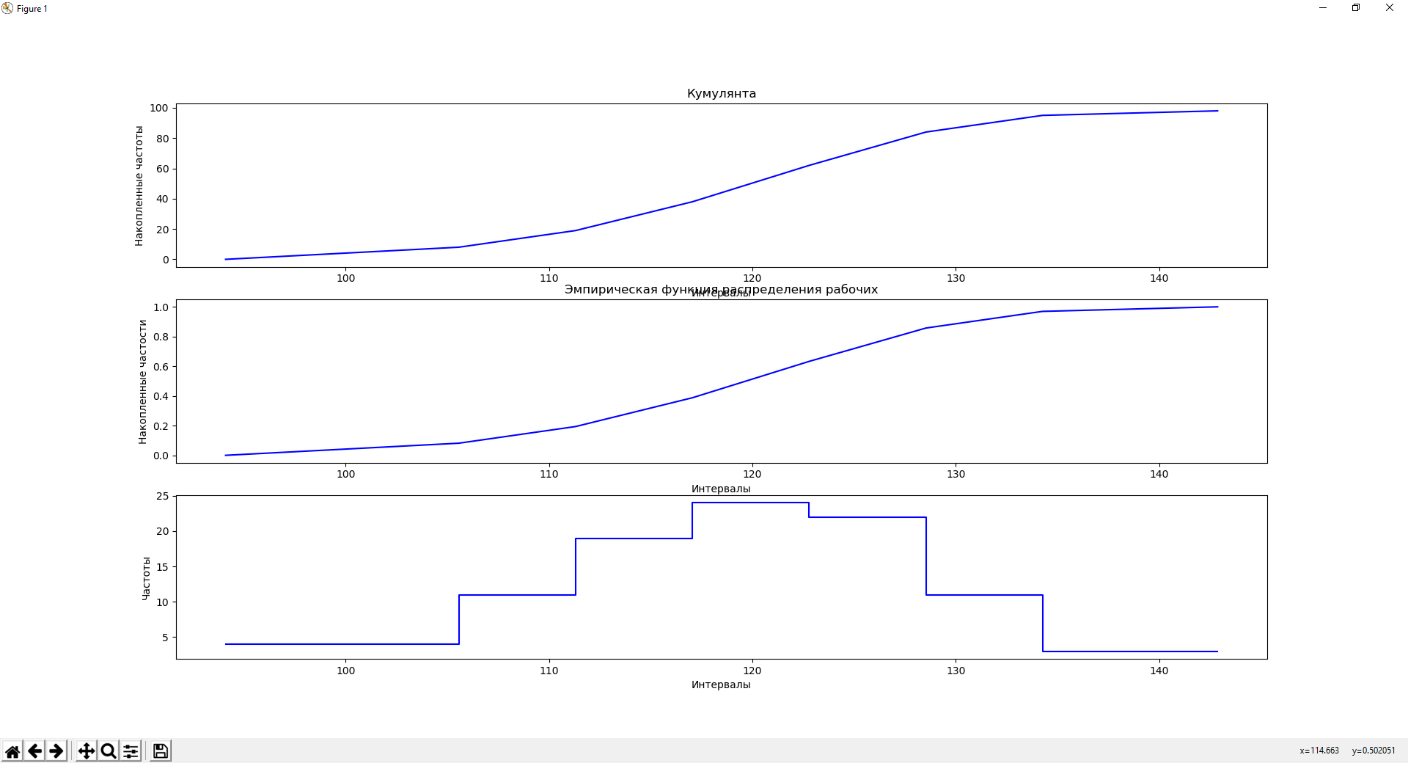
**Лабораторная работа №2**

Задача №1

Постановка задачи: имеются данные о распределении 100 рабочих цеха по выработке в отчетном году (в процентах к предыдущему году). Всего n=100 значений.

Результаты:

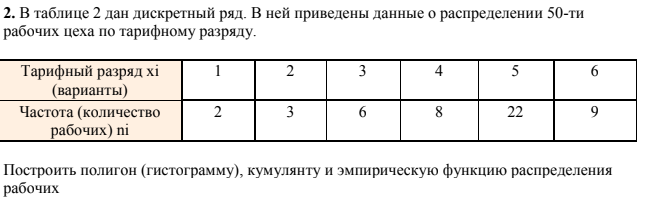


Код программы:

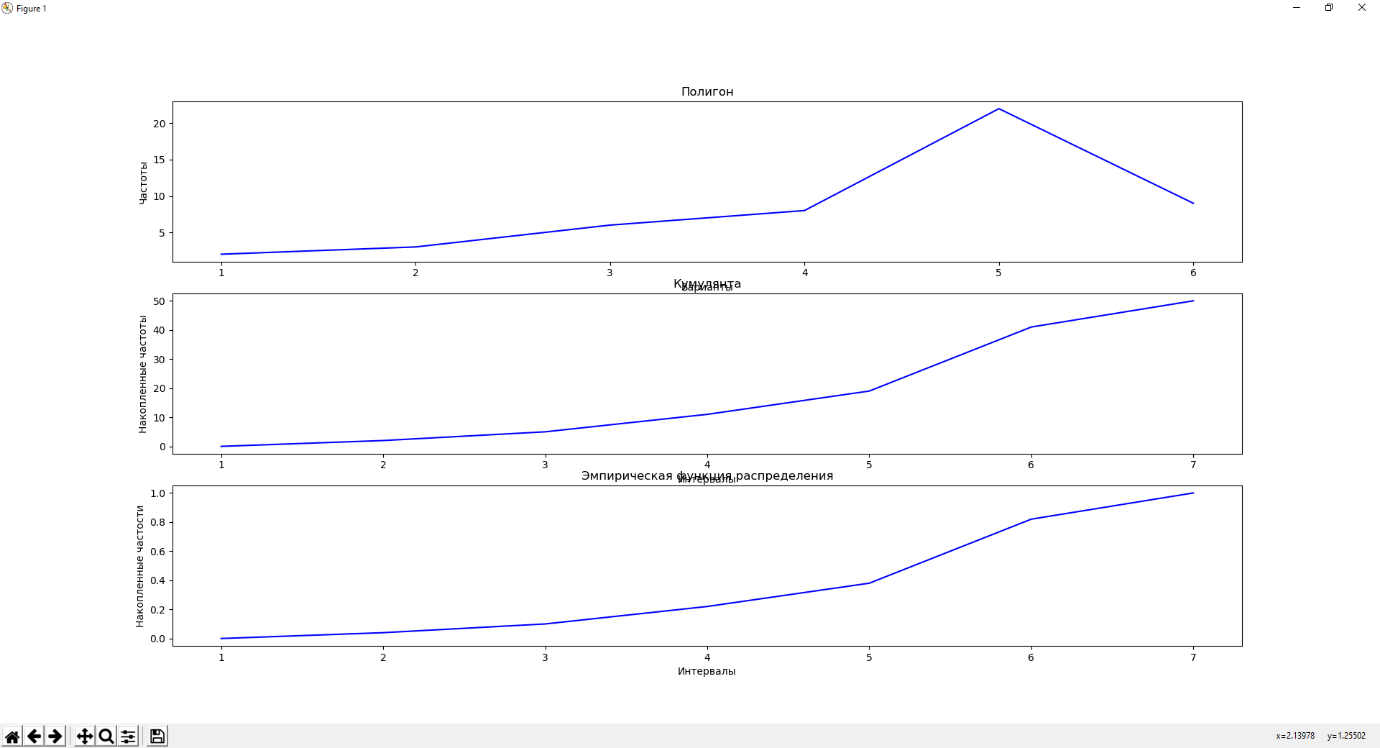
def task1():  
 values = [  
 103.4, 115.2, 127, 131, 114, 114.1, 119.6, 125.5, 116.9, 118.1, 123.5,  
 113.5, 112.3, 123, 125, 129.9, 99.2, 111, 122, 134, 107.1, 117, 117.5,  
 118.5, 124, 127.8, 108, 119.5, 123, 126.1, 100.1, 120.2, 122.2, 124.8,  
 109, 113, 122.5, 135.8, 97, 121.1, 123.8, 123.2, 105.9, 122.6, 123.9,  
 129.5, 107, 123.5, 128.5, 117.5, 121.5, 127.5, 113.2, 120.6, 126.5,  
 116, 122.9, 138, 115, 123.1, 140, 94.1, 110, 112.9, 132, 102, 109.5,  
 118.3, 135, 112.5, 115.5, 120, 126, 130, 105.5, 108.2, 119.2, 131.4,  
 106.5, 112, 120.8, 121.9, 134.2, 115.7, 118.9, 124.5, 111.5, 121, 133,  
 116.5, 119, 129, 106.1, 119.8, 133.6, 114.5, 118, 128  
 ]  
 VariationSeries.PRECISION = 2  
  
 v = ContinuousVS(values)  
 v.draw\_cumulate(3, 1, 1)  
 v.draw\_empiric\_dist\_func(3, 1, 2, postfix='рабочих')  
 v.draw\_hist(3, 1, 3)  
  
 ContinuousVS.show()

Задача №2

Постановка задачи:



Результаты:

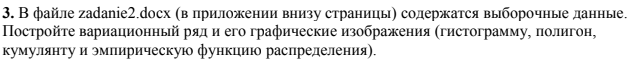


Код программы:

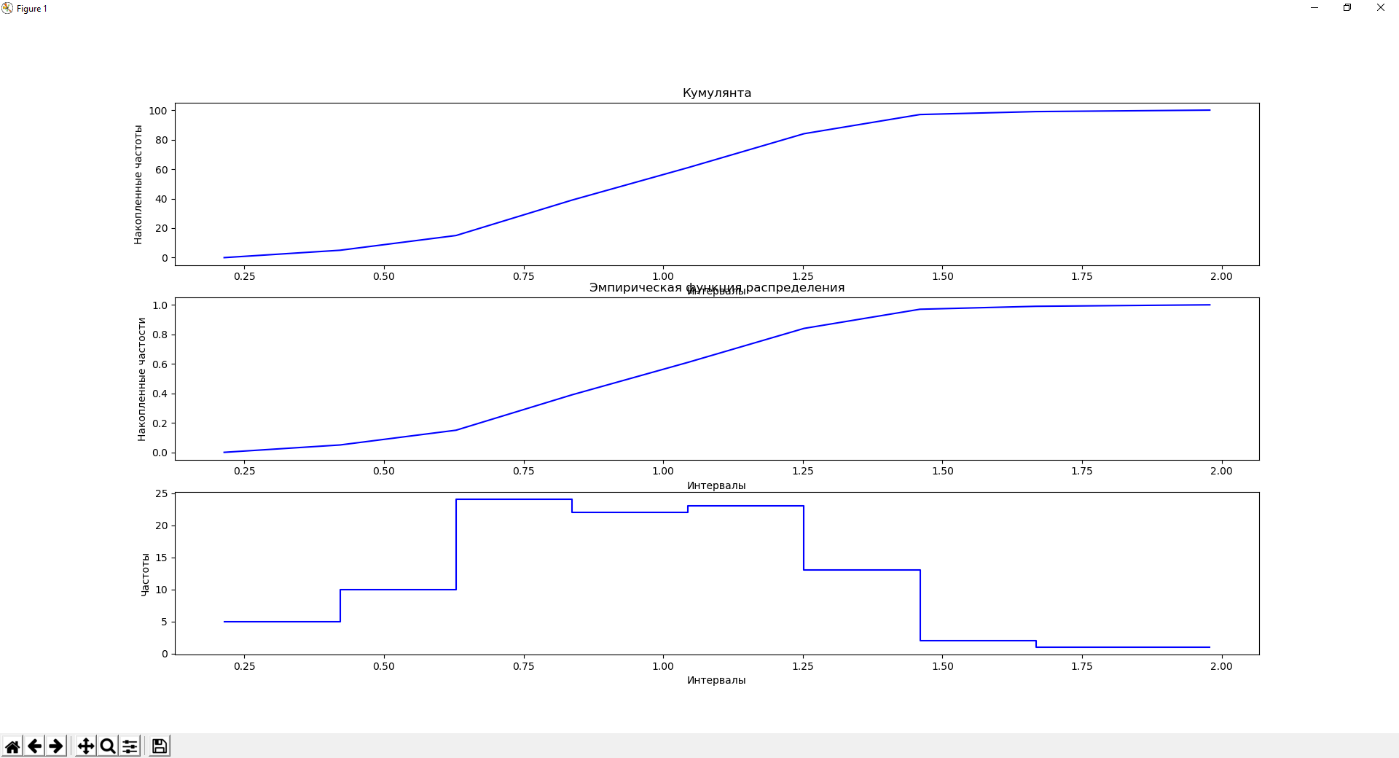
def task2():  
 values = {  
 1: 2, 2: 3, 3: 6, 4: 8, 5: 22, 6: 9  
 }  
 VariationSeries.PRECISION = 2  
  
 v = DiscreteVS(values)  
 v.draw\_polygon(3, 1, 1)  
 v.draw\_cumulate(3, 1, 2)  
 v.draw\_empiric\_dist\_func(3, 1, 3)  
  
 DiscreteVS.show()

Задача №3

Постановка задачи:



Результаты:

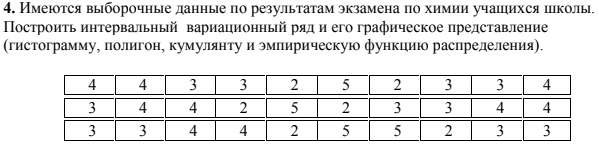


Код программы:

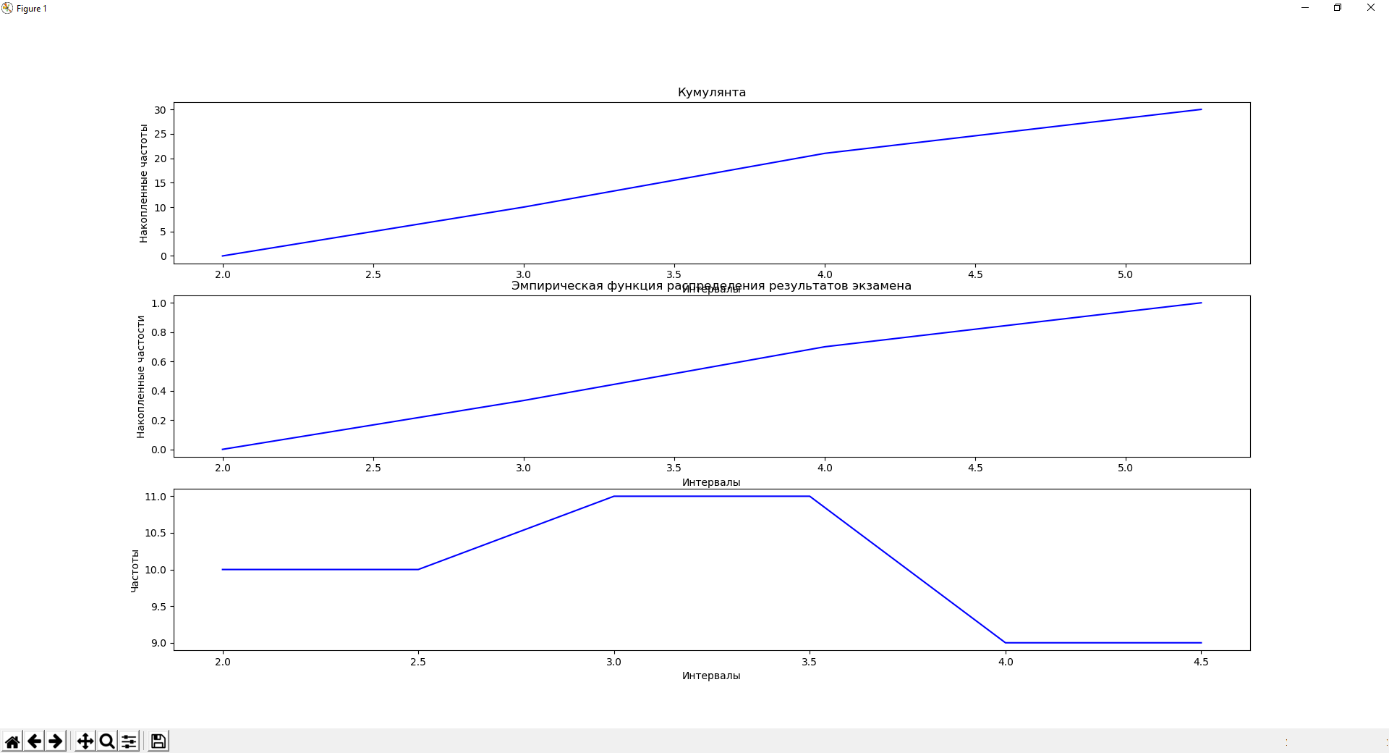
def task3():  
 values = [  
 1.14285583, 0.21398374, 1.25641624, 0.67329946, 1.21496283, 0.99101069,  
 1.39925669, 0.61109646, 0.85890088, 0.78632108, 0.9738463, 1.3846759,  
 0.49488379, 1.0979642, 1.02453946, 1.06382694, 0.78161594, 1.20567321,  
 1.38270281, 0.88719158, 0.75776634, 1.16915277, 1.23004829, 0.71265086,  
 1.02887585, 0.82302015, 1.24597822, 1.45686546, 0.91103144, 0.77406981,  
 1.09453619, 0.79865011, 0.88126134, 1.10711803, 1.00136848, 0.92217984,  
 1.24560914, 0.78720264, 0.954333, 0.99578226, 0.81526016, 0.77680747,  
 1.23527671, 1.73649997, 1.25015887, 0.71522997, 0.76771727, 1.0515177,  
 0.53930926, 1.32623785, 0.59025817, 0.84943463, 1.0391314, 0.87918459,  
 0.60738125, 1.18346139, 0.83580503, 0.95130778, 1.40929416, 0.60987357,  
 1.39038211, 1.06430415, 0.6048676, 1.36443751, 0.98420392, 1.31749231,  
 1.10304182, 0.25832193, 0.31529515, 0.43993342, 0.90625883, 1.49160615,  
 0.66502074, 0.3382135, 0.5468639, 0.66566206, 1.22896107, 1.32777678,  
 1.21582933, 1.00298477, 0.8827651, 1.07884146, 1.45221163, 0.63185447,  
 0.9416058, 1.07515286, 0.97412237, 1.51354811, 1.12753343, 0.72361969,  
 0.50409524, 0.68639066, 0.82355366, 0.69646316, 0.65239474, 0.72192621,  
 1.05932474, 1.25494818, 1.87487639, 0.74979352  
 ]  
 VariationSeries.PRECISION = 8  
  
 v = ContinuousVS(values)  
 v.draw\_cumulate(3, 1, 1)  
 v.draw\_empiric\_dist\_func(3, 1, 2)  
 v.draw\_hist(3, 1, 3)  
  
 ContinuousVS.show()

Задача №4

Постановка задачи:



Результаты:

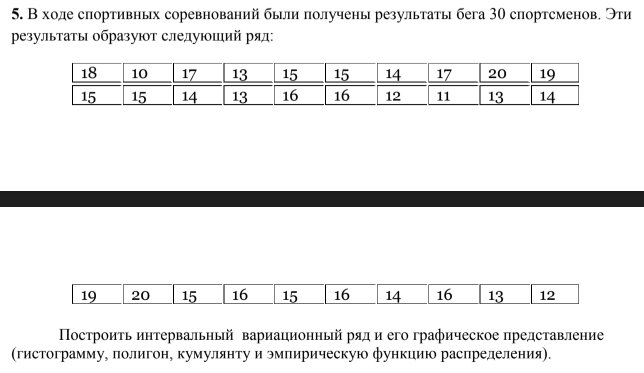


Код программы:

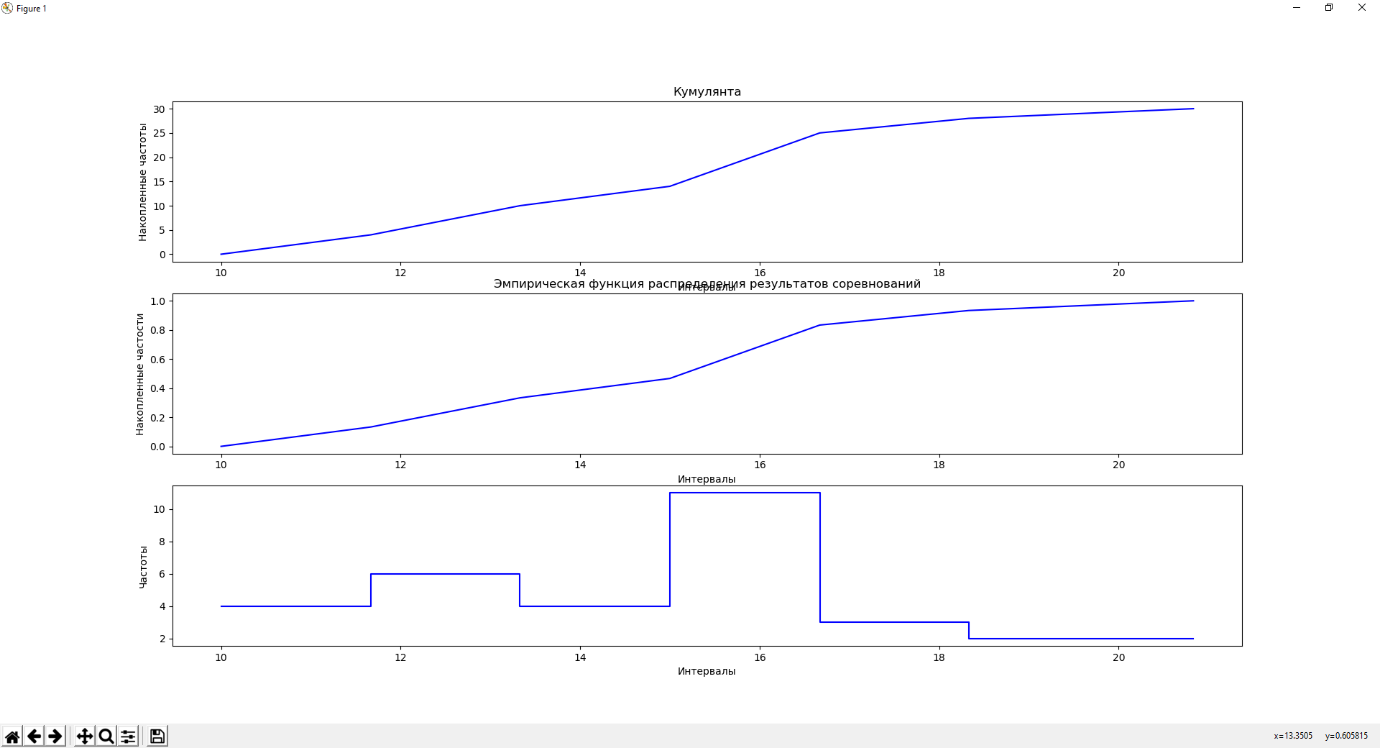
def task4():  
 values = [  
 4, 4, 3, 3, 2, 5, 2, 3, 3, 4,  
 3, 4, 4, 2, 5, 2, 3, 3, 4, 4,  
 3, 3, 4, 4, 2, 5, 5, 2, 3, 3  
 ]  
 VariationSeries.PRECISION = 2  
  
 v = ContinuousVS(values)  
 v.draw\_cumulate(3, 1, 1)  
 v.draw\_empiric\_dist\_func(3, 1, 2, postfix='результатов экзамена')  
 v.draw\_hist(3, 1, 3)  
  
 ContinuousVS.show()

Задача №5

Постановка задачи:



Результаты:



Код программы:

def task5():  
 values = [  
 18, 10, 17, 13, 15, 15, 14, 17, 20, 19,  
 15, 15, 14, 13, 16, 16, 12, 11, 13, 14,  
 19, 20, 15, 16, 15, 16, 14, 16, 13, 12  
 ]  
 VariationSeries.PRECISION = 2  
  
 v = ContinuousVS(values)  
 v.draw\_cumulate(3, 1, 1)  
 v.draw\_empiric\_dist\_func(3, 1, 2, postfix='результатов соревнований')  
 v.draw\_hist(3, 1, 3)  
  
 ContinuousVS.show()

**Deep dark back end**

import math  
import abc  
import matplotlib.pyplot as plt  
  
  
def sorted\_by\_keys(dict\_: dict) -> dict:  
 *"""Return copy of the passed dict sorted by its keys"""* return {k: dict\_[k] for k in sorted(dict\_.keys())}  
  
  
class VariationSeries:  
 *"""Abstract"""* PRECISION = 1 # {number},{PRECISION}  
  
 def \_\_init\_\_(self, values):  
 *"""* ***:param*** *values: all the values that make up the series  
 """* self.\_vs = None # Вариационный ряд  
 self.\_values = sorted(values) # Все значения ряда  
 self.\_n = len(values) # Количество значений в ряду  
 self.\_x\_max = max(self.\_values)  
 self.\_x\_min = min(self.\_values)  
  
 self.\_var\_frequencies = None # Частота варианты - m(i)  
 self.\_rel\_frequencies = None # Относительная частота (частость) - w(i)  
 self.\_acc\_frequencies = None # Накопленная частота - m(x)  
 self.\_acc\_rel\_frequencies = None # Накопленная частость w(x)  
  
 def \_gen\_acc\_frequencies(self):  
 *"""Generator"""* accumulated\_sum = 0  
 for m\_i in self.\_vs.values():  
 yield accumulated\_sum  
 accumulated\_sum += m\_i  
 yield accumulated\_sum  
  
 @staticmethod  
 def show():  
 *"""Display prepared plots"""* plt.show()  
  
 def draw\_cumulate(self, nrows: int, ncols: int, index: int):  
 *"""Prepare cumulate function"""* xs = self.get\_cumulate\_xs()  
 ys = self.get\_cumulate\_ys()  
 plt.subplot(nrows, ncols, index)  
 plt.plot(xs, ys, 'b')  
 plt.title('Кумулянта')  
 plt.xlabel('Интервалы')  
 plt.ylabel('Накопленные частоты')  
 return self  
  
 def draw\_empiric\_dist\_func(  
 self, nrows: int, ncols: int, index: int,  
 title='Эмпирическая функция распределения',  
 postfix=''  
 ):  
 *"""Prepare empiric distinction function"""* xs = self.get\_empiric\_dist\_xs()  
 ys = self.get\_empiric\_dist\_ys()  
 plt.subplot(nrows, ncols, index)  
 plt.plot(xs, ys, 'b')  
 if postfix:  
 title += f' {postfix}'  
 plt.title(title)  
 plt.xlabel('Интервалы')  
 plt.ylabel('Накопленные частости')  
 return self  
  
 @abc.abstractmethod  
 def get\_cumulate\_xs(self) -> list:  
 pass  
  
 @abc.abstractmethod  
 def get\_cumulate\_ys(self) -> list:  
 pass  
  
 @abc.abstractmethod  
 def get\_empiric\_dist\_xs(self) -> list:  
 pass  
  
 @abc.abstractmethod  
 def get\_empiric\_dist\_ys(self) -> list:  
 pass  
  
 def \_\_str\_\_(self):  
 return f'Вариационный ряд: {self.\_vs}'  
  
  
class DiscreteVS(VariationSeries):  
 *"""Discrete variation series"""* def \_\_init\_\_(self, values):  
 # User's input is a ready variable series: build the list of values  
 if isinstance(values, dict):  
 vs = sorted\_by\_keys(values)  
 values = []  
 for k, v in vs.items():  
 values += [k] \* v  
 # User's input is a list of values: build the variable series  
 else:  
 vs = {}  
 for val in sorted(values):  
 vs[val] = vs.get(val, 0) + 1  
  
 super(DiscreteVS, self).\_\_init\_\_(values)  
 self.\_vs = vs  
  
 self.\_variants = list(self.\_vs.keys())  
 self.\_var\_frequencies = list(self.\_vs.values())  
 self.\_rel\_frequencies = list(map(  
 lambda m\_i: m\_i / self.\_n, self.\_var\_frequencies  
 ))  
 self.\_acc\_frequencies = list(self.\_gen\_acc\_frequencies())  
 self.\_acc\_rel\_frequencies = list(map(  
 lambda m\_x: m\_x / self.\_n, self.\_acc\_frequencies  
 ))  
  
 self.\_x = self.\_\_find\_next\_x() # Следующий x  
  
 def \_\_find\_next\_x(self):  
 keys = list(self.\_vs.keys())  
 biggest = keys[-1]  
 diff = biggest - keys[-2]  
 return biggest + diff  
  
 def draw\_polygon(self, nrows: int, ncols: int, index: int):  
 *"""Prepare polygon function"""* xs = self.get\_polygon\_xs()  
 ys = self.get\_polygon\_ys()  
 plt.subplot(nrows, ncols, index)  
 plt.plot(xs, ys, 'b')  
 plt.title('Полигон')  
 plt.xlabel('Варианты')  
 plt.ylabel('Частоты')  
 return self  
  
 def get\_polygon\_xs(self):  
 return self.\_variants  
  
 def get\_polygon\_ys(self):  
 return self.\_var\_frequencies  
  
 def get\_cumulate\_xs(self): # x(i)  
 return self.\_variants + [self.\_x]  
  
 def get\_cumulate\_ys(self): # m(x(i))  
 return self.\_acc\_frequencies  
  
 def get\_empiric\_dist\_xs(self): # x(i)  
 return self.get\_cumulate\_xs()  
  
 def get\_empiric\_dist\_ys(self): # w(x(i))  
 return self.\_acc\_rel\_frequencies  
  
  
class ContinuousVS(VariationSeries):  
 *"""Continuous interval"""* def \_\_init\_\_(self, values):  
 # User's input is a dict with intervals  
 if isinstance(values, dict):  
 # Complete redefinition  
 self.\_vs = values  
 self.\_intervals = list(values.keys())  
  
 values = list(values.keys()) # tuples  
 self.\_n = sum(self.\_vs.values())  
 self.\_x\_max = values[-1][-1]  
 self.\_x\_min = values[0][0]  
  
 self.\_k = math.ceil(1 + 1.4 \* math.log(self.\_n)) # Количество интервалов  
 self.\_delta = (self.\_x\_max - self.\_x\_min) / self.\_k # Длина интервала  
  
 # User's input is a list of values  
 else:  
 super(ContinuousVS, self).\_\_init\_\_(values)  
  
 self.\_k = math.ceil(1 + 1.4 \* math.log(self.\_n)) # Количество интервалов  
 self.\_delta = (self.\_x\_max - self.\_x\_min) / self.\_k # Длина интервала  
  
 self.\_intervals = self.\_\_make\_intervals()  
 self.\_vs = self.\_\_make\_vs()  
  
 # self.\_variants = self.\_intervals # Here intervals  
 self.\_var\_frequencies = list(self.\_vs.values())  
 self.\_rel\_frequencies = list(map(  
 lambda m\_i: m\_i / self.\_n, self.\_var\_frequencies  
 ))  
 self.\_acc\_frequencies = list(self.\_gen\_acc\_frequencies())  
 self.\_acc\_rel\_frequencies = list(map(  
 lambda m\_x: m\_x / self.\_n, self.\_acc\_frequencies  
 ))  
  
 def \_\_make\_intervals(self) -> list:  
 *"""Return list of tuples intervals"""* p = VariationSeries.PRECISION # Precision  
 x\_start = self.\_x\_min # - self.\_delta / 2 # x(нач)  
 x\_end = self.\_x\_max + self.\_delta / 2 # Catch the biggest value  
   
 intervals = []  
 for i in range(self.\_k): # Number of interval  
 bias = i \* self.\_delta  
 left = round(x\_start + bias, p)  
 right = round(x\_start + self.\_delta + bias, p)  
 intervals += [(left, right)]  
 last\_interval = intervals[-1]  
 new\_last\_interval = (last\_interval[0], x\_end)  
 intervals[-1] = new\_last\_interval  
 return intervals  
  
 def \_\_make\_vs(self) -> dict:  
 *"""Return dict with variable series"""* vs = {} # Frequencies  
 for val in sorted(self.\_values):  
 # Choose the correct interval  
 for interval in self.\_intervals:  
 left, right = interval  
 if left <= val < right or val == self.\_x\_max:  
 vs[interval] = vs.get(interval, 0) + 1  
 break  
 return vs  
  
 def draw\_hist(self, nrows, ncols, index):  
 *"""Prepare histogram function"""* xs = self.get\_hist\_xs()  
 ys = self.get\_hist\_ys()  
 plt.subplot(nrows, ncols, index)  
 plt.plot(xs, ys, 'b')  
 plt.xlabel('Интервалы')  
 plt.ylabel('Частоты')  
 return self  
  
 def get\_hist\_xs(self):  
 xs = []  
 for left, right in (interval for interval in self.\_vs.keys()):  
 xs += [left, right]  
 return xs  
  
 def get\_hist\_ys(self):  
 ys = []  
 for y in self.\_var\_frequencies:  
 ys += [y, y]  
 return ys  
  
 def get\_cumulate\_xs(self): # a(i)  
 xs = [interval[0] for interval in self.\_vs.keys()] + [self.\_intervals[-1][1]]  
 return xs  
  
 def get\_cumulate\_ys(self): # m(a(i))  
 return self.\_acc\_frequencies  
  
 def get\_empiric\_dist\_xs(self):  
 return self.get\_cumulate\_xs()  
  
 def get\_empiric\_dist\_ys(self):  
 return self.\_acc\_rel\_frequencies