Name : Ayush Makwana

Roll No : 18BCE107

Practical : 2

Apply Data Smoothing using Binning Methods:

1) Smoothing using Bin Mean 2) By Bin Median 3) bin boundary smoothing

Bin Mean

In [7]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
import statistics
import math
from collections import OrderedDict
```

In [9]:
```
x =[]
print("enter the data")
x = list(map(float, input().split()))

print("enter the number of bins")
bi = int(input())

X_dict = OrderedDict()
x_old ={}
x_new ={}


for i in range(len(x)):
    X_dict[i]= x[i]
    x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])

binn =[]
avrg = 0

i = 0
k = 0
num_of_data_in_each_bin = int(math.ceil(len(x)/bi))
```

```
enter the data
10 20 30 40 50 40 30 20 10
enter the number of bins
3
```

In [10]:
```
# performing binning
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        avrg = avrg + h
        i = i + 1
    elif(i == num_of_data_in_each_bin):
        k = k + 1
        i = 0
        binn.append(round(avrg / num_of_data_in_each_bin, 3))
        avrg = 0
        avrg = avrg + h
        i = i + 1
rem = len(x)% bi
if(rem == 0):
    binn.append(round(avrg / num_of_data_in_each_bin, 3))
else:
    binn.append(round(avrg / rem, 3))

# store the new value of each data
i = 0
j = 0
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        x_new[g]= binn[j]
        i = i + 1
    else:
        i = 0
        j = j + 1
        x_new[g]= binn[j]
        i = i + 1
print("number of data in each bin")
print(math.ceil(len(x)/bi))

for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

```
number of data in each bin
3
index 0 old value 10.0 new value 20.0
index 1 old value 20.0 new value 20.0
```

```
index 2 old value 30.0 new value 20.0
index 3 old value 40.0 new value 43.333
index 4 old value 50.0 new value 43.333
index 5 old value 40.0 new value 43.333
index 6 old value 30.0 new value 20.0
index 7 old value 20.0 new value 20.0
index 8 old value 10.0 new value 20.0
```

Bin Median

In [13]:
```
x =[]
print("enter the data")
x = list(map(float, input().split()))

print("enter the number of bins")
bi = int(input())

# X_dict will store the data in sorted order
X_dict = OrderedDict()
# x_old will store the original data
x_old ={}
# x_new will store the data after binning
x_new ={}
```

```
enter the data
10 20 30 40 50 40 30 20 10
enter the number of bins
3
```

In [14]:
```
for i in range(len(x)):
    X_dict[i]= x[i]
    x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])


# list of lists(bins)
binn =[]
# a variable to find the mean of each bin
avrg =[]

i = 0
k = 0
num_of_data_in_each_bin = int(math.ceil(len(x)/bi))
# performing binning
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        avrg.append(h)
        i = i + 1
    elif(i == num_of_data_in_each_bin):
        k = k + 1
        i = 0
        binn.append(statistics.median(avrg))
        avrg =[]
        avrg.append(h)
        i = i + 1

binn.append(statistics.median(avrg))

# store the new value of each data
i = 0
j = 0
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        x_new[g]= round(binn[j], 3)
        i = i + 1
    else:
        i = 0
        j = j + 1
        x_new[g]= round(binn[j], 3)
        i = i + 1
```

```
print("number of data in each bin")
print(math.ceil(len(x)/bi))
for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

```
number of data in each bin
3
index 0 old value 10.0 new value 20.0
index 1 old value 20.0 new value 20.0
index 2 old value 30.0 new value 20.0
index 3 old value 40.0 new value 40.0
index 4 old value 50.0 new value 40.0
index 5 old value 40.0 new value 40.0
index 6 old value 30.0 new value 20.0
index 7 old value 20.0 new value 20.0
index 8 old value 10.0 new value 20.0
```

Bin Boundry

In [11]:
```
x =[]
print("enter the data")
x = list(map(float, input().split()))

print("enter the number of bins")
bi = int(input())

# X_dict will store the data in sorted order
X_dict = OrderedDict()
# x_old will store the original data
x_old ={}
# x_new will store the data after binning
x_new ={}
```

```
enter the data
10 20 30 40 50 40 30 20 10
enter the number of bins
3
```

In [12]:
```
for i in range(len(x)):
    X_dict[i]= x[i]
    x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])

# list of lists(bins)
binn =[]
# a variable to find the mean of each bin
avrg =[]

i = 0
k = 0
num_of_data_in_each_bin = int(math.ceil(len(x)/bi))

for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        avrg.append(h)
        i = i + 1
    elif(i == num_of_data_in_each_bin):
        k = k + 1
        i = 0
        binn.append([min(avrg), max(avrg)])
        avrg =[]
        avrg.append(h)
        i = i + 1
binn.append([min(avrg), max(avrg)])

i = 0
j = 0

for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        if(abs(h-binn[j][0]) >= abs(h-binn[j][1])):
            x_new[g]= binn[j][1]
            i = i + 1
        else:
            x_new[g]= binn[j][0]
            i = i + 1
    else:
        i = 0
        j = j + 1
```

```
        if(abs(h-binn[j][0]) >= abs(h-binn[j][1])):
            x_new[g]= binn[j][1]
        else:
            x_new[g]= binn[j][0]
        i = i + 1

print("number of data in each bin")
print(math.ceil(len(x)/bi))
for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

```
number of data in each bin
3
index 0 old value 10.0 new value 10.0
index 1 old value 20.0 new value 30.0
index 2 old value 30.0 new value 30.0
index 3 old value 40.0 new value 40.0
index 4 old value 50.0 new value 50.0
index 5 old value 40.0 new value 40.0
index 6 old value 30.0 new value 30.0
index 7 old value 20.0 new value 30.0
index 8 old value 10.0 new value 10.0
```

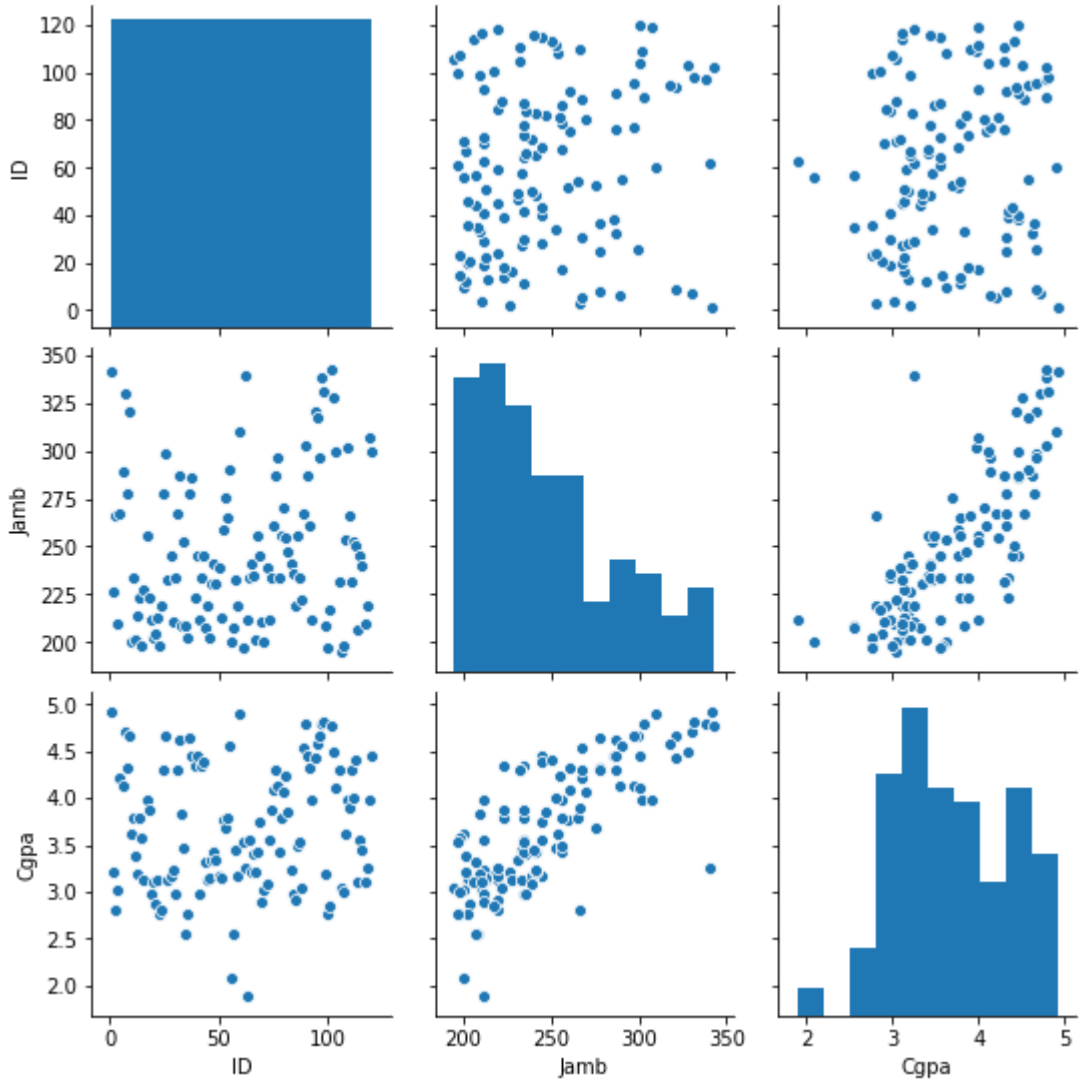
SCHOOL CGPA

In [40]:
```
import matplotlib.pyplot as plt

a = pd.read_csv('school_dataset.csv')
print(a)

bin0 = pd.DataFrame(a)
#plt.hist(a[:,8])
sns.pairplot(bin0)
```

```
      ID    Name Maths Eng Phy Chm Bio  Jamb  Cgpa Place_of_stay  \
0      1    Tola     A   A   A   A   A   342  4.91     in_campus
1      2    kola     C   B   B   C   C   226  3.21    off_campus
2      3   Samad     C   B   C   C   C   266  2.81    off_campus
3      4    Niyi     C   C   B   B   C   210  3.01    off_campus
4      5    Dami     B   A   A   B   C   267  4.21     in_campus
..   ...     ...   ...  ..  ..  ..  ..   ...   ...           ...
115  116     Ayo     B   C   C   C   B   240  3.44     in_campus
116  117    Sola     C   C   B   C   C   210  3.10    off_campus
117  118  Sikemi     A   C   C   C   C   219  3.26    off_campus
118  119    Dele     C   A   A   A   C   307  3.98    off_campus
119  120   Ademi     B   A   A   A   C   300  4.46    off_campus

       Parent_status Gender Pstatus Guardian Romantic_rel Internet  \
0        Academician      F       T   Mother           No      Yes
1    Non_academician      M       T   Father          Yes      Yes
2    Non_academician      M       T   Father          Yes      Yes
3    Non_academician      M       T   Father          Yes      Yes
4        Academician      F       T   Mother           No      Yes
..               ...    ...     ...      ...          ...      ...
115  Non_academician      M       T   Father           No      Yes
116  Non_academician      M       A   Mother           No      Yes
117  Non_academician      F       T   Mother           No      Yes
118      Academician      M       T   Mother           No      Yes
119      Academician      F       T   Mother           No      Yes

    Home_address
0          Urban
1          Rural
2          Rural
3          Rural
4          Urban
..           ...
```

```
115        Urban
116        Urban
117        Urban
118        Urban
119        Urban

[120 rows x 17 columns]
```

Out[40]: <seaborn.axisgrid.PairGrid at 0x27a825da148>



In [41]:
```
a=a.to_numpy()
b = np.zeros(120)
```

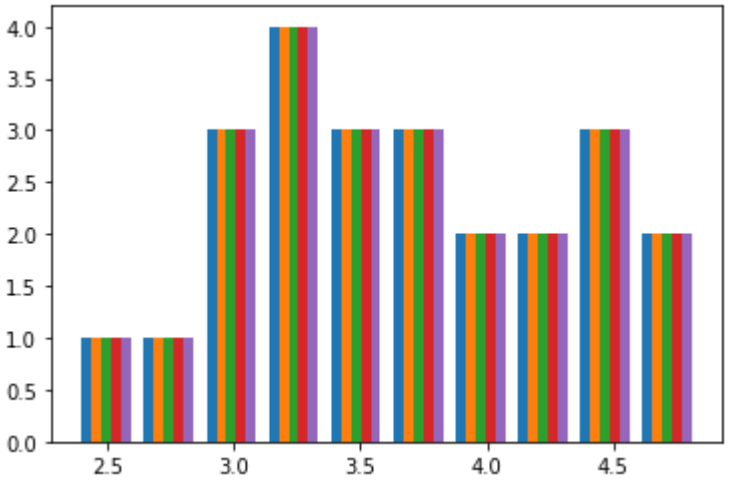
Bin Mean

In [32]:
```
# take 1st column among 4 column of data set
for i in range (120):
    b[i]=a[i,8]

b=np.sort(b)
#print(b)

bin1=np.zeros((24,5))
bin2=np.zeros((24,5))
bin3=np.zeros((24,5))

# Bin mean
for i in range (0,120,5):
    k=int(i/5)
    mean=(b[i] + b[i+1] + b[i+2] + b[i+3] + b[i+4])/5
    #print(mean)
    for j in range(5):
        bin1[k,j]=mean
print("Bin Mean: \n\n",bin1)
plt.hist(bin1)
#bin0 = pd.DataFrame(bin1)
#sns.pairplot(bin0)
```

```
Bin Mean:

 [[2.372 2.372 2.372 2.372 2.372]
 [2.8   2.8   2.8   2.8   2.8  ]
 [2.928 2.928 2.928 2.928 2.928]
 [3.    3.    3.    3.    3.   ]
 [3.072 3.072 3.072 3.072 3.072]
 [3.122 3.122 3.122 3.122 3.122]
 [3.16  3.16  3.16  3.16  3.16 ]
 [3.204 3.204 3.204 3.204 3.204]
 [3.262 3.262 3.262 3.262 3.262]
 [3.382 3.382 3.382 3.382 3.382]
 [3.442 3.442 3.442 3.442 3.442]
 [3.536 3.536 3.536 3.536 3.536]
 [3.614 3.614 3.614 3.614 3.614]
 [3.77  3.77  3.77  3.77  3.77 ]
 [3.842 3.842 3.842 3.842 3.842]
 [3.962 3.962 3.962 3.962 3.962]
 [4.076 4.076 4.076 4.076 4.076]
```

```
[4.234 4.234 4.234 4.234 4.234]
[4.312 4.312 4.312 4.312 4.312]
[4.38  4.38  4.38  4.38  4.38 ]
[4.462 4.462 4.462 4.462 4.462]
[4.582 4.582 4.582 4.582 4.582]
[4.698 4.698 4.698 4.698 4.698]
[4.836 4.836 4.836 4.836 4.836]]
```

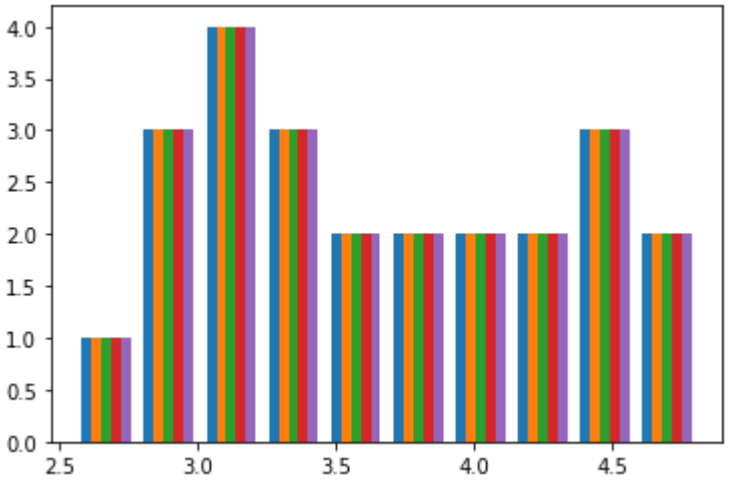
Out[32]:
```
([array([1., 1., 3., 4., 3., 3., 2., 2., 3., 2.]),
  array([1., 1., 3., 4., 3., 3., 2., 2., 3., 2.]),
  array([1., 1., 3., 4., 3., 3., 2., 2., 3., 2.]),
  array([1., 1., 3., 4., 3., 3., 2., 2., 3., 2.]),
  array([1., 1., 3., 4., 3., 3., 2., 2., 3., 2.])],
 array([2.372 , 2.6184, 2.8648, 3.1112, 3.3576, 3.604 , 3.8504, 4.0968,
        4.3432, 4.5896, 4.836 ]),
 <a list of 5 Lists of Patches objects>)
```



Bin Median

In [33]:
```
# Bin median
for i in range (0,120,5):
    k=int(i/5)
    for j in range (5):
        bin3[k,j]=b[i+2]
print("Bin Median: \n\n",bin3)
plt.hist(bin3)

bin0 = pd.DataFrame(bin3)
#sns.pairplot(bin0)
```

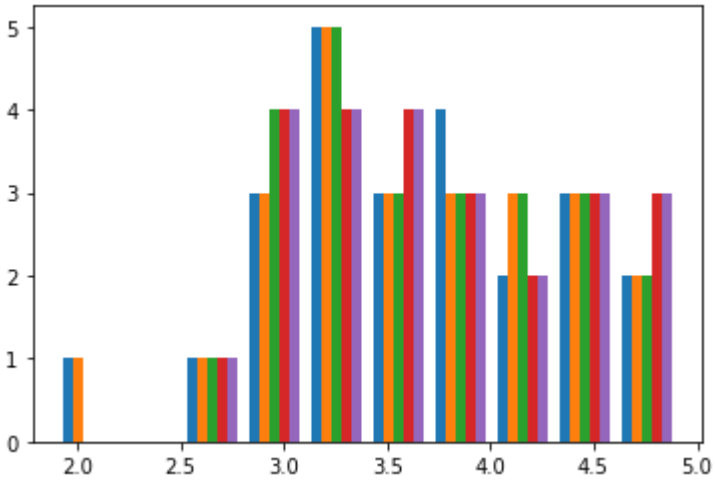
```
Bin Median:

 [[2.56 2.56 2.56 2.56 2.56]
 [2.81 2.81 2.81 2.81 2.81]
 [2.92 2.92 2.92 2.92 2.92]
 [3.   3.   3.   3.   3.  ]
 [3.08 3.08 3.08 3.08 3.08]
 [3.12 3.12 3.12 3.12 3.12]
 [3.16 3.16 3.16 3.16 3.16]
 [3.21 3.21 3.21 3.21 3.21]
 [3.26 3.26 3.26 3.26 3.26]
 [3.38 3.38 3.38 3.38 3.38]
 [3.44 3.44 3.44 3.44 3.44]
 [3.54 3.54 3.54 3.54 3.54]
 [3.62 3.62 3.62 3.62 3.62]
 [3.78 3.78 3.78 3.78 3.78]
 [3.85 3.85 3.85 3.85 3.85]
 [3.98 3.98 3.98 3.98 3.98]
 [4.09 4.09 4.09 4.09 4.09]
 [4.23 4.23 4.23 4.23 4.23]
 [4.31 4.31 4.31 4.31 4.31]
 [4.39 4.39 4.39 4.39 4.39]
 [4.45 4.45 4.45 4.45 4.45]
 [4.57 4.57 4.57 4.57 4.57]
 [4.67 4.67 4.67 4.67 4.67]
 [4.81 4.81 4.81 4.81 4.81]]
```



Bin Boundry

In [35]:
```
# Bin boundaries
for i in range (0,120,5):
    k=int(i/5)
    for j in range (5):
        if (b[i+j]-b[i]) < (b[i+4]-b[i+j]):
            bin2[k,j]=b[i]
        else:
            bin2[k,j]=b[i+4]
print("Bin Boundaries: \n\n",bin2)
plt.hist(bin2)

bin0 = pd.DataFrame(bin2)
#sns.pairplot(bin0)
```

```
Bin Boundaries:

 [[1.9  1.9  2.76 2.76 2.76]
 [2.76 2.76 2.86 2.86 2.86]
 [2.87 2.87 2.87 2.98 2.98]
 [2.98 2.98 2.98 3.03 3.03]
 [3.04 3.04 3.1  3.1  3.1 ]
 [3.11 3.13 3.13 3.13 3.13]
 [3.14 3.14 3.18 3.18 3.18]
 [3.19 3.21 3.21 3.21 3.21]
 [3.23 3.23 3.23 3.23 3.32]
 [3.35 3.35 3.35 3.42 3.42]
 [3.43 3.43 3.43 3.46 3.46]
 [3.48 3.56 3.56 3.56 3.56]
 [3.56 3.56 3.56 3.56 3.69]
 [3.75 3.75 3.78 3.78 3.78]
 [3.79 3.87 3.87 3.87 3.87]
 [3.9  3.98 3.98 3.98 3.98]
 [4.   4.12 4.12 4.12 4.12]
 [4.13 4.13 4.3  4.3  4.3 ]
 [4.3  4.3  4.3  4.32 4.32]
 [4.34 4.34 4.43 4.43 4.43]
 [4.45 4.45 4.45 4.45 4.5 ]
 [4.53 4.53 4.53 4.64 4.64]
 [4.67 4.67 4.67 4.67 4.77]
 [4.78 4.78 4.78 4.91 4.91]]
```



CONCLUSION

By Performing Binning Methods on CGPA data we come to conclusion that Bin Mean and Bin Median can perform well in most of the data set and can smooth data very well as compare to Bin Boundry. Bin Boundry can smooth data but some time it well worst in some situation but Bin Mean and Median are well for any data.

In [ ]: