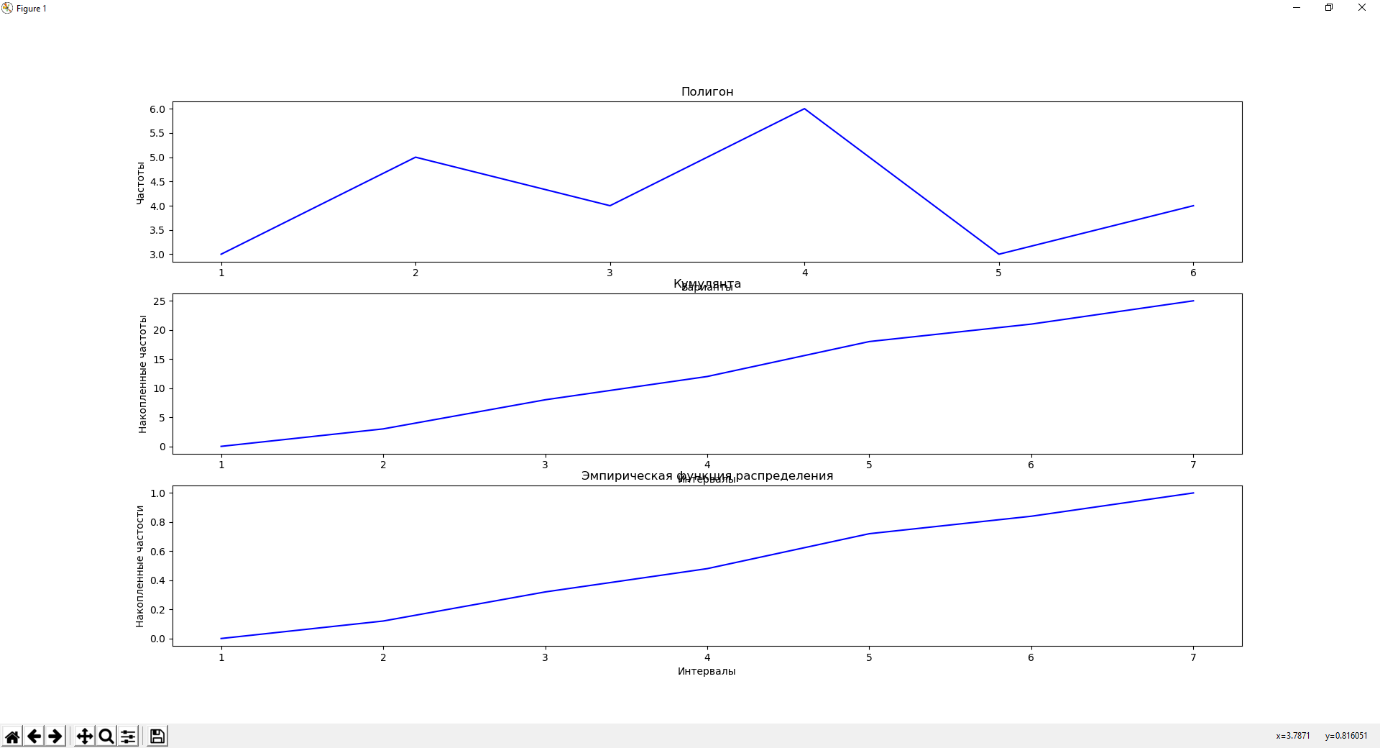
**Задачи к лабораторной работе №2**

Задача №1

Постановка задачи: приводятся данные о распределении 25 работников одного из предприятий по тарифным разрядам: 4; 2; 4; 6; 5; 6; 4; 1; 3; 1; 2; 5; 2; 6; 3; 1; 2; 3; 4; 5; 4; 6; 2; 3; 4. Построить дискретный вариационный ряд и изобразить его графически.

Результаты:



Код программы:

def task1():

    values = [

        4, 2, 4, 6, 5, 6, 4, 1, 3, 1, 2, 5, 2, 6, 3, 1, 2, 3, 4, 5, 4, 6, 2, 3, 4

    ]

    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 №2

Постановка задачи: приведены данные о размерах вкладов 20 физических лиц в одном банке (тыс.руб): 60; 25; 12; 10; 68; 35; 2; 17; 51; 9; 3; 130; 24; 85; 100; 152; 6; 18; 7; 42. Построить интервальный вариационный ряд с равными интервалами.

Результаты:



Код программы:

def task2():

    values = [

        60, 25, 12, 10, 68, 35, 2, 17, 51, 9, 3, 130, 24, 85, 100, 152, 6, 18, 7, 42

    ]

    VariationSeries.PRECISION = 2

    v = ContinuousVS(values)

    print(v)

Задача №3

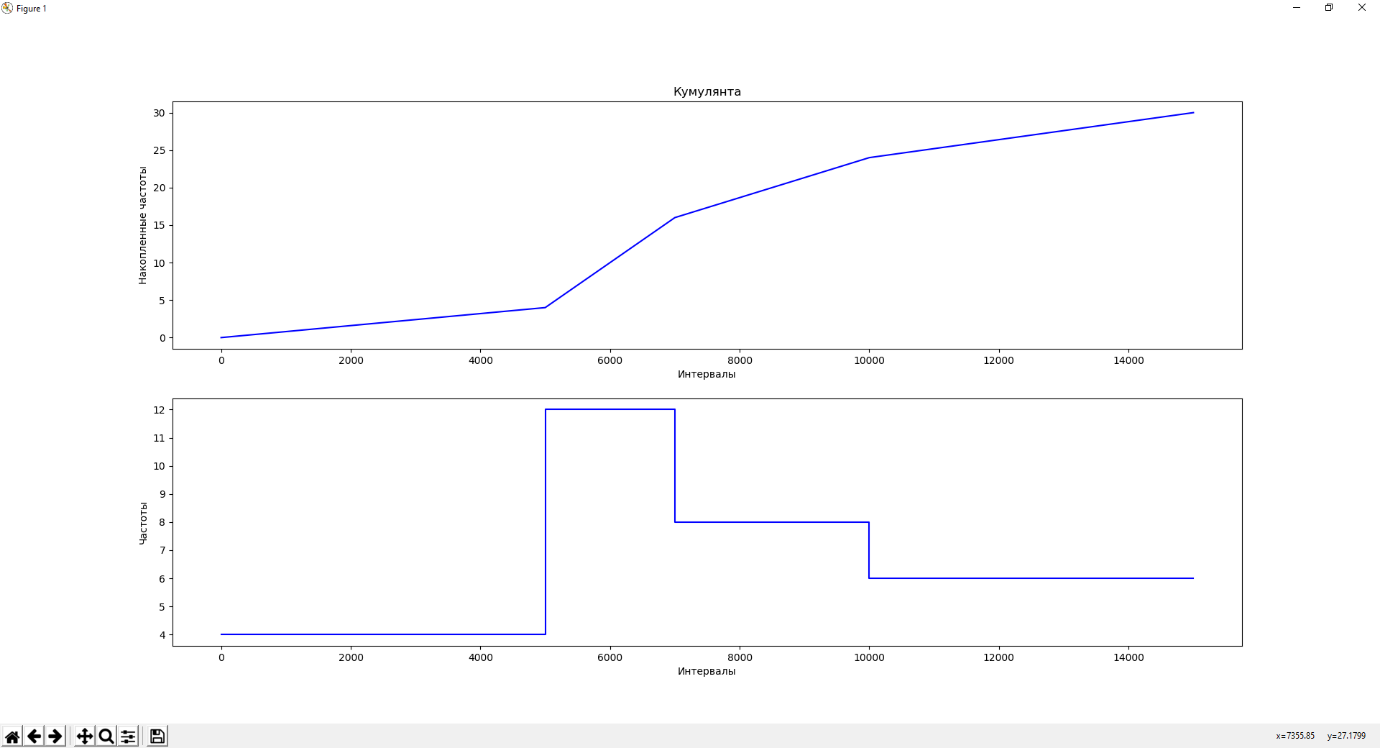
Постановка задачи: приводится распределение 30 работников фирмы по размеру месячной заработной платы



Построить и изобразить интервальный вариационный ряд графически в виде гистограммы и

кумуляты.

Результаты:



Код программы:

def task3():

    values = {

        (0, 5\*10\*\*3): 4, (5\*10\*\*3, 7\*10\*\*3): 12,

        (7\*10\*\*3, 10\*\*4): 8, (10\*\*4, 1.5\*10\*\*4): 6

    }

    VariationSeries.PRECISION = 2

    v = ContinuousVS(values)

    v.draw\_cumulate(2, 1, 1)

    v.draw\_hist(2, 1, 2)

    ContinuousVS.show()

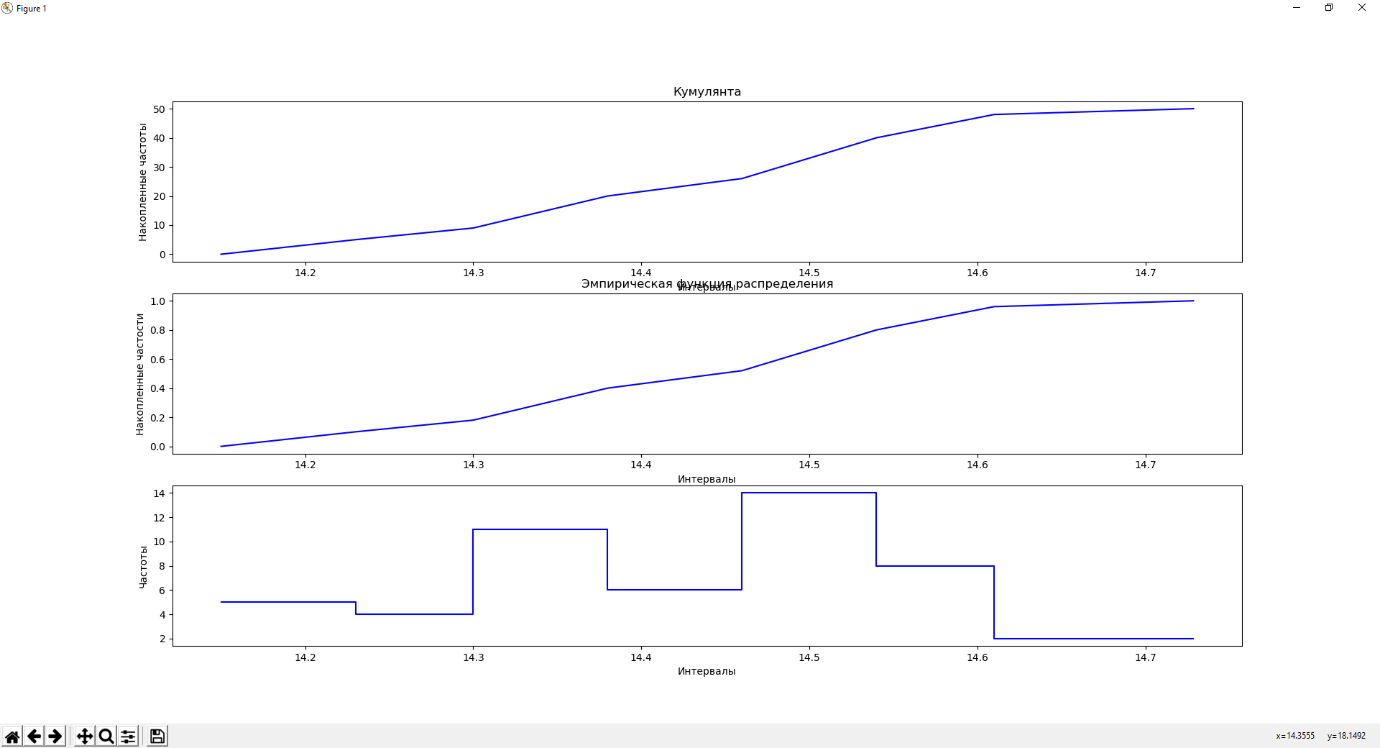
Задача №4

Постановка задачи: Измерения диаметров 50 валиков, выточенных на станке, дали следующие

результаты (в мм): 14,51 14,42 14,56 14,47 14,46 14,35 14,48 14,53 14,21 14,31 14,35 14,68 14,56 14,28 14,36 14,21 14,52 14,23 14,41 14,46 14,69 14,54 14,36 14,15 14,37 14,51 14,25 14,55 14,51 14,36 14,62 14,55 14,38 14,33 14,40 14,52 14,48 14,51 14,55 14,39 14,54 14,58 14,48 14,37 14,38 14,51 14,36 14,15 14,24 14,32

Построить интервальный вариационный ряд и графически отобразить

Результаты:



Код программы:

def task4():

    values = [

        14.51, 14.42, 14.56, 14.47, 14.46, 14.35, 14.48, 14.53,

        14.21, 14.31, 14.35, 14.68, 14.56, 14.28, 14.36, 14.21,

        14.52, 14.23, 14.41, 14.46, 14.69, 14.54, 14.36, 14.15,

        14.37, 14.51, 14.25, 14.55, 14.51, 14.36, 14.62, 14.55,

        14.38, 14.33, 14.40, 14.52, 14.48, 14.51, 14.55, 14.39,

        14.54, 14.58, 14.48, 14.37, 14.38, 14.51, 14.36, 14.15,

        14.24, 14.32

    ]

    VariationSeries.PRECISION = 2

    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()

**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