# Hrishikesh Rajesh Menon COE18B024 Problem-Set 3

## QUESTION 1

**Suppose that the data for analysis includes the attribute age. The age values for the data tuples are (in increasing order) 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.**

## Modules Used:-

### Numpy

### Pandas

## Approach:-

- Input the age details into a numpy array
- Preprocessing data
- Using Numpy Methods to find min max normalisation and z score
- Store all the values in a pandas DataFrame

In [1]:
```
import pandas as pan
import numpy as nps
import random
import matplotlib.pyplot as plt

age = nps.array([13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70])

# Min Max
age_min = min(age)
age_max = max(age)
min_max_age = [(i-age_min)/(age_max-age_min) for i in age]

#Z score
mean_age = nps.mean(age)
std_age = nps.std(age)

z_score_age = [(i-mean_age)/std_age for i in age]

#as we can see that the dataset has at max 2 digit numbers, we can easily realize that j = 2
decimal_scale_age = [i/100 for i in age]

Output = pan.DataFrame({"Age":age , "Min-Max":min_max_age , "Z-Score":z_score_age, "Decimal Scale":decimal_scale_age})
```

In [2]:

```
display(Output)
```

| | Age | Min-Max | Z-Score | Decimal Scale |
|---|---|---|---|---|
| **0** | 13 | 0.000000 | -1.335646 | 0.13 |
| **1** | 15 | 0.035088 | -1.178168 | 0.15 |
| **2** | 16 | 0.052632 | -1.099429 | 0.16 |
| **3** | 16 | 0.052632 | -1.099429 | 0.16 |
| **4** | 19 | 0.105263 | -0.863212 | 0.19 |
| **5** | 20 | 0.122807 | -0.784473 | 0.20 |
| **6** | 20 | 0.122807 | -0.784473 | 0.20 |
| **7** | 21 | 0.140351 | -0.705734 | 0.21 |
| **8** | 22 | 0.157895 | -0.626995 | 0.22 |
| **9** | 22 | 0.157895 | -0.626995 | 0.22 |
| **10** | 25 | 0.210526 | -0.390779 | 0.25 |
| **11** | 25 | 0.210526 | -0.390779 | 0.25 |
| **12** | 25 | 0.210526 | -0.390779 | 0.25 |
| **13** | 25 | 0.210526 | -0.390779 | 0.25 |
| **14** | 30 | 0.298246 | 0.002916 | 0.30 |
| **15** | 33 | 0.350877 | 0.239133 | 0.33 |
| **16** | 33 | 0.350877 | 0.239133 | 0.33 |
| **17** | 35 | 0.385965 | 0.396611 | 0.35 |
| **18** | 35 | 0.385965 | 0.396611 | 0.35 |
| **19** | 35 | 0.385965 | 0.396611 | 0.35 |
| **20** | 35 | 0.385965 | 0.396611 | 0.35 |
| **21** | 36 | 0.403509 | 0.475350 | 0.36 |
| **22** | 40 | 0.473684 | 0.790306 | 0.40 |
| **23** | 45 | 0.561404 | 1.184001 | 0.45 |
| **24** | 46 | 0.578947 | 1.262740 | 0.46 |
| **25** | 52 | 0.684211 | 1.735173 | 0.52 |
| **26** | 70 | 1.000000 | 3.152474 | 0.70 |

# QUESTION 2

## Dataset Description

**It is a well-known fact that Millenials LOVE Avocado Toast. It's also a well known fact that all Millenials live in their parents basements.**

**Clearly, they aren't buying home because they are buying too much Avocado Toast!**

**But maybe there's hope... if a Millenial could find a city with cheap avocados, they could live out the Millenial American Dream. Help them to filter out the clutter using some pre-processing techniques.**

### Some relevant columns in the dataset:

- Date - The date of the observation
- Average Price - 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

**(Product Lookup codes (PLU's))**

## Modules Used:-

### Numpy

### Pandas

### Matplotlib

## Approach:-

- Input the dataset using pandas
- a.By
  - Creating a list Bins to store the values in each bin of size 250
  - Using Numpy methods mean and median to smooth the bins and by using boundary formula to smooth by boundary
- b.Using loc and str we can take the weekly sales and convert to Monthly and Annual sales
- c.Using isnull() along with sum() to calculate the number of null/missing values in each attribute
- d.By using illoc we can see which row/tuple have the same region so we can assign it to that regions Average Price Mean
- e.By using illoc to take the year of each tuple we can discretize the data into the concept hierachy

In [90]:
```
#Read the dataset

Ava_data= pan.read_csv('Avocado Dataset.csv')

Ava_data.head()
```

Out[90]:

| | Date | AveragePrice | Total Volume | 4046 | 4225 | 4770 | Total Bags | Small Bags | Large Bags | XLarge Bags | type | year | region |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 27-12-2015 | 1.33 | 64236.62 | 1036.74 | 54454.85 | 48.16 | 8696.87 | 8603.62 | 93.25 | 0.0 | conventional | 2015 | Albany |
| **1** | 20-12-2015 | 1.35 | 54876.98 | 674.28 | 44638.81 | 58.33 | 9505.56 | 9408.07 | 97.49 | 0.0 | conventional | 2015 | Albany |
| **2** | 13-12-2015 | 0.93 | 118220.22 | 794.70 | 109149.67 | 130.50 | 8145.35 | 8042.21 | 103.14 | 0.0 | conventional | 2015 | Albany |
| **3** | 06-12-2015 | 1.08 | 78992.15 | 1132.00 | 71976.41 | 72.58 | 5811.16 | 5677.40 | 133.76 | 0.0 | conventional | 2015 | Albany |
| **4** | 29-11-2015 | 1.29 | 51039.60 | 941.48 | 43838.39 | 75.78 | 6183.95 | 5986.26 | 197.69 | 0.0 | conventional | 2015 | Albany |

a. Sort the attribute “Total Volume” in the given dataset and distribute the data into equal sized/frequency bins. Let the number of bins be 250. Smooth the sorted data by

- bin-means
- bin-medians
- bin-boundaries

In [91]:
```
#Store Total Volume in a list
Tot_Vol=list(Ava_data['Total Volume'])

#Sort
Tot_Vol.sort()
print(Tot_Vol[:10])
```

```
[84.56, 379.82, 385.55, 419.98, 472.82, 482.26, 515.01, 530.96, 542.85, 561.1]
```

In [92]:
```
#Create bins of size 250

Bins=[]
size=250

for i in range( int(len(Tot_Vol)/size) ):
    Bins.append( Tot_Vol[size*i: size*i+size] )

print("Bin Size:-",len(Bins[0]))
print("First Bin:-",Bins[0])
```

```
Bin Size:- 250
First Bin:- [84.56, 379.82, 385.55, 419.98, 472.82, 482.26, 515.01, 530.96, 542.85, 561.1, 562.64, 563.06, 566.57, 571.4, 58
8.87, 593.39, 614.3, 627.8, 634.09, 657.42, 657.96, 667.95, 679.51, 683.76, 697.76, 706.15, 711.52, 712.4, 716.29, 740.33, 7
49.03, 753.64, 753.78, 761.86, 762.15, 769.05, 774.2, 775.8, 787.72, 792.97, 795.95, 809.41, 810.61, 814.13, 821.99, 823.5,
831.69, 841.53, 845.05, 845.11, 846.47, 849.44, 852.98, 858.83, 862.59, 867.18, 872.07, 876.35, 879.94, 881.14, 887.29, 892.
02, 894.87, 896.16, 897.77, 901.53, 902.5, 908.95, 911.96, 924.46, 929.61, 930.74, 934.95, 936.69, 937.64, 940.43, 950.14, 9
56.97, 961.36, 964.25, 969.29, 971.81, 974.27, 979.74, 980.98, 989.55, 991.33, 992.32, 994.05, 995.96, 996.79, 998.28, 1000.
86, 1005.21, 1005.8, 1007.03, 1007.51, 1011.16, 1012.61, 1014.02, 1015.04, 1015.29, 1016.97, 1021.68, 1023.37, 1023.51, 102
8.63, 1030.98, 1034.32, 1035.0, 1038.67, 1041.31, 1043.41, 1047.42, 1047.82, 1052.37, 1053.73, 1054.07, 1057.26, 1059.21, 10
59.69, 1062.54, 1069.67, 1070.6, 1076.23, 1079.0, 1081.18, 1081.74, 1082.44, 1082.52, 1083.03, 1083.77, 1083.8, 1095.77, 109
8.66, 1098.67, 1100.1, 1106.24, 1106.41, 1106.86, 1108.9, 1111.9, 1112.77, 1115.89, 1117.32, 1117.44, 1118.47, 1118.55, 112
2.69, 1130.91, 1137.03, 1140.32, 1140.95, 1141.59, 1145.5, 1145.88, 1146.07, 1151.64, 1152.33, 1153.0, 1155.28, 1155.71, 115
5.99, 1156.05, 1156.12, 1158.22, 1158.42, 1159.5, 1161.01, 1161.9, 1162.28, 1162.68, 1163.03, 1163.67, 1164.87, 1168.86, 116
9.26, 1169.48, 1169.94, 1170.01, 1174.53, 1175.34, 1175.95, 1178.2, 1179.16, 1180.38, 1181.37, 1182.3, 1182.56, 1186.44, 118
7.71, 1187.99, 1188.82, 1190.83, 1193.06, 1196.53, 1200.0, 1200.13, 1200.48, 1200.59, 1200.61, 1201.07, 1201.46, 1202.46, 12
03.52, 1206.53, 1208.54, 1211.27, 1212.71, 1219.02, 1219.69, 1221.24, 1221.36, 1222.54, 1223.94, 1227.21, 1228.51, 1228.73,
1233.27, 1233.92, 1235.29, 1237.02, 1238.66, 1238.86, 1239.16, 1241.24, 1241.84, 1243.29, 1243.4, 1247.8, 1248.89, 1249.75,
1250.53, 1252.86, 1253.46, 1253.5, 1254.81, 1256.47, 1258.62, 1259.46, 1259.86, 1263.25, 1267.44, 1269.64, 1269.74, 1272.64,
1274.53, 1275.64, 1278.14, 1279.45]
```

In [93]:
```
#Bin by mean

Mean_Bins = []

for bin_ in Bins:
    mean = nps.array(bin_).mean()
    Mean_Bins.append([mean]*size) #To put the mean for all values of the bin

print("First Bin:-",Mean_Bins[0])
```

```
First Bin:- [1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 101
7.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19
488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488, 1017.19488,
1017.19488, 1017.19488, 1017.19488]
```

In [94]:

```
#Similarily,Bin by median

Median_Bins = []

for bin_ in Bins:
    median = nps.median(nps.array(bin_))
    Median_Bins.append([median]*size) #To put the mean for all values of the bin

print("First Bin:-",Median_Bins[0])
```

```
First Bin:- [1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1
077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.
615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615,
1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 107
7.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.61
5, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1
077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.
615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615,
1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 107
7.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.61
5, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1
077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.
615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615,
1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 107
7.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.61
5, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1
077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.
615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615,
1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 107
7.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.615, 1077.61
5, 1077.615, 1077.615, 1077.615]
```

In [95]:

```
#Bin by Boundary

Boundary_Bins = []
for bin_ in Bins:
    bins = []
    for val in bin_:
        if((val-bin_[0])<=(bin_[249]-val)): #Append Closest Boundary For each value
            bins.append(bin_[0])
        else:
            bins.append(bin_[249])
    Boundary_Bins.append(bins)

print("First Bin:\n",Boundary_Bins[0])
```

```
First Bin:
 [84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.56, 84.
56, 84.56, 84.56, 84.56, 84.56, 84.56, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 127
9.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1
279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45,
1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.4
5, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 127
9.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1
279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45,
1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.4
5, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 127
9.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1
279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45,
1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.4
5, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 127
9.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1
279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45,
1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.4
5, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45, 1279.45]
```

b. The dataset represents weekly retail scan data for National retail volume (units) and price. However, the company is interested in knowing the monthly (total per month) and annual sales (total per year), rather than the total per week. So, reduce the data accordingly.

In [96]:
```
#Monthly Sales

dates = Ava_data.loc[:,'Date']
months = list(set([i[3:] for i in dates]))

AveragePrice = pan.to_numeric(Ava_data.iloc[:,1],errors='coerce')
Ava_data['AveragePrice'] = AveragePrice

month = []
AvgPrice = []
TotVol = []
region = []

for i in months:
    monthly = Ava_data.loc[Ava_data['Date'].str.contains("[0-9][0-9]-"+i,na=False)].groupby('region').sum()
    month.extend([i]*len(monthly.index))
    AvgPrice.extend(list(monthly['AveragePrice']))
    TotVol.extend(list(monthly['Total Volume']))
    region.extend(list(monthly.index.values))

MonthlyData = pan.DataFrame.from_dict({'Month':month,'region':region,'Avg Price':AvgPrice,"Total Volume":TotVol})
MonthlyData.head()
```

Out[96]:

| | Month | region | Avg Price | Total Volume |
|---|---|---|---|---|
| **0** | 07-2017 | Albany | 16.56 | 483870.08 |
| **1** | 07-2017 | Atlanta | 14.79 | 2701116.40 |
| **2** | 07-2017 | BaltimoreWashington | 16.81 | 3565763.00 |
| **3** | 07-2017 | Boise | 16.89 | 451414.21 |
| **4** | 07-2017 | Boston | 16.88 | 3072344.23 |

In [97]:
```
#Annual Sales
dates = Ava_data.loc[:,'Date']
years = list(set([i[6:] for i in dates]))

AveragePrice = pan.to_numeric(Ava_data.iloc[:,1],errors='coerce')
Ava_data['AveragePrice'] = AveragePrice

year = []
AvgPrice = []
TotVol = []
region = []

for i in years:
    yearly = Ava_data.loc[Ava_data['Date'].str.contains("[0-9][0-9]-[0-9][0-9]-"+i,na=False)].groupby('region').sum()
    year.extend([i]*len(yearly.index))
    AvgPrice.extend(list(yearly['AveragePrice']))
    TotVol.extend(list(yearly['Total Volume']))
    region.extend(list(yearly.index.values))

YearlyData = pan.DataFrame.from_dict({'Year':year,'region':region,'Avg Price':AvgPrice,"Total Volume":TotVol})
YearlyData.head()
```

Out[97]:

| | Year | region | Avg Price | Total Volume |
|---|---|---|---|---|
| **0** | 2015 | Albany | 152.32 | 4029896.43 |
| **1** | 2015 | Atlanta | 143.58 | 23231698.12 |
| **2** | 2015 | BaltimoreWashington | 134.21 | 40645579.54 |
| **3** | 2015 | Boise | 137.36 | 3784357.34 |
| **4** | 2015 | Boston | 137.11 | 27454991.64 |

c. Summarize the number of missing values for each attribute

In [98]:
```
Ava_data.isnull().sum()
```

Out[98]:
```
Date             0
AveragePrice    48
Total Volume     0
4046             0
4225             0
4770             0
Total Bags       0
Small Bags       0
Large Bags       0
XLarge Bags      0
type             0
year             0
```

```
region          0
dtype: int64
```

d. Populate data for the missing values of the attribute= “Average Price” by averaging all the values of the “Avg Price” attribute that fall under the same “REGION” attribute value.

In [99]:
```
for i in range(len(Ava_data)):
    if(nps.isnan(Ava_data.iloc[i]['AveragePrice'])):
        Ava_data.iloc[i,1]=Ava_data[Ava_data['region']==Ava_data.iloc[i]['region']]['AveragePrice'].mean()

#To show there is no more missing values
Ava_data.isnull().sum()
```

Out[99]:
```
Date            0
AveragePrice    0
Total Volume    0
4046            0
4225            0
4770            0
Total Bags      0
Small Bags      0
Large Bags      0
XLarge Bags     0
type            0
year            0
region          0
dtype: int64
```

e. Discretize the attribute= “Date” using concept hierarchy into {Old, New, Recent} (Consider 2015,2016 : Old, 2017: New, 2018: Recent).

In [100…
```
DiscreteDate = []
for i in range(len(Ava_data)):
    year = Ava_data.iloc[i,0][6:]
    if(int(year)<=2016):
        DiscreteDate.append('Old')
    elif(int(year)==2017):
        DiscreteDate.append('New')
    elif(int(year)==2018):
        DiscreteDate.append('Recent')
    else:
        DiscreteDate.append(nps.nan)

Ava_data = Ava_data.drop(['Date'],axis=1)
Ava_data.insert(0,"Date",DiscreteDate)

Ava_data
```

Out[100…

| | Date | AveragePrice | Total Volume | 4046 | 4225 | 4770 | Total Bags | Small Bags | Large Bags | XLarge Bags | type | year | region |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Old | 1.33 | 64236.62 | 1036.74 | 54454.85 | 48.16 | 8696.87 | 8603.62 | 93.25 | 0.0 | conventional | 2015 | Albany |
| **1** | Old | 1.35 | 54876.98 | 674.28 | 44638.81 | 58.33 | 9505.56 | 9408.07 | 97.49 | 0.0 | conventional | 2015 | Albany |
| **2** | Old | 0.93 | 118220.22 | 794.70 | 109149.67 | 130.50 | 8145.35 | 8042.21 | 103.14 | 0.0 | conventional | 2015 | Albany |
| **3** | Old | 1.08 | 78992.15 | 1132.00 | 71976.41 | 72.58 | 5811.16 | 5677.40 | 133.76 | 0.0 | conventional | 2015 | Albany |
| **4** | Old | 1.29 | 51039.60 | 941.48 | 43838.39 | 75.78 | 6183.95 | 5986.26 | 197.69 | 0.0 | conventional | 2015 | Albany |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **18245** | Recent | 1.71 | 13888.04 | 1191.70 | 3431.50 | 0.00 | 9264.84 | 8940.04 | 324.80 | 0.0 | organic | 2018 | WestTexNewMexico |
| **18246** | Recent | 1.87 | 13766.76 | 1191.92 | 2452.79 | 727.94 | 9394.11 | 9351.80 | 42.31 | 0.0 | organic | 2018 | WestTexNewMexico |
| **18247** | Recent | 1.93 | 16205.22 | 1527.63 | 2981.04 | 727.01 | 10969.54 | 10919.54 | 50.00 | 0.0 | organic | 2018 | WestTexNewMexico |
| **18248** | Recent | 1.62 | 17489.58 | 2894.77 | 2356.13 | 224.53 | 12014.15 | 11988.14 | 26.01 | 0.0 | organic | 2018 | WestTexNewMexico |
| **18249** | Recent | 1.56 | 15896.38 | 2055.35 | 1499.55 | 0.00 | 12341.48 | 12114.81 | 226.67 | 0.0 | organic | 2018 | WestTexNewMexico |

18250 rows × 13 columns