CSM148 Project 3

BRENDAN ROSSMANGO, ADITYA MISHRA

TOTAL POINTS

120 / 120

QUESTION 1

Coding Requirements 60 pts

1.1 Merge Datasets and Link Info 5 / 5

✓ **- 0 pts** Correct

- 5 pts missing

1.2 Develop basic time series feature extraction plan 5 / 5

✓ **- 0 pts** Correct

1.3 Run Statistics 5 / 5

✓ **- 0 pts** Correct

1.4 Feature extraction and pipeline 5 / 5

✓ **- 0 pts** Correct

1.5 Pipeline 5 / 5

✓ **- 0 pts** Correct

1.6 Linear Regression 5 / 5

✓ **- 0 pts** Correct

1.7 PCA 5 / 5

✓ **- 0 pts** Correct

1.8 Ensemble Method 5 / 5

✓ **- 0 pts** Correct

1.9 Cross-Validate results 5 / 5

✓ **- 0 pts** Correct

1.10 GridSearch 5 / 5

✓ **- 0 pts** Correct

- 5 pts missing

1.11 Experiment with Custom Models 10 / 10

✓ **- 0 pts** Correct

- 10 pts missing

QUESTION 2

Report 60 pts

2.1 Executive Summary 5 / 5

✓ **- 0 pts** Correct

- 2.5 pts not enough explanation

- 5 pts missing

2.2 Background/Info 10 / 10

✓ **- 0 pts** Correct

- 5 pts not enough detailed information

- 10 pts missing

2.3 Methodology 15 / 15

✓ **- 0 pts** Correct

- 5 pts not enough explanation

- 15 pts missing

2.4 Results 15 / 15

✓ **- 0 pts** Correct

- 5 pts no detailed report

- 15 pts missing

2.5 Discussion 10 / 10

✓ **- 0 pts** Correct

- 3 pts Click here to replace this description.

- 10 pts missing

2.6 Conclusion 5 / 5

✓ **- 0 pts** Correct

- 2 pts Click here to replace this description.

- 5 pts missing

PROJECT 3

In [ ]:
```
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, cross_val_score, GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import sklearn.metrics.cluster as smc
from matplotlib import pyplot

import os
import itertools
import random

%matplotlib inline

random.seed(148)
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
```

In [ ]:
```
DATA_PATH = '/content/drive/MyDrive/Project3CSm148/data/'
arp = pd.read_csv(DATA_PATH+'BrandAverageRetailPrice.csv')
brands = pd.read_csv(DATA_PATH+'BrandDetails.csv')
sales = pd.read_csv(DATA_PATH+'BrandTotalSales.csv')
units = pd.read_csv(DATA_PATH+'BrandTotalUnits.csv')
products = pd.read_csv(DATA_PATH+'Top50ProductsbyTotalSales-Timeseries.csv')
```

In [ ]:
```
sales.info()
sales.head()
```

```
<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
```

Out[ ]:

	Months	Brand	Total Sales ($)
0	09/2018	10x Infused	1,711.334232
1	09/2018	1964 Supply Co.	25,475.21594500000
2	09/2018	3 Bros Grow	120,153.644757
3	09/2018	3 Leaf	6,063.5297850000000
4	09/2018	350 Fire	631,510.0481550000

In [ ]:
```
units.info()
units.head()
```

```
<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
```

Out[ ]:

	Brands	Months	Total Units	vs. Prior Period
0	#BlackSeries	08/2020	1,616.3390040000000	NaN
1	#BlackSeries	09/2020	NaN	-1.000000
2	#BlackSeries	01/2021	715.5328380000000	NaN
3	#BlackSeries	02/2021	766.669135	0.071466
4	#BlackSeries	03/2021	NaN	-1.000000

In [ ]:
```
# TURN THE MONTHS OBJECT TO A DATETIME SO WE CAN EASILY EXTRACT THE MONTH AND YEAR
units['Months'] = pd.to_datetime(units['Months'])
sales['Months'] = pd.to_datetime(sales['Months'])
arp['Months'] = pd.to_datetime(arp['Months'])
```

In [ ]:
```
# TURN THE SALES AND UNITS INTO FLOATS
sales['Total Sales ($)'] = sales['Total Sales ($)'].str.replace(",", "")
sales['Total Sales ($)'] = sales['Total Sales ($)'].str[:8]
sales['Total Sales ($)'] = pd.to_numeric(sales["Total Sales ($)"])
units['Total Units'] = units['Total Units'].str.replace(",","")
units['Total Units'] = units['Total Units'].str[:8]
units['Total Units'] = pd.to_numeric(units["Total Units"])
```

In [ ]:
```
sales.info()
sales.head()
```

```
<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
```

Out[ ]:

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

In [ ]:

```
units.info()
units.head()
```

```
<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
```

Out[ ]:

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

In [ ]:

```
units["Total Units"] = units["Total Units"].fillna(0)
units
```

Out[ ]:

	Brands	Months	Total Units	vs. Prior Period
0	#BlackSeries	2020-08-01	1616.3390	NaN
1	#BlackSeries	2020-09-01	0.0000	-1.000000
2	#BlackSeries	2021-01-01	715.5328	NaN
3	#BlackSeries	2021-02-01	766.6691	0.071466

	Brands	Months	Total Units	vs. Prior Period
4	#BlackSeries	2021-03-01	0.0000	-1.000000
...	...	...	...	...
27681	Zuma Topicals	2019-08-01	312.5153	NaN
27682	Zuma Topicals	2019-09-01	464.3063	0.485707
27683	Zuma Topicals	2019-10-01	348.0579	-0.250370
27684	Zuma Topicals	2019-11-01	135.9220	-0.609484
27685	Zuma Topicals	2019-12-01	0.0000	-1.000000

27686 rows × 4 columns

In [ ]:
```
# RENAME THE PRIOR PERIOD COLUMNS SINCE THEY ARE DIFFERENT
units["Units vs. Prior Period"] = units["vs. Prior Period"]
units = units.drop(columns=['vs. Prior Period'])
arp["ARP vs. Prior Period"] = arp["vs. Prior Period"]
arp = arp.drop(columns=['vs. Prior Period'])
arp
```

Out[ ]:

	Brands	Months	ARP	ARP vs. Prior Period
0	#BlackSeries	2020-08-01	15.684913	NaN
1	#BlackSeries	2020-09-01	NaN	-1.000000
2	#BlackSeries	2021-01-01	13.611428	NaN
3	#BlackSeries	2021-02-01	11.873182	-0.127705
4	#BlackSeries	2021-03-01	NaN	-1.000000
...	...	...	...	...
27206	Zuma Topicals	2019-08-01	31.598214	NaN
27207	Zuma Topicals	2019-09-01	37.860964	0.198199
27208	Zuma Topicals	2019-10-01	34.546154	-0.087552
27209	Zuma Topicals	2019-11-01	36.850000	0.066689
27210	Zuma Topicals	2019-12-01	NaN	-1.000000

27211 rows × 4 columns

In [ ]:
```
all_brands = units["Brands"].unique()
all_brands
```

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

In [ ]:
```
data = pd.DataFrame()
for brand in all_brands:
  units_data = units[units.Brands == brand]
```

```
# PREVIOUS MONTH: THE TOTAL UNITS OF THE PREVIOUS MONTH
units_data.loc[:,'Previous Month'] = units_data.loc[:,'Total Units'].shift(1)
# ROLLING AVERAGE: THE AVERAGE TOTAL UNITS OF THE PAST 3 MONTHS
units_data.loc[:,'Rolling Average'] = (units_data.loc[:,'Total Units'].shift(3) + \
                                       units_data.loc[:,'Total Units'].shift(2) + uni

first = units_data.first_valid_index()
# IF THERE ARE NO, ONE, OR TWO PRIOR MONTHS, MANUALLY CALCULATE THE ROLLING AVERAGE
units_data.loc[first,['Rolling Average']] = 0
if units_data.shape[0] > 2:
  units_data.loc[first+2,'Rolling Average'] = (units_data.loc[first+1,'Total Units']
                                               units_data.loc[first, "Total Units"])
if units_data.shape[0] > 1:
  units_data.loc[first+1,'Rolling Average'] = units_data.loc[first, "Total Units"]

# YEAR: GET THE YEAR FROM THE MONTHS COLUMN
units_data["Month"] = pd.DatetimeIndex(units_data["Months"]).month
units_data['Year'] = pd.DatetimeIndex(units_data['Months']).year
# SEASON: WINTER IS 1, SPRING IS 2, SUMMER IS 3, FALL IS 4
units_data["Season"] = pd.DatetimeIndex(units_data['Months']).month%12 // 3 + 1

# MERGE WITH TOTAL SALES AND ARP
sales_data = sales[sales.Brand == brand]
units_data = units_data.merge(sales_data, left_on='Months', right_on='Months', how ="
units_data = units_data.drop(['Brand'], 1)

arp_data = arp[arp.Brands == brand]
units_data = units_data.merge(arp_data, left_on='Months', right_on='Months', how ="ou
units_data = units_data.drop(['Brands_y', "Months"], 1)
units_data = units_data.rename(columns={"Brands_x": "Brands"})

units_details = brands[brands.Brand == brand]

# INHALEABLE: IF THE BRAND HAS INHALEABLE PRODUCTS
# EDIBLE: LIKEWISE BUT FOR EDIBLE PRODUCTS
i = 0
e = 0
if 'Inhaleables' in units_details['Category L1'].values:
    i = 1
if 'Edibles' in units_details['Category L1'].values:
    e = 1
units_data['Inhaleable'] = i
units_data['Edible'] = e

# PRODUCT COUNT (NUMBER OF PRODUCTS IN BRAND DETAILS TABLE)
units_data['Product Count'] = (units_details.Brand == brand).count()

# MOOD SPECIFIC: IF ANY PRODUCT IS MOOD SPECIFIC
mood_specific = 0
if "Mood Specific" in units_details["Mood Effect"].values:
  mood_specific = 1
units_data["Mood Specific"] = mood_specific

# FLAVORED: IF ANY PRODUCT IS FLAVORED
flavored = 0
if "Flavored" in units_details["Is Flavored"].values:
  flavored = 1
units_data["Flavored"] = flavored

# PRODUCT TYPE: MOST COMMON TYPE (CONCENTRATES, FLOWER, PRE-ROLLED, ETC.)
```

```
  most_common_type = "None"
  lst = list(units_details["Category L2"].values)
  if lst:
    most_common_type = max(set(lst), key=lst.count)
  units_data["Predominant Product Type"] = most_common_type

  # FLAVOR: MOST COMMON FLAVOR
  # NUMBER OF FLAVORS: ALL FLAVORS OF BRAND EXCEPT NONE FLAVOR
  most_common_flavor = "None"
  num_of_flavors = 0
  lst = list(units_details["Flavor"].values)
  if lst:
    most_common_flavor = max(set(lst), key=lst.count)
    flavors = list(filter(pd.notna, lst))
    num_of_flavors = len(set(flavors))
  units_data["Predominant Flavor"] = most_common_flavor
  units_data["Predominant Flavor"] = units_data["Predominant Flavor"].fillna("None")
  units_data["Number of Flavors"] = num_of_flavors

  data = data.append(units_data, ignore_index=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1596: SettingWithCopyWarn
ing:
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)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1743: SettingWithCopyWarn
ing:
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)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1763: SettingWithCopyWarn
ing:
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(loc, value)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: 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
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: 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
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: 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
```

In [ ]:
```
data
```

Out[ ]:

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	Total Sales ($)	
0	#BlackSeries	1616.3390	NaN	NaN	0.000000	8	2020	3	25352.130	15.
1	#BlackSeries	0.0000	-1.000000	1616.3390	1616.339000	9	2020	4	NaN	
2	#BlackSeries	715.5328	NaN	0.0000	808.169500	1	2021	1	9739.423	13.
3	#BlackSeries	766.6691	0.071466	715.5328	777.290600	2	2021	1	9102.802	11.
4	#BlackSeries	0.0000	-1.000000	766.6691	494.067300	3	2021	2	NaN	
...	...	...	...	...	...	...	...	...	...	
27681	Zuma Topicals	312.5153	NaN	NaN	0.000000	8	2019	3	9874.926	31.
27682	Zuma Topicals	464.3063	0.485707	312.5153	312.515300	9	2019	4	17579.080	37.
27683	Zuma Topicals	348.0579	-0.250370	464.3063	388.410800	10	2019	4	12024.060	34.
27684	Zuma Topicals	135.9220	-0.609484	348.0579	374.959833	11	2019	4	5008.728	36.
27685	Zuma Topicals	0.0000	-1.000000	135.9220	316.095400	12	2019	1	NaN	

27686 rows × 19 columns

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27686 entries, 0 to 27685
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Brands                  27686 non-null  object
 1   Total Units             27686 non-null  float64
 2   Units vs. Prior Period  24935 non-null  float64
 3   Previous Month          26046 non-null  float64
 4   Rolling Average         27686 non-null  float64
 5   Month                   27686 non-null  int64
 6   Year                    27686 non-null  int64
 7   Season                  27686 non-null  int64
 8   Total Sales ($)         25279 non-null  float64
 9   ARP                     25279 non-null  float64
```

1.1 Merge Datasets and Link Info **5 / 5**

✓ **- 0 pts** **Correct**

- 5 pts missing

	Brands	Months	Total Units	vs. Prior Period
4	#BlackSeries	2021-03-01	0.0000	-1.000000
...	...	...	...	...
27681	Zuma Topicals	2019-08-01	312.5153	NaN
27682	Zuma Topicals	2019-09-01	464.3063	0.485707
27683	Zuma Topicals	2019-10-01	348.0579	-0.250370
27684	Zuma Topicals	2019-11-01	135.9220	-0.609484
27685	Zuma Topicals	2019-12-01	0.0000	-1.000000

27686 rows × 4 columns

In [ ]:
```
# RENAME THE PRIOR PERIOD COLUMNS SINCE THEY ARE DIFFERENT
units["Units vs. Prior Period"] = units["vs. Prior Period"]
units = units.drop(columns=['vs. Prior Period'])
arp["ARP vs. Prior Period"] = arp["vs. Prior Period"]
arp = arp.drop(columns=['vs. Prior Period'])
arp
```

Out[ ]:

	Brands	Months	ARP	ARP vs. Prior Period
0	#BlackSeries	2020-08-01	15.684913	NaN
1	#BlackSeries	2020-09-01	NaN	-1.000000
2	#BlackSeries	2021-01-01	13.611428	NaN
3	#BlackSeries	2021-02-01	11.873182	-0.127705
4	#BlackSeries	2021-03-01	NaN	-1.000000
...	...	...	...	...
27206	Zuma Topicals	2019-08-01	31.598214	NaN
27207	Zuma Topicals	2019-09-01	37.860964	0.198199
27208	Zuma Topicals	2019-10-01	34.546154	-0.087552
27209	Zuma Topicals	2019-11-01	36.850000	0.066689
27210	Zuma Topicals	2019-12-01	NaN	-1.000000

27211 rows × 4 columns

In [ ]:
```
all_brands = units["Brands"].unique()
all_brands
```

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

In [ ]:
```
data = pd.DataFrame()
for brand in all_brands:
  units_data = units[units.Brands == brand]
```

```
# PREVIOUS MONTH: THE TOTAL UNITS OF THE PREVIOUS MONTH
units_data.loc[:,'Previous Month'] = units_data.loc[:,'Total Units'].shift(1)
# ROLLING AVERAGE: THE AVERAGE TOTAL UNITS OF THE PAST 3 MONTHS
units_data.loc[:,'Rolling Average'] = (units_data.loc[:,'Total Units'].shift(3) + \
                                       units_data.loc[:,'Total Units'].shift(2) + uni

first = units_data.first_valid_index()
# IF THERE ARE NO, ONE, OR TWO PRIOR MONTHS, MANUALLY CALCULATE THE ROLLING AVERAGE
units_data.loc[first,['Rolling Average']] = 0
if units_data.shape[0] > 2:
  units_data.loc[first+2,'Rolling Average'] = (units_data.loc[first+1,'Total Units']
                                               units_data.loc[first, "Total Units"])
if units_data.shape[0] > 1:
  units_data.loc[first+1,'Rolling Average'] = units_data.loc[first, "Total Units"]

# YEAR: GET THE YEAR FROM THE MONTHS COLUMN
units_data["Month"] = pd.DatetimeIndex(units_data["Months"]).month
units_data['Year'] = pd.DatetimeIndex(units_data['Months']).year
# SEASON: WINTER IS 1, SPRING IS 2, SUMMER IS 3, FALL IS 4
units_data["Season"] = pd.DatetimeIndex(units_data['Months']).month%12 // 3 + 1

# MERGE WITH TOTAL SALES AND ARP
sales_data = sales[sales.Brand == brand]
units_data = units_data.merge(sales_data, left_on='Months', right_on='Months', how ="
units_data = units_data.drop(['Brand'], 1)

arp_data = arp[arp.Brands == brand]
units_data = units_data.merge(arp_data, left_on='Months', right_on='Months', how ="ou
units_data = units_data.drop(['Brands_y', "Months"], 1)
units_data = units_data.rename(columns={"Brands_x": "Brands"})

units_details = brands[brands.Brand == brand]

# INHALEABLE: IF THE BRAND HAS INHALEABLE PRODUCTS
# EDIBLE: LIKEWISE BUT FOR EDIBLE PRODUCTS
i = 0
e = 0
if 'Inhaleables' in units_details['Category L1'].values:
    i = 1
if 'Edibles' in units_details['Category L1'].values:
    e = 1
units_data['Inhaleable'] = i
units_data['Edible'] = e

# PRODUCT COUNT (NUMBER OF PRODUCTS IN BRAND DETAILS TABLE)
units_data['Product Count'] = (units_details.Brand == brand).count()

# MOOD SPECIFIC: IF ANY PRODUCT IS MOOD SPECIFIC
mood_specific = 0
if "Mood Specific" in units_details["Mood Effect"].values:
  mood_specific = 1
units_data["Mood Specific"] = mood_specific

# FLAVORED: IF ANY PRODUCT IS FLAVORED
flavored = 0
if "Flavored" in units_details["Is Flavored"].values:
  flavored = 1
units_data["Flavored"] = flavored

# PRODUCT TYPE: MOST COMMON TYPE (CONCENTRATES, FLOWER, PRE-ROLLED, ETC.)
```

```
  most_common_type = "None"
  lst = list(units_details["Category L2"].values)
  if lst:
    most_common_type = max(set(lst), key=lst.count)
  units_data["Predominant Product Type"] = most_common_type

  # FLAVOR: MOST COMMON FLAVOR
  # NUMBER OF FLAVORS: ALL FLAVORS OF BRAND EXCEPT NONE FLAVOR
  most_common_flavor = "None"
  num_of_flavors = 0
  lst = list(units_details["Flavor"].values)
  if lst:
    most_common_flavor = max(set(lst), key=lst.count)
    flavors = list(filter(pd.notna, lst))
    num_of_flavors = len(set(flavors))
  units_data["Predominant Flavor"] = most_common_flavor
  units_data["Predominant Flavor"] = units_data["Predominant Flavor"].fillna("None")
  units_data["Number of Flavors"] = num_of_flavors

  data = data.append(units_data, ignore_index=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1596: SettingWithCopyWarn
ing:
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)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1743: SettingWithCopyWarn
ing:
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)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1763: SettingWithCopyWarn
ing:
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(loc, value)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: 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
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: 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
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

1.2 Develop basic time series feature extraction plan **5 / 5**

✓ **- 0 pts** **Correct**

```
 10  ARP vs. Prior Period      24499 non-null  float64
 11  Inhaleable                27686 non-null  int64
 12  Edible                    27686 non-null  int64
 13  Product Count             27686 non-null  int64
 14  Mood Specific             27686 non-null  int64
 15  Flavored                  27686 non-null  int64
 16  Predominant Product Type  27686 non-null  object
 17  Predominant Flavor        27686 non-null  object
 18  Number of Flavors         27686 non-null  int64
dtypes: float64(7), int64(9), object(3)
memory usage: 4.0+ MB
```

In [ ]:

```
# IMPUTING TIME
# VS. PRIOR PERIOD is NULL if it is the first month of the brand being sold or if 0 uni
#     SO, we can just set this value to 1
data["Units vs. Prior Period"] = data["Units vs. Prior Period"].fillna(1)
data["ARP vs. Prior Period"] = data["ARP vs. Prior Period"].fillna(1)

# PREVIOUS MONTH is NULL if this is the first month of the brand being sold
#     SO, set this value to a 0 (no units sold in previous month)
data["Previous Month"] = data["Previous Month"].fillna(0)

# TOTAL SALES ($) and ARP are NULL if no units are sold in that month for the brand
#     SO, set this value to 0
data["Total Sales ($)"] = data["Total Sales ($)"].fillna(0)
data["ARP"] = data["ARP"].fillna(0)
```

In [ ]:

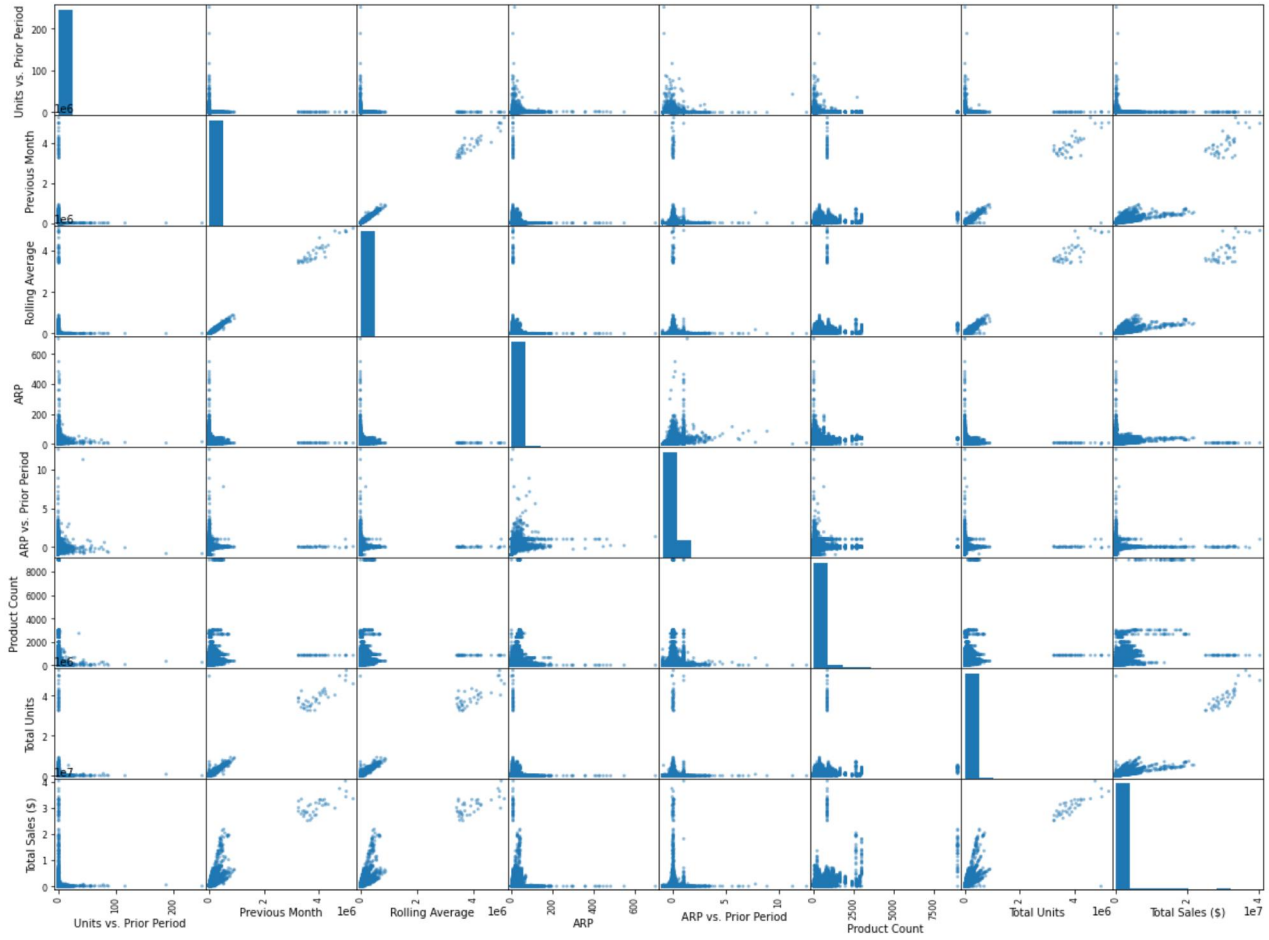
```
data.isna().sum().sum() # NO NULL
```

Out[ ]:

```
0
```

In [ ]:

```
# CORRELATION MATRIX: MANY TIME SERIES FEATURES ARE CORRELATED
data_corr = data.corr()
from pandas.plotting import scatter_matrix
attrs = ["Units vs. Prior Period", "Previous Month", "Rolling Average", "ARP", "ARP vs.
scatter_matrix(data[attrs], figsize=(20,15))
plt.show()
```



In [ ]:

```
# WE WILL PREDICT TOTAL SALES
data_y = data["Total Sales ($)"].copy()
data_X = data.drop(columns=["Total Sales ($)"])
data_X
```

Out[ ]:

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	ARP	A
0	#BlackSeries	1616.3390	1.000000	0.0000	0.000000	8	2020	3	15.684913	1.(
1	#BlackSeries	0.0000	-1.000000	1616.3390	1616.339000	9	2020	4	0.000000	-1.(
2	#BlackSeries	715.5328	1.000000	0.0000	808.169500	1	2021	1	13.611428	1.(
3	#BlackSeries	766.6691	0.071466	715.5328	777.290600	2	2021	1	11.873182	-0.'
4	#BlackSeries	0.0000	-1.000000	766.6691	494.067300	3	2021	2	0.000000	-1.(
...	...	...	...	...	...	...	...	...	...	
27681	Zuma Topicals	312.5153	1.000000	0.0000	0.000000	8	2019	3	31.598214	1.(
27682	Zuma Topicals	464.3063	0.485707	312.5153	312.515300	9	2019	4	37.860964	0.'
27683	Zuma Topicals	348.0579	-0.250370	464.3063	388.410800	10	2019	4	34.546154	-0.(

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	ARP	A
27684	Zuma Topicals	135.9220	-0.609484	348.0579	374.959833	11	2019	4	36.850000	0.(
27685	Zuma Topicals	0.0000	-1.000000	135.9220	316.095400	12	2019	1	0.000000	-1.(

27686 rows × 18 columns

In [ ]:
```
# PIPELINE TIME
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

# cat features are Brands, Month, Year, Season, Predominant Type, Predominant Flavor
data_num = data_X.drop(["Brands", "Month", "Year", "Season", "Predominant Product Type"
numerical_features = list(data_num)
categorical_features = ["Brands", "Month", "Year", "Season", "Predominant Product Type"

num_flavors_idx, product_count_idx = 11, 8

class AugmentFeatures(BaseEstimator, TransformerMixin):
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    flavored_products = X[:, num_flavors_idx] * X[:, product_count_idx]
    return np.c_[X, flavored_products]

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

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
data_prepared = full_pipeline.fit_transform(data_X)
data_prepared
```

Out[ ]:
```
<27686x1750 sparse matrix of type '<class 'numpy.float64'>'
        with 498348 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
# STATISTICS
import statsmodels.api as sm
# build the OLS model (ordinary least squares) from the training data
```

```
trimmed_data_num = data_X.drop(["Brands", "Month", "Year", "Season", "Predominant Produ
numerical_features = list(data_num)
categorical_features = ["Month", "Year", "Season", "Predominant Product Type", "Predomi


full_pipeline = ColumnTransformer([
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
trimmed_data_prepared = full_pipeline.fit_transform(data_X)

avo_stats = sm.OLS(data_y, trimmed_data_prepared.toarray())

results_stats = avo_stats.fit()
print(results_stats.summary())
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing
instead.
  import pandas.util.testing as tm
                            OLS Regression Results
==============================================================================
Dep. Variable:        Total Sales ($)   R-squared:                       0.874
Model:                            OLS   Adj. R-squared:                  0.873
Method:                 Least Squares   F-statistic:                     1905.
Date:                Thu, 02 Dec 2021   Prob (F-statistic):               0.00
Time:                        19:45:10   Log-Likelihood:            -4.0492e+05
No. Observations:               27686   AIC:                         8.100e+05
Df Residuals:                   27585   BIC:                         8.109e+05
Df Model:                         100
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1          2.601e+05   1.45e+04     17.888      0.000    2.32e+05    2.89e+05
x2          3543.7790   3294.737      1.076      0.282   -2914.070       1e+04
x3          7.535e+05   4.36e+04     17.301      0.000    6.68e+05    8.39e+05
x4          2.213e+05   4.25e+04      5.200      0.000    1.38e+05    3.05e+05
x5          4.025e+04   3690.308     10.907      0.000     3.3e+04    4.75e+04
x6         -1.434e+04   3481.057     -4.119      0.000   -2.12e+04   -7515.813
x7          5.833e+04   1.06e+04      5.513      0.000    3.76e+04    7.91e+04
const       1.193e-10   2.65e-10      0.451      0.652      -4e-10    6.38e-10
x8          5.091e+05   3694.356    137.812      0.000    5.02e+05    5.16e+05
x9          6.427e+04   3981.650     16.140      0.000    5.65e+04    7.21e+04
x10         8.104e+04   5181.786     15.640      0.000    7.09e+04    9.12e+04
x11        -1.589e+04   3855.895     -4.120      0.000   -2.34e+04   -8327.503
x12        -6056.8122   1.05e+04     -0.576      0.564   -2.66e+04    1.45e+04
x13        -1.328e+04   1.05e+04     -1.269      0.204   -3.38e+04    7222.858
x14         2.042e+04   1.02e+04      1.998      0.046     389.934    4.04e+04
x15        -9031.0836   1.02e+04     -0.888      0.374    -2.9e+04    1.09e+04
x16         7035.3866      1e+04      0.704      0.482   -1.26e+04    2.66e+04
x17         -566.3172   9835.101     -0.058      0.954   -1.98e+04    1.87e+04
x18           2.5e+04   9813.576      2.548      0.011    5765.261    4.42e+04
x19        -1.444e+04   9348.061     -1.544      0.122   -3.28e+04    3885.019
x20         5476.5938   9479.954      0.578      0.563   -1.31e+04    2.41e+04
x21         5543.8394   1.05e+04      0.529      0.597    -1.5e+04    2.61e+04
x22        -2996.0157   1.04e+04     -0.288      0.773   -2.34e+04    1.74e+04
x23         1.917e+04   1.07e+04      1.789      0.074   -1829.734    4.02e+04
x24        -1.286e+04   1.47e+04     -0.872      0.383   -4.18e+04     1.6e+04
```

x25	9287.4084	1.17e+04	0.792	0.428	-1.37e+04	3.23e+04
x26	1.486e+04	1.14e+04	1.303	0.192	-7488.495	3.72e+04
x27	2.5e+04	1.18e+04	2.127	0.033	1963.753	4.8e+04
x28	-159.5727	1.12e+04	-0.014	0.989	-2.2e+04	2.17e+04
x29	1.842e+04	1.13e+04	1.637	0.102	-3635.743	4.05e+04
x30	9996.3953	1.1e+04	0.908	0.364	-1.16e+04	3.16e+04
x31	8024.4175	1.1e+04	0.727	0.467	-1.36e+04	2.96e+04
x32	2.701e+05	2.76e+04	9.775	0.000	2.16e+05	3.24e+05
x33	4.02e+04	1.56e+04	2.582	0.010	9683.233	7.07e+04
x34	2.087e+05	2.02e+04	10.317	0.000	1.69e+05	2.48e+05
x35	2.662e+05	1.62e+04	16.406	0.000	2.34e+05	2.98e+05
x36	1.203e+05	1.61e+04	7.466	0.000	8.87e+04	1.52e+05
x37	3.049e+05	1.6e+04	19.063	0.000	2.74e+05	3.36e+05
x38	2.154e+04	2.7e+04	0.796	0.426	-3.15e+04	7.46e+04
x39	7.858e+04	1.68e+04	4.681	0.000	4.57e+04	1.11e+05
x40	-1.479e+06	8.26e+04	-17.913	0.000	-1.64e+06	-1.32e+06
x41	6.514e+04	1.99e+04	3.280	0.001	2.62e+04	1.04e+05
x42	1.398e+05	2.12e+04	6.588	0.000	9.82e+04	1.81e+05
x43	2920.6996	1.5e+05	0.019	0.985	-2.92e+05	2.98e+05
x44	7.46e+04	1.17e+05	0.640	0.522	-1.54e+05	3.03e+05
x45	1.23e+04	3.8e+05	0.032	0.974	-7.33e+05	7.58e+05
x46	2.899e+04	1.45e+05	0.200	0.842	-2.56e+05	3.14e+05
x47	6948.2421	3.8e+05	0.018	0.985	-7.38e+05	7.52e+05
x48	-1.367e+05	1.04e+05	-1.312	0.190	-3.41e+05	6.76e+04
x49	4.063e+04	1.04e+05	0.391	0.696	-1.63e+05	2.44e+05
x50	3.157e+04	1.8e+05	0.175	0.861	-3.22e+05	3.85e+05
x51	2.802e+04	2.41e+05	0.116	0.907	-4.45e+05	5.01e+05
x52	-1468.0816	1.63e+05	-0.009	0.993	-3.22e+05	3.19e+05
x53	-6128.6401	5.46e+04	-0.112	0.911	-1.13e+05	1.01e+05
x54	-8.662e+04	9.02e+04	-0.960	0.337	-2.63e+05	9.02e+04
x55	-2.658e+04	8.99e+04	-0.296	0.767	-2.03e+05	1.5e+05
x56	-1109.8559	1.25e+05	-0.009	0.993	-2.46e+05	2.44e+05
x57	4291.8154	9.74e+04	0.044	0.965	-1.87e+05	1.95e+05
x58	4585.6723	1.32e+05	0.035	0.972	-2.54e+05	2.63e+05
x59	3136.8857	1.4e+05	0.022	0.982	-2.72e+05	2.78e+05
x60	-1.66e+04	4.89e+04	-0.340	0.734	-1.12e+05	7.91e+04
x61	-1.464e+05	6.07e+04	-2.412	0.016	-2.65e+05	-2.74e+04
x62	-1.656e+05	1.51e+05	-1.098	0.272	-4.61e+05	1.3e+05
x63	2.054e+04	1.32e+05	0.156	0.876	-2.38e+05	2.79e+05
x64	1.193e+05	1.12e+05	1.067	0.286	-9.98e+04	3.38e+05
x65	2.152e+04	1.19e+05	0.181	0.857	-2.12e+05	2.55e+05
x66	2.187e+04	1.57e+05	0.140	0.889	-2.85e+05	3.29e+05
x67	3.691e+04	1.32e+05	0.280	0.780	-2.22e+05	2.96e+05
x68	8514.9134	1.14e+05	0.075	0.940	-2.15e+05	2.32e+05
x69	1.131e+04	1.5e+05	0.075	0.940	-2.84e+05	3.06e+05
x70	-1.395e+05	8.49e+04	-1.643	0.100	-3.06e+05	2.69e+04
x71	3.605e+04	1e+05	0.359	0.720	-1.61e+05	2.33e+05
x72	1.616e+04	2.41e+05	0.067	0.947	-4.56e+05	4.89e+05
x73	-1.808e+05	1.14e+05	-1.580	0.114	-4.05e+05	4.34e+04
x74	-1.757e+05	9.65e+04	-1.821	0.069	-3.65e+05	1.34e+04
x75	-1.426e+04	1.01e+05	-0.142	0.887	-2.11e+05	1.83e+05
x76	2.144e+05	2.6e+04	8.235	0.000	1.63e+05	2.65e+05
x77	1.002e+04	2.04e+05	0.049	0.961	-3.9e+05	4.1e+05
x78	1.571e+04	2.69e+05	0.058	0.953	-5.12e+05	5.44e+05
x79	2.258e+05	6.63e+04	3.408	0.001	9.59e+04	3.56e+05
x80	-1.386e+05	9.13e+04	-1.518	0.129	-3.17e+05	4.04e+04
x81	2.019e+05	1.12e+05	1.806	0.071	-1.72e+04	4.21e+05
x82	-1.559e+05	9.93e+04	-1.570	0.116	-3.51e+05	3.87e+04
x83	-1.814e+04	2.41e+05	-0.075	0.940	-4.91e+05	4.54e+05
x84	3.396e+04	1.57e+05	0.217	0.828	-2.73e+05	3.41e+05

```
x85        -1.777e+05   1.02e+05     -1.735      0.083   -3.79e+05    2.31e+04
x86         7212.9297   1.51e+05      0.048      0.962   -2.88e+05    3.02e+05
x87        -2069.5978   1.17e+05     -0.018      0.986   -2.31e+05    2.26e+05
x88         1.588e+04    2.2e+05      0.072      0.943   -4.16e+05    4.48e+05
x89        -1.528e+05    2.2e+05     -0.693      0.488   -5.85e+05    2.79e+05
x90         3.245e+04   1.57e+05      0.207      0.836   -2.74e+05    3.39e+05
x91          2.31e+04    3.8e+05      0.061      0.952   -7.22e+05    7.68e+05
x92         3.676e+04   1.06e+05      0.348      0.728   -1.71e+05    2.44e+05
x93        -3.217e+04   9.07e+04     -0.355      0.723    -2.1e+05    1.46e+05
x94         -1.98e+05   9.26e+04     -2.140      0.032   -3.79e+05   -1.66e+04
x95          9.23e+05      9e+04     10.260      0.000    7.47e+05     1.1e+06
x96         9.632e+04   8.99e+04      1.071      0.284   -7.99e+04    2.73e+05
x97         3.091e+04   4.54e+04      0.681      0.496   -5.81e+04     1.2e+05
x98         1.949e+04   9.21e+04      0.212      0.832   -1.61e+05       2e+05
x99        -2.122e+04   5.37e+05     -0.039      0.969   -1.07e+06    1.03e+06
x100       -5.374e+04   9.02e+04     -0.595      0.552   -2.31e+05    1.23e+05
x101        4402.1715   3.11e+05      0.014      0.989   -6.05e+05    6.14e+05
x102        1.278e+04   1.91e+05      0.067      0.947   -3.62e+05    3.87e+05
x103        1.677e+04    5.3e+04      0.317      0.751    -8.7e+04    1.21e+05
x104       -2.038e+05   2.69e+05     -0.756      0.450   -7.32e+05    3.24e+05
x105        -162.7544   1.14e+05     -0.001      0.999   -2.24e+05    2.24e+05
x106         1.92e+04   2.69e+05      0.071      0.943   -5.09e+05    5.47e+05
x107       -1.898e+05   1.06e+05     -1.792      0.073   -3.97e+05    1.78e+04
x108        7574.1780   3.11e+05      0.024      0.981   -6.02e+05    6.17e+05
==============================================================================
Omnibus:                    32913.853   Durbin-Watson:                   0.262
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         16978491.904
Skew:                           5.778   Prob(JB):                         0.00
Kurtosis:                     123.766   Cond. No.                     1.25e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The smallest eigenvalue is 5.4e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [ ]:

```
# METRICS (FROM LIONEL CODE)
import sklearn.metrics as metrics
def regression_results(y_true, y_pred):
    # Regression metrics
    explained_variance=metrics.explained_variance_score(y_true, y_pred)
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
    median_absolute_error=metrics.median_absolute_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    print('explained_variance: ', round(explained_variance,4))
    print('r2: ', round(r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))
```

In [ ]:

```
# SIMPLE TRAIN TEST SPLIT ON PIPELINE DATA + BASIC LINEAR REGRESSION MODEL
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_prepared, data_y, test_size=.2

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

1.3 Run Statistics **5 / 5**

✓ **- 0 pts** **Correct**

	Months	Brand	Total Sales ($)
0	09/2018	10x Infused	1,711.334232
1	09/2018	1964 Supply Co.	25,475.21594500000
2	09/2018	3 Bros Grow	120,153.644757
3	09/2018	3 Leaf	6,063.5297850000000
4	09/2018	350 Fire	631,510.0481550000

In [ ]:
```
units.info()
units.head()
```

```
<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
```

Out[ ]:

	Brands	Months	Total Units	vs. Prior Period
0	#BlackSeries	08/2020	1,616.3390040000000	NaN
1	#BlackSeries	09/2020	NaN	-1.000000
2	#BlackSeries	01/2021	715.5328380000000	NaN
3	#BlackSeries	02/2021	766.669135	0.071466
4	#BlackSeries	03/2021	NaN	-1.000000

In [ ]:
```
# TURN THE MONTHS OBJECT TO A DATETIME SO WE CAN EASILY EXTRACT THE MONTH AND YEAR
units['Months'] = pd.to_datetime(units['Months'])
sales['Months'] = pd.to_datetime(sales['Months'])
arp['Months'] = pd.to_datetime(arp['Months'])
```

In [ ]:
```
# TURN THE SALES AND UNITS INTO FLOATS
sales['Total Sales ($)'] = sales['Total Sales ($)'].str.replace(",", "")
sales['Total Sales ($)'] = sales['Total Sales ($)'].str[:8]
sales['Total Sales ($)'] = pd.to_numeric(sales["Total Sales ($)"])
units['Total Units'] = units['Total Units'].str.replace(",","")
units['Total Units'] = units['Total Units'].str[:8]
units['Total Units'] = pd.to_numeric(units["Total Units"])
```

In [ ]:
```
sales.info()
sales.head()
```

```
<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
```

Out[ ]:

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

In [ ]:

```
units.info()
units.head()
```

```
<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
```

Out[ ]:

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

In [ ]:

```
units["Total Units"] = units["Total Units"].fillna(0)
units
```

Out[ ]:

	Brands	Months	Total Units	vs. Prior Period
0	#BlackSeries	2020-08-01	1616.3390	NaN
1	#BlackSeries	2020-09-01	0.0000	-1.000000
2	#BlackSeries	2021-01-01	715.5328	NaN
3	#BlackSeries	2021-02-01	766.6691	0.071466

	Brands	Months	Total Units	vs. Prior Period
4	#BlackSeries	2021-03-01	0.0000	-1.000000
...	...	...	...	...
27681	Zuma Topicals	2019-08-01	312.5153	NaN
27682	Zuma Topicals	2019-09-01	464.3063	0.485707
27683	Zuma Topicals	2019-10-01	348.0579	-0.250370
27684	Zuma Topicals	2019-11-01	135.9220	-0.609484
27685	Zuma Topicals	2019-12-01	0.0000	-1.000000

27686 rows × 4 columns

In [ ]:
```
# RENAME THE PRIOR PERIOD COLUMNS SINCE THEY ARE DIFFERENT
units["Units vs. Prior Period"] = units["vs. Prior Period"]
units = units.drop(columns=['vs. Prior Period'])
arp["ARP vs. Prior Period"] = arp["vs. Prior Period"]
arp = arp.drop(columns=['vs. Prior Period'])
arp
```

Out[ ]:

	Brands	Months	ARP	ARP vs. Prior Period
0	#BlackSeries	2020-08-01	15.684913	NaN
1	#BlackSeries	2020-09-01	NaN	-1.000000
2	#BlackSeries	2021-01-01	13.611428	NaN
3	#BlackSeries	2021-02-01	11.873182	-0.127705
4	#BlackSeries	2021-03-01	NaN	-1.000000
...	...	...	...	...
27206	Zuma Topicals	2019-08-01	31.598214	NaN
27207	Zuma Topicals	2019-09-01	37.860964	0.198199
27208	Zuma Topicals	2019-10-01	34.546154	-0.087552
27209	Zuma Topicals	2019-11-01	36.850000	0.066689
27210	Zuma Topicals	2019-12-01	NaN	-1.000000

27211 rows × 4 columns

In [ ]:
```
all_brands = units["Brands"].unique()
all_brands
```

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

In [ ]:
```
data = pd.DataFrame()
for brand in all_brands:
  units_data = units[units.Brands == brand]
```

```
# PREVIOUS MONTH: THE TOTAL UNITS OF THE PREVIOUS MONTH
units_data.loc[:,'Previous Month'] = units_data.loc[:,'Total Units'].shift(1)
# ROLLING AVERAGE: THE AVERAGE TOTAL UNITS OF THE PAST 3 MONTHS
units_data.loc[:,'Rolling Average'] = (units_data.loc[:,'Total Units'].shift(3) + \
                                       units_data.loc[:,'Total Units'].shift(2) + uni

first = units_data.first_valid_index()
# IF THERE ARE NO, ONE, OR TWO PRIOR MONTHS, MANUALLY CALCULATE THE ROLLING AVERAGE
units_data.loc[first,['Rolling Average']] = 0
if units_data.shape[0] > 2:
  units_data.loc[first+2,'Rolling Average'] = (units_data.loc[first+1,'Total Units']
                                               units_data.loc[first, "Total Units"])
if units_data.shape[0] > 1:
  units_data.loc[first+1,'Rolling Average'] = units_data.loc[first, "Total Units"]

# YEAR: GET THE YEAR FROM THE MONTHS COLUMN
units_data["Month"] = pd.DatetimeIndex(units_data["Months"]).month
units_data['Year'] = pd.DatetimeIndex(units_data['Months']).year
# SEASON: WINTER IS 1, SPRING IS 2, SUMMER IS 3, FALL IS 4
units_data["Season"] = pd.DatetimeIndex(units_data['Months']).month%12 // 3 + 1

# MERGE WITH TOTAL SALES AND ARP
sales_data = sales[sales.Brand == brand]
units_data = units_data.merge(sales_data, left_on='Months', right_on='Months', how ="
units_data = units_data.drop(['Brand'], 1)

arp_data = arp[arp.Brands == brand]
units_data = units_data.merge(arp_data, left_on='Months', right_on='Months', how ="ou
units_data = units_data.drop(['Brands_y', "Months"], 1)
units_data = units_data.rename(columns={"Brands_x": "Brands"})

units_details = brands[brands.Brand == brand]

# INHALEABLE: IF THE BRAND HAS INHALEABLE PRODUCTS
# EDIBLE: LIKEWISE BUT FOR EDIBLE PRODUCTS
i = 0
e = 0
if 'Inhaleables' in units_details['Category L1'].values:
    i = 1
if 'Edibles' in units_details['Category L1'].values:
    e = 1
units_data['Inhaleable'] = i
units_data['Edible'] = e

# PRODUCT COUNT (NUMBER OF PRODUCTS IN BRAND DETAILS TABLE)
units_data['Product Count'] = (units_details.Brand == brand).count()

# MOOD SPECIFIC: IF ANY PRODUCT IS MOOD SPECIFIC
mood_specific = 0
if "Mood Specific" in units_details["Mood Effect"].values:
  mood_specific = 1
units_data["Mood Specific"] = mood_specific

# FLAVORED: IF ANY PRODUCT IS FLAVORED
flavored = 0
if "Flavored" in units_details["Is Flavored"].values:
  flavored = 1
units_data["Flavored"] = flavored

# PRODUCT TYPE: MOST COMMON TYPE (CONCENTRATES, FLOWER, PRE-ROLLED, ETC.)
```

```
  most_common_type = "None"
  lst = list(units_details["Category L2"].values)
  if lst:
    most_common_type = max(set(lst), key=lst.count)
  units_data["Predominant Product Type"] = most_common_type

  # FLAVOR: MOST COMMON FLAVOR
  # NUMBER OF FLAVORS: ALL FLAVORS OF BRAND EXCEPT NONE FLAVOR
  most_common_flavor = "None"
  num_of_flavors = 0
  lst = list(units_details["Flavor"].values)
  if lst:
    most_common_flavor = max(set(lst), key=lst.count)
    flavors = list(filter(pd.notna, lst))
    num_of_flavors = len(set(flavors))
  units_data["Predominant Flavor"] = most_common_flavor
  units_data["Predominant Flavor"] = units_data["Predominant Flavor"].fillna("None")
  units_data["Number of Flavors"] = num_of_flavors

  data = data.append(units_data, ignore_index=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1596: SettingWithCopyWarn
ing:
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)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1743: SettingWithCopyWarn
ing:
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)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1763: SettingWithCopyWarn
ing:
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(loc, value)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: 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
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: 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
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: 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
```

In [ ]:
```
data
```

Out[ ]:

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	Total Sales ($)	
0	#BlackSeries	1616.3390	NaN	NaN	0.000000	8	2020	3	25352.130	15.
1	#BlackSeries	0.0000	-1.000000	1616.3390	1616.339000	9	2020	4	NaN	
2	#BlackSeries	715.5328	NaN	0.0000	808.169500	1	2021	1	9739.423	13.
3	#BlackSeries	766.6691	0.071466	715.5328	777.290600	2	2021	1	9102.802	11.
4	#BlackSeries	0.0000	-1.000000	766.6691	494.067300	3	2021	2	NaN	
...	...	...	...	...	...	...	...	...	...	
27681	Zuma Topicals	312.5153	NaN	NaN	0.000000	8	2019	3	9874.926	31.
27682	Zuma Topicals	464.3063	0.485707	312.5153	312.515300	9	2019	4	17579.080	37.
27683	Zuma Topicals	348.0579	-0.250370	464.3063	388.410800	10	2019	4	12024.060	34.
27684	Zuma Topicals	135.9220	-0.609484	348.0579	374.959833	11	2019	4	5008.728	36.
27685	Zuma Topicals	0.0000	-1.000000	135.9220	316.095400	12	2019	1	NaN	

27686 rows × 19 columns

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27686 entries, 0 to 27685
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Brands                  27686 non-null  object
 1   Total Units             27686 non-null  float64
 2   Units vs. Prior Period  24935 non-null  float64
 3   Previous Month          26046 non-null  float64
 4   Rolling Average         27686 non-null  float64
 5   Month                   27686 non-null  int64
 6   Year                    27686 non-null  int64
 7   Season                  27686 non-null  int64
 8   Total Sales ($)         25279 non-null  float64
 9   ARP                     25279 non-null  float64
```

```
 10  ARP vs. Prior Period      24499 non-null  float64
 11  Inhaleable                27686 non-null  int64
 12  Edible                    27686 non-null  int64
 13  Product Count             27686 non-null  int64
 14  Mood Specific             27686 non-null  int64
 15  Flavored                  27686 non-null  int64
 16  Predominant Product Type  27686 non-null  object
 17  Predominant Flavor        27686 non-null  object
 18  Number of Flavors         27686 non-null  int64
dtypes: float64(7), int64(9), object(3)
memory usage: 4.0+ MB
```

In [ ]:

```
# IMPUTING TIME
# VS. PRIOR PERIOD is NULL if it is the first month of the brand being sold or if 0 uni
#     SO, we can just set this value to 1
data["Units vs. Prior Period"] = data["Units vs. Prior Period"].fillna(1)
data["ARP vs. Prior Period"] = data["ARP vs. Prior Period"].fillna(1)

# PREVIOUS MONTH is NULL if this is the first month of the brand being sold
#     SO, set this value to a 0 (no units sold in previous month)
data["Previous Month"] = data["Previous Month"].fillna(0)

# TOTAL SALES ($) and ARP are NULL if no units are sold in that month for the brand
#     SO, set this value to 0
data["Total Sales ($)"] = data["Total Sales ($)"].fillna(0)
data["ARP"] = data["ARP"].fillna(0)
```

In [ ]:

```
data.isna().sum().sum() # NO NULL
```

Out[ ]:

```
0
```

In [ ]:

```
# CORRELATION MATRIX: MANY TIME SERIES FEATURES ARE CORRELATED
data_corr = data.corr()
from pandas.plotting import scatter_matrix
attrs = ["Units vs. Prior Period", "Previous Month", "Rolling Average", "ARP", "ARP vs.
scatter_matrix(data[attrs], figsize=(20,15))
plt.show()
```

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	ARP	A
27684	Zuma Topicals	135.9220	-0.609484	348.0579	374.959833	11	2019	4	36.850000	0.(
27685	Zuma Topicals	0.0000	-1.000000	135.9220	316.095400	12	2019	1	0.000000	-1.(

27686 rows × 18 columns

In [ ]:
```
# PIPELINE TIME
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

# cat features are Brands, Month, Year, Season, Predominant Type, Predominant Flavor
data_num = data_X.drop(["Brands", "Month", "Year", "Season", "Predominant Product Type"
numerical_features = list(data_num)
categorical_features = ["Brands", "Month", "Year", "Season", "Predominant Product Type"

num_flavors_idx, product_count_idx = 11, 8

class AugmentFeatures(BaseEstimator, TransformerMixin):
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    flavored_products = X[:, num_flavors_idx] * X[:, product_count_idx]
    return np.c_[X, flavored_products]

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

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
data_prepared = full_pipeline.fit_transform(data_X)
data_prepared
```

Out[ ]:
```
<27686x1750 sparse matrix of type '<class 'numpy.float64'>'
        with 498348 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
# STATISTICS
import statsmodels.api as sm
# build the OLS model (ordinary least squares) from the training data
```

1.4 Feature extraction and pipeline **5 / 5**

✓ **- 0 pts** **Correct**

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	ARP	A
27684	Zuma Topicals	135.9220	-0.609484	348.0579	374.959833	11	2019	4	36.850000	0.(
27685	Zuma Topicals	0.0000	-1.000000	135.9220	316.095400	12	2019	1	0.000000	-1.(

27686 rows × 18 columns

In [ ]:
```
# PIPELINE TIME
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

# cat features are Brands, Month, Year, Season, Predominant Type, Predominant Flavor
data_num = data_X.drop(["Brands", "Month", "Year", "Season", "Predominant Product Type"
numerical_features = list(data_num)
categorical_features = ["Brands", "Month", "Year", "Season", "Predominant Product Type"

num_flavors_idx, product_count_idx = 11, 8

class AugmentFeatures(BaseEstimator, TransformerMixin):
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    flavored_products = X[:, num_flavors_idx] * X[:, product_count_idx]
    return np.c_[X, flavored_products]

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

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
data_prepared = full_pipeline.fit_transform(data_X)
data_prepared
```

Out[ ]:
```
<27686x1750 sparse matrix of type '<class 'numpy.float64'>'
        with 498348 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
# STATISTICS
import statsmodels.api as sm
# build the OLS model (ordinary least squares) from the training data
```

1.5 Pipeline **5 / 5**

✓ **- 0 pts** Correct

```
x85        -1.777e+05   1.02e+05     -1.735      0.083   -3.79e+05    2.31e+04
x86         7212.9297   1.51e+05      0.048      0.962   -2.88e+05    3.02e+05
x87        -2069.5978   1.17e+05     -0.018      0.986   -2.31e+05    2.26e+05
x88         1.588e+04    2.2e+05      0.072      0.943   -4.16e+05    4.48e+05
x89        -1.528e+05    2.2e+05     -0.693      0.488   -5.85e+05    2.79e+05
x90         3.245e+04   1.57e+05      0.207      0.836   -2.74e+05    3.39e+05
x91          2.31e+04    3.8e+05      0.061      0.952   -7.22e+05    7.68e+05
x92         3.676e+04   1.06e+05      0.348      0.728   -1.71e+05    2.44e+05
x93        -3.217e+04   9.07e+04     -0.355      0.723    -2.1e+05    1.46e+05
x94         -1.98e+05   9.26e+04     -2.140      0.032   -3.79e+05   -1.66e+04
x95          9.23e+05      9e+04     10.260      0.000    7.47e+05     1.1e+06
x96         9.632e+04   8.99e+04      1.071      0.284   -7.99e+04    2.73e+05
x97         3.091e+04   4.54e+04      0.681      0.496   -5.81e+04     1.2e+05
x98         1.949e+04   9.21e+04      0.212      0.832   -1.61e+05       2e+05
x99        -2.122e+04   5.37e+05     -0.039      0.969   -1.07e+06    1.03e+06
x100       -5.374e+04   9.02e+04     -0.595      0.552   -2.31e+05    1.23e+05
x101        4402.1715   3.11e+05      0.014      0.989   -6.05e+05    6.14e+05
x102        1.278e+04   1.91e+05      0.067      0.947   -3.62e+05    3.87e+05
x103        1.677e+04    5.3e+04      0.317      0.751    -8.7e+04    1.21e+05
x104       -2.038e+05   2.69e+05     -0.756      0.450   -7.32e+05    3.24e+05
x105        -162.7544   1.14e+05     -0.001      0.999   -2.24e+05    2.24e+05
x106         1.92e+04   2.69e+05      0.071      0.943   -5.09e+05    5.47e+05
x107       -1.898e+05   1.06e+05     -1.792      0.073   -3.97e+05    1.78e+04
x108        7574.1780   3.11e+05      0.024      0.981   -6.02e+05    6.17e+05
==============================================================================
Omnibus:                    32913.853   Durbin-Watson:                   0.262
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         16978491.904
Skew:                           5.778   Prob(JB):                         0.00
Kurtosis:                     123.766   Cond. No.                     1.25e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The smallest eigenvalue is 5.4e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [ ]:

```
# METRICS (FROM LIONEL CODE)
import sklearn.metrics as metrics
def regression_results(y_true, y_pred):
    # Regression metrics
    explained_variance=metrics.explained_variance_score(y_true, y_pred)
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
    median_absolute_error=metrics.median_absolute_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    print('explained_variance: ', round(explained_variance,4))
    print('r2: ', round(r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))
```

In [ ]:

```
# SIMPLE TRAIN TEST SPLIT ON PIPELINE DATA + BASIC LINEAR REGRESSION MODEL
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_prepared, data_y, test_size=.2

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

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

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

```
explained_variance:  0.947
r2:  0.947
MAE:  106510.7698
MSE:  125684627468.7391
RMSE:  354520.2779
```

In [ ]:
```
data_prepared.shape # HELLA DIMENSIONS
```

Out[ ]:
```
(27686, 1750)
```

In [ ]:
```
# DO PCA
from sklearn.decomposition import PCA, TruncatedSVD
pca = TruncatedSVD(n_components=30) # sparse data, PCA does not work

data_pca = pca.fit_transform(data_prepared)
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(data_pca, data_y, t

# LINEAR REGRESSION AGAIN WITH NEW PIPELINED DATA
lin_reg = LinearRegression()
lin_reg.fit(new_X_train, new_y_train)

preds = lin_reg.predict(new_X_test)
regression_results(new_y_test, preds)
```

```
explained_variance:  0.8751
r2:  0.8751
MAE:  215803.0669
MSE:  296002496838.9682
RMSE:  544061.115
```

In [ ]:
```
# ENSEMBLE METHOD: RANDOM FOREST
from sklearn.ensemble import RandomForestRegressor

tree = RandomForestRegressor(min_samples_leaf=5, max_depth=100, random_state=42)

tree.fit(X_train, y_train)

preds_test = tree.predict(X_test)

regression_results(preds_test, y_test)
```

```
explained_variance:  0.992
r2:  0.992
MAE:  16733.0223
MSE:  18721233198.4262
RMSE:  136825.5575
```

In [ ]:
```
# CROSS VALIDATE BOTH SIMPLE REGRESSION AND RANDOM FOREST
from sklearn.model_selection import  KFold
```

1.6 Linear Regression **5 / 5**

✓ **- 0 pts** **Correct**

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

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

```
explained_variance:  0.947
r2:  0.947
MAE:  106510.7698
MSE:  125684627468.7391
RMSE:  354520.2779
```

In [ ]:
```
data_prepared.shape # HELLA DIMENSIONS
```

Out[ ]:
```
(27686, 1750)
```

In [ ]:
```
# DO PCA
from sklearn.decomposition import PCA, TruncatedSVD
pca = TruncatedSVD(n_components=30) # sparse data, PCA does not work

data_pca = pca.fit_transform(data_prepared)
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(data_pca, data_y, t

# LINEAR REGRESSION AGAIN WITH NEW PIPELINED DATA
lin_reg = LinearRegression()
lin_reg.fit(new_X_train, new_y_train)

preds = lin_reg.predict(new_X_test)
regression_results(new_y_test, preds)
```

```
explained_variance:  0.8751
r2:  0.8751
MAE:  215803.0669
MSE:  296002496838.9682
RMSE:  544061.115
```

In [ ]:
```
# ENSEMBLE METHOD: RANDOM FOREST
from sklearn.ensemble import RandomForestRegressor

tree = RandomForestRegressor(min_samples_leaf=5, max_depth=100, random_state=42)

tree.fit(X_train, y_train)

preds_test = tree.predict(X_test)

regression_results(preds_test, y_test)
```

```
explained_variance:  0.992
r2:  0.992
MAE:  16733.0223
MSE:  18721233198.4262
RMSE:  136825.5575
```

In [ ]:
```
# CROSS VALIDATE BOTH SIMPLE REGRESSION AND RANDOM FOREST
from sklearn.model_selection import  KFold
```

1.7 PCA 5 / 5

✓ - 0 pts Correct

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

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

```
explained_variance:  0.947
r2:  0.947
MAE:  106510.7698
MSE:  125684627468.7391
RMSE:  354520.2779
```

In [ ]:
```
data_prepared.shape # HELLA DIMENSIONS
```

Out[ ]:
```
(27686, 1750)
```

In [ ]:
```
# DO PCA
from sklearn.decomposition import PCA, TruncatedSVD
pca = TruncatedSVD(n_components=30) # sparse data, PCA does not work

data_pca = pca.fit_transform(data_prepared)
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(data_pca, data_y, t

# LINEAR REGRESSION AGAIN WITH NEW PIPELINED DATA
lin_reg = LinearRegression()
lin_reg.fit(new_X_train, new_y_train)

preds = lin_reg.predict(new_X_test)
regression_results(new_y_test, preds)
```

```
explained_variance:  0.8751
r2:  0.8751
MAE:  215803.0669
MSE:  296002496838.9682
RMSE:  544061.115
```

In [ ]:
```
# ENSEMBLE METHOD: RANDOM FOREST
from sklearn.ensemble import RandomForestRegressor

tree = RandomForestRegressor(min_samples_leaf=5, max_depth=100, random_state=42)

tree.fit(X_train, y_train)

preds_test = tree.predict(X_test)

regression_results(preds_test, y_test)
```

```
explained_variance:  0.992
r2:  0.992
MAE:  16733.0223
MSE:  18721233198.4262
RMSE:  136825.5575
```

In [ ]:
```
# CROSS VALIDATE BOTH SIMPLE REGRESSION AND RANDOM FOREST
from sklearn.model_selection import  KFold
```

1.8 Ensemble Method 5 / 5

✓ - 0 pts Correct

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

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

```
explained_variance:  0.947
r2:  0.947
MAE:  106510.7698
MSE:  125684627468.7391
RMSE:  354520.2779
```

In [ ]:
```
data_prepared.shape # HELLA DIMENSIONS
```

Out[ ]:
```
(27686, 1750)
```

In [ ]:
```
# DO PCA
from sklearn.decomposition import PCA, TruncatedSVD
pca = TruncatedSVD(n_components=30) # sparse data, PCA does not work

data_pca = pca.fit_transform(data_prepared)
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(data_pca, data_y, t

# LINEAR REGRESSION AGAIN WITH NEW PIPELINED DATA
lin_reg = LinearRegression()
lin_reg.fit(new_X_train, new_y_train)

preds = lin_reg.predict(new_X_test)
regression_results(new_y_test, preds)
```

```
explained_variance:  0.8751
r2:  0.8751
MAE:  215803.0669
MSE:  296002496838.9682
RMSE:  544061.115
```

In [ ]:
```
# ENSEMBLE METHOD: RANDOM FOREST
from sklearn.ensemble import RandomForestRegressor

tree = RandomForestRegressor(min_samples_leaf=5, max_depth=100, random_state=42)

tree.fit(X_train, y_train)

preds_test = tree.predict(X_test)

regression_results(preds_test, y_test)
```

```
explained_variance:  0.992
r2:  0.992
MAE:  16733.0223
MSE:  18721233198.4262
RMSE:  136825.5575
```

In [ ]:
```
# CROSS VALIDATE BOTH SIMPLE REGRESSION AND RANDOM FOREST
from sklearn.model_selection import  KFold
```

```
from sklearn import model_selection
from sklearn.model_selection import cross_validate, GridSearchCV, cross_val_score
import sklearn

kfold = model_selection.KFold(n_splits=5, random_state=42, shuffle=True)
clf = LinearRegression()
scores = cross_validate(clf, data_prepared, data_y, scoring=('r2', 'neg_mean_absolute_e

# GET MAE BY JUST MULTILPYING NMAE BY -1
print("Linear Regression MAE: " + str(( round(scores['test_neg_mean_absolute_error'].me
print("Linear Regression R2: " + str(( round(scores['test_r2'].mean(), 4) )))
print("Linear Regression Explained Variance: " + str(( round(scores['test_explained_var

clf = RandomForestRegressor(max_depth=50, n_estimators=75, random_state=42)

scores = cross_validate(clf, data_prepared, data_y, scoring=('r2', 'neg_mean_absolute_e

print("Random Forest MAE: " + str(( round(scores['test_neg_mean_absolute_error'].mean()
print("Random Forest R2: " + str(( round(scores['test_r2'].mean(), 4) )))
print("Random Forest Explained Variance: " + str(( round(scores['test_explained_varianc
```

```
Linear Regression MAE: 103071.2712
Linear Regression R2: 0.9405
Linear Regression Explained Variance: 0.9405
Random Forest MAE: 13530.9548
Random Forest R2: 0.9921
Random Forest Explained Variance: 0.9921
```

As we can see, the random forest model is better; we will try to optimize the paramters of the random forest at the expense of running the cell for long.

In [28]:

```
# GRID SEARCH: THIS CELL ACTUALLY TAKES LIKE 50 MINUTES
# OPTIMIZE max_depth AND n_estimators
parameters = {'max_depth':[25, 50], 'n_estimators':[25, 50, 75, 150]}
rf = RandomForestRegressor(random_state=42)
kfold = model_selection.KFold(n_splits=3, random_state=42, shuffle=True)
clf = GridSearchCV(rf, parameters, scoring='neg_mean_absolute_error', cv=kfold, verbose
clf.fit(data_prepared, data_y)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[CV 1/3] END max_depth=25, n_estimators=25;, score=-16472.223 total time=  58.4s
[CV 2/3] END max_depth=25, n_estimators=25;, score=-13479.780 total time=  59.5s
[CV 3/3] END max_depth=25, n_estimators=25;, score=-17322.710 total time=  59.8s
[CV 1/3] END max_depth=25, n_estimators=50;, score=-15654.330 total time= 2.0min
[CV 2/3] END max_depth=25, n_estimators=50;, score=-13333.117 total time= 2.0min
[CV 3/3] END max_depth=25, n_estimators=50;, score=-16721.667 total time= 2.0min
[CV 1/3] END max_depth=25, n_estimators=75;, score=-15515.506 total time= 2.9min
[CV 2/3] END max_depth=25, n_estimators=75;, score=-13037.416 total time= 2.9min
[CV 3/3] END max_depth=25, n_estimators=75;, score=-16380.360 total time= 2.9min
[CV 1/3] END max_depth=25, n_estimators=150;, score=-15416.385 total time= 5.7min
[CV 2/3] END max_depth=25, n_estimators=150;, score=-12696.708 total time= 5.8min
[CV 3/3] END max_depth=25, n_estimators=150;, score=-16477.157 total time= 5.8min
[CV 1/3] END max_depth=50, n_estimators=25;, score=-16214.121 total time=  58.0s
[CV 2/3] END max_depth=50, n_estimators=25;, score=-13404.091 total time=  58.2s
[CV 3/3] END max_depth=50, n_estimators=25;, score=-17298.181 total time=  57.9s
[CV 1/3] END max_depth=50, n_estimators=50;, score=-15524.397 total time= 1.9min
[CV 2/3] END max_depth=50, n_estimators=50;, score=-13289.045 total time= 2.0min
[CV 3/3] END max_depth=50, n_estimators=50;, score=-16690.104 total time= 2.0min
[CV 1/3] END max_depth=50, n_estimators=75;, score=-15633.772 total time= 2.9min
```

1.9 Cross-Validate results **5 / 5**

✓ **- 0 pts** **Correct**

```
from sklearn import model_selection
from sklearn.model_selection import cross_validate, GridSearchCV, cross_val_score
import sklearn

kfold = model_selection.KFold(n_splits=5, random_state=42, shuffle=True)
clf = LinearRegression()
scores = cross_validate(clf, data_prepared, data_y, scoring=('r2', 'neg_mean_absolute_e

# GET MAE BY JUST MULTILPYING NMAE BY -1
print("Linear Regression MAE: " + str(( round(scores['test_neg_mean_absolute_error'].me
print("Linear Regression R2: " + str(( round(scores['test_r2'].mean(), 4) )))
print("Linear Regression Explained Variance: " + str(( round(scores['test_explained_var

clf = RandomForestRegressor(max_depth=50, n_estimators=75, random_state=42)

scores = cross_validate(clf, data_prepared, data_y, scoring=('r2', 'neg_mean_absolute_e

print("Random Forest MAE: " + str(( round(scores['test_neg_mean_absolute_error'].mean()
print("Random Forest R2: " + str(( round(scores['test_r2'].mean(), 4) )))
print("Random Forest Explained Variance: " + str(( round(scores['test_explained_varianc
```

```
Linear Regression MAE: 103071.2712
Linear Regression R2: 0.9405
Linear Regression Explained Variance: 0.9405
Random Forest MAE: 13530.9548
Random Forest R2: 0.9921
Random Forest Explained Variance: 0.9921
```

As we can see, the random forest model is better; we will try to optimize the paramters of the random forest at the expense of running the cell for long.

In [28]:

```
# GRID SEARCH: THIS CELL ACTUALLY TAKES LIKE 50 MINUTES
# OPTIMIZE max_depth AND n_estimators
parameters = {'max_depth':[25, 50], 'n_estimators':[25, 50, 75, 150]}
rf = RandomForestRegressor(random_state=42)
kfold = model_selection.KFold(n_splits=3, random_state=42, shuffle=True)
clf = GridSearchCV(rf, parameters, scoring='neg_mean_absolute_error', cv=kfold, verbose
clf.fit(data_prepared, data_y)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[CV 1/3] END max_depth=25, n_estimators=25;, score=-16472.223 total time=  58.4s
[CV 2/3] END max_depth=25, n_estimators=25;, score=-13479.780 total time=  59.5s
[CV 3/3] END max_depth=25, n_estimators=25;, score=-17322.710 total time=  59.8s
[CV 1/3] END max_depth=25, n_estimators=50;, score=-15654.330 total time= 2.0min
[CV 2/3] END max_depth=25, n_estimators=50;, score=-13333.117 total time= 2.0min
[CV 3/3] END max_depth=25, n_estimators=50;, score=-16721.667 total time= 2.0min
[CV 1/3] END max_depth=25, n_estimators=75;, score=-15515.506 total time= 2.9min
[CV 2/3] END max_depth=25, n_estimators=75;, score=-13037.416 total time= 2.9min
[CV 3/3] END max_depth=25, n_estimators=75;, score=-16380.360 total time= 2.9min
[CV 1/3] END max_depth=25, n_estimators=150;, score=-15416.385 total time= 5.7min
[CV 2/3] END max_depth=25, n_estimators=150;, score=-12696.708 total time= 5.8min
[CV 3/3] END max_depth=25, n_estimators=150;, score=-16477.157 total time= 5.8min
[CV 1/3] END max_depth=50, n_estimators=25;, score=-16214.121 total time=  58.0s
[CV 2/3] END max_depth=50, n_estimators=25;, score=-13404.091 total time=  58.2s
[CV 3/3] END max_depth=50, n_estimators=25;, score=-17298.181 total time=  57.9s
[CV 1/3] END max_depth=50, n_estimators=50;, score=-15524.397 total time= 1.9min
[CV 2/3] END max_depth=50, n_estimators=50;, score=-13289.045 total time= 2.0min
[CV 3/3] END max_depth=50, n_estimators=50;, score=-16690.104 total time= 2.0min
[CV 1/3] END max_depth=50, n_estimators=75;, score=-15633.772 total time= 2.9min
```

```
[CV 2/3] END max_depth=50, n_estimators=75;, score=-13076.798 total time= 2.9min
[CV 3/3] END max_depth=50, n_estimators=75;, score=-16314.827 total time= 2.9min
[CV 1/3] END max_depth=50, n_estimators=150;, score=-15438.368 total time= 5.8min
[CV 2/3] END max_depth=50, n_estimators=150;, score=-12600.409 total time= 5.9min
[CV 3/3] END max_depth=50, n_estimators=150;, score=-16501.168 total time= 5.8min
```

Out[28]:

```
GridSearchCV(cv=KFold(n_splits=3, random_state=42, shuffle=True),
             estimator=RandomForestRegressor(random_state=42),
             param_grid={'max_depth': [25, 50],
                         'n_estimators': [25, 50, 75, 150]},
             scoring='neg_mean_absolute_error', verbose=3)
```

In [29]:

```
print("Best estimator: " + str(clf.best_estimator_))
print("Best score: " + str(round(-1*clf.best_score_, 4)))
```

```
Best estimator: RandomForestRegressor(max_depth=50, n_estimators=150, random_state=42)
Best score: 14846.6484
```

In [30]:

```
# EXPERIMENT: USE SEPT 2021 AS TEST SET, 2021 AS TRAINING SET
# GET ALL BRANDS FOR 2021, FILTER OUT BRANDS THAT ARE NOT IN SEPT 2021 AND THEN SPLIT

data2021 = data[data.Year == 2021]
data2021_nosept = data2021[data.Month < 9]
brands_nosept = list(data2021_nosept.Brands.unique())
brands_nosept
len(brands_nosept)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: Boolean Ser
ies key will be reindexed to match DataFrame index.
  """
```

Out[30]:

```
1135
```

In [31]:

```
data2021_sept = data2021[data2021.Month == 9]
data2021_sept
```

Out[31]:

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	Total Sales ($)	
27	101 Cannabis Co.	483.5366	0.147842	421.2570	283.728733	9	2021	4	12276.800	25
50	11:11	1940.3680	0.485852	1305.8960	1841.375000	9	2021	4	74277.070	38
100	1Lyfe	17504.2000	0.189840	14711.4000	11116.417000	9	2021	4	125998.500	7
136	22 Red	0.0000	-1.000000	984.1616	2249.841867	9	2021	4	0.000	(
244	3C Farms	11988.1700	3.218593	2841.7470	5380.730667	9	2021	4	160017.200	13
...	...	...	...	...	...	...	...	...	...	
27498	Yummi Karma	12757.2300	-0.074458	13783.5200	14165.503333	9	2021	4	458600.900	35
27499	Zanna	456.1666	1.000000	0.0000	0.000000	9	2021	4	3681.159	8
27585	Zendo Edibles	719.9865	-0.226109	930.3460	1261.215033	9	2021	4	11508.210	15

1.10 GridSearch 5 / 5

✓ - 0 pts Correct

- 5 pts missing

```
[CV 2/3] END max_depth=50, n_estimators=75;, score=-13076.798 total time= 2.9min
[CV 3/3] END max_depth=50, n_estimators=75;, score=-16314.827 total time= 2.9min
[CV 1/3] END max_depth=50, n_estimators=150;, score=-15438.368 total time= 5.8min
[CV 2/3] END max_depth=50, n_estimators=150;, score=-12600.409 total time= 5.9min
[CV 3/3] END max_depth=50, n_estimators=150;, score=-16501.168 total time= 5.8min
```

Out[28]:

```
GridSearchCV(cv=KFold(n_splits=3, random_state=42, shuffle=True),
             estimator=RandomForestRegressor(random_state=42),
             param_grid={'max_depth': [25, 50],
                         'n_estimators': [25, 50, 75, 150]},
             scoring='neg_mean_absolute_error', verbose=3)
```

In [29]:

```
print("Best estimator: " + str(clf.best_estimator_))
print("Best score: " + str(round(-1*clf.best_score_, 4)))
```

```
Best estimator: RandomForestRegressor(max_depth=50, n_estimators=150, random_state=42)
Best score: 14846.6484
```

In [30]:

```
# EXPERIMENT: USE SEPT 2021 AS TEST SET, 2021 AS TRAINING SET
# GET ALL BRANDS FOR 2021, FILTER OUT BRANDS THAT ARE NOT IN SEPT 2021 AND THEN SPLIT

data2021 = data[data.Year == 2021]
data2021_nosept = data2021[data.Month < 9]
brands_nosept = list(data2021_nosept.Brands.unique())
brands_nosept
len(brands_nosept)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: Boolean Ser
ies key will be reindexed to match DataFrame index.
  """
```

Out[30]:

```
1135
```

In [31]:

```
data2021_sept = data2021[data2021.Month == 9]
data2021_sept
```

Out[31]:

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	Total Sales ($)	
27	101 Cannabis Co.	483.5366	0.147842	421.2570	283.728733	9	2021	4	12276.800	25
50	11:11	1940.3680	0.485852	1305.8960	1841.375000	9	2021	4	74277.070	38
100	1Lyfe	17504.2000	0.189840	14711.4000	11116.417000	9	2021	4	125998.500	7
136	22 Red	0.0000	-1.000000	984.1616	2249.841867	9	2021	4	0.000	(
244	3C Farms	11988.1700	3.218593	2841.7470	5380.730667	9	2021	4	160017.200	13
...	...	...	...	...	...	...	...	...	...	
27498	Yummi Karma	12757.2300	-0.074458	13783.5200	14165.503333	9	2021	4	458600.900	35
27499	Zanna	456.1666	1.000000	0.0000	0.000000	9	2021	4	3681.159	8
27585	Zendo Edibles	719.9865	-0.226109	930.3460	1261.215033	9	2021	4	11508.210	15

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	Total Sales ($)	
27627	Zig Zag	6688.6430	-0.184473	8201.6210	8021.726667	9	2021	4	18077.300	2
27635	Zips Weed Co.	210.2150	0.084823	193.7782	398.608633	9	2021	4	3811.072	18

854 rows × 19 columns

In [32]:
```
brands_sept = list(data2021_sept.Brands.unique())
brands_sept
len(brands_sept)
```

Out[32]: 854

In [33]:
```
# THE BRANDS NEED TO BE THE SAME IN BOTH THE TEST SET AND TRAIN SET OR ELSE WE WOULD US
# BRANDS THAT ARE NOT IN SEPTEMBER AND VICE VERSA
for brand in brands_nosept:
  if brand not in brands_sept:
      data2021_nosept = data2021_nosept.drop(data2021_nosept.index[data2021_nosept['Bra

for brand in brands_sept:
  if brand not in brands_nosept:
      data2021_sept = data2021_sept.drop(data2021_sept.index[data2021_sept['Brands'] ==


data2021_nosept
```

Out[33]:

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	Total Sales ($)	
19	101 Cannabis Co.	250.2320	-0.259582	337.9605	644.694333	1	2021	1	8059.176	32.206
20	101 Cannabis Co.	395.8241	0.581828	250.2320	345.117667	2	2021	1	13712.770	34.643
21	101 Cannabis Co.	686.8574	0.735259	395.8241	328.005533	3	2021	2	24347.900	35.448
22	101 Cannabis Co.	624.6255	-0.090604	686.8574	444.304500	4	2021	2	20784.920	33.275
23	101 Cannabis Co.	345.7618	-0.446449	624.6255	569.102333	5	2021	2	12116.960	35.044
...	...	...	...	...	...	...	...	...	...	

	Brands	Total Units	Units vs. Prior Period	Previous Month	Rolling Average	Month	Year	Season	Total Sales ($)	A
27626	Zig Zag	8201.6210	0.031177	7953.6470	7708.604333	8	2021	3	22290.440	2.717
27631	Zips Weed Co.	263.8763	1.000000	0.0000	2710.397567	5	2021	2	4749.104	17.997
27632	Zips Weed Co.	401.8849	0.523005	263.8763	376.986333	6	2021	3	7471.544	18.591
27633	Zips Weed Co.	600.1628	0.493370	401.8849	221.920400	7	2021	3	10964.800	18.269
27634	Zips Weed Co.	193.7782	-0.677124	600.1628	421.974667	8	2021	3	3477.729	17.946

5861 rows × 19 columns

In [34]:
```
list(data2021_nosept.Brands.unique()) == list(data2021_sept.Brands.unique())
```

Out[34]:
```
True
```

In [35]:
```
# NEW PIPELINE TO DROP THE MONTH, YEAR, SEASON COLUMNS SINCE WE DONT WANT TO USE DIFFER
# TO PREDICT ONLY SEPTEMBER 2021
data_num = data_X.drop(["Brands", "Month", "Year", "Season", "Predominant Product Type"
numerical_features = list(data_num)
categorical_features = ["Brands", "Predominant Product Type", "Predominant Flavor"]

new_pipeline = ColumnTransformer([
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
```

In [36]:
```
data2021_nosept = data2021_nosept.drop(columns=["Year", "Month", "Season"])
data2021_sept = data2021_sept.drop(columns=["Year", "Month", "Season"])

data2021_nosept_X = data2021_nosept.drop(columns=["Total Sales ($)"])
data2021_nosept_y = data2021_nosept["Total Sales ($)"].copy()

data2021_sept_X = data2021_sept.drop(columns=["Total Sales ($)"])
data2021_sept_y = data2021_sept["Total Sales ($)"].copy()

prepared2021 = new_pipeline.fit_transform(data2021_nosept_X)
preparedsept = new_pipeline.fit_transform(data2021_sept_X)

lin_reg = LinearRegression()
lin_reg.fit(prepared2021, data2021_nosept_y)
```

```
preds = lin_reg.predict(preparedsept)
regression_results(data2021_sept_y, preds)
```

```
explained_variance:  0.9673
r2:  0.9622
MAE:  110818.9963
MSE:  79616110226.266
RMSE:  282163.2687
```

In [37]:

```
tree = RandomForestRegressor(max_depth=100, n_estimators=200, random_state=42)

tree.fit(prepared2021, data2021_nosept_y)

preds_test = tree.predict(preparedsept)

regression_results(preds_test, data2021_sept_y)
```

```
explained_variance:  0.9749
r2:  0.9664
MAE:  153118.6856
MSE:  90841911243.1655
RMSE:  301399.9191
```

In [39]:

```
# try Lasso regularization on data
from sklearn.linear_model import Lasso

l = Lasso(tol=0.001, max_iter=10000)
l.fit(X_train, y_train)

preds = l.predict(X_test)
regression_results(y_test, preds)

# NO POINT TO OPTIMIZE PARAMETERS, IT IS CLEARLY WORSE THAN RANDOM FOREST
```

```
explained_variance:  0.947
r2:  0.947
MAE:  105946.7952
MSE:  125625588689.2336
RMSE:  354437.0024
```

In [38]:

1.11 Experiment with Custom Models 10 / 10

✓ **- 0 pts** **Correct**

- 10 pts missing

Analysis to determine Primary Factors for Higher Sales Volume and Forecast Revenue of Cannabis Distributor *Cookies*

December 4, 2021

Aditya Mishra

UID: 405411117

Brendan Rossmango

UID: 505370692

Table of Contents

Executive Summary:

The aim of this project was to look at the dataset from one of the largest and fastest growing Cannabis Brands in the world, Cookies. Our goal was to develop a predictive model to help forecast sales and to conduct an analysis determining the key factors impacting the success of a product and based on that data propose potential growth areas to the company. We use time-series features and brand level detailed features to predict the total sales of a brand per each month.

Our analysis began by performing data wrangling by cleaning our data, merging our datasets and effectively linking information. For instance, to make our data more usable we converted the sales revenue and units sold to numerical data. To prepare for merging data, we renamed some features. We merged sales data, units data and Average Retail Price (ARP) data.

We then ran some basic statistics. First, we plotted a correlation matrix and found that besides time series features, there were no visibly strong relationships between any two sets of data. This was not ideal as we didn't get a better idea of potential collinearity in our dataset.

We used a single pipeline to implement this transformation. In our initial experiments we noticed the inclusion of the Brands data field made the model perform better (our Mean Absolute Error (MAE) decreased by a factor of 2) indicating an association between Total Sales and brands, so we decided to keep all 1640 brands. However, this increased the complexity of our model. We included the rest of the data because we believed it would all be beneficial to model performance. We encoded the categorical features Month, Year, Season, Predominant Type, Predominant Flavor. We also had lots of null data that we resolved using imputation. For example, there were many null fields in our 'previous period' columns which we resolved by filling the nulls with ones. We also had many null fields in our previous month, total sales ($) and ARP fields, null values in these fields meant that either it was the first time the brand was being sold or that there were no sales in that timeframe, so, we filled these null values with 1 or 0 accordingly. Prior to the pipeline, we augmented many time-series features, such as Rolling Average, and brand level features, such as predominant flavor and predominant product type. In the pipeline, we augmented a feature, flavored_products (number of flavors * product count).

We then performed basic regression analysis, principal component analysis (PCA) and used a Random Forest. PCA performed the worst out of these three models because of the sparseness of the dataset with an average error of $200,000. Regression worked better as the average error was $100,000 - twice better than PCA. However, the Random Forest method performed the best by far. Random Forest gave us an average error of $16,000, more than 5 times better than our next best method.

We then performed a KFold cross validation on both our OLS regression and random forest models to estimate the skill of the respective models on "new" data to get a more accurate representation of our models' results. The Random Forest had a MAE nearly 10 times lower than OLS Regression; the lower MAE indicates we can expect a lower error on average from Random forest than our regression model on new data.

We performed grid search to optimize our parameters and found our Random Forest Model performed the best (Average Error of $14,846) with a depth of 50 and 150 estimators.

In conclusion, the ensemble method of a random forest was most useful to help forecast sales and from our regression analysis, the key factors impacting the success of a product are the months it was sold in (specifically, the summer months are better to sell a product), the product's brand and whether or not a product is an inhalable.

2.1 Executive Summary 5 / 5

✓ - **0 pts** **Correct**

- **2.5 pts** not enough explanation

- **5 pts** missing

Background/Introduction:

In 2016, voters in California approved the legalization of Marijuana. In the last five years since its legalization, the Cannabis industry has had incredible growth. In 2018, cannabis market size reached 10 billion dollars, however, this number is expected to reach 97.35 billion by 2026. There has been a compounded growth rate of 26.7%. However, due to the massive financial incentive, there are many companies competing in this field. There are now over 30,000 businesses in the marijuana industry and companies are now using data science to determine how to best serve their customers and differentiate from their competitors.

For this project we were given a dataset from one of the largest and fastest growing Cannabis Brands in the world, Cookies. Cookies is known for their massive, high-quality variety of cannabis products - ranging from indoor grown, outdoor grown, sungrown flower, pre-rolls, gel caps and more. However, producing and storing such a large variety of products comes with a cost. So, it is necessary to extract information about the highest selling products to determine which products are a more valuable investment. As a result, our goal was to develop a predictive model to help forecast sales and to conduct an analysis determining the key factors impacting the success of a product and based on that data propose potential growth areas to the company.

Methodology:

We began by developing a basic time series feature extraction plan. To do this we turned the Months feature from each of the data files we received into Datetime objects to more easily extract month and year information.

```
units['Months'] = pd.to_datetime(units['Months'])
sales['Months'] = pd.to_datetime(sales['Months'])
arp['Months'] = pd.to_datetime(arp['Months'])
```

We performed time series feature engineering and augmented pre-existing features to better serve our model. We created a previous month and rolling average feature that helped show relationships in our correlation matrix. These features were useful to show the average change over time as the product continued to be on the market (i.e, if it just had a lot of hype when it was put on the market or if it consistently became more popular - indicating a better product).

```
for brand in all_brands:
  units_data = units[units.Brands == brand]
  # PREVIOUS MONTH: THE TOTAL UNITS OF THE PREVIOUS MONTH
  units_data.loc[:,'Previous Month'] = units_data.loc[:,'Total Units'].shift(1)
  # ROLLING AVERAGE: THE AVERAGE TOTAL UNITS OF THE PAST 3 MONTHS
  units_data.loc[:,'Rolling Average'] = (units_data.loc[:,'Total Units'].shift(3) + units_data.loc[:,'Total Units'].shift(2) + units_data.loc[:,'Total Units'].shift(1))/3

  first = units_data.first_valid_index()
  # IF THERE ARE NO, ONE, OR TWO PRIOR MONTHS, MANUALLY CALCULATE THE ROLLING AVERAGE
  units_data.loc[first,['Rolling Average']] = 0
  if units_data.shape[0] > 2:
    units_data.loc[first+2,'Rolling Average'] = (units_data.loc[first+1,'Total Units'] + units_data.loc[first, "Total Units"]) / 2
  if units_data.shape[0] > 1:
    units_data.loc[first+1,'Rolling Average'] = units_data.loc[first, "Total Units"]

  # YEAR: GET THE YEAR FROM THE MONTHS COLUMN
  units_data["Month"] = pd.DatetimeIndex(units_data["Months"]).month
  units_data['Year'] = pd.DatetimeIndex(units_data['Months']).year
  # SEASON: WINTER IS 1, SPRING IS 2, SUMMER IS 3, FALL IS 4
  units_data["Season"] = pd.DatetimeIndex(units_data['Months']).month%12 // 3 + 1
```

Our general data strategy depended on our initial experiments and the dependent variable of our model. We plotted a correlation matrix and noticed that time series data seemed to have a

2.2 Background/Info 10 / 10

✓ **- 0 pts** **Correct**

- 5 pts not enough detailed information

- 10 pts missing

Background/Introduction:

In 2016, voters in California approved the legalization of Marijuana. In the last five years since its legalization, the Cannabis industry has had incredible growth. In 2018, cannabis market size reached 10 billion dollars, however, this number is expected to reach 97.35 billion by 2026. There has been a compounded growth rate of 26.7%. However, due to the massive financial incentive, there are many companies competing in this field. There are now over 30,000 businesses in the marijuana industry and companies are now using data science to determine how to best serve their customers and differentiate from their competitors.

For this project we were given a dataset from one of the largest and fastest growing Cannabis Brands in the world, Cookies. Cookies is known for their massive, high-quality variety of cannabis products - ranging from indoor grown, outdoor grown, sungrown flower, pre-rolls, gel caps and more. However, producing and storing such a large variety of products comes with a cost. So, it is necessary to extract information about the highest selling products to determine which products are a more valuable investment. As a result, our goal was to develop a predictive model to help forecast sales and to conduct an analysis determining the key factors impacting the success of a product and based on that data propose potential growth areas to the company.

Methodology:

We began by developing a basic time series feature extraction plan. To do this we turned the Months feature from each of the data files we received into Datetime objects to more easily extract month and year information.

```
units['Months'] = pd.to_datetime(units['Months'])
sales['Months'] = pd.to_datetime(sales['Months'])
arp['Months'] = pd.to_datetime(arp['Months'])
```

We performed time series feature engineering and augmented pre-existing features to better serve our model. We created a previous month and rolling average feature that helped show relationships in our correlation matrix. These features were useful to show the average change over time as the product continued to be on the market (i.e, if it just had a lot of hype when it was put on the market or if it consistently became more popular - indicating a better product).

```
for brand in all_brands:
  units_data = units[units.Brands == brand]
  # PREVIOUS MONTH: THE TOTAL UNITS OF THE PREVIOUS MONTH
  units_data.loc[:,'Previous Month'] = units_data.loc[:,'Total Units'].shift(1)
  # ROLLING AVERAGE: THE AVERAGE TOTAL UNITS OF THE PAST 3 MONTHS
  units_data.loc[:,'Rolling Average'] = (units_data.loc[:,'Total Units'].shift(3) + units_data.loc[:,'Total Units'].shift(2) + units_data.loc[:,'Total Units'].shift(1))/3

  first = units_data.first_valid_index()
  # IF THERE ARE NO, ONE, OR TWO PRIOR MONTHS, MANUALLY CALCULATE THE ROLLING AVERAGE
  units_data.loc[first,['Rolling Average']] = 0
  if units_data.shape[0] > 2:
    units_data.loc[first+2,'Rolling Average'] = (units_data.loc[first+1,'Total Units'] + units_data.loc[first, "Total Units"]) / 2
  if units_data.shape[0] > 1:
    units_data.loc[first+1,'Rolling Average'] = units_data.loc[first, "Total Units"]

  # YEAR: GET THE YEAR FROM THE MONTHS COLUMN
  units_data["Month"] = pd.DatetimeIndex(units_data["Months"]).month
  units_data['Year'] = pd.DatetimeIndex(units_data['Months']).year
  # SEASON: WINTER IS 1, SPRING IS 2, SUMMER IS 3, FALL IS 4
  units_data["Season"] = pd.DatetimeIndex(units_data['Months']).month%12 // 3 + 1
```

Our general data strategy depended on our initial experiments and the dependent variable of our model. We plotted a correlation matrix and noticed that time series data seemed to have a

high degree of correlation with total sales ($), this indicated a higher level of association that warranted further investigation. In our initial experiments we noticed the exclusion of the brands data field made our first iterations of our model perform significantly worse so we decided against dropping it despite having the 1640 unique brands increasing the complexity of our model. Later on when we created our own custom model being trained using 2021 data and testing against only September 2021. We dropped the month, year and season columns to predict total sales, because we only wanted to predict one month's sales.

```
# EXPERIMENT: USE SEPT 2021 AS TEST SET, 2021 AS TRAINING SET
# GET ALL BRANDS FOR 2021, FILTER OUT BRANDS THAT ARE NOT IN SEPT 2021 AND THEN SPLIT

data2021 = data[data.Year == 2021]
data2021_nosept = data2021[data.Month < 9]
brands_nosept = list(data2021_nosept.Brands.unique())
brands_nosept
len(brands_nosept)

# NEW PIPELINE TO DROP THE MONTH, YEAR, SEASON COLUMNS SINCE WE DONT WANT TO USE DIFFER
# TO PREDICT ONLY SEPTEMBER 2021
data_num = data_X.drop(["Brands", "Month", "Year", "Season", "Predominant Product Type"
numerical_features = list(data_num)
categorical_features = ["Brands", "Predominant Product Type", "Predominant Flavor"]

new_pipeline = ColumnTransformer([
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
```

We also implemented a random forest, but we did not remove or include any additional data from our model. However, the large amount of data we had increased our complexity and caused our model to take nearly 50 minutes to complete, although the model was by far more accurate.

Results:

In order to find a better idea of what data to focus on in our results, we ran some basic statistics. First, we plotted a correlation matrix and found that besides time series features, there were no visibly strong relationships between any two sets of data. This was not ideal as we didn't get a better idea of potential collinearity in our dataset.

We implemented a basic Linear Regression predictive model, Ordinary Least Squares regression with Total Sales ($) as our dependent variable. We obtained an R^2 value of 0.874 and because this is greater than 0.85, this indicates a strong positive correlation within our data. The regression models indicate that the month in which a product was sold, the product's brand strength, and whether or not a product is inhalable are predictors of Total Sales ($). For the months, it is best to sell a product from April - June (end of spring, beginning of summer).

2.3 Methodology **15 / 15**

✓ **- 0 pts** **Correct**

- 5 pts not enough explanation

- 15 pts missing

high degree of correlation with total sales ($), this indicated a higher level of association that warranted further investigation. In our initial experiments we noticed the exclusion of the brands data field made our first iterations of our model perform significantly worse so we decided against dropping it despite having the 1640 unique brands increasing the complexity of our model. Later on when we created our own custom model being trained using 2021 data and testing against only September 2021. We dropped the month, year and season columns to predict total sales, because we only wanted to predict one month's sales.

```
# EXPERIMENT: USE SEPT 2021 AS TEST SET, 2021 AS TRAINING SET
# GET ALL BRANDS FOR 2021, FILTER OUT BRANDS THAT ARE NOT IN SEPT 2021 AND THEN SPLIT

data2021 = data[data.Year == 2021]
data2021_nosept = data2021[data.Month < 9]
brands_nosept = list(data2021_nosept.Brands.unique())
brands_nosept
len(brands_nosept)

# NEW PIPELINE TO DROP THE MONTH, YEAR, SEASON COLUMNS SINCE WE DONT WANT TO USE DIFFER
# TO PREDICT ONLY SEPTEMBER 2021
data_num = data_X.drop(["Brands", "Month", "Year", "Season", "Predominant Product Type"
numerical_features = list(data_num)
categorical_features = ["Brands", "Predominant Product Type", "Predominant Flavor"]

new_pipeline = ColumnTransformer([
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
```

We also implemented a random forest, but we did not remove or include any additional data from our model. However, the large amount of data we had increased our complexity and caused our model to take nearly 50 minutes to complete, although the model was by far more accurate.

Results:

In order to find a better idea of what data to focus on in our results, we ran some basic statistics. First, we plotted a correlation matrix and found that besides time series features, there were no visibly strong relationships between any two sets of data. This was not ideal as we didn't get a better idea of potential collinearity in our dataset.

We implemented a basic Linear Regression predictive model, Ordinary Least Squares regression with Total Sales ($) as our dependent variable. We obtained an R^2 value of 0.874 and because this is greater than 0.85, this indicates a strong positive correlation within our data. The regression models indicate that the month in which a product was sold, the product's brand strength, and whether or not a product is inhalable are predictors of Total Sales ($). For the months, it is best to sell a product from April - June (end of spring, beginning of summer).

```
trimmed_data_prepared = full_pipeline.fit_transform(data_X)

avo_stats = sm.OLS(data_y, trimmed_data_prepared.toarray())

results_stats = avo_stats.fit()
print(results_stats.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Total Sales ($)   R-squared:                       0.874
Model:                            OLS   Adj. R-squared:                  0.873
Method:                 Least Squares   F-statistic:                     1905.
Date:                Thu, 02 Dec 2021   Prob (F-statistic):               0.00
Time:                        19:45:10   Log-Likelihood:            -4.0492e+05
No. Observations:               27686   AIC:                         8.100e+05
Df Residuals:                   27585   BIC:                         8.109e+05
Df Model:                         100
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1          2.601e+05   1.45e+04     17.888      0.000    2.32e+05    2.89e+05
x2          3543.7790   3294.737      1.076      0.282   -2914.070       1e+04
x3          7.535e+05   4.36e+04     17.301      0.000    6.68e+05    8.39e+05

                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1          6561.9237   3309.441      1.983      0.047      75.253     1.3e+04
x2          9.324e+05   4.26e+04     21.872      0.000    8.49e+05    1.02e+06
x3          2.951e+05   4.26e+04      6.929      0.000    2.12e+05    3.79e+05
x4          3.887e+04   3711.604     10.473      0.000    3.16e+04    4.61e+04
x5         -1.386e+04   3501.140     -3.959      0.000   -2.07e+04   -6999.453
x6          5.815e+04   1.04e+04      5.603      0.000    3.78e+04    7.85e+04
const      -1.482e-12   2.76e-10     -0.005      0.996   -5.43e-10     5.4e-10
x7           5.12e+05   3714.578    137.847      0.000    5.05e+05    5.19e+05
x8          6.456e+04   4006.401     16.114      0.000    5.67e+04    7.24e+04
x9          8.172e+04   5212.462     15.678      0.000    7.15e+04    9.19e+04
x10        -1.598e+04   3880.856     -4.118      0.000   -2.36e+04   -8375.557
x11        -5507.7088   1.05e+04     -0.523      0.601   -2.61e+04    1.51e+04
x12        -1.318e+04   1.05e+04     -1.258      0.209   -3.37e+04    7360.923
x13         2.422e+04   1.02e+04      2.367      0.018    4161.701    4.43e+04
x14        -9986.7240   1.02e+04     -0.981      0.327   -2.99e+04    9964.427
x15         7592.6849      1e+04      0.757      0.449   -1.21e+04    2.72e+04
x16        -3105.8969   9858.646     -0.315      0.753   -2.24e+04    1.62e+04
x17         2.593e+04   9826.131      2.638      0.008    6665.326    4.52e+04
x18        -9349.1159   9352.072     -1.000      0.317   -2.77e+04    8981.412
x19         5008.9118   9494.231      0.528      0.598   -1.36e+04    2.36e+04
x20         4939.6713   1.05e+04      0.471      0.638   -1.56e+04    2.55e+04
x21        -3899.1580   1.04e+04     -0.375      0.708   -2.43e+04    1.65e+04
x22         1.867e+04   1.07e+04      1.738      0.082   -2382.263    3.97e+04
x23        -2209.5910   1.46e+04     -0.152      0.880   -3.08e+04    2.64e+04
```

In these statistics results, the first 3 rows are time-series features, the next 12 (x4 - x15) rows are the months, the next 4 rows (x16 - x19) are the years (2018-2021), and the next 4 rows (x20 - x23) are the seasons (winter, spring, summer, fall). The months with the highest T-scores are April, May, June, and the season with the highest T-score is summer.

We then did a simple train test split with the regression model and got the summary statistics. We have a high explained variance which indicates a strong level of association and additionally, we had a R^2 value of 0.947, which is incredibly high. However, we also had a MAE of more than $100,000, which is not ideal.

```
# SIMPLE TRAIN TEST SPLIT ON PIPELINE DATA + BASIC LINEAR REGRESSION MODEL
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_prepared, data_y, test_size=.25, random_state=42)

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

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

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

```
explained_variance:  0.9427
r2:  0.9427
MAE:  117211.0687
MSE:  135734309451.6578
RMSE:  368421.3749
```

We also performed Principal Component Analysis however, this did not give us valuable information due to the sparseness of the dataset. The explained variance rose however the MAE rose to over 200,000. We were unable to get additional useful relationships using PCA.

```
# DO PCA
from sklearn.decomposition import PCA, TruncatedSVD
pca = TruncatedSVD(n_components=30) # sparse data, PCA does not work

data_pca = pca.fit_transform(data_prepared)
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(data_pca, data_y, test_size=0.25, random_state=42)

# LINEAR REGRESSION AGAIN WITH NEW PIPELINED DATA
lin_reg = LinearRegression()
lin_reg.fit(new_X_train, new_y_train)

preds = lin_reg.predict(new_X_test)
regression_results(new_y_test, preds)
```

```
explained_variance:  0.947
r2:  0.947
MAE:  106510.7698
MSE:  125684627468.7391
RMSE:  354520.2779
```

We implemented an ensemble method, a random forest. This performed the best by far out of all of our methods with an explained variance of 0.992 and a MAE of 16733. This high performance is likely due to the ensemble nature of the random forest method; we combined a set of weak learners that use random subsets of the data to create a strong learner.

```
# ENSEMBLE METHOD: RANDOM FOREST
from sklearn.ensemble import RandomForestRegressor

tree = RandomForestRegressor(min_samples_leaf=5, max_depth=100, random_state=42)

tree.fit(X_train, y_train)

preds_test = tree.predict(X_test)

regression_results(preds_test, y_test)
```

```
explained_variance:  0.992
r2:  0.992
MAE:  16733.0223
MSE:  18721233198.4262
RMSE:  136825.5575
```

We also performed a KFold cross validation on both our OLS regression and random forest models to estimate the skill of the respective models on "new" data. We discovered that both performed well (Linear Regression R2: 0.9405, Random Forest R2: 0.9921) however, Random forest had a MAE nearly 10 times lower than OLS Regression; the lower MAE indicates we can expect a lower error on average from Random forest than our regression model.

```
# CROSS VALIDATE BOTH SIMPLE REGRESSION AND RANDOM FOREST
from sklearn.model_selection import  KFold
from sklearn import model_selection
from sklearn.model_selection import cross_validate, GridSearchCV, cross_val_score
import sklearn

kfold = model_selection.KFold(n_splits=5, random_state=42, shuffle=True)
clf = LinearRegression()
scores = cross_validate(clf, data_prepared, data_y, scoring=('r2', 'neg_mean_absolute_error', 'explained_variance'), cv=kfold)

# GET MAE BY JUST MULTILPYING NMAE BY -1
print("Linear Regression MAE: " + str(( round(scores['test_neg_mean_absolute_error'].mean() * -1, 4) )))
print("Linear Regression R2: " + str(( round(scores['test_r2'].mean(), 4) )))
print("Linear Regression Explained Variance: " + str(( round(scores['test_explained_variance'].mean(), 4) )))

clf = RandomForestRegressor(max_depth=50, n_estimators=75, random_state=42)

scores = cross_validate(clf, data_prepared, data_y, scoring=('r2', 'neg_mean_absolute_error', 'explained_variance'), cv=kfold)

print("Random Forest MAE: " + str(( round(scores['test_neg_mean_absolute_error'].mean() * -1, 4) )))
print("Random Forest R2: " + str(( round(scores['test_r2'].mean(), 4) )))
print("Random Forest Explained Variance: " + str(( round(scores['test_explained_variance'].mean(), 4) )))
```

```
Linear Regression MAE: 103071.2712
Linear Regression R2: 0.9405
Linear Regression Explained Variance: 0.9405
Random Forest MAE: 13530.9548
Random Forest R2: 0.9921
Random Forest Explained Variance: 0.9921
```

We performed grid search to optimize our parameters for our Random Forest model and found our Model performed the best (Average Error of $14,846) with a depth of 50 and 150 decision trees.

```
# GRID SEARCH: THIS CELL ACTUALLY TAKES LIKE 50 MINUTES
# OPTIMIZE max_depth AND n_estimators
parameters = {'max_depth':[25, 50], 'n_estimators':[25, 50, 75, 150]}
rf = RandomForestRegressor(random_state=42)
kfold = model_selection.KFold(n_splits=3, random_state=42, shuffle=True)
clf = GridSearchCV(rf, parameters, scoring='neg_mean_absolute_error', cv=kfold, verbose=3)
clf.fit(data_prepared, data_y)
```

```
Best estimator: RandomForestRegressor(max_depth=50, n_estimators=150, random_state=42)
Best score: 14846.6484
```

We further experimented with our custom models using a different pipeline. We attempted to predict the revenue of the year using just the data from the month of September as the training set. Using the same model, we wanted to see if we could get an accurate prediction without the use of month, year and season columns. However, none of them performed as well as our random forest. Something worth noting is that in our new model, linear regression seemed to give a lower degree of error than Random Forest. This may be because we had less data, so our Random Forest wasn't able to "average out" in the long run.

```
new_pipeline = ColumnTransformer([
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(handle_unknown="ignore", categories="auto"), categorical_feat
])
```

```
data2021_nosept = data2021_nosept.drop(columns=["Year", "Month", "Season"])
data2021_sept = data2021_sept.drop(columns=["Year", "Month", "Season"])

data2021_nosept_X = data2021_nosept.drop(columns=["Total Sales ($)"])
data2021_nosept_y = data2021_nosept["Total Sales ($)"].copy()

data2021_sept_X = data2021_sept.drop(columns=["Total Sales ($)"])
data2021_sept_y = data2021_sept["Total Sales ($)"].copy()

prepared2021 = new_pipeline.fit_transform(data2021_nosept_X)
preparedsept = new_pipeline.fit_transform(data2021_sept_X)

lin_reg = LinearRegression()
lin_reg.fit(prepared2021, data2021_nosept_y)
```

```
preds = lin_reg.predict(preparedsept)
regression_results(data2021_sept_y, preds)
```

```
explained_variance:  0.9673
r2:  0.9622
MAE:  110818.9963
MSE:  79616110226.266
RMSE:  282163.2687
```

```
tree = RandomForestRegressor(max_depth=100, n_estimators=200, random_state=42)

tree.fit(prepared2021, data2021_nosept_y)

preds_test = tree.predict(preparedsept)

regression_results(preds_test, data2021_sept_y)
```

```
explained_variance:  0.9749
r2:  0.9664
MAE:  153118.6856
MSE:  90841911243.1655
RMSE:  301399.9191
```

In conclusion, the ensemble method of a random forest was most useful to help forecast sales. This can be demonstrated by the summary statistics after performing KFold cross validation (MAE: $13,530; R^2: 0.9921). From our regression analysis, we can extrapolate the key factors impacting the success of a product which are the months it was sold in, the product's brand and whether or not a product is inhalable.

Discussion:

From our results we determined the best predict model to forecast product sales is a Random Forest Regression model. On new data, this model has an R^2 value of 0.9921 indicating an extremely strong positive association between Total Sales ($) and the data fields in our model. Additionally, the MAE from this model is $13,530 which is nearly 10 times lower than the next best performing model. Cookies ought to use this model to better gauge which features better indicate a higher volume of sales. However, this model is complex and takes an extremely long time to run. From our regression model, we determined that the primary factors affecting the model are months it was sold in, the product's brand strength, and whether or not a product is inhalable.

From our results, we noticed that the inclusion of the cannabis' brand in our model reduced our average error by a factor of 2. This difference indicates some level of association

2.4 Results 15 / 15

✓ **- 0 pts** **Correct**

- 5 pts no detailed report

- 15 pts missing

```
preds = lin_reg.predict(preparedsept)
regression_results(data2021_sept_y, preds)
```

```
explained_variance:  0.9673
r2:  0.9622
MAE:  110818.9963
MSE:  79616110226.266
RMSE:  282163.2687
```

```
tree = RandomForestRegressor(max_depth=100, n_estimators=200, random_state=42)

tree.fit(prepared2021, data2021_nosept_y)

preds_test = tree.predict(preparedsept)

regression_results(preds_test, data2021_sept_y)
```

```
explained_variance:  0.9749
r2:  0.9664
MAE:  153118.6856
MSE:  90841911243.1655
RMSE:  301399.9191
```

In conclusion, the ensemble method of a random forest was most useful to help forecast sales. This can be demonstrated by the summary statistics after performing KFold cross validation (MAE: $13,530; R^2: 0.9921). From our regression analysis, we can extrapolate the key factors impacting the success of a product which are the months it was sold in, the product's brand and whether or not a product is inhalable.

Discussion:

From our results we determined the best predict model to forecast product sales is a Random Forest Regression model. On new data, this model has an R^2 value of 0.9921 indicating an extremely strong positive association between Total Sales ($) and the data fields in our model. Additionally, the MAE from this model is $13,530 which is nearly 10 times lower than the next best performing model. Cookies ought to use this model to better gauge which features better indicate a higher volume of sales. However, this model is complex and takes an extremely long time to run. From our regression model, we determined that the primary factors affecting the model are months it was sold in, the product's brand strength, and whether or not a product is inhalable.

From our results, we noticed that the inclusion of the cannabis' brand in our model reduced our average error by a factor of 2. This difference indicates some level of association

between brand and total sales. This significant difference indicates further market research should be done to determine which brands draw the most purchasers. Additionally, the company should likely advertise more in months in which it is not selling as well because of the apparent association of month and total sales; we know now that it is best to launch/sell a product from April-June, but in general, over the summer. Some next steps would be to perform regularization to increase performance (however, when we did this it decreased the accuracy of our model) and to systematically remove features from the model to determine which features have the greatest effect on our test accuracy. Additionally, for dimensionality reduction techniques like PCA to perform better, more data needs to be collected about the brands and their sales. There were limited amounts of data for each brand which is not ideal considering the high impact of that feature on our model.

Conclusion:

The aim of this project was to develop a predictive model to help forecast sales and to conduct an analysis determining the key factors impacting the success of a product and based on that data, propose potential growth areas to the company. We applied different models to determine which one could best forecast sales and determined the Random Forest Regression model was best able to forecast sales. Additionally, we learned the key factors impacting success of a product are its brand, the month distributors choose to sell it in (specifically, the summer months are better for a product launch) and the ability to inhale the product. However, due to the sparseness of the data, our results were not incredibly accurate. As a result, Cookies should continue collecting more data and perhaps do additional analysis to determine if they can in fact create a model that can better forecast sales.

2.5 Discussion **10 / 10**

✓ **- 0 pts** **Correct**

- 3 pts Click here to replace this description.

- 10 pts missing

between brand and total sales. This significant difference indicates further market research should be done to determine which brands draw the most purchasers. Additionally, the company should likely advertise more in months in which it is not selling as well because of the apparent association of month and total sales; we know now that it is best to launch/sell a product from April-June, but in general, over the summer. Some next steps would be to perform regularization to increase performance (however, when we did this it decreased the accuracy of our model) and to systematically remove features from the model to determine which features have the greatest effect on our test accuracy. Additionally, for dimensionality reduction techniques like PCA to perform better, more data needs to be collected about the brands and their sales. There were limited amounts of data for each brand which is not ideal considering the high impact of that feature on our model.

Conclusion:

The aim of this project was to develop a predictive model to help forecast sales and to conduct an analysis determining the key factors impacting the success of a product and based on that data, propose potential growth areas to the company. We applied different models to determine which one could best forecast sales and determined the Random Forest Regression model was best able to forecast sales. Additionally, we learned the key factors impacting success of a product are its brand, the month distributors choose to sell it in (specifically, the summer months are better for a product launch) and the ability to inhale the product. However, due to the sparseness of the data, our results were not incredibly accurate. As a result, Cookies should continue collecting more data and perhaps do additional analysis to determine if they can in fact create a model that can better forecast sales.

2.6 Conclusion 5 / 5

✓ **- 0 pts** **Correct**

- 2 pts Click here to replace this description.

- 5 pts missing