**def** getMode(numbers):

max\_occur **=** **-**1

**if** len(numbers) **==** 0:

**return** **None**

**else**:

occurences **=** {}

**for** i **in** numbers:

**if** occurences**.**get(i) **==** **None**:

occurences[i] **=** 1

**else**:

occurences[i] **+=** 1

**if** occurences[i] **>** max\_occur:

max\_occur **=** occurences[i]

*# get max occurence number*

**for** i **in** occurences:

**if** occurences[i] **==** max\_occur:

**return** i

**return** **None**

**def** getMean(numbers):

**if** len(numbers) **==** 0:

**return** **None**

**else**:

current\_sum **=** 0

**for** i **in** numbers:

current\_sum **+=** i

current\_avg **=** current\_sum**/**len(numbers)

**return** current\_avg

**def** getMedian():

numbers **=** []

inp **=** 0

**while** **True**:

inp **=** int(input("Enter a number OR type 'exit'"))

**if** inp **==** 'exit':

**break**

**else**:

numbers**.**append(inp)

**if** len(numbers) **==** 0:

**return** **None**

**else**:

middle\_index **=** len(numbers)**//**2

**return** numbers[middle\_index]

**def** getStandardDeviation(numbers):

**if** len(numbers) **==** 0:

**return** 0

**else**:

mean **=** getMean(numbers)

std\_deviation **=** 0

**for** i **in** numbers:

std\_deviation **+=** (i **-** mean)**\*\***2

**return** (std\_deviation**/**len(numbers))**\*\***0.5

**def** getVariance(numbers)

**return** getStandardDeviation(numbers)**\*\***2

**def** getNormalization(features):

x\_min **=** min(features)

x\_max **=** max(features)

normalized\_vals **=** []

**for** i **in** features:

normalized\_vals**.**append((i **-** x\_min)**/**(x\_max **-** x\_min))

**return** normalized\_vals

getNormalization([10,20,30,40])

**def** getStandardization(features):

mean **=** getMean(features)

std\_deviation **=** getStandardDeviation(features)

standardized\_vals **=** []

**for** i **in** features:

standardized\_vals**.**append((i **-** mean)**/**std\_deviation)

**return** standardized\_vals

getStandardization([10,20,30,40])

*# MinMax Normalization*

**def** doMinMaxNormalization(numbers):

result **=** []

**if** len(numbers) **==** 0:

**return** result

**else**:

min\_value **=** min(numbers)

max\_value **=** max(numbers)

**for** i **in** numbers:

result**.**append((i **-** min\_value)**/**(max\_value **-** min\_value))

**return** result

features **=** [100000,**-**2,50,12,700,9000]

print(doMinMaxNormalization(features))