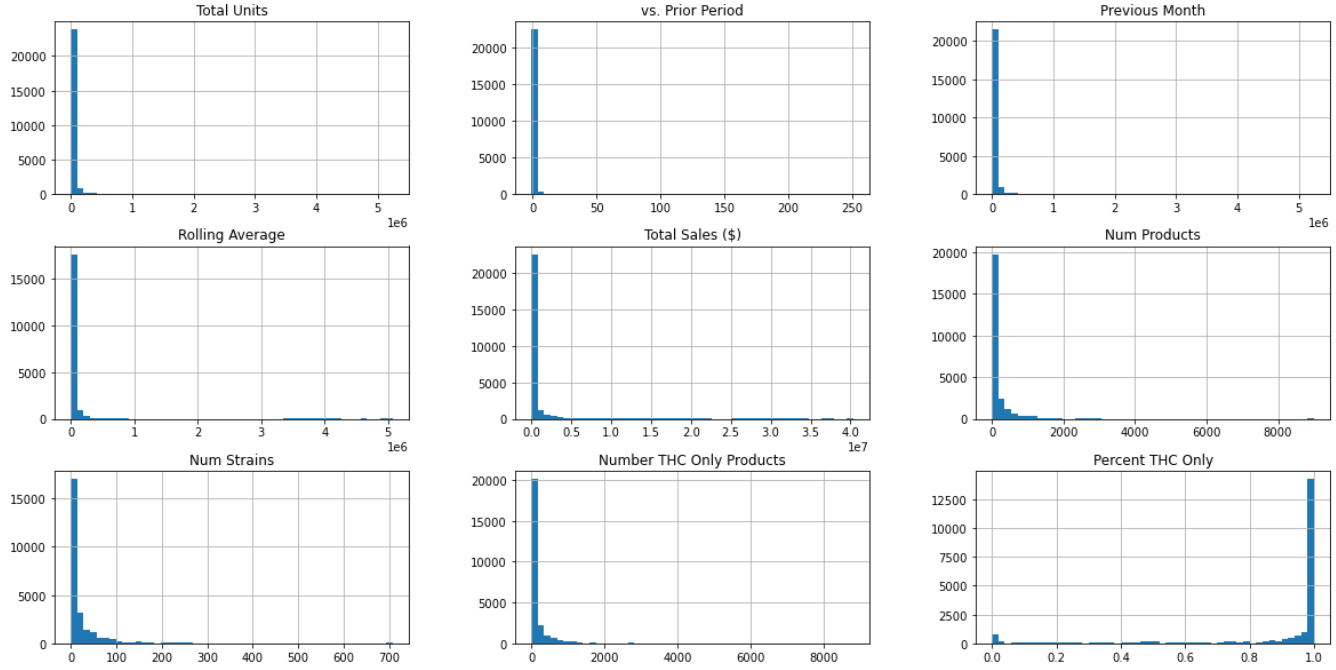
```
/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1596: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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] = _infer_fill_value(value)
/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1745: 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
  isetter(ilocs[0], value)
```

In [18]:
```
df.head(10)
```

Out[18]:

	Brands	Months	Total Units	vs. Prior Period	Previous Month	Rolling Average	Total Sales ($)	Num Products	Num Strains	Number THC Only Products	Percent THC Only
0	#BlackSeries	2020-08-01	1616.339004	NaN	NaN	NaN	25352.135918	4	2	4	1.0
1	#BlackSeries	2021-01-01	715.532838	NaN	NaN	NaN	9739.423400	4	2	4	1.0
2	#BlackSeries	2021-02-01	766.669135	0.071466	715.532838	NaN	9102.802187	4	2	4	1.0
0	101 Cannabis Co.	2019-11-01	131.067720	NaN	NaN	NaN	4465.040321	77	21	77	1.0
1	101 Cannabis Co.	2020-01-01	345.413448	NaN	NaN	NaN	11790.663567	77	21	77	1.0
2	101 Cannabis Co.	2020-02-01	696.658431	1.016883	345.413448	NaN	20266.761007	77	21	77	1.0
3	101 Cannabis Co.	2020-03-01	943.393328	0.354169	696.658431	NaN	30465.470533	77	21	77	1.0
4	101 Cannabis Co.	2020-04-01	712.498102	-0.244750	943.393328	661.821736	23465.657692	77	21	77	1.0
5	101 Cannabis Co.	2020-05-01	619.841032	-0.130045	712.498102	784.183287	21348.394472	77	21	77	1.0
6	101 Cannabis Co.	2020-06-01	426.150450	-0.312484	619.841032	758.577487	14111.757773	77	21	77	1.0

In [108…
```
#visualizing my data
import matplotlib.pyplot as plt
%matplotlib inline
df.hist(bins=50, figsize=(20, 10))
plt.show()
```



In [20]:
```
#prep data
x = df.drop('Total Sales ($)', axis = 1)
y = df['Total Sales ($)']
```

In [21]:
```
#split data
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2)
print('xtrain shape:', xtrain.shape)
print('xtest shape:', xtest.shape)
print('ytrain shape:', ytrain.shape)
print('ytest shape:', ytest.shape)
```

```
xtrain shape: (20223, 10)
xtest shape: (5056, 10)
ytrain shape: (20223,)
ytest shape: (5056,)
```

In [22]:
```
x.head()
```

Out[22]:

	Brands	Months	Total Units	vs. Prior Period	Previous Month	Rolling Average	Num Products	Num Strains	Number THC Only Products	Percent THC Only
0	#BlackSeries	2020-08-01	1616.339004	NaN	NaN	NaN	4	2	4	1.0
1	#BlackSeries	2021-01-01	715.532838	NaN	NaN	NaN	4	2	4	1.0
2	#BlackSeries	2021-02-01	766.669135	0.071466	715.532838	NaN	4	2	4	1.0
0	101 Cannabis Co.	2019-11-01	131.067720	NaN	NaN	NaN	77	21	77	1.0
1	101 Cannabis Co.	2020-01-01	345.413448	NaN	NaN	NaN	77	21	77	1.0

Categorical Features:

```
Months
Brands
```

Numerical Features:

```
All other featuers
```

Pipeline

In [23]:
```
#pipeline
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('std_scaler', StandardScaler()),
    ])
```

```
categorical_features = ['Brands', 'Months']
numerical_features = [feat for feat in x.columns if feat not in categorical_features]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])


features_train_transformed = full_pipeline.fit_transform(x)
xpiped = features_train_transformed
```

In [24]:
```
#new split using pipelined data
xtrain, xtest, ytrain, ytest = train_test_split(xpiped, y, test_size = 0.2)
print('xtrain shape:', xtrain.shape)
print('xtest shape:', xtest.shape)
print('ytrain shape:', ytrain.shape)
print('ytest shape:', ytest.shape)
```

```
xtrain shape: (20223, 1672)
xtest shape: (5056, 1672)
ytrain shape: (20223,)
ytest shape: (5056,)
```

In [25]:
```
#regression results summary
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))
```

Linear Regression

In [26]:
```
#implement linear regression
lr = LinearRegression()
lr.fit(xtrain, ytrain)
test_pred = lr.predict(xtest)
lr.score(xtest, ytest)
```

Out[26]: 0.9494238233137843

In [27]:
```
regression_results(ytest, test_pred)
```

```
explained_variance:  0.9494
r2:  0.9494
MAE:  93367.1372
MSE:  126174475777.5105
RMSE:  355210.4669
```

Ensemble Method Regression - Random Forest

In [28]:
```
#ensemble method - random forrest
rf = RandomForestRegressor()
rf.fit(xtrain, ytrain)
rf_ypred = rf.predict(xtest)
rf.score(xtest, ytest)
```

Out[28]: 0.9914594223997921

In [29]:
```
regression_results(ytest, rf_ypred)
```

```
explained_variance:  0.9915
r2:  0.9915
MAE:  45462.0019
MSE:  21306531496.6182
RMSE:  145967.57
```

Implement PCA

Data was too complex and took forever to run. Use PCA to make it more simple

In [30]:
```
#data is too complex make it simpler
'''
pca = decomposition.PCA(n_components=4)
```

```
x_pca = pca.fit_transform(xpiped)
'''
```

Out[30]:
```
'\npca = decomposition.PCA(n_components=4)\nx_pca = pca.fit_transform(xpiped)\n'
```

Note

Tried to run PCA to make data better but the PCA was not supported.

Given the following error:

```
PCA does not support sparse input. See TruncatedSVD for a possible alternative.
```

Will be running a truncatedSVD instead

In [75]:
```
#data is too complex, make it simpler
pca = TruncatedSVD(n_components=3)
x_pca = pca.fit_transform(xpiped)

xtrain, xtest, ytrain, ytest = train_test_split(x_pca, y, test_size = 0.2)
```

In [76]:
```
#redo linear regression
lr = LinearRegression()
lr.fit(xtrain, ytrain)
test_pred = lr.predict(xtest)

regression_results(ytest, test_pred)
```

```
explained_variance:  0.8831
r2:  0.8831
MAE:  199394.7118
MSE:  328627847603.2556
RMSE:  573260.7152
```

In [77]:
```
#redo ensemble method - random forrest
rf = RandomForestRegressor()
rf.fit(xtrain, ytrain)
rf_ypred = rf.predict(xtest)
regression_results(ytest, rf_ypred)
```

```
explained_variance:  0.9654
r2:  0.9654
MAE:  83422.8955
MSE:  97322593412.6802
RMSE:  311965.6927
```

Cross Validate with KFolds and GridSearchCV

In [78]:
```
#crossvalidation model paramters
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

#range of paramters
hyper_params = [{'n_features_to_select': (1,2,3,4,5)}]
```

In [79]:
```
#grid search for linear regression
lr = LinearRegression()
lr.fit(xtrain, ytrain)
rfe = RFE(lr)

crossval_model = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)
crossval_model.fit(xtrain, ytrain)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:    0.1s finished
```

Out[79]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=RFE(estimator=LinearRegression()),
             param_grid=[{'n_features_to_select': (1, 2, 3, 4, 5)}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [80]:
```
crossval_results = pd.DataFrame(crossval_model.cv_results_)
crossval_results
```

Out[80]:

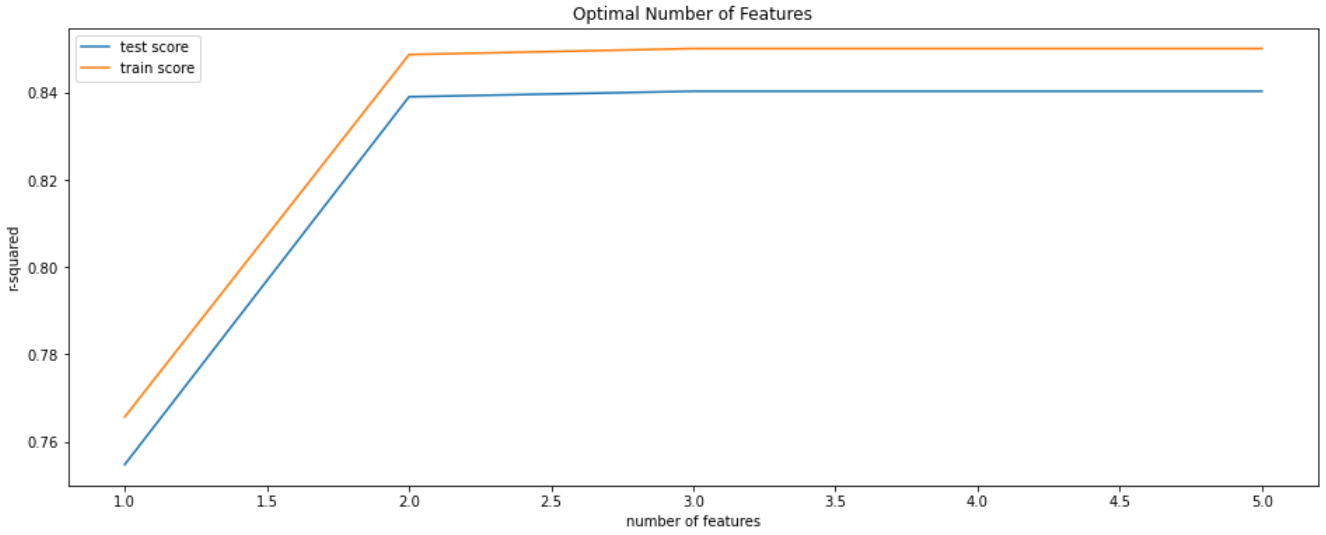
mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_test_score	split1_test_scor

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_test_score	split1_test_scor
0	0.003794	0.001077	0.000716	0.000378	1	{'n_features_to_select': 1}	0.808398	0.68029
1	0.002652	0.000128	0.000525	0.000040	2	{'n_features_to_select': 2}	0.895846	0.79361
2	0.001816	0.000113	0.000533	0.000038	3	{'n_features_to_select': 3}	0.897117	0.79402
3	0.001889	0.000080	0.000593	0.000054	4	{'n_features_to_select': 4}	0.897117	0.79402
4	0.001743	0.000045	0.000545	0.000065	5	{'n_features_to_select': 5}	0.897117	0.79402

5 rows × 21 columns

In [81]:
```
# plotting crossval results
plt.figure(figsize=(16,6))

plt.plot(crossval_results["param_n_features_to_select"], crossval_results["mean_test_score"])
plt.plot(crossval_results["param_n_features_to_select"], crossval_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'])
```

Out[81]:
```
<matplotlib.legend.Legend at 0x7ff83ba63c40>
```



In [82]:
```
#grid search for random forrest
rf = RandomForestRegressor()
rf.fit(xtrain, ytrain)
rfe = RFE(rf)

crossval_model = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)
crossval_model.fit(xtrain, ytrain)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:  2.7min finished
```

Out[82]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=RFE(estimator=RandomForestRegressor()),
             param_grid=[{'n_features_to_select': (1, 2, 3, 4, 5)}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [83]:
```
crossval_results = pd.DataFrame(crossval_model.cv_results_)
crossval_results
```

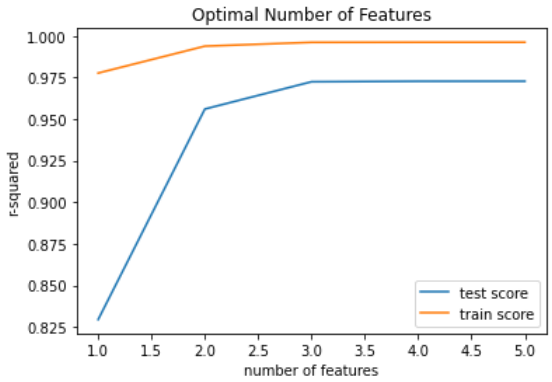
Out[83]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_test_score	split1_test_scor
0	9.734009	0.318775	0.109537	0.003124	1	{'n_features_to_select': 1}	0.885160	0.81209
1	7.553310	0.101259	0.098317	0.000916	2	{'n_features_to_select': 2}	0.974348	0.94061

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_test_score	split1_test_scor
2	4.382365	0.035150	0.095261	0.002205	3	{'n_features_to_select': 3}	0.981224	0.96663
3	4.379332	0.047040	0.095048	0.002970	4	{'n_features_to_select': 4}	0.982154	0.96575
4	4.429725	0.068131	0.093585	0.000746	5	{'n_features_to_select': 5}	0.981464	0.96635

5 rows × 21 columns

In [84]:
```
# plotting crossval results
plt.figure(figsize=(16,6))
%matplotlib inline
plt.plot(crossval_results["param_n_features_to_select"], crossval_results["mean_test_score"])
plt.plot(crossval_results["param_n_features_to_select"], crossval_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'])
plt.show()
```



Bagging Regressor

method of choice

In [85]:
```
bag = BaggingRegressor()
bag.fit(xtrain,ytrain)
bag_ypred = bag.predict(xtest)
regression_results(ytest, bag_ypred)
```

```
explained_variance:  0.9655
r2:  0.9655
MAE:  86984.4296
MSE:  97131452431.2056
RMSE:  311659.1928
```

In [90]:
```
#grid search cross val for bag
rfe = RFE(bag)

crossval_model = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)
crossval_model.fit(xtrain, ytrain)
crossval_results = pd.DataFrame(crossval_model.cv_results_)
crossval_results
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
```

```
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes
```

```
  warnings.warn("Estimator fit failed. The score on this train-test"
/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:548: FitFailedWarning: Estimator fit fa
iled. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 151, in fit
    return self._fit(X, y)
  File "/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_rfe.py", line 204, in _fit
    raise RuntimeError('The classifier does not expose '
RuntimeError: The classifier does not expose "coef_" or "feature_importances_" attributes

  warnings.warn("Estimator fit failed. The score on this train-test"
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:   13.0s finished
```

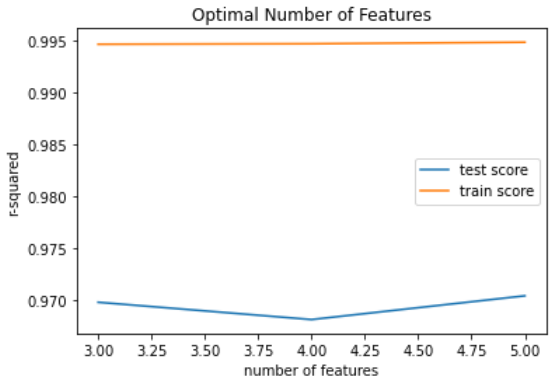
Out[90]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_test_score	split1_test_scor
0	0.472460	0.013622	0.000000	0.000000	1	{'n_features_to_select': 1}	NaN	Nal
1	0.473862	0.012873	0.000000	0.000000	2	{'n_features_to_select': 2}	NaN	Nal
2	0.475292	0.014106	0.012247	0.001304	3	{'n_features_to_select': 3}	0.978987	0.96442
3	0.487780	0.031540	0.011847	0.001333	4	{'n_features_to_select': 4}	0.978513	0.96109
4	0.534654	0.070352	0.012208	0.000914	5	{'n_features_to_select': 5}	0.979746	0.96588

5 rows × 21 columns

In [91]:
```
# plotting crossval results
plt.figure(figsize=(16,6))
%matplotlib inline
plt.plot(crossval_results["param_n_features_to_select"], crossval_results["mean_test_score"])
plt.plot(crossval_results["param_n_features_to_select"], crossval_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'])
```

Out[91]:
```
<matplotlib.legend.Legend at 0x7ff834afc250>
```



Report

Introduction

With my report I am aiming to model/predict the total sales of marijuana based on a variety of factors. From my background research, I learned that marijuana use seems to be steadily increasing as time goes on. The majority of marijuana use is located in the US with medical and recreational adult use being the largest streams of usage according to the BDSA. This may be due to the US legalizing the use of recreational marijuana where as many other countries have stricter laws. With my knowledge of data science modeling, I will try and predict the total sales based information given from brands to see how each brand's growth is expected in the future. It is important to know the increase in sales as marijuana becomes a bigger part of our economy.

Methodology

With my data I first had look at all the data and make sure they were in a form where I could use them. I had to get rid of columns that were not contributing any information as they would only take up unnecessary storage. I shaved off columns from customers and data dictionary. I then went through the rest of the dataframes and made sure to translate the data into a form where they could be used for any operations in mind. For

example, the month column was in a string and I reformatted it to display as a date time. I also would go and change any numbers from a string to a float.

With my data I decided to work mostly with the datasets concerning the brand. The information given by the brands allow me the most information. For example, the customers dataset only had 3 entries and the information was not enough for me to use. Similarly data dictionary had a very small set of information that I decided would not be as helpful as my other datasets. The Top 50 Products by Total Sales had a greater amount of information but it did not allow me the flexibility that the other data sets allowed. I ended up working primarily with the brand details, total units, and total sales.

In order to separate the data and see how they all connected, I went through and created new features based on the brand of each product. I was able to combine information from the Total Sales and Total Units dataframes to be able to see how the products interacted with each other based on what time the sale was out. I also created features such as the number of products a brand had, the number of strains the brand holds, the number of THC only products and the percentage of THC only products that the brand sold. These new features were created by combining the information we had from the brand details page with the information we had from the units and sales dataframes. Something to note here was that the percentage of THC only products was created with two columns we had already created and may overlap in terms of information.

As I had used a pipeline to make my data more expressive, running my data through the regressors was really slow. I decided to implement a principal analysis component (PCA) so that the data could run smoothly. The PCA made the data less dimensional and vastly decreased the runtime of my regressors as well as my cross validation executions possible (as they over an hour to run if I did not simplify my data with a pca). I decided to use a parameter of 3 components for my truncatedSVD as it allowed my data to be more complex than just 2 components but did not take an overwhelmingly long time to run.

Some of the classifiers I trained my model were Linear Regression, Random Forest and Bagging regressors. The linear regressor is a single regressor and only shows one model where the Random Forest and Bagging regressors were ensemble methods. Ensemble methods are used as they show more depth in data as they combine multiple models. I chose these two ensemble methods as they were intuitive and easy to implement. The random forrest was my choice for ensemble method as it a combination of decision trees and easy to understand. I also used the bagging regressor for my method of choice as it would allow for in-depth data compared to other single regressor classifiers.

Results

Data Shapes before Pipeline

```
xtrain shape: (20223, 10)
xtest shape: (5056, 10)
ytrain shape: (20223,)
ytest shape: (5056,)
```

Data Shapes after Pipeline

```
xtrain shape: (20223, 1672)
xtest shape: (5056, 1672)
ytrain shape: (20223,)
ytest shape: (5056,)
```

Regression Summary before PCA

Linear:

```
explained_variance:  0.9494
r2:  0.9494
MAE:  93367.1372
MSE:  126174475777.5105
RMSE:  355210.4669
```

Random Forrest:

```
explained_variance:  0.9915
r2:  0.9915
MAE:  45462.0019
MSE:  21306531496.6182
RMSE:  145967.57
```

Regression Summary after PCA

Linear:

```
explained_variance:  0.8831
r2:  0.8831
MAE:  199394.7118
MSE:  328627847603.2556
RMSE:  573260.7152
```

Random Forrest:

```
explained_variance:  0.9654
r2:  0.9654
MAE:  83422.8955
MSE:  97322593412.6802
RMSE:  311965.6927
```

Bagging:

```
explained_variance:  0.9655
r2:  0.9655
MAE:  86984.4296
MSE:  97131452431.2056
RMSE:  311659.1928
```

Optimal Features Minimum

```
Linear: 2
Random Forest: 3
Bagging: 4
```

Discussion:

Overall, we see a lot of high R^2 values which indicates the models are performing well. We do see a slight drop in R^2 values from before to after including the PCA. This is due to the PCA limiting the amount of components we can have. Due to the limited amount of componenents that are used the data is less expressive and then will be less acurrate as we train and test our models.

The complexity decrease in the training and testing sets comes from the inclusion of the PCA. This really helped the data be processed in a more effcient manner. It is also possible that there may be overfitting if we did not used the PCA as it would be too complex and specific to this set of data.

From the cross validation with kfolds and grid search, we see that the optimal amount of features around 2-4. This may be due the PCA components being 3 but this also shows that we do not need too many features to have a good model.

In [ ]: