PROBLEM SET 3, MRIDUL HARISH, CED18I034

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.

(a) Use min-max normalization to transform the values of age to the range [0:1].

(b) Use z-score normalization to transform the values of age.

(c) Use normalization by decimal scaling to transform the values of age such that the transformed value is less than 1.

In [ ]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
```

In [ ]:
```
Age = np.array([13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 3
```

(i) Min-Max

In [ ]:
```
Age_min = min(Age)
Age_max = max(Age)
Age_min_max = [(i - Age_min)/(Age_max - Age_min) for i in Age]
```

In [ ]:
```
Age_min
```

Out[ ]:
```
13
```

In [ ]:
```
Age_max
```

Out[ ]:
```
70
```

In [ ]:
```
Age_min_max
```

Out[ ]:
```
[0.0,
 0.03508771929824561,
 0.05263157894736842,
 0.05263157894736842,
 0.10526315789473684,
 0.12280701754385964,
 0.12280701754385964,
 0.14035087719298245,
 0.15789473684210525,
 0.15789473684210525,
 0.21052631578947367,
 0.21052631578947367,
 0.21052631578947367,
```

```
0.21052631578947367,
0.2982456140350877,
0.3508771929824561,
0.3508771929824561,
0.38596491228070173,
0.38596491228070173,
0.38596491228070173,
0.38596491228070173,
0.40350877192982454,
0.47368421052631576,
0.5614035087719298,
0.5789473684210527,
0.6842105263157895,
```

In [ ]:
```
Age_mean = np.mean(Age)
Age_std = np.std(Age)
```

(ii)Z score

In [ ]:
```
Age_mean
```

Out[ ]:
```
29.962962962962962
```

In [ ]:
```
Age_std
```

Out[ ]:
```
12.700193878606099
```

In [ ]:
```
Age_Zscore = [(i - Age_mean)/Age_std for i in Age]
```

In [ ]:
```
Age_Zscore
```

Out[ ]:
```
[-1.3356459850221374,
 -1.1781680741243308,
 -1.0994291186754275,
 -1.0994291186754275,
 -0.8632122523287176,
 -0.7844732968798143,
 -0.7844732968798143,
 -0.705734341430911,
 -0.6269953859820077,
 -0.6269953859820077,
 -0.39077851963529775,
 -0.39077851963529775,
 -0.39077851963529775,
 -0.39077851963529775,
 0.0029162576092187234,
 0.23913312395592862,
 0.23913312395592862,
 0.3966110348537352,
 0.3966110348537352,
 0.3966110348537352,
 0.3966110348537352,
 0.4753499903026385,
 0.7903058120982517,
 1.1840005893427683,
 1.2627395447916716,
```

```
1.7351732774850914,
```

We can see that here that the dataset has at max 2 digit numbers, we can easily realize that j = 2

In [ ]:
```
Age_decimalscale = [i/100 for i in Age]
```

In [ ]:
```
Age_decimalscale
```

Out[ ]:
```
[0.13,
 0.15,
 0.16,
 0.16,
 0.19,
 0.2,
 0.2,
 0.21,
 0.22,
 0.22,
 0.25,
 0.25,
 0.25,
 0.25,
 0.3,
 0.33,
 0.33,
 0.35,
 0.35,
 0.35,
 0.35,
 0.36,
 0.4,
 0.45,
 0.46,
 0.52,
 0.7]
```

In [ ]:
```
Output = pd.DataFrame({"Age":Age, "Min-Max":Age_min_max, "Z-score":Age_Zscore,
```

In [ ]:
```
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

	Age	Min-Max	Z-score	Decimal Scale
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

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)) *

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

(i)bin-means

(ii) bin-medians

(iii) bin-boundaries

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.

c. Summarize the number of missing values for each attribute

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.

e. Discretize the attribute= "Date" using concept hierarchy into {Old, New, Recent} (Consider

2015,2016 : Old, 2017: New, 2018: Recent).

In [ ]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
```

In [ ]:
```
Avacado_data = pd.read_csv('Avocado Dataset.csv')
```

In [ ]:
```
Avacado_data.head()
```

Out[ ]:

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

In [ ]:
```
Total_Volume = list(Avacado_data['Total Volume'])
```

In [ ]:
```
Total_Volume = sorted(Total_Volume)
```

In [ ]:
```
print(Total_Volume[:10])
```

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

In [ ]:
```
Bins = []
Size = 250

for i in range(int(len(Total_Volume)/Size)):
    Bins.append(Total_Volume[Size*i: Size*i + Size] )
```

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

```
Bin Size:  250
```

In [ ]:
```
Mean_Bins = []

for i in Bins:
    Mean = np.array(i).mean()
    Mean_Bins.append([Mean]*Size)
```

In [ ]:
```
print("Mean Bin Size: ", len(Mean_Bins[0]))
```

```
Mean Bin Size:  250
```

In [ ]:
```
Median_Bins = []

for j in Bins:
    Median = np.median(np.array(j))
    Median_Bins.append([Median]*Size)
```

In [ ]:
```
print("Median Bin Size: ", len(Median_Bins[0]))
```

```
Median Bin Size:  250
```

In [ ]:
```
Boundary_Bins = []

for k in Bins:
    bins = []
    for val in k:
        if((val - k[0])<=(k[249] - val)):
            bins.append(k[0])
        else:
            bins.append(k[249])
    Boundary_Bins.append(bins)
```

In [ ]:
```
print("Boundary Bin Size: ", len(Boundary_Bins[0]))
```

```
Boundary Bin Size:  250
```

In [ ]:
```
Dates = Avacado_data.loc[:,'Date']
Months = list(set([i[3:] for i in Dates]))
```

In [ ]:
```
Dates
```

Out[ ]:
```
0        27-12-2015
1        20-12-2015
2        13-12-2015
3        06-12-2015
4        29-11-2015
            ...
18245    28-01-2018
18246    21-01-2018
18247    14-01-2018
18248    07-01-2018
18249    18-03-2018
Name: Date, Length: 18250, dtype: object
```

In [ ]:
```
Months
```

Out[ ]:
```
['03-2018',
 '01-2018',
 '02-2015',
```

```
'03-2015',
'03-2016',
'10-2017',
'12-2016',
'05-2015',
'01-2015',
'10-2016',
'12-2017',
'08-2017',
'02-2016',
'06-2015',
'11-2017',
'08-2015',
'07-2016',
'09-2016',
'01-2017',
'05-2017',
'02-2017',
'12-2015',
'06-2016',
'07-2017',
'05-2016',
'04-2015',
'11-2016',
'09-2017',
'09-2015',
'03-2017',
'02-2018',
'04-2017',
'01-2016',
'08-2016',
'06-2017',
'04-2016',
'11-2015',
'07-2015',
```

In [ ]:

```
AveragePrice = pd.to_numeric(Avacado_data.iloc[:,1], errors='coerce')
Avacado_data['AveragePrice']

Month = []
AvgPrice = []
TotVolume =[]
Region = []

for i in Months:
    Monthly = Avacado_data.loc[Avacado_data['Date'].str.contains("[0-9][0-9]-'
    Month.extend([i]*len(Monthly.index))
    AvgPrice.extend(list(Monthly['AveragePrice']))
    TotVolume.extend(list(Monthly['Total Volume']))
    Region.extend(list(Monthly.index.values))
```

In [ ]:

```
MonthlyData = pd.DataFrame.from_dict({'Month':Month,'region':Region, 'Averagel
MonthlyData.head()
```

Out[ ]:

	Month	region	AveragePrice	Total Volume
0	03-2018	Albany	11.65	550768.59
1	03-2018	Atlanta	10.16	2723324.59

	Month	region	AveragePrice	Total Volume
2	03-2018	BaltimoreWashington	10.61	4157091.43
3	03-2018	Boise	12.04	388664.58

In [ ]:
```
Dates = Avacado_data.loc[:,'Date']
Years = list(set([i[6:] for i in dates]))

AveragePrice = pd.to_numeric(Avacado_data.iloc[:,1],errors='coerce')
Avacado_data['AveragePrice'] = AveragePrice

Year = []
AvgPrice = []
TotVolume = []
Region = []

for i in Years:
    Yearly = Avacado_data.loc[Avacado_data['Date'].str.contains("[0-9][0-9]-[(
    Year.extend([i]*len(Yearly.index))
    AvgPrice.extend(list(Yearly['AveragePrice']))
    TotVolume.extend(list(Yearly['Total Volume']))
    Region.extend(list(Yearly.index.values))

YearlyData = pd.DataFrame.from_dict({'Year':year,'region':region,'Avg Price':A
YearlyData.head()
```

Out[ ]:

	Year	region	Avg Price	Total Volume
0	2016	Albany	159.53	5264335.59
1	2016	Atlanta	126.27	28326878.30
2	2016	BaltimoreWashington	165.11	40893802.32
3	2016	Boise	118.76	4653509.53
4	2016	Boston	148.32	30571315.07

In [ ]:
```
Avacado_data.isnull().sum()
```

Out[ ]:
```
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
```

In [ ]:
```
for i in range(len(Avacado_data)):
    if(np.isnan(Avacado_data.iloc[i]['AveragePrice'])):
        Avacado_data.iloc[i,1]=Avacado_data[Avacado_data['region']==Avacado_da
```

In [ ]:
```
Avacado_data.isnull().sum()
```

Out[ ]:
```
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
```

In [ ]:
```
Discrete_Date = []
for i in range(len(Avacado_data)):
    Year = Avacado_data.iloc[i,0][6:]
    if(int(Year)<=2016):
        Discrete_Date.append('Old')
    elif(int(Year)==2017):
        Discrete_Date.append('New')
    elif(int(Year)==2018):
        Discrete_Date.append('Recent')
    else:
        Discrete_Date.append(np.nan)
```

In [ ]:
```
Avacado_data = Avacado_data.drop(['Date'],axis=1)
Avacado_data.insert(0,"Date",Discrete_Date)
```

In [ ]:
```
Avacado_data
```

Out[ ]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarg Bag
0	Old	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0
1	Old	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0
2	Old	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0
3	Old	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0
4	Old	1.29	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0
...	...	...	...	...	...	...	...	...	...	
18245	Recent	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarg Bag
18246	Recent	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0
18247	Recent	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0
18248	Recent	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0