```
import math

class StatLab:
    def __init__(self):
        return

    #
    # Utilities
    #

    def iseven(self, n):
        return n % 2 == 0

    def isodd(self, n):
        return n % 2 != 0

    #
    # Graphs
    #

    def histogram(self, sample):
        xlist = []
        ylist = []
        if len(sample) > 0:
            sorted_sample = sorted(sample)
            xlist.append(sorted_sample[0])
            ylist.append(0)
            i = 0
            for x in sorted_sample:
                if x <= xlist[i]:
                    ylist[i] += 1
                else:
                    xlist.append(x)
                    ylist.append(1)
                    i += 1
        result = xlist, ylist
        return result

    def ogive(self, sample):
        xlist, ylist = self.histogram(sample)
        if len(ylist) > 1:
            for i in range(1, len(ylist)):
                ylist[i] += ylist[i - 1]
        result = xlist, ylist
        return result

    def pareto(self, sample):
        pxlist = []
        pylist = []
        if len(sample) > 0:
            xlist, ylist = self.histogram(sample)
            pxlist = [xlist[0]]
            pylist = [ylist[0]]
            for i in range(1, len(ylist)):
                n_pylist = len(pylist)
                inserted = False
                for j in range(0, n_pylist):
                    if ylist[i] >= pylist[j]:
                        pxlist.insert(j, xlist[i])
                        pylist.insert(j, ylist[i])
                        inserted = True
                        break
                if not inserted:
                    pxlist.append(xlist[i])
                    pylist.append(ylist[i])
        result = pxlist, pylist
```

```
        return result

    def pmf(self, sample):
        xlist, ylist = self.histogram(sample)
        n = float(len(sample))
        for i in range(0, len(ylist)):
            ylist[i] = float(ylist[i]) / n
        result = xlist, ylist
        return result

    def cmf(self, sample):
        xlist, ylist = self.ogive(sample)
        n = float(len(sample))
        for i in range(0, len(ylist)):
            ylist[i] = float(ylist[i]) / n
        result = xlist, ylist
        return result

    #
    # Measures of Central Tendency
    #

    def weighted_mean(self, weights, sample):
        result = float("NaN")
        n = len(sample)
        if (n > 0) and (n == len(weights)):
            sum = 0.0
            for i in range(0, n):
                sum += weights[i] * sample[i]
            result = sum / math.fsum(weights)
        return result

    def mean(self, sample):
        result = float("NaN")
        n = len(sample)
        if n > 0:
            sum = math.fsum(sample)
            result = sum / float(n)
        return result

    def median(self, sample):
        result = float("NaN")
        n = len(sample)
        if n > 0:
            sorted_sample = sorted(sample)
            i = int(n) / 2
            if self.isodd(n):
                result = sorted_sample[i]
            else:
                result = float(sorted_sample[i - 1] + sorted_sample[i]) / 2.0
        return result

    def mode(self, sample):
        result = float("NaN")
        n = len(sample)
        if n > 0:
            xlist, ylist = self.histogram(sample)
            m = xlist[0]
            f = ylist[0]
            for i in range(0, len(xlist)):
                if ylist[i] > f:
                    m = xlist[i]
                    f = ylist[i]
            result = m
        return result
```

```
def range(self, sample):
    min = float("NaN")
    max = float("NaN")
    n = len(sample)
    if n > 0:
        sorted_sample = sorted(sample)
        min = sorted_sample[0]
        max = sorted_sample[n - 1]
    result = min, max
    return result

def amplitude(self, sample):
    result = float("NaN")
    n = len(sample)
    if n > 0:
        min, max = self.range(sample)
        result = max - min
    return result

def midrange(self, sample):
    result = float("NaN")
    n = len(sample)
    if n > 0:
        sorted_sample = sorted(sample)
        result = (sorted_sample[0] + sorted_sample[len(sorted_sample) - 1]) / 2.0
    return result

midpoint = midrange

#
# Measures of Variation
#

def sum_squares(self, sample):
    result = float("NaN")
    n = len(sample)
    if n > 0:
        mean = self.mean(sample)
        sum = 0.0
        for x in sample:
            dif = float(x) - mean
            sum += dif ** 2
        result = sum
    return result

def variance_pop(self, pop):
    result = float("NaN")
    n = len(pop)
    if n > 0:
        var = self.sum_squares(pop) / float(n)
        result = var
    return result

def variance_sample(self, sample):
    result = float("NaN")
    n = len(sample)
    if n > 1:
        var = self.sum_squares(sample) / float(n - 1)
        result = var
    return result

variance = variance_sample

def std_dev_pop(self, pop):
    result = float("NaN")
    n = len(pop)
```

```
    if n > 0:
        result = math.sqrt(self.variance_pop(pop))
    return result

def std_dev_sample(self, sample):
    result = float("NaN")
    n = len(sample)
    if n > 1:
        result = math.sqrt(self.variance_sample(sample))
    return result

std_dev = std_dev_sample

def chebyshev(self, k):
    fk = float(k);
    if fk > 1.0:
        result = 1.0 - 1.0 / (fk * fk)
    else:
        raise ValueError("k must be > 1")
    return result

#
# Measures of Position
#

def zscores_pop(self, sample):
    # ... Sorry, I'm too lazy to code this right now. By the way, code yourselves!
    return result

def zscores_sample(self, sample):
    # ... Sorry, I'm too lazy to code this right now. By the way, code yourselves!
    return result

zscores = zscores_sample

def percentile_rank(self, x, sample):
    # ... Sorry, I'm too lazy to code this right now. By the way, code yourselves!
    return result

def percentile(self, rank, sample):
    result = float("NaN")
    n = len(sample)
    if n > 0:
        k = float(rank)
        if (k > 0.0) and (k <= 100.0):
            sorted_sample = sorted(sample)
            i = float(n - 1) * (k / 100.0)
            f = math.floor(i)
            c = math.ceil(i)
            if f == c:
                result = sorted_sample[int(i)]
            else:
                d1 = sorted_sample[int(f)] * (c - i)
                d2 = sorted_sample[int(c)] * (i - f)
                result = d1 + d2
    return result

def decile(self, rank, sample):
    k = 10 * rank
    result = self.percentile(k, sample)
    return result

def quartile(self, rank, sample):
    k = 25 * rank
    result = self.percentile(k, sample)
    return result
```