Problem Definition :

Avocado is a fruit consumed by people heavily in the United States.

Content

This data was downloaded from the Hass Avocado Board website in May of 2018 & compiled into a single CSV. Here's how the Hass Avocado Board describes the data on their website:

The table below represents weekly 2018 retail scan data for National retail volume (units) and price. Retail scan data comes directly from retailers' cash registers based on actual retail sales of Hass avocados. Starting in 2013, the table below reflects an expanded, multi-outlet retail data set. Multi-outlet reporting includes an aggregation of the following channels: grocery, mass, club, drug, dollar and military. The Average Price (of avocados) in the table reflects a per unit (per avocado) cost, even when multiple units (avocados) are sold in bags. The Product Lookup codes (PLU's) in the table are only for Hass avocados. Other varieties of avocados (e.g. greenskins) are not included in this table.

Some relevant columns in the dataset:

- Date - The date of the observation
- AveragePrice - the average price of a single avocado
- type - conventional or organic
- year - the year
- Region - the city or region of the observation
- Total Volume - Total number of avocados sold
- 4046 - Total number of avocados with PLU 4046 sold
- 4225 - Total number of avocados with PLU 4225 sold
- 4770 - Total number of avocados with PLU 4770 sold

Inspiration /Label

The dataset can be seen in one angle to find the average price .

Task: Regression

Exploratory Analysis :

Importing All necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading the data :

```
df=pd.read_csv("avocado.csv")
```

```
df.head(10)
```

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
0	0.0	27-12-2015	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015.0	Albany
1	1.0	20-12-2015	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015.0	Albany
2	2.0	13-12-2015	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015.0	Albany
3	3.0	06-12-2015	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015.0	Albany
4	4.0	29-11-2015	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015.0	Albany
5	5.0	22-11-2015	1.26	55979.78	1184.27	48067.99	43.61	6683.91	6556.47	127.44	0.0	conventional	2015.0	Albany
6	6.0	15-11-2015	0.99	83453.76	1368.92	73672.72	93.26	8318.86	8196.81	122.05	0.0	conventional	2015.0	Albany
7	7.0	08-11-2015	0.98	109428.33	703.75	101815.36	80.00	6829.22	6266.85	562.37	0.0	conventional	2015.0	Albany
8	8.0	01-11-2015	1.02	99811.42	1022.15	87315.57	85.34	11388.36	11104.53	283.83	0.0	conventional	2015.0	Albany
9	9.0	25-10-2015	1.07	74338.76	842.40	64757.44	113.00	8625.92	8061.47	564.45	0.0	conventional	2015.0	Albany

"read_csv" is an important function of pandas which allows to read csv files and we can make various operations on the dataset. As my file is a CSV file that's why I have used "read_csv" function to load the data from the specific directory.The name of the dataset id df.

Description of Data :

Rename Unnamed column to another name :

```
avocado.rename(columns={'Unnamed: 0':'ID'},inplace=True)
```

In the dataset, there is one column names "Unnamed: 0" . I have changed the column name to "ID".

```
df.info()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16468 entries, 0 to 16467
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    1517 non-null   float64
 1   Date          1517 non-null   object
 2   AveragePrice  1517 non-null   float64
 3   Total Volume  1517 non-null   float64
 4   4046          1517 non-null   float64
 5   4225          1517 non-null   float64
 6   4770          1517 non-null   float64
 7   Total Bags    1517 non-null   float64
 8   Small Bags    1517 non-null   float64
 9   Large Bags    1517 non-null   float64
 10  XLarge Bags   1517 non-null   float64
 11  type          1517 non-null   object
 12  year          1517 non-null   float64
 13  region        1517 non-null   object
dtypes: float64(11), object(3)
memory usage: 1.8+ MB
```

Normaly to explore the data we can use various functions such as shape, columns, dtypes, info(), head(), tail(), describe(). Here, I have used df.info()
By using info() we can get a concise summary of a DataFrame. It includes the index dtype and column dtypes, non-null values and memory usage.
In our dataset we can see that there are 11 Numeric columns and three object type column.
Observations:

Numeric features
Numeric features = [' Unnamed: 0', 'AveragePrice', 'Total Volume', '4046', '4225', '4770','Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 'year']

Catagorical features

Catagorical features =['type', 'region', 'Date']

Missing Values

There are different ways to check the missing values in our dataset.

```
df.isnull().values.any()
```

```
True
```

Here , we can see that there are null values present in our dataset.
But, If we want to see column wise we can use another method.

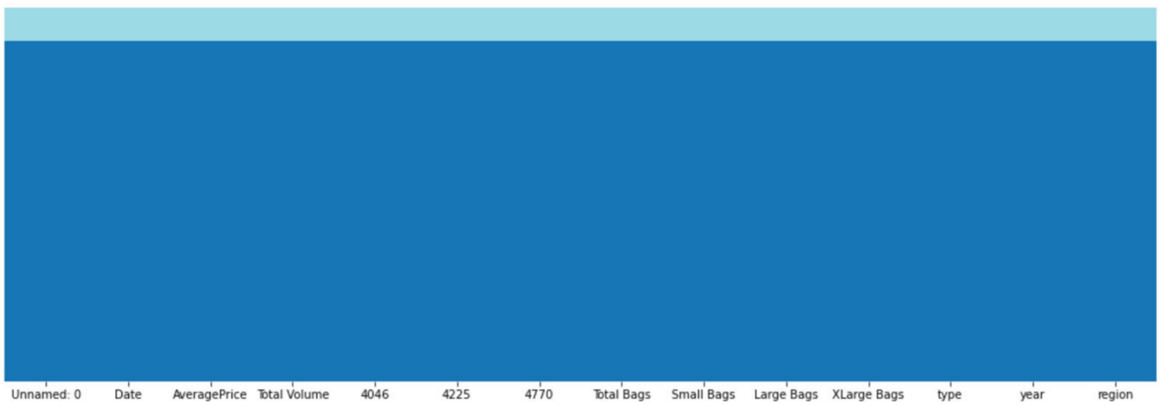
```
df.isnull().sum()
```

```
Unnamed: 0        14951
Date              14951
AveragePrice      14951
Total Volume      14951
4046              14951
4225              14951
4770              14951
Total Bags        14951
Small Bags        14951
Large Bags        14951
XLarge Bags       14951
type              14951
year              14951
region            14951
dtype: int64
```

In every colulm there are Null values in the datset.
If we want to visualizise the null values then there are another method. Here, we can see that in every column there are same number of Null Values. That means our dataset have some rows where all the elements are missing. So, we will fix the issue.

```
plt.figure(figsize=(18,6))
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='tab20_r')
```

```
<AxesSubplot:>
```



By any method we will get the same results.

Imputing Missing values:

```
df.dropna(axis=0, how='all', thresh=None, subset=None, inplace=True)
```

By using the above code I have imputed the missing values. If all the values in one row are Null then I have deleted those rows.

Exploratory the catagorical columns :

```
for column in df.columns:
    if df[column].dtypes==object:
        print(str(column)+':'+str(df[column].unique()))
        print(df[column].value_counts())
        print("**********************************************************************************************")
        print('\n')
```

```
region:['Albany' 'Atlanta' 'BaltimoreWashington' 'Boise' 'Boston'
 'BuffaloRochester' 'California' 'Charlotte' 'Chicago' 'Columbus'
 'DallasFtWorth' 'Denver' 'Detroit' 'GrandRapids' 'GreatLakes'
 'HarrisburgScranton' 'HartfordSpringfield' 'Houston' 'Indianapolis'
 'Jacksonville' 'LasVegas' 'LosAngeles' 'Louisville' 'MiamiFtLauderdale'
 'Midsouth' 'Nashville' 'NewYork' 'Northeast' 'NorthernNewEngland'
 'Orlando' 'Philadelphia' 'PhoenixTucson' 'Pittsburgh' 'Plains' 'Portland'
 'RaleighGreensboro' 'RichmondNorfolk' 'Roanoke' 'SanDiego' 'SanFrancisco'
 'Seattle' 'SouthCarolina' 'SouthCentral' 'Southeast' 'Spokane' 'StLouis'
 'Syracuse' 'Tampa' 'TotalUS' 'West' 'WestTexNewMexico']
California             76
Albany                 67
BaltimoreWashington    65
Boise                  65
Boston                 62
Atlanta                54
PhoenixTucson          52
BuffaloRochester       51
Spokane                49
Columbus               47
NewYork                44
Jacksonville           41
Detroit                40
SouthCentral           39
SanDiego               38
West                   36
Tampa                  34
Louisville             34
Charlotte              31
Portland               30
Houston                29
NorthernNewEngland     29
WestTexNewMexico       27
Nashville              25
TotalUS                25
Denver                 24
SouthCarolina          24
GrandRapids            23
Chicago                23
Pittsburgh             22
RichmondNorfolk        21
Orlando                21
Syracuse               19
```

```
HarrisburgScranton     19
Midsouth               18
GreatLakes             18
MiamiFtLauderdale      17
DallasFtWorth          17
Roanoke                17
StLouis                16
Indianapolis           16
RaleighGreensboro      16
SanFrancisco           15
Philadelphia           13
HartfordSpringfield    13
Northeast              12
Plains                 12
LasVegas               10
Southeast               9
Seattle                 9
LosAngeles              3
Name: region, dtype: int64
****************************************************************************
********************
```

To explore the catagorical columns and to count the number of values we can use the following code. We can get the following observations after using this code :

Checking Unique values :

```
df.nunique()
```

```
ID                 52
Date              104
AveragePrice      113
Total Volume     1517
4046             1517
4225             1517
4770             1516
Total Bags       1517
Small Bags       1517
Large Bags       1377
XLarge Bags       711
type                1
year                2
region             51
Year                2
Month              12
Day                31
dtype: int64
```

To explore the dataset it's also necessary to explore the unique values. If we see our dataset we can see that in some columns there are more unique values and some columns contains less unique values. Columns with less unique values normally effect more to predict the outcome. But if there are constant value in this case there will have no use of that column. From the dataset we can see that 'type'has only one unique value. So, this column should be deleted. It will not effect our dataset.

Other observations :

1. All the values in "Total Volume","4046","4225","Total Bags" these 4 columns are unique.So, these columns may not effect much to predict the outcome.

2.'4770' columns has 1516 unique values which is huge. So,this column may not effect the target variable as well.

3.The other columns have unique values. But not all the values. Some values are same. So these columns will effect much the final prediction.
4.I have split the Date column.

Split Date Column :

```
avocado['Date']=pd.to_datetime(avocado['Date'])
avocado['Date']
avocado["Year"]=avocado['Date'].dt.year
avocado["Month"]=avocado['Date'].dt.month
avocado["Day"]=avocado['Date'].dt.day
```

To split the ‘Date’ column into year, month and day in different columns firstly I have changed the datatype of Date column to datetime64. Then I have splited into different columns by using the previous code.

Change class into numeric type:

```
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for column in df.columns:
    if df[column].dtype==np.number:
        continue
    df[column]=le.fit_transform(df[column])
```

```
df.dtypes
```

```
ID              float64
AveragePrice    float64
Total Volume    float64
4046            float64
4225            float64
4770            float64
year            float64
region            int64
Month             int64
Day               int64
dtype: object
```

For analyzing the data with target all the columns should be numeric type. Also, if we want build our model we need to use the numeric column. So, it’ s necessary to convert all the object type column into numeric type. As multiple columns are object type and region column has many categorical values ; thets why I have used Label Encoder to convert all the columns by using some lines of codes.

Summary Statistics :

```
df.describe()
```

	ID	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	year	Month	Day
count	1517.000000	1517.000000	1.517000e+03	1.517000e+03	1.517000e+03	1.517000e+03	1.517000e+03	1.517000e+03	1.517000e+03	1517.000000	1517.000000	1517.000000	1517.000000
mean	26.995386	1.074990	1.601879e+06	6.464387e+05	6.114375e+05	5.040550e+04	2.935974e+05	2.487736e+05	4.264205e+04	2181.771074	2015.162821	6.362558	15.253790
std	14.848287	0.188891	4.433143e+06	1.947614e+06	1.672906e+06	1.377812e+05	7.579765e+05	6.474765e+05	1.182157e+05	7455.712144	0.369324	3.472146	8.965902
min	0.000000	0.490000	3.875074e+04	4.677200e+02	1.783770e+03	0.000000e+00	3.311770e+03	3.311770e+03	0.000000e+00	0.000000	2015.000000	1.000000	1.000000
25%	14.000000	0.980000	1.474700e+05	2.040034e+04	4.147606e+04	9.112500e+02	3.620689e+04	2.972722e+04	5.407400e+02	0.000000	2015.000000	3.000000	7.000000
50%	29.000000	1.080000	4.027919e+05	8.175117e+04	1.186649e+05	7.688170e+03	7.397906e+04	6.237569e+04	5.044350e+03	0.000000	2015.000000	6.000000	15.000000
75%	39.000000	1.190000	9.819751e+05	3.775785e+05	4.851503e+05	2.916730e+04	1.576097e+05	1.461994e+05	2.926767e+04	401.480000	2015.000000	9.000000	22.000000
max	51.000000	1.680000	4.465546e+07	1.893304e+07	1.895648e+07	1.381516e+06	6.736304e+06	5.893642e+06	1.121076e+06	108072.790000	2016.000000	12.000000	31.000000

By using describe() function we can explore the count, mean , median, standard deviation, minimum value, 25^{th}, 50^{th} and 75^{th} percentile , maximum value.

We can find the following observations from the dataset :

1.Maximum values of ID,AveragePrice, Total Volume, 4046 ,4225 ,4770 ,Total Bags ,Small Bags ,Large Bags ,XLarge Bags ,year ,Month ,Day are : 51.000000 ,1.680000 ,4.465546e+07 ,1.893304e+07 ,1.895648e+07 ,1.381516e+06 ,6.736304e+06 ,5.893642e+06 ,1.121076e+06 ,108072.790000 ,2016.000000 ,12.000000 ,31.000000

2. Minimum values of ID,AveragePrice, Total Volume, 4046 ,4225 ,4770 ,Total Bags ,Small Bags ,Large Bags ,XLarge Bags ,year ,Month ,Day are 0.000000 ,0.490000 ,3.875074e+04 ,4.677200e+02 ,1.783770e+03 ,0.000000e+00 ,3.311770e+03 ,3.311770e+03 ,0.000000e+00 ,0.000000 ,2015.000000 ,1.000000 ,1.000000

75 percentile and max value

1. In the 'Total Volume' 75 percentile and max value has huge diffenece. Most probably there are outliers.

2. In the "4046","4225","4770","Total Bags","Small Bags","Large Bags","XLarge Bags" colums 75 percentile and max value has huge diffenece. Most probably there are outliers. So, there are outliers too.

3. In the "ID", "AveragePrice", "year", "Month", "Day" columns 75 percentile and max value has no huge diffenece. It looks normal.

50 percentile(median) and mean value

1. In the 'Total Volume',"4046","4225","4770","Total Bags","Small Bags","Large Bags","XLarge Bags" 50 percentile and mean value has huge diffenece. Most probably there are outliers.

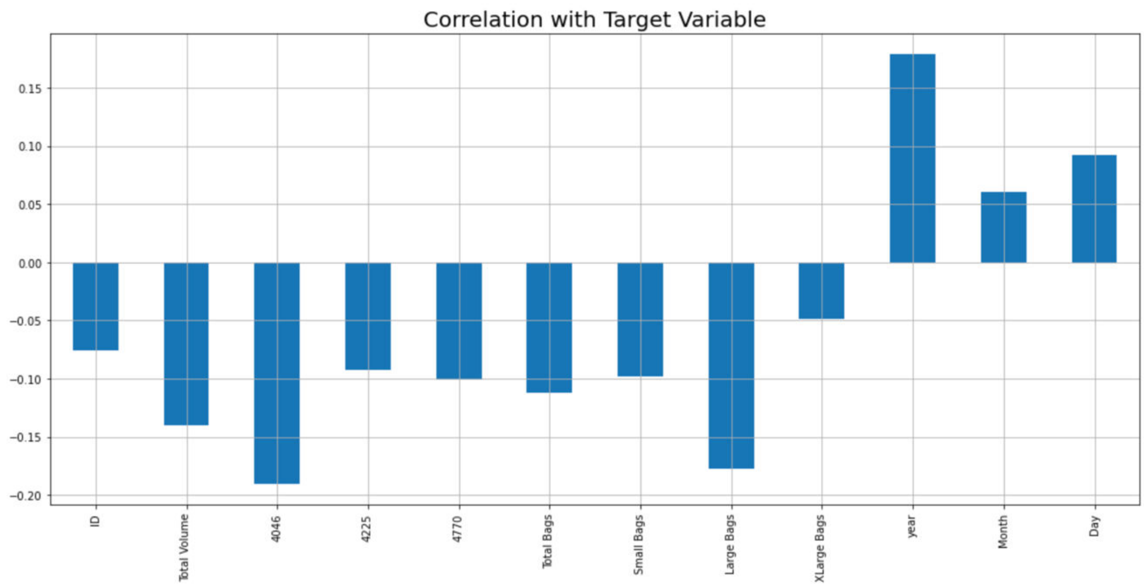
2.In the "ID", "AveragePrice", "year", "Month", "Day" columns 50 percentile and mean value has no huge difference. It looks normal.

Correlation:

Correlation only with target variable :

```
plt.figure(figsize=(18,8))
df.drop('AveragePrice',axis=1).corrwith(df['AveragePrice']).plot(kind='bar',grid=True)
plt.xticks(rotation='vertical')
plt.title("Correlation with Target Variable",fontsize=20)
```

```
Text(0.5, 1.0, 'Correlation with Target Variable')
```



We can see the correlation of every column with the target variable by using barplot. The peak points which are above 0 are positively correlated with the target variable. And the peak points which are under 0 are negatively correlated with the target variable. But the problem is we can no know the exact value of correlation by using barplot. To know the exact correlation we have another technique.

```
df.corr()
```

	ID	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	year	Month	Day
ID	1.000000	-0.075531	0.091934	0.092013	0.093566	0.075647	0.081007	0.083485	0.065364	-0.050993	-0.012854	-0.598729	-0.261166
AveragePrice	-0.075531	1.000000	-0.140470	-0.190096	-0.092084	-0.100567	-0.111597	-0.097682	-0.177480	-0.048284	0.178683	0.060642	0.092589
Total Volume	0.091934	-0.140470	1.000000	0.985568	0.987154	0.934347	0.967689	0.960642	0.910226	0.521717	-0.004189	-0.055212	-0.034857
4046	0.092013	-0.190096	0.985568	1.000000	0.951952	0.898570	0.930395	0.922278	0.884229	0.474063	-0.040439	-0.053007	-0.031806
4225	0.093566	-0.092084	0.987154	0.951952	1.000000	0.927757	0.951779	0.943988	0.900496	0.504718	-0.007254	-0.058429	-0.044008
4770	0.075647	-0.100567	0.934347	0.898570	0.927757	1.000000	0.926405	0.923607	0.840523	0.646053	0.029222	-0.055079	-0.021318
Total Bags	0.081007	-0.111597	0.967689	0.930395	0.951779	0.926405	1.000000	0.997341	0.911336	0.601856	0.090108	-0.047747	-0.021137
Small Bags	0.083485	-0.097682	0.960642	0.922278	0.943988	0.923607	0.997341	1.000000	0.879131	0.611300	0.106382	-0.048978	-0.016462
Large Bags	0.065364	-0.177480	0.910226	0.884229	0.900496	0.840523	0.911336	0.879131	1.000000	0.447779	-0.018678	-0.039242	-0.046578
XLarge Bags	-0.050993	-0.048284	0.521717	0.474063	0.504718	0.646053	0.601856	0.611300	0.447779	1.000000	0.218342	0.021452	0.019315
year	-0.012854	0.178683	-0.004189	-0.040439	-0.007254	0.029222	0.090108	0.106382	-0.018678	0.218342	1.000000	-0.013144	0.029744
Month	-0.598729	0.060642	-0.055212	-0.053007	-0.058429	-0.055079	-0.047747	-0.048978	-0.039242	0.021452	-0.013144	1.000000	-0.024337
Day	-0.261166	0.092589	-0.034857	-0.031806	-0.044008	-0.021318	-0.021137	-0.016462	-0.046578	0.019315	0.029744	-0.024337	1.000000

From the above figure we we be able to know the exact correlation of each column with the target variable.
The findings are mentioned below:

1. Negative correlation with Average price : 'ID','Total Volume', '4046', '4225', '4770', 'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags'

2. Positive correlation : 'year','Month', 'Day'

3. Strong correlation : 'Total Volume', '4046', 'Large Bags', 'year'

4. Weak correlation: 'ID','4225', '4770','Total Bags', 'Small Bags','XLarge Bags' ,'Month', 'Day'

Correlation among all the variables:

```
plt.subplots(figsize=(25,15))
sns.heatmap(df.corr(),annot=True)
```

From the previous steps we have only known about the correlation of each column with the target variable. But by using heatmap we will be able to explore the correlation among all the numeric variables present in the dataset.

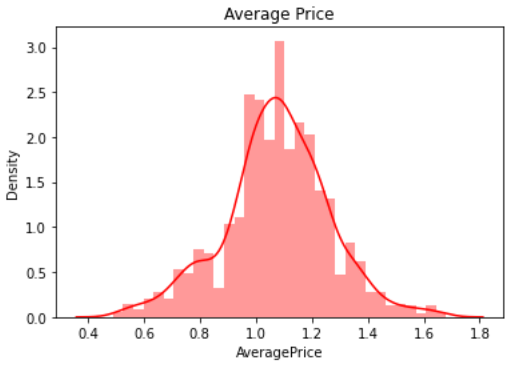
We can also get the same result using df.corr() method.

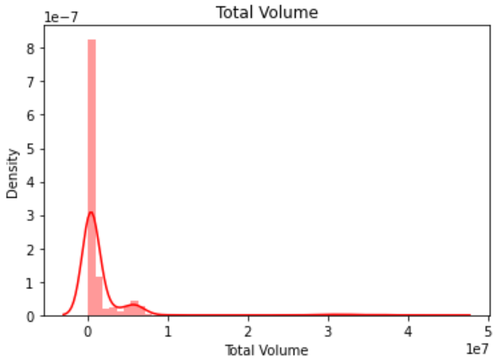
Data Visualization :

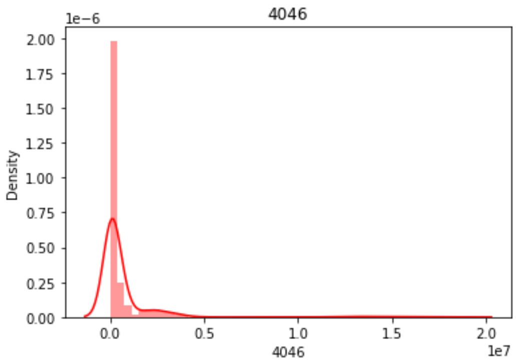
Our feature variable of interest is AveragePrice.

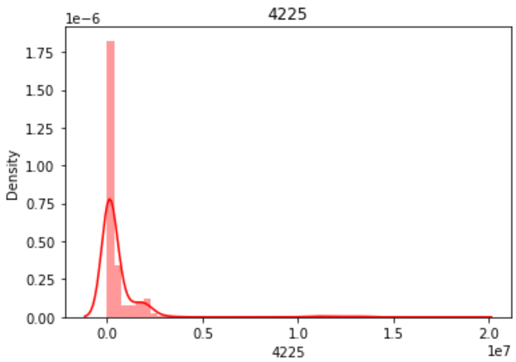
This column shows that the target variable is regression type. So, finally we need to use regression type algorithm.

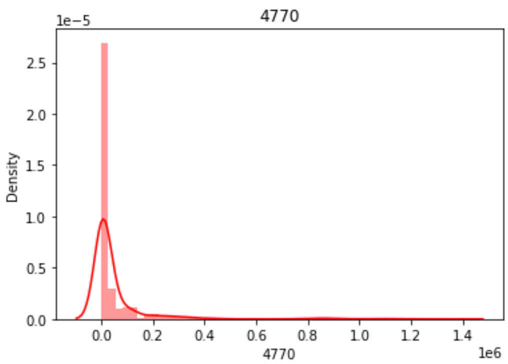
Univariate anslysis :

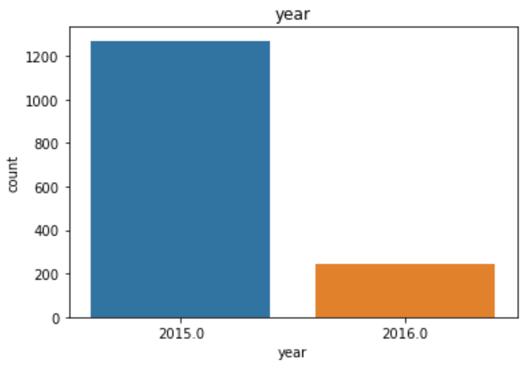

Average Price
Density
AveragePrice

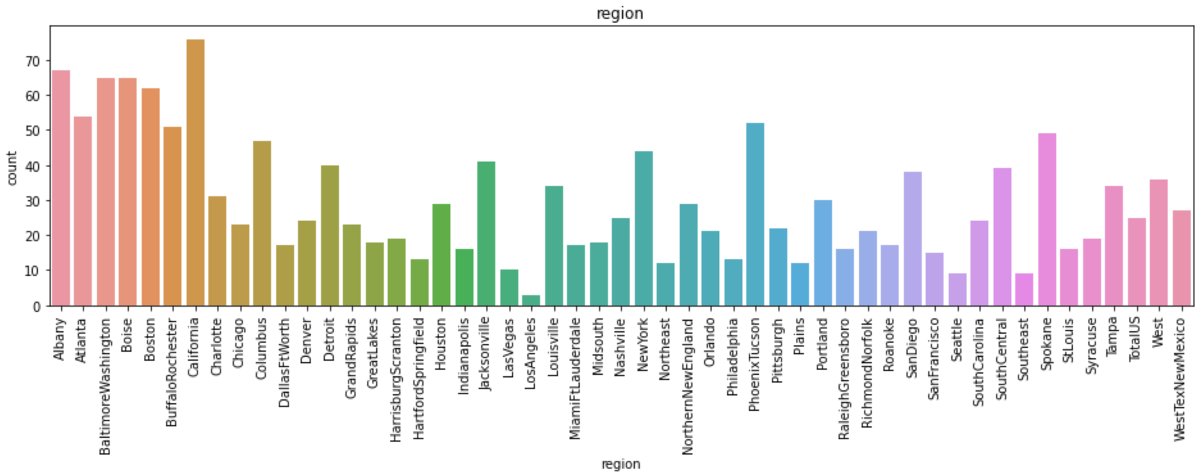

Total Volume
1e−7
Density
Total Volume
1e7

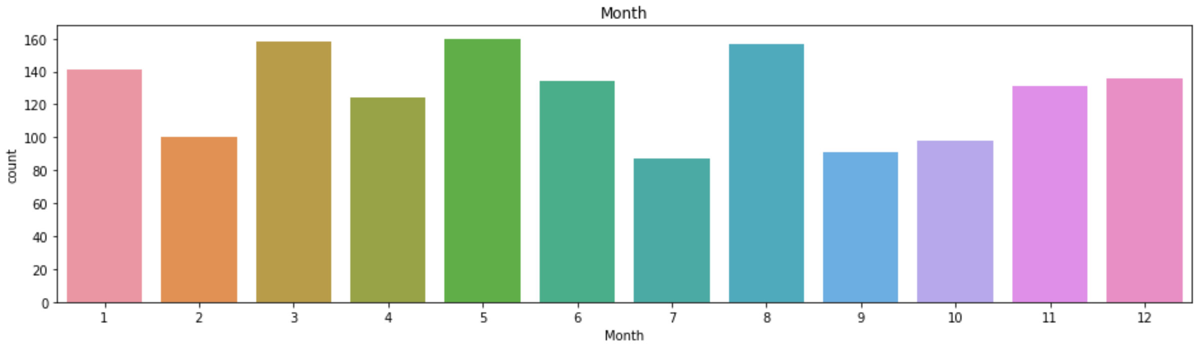

4046
1e−6
Density
4046
1e7

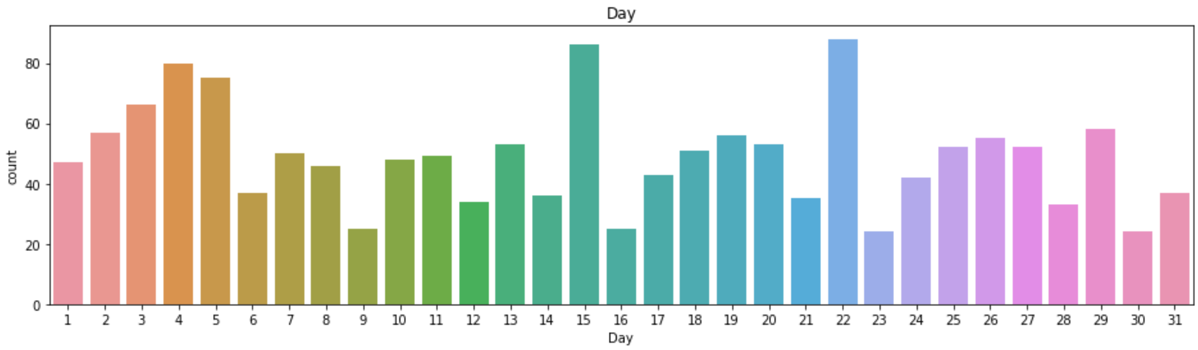

4225
1e−6
Density
4225
1e7


4770
1e−5
Density
4770
1e6









In some columns the values are continuous type. For continuous type of data I have used distplot with the red coloured mean value to visualize every column perfectly. In some columns such as 4770,4225,4046 the peak point is too high. That means there are outliers. Some of the columns are catagorical and there are few specific values. In these cases I have used countplot.

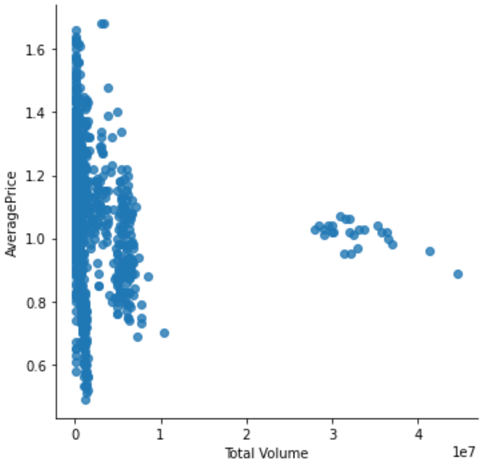
Sample code for histogram :

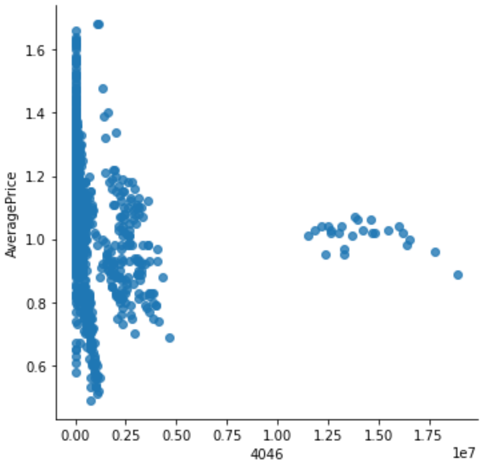
```
sns.distplot(df['column_name'],color='r')
plt.title('column_name’)
plt.show()
```

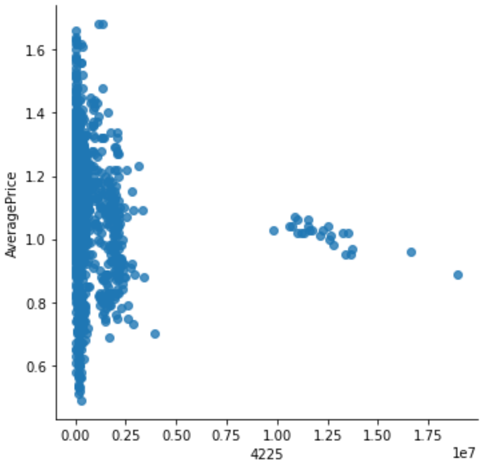
Sample code for countplot :

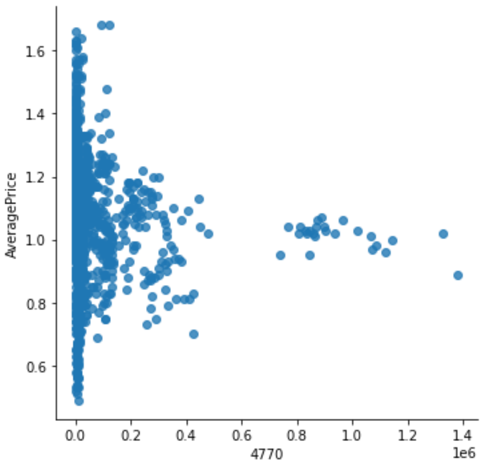
```
plt.subplots(figsize=(20,6))
sns.countplot(x='Column Name',data=df)
plt.title('Column Name')
plt.xticks(rotation=20)
plt.show()

print(df.Column Name.value_counts())
```

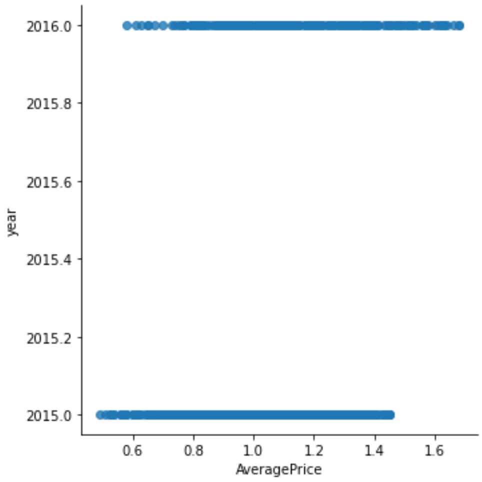
Bivariate Analysis :

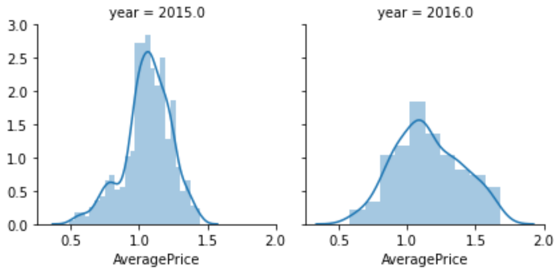


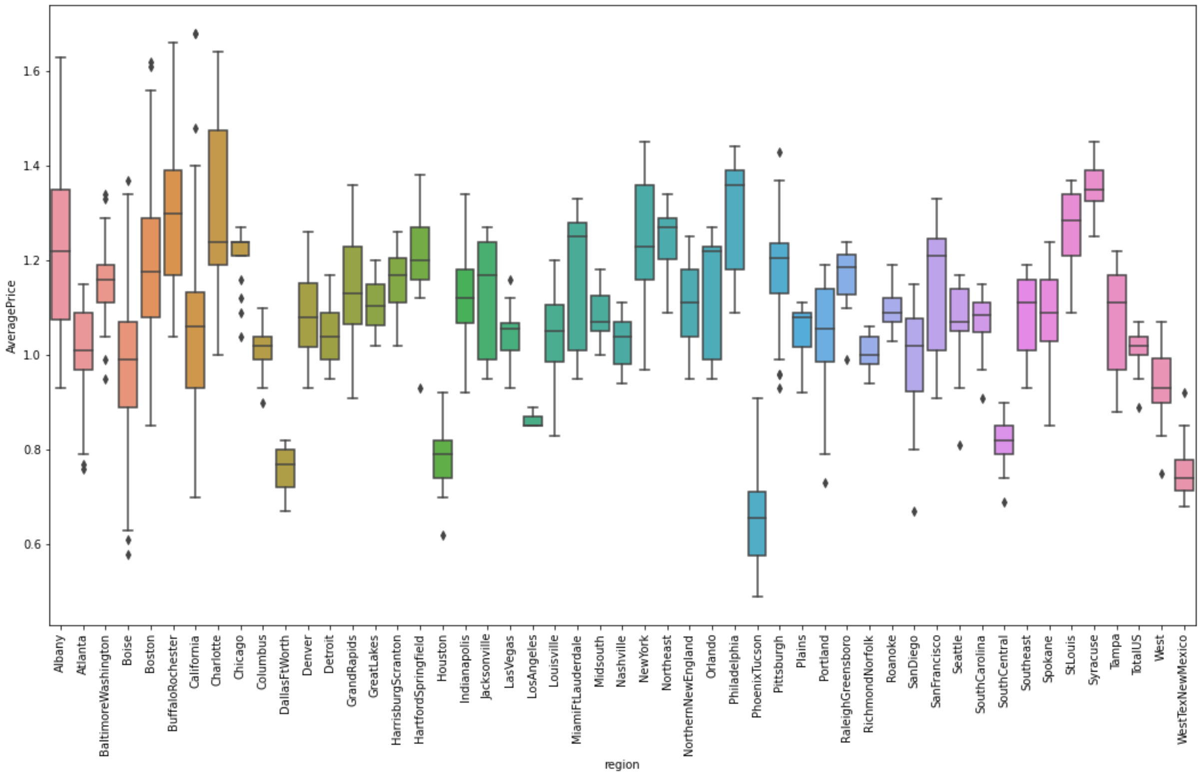

AveragePrice
0.6
0.8
1.0
1.2
1.4
1.6
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
1e7
4046

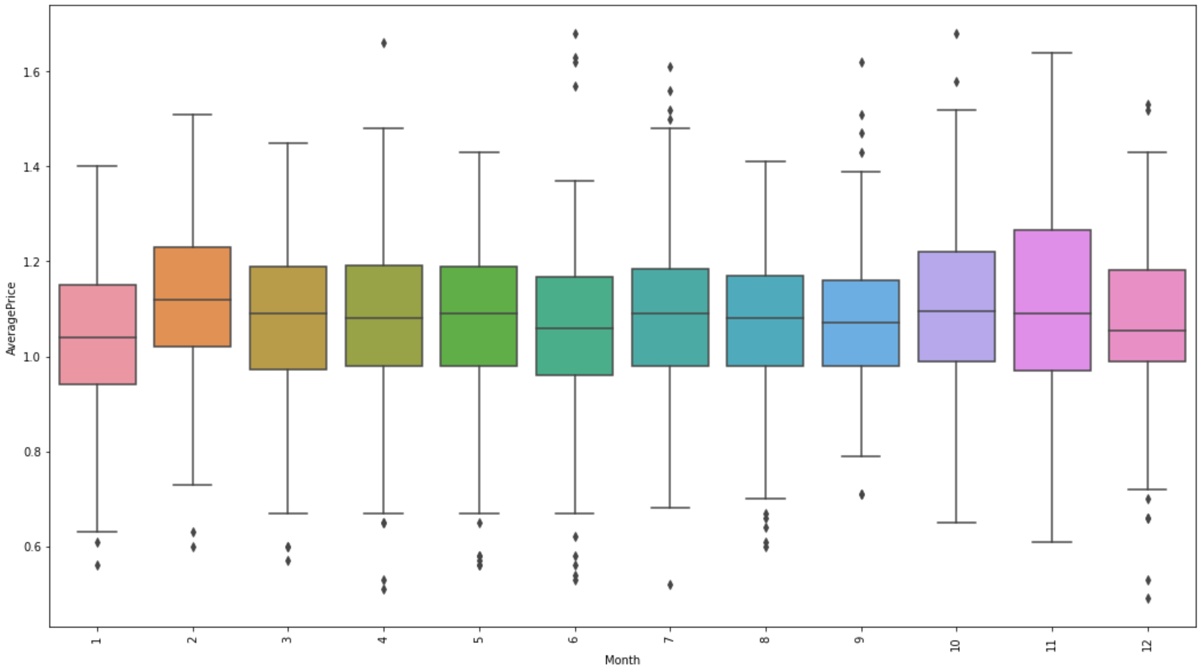

AveragePrice
0.6
0.8
1.0
1.2
1.4
1.6
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
1e7
4225

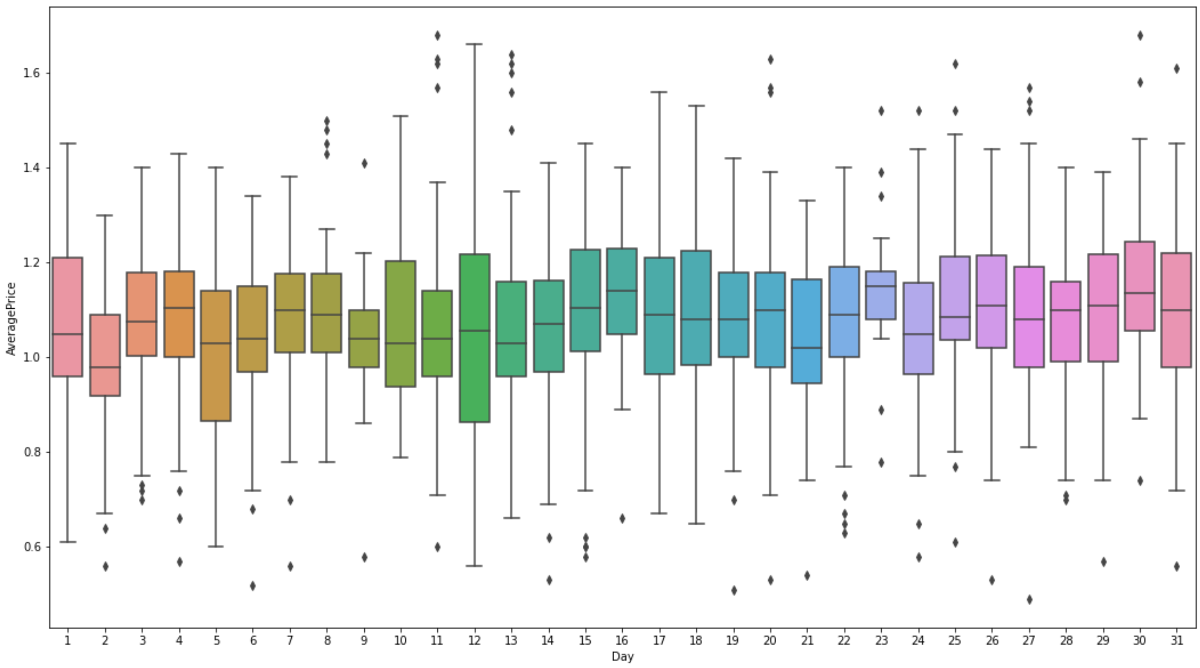

AveragePrice
0.6
0.8
1.0
1.2
1.4
1.6
0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4
1e6
4770


year
AveragePrice
2016.0
2015.8
2015.6
2015.4
2015.2
2015.0
0.6
0.8
1.0
1.2
1.4
1.6


year = 2015.0
year = 2016.0
AveragePrice
AveragePrice


AveragePrice
region
Albany
Atlanta
BaltimoreWashington
Boise
Boston
BuffaloRochester
California
Charlotte
Chicago
Columbus
DallasFtWorth
Denver
Detroit
GrandRapids
GreatLakes
HarrisburgScranton
HartfordSpringfield
Houston
Indianapolis
Jacksonville
LasVegas
LosAngeles
Louisville
MiamiFtLauderdale
Midsouth
Nashville
NewYork
Northeast
NorthernNewEngland
Orlando
Philadelphia
PhoenixTucson
Pittsburgh
Plains
Portland
RaleighGreensboro
RichmondNorfolk
Roanoke
SanDiego
SanFrancisco
Seattle
SouthCarolina
SouthCentral
Southeast
Spokane
StLouis
Syracuse
Tampa
TotalUS
West
WestTexNewMexico





In bivariate analysis most of the times I have used scatterplot. Because most of the values are continuous types. We have used x as the column name and y for target variable named ‘AveragePrice’. In some cases such as ‘region’ , ‘month’, ‘day’ vs ‘AveragePrice’ I have used boxplot. Because ‘region’ , ‘month’, and ‘day’ are categorical type variable and there are huge data as well. The sample codes are given below :

For boxplot:

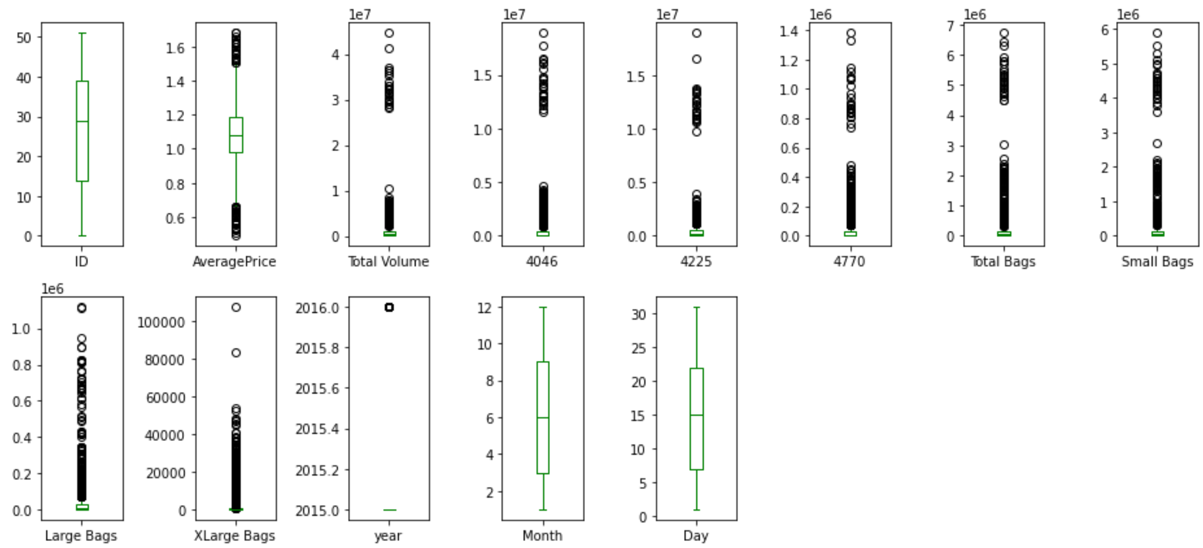
```
plt.figure(figsize=(18,10))
plt.xticks(rotation='vertical')
sns.boxplot(x='column_name',y='AveragePrice', data=df)
```

For Scatterplot :

```
plt.figure(figsize=(8,4))
sns.lmplot(x='column_name',y='Target_variable',fit_reg=False,data=df)
plt.show()
```

Check Outliers :

```
df.plot(kind='box',subplots=True,layout=(4,5),color='green',figsize=(13,
13))
plt.tight_layout()
```



After illustrating the above figure we can explore that AveragePrice,Total Volume, 4046, 4225, 4770, Total Bags, Small Bags, Large Bags, Xlarge Bags having outliers. T he other columns don't have outliers. So, we need to treat the outliers.

We can also check the outliers individually.

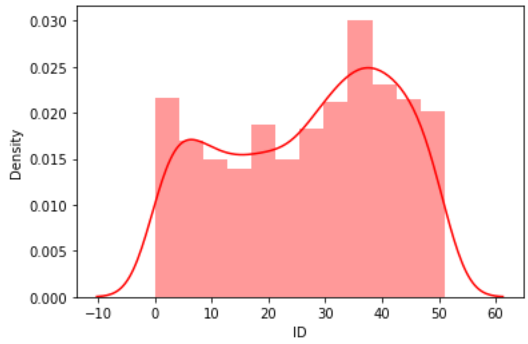
Dropping columns :

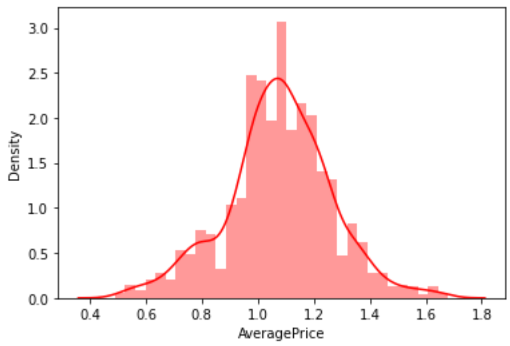
```
df.drop(['Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags'],axis=1,inplace=True)
```

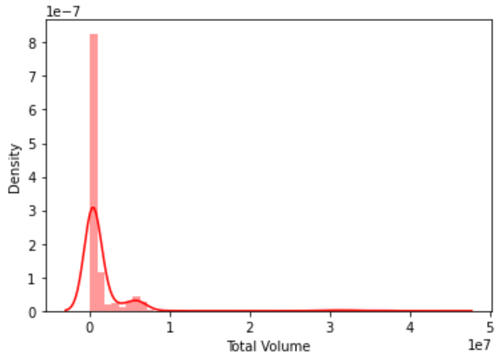
```
df.drop(['type'],axis=1,inplace=True)
```

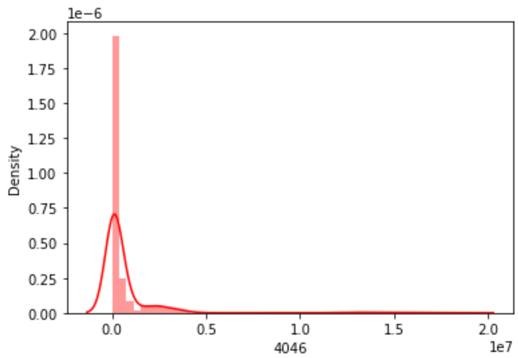
1. These 'type' colum will be deleted. Because there are either one single value.
2. Total Bags, Small Bags, Large Bags, Xlarge Bags having a lot of outliers.And most of the values are unique. That's why I have deleted these colums as well.

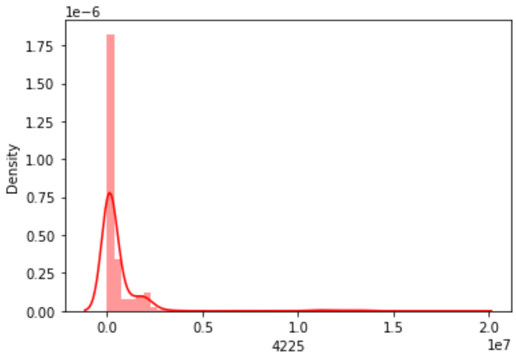
Skewness:

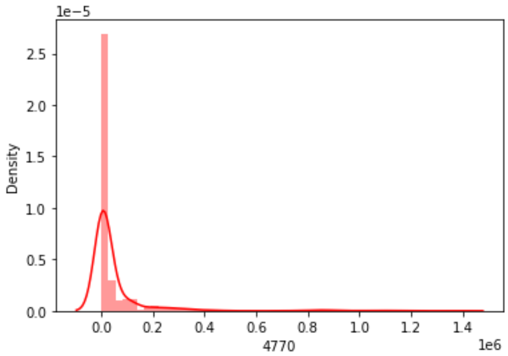

0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
−10
0
10
20
30
40
50
60
ID

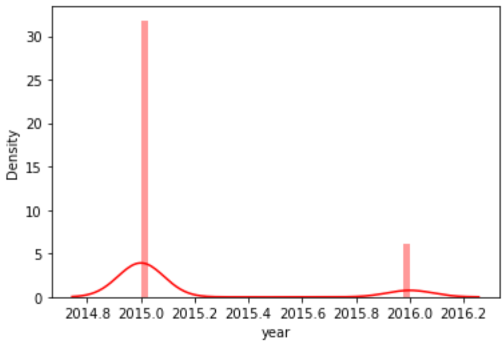

3.0
2.5
2.0
1.5
1.0
0.5
0.0
Density
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
AveragePrice

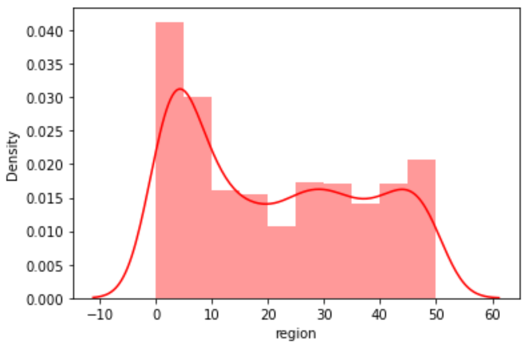

1e−7
8
7
6
5
4
3
2
1
0
Density
0
1
2
3
4
5
Total Volume
1e7

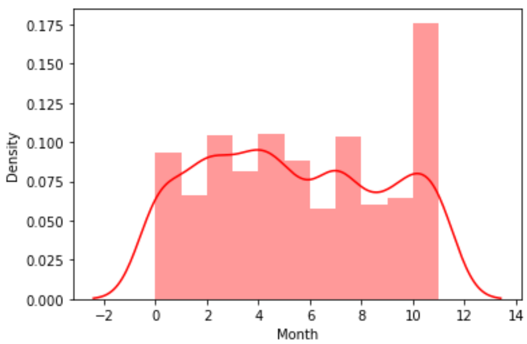

1e−6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
0.0
0.5
1.0
1.5
2.0
4046
1e7

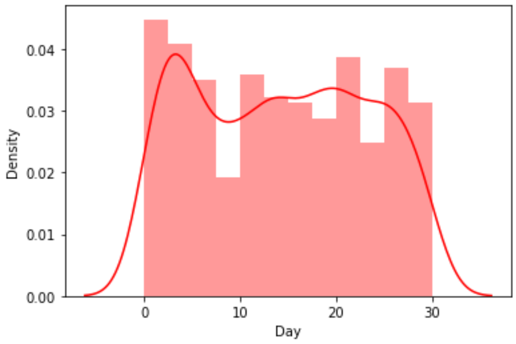

1e−6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
0.0
0.5
1.0
1.5
2.0
1e7
4225


1e−5
2.5
2.0
1.5
1.0
0.5
0.0
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6
4770


30
25
20
15
10
5
0
Density
2014.8
2015.0
2015.2
2015.4
2015.6
2015.8
2016.0
2016.2
year


0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
−10
0
10
20
30
40
50
60
region


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
−2
0
2
4
6
8
10
12
14
Month



In most of the columns skewness is present. In the Total Volume, 4046, 4225, 4770, year columns right skewness is present.

df.skew()

```
: df.skew()

: ID              -0.234824
  AveragePrice    -0.109444
  Total Volume     6.200138
  4046             6.051830
  4225             6.394926
  4770             5.405164
  year             1.828332
  region           0.288146
  Month            0.101439
  Day              0.041303
  dtype: float64
```

If we want to know the exact value then skew() function is the best way to know the skewness of the variavles. Here, The standard value I have used is 0.56. If the value is not in between -0.56 and 0.56 that means skewness is present in those columns.

Removing Outliers :

```
from scipy.stats import zscore
import numpy as np
z=np.abs(zscore(df))
threshold=3
df_new=df[(z<3).all(axis=1)]
print(df.shape)
print(df_new.shape)
```

```
(1517, 10)
(1487, 10)
```

```
q1=df.quantile(0.25)
q3=df.quantile(0.75)
IQR=q3-q1
print(IQR)
df_new1=df[~((df<(q1-1.5*IQR))|(df>(q3+1.5*IQR))).any(axis=1)]
print(df_new1.shape)
```

```
ID                    25.00
AveragePrice           0.21
Total Volume      834505.09
4046              357178.14
4225              443674.28
4770               28256.05
year                   0.00
region                29.00
Month                  6.00
Day                   15.00
dtype: float64
(990, 10)
```

```
#z-score
percentage_loss_z=(30/1517)*100
print(percentage_loss_z)

#IQR
percentage_loss_IQR=(527/1517)*100
print(percentage_loss_IQR)
```

```
1.977587343441002
34.73961766644694
```

I have used both z-score and IQR method to remove the outliers. But using both of the methods we can see that the percentage of data losing of z-score is almost 2 % and IQR is almost 35%. By using IQR I have loosen a lot of data. That's why I have used z_score.

Split the data into x and y

```
x=df_new.drop('AveragePrice',axis=1)
y=df_new['AveragePrice']
```

I have splitted the dataset into x and y where x represents all the columns except the target variable AveragePrice and y represents the target variable.

Treating Skewness via yeo-johnson method:

```
from sklearn.preprocessing import power_transform
x=power_transform(x,method='yeo-johnson')
```

```
x=pd.DataFrame(x)
x.skew()
```

```
0   -0.341696
1    0.000000
2   -0.030088
3    0.000733
4   -0.054507
5    1.815823
6   -0.204240
7   -0.189196
8   -0.246501
dtype: float64
```

After treating the skewness we can explore that the skewness of almost all the columns has been removed. Now, only one column having skewness which is 1.82.

Model Training :

Spilitting the data into input and output variable :

```
x=df_new.drop('AveragePrice',axis=1)
y=df_new['AveragePrice']
```

We can split the data into x and y. x is having all the columns except the target variable. Y is having only the target column.

Spliting the data into training and testing set :

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=42)
```

Build Model :

Scaling :

As I have used yeo-johnson method to remove the skewness that's why there is no need to scale the dataset. It automatically scale the data as well.

Importing all the model Library :

```
# Libraries for data modelling
from sklearn.linear_model import LogisticRegression,Ridge,Lasso,LinearRegression,ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVR,SVC

#Importinf boosting models
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor,BaggingRegressor,ExtraT

#Importing error metrics
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import GridSearchCV,cross_val_score
```

All algorithms are in one code :

```
model=[Lasso(),Ridge(),DecisionTreeRegressor(),KNeighborsRegressor(),RandomForestRegressor(),AdaBoostRegressor(),Gradie

for m in model:
    m.fit(x_train,y_train)
    m.score(x_train,y_train)
    predm=m.predict(x_test)
    pred_train=m.predict(x_train)
    pred_test=m.predict(x_test)
    print("\033[1m"+ 'R2 score of',m,'is : ' + "\033[0m" )  # Make the line bold
    print(r2_score(y_test,predm))
    print("Training r2_score is :",r2_score(y_train,pred_train)*100)
    print("Testing r2_score is :",r2_score(y_test,pred_test)*100)
    print("Mean Absolute Eror : ",mean_absolute_error(y_test,predm))
    print("Mean Squared Eror : ",mean_squared_error(y_test,predm))
    print("Root Mean Squared Eror : ",np.sqrt(mean_squared_error(y_test,predm)))
    print('*********************************************************************')
    print('\n')
```

By using all the algoriths one by one we can use one function to implement all the algorithms. If we summarize the result we get the following r2 score :
Lasso(),Ridge(),DecisionTreeRegressor(),KNeighborsRegressor(),RandomForestRegressor(), AdaBoostRegressor(),GradientBoostingRegressor(),BaggingRegressor(),ExtraTreesRegressor(),LinearRegression(),SVR(),ElasticNet(),RandomForestRegressor()

1. R2 Score of Lasso()is : -0.002700101752157291

2. R2 Score of Ridge() is : 0.314036312443166

3. R2 Score of DecisionTreeRegressor() is : 0.7004478541441683

4. R2 Score of KNeighborsRegressor() is : 0.7012701159105127

5. R2 Score of RandomForestRegressor() is : 0.8323141228135249

6. R2 Score of GradientBoostingRegressor() is : 0.7386288417238536

7. R2 Score of BaggingRegressor() is : 0.8066834502482474

8. R2 Score of ExtraTreesRegressor() is : 0.8780266306177331

9. R2 Score of LinearRegression() is : 0.31405425973319234

10. R2 Score of SVR() is : 0.6902969949351738

11. R2 Score of ElasticNet() is : -0.002700101752157291

12. R2 Score of AdaBoostRegressor() is : 0.6352453884260243

We have got good R2 Score by using the following algorithms :
ExtraTreesRegressor(), BaggingRegressor(), RandomForestRegressor()
But out of these algorithms ExtraTreesRegressor () is giving the best result. We will do the hyperparameter tuning to reduce the overfitting.

Using Best Parameter :

```
parameters={'n_estimators':[400,500,600,700],'random_state':[100,200,300,400]}
ETR=ExtraTreesRegressor()

clf=GridSearchCV(ETR,parameters)
clf.fit(x,y)
print(clf.best_params_)
```

```
{'n_estimators': 400, 'random_state': 400}
```

Here the best parametes for ExtraTreesRegressor () are 'n_estimator': '400', 'random_state': 400.

Using Best Parameter :

After using the best parametes we have got the accuracy for `ExtraTreesRegressor` (). The r2_score is 0.9890167364016736.

Cross Validation score :

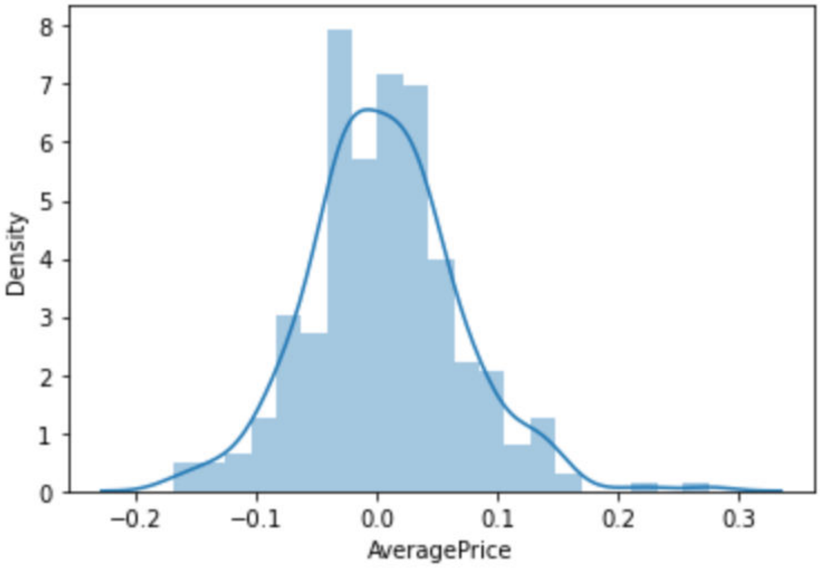
```
model=[LogisticRegression(),DecisionTreeClassifier(),KNeighborsClassifier(),GaussianNB(),SVC(),RandomForestClassifier(),AdaBoostClassifier(),GradientBoostingClassifier(
for m in model:
    score=cross_val_score(m,X,Y,cv=10,scoring='accuracy')
    print("Model : ",m)
    print("Mean Score : ",score.mean())
    print("Standard deviation : ",score.std())
    print('**********************************************************************')
    print('\n')
```

Result :

```
odel :  ExtraTreesRegressor()
Mean Score :  0.17814803556094547
Standard deviation :  0.30906649700725297
*******************************************************************
```

Plotting the distribution plot and we find the Gaussian plot

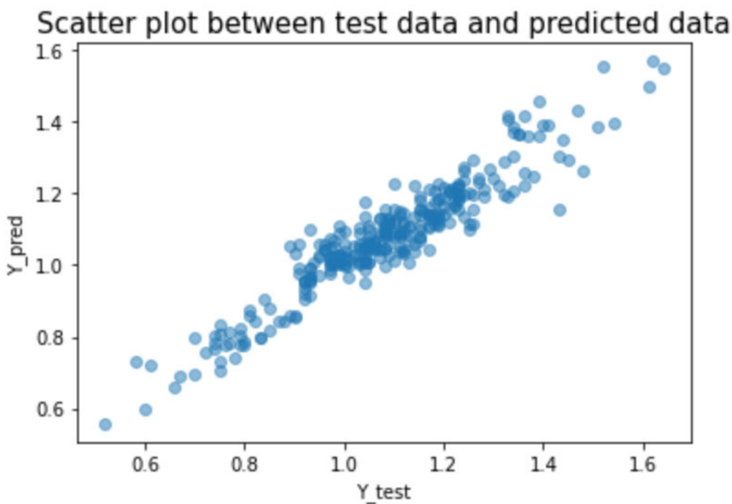
```
sns.distplot(y_test-predETR)
plt.show()
```



We can explore that the after using the `ExtraTreesRegressor()` algorithm and hyperparameter tuning the distribution looks normal.

Scatter plot between test data and prediction

```
plt.scatter(y_test,predETR,alpha=0.5)
plt.xlabel("Y_test")
plt.ylabel("Y_pred")
plt.title("Scatter plot between test data and predicted data",fontsize=15)
plt.show()
```



The predicted data and the original are almost on the same line. So, this model will be accepted.

Saving the model

import joblib

Save the model as a pickle in a file

```
joblib.dump(ETR,'Avocado_Prediction.pkl')
```

```
['Avocado_Prediction.pkl']
```

To save our model first we need to import joblib. Then we can save our model as pkl file.

Name : Md Rashidunnabi
Email : rashidunnabi11@gmail.com
Phone : +8801319527349
Whatsapp : +8801319527349