Task 1: Normalization

Normalization-feature scaling techniques in machine learning, which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Standardization-- feature scaling techniques in machine learning, where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Difference between standardization and normalization: 1)Standartization:Distribution of data does follow a Gaussian distribution Normalization:Distribution of data does not follow a Gaussian distribution. This can be useful in algorithms that do not assume any distribution of the data like K-Nearest Neighbors and Neural Networks. 2)Standartization:Does not have a bounding range. Normalization:Does have a bounding range [0,1].

Summary of types normalization / standardization:

1)Min-Max Normalization-performs linear transformation on the original data. The minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. It has it's disadvantages as it does not handle outliers very well.

2)Z-score normalization-this is a function to apply z-Score normalization to a matrix or data frame. By using this type of normalization, the mean of the transformed set of data points is reduced to zero by subtracting the mean of each attribute from the values of the attributes and dividing the result by the standard deviation of the attribute. Uses the function scale found in the base library.

3)Decimal Scaling-is another technique for normalization in data mining. It functions by converting a number to a decimal point.

Sources of information:

1)https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/ (https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/)

2)Book:A Machine-Learning Approach to Phishing Detection and Defense by Oluwatobi Ayodeji Akanbi,Iraj Sadegh Amiri ,Elahe Fazeldehkordi

3)https://www.codecademy.com/articles/normalization (https://www.codecademy.com/articles/normalization)

4)http://finzi.psych.upenn.edu/R/library/dprep/html/znorm.html (http://finzi.psych.upenn.edu/R/library/dprep/html/znorm.html)

Implementation of a function that receives a series of relevant data and parameters and returns it after normalization (for each type of normalization a separate function)

Below implementation of normalization/standartization by "pure" python

In [51]:

```
#Min-Max scaling
array=[1,2,3,4,5,6,9,10,22]
def MinMaxScaling(array):
    return [(var - min(array)) / (max(array) - min(array)) for var in array]
minmax = MinMaxScaling(array)

print("After Min-Max Scalling:",end=" ")
print(minmax)
```

```
After Min-Max Scalling: [0.0, 0.047619047619047616, 0.09523809523809523, 0.1
4285714285714285, 0.19047619047619047, 0.23809523809523808, 0.38095238095238
093, 0.42857142857142855, 1.0]
```

In [50]:

```
#Z-score Scaling
array=[1,2,3,4,5,6,9,10,22]
def ZScore(array):
    avg_of_the_array = sum(array)/len(array)
    standart_devitiation_from_avg = (1/len(array) * sum([ (var - avg_of_the_array)**2 for v
    return [(var - avg_of_the_array)/standart_devitiation_from_avg for var in array]
z_score_scalling = ZScore(array)

print("After Z-score Scalling:",end=" ")
print(z_score_scalling)
```

```
After Z-score Scalling: [-0.9741593742742953, -0.8087360843031886, -0.643312
7943320819, -0.4778895043609751, -0.3124662143898684, -0.14704292441876163,
0.3492269454945586, 0.5146502354656654, 2.499729715118946]
```

In [48]:

```
#Decimal Scaling
array=[1,2,3,4,5,6,9,10,22]
after_scalling=list()

def decimal_scaling(arr):
    for var in arr:
        global after_scalling
        q=len(str(var))
        after_scaling.append(var/(10**q))

decimal_scalling(array)
print("After Decimal Scaling:",end=" ")
print(after_scalling)
```

```
After Decimal Scaling: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.9, 0.1, 0.22]
```

Implementation of the same types of normalization/standartization but with libraries of python :

In [40]:

```
#Min-Max scaling
#For this type of scaling you can use sklearn.perpocessing library.
from sklearn.preprocessing import minmax_scale

array=[1,2,3,4,5,6,9,10,22]

after = minmax_scale(array)

print(after)
```

```
[0.         0.04761905 0.0952381  0.14285714 0.19047619 0.23809524
 0.38095238 0.42857143 1.        ]
```

In [41]:

```
#Z-score Scaling
#For this type of scaling you can use sklearn.perpocessing library.
import numpy as np
from sklearn.preprocessing import StandardScaler

array=[1,2,3,4,5,6,9,10,22]

data = np.array(array).reshape(-1, 1)
scaler = StandardScaler()
scaler.fit(data)

after = scaler.transform(data)
print(after)
```

```
[[-0.97415937]
 [-0.80873608]
 [-0.64331279]
 [-0.4778895 ]
 [-0.31246621]
 [-0.14704292]
 [ 0.34922695]
 [ 0.51465024]
 [ 2.49972972]]
```

as for decimal scalling i wasn't able to find any library that can execute it(with her resources).

Task 2:Discretization using binning

Discretization-the process of putting values into buckets so that there are a limited number of possible states. The buckets themselves are treated as ordered and discrete values. You can discretize both numeric and string columns.

Equal frequency discretization-it is the process of separating all possible values into k number of bins, each of them having the same number of observations. Intervals may correspond to quantile values.

In [6]:

```
#Implementation of Equal-frequency Discretizer
array=[0,4,12,16,16,18,24,26,28]
def equalFreqDisc(array, number_of_bins):
    temp=list()
    a = len(array)
    numElement_for_each_been = int(a / number_of_bins)
    for i in range(0, number_of_bins):
        arr = []
        for j in range(i * numElement_for_each_been, (i + 1) * numElement_for_each_been):
            if j >= a:
                break
            arr = arr + [array[j]]
        temp.append(arr)
    return temp

print(equalFreqDisc(array,3))
```

```
[[0, 4, 12], [16, 16, 18], [24, 26, 28]]
```

Equal-width discretization- is a way of doing discretization. This means that after the binning, all bins have equal width, or represent an equal range of the original variable values, no matter how many cases are in each bin.

In [5]:

```
#Implementation of Equal-Width Discretization

array=[0,4,12,16,16,18,24,26,28]

def equalWdisc(arr,num_bin):
    a = len(arr)
    w = int((max(arr) - min(arr)) / num_bin)
    min1 = min(arr)
    ar = []
    for i in range(0, num_bin + 1):
        ar = ar + [min1 + w * i]
    array=[]

    for i in range(0, num_bin):
        temp = []
        for j in ar:
            if j >= ar[i] and j <= ar[i+1]:
                temp += [j]
        array += [temp]
    return array

print(equalWdisc(array,3))
```

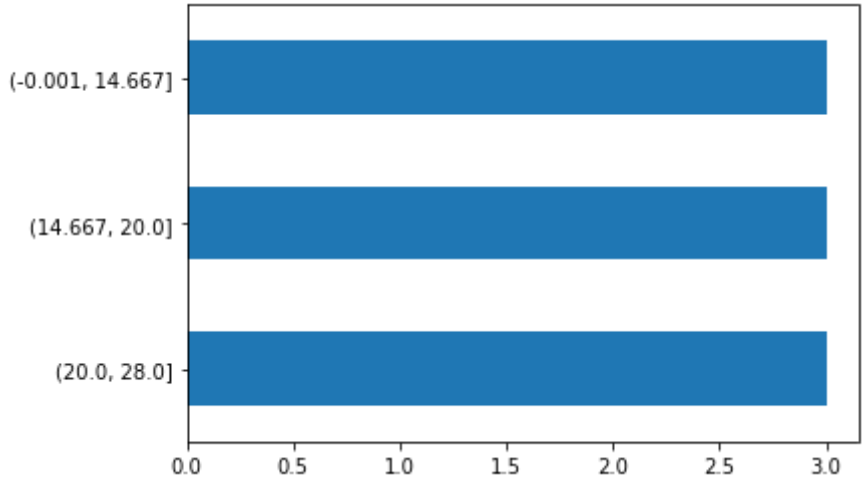
```
[[0, 9], [9, 18], [18, 27]]
```

Implementation of the same types of discretization, but with libraries of python : sklearn.preprocessing

In [68]:

```
#Implementation of Equal-frequency Discretizer
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

array=np.array([0,4,12,16,16,18,24,26,28])
ar  = pd.DataFrame(array,columns=['val'])
ar['bin']=pd.qcut(ar['val'],3).value_counts().plot(kind='barh')
plt.show()
```



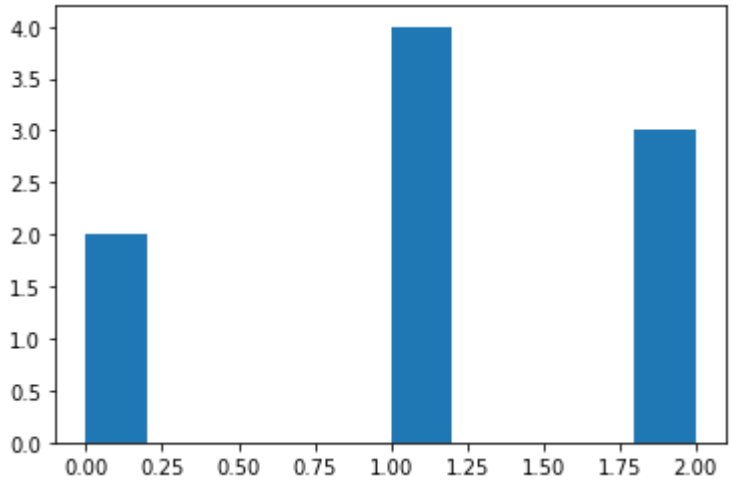
In [67]:

```
### Implementation of Equal-Width Discretizer
from sklearn.preprocessing import KBinsDiscretizer
from matplotlib import pyplot
import numpy as np

array=np.array([0,4,12,16,16,18,24,26,28])

def equalWDisc(arr):
    arr = arr.reshape((len(arr),1))
    bins = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
    arr_trans = bins.fit_transform(arr)
    pyplot.hist(arr_trans,10)
    pyplot.show()
    return arr_trans

print(equalWDisc(array))
```




```
[[0.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [2.]
 [2.]
 [2.]]
```

Task 3:Smoothing

Data smoothing refers to a statistical approach of eliminating outliers from datasets to make the patterns more noticeable.

Moving Average&How it helps in smoothing

Definition: it's series of averages calculated using sequential segments of data points over a series of values. They have a length, which defines the number of data points to include in each average.

How it helps in smoothing: It help's determine the underlying trend in housing permits and other volatile data. A moving average smoothes a series by consolidating the monthly data points into longer units of time—namely an average of several months' data.

Data Smoothing Methods: 1)Simple Moving Average(already talked about it above, so let's do implementation)

In [7]:

```
import numpy as np

data=[8,16, 9, 15, 21, 21, 24, 30,26, 27, 30, 34]
dataset=np.array(data)

def SMA(data, Period):
    sma = []
    count = 0
    for i in range(data.size):
        if data[i] is None:
            sma.append(None)
        else:
            count += 1
            if count < Period:
                sma.append(None)
            else:
                sma.append(np.mean(data[i-Period+1:i+1]))

    return np.array(sma)

print(f'SMA:{SMA(dataset,2)}')
```

```
SMA:[None 12.0 12.5 12.0 18.0 21.0 22.5 27.0 28.0 26.5 28.5 32.0]
```

2)Weighted Moving Average-is a technical indicator that assigns a greater weighting to the most recent data points, and less weighting to data points in the distant past. Below you can see it's implementation

In [74]:

```
#Weighted Moving Average
def helpFunc(k):
    return k+1.0

def WMA(ar):
    ws=[helpFunc(k) for k in range(len(ar))]
    denom=sum(ws)
    wps=[helpFunc(k)*p for k,p in enumerate(ar)]
    return sum(wps)/denom

array=[290,260,288,300,310,303,329,340,316,330,308,310]
print(f'Weighted Moving Average:{WMA(array)}')
```

```
Weighted Moving Average:314.3205128205128
```

3)Exponential Moving Average-is the 2nd most widely used techical indicator whten it comes to moving averages and stock charts. It gives more weight to recent prices and are calculated by applying to a percentage of today's closing price to yestarfay's moving average.

In [2]:

```
#Exponential Moving Average
import numpy as np

dataset=[290,260,288,300,310,303,329,340,316,330,308,310]

def EMA(val,wind):
    weight=np.exp(np.linspace(-1.,0.,wind))
    weight/=weight.sum()
    temp=np.convolve(val,weight)[:len(val)]
    temp[:wind]=temp[wind]
    return temp

print(f'Exponential Moving Average:{EMA(dataset,3)}')
```

```
Exponential Moving Average:[276.05443373 276.05443373 276.05443373 276.05443
373 295.78547254
 303.63093003 311.38977954 317.88107079 329.95694634 330.76406151
 318.81015261 319.51521605]
```

Binning Methods for Data Smoothing

Most of the data us fill of noise, so binning method can be used for smoothing the data or in another words to handle noisy data.In this method, the data is first sorted and then the sorted values are distributed into a number of buckets or bins.

1)Smoothing by bin means-each value stored in the bin will be replaced by bin means.

In [118]:

```
#Smoothing by bin mean implementations
import numpy as np
data=[8,16, 9, 15, 21, 21, 24, 30,   26, 27, 30, 34]
dataset=np.sort(np.array(data))

def equalFreqDisc(array, number_of_bins):
    temp=list()
    a = len(array)
    numElement_for_each_been = int(a / number_of_bins)
    for i in range(0, number_of_bins):
        arr = []
        for j in range(i * numElement_for_each_been, (i + 1) * numElement_for_each_been):
            if j >= a:
                break
            arr = arr + [array[j]]
        temp.append(arr)
    return temp

def SmoothBinMean(data):
    dataSeq=equalFreqDisc(data,3)
    for k in dataSeq:
        val=sum(k)/len(k)
        for i  in range(len(k)):
            k[i]=val
    return dataSeq

print(SmoothBinMean(dataset))
```

```
[[12.0, 12.0, 12.0, 12.0], [23.0, 23.0, 23.0, 23.0], [30.25, 30.25, 30.25, 3
0.25]]
```

2)Smoothing by bin boundaries-in this case the minimum and maximum values in a given bin are identified as the bin boundaries. Each bin value is then replaced by the closest boundary value.

In [120]:

```
#Smoothing by bin boundaries implementations
import numpy as np
data=[8,16, 9, 15, 21, 21, 24, 30,   26, 27, 30, 34]
dataset=np.sort(np.array(data))

def equalFreqDisc(array, number_of_bins):
    temp=list()
    a = len(array)
    numElement_for_each_been = int(a / number_of_bins)
    for i in range(0, number_of_bins):
        arr = []
        for j in range(i * numElement_for_each_been, (i + 1) * numElement_for_each_been):
            if j >= a:
                break
            arr = arr + [array[j]]
        temp.append(arr)
    return temp

def SmoothBinBound(data):
    dataSeq=equalFreqDisc(data,3)
    for k in dataSeq:
        min_val=min(k)
        max_val=max(k)
        val=sum(k)/len(k)
        for i  in range(len(k)):
            print()
            if k[i]<val:
                k[i]=min_val
            else:
                k[i]=max_val
    return dataSeq
print(SmoothBinBound(dataset))
```

```
[[8, 8, 16, 16], [21, 21, 26, 26], [27, 27, 27, 34]]
```

SMA with using pandas and numpy

In [102]:

```
#Simple Moving Average
import pandas as pd
import numpy as np

product = {'month' : list(range(1,13)),'clients demand':[290,260,288,300,310,303,329,340,31
df = pd.DataFrame(product)

def simple_moving_avg():
    for i in range(0,df.shape[0]-2):
        df.loc[df.index[i+2],'Simple Moving Average'] = np.round(((df.iloc[i,1]+ df.iloc[i+
    print(df)

simple_moving_avg()
```

```
    month  clients demand  Simple Moving Average
0       1             290                    NaN
1       2             260                    NaN
2       3             288                  279.3
3       4             300                  282.7
4       5             310                  299.3
5       6             303                  304.3
6       7             329                  314.0
7       8             340                  324.0
8       9             316                  328.3
9      10             330                  328.7
10     11             308                  318.0
11     12             310                  316.0
```

Weighted Moving Average with numpy

In [58]:

```
import numpy as np
array=[290,260,288,300,310,303,329,340,316,330,308,310]

def WMA(data):
    res = np.ma.array(data, mask=[True,True, False, True,True, True, True, False,False, Fal
    print(f'WMA:{np.ma.average(res, weights=[3, 1, 0,3,3, 3, 3, 0,3, 1, 3, 0])}')

WMA(array)
```

```
WMA:314.57142857142856
```

Exponential Moving Average with numpy

In [71]:

```
import pandas as pd
import numpy as np

dataset=[290,260,288,300,310,303,329,340,316,330,308,310]
period = 12

def EMA(val, period):
    df=pd.Series(val)
    df['ema10'] = pd.Series.ewm(df, span=period).mean()
    return df['ema10']

print(f'EMA: {EMA(dataset, period)}')
```

```
EMA: 0     290.000000
1     273.750000
2     279.311778
3     285.842241
4     292.405818
5     294.980762
6     302.572036
7     310.382675
8     311.493982
9     315.000865
10    313.719876
11    313.058494
dtype: float64
```

I haven't found libraries in python that with them a could implement smothing by bin means and bin boundaries. Below links where i checked: https://t4tutorials.com/binning-methods-for-data-smoothing-in-data-mining/ (https://t4tutorials.com/binning-methods-for-data-smoothing-in-data-mining/) http://mercury.webster.edu/aleshunas/Support%20Materials/Data_preprocessing.pdf (http://mercury.webster.edu/aleshunas/Support%20Materials/Data_preprocessing.pdf) https://www.coursehero.com/file/p4orler7/Bin-3-25-28-34-Smoothing-by-bin-means-Bin-1-9-9-9-Bin-2-22-22-22-Bin-3-29-29-29/ (https://www.coursehero.com/file/p4orler7/Bin-3-25-28-34-Smoothing-by-bin-means-Bin-1-9-9-9-Bin-2-22-22-22-Bin-3-29-29-29/)

In [ ]: