Avocado Dataset (Project 3)

Name = Chandrima Chatterjee

Problem Statement for the Dataset

"The Avocado dataset we are classifying Organic & Conventional Type and predicting the Average price using Regression model from year 2015, 2016, 2017 and 2018 data."

This Dataset includes the data of consumption of the Avocado fruit in different city of the USA ranging from years from 2015 to 2018.

We have two types of Avocado available here:

1. Organic which is healthy
2. Conventional

The variables on this dataset available are as follows:

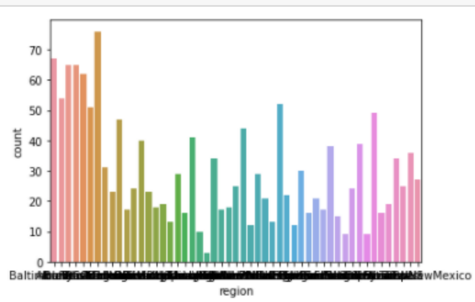
1. Categorical: ‘region’,’type’
2. Date: ‘Date’
3. Numerical:’Total Volume’, ‘4046’, ‘4225’, ‘4770’, ‘Total Bags’, ‘Small Bags’,’Large Bags’,’XLarge Bags’,’Year’
4. Target:‘AveragePrice’

The below dataset is extracted from the different outlets which includes Grocery, mass, clubs, drug, dollar, millitary units as we can see that the Avocado's are being sold in small to large bags.

The Average Price (of avocados) in the table reflects a per unit cost (per avocado), 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.

Data Analysis

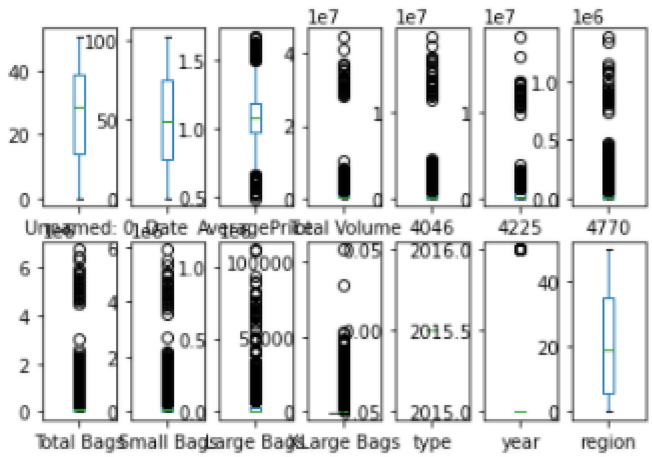
```
]: sns.countplot(x='region',data=df)
plt.show()
```



Target/dependent variable is discrete and categorical in nature -- highest count of region is of california i.e 76. -- the no. of counts of region ranges from 0 to 80. -- lowest count of region is of LosAngles i.e 3.#Lets use LabelEncoder to convert all categorical data into numerical data, so that EDA could be done properly to undertand the dataset better.

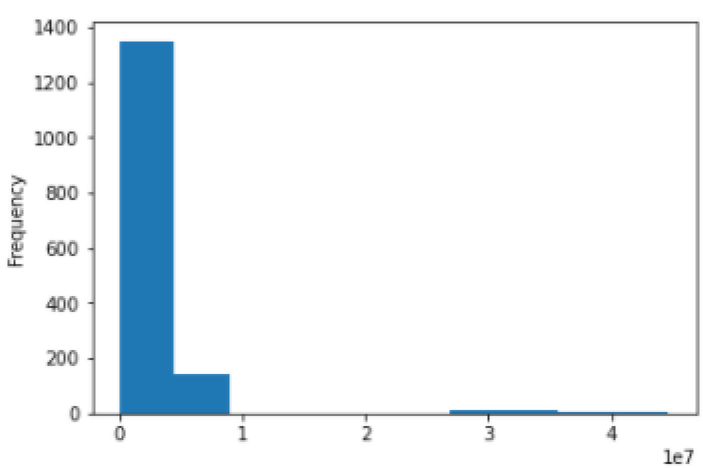
```
df.plot(kind ='box',subplots = True,layout=(2,7))
```

```
Unnamed: 0        AxesSubplot(0.125,0.536818;0.0945122x0.343182)
Date           AxesSubplot(0.238415,0.536818;0.0945122x0.343182)
AveragePrice   AxesSubplot(0.351829,0.536818;0.0945122x0.343182)
Total Volume   AxesSubplot(0.465244,0.536818;0.0945122x0.343182)
4046           AxesSubplot(0.578659,0.536818;0.0945122x0.343182)
4225           AxesSubplot(0.692073,0.536818;0.0945122x0.343182)
4770           AxesSubplot(0.805488,0.536818;0.0945122x0.343182)
Total Bags           AxesSubplot(0.125,0.125;0.0945122x0.343182)
Small Bags        AxesSubplot(0.238415,0.125;0.0945122x0.343182)
Large Bags        AxesSubplot(0.351829,0.125;0.0945122x0.343182)
XLarge Bags       AxesSubplot(0.465244,0.125;0.0945122x0.343182)
type              AxesSubplot(0.578659,0.125;0.0945122x0.343182)
year              AxesSubplot(0.692073,0.125;0.0945122x0.343182)
region            AxesSubplot(0.805488,0.125;0.0945122x0.343182)
dtype: object
```



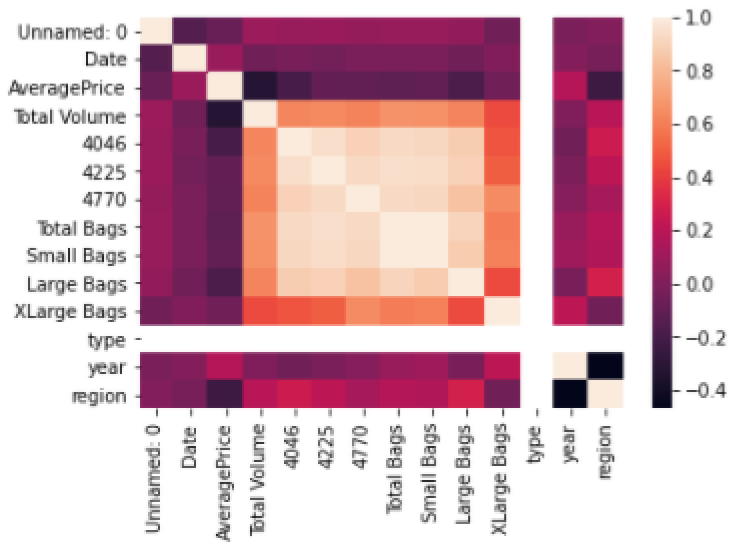

```
df['Total Volume'].plot.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f21323bac0>
```

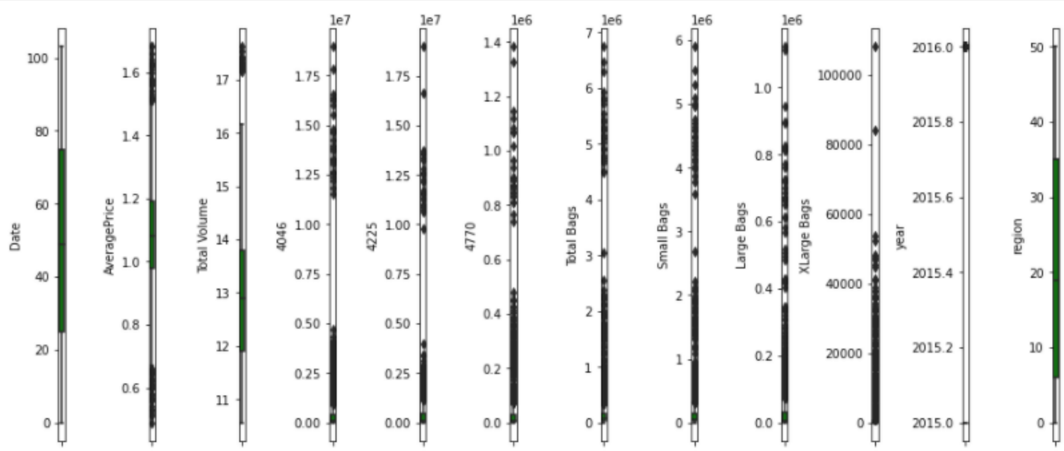


```python
#Checking correlation with the help of heatmap.
sns.heatmap(dfcor)
```

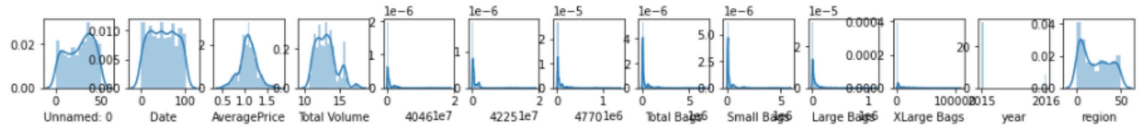
<matplotlib.axes._subplots.AxesSubplot at 0x1f21b584400>




```python
#Lets visualize outliers through boxplots
plt.figure(figsize=(ncol,5*ncol))
for i in range (1,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(df[collist[i]],color='green',orient='v')
    plt.tight_layout()
```



```
: plt.figure(figsize=(20,20))
for i in range (0,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.distplot(df[collist[i]])
```




```
: #lets again check the skewness
df.skew()
```

```
: Unnamed: 0     -0.234824
Date            0.012623
AveragePrice   -0.109444
Total Volume    0.442493
4046           -0.160268
4225            0.184436
4770           -0.355508
Total Bags      0.695502
Small Bags      0.713843
Large Bags     -0.912766
XLarge Bags     0.783913
year            1.828332
region          0.288146
dtype: float64
```

Preprocessing pipeline

```
: #Now treating the outliers
from scipy.stats import zscore
z = np.abs(zscore(df))
z
```

```
: array([[1.81868039, 1.37776563, 1.35048079, ..., 0.81077519, 0.44100815,
         1.3143384 ],
        [1.75131034, 0.57857991, 1.45639674, ..., 0.81077519, 0.44100815,
         1.3143384 ],
        [1.6839403 , 0.22060582, 0.76783831, ..., 0.81077519, 0.44100815,
         1.3143384 ],
        ...,
        [1.01023983, 1.51928262, 2.14485045, ..., 1.10389091, 2.26753179,
         0.88028586],
        [0.94286978, 1.07807099, 2.09189247, ..., 0.81077519, 2.26753179,
         0.88028586],
        [0.87549974, 0.27888526, 1.88006056, ..., 0.81077519, 2.26753179,
         0.88028586]])
```

```
: threshold = 3
print(np.where(z>3))
```

```
(array([ 760, 1182, 1182, 1183, 1183, 1184, 1184, 1185, 1185, 1186, 1186,
       1187, 1188, 1188, 1189, 1191, 1346, 1411, 1457, 1458], dtype=int64), array([2, 7, 8, 7, 8, 7, 8, 7, 8, 7, 8, 8, 7, 8, 7, 7, 6, 2,
2, 2],
      dtype=int64))
```

```
: df_new=df[((z<3).all(axis=1))] #Removing the outliers
```

```
: z[760][2]
```

```
: 3.097989311954043
```

```
: z[1182][3]
```



```
: 2.761987656202092
```

Building Machine Learning Models

```
from sklearn.svm import SVC

svc=SVC(kernel="linear", C=1)
svc.fit(train_x,train_y)

predsvc =svc.predict(test_x)

print('actual and predicted value score',accuracy_score(test_y,predsvc))
print(confusion_matrix(test_y,predsvc))
print(classification_report(test_y,predsvc ))
```

```
actual and predicted value score 0.9461077844311377
[[20  0  0 ...  0  0  0]
 [ 0  6  0 ...  0  0  0]
 [ 0  0 15 ...  0  0  0]
 ...
 [ 0  0  0 ...  2  0  0]
 [ 0  0  0 ...  0  5  0]
 [ 0  0  0 ...  0  0  5]]
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        20
           1       0.60      0.86      0.71         7
           2       0.94      1.00      0.97        15
           3       0.94      1.00      0.97        16
           4       0.92      1.00      0.96        11
           5       1.00      0.89      0.94         9
           6       1.00      1.00      1.00        13
           7       0.82      0.90      0.86        10
           8       1.00      1.00      1.00         5
           9       0.89      0.89      0.89         9
          10       1.00      0.67      0.80         3
          11       1.00      1.00      1.00         4
          12       1.00      1.00      1.00        11
          13       1.00      1.00      1.00         6
          15       1.00      1.00      1.00         1
          16       1.00      1.00      1.00         3
```

```python
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n_neighbors = 5)
knn.fit(train_x,train_y)

predknn = knn.predict(test_x)

print(accuracy_score(predknn,test_y))
print(confusion_matrix(test_y,predknn))
print(classification_report(test_y,predknn))
```

```
0.2874251497005988
[[19  0  0 ...  0  0  0]
 [ 0  4  0 ...  0  0  0]
 [ 0  0 11 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  1  0]
 [ 0  0  0 ...  0  0  0]]
              precision    recall  f1-score   support

           0       0.51      0.95      0.67        20
           1       0.11      0.57      0.18         7
           2       0.28      0.73      0.40        15
           3       0.39      0.69      0.50        16
           4       0.10      0.09      0.10        11
           5       0.50      0.33      0.40         9
           6       0.62      0.62      0.62        13
           7       0.20      0.10      0.13        10
           8       0.00      0.00      0.00         5
           9       0.30      0.78      0.44         9
          10       0.00      0.00      0.00         3
          11       0.00      0.00      0.00         4
          12       0.29      0.18      0.22        11
          13       0.00      0.00      0.00         6
          14       0.00      0.00      0.00         0
          15       0.00      0.00      0.00         1
```

```python
from sklearn.tree import DecisionTreeClassifier

dct=DecisionTreeClassifier(criterion='entropy')
dct.fit(train_x,train_y)
preddct=dct.predict(test_x)


print(accuracy_score(preddct,test_y))
print(confusion_matrix(test_y,preddct))
print(classification_report(test_y,preddct))
```

```
0.8652694610778443
[[18  0  0 ...  0  0  0]
 [ 0  7  0 ...  0  0  0]
 [ 0  0 13 ...  0  0  0]
 ...
 [ 0  0  0 ...  2  0  0]
 [ 0  0  0 ...  0  5  0]
 [ 0  0  0 ...  0  0  2]]
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        20
           1       0.58      1.00      0.74         7
           2       0.93      0.87      0.90        15
           3       0.89      1.00      0.94        16
           4       1.00      0.82      0.90        11
           5       0.89      0.89      0.89         9
           6       1.00      0.92      0.96        13
           7       0.67      1.00      0.80        10
           8       1.00      1.00      1.00         5
           9       0.89      0.89      0.89         9
          10       0.00      0.00      0.00         3
          11       0.80      1.00      0.89         4
          12       1.00      0.91      0.95        11
          13       0.86      1.00      0.92         6
          15       0.50      1.00      0.67         1
          16       0.50      0.33      0.40         3
```

Conclusion

```
#From above we can see that svc model has the best score, so we save this model
```

```
import pickle



# Save to file in the current working directory
pkl_filename = "pickle_model.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(svc, file)

# Load from file
with open(pkl_filename, 'rb') as file:
    pickle_model = pickle.load(file)

# Calculate the accuracy score and predict target values
score = pickle_model.score(test_x, test_y)
print("Test score: {0:.2f} %".format(100 * score))
Ypredict = pickle_model.predict(test_x)
```

```
Test score: 94.61 %
```

```
import pickle

filename='picklesvcfile.pkl'
pickle.dump(svc, open(filename, 'wb'))

#load the model from disk

loaded_model=pickle.load(open(filename, 'rb'))

loaded_model.predict(test_x)
```

```
array([42, 13, 34, 50, 19, 12, 15,  1, 31, 31,  2,  0,  7, 34,  2,  1, 26,
        4, 41,  9,  5,  9, 31,  6, 11, 12, 17, 22, 20,  1, 44, 12, 37,  3,
       43,  0,  0, 42, 46,  3, 33, 44,  6, 47,  4, 19,  0,  0,  1,  7, 12,
        8, 24, 22, 26,  0,  4, 26, 44,  0,  4, 39,  0, 47, 48,  3, 19, 26,
       31, 34, 38, 47,  2,  4,  0, 34, 38, 46, 26, 18, 47, 50, 26, 44, 34,
       44, 44, 42,  3,  0, 46, 22, 24,  3,  2,  9, 46, 44, 28, 42, 47, 16,
        6,  3, 29,  6, 37, 46, 29,  2, 17, 37, 10, 26, 12, 33, 12, 12, 26,
        9, 31, 36, 36,  7, 26, 49,  2, 35, 32,  6, 31, 33,  8, 17, 17, 30,
       34, 36,  1, 31, 36, 31,  7,  6, 37, 12,  0,  5, 21, 31, 44, 32,  7,
       44,  6, 27,  5, 12,  0,  9,  3,  2,  3,  4, 50,  9, 11,  4,  3, 40,
        0, 33, 33, 42, 49, 25,  6, 19, 42, 49, 17, 26,  7, 25, 44, 43, 37,
       11,  9,  3,  1,  0,  4,  0,  2, 34,  4, 47,  9,  3, 25, 28, 45, 17,
       19, 31, 32,  2, 28, 12, 40, 17,  6, 17,  7, 13, 50,  1, 37,  1,  2,
       44,  6, 23, 29, 31,  2,  6,  3, 41,  6,  0,  0,  0,  1,  7, 20, 45,
       25,  5, 32, 44, 12,  4,  5, 25, 22, 49, 47, 44, 45,  0, 39,  3, 31,
       34, 41, 48,  3, 18, 47,  8,  7, 28,  0, 31,  3,  9,  4,  3, 17, 44,
       47,  2, 18, 49, 18,  4, 25,  7,  2, 50, 36, 31, 29,  5, 38, 17, 32,
       34, 16,  2, 26,  0, 43, 17, 26, 16, 38, 13, 13,  1, 44,  7,  5, 22,
       35, 19, 38, 17,  5, 49, 19, 38, 10,  3, 47,  8, 11,  2, 23, 26,  8,
       39, 13, 47, 33, 13, 42,  2, 47, 23,  6, 38])
```

SVC is the best model for performance where I got **94.61%** of accuracy among of the entire performing machine learning model.