```
class statistic:

    def __init__(self,data):
        if not data:
            raise ValueError('The data list cannot be empty')
        self. data = data

    def mean(self):
        try:
            return sum(self.data) / len(self.data)
        except Exception as e:
            return f'Error calculating mean: {e}'

    def median(self):
        try:
            sort_data = sorted(self.data)
            n = len(sort_data)
            mid = n // 2

            if n % 2 == 0:
                return (sort_data[mid-1] + sort_data[mid])/2
            else:
                return sort_data[mid]
        except Exception as e:
            return f'Error calculating median: {e}'


    def mode(self):
        try:
            fre = {}

            for num in self.data:
                if num not in fre:
                    fre[num] = 1
                else:
                    fre[num] +=1
            return [k for k,v in fre.items() if v ==
max(fre.values())]
        except Exception as e:
            return f'Error calculating mode: {e}'

    def variance(self):
        try:
            mean = self.mean()
            return sum((x-mean)** 2 for x in self.data)/
len(self.data)
        except Exception as e:
            return f'Error calculating variance: {e}'

    def standard_deviation(self):
```

```
        try:
            variance = self.variance()
            return variance **0.5
        except Exception as e:
            return f'Error calculating standard_deviation: {e}'

    def cofficient_of_variation(self):
        try:
            mean = self.mean()
            if mean == 0:
                raise ValueError('mean is zero coefficient of
variation is undefined')
            std = self.standard_deviation()
            return (std/mean)* 100
        except Exception as e:
            return f'Error calculating cofficient of variation: {e}'

    def covariance(self,other_data):
        try:
            if len(self.data) != len(other_data):
                raise ValueError('Both datasets must have the same
number of elements.')
            mean_self = self.mean()
            mean_other = sum(other_data)/len(other_data)
            return sum((x-mean_self)*(y-mean_other) for x,y in
zip(self.data,other_data))
        except Exception as e:
            return f'Error calculating covariance: {e}'
```