Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Аттестационная работа
по дисциплине: **Математическая статистика**
Тема: Выборочный метод. Элементы теории корреляции

Выполнил
студент 2 курса
Корнасевич И. Д.

Проверил
Юхимук Т. Ю.

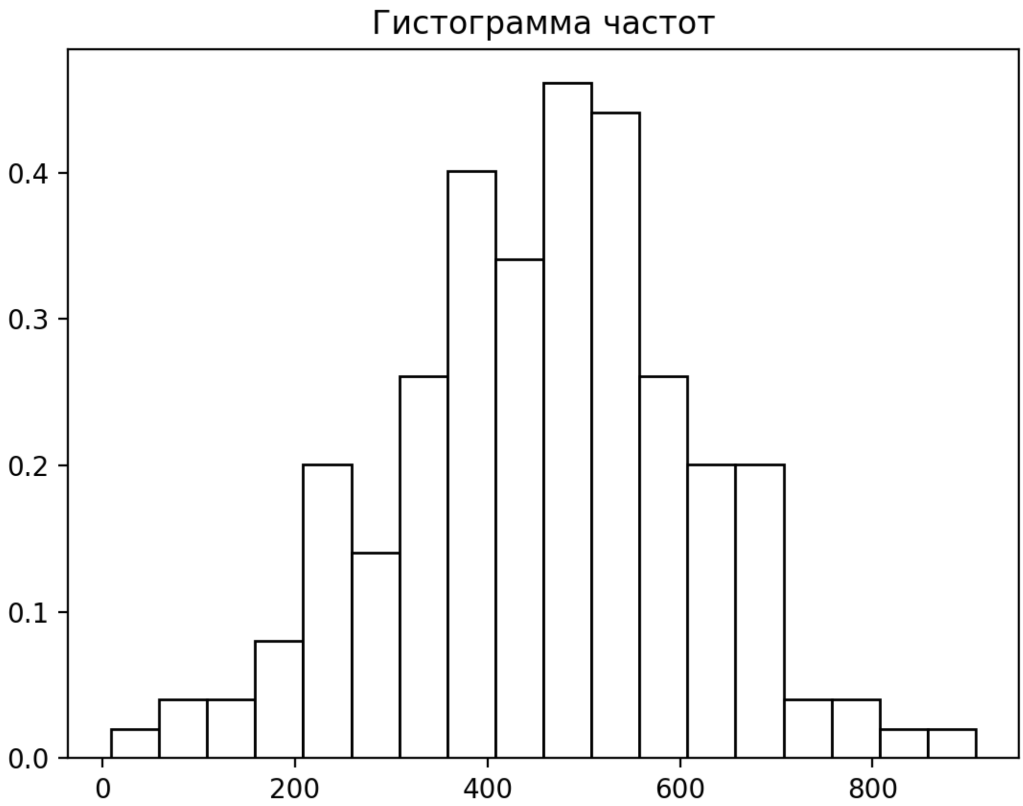
Брест 2021

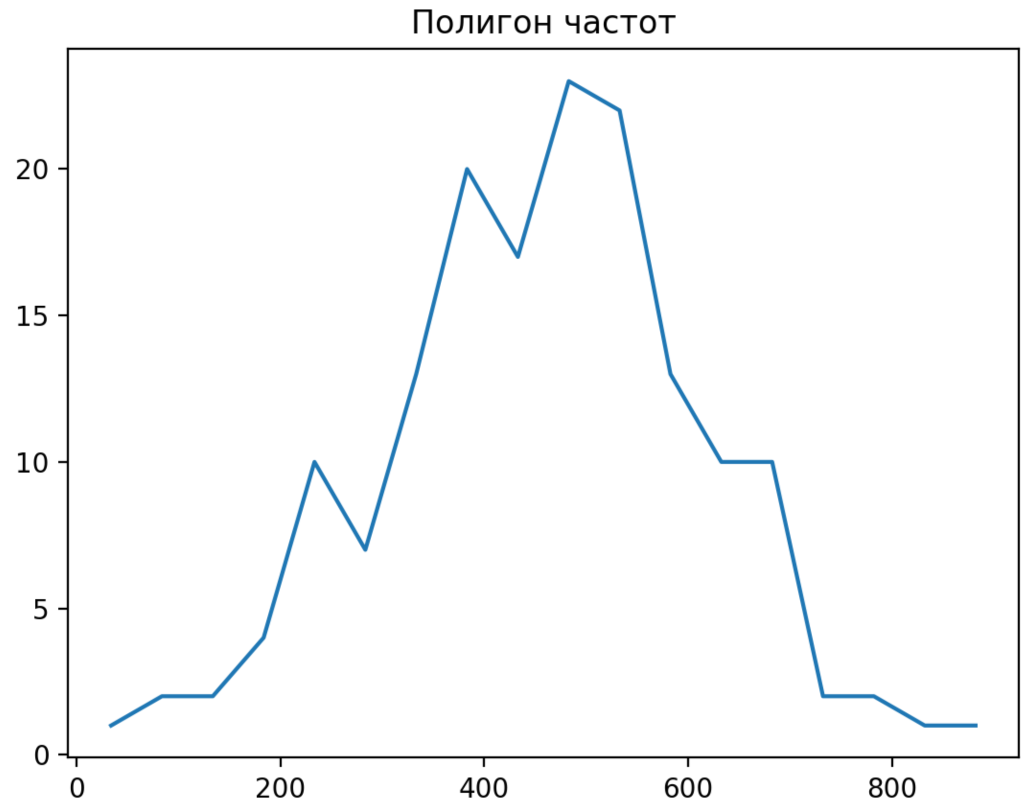
Задание 1 Изначальная выборка:

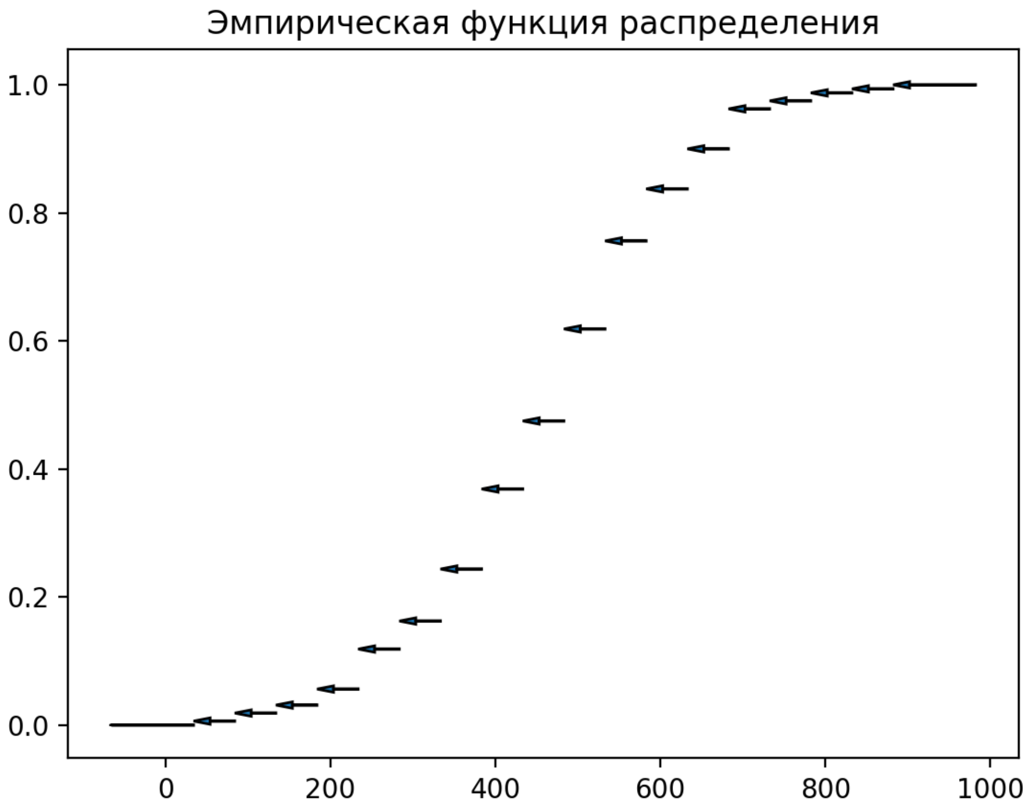
596.33 383.19 647.44 477.37 792.69 482.05 400.08 381.28 313.42 900.00 463.59 488.34 471.48 475.06 414.22 321.03 534.87 667.84 472.06 593.00 583.34 454.50 397.68 553.65 684.05 796.19 715.45 688.49 324.96 661.44 559.07 698.96 382.79 568.19 627.06 552.12 338.17 243.64 599.94 548.57 532.31 209.61 611.21 320.79 717.78 567.37 403.42 705.66 624.11 471.04 354.48 429.41 388.35 670.01 438.85 621.10 611.86 662.92 362.38 358.76 398.89 273.35 469.37 434.85 494.38 188.17 477.30 216.81 190.81 408.04 9.00 240.83 585.51 705.23 564.10 558.02 400.77 428.32 405.27 221.73 496.03 372.93 428.31 457.70 518.59 508.23 342.10 314.49 417.41 461.55 517.25 584.51 435.63 484.29 808.20 506.08 359.76 489.47 235.19 393.88 542.45 656.96 518.78 537.45 465.59 551.23 329.04 127.73 498.11 469.92 268.03 74.54 491.10 289.69 354.53 533.77 552.07 231.61 517.18 435.38 59.80 225.38 468.84 528.83 161.00 259.54 151.32 260.25 544.46 374.75 317.44 374.20 595.16 516.74 290.70 455.10 392.82 532.09 269.01 324.98 552.81 414.07 388.27 632.44 670.02 249.29 642.66 641.95 318.12 482.56 532.18 489.71 445.34 224.29 594.73 425.48 437.70 185.61 543.90 455.33

$$h = \frac{x_{\max} - x_{\min}}{1 + 3.322 \cdot \ln n} = 48.889$$

$[x_i..x_{i+1}]$		n_i	W_i	W_i/h_i	$F'(x)$	x_i
9	58.889	1	0.006	0.020	0.006	33.945
58.889	108.778	2	0.013	0.040	0.019	83.834
108.778	158.667	2	0.013	0.040	0.031	133.722
158.667	208.556	4	0.025	0.080	0.056	183.612
208.556	258.445	10	0.062	0.200	0.119	233.500
258.445	308.334	7	0.044	0.140	0.162	283.389
308.334	358.223	13	0.081	0.261	0.244	333.279
358.223	408.112	20	0.125	0.401	0.369	383.168
408.112	458.001	17	0.106	0.341	0.475	433.057
458.001	507.890	23	0.144	0.461	0.619	482.946
507.890	557.779	22	0.138	0.441	0.756	532.835
557.779	607.668	13	0.081	0.261	0.837	582.724
607.668	657.557	10	0.062	0.200	0.900	632.612
657.557	707.446	10	0.062	0.200	0.962	682.502
707.446	757.335	2	0.013	0.040	0.975	732.390
757.335	807.224	2	0.013	0.040	0.987	782.280
807.224	857.113	1	0.006	0.020	0.994	832.168
857.113	907.002	1	0.006	0.020	1.000	882.058


Гистограмма частот
0.0
0.1
0.2
0.3
0.4
0
200
400
600
800


Полигон частот
0
5
10
15
20
0
200
400
600
800



$$\bar{x_s} = 457.3773$$
$$\bar{D}_s = 24017.6150$$
$$\sigma = 154.9761$$
$$S^2 = 24168.6691$$
$$S = 155.4627$$

Задание 2 Доверительный интервал для мат ожидания:

$$\bar{x_s} - t\frac{s}{\sqrt{n}} < \bar{x_{tr}} < \bar{x_s} + t\frac{s}{\sqrt{n}}$$
$$t_{0.025;159} = 1.975$$
$$\bar{x_{tr}} \in (433.1038; 481.6509)$$

Доверительный интервал среднего квадратического отклонения генеральной совокупности:

$$s \cdot q_1 < \sigma_{tr} < s \cdot q_2$$
$$q_1 = \sqrt{\frac{n-1}{\chi_{0.025;159}}}$$
$$q_2 = \sqrt{\frac{n-1}{\chi_{0.975;159}}}$$
$$\sigma_{tr} \in (142.0230; 170.7810)$$

$$(\alpha - 3\sigma; \alpha + 3\sigma) \approx (\bar{x_s} - 3s; \bar{x_s} + 3s) \approx (-9.0108; 923.7656)$$

Выборочные данные удовлетворяют правилу 3σ нормального распределения.

$$p_i = P(x_{i-1} < X < xi) = \Phi(\frac{x_i - \bar{x}}{\sigma}) - \Phi(\frac{x_{i-1} - \bar{x}}{\sigma})$$

$[x_i..x_{i+1}]$		Частоты n_i	Выравнивающие частоты $n' = np_i$
9	58.889	1	0.505
58.889	108.778	2	1.149
108.778	158.667	2	2.355
158.667	208.556	4	4.356
208.556	258.445	10	7.272
258.445	308.334	7	10.954
308.334	358.223	13	14.889
358.223	408.112	20	18.261
408.112	458.001	17	20.211
458.001	507.890	23	20.185
507.890	557.779	22	18.191
557.779	607.668	13	14.793
607.668	657.557	10	10.856
657.557	707.446	10	7.188
707.446	757.335	2	4.295
757.335	807.224	2	2.316
807.224	857.113	1	1.127
857.113	907.002	1	0.495

Частоты n_i	Выравнивающие частоты $n' = np_i$	$(n'_i - n_i)/n'_i$
5	4.009	0.245
4	4.356	0.029
10	7.272	1.024
7	10.954	1.427
13	14.889	0.240
20	18.261	0.166
17	20.211	0.510
23	20.185	0.392
22	18.191	0.797
13	14.793	0.217
10	10.856	0.067
10	7.188	1.100
6	8.232	0.605

$\chi^2_{watch} = \sum(n'_i - n_i)/n'_i = 6.8199$ Для уровня значимости $\alpha = 0.05$ и $k = 10$ соответствует значение $\chi^2_{crit} = 18.3070$.

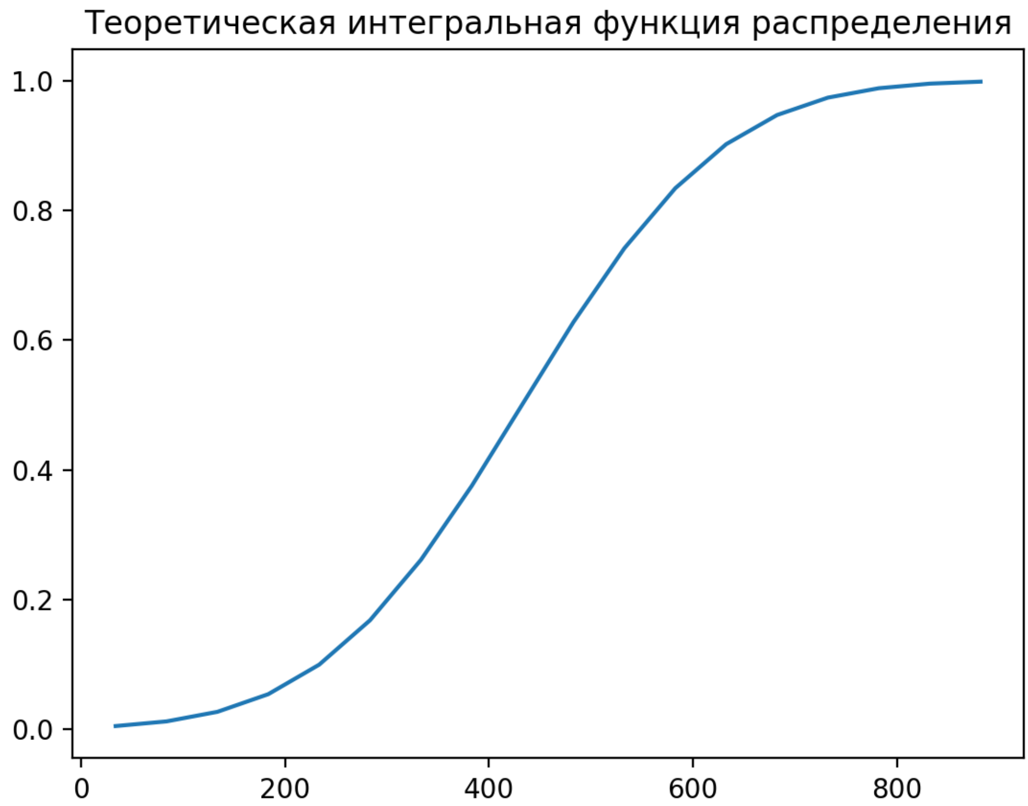
Частоты	Эмпирическая функция распределения $F'(x)$	Теоретическая функция распределения $F(x)$	Разности $\lvert F'(x) - F(x)\rvert$
1	0.006	0.005	0.001
2	0.019	0.012	0.007
2	0.031	0.027	0.004
4	0.056	0.054	0.002
10	0.119	0.100	0.019
7	0.162	0.168	0.006
13	0.244	0.261	0.017
20	0.369	0.375	0.007
17	0.475	0.502	0.027
23	0.619	0.628	0.009
22	0.756	0.741	0.015
13	0.837	0.834	0.004
10	0.900	0.902	0.002
10	0.962	0.947	0.016
2	0.975	0.974	0.001
2	0.987	0.988	0.001
1	0.994	0.995	0.001
1	1.000	0.998	0.002

$$\lambda_{watch} = \sqrt{n} \cdot \max_{x} |F'(x) - F(x)| = 0.3365$$
$$\lambda_{crit} = 1.36$$

Так как:

$$\chi^2_{watch} < \chi^2_{crit}$$
$$\lambda_{watch} < \lambda_{crit}$$

Нет оснований отвергать гипотезу о нормальном распределении.



Задание 3 Для начала нужно заполнить таблицы:

	Y	18.500	19.700	20.900	22.100	23.300	24.500	25.700	26.900
X	$u \backslash v$	-3.500	-2.500	-1.500	-0.500	0.500	1.500	2.500	3.500
12.500	-2.500	4	8	6					
20	-1.500		7	14	8				
27.500	-0.500				15	13	7		
35	0.500				9	18	9	6	
42.500	1.500						9	5	1
50	2.500							6	3

n_i	$n_i u_i$	$\sum n_{ij} v_j$	$n_i u_i^2$	$u_i \sum n_{ij} v_j$
18	-45	-43	112.500	107.500
29	-43.500	-42.500	65.250	63.750
35	-17.500	9.500	8.750	-4.750
42	21	33.000	10.500	16.500
15	22.500	29.500	33.750	44.250
9	22.500	25.500	56.250	63.750

m_i	$m_i v_i$	$\sum n_{ij} u_i$	$m_j v_j^2$	$v_j \sum n_{ij} u_i$
-3.500	-14.000	-10	49.000	35.000
-2.500	-37.500	-30.500	93.750	76.250
-1.500	-30.000	-36	45.000	54.000
-0.500	-16.000	-15	8.000	7.500
0.500	15.500	2.500	7.750	1.250
1.500	37.500	14.500	56.250	21.750
2.500	42.500	25.500	106.250	63.750
3.500	14.000	9	49.000	31.500

Выборочные средние X и Y:

$$\bar{u} = \frac{\sum n_i u_i}{n} = -0.2702$$
$$\bar{v} = \frac{\sum m_j v_j}{n} = 0.0810$$
$$\bar{x} = 29.2229$$
$$\bar{y} = 22.7972$$

Дисперсии признаков X и Y:

$$\sigma_u = \sqrt{\bar{u^2} - (\bar{u})^2 = 1.3660}$$
$$\sigma_v = \sqrt{\bar{v^2} - (\bar{v})^2 = 1.6725}$$
$$\sigma_x = \sigma_u h_x = 10.2455$$
$$\sigma_y = \sigma_v h_y = 2.0070$$

Таблицы условных средних:

X	12.500	20	27.500	35	42.500	50
$\bar{x_y}$	19.833	20.941	23.026	23.643	25.060	26.100

Y	18.500	19.700	20.900	22.100	23.300	24.500	25.700	26.900
$\bar{y_x}$	12.500	16	17.750	27.734	31.855	35.600	42.500	48.125

Коэффициент корреляции признаков X и Y совпадает с коэффициентом корреляции условных вариант:

$$r = \frac{1}{\sigma_u \sigma_v}\left(\frac{\sum n_{ij} u_i v_j}{n} - \bar{u}\bar{v}\right) = 0.9328$$

Следовательно коэффициент детерминации $r^2 = 0.8701$. Значит 87% рассеивания зависимой переменной объясняется линейной регрессией Y на X.

$$\sigma_{\bar{x}}^2 = \sqrt{\frac{1}{n}\sum(\bar{x}_i - \bar{x})^2 n_i} = 9.0427$$

$$\sigma_{\bar{y}} = \sqrt{\frac{1}{n}\sum(\bar{y}_i - \bar{y})^2 n_i} = 1.7722$$

$$\eta_{Y/X} = \frac{\sigma_{\bar{y}}}{\sigma_y} = 0.88299$$

$$\eta_{Y/X} = \frac{\sigma_{\bar{x}}}{\sigma_x} = 0.88260$$

Эмпирическая линейная регрессия Y на X:

$$\bar{y}_x - \bar{y} = b_1(x - \bar{x})$$

$$b = r\frac{\sigma_y}{\sigma_x} = 0.1827$$

$$y - 22.7972 = 0.1827(x - 29.2229)$$

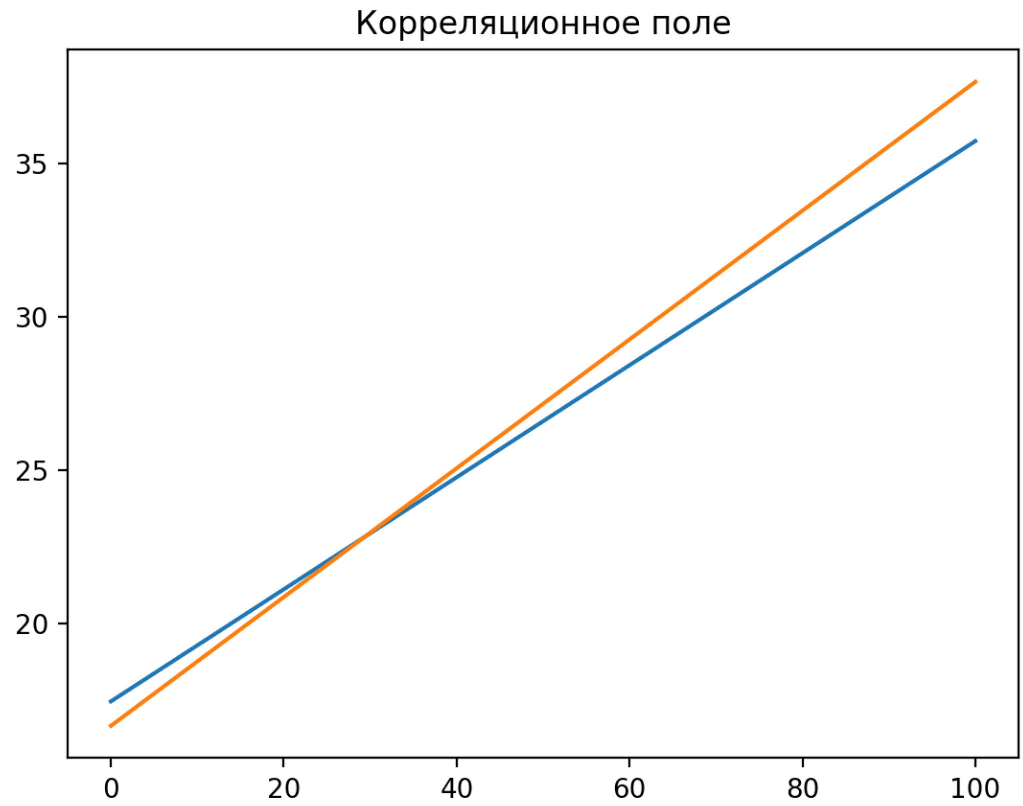
$$y = 0.1827x + 17.4586$$

Эмпирическая линейная регрессия X на Y:

$$\bar{x}_y - \bar{x} = a(y - \bar{y})$$

$$a = r\frac{\sigma_x}{\sigma_y} = 4.7617$$

$$x - 29.2229 = 4.7617(y - 22.7972)$$



95% доверительный интервал для коэффициентов регрессии. Если уравнение имеет вид $\bar{y}_x = b_0 + b_1 x$, то $b_0 \in (-0.1754; 0.5409)$, а $b_1 \in (17.4456; 17.8154)$

Для оценки значимости выборочного коэффициента вычислим:

$$t_{watch} = r\sqrt{\frac{n-2}{1-r^2}} = 31.2775$$

$$t_{crit} = 1.98$$

$$t_{watch} > t_{crit}$$

Нулевую гипотезу отвергаем. Коэффициент корреляции значимо отличается от нуля.

data.py

```
import numpy as np

initArray = np.sort(
    np.array(
        [
            596.33, 383.19, 647.44, 477.37, 792.69, 482.05, 400.08,
            381.28, 313.42, 900.00, 463.59, 488.34, 471.48, 475.06,
            414.22, 321.03, 534.87, 667.84, 472.06, 593.00, 583.34,
            454.50, 397.68, 553.65, 684.05, 796.19, 715.45, 688.49,
            324.96, 661.44, 559.07, 698.96, 382.79, 568.19, 627.06,
            552.12, 338.17, 243.64, 599.94, 548.57, 532.31, 209.61,
            611.21, 320.79, 717.78, 567.37, 403.42, 705.66, 624.11,
            471.04, 354.48, 429.41, 388.35, 670.01, 438.85, 621.10,
            611.86, 662.92, 362.38, 358.76, 398.89, 273.35, 469.37,
            434.85, 494.38, 188.17, 477.30, 216.81, 190.81, 408.04,
```

```
            9.00, 240.83, 585.51, 705.23, 564.10, 558.02, 400.77,
            428.32, 405.27, 221.73, 496.03, 372.93, 428.31, 457.70,
            518.59, 508.23, 342.10, 314.49, 417.41, 461.55, 517.25,
            584.51, 435.63, 484.29, 808.20, 506.08, 359.76, 489.47,
            235.19, 393.88, 542.45, 656.96, 518.78, 537.45, 465.59,
            551.23, 329.04, 127.73, 498.11, 469.92, 268.03, 74.54,
            491.10, 289.69, 354.53, 533.77, 552.07, 231.61, 517.18,
            435.38, 59.80, 225.38, 468.84, 528.83, 161.00, 259.54,
            151.32, 260.25, 544.46, 374.75, 317.44, 374.20, 595.16,
            516.74, 290.70, 455.10, 392.82, 532.09, 269.01, 324.98,
            552.81, 414.07, 388.27, 632.44, 670.02, 249.29, 642.66,
            641.95, 318.12, 482.56, 532.18, 489.71, 445.34, 224.29,
            594.73, 425.48, 437.70, 185.61, 543.90, 455.33
        ]
    )
)
```

task.py

```
from scipy import *
from scipy import stats

from data import initArray
import matplotlib.pyplot as plt
import numpy as np
from latextable import *


def task():
    n = initArray.size
    step = (initArray.max() - initArray.min()) / (1 + 3.322 * np.log(initArray.size))
    step = np.round(step, 3)
    intervals = create_intervals(step)
    intervalMiddles = create_interval_middles(intervals)
    frequencies = create_frequencies(intervals)
    relativeFrequencies = np.array([x / initArray.size for x in frequencies])
    frequenciesOfDensities = np.array([x / step for x in frequencies])
    distributionFunction = create_distribution_function(relativeFrequencies)
    Exini = np.sum(np.array([intervalMiddles[i] * frequencies[i] for i in range(intervalMiddles.size)]))
    Exi2ni = np.sum(np.array([(intervalMiddles[i] ** 2) * frequencies[i] for i in range(intervalMiddles.size)]))
    En = np.sum(frequencies)
    X = Exini / En
    D = (Exi2ni / En) - (X ** 2)
    sigma = D ** 0.5
    S2 = En / (En - 1) * D
    S = S2 ** 0.5

    a = 0.05

    t = 1.975
```

```
    alpha = (X - t * S / (n ** 0.5), X + t * S / (n ** 0.5))
    print("Доверительный интервал alpha:", alpha)
    delta = (np.sqrt((n - 1) / stats.chi2.ppf((1 - a)/2, df=n)) * S, np.sqrt((n - 1) / stats.chi2.ppf(a / 2, df=n)) * S)
    print("Доверительный интервал delta:", delta)

    laplassianAplphas = np.array([(intervals[i] - X) / sigma for i in range(intervals.size)])
    laplassians = np.array(
        [laplas_integral(laplassianAplphas[i]) - laplas_integral(laplassianAplphas[i - 1])
         for i in range(1, intervals.size)])

    npi = np.array([x * n for x in laplassians])
    merge_config = create_merge_config(5, frequencies)
    mergedFrequencies = merge_limits(merge_config, frequencies)
    mergedNpi = merge_limits(merge_config, npi)

    niNpiDeltas = np.array([mergedNpi[i] - mergedFrequencies[i] for i in range(mergedNpi.size)])
    niNpiSquaredDeltas = np.array([(mergedNpi[i] - mergedFrequencies[i]) ** 2 for i in range(mergedNpi.size)])
    niNpiSquaredDividedDeltas = np.array([niNpiSquaredDeltas[i] / mergedNpi[i] for i in range(mergedNpi.size)])

    xSquaredWatched = np.sum(niNpiSquaredDividedDeltas)
    k = niNpiDeltas.size - 2 - 1

    print("k =", k)
    X2Crit = stats.chi2.ppf(1 - a, df=k)
    print("X^2Watched =", xSquaredWatched)
    print("X^2Crit =", X2Crit, xSquaredWatched < X2Crit)

    hypotheticalFunction = np.array([laplas_integral(laplassianAplphas[i]) for i in range(1, laplassianAplphas.size)])
    hypotheticalDistributionFunctionDifferences = np.array(
        [np.abs(distributionFunction[i] - hypotheticalFunction[i])
         for i in range(distributionFunction.size)])
    maxDifference = np.max(hypotheticalDistributionFunctionDifferences)
    print(hypotheticalFunction)
    print(distributionFunction)

    lambdaWatched = (n ** 0.5) * maxDifference
    print("lambdaWatched =", lambdaWatched)
    lambdaCrit = 1.36
    print("lambdaCrit =", lambdaCrit, lambdaWatched < lambdaCrit)

    print(X - 3*S, X + 3 * S)

    def print_values1():
        print(f"Xв = {X}")
        print(f"Dв = {D}")
        print(f"Dв ^ 0.5 = {D ** 0.5}")
        print(f"S ^ 2 = {S2}")
        print(f"S = {S}")
```

```

    def draw_graphs1():
        plt.figure(dpi=200)
        plt.bar(intervalMiddles, frequenciesOfDensities, width=step, edgecolor="black", color='white')
        plt.title('Гистограмма частот')
        plt.savefig("gr11.png")

        plt.figure(dpi=200)
        plt.plot(intervalMiddles, frequencies)
        plt.title('Полигон частот')
        plt.savefig("gr12.png")

        plt.figure(dpi=200)
        x = np.insert(intervalMiddles, 0, [intervalMiddles[0] - 100])
        x = np.append(x, [intervalMiddles[-1] + 100])
        y = np.insert(distributionFunction, 0, [0])
        plt.title('Эмпирическая функция распределения')
        for i in range(1, y.size):
            plt.arrow(x[i + 1], y[i], x[i] - x[i + 1] + 20, 0, head_width=0.01, head_length=20)
        plt.arrow(x[1], y[0], x[0] - x[1], 0)
        plt.savefig("gr13.png")

    def draw_graphs2():
        plt.figure(dpi=200)
        plt.title('Плотность вероятности нормального распределения')
        x = np.linspace(X - 3 * sigma, X + 3 * sigma, 100)
        plt.plot(x, stats.norm.pdf(x, X, sigma))
        plt.savefig("gr21.png")

        plt.figure(dpi=200)
        plt.title('Теоретическая интегральная функция распределения')
        plt.plot(intervalMiddles, hypotheticalFunction)
        plt.savefig("gr22.png")

    def print_tables1():
        table = Texttable()
        table.set_cols_align(["l", "l", "l", "l", "l", "l", "l"])
        table.header(['start', 'end', 'ni', 'Wi', 'Wi / h', 'F\'(x)', 'xi'])
        for i in range(intervalMiddles.size):
            table.add_row(
                [intervals[i], intervals[i + 1], frequencies[i], relativeFrequencies[i],
                 frequenciesOfDensities[i], distributionFunction[i], intervalMiddles[i]])
        print(draw_latex(table))

    def print_tables2():
        table = Texttable()
        table.set_cols_align(["l", "l", "l", "l"])
        table.header(['start', 'end', 'ni', 'n\' = npi'])
        for i in range(intervalMiddles.size):
            table.add_row(
                [intervals[i], intervals[i + 1], frequencies[i], npi[
```

```
i]])
        print(draw_latex(table))

        table = Texttable()
        table.set_cols_align(["l", "l", "l"])
        table.header(['Частоты', 'Выравнивающие частоты', '(n\'i - ni) / n\'i'])
        for i in range(mergedFrequencies.size):
            table.add_row(
                [mergedFrequencies[i], mergedNpi[i], niNpiSquaredDividedDeltas[i]])
        print(draw_latex(table))

        table = Texttable()
        table.set_cols_align(["l", "l", "l", "l"])
        table.header(
            ['Частоты', 'Эмпирическая функция распределения', 'Теоретическая функция распределения', 'Разности'])
        for i in range(intervalMiddles.size):
            table.add_row(
                [frequencies[i], distributionFunction[i], hypotheticalFunction[i],
                  hypotheticalDistributionFunctionDifferences[i]])
        print(draw_latex(table))

    print_values1()
    print_tables1()
    draw_graphs1()
    draw_graphs2()
    print_tables2()


def create_merge_config(count: int, array: np.ndarray) -> []:
    left = 0
    right = 0
    acc = 0
    for i in range(array.size):
        acc += array[i]
        if acc >= count:
            left = i + 1
            break
    acc = 0
    for i in reversed(range(array.size)):
        acc += array[i]
        if acc >= count:
            right = i
            break
    return [left, right]


def merge_limits(config: [], arr: np.ndarray):
    arr = np.split(arr, config)
    return np.array([np.sum(arr[0]), *arr[1], np.sum(arr[2])])


def laplas_integral(x: float) -> float:
```

```
    return stats.norm.cdf(x)


def gaussian(x: float) -> float:
    return 1 / np.sqrt(2 * np.pi) * np.exp(-x ** 2 / 2)


def create_frequencies(intervals: np.ndarray) -> np.ndarray:
    frequencies = np.zeros(intervals.size - 1)
    for i in range(intervals.size - 1):
        if i == intervals.size - 1:
            ni = np.count_nonzero((initArray >= intervals[i]) & (
initArray <= intervals[i + 1]))
        else:
            ni = np.count_nonzero((initArray >= intervals[i]) & (
initArray < intervals[i + 1]))
        frequencies[i] = ni
    return frequencies


def create_intervals(step: float) -> np.ndarray:
    intervals = np.array([initArray.min()])
    while intervals.max() < initArray.max():
        intervals = np.append(intervals, intervals.max() + step)
    return intervals


def create_interval_middles(intervals: np.ndarray) -> np.ndarray:
    intervalMiddles = np.zeros(intervals.size - 1)
    for i in range(intervals.size - 1):
        intervalMiddles[i] = (intervals[i] + intervals[i + 1]) / 2
    return intervalMiddles


def create_distribution_function(relative_frequencies: np.ndarray) ->
 np.ndarray:
    distributionFunction = np.zeros(relative_frequencies.size)
    distributionFunction[0] = relative_frequencies[0]
    for i in range(1, relative_frequencies.size):
        distributionFunction[i] = distributionFunction[i - 1] +
relative_frequencies[i]
    return distributionFunction


if __name__ == "__main__":
    task()
```

task3.py

```
import matplotlib.pyplot as plt
import numpy as np
from latextable import draw_latex
from scipy import stats
from texttable import Texttable

matrix = np.asarray([
```

```
    [4, 8, 6, 0, 0, 0, 0, 0],
    [0, 7, 14, 8, 0, 0, 0, 0],
    [0, 0, 0, 15, 13, 7, 0, 0],
    [0, 0, 0, 9, 18, 9, 6, 0],
    [0, 0, 0, 0, 0, 9, 5, 1],
    [0, 0, 0, 0, 0, 0, 6, 3],
])

stepX = 7.5
stepY = 1.2


def task():
    n = np.sum(matrix)
    xAxis = create_axis(12.5, stepX, 6)
    yAxis = create_axis(18.5, stepY, 8)
    Ui = np.array((xAxis - np.average(xAxis)) / stepX)
    Vj = np.array((yAxis - np.average(yAxis)) / stepY)
    Ni = np.array(matrix.sum(1)).flatten()
    Mj = np.array(matrix.sum(0)).flatten()
    NiUi = np.array(Ni * Ui)
    MjVj = np.array(Mj * Vj)

    sumNijVj = np.zeros(matrix.shape[0])
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            sumNijVj[i] += Vj[j] * matrix.item((i, j))

    sumNijUi = np.zeros(matrix.shape[1])
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            sumNijUi[j] += Ui[i] * matrix.item((i, j))

    MjVjpow2 = np.array(Mj * (Vj ** 2))
    NiUipow2 = np.array(Ni * (Ui ** 2))

    UiSumNijVj = np.array(sumNijVj * Ui)
    ViSumNijUi = np.array(sumNijUi * Vj)

    barU = NiUi.sum() / n
    barV = MjVj.sum() / n

    barX = np.average(xAxis) + stepX * barU
    barY = np.average(yAxis) + stepY * barV

    sigmaU = (NiUipow2.sum() / n - barU ** 2) ** 0.5
    sigmaV = (MjVjpow2.sum() / n - barV ** 2) ** 0.5

    sigmaX = sigmaU * stepX
    sigmaY = sigmaV * stepY

    rSquare = 1 / (sigmaU * sigmaV) * (UiSumNijVj.sum() / n - barU * barV)
    r = rSquare ** 0.5

    barYs = np.array([(matrix[i:i + 1] * yAxis).sum() / matrix[i:i +
```

```
1].sum() for i in range(matrix.shape[0])])
    barXs = np.array(
        [(matrix[:, i:i + 1].flatten() * xAxis).sum() / matrix[:, i:i + 1].sum() for i in range(matrix.shape[1])])

    sigmaBarX = ((1 / n) * np.sum(Mj * (barXs - barX) ** 2)) ** 0.5
    sigmaBarY = ((1 / n) * np.sum(Ni * (barYs - barY) ** 2)) ** 0.5

    NuyYtoX = sigmaBarY / sigmaY
    NuyXtoY = sigmaBarX / sigmaX

    XtoYRegressionA = r * sigmaX / sigmaY
    YtoXRegressionB = r * sigmaY / sigmaX

    B0 = YtoXRegressionB
    B1 = barY - barX * YtoXRegressionB
    a = 0.05
    t = 1.98
    sigmaYX = (n * sigmaY ** 2 * (1 - rSquare) / (n - 2)) ** 0.5
    bigSquareRoot = (1 / sigmaX) * ((sigmaX ** 2 + barX ** 2) / n) ** 0.5
    bigSquareRoot2 = 1 / (sigmaX * n ** 0.5)
    B0Interval = (B0 - t * sigmaYX * bigSquareRoot, B0 + t * sigmaYX * bigSquareRoot)
    B1Interval = (B1 - t * sigmaYX * bigSquareRoot2, B1 + t * sigmaYX * bigSquareRoot)

    TWatch = r * ((n - 2) / (1 - rSquare)) ** 0.5

    TCrit = 1.984

    print('barU', barU)
    print('barV', barV)
    print('barX', barX)
    print('barY', barY)
    print('sigmaU', sigmaU)
    print('sigmaV', sigmaV)
    print('sigmaX', sigmaX)
    print('sigmaY', sigmaY)
    print('r', r)
    print('rSquare', rSquare)
    print('sigmaBarX', sigmaBarX)
    print('sigmaBarY', sigmaBarY)
    print('NuyYtoX', NuyYtoX)
    print('NuyXtoY', NuyXtoY)
    print('sigmaBarX', sigmaBarX)
    print('sigmaBarY', sigmaBarY)
    print('XtoYRegressionA', XtoYRegressionA)
    print('YtoXRegressionB', YtoXRegressionB)
    print('B0', B0)
    print('B1', B1)
    print('t', t)
    print('sigmaYX', sigmaYX)
    print('bigSquareRoot', bigSquareRoot)
    print('bigSquareRoot2', bigSquareRoot2)
    print('B0Interval', B0Interval)
```

```
    print('B1Interval', B1Interval)
    print('Twatch', TWatch)
    print('TCrit', TCrit, TWatch < TCrit)

    def draw_correlation_field():
        plt.figure(dpi=200)
        plt.title('Корреляционное поле')

        x = np.linspace(0, 100, 2)

        y1 = (x - barX) * YtoXRegressionB + barY
        plt.plot(x, y1)

        y2 = (x - barX) / XtoYRegressionA + barY
        plt.plot(x, y2)

        plt.savefig("gr31.png")

    def print_tables():
        table = Texttable()
        table.set_cols_align(["l"] * (matrix.shape[1] + 2))
        table.add_row(['~', '~', *yAxis])
        table.add_row(['~', '~', *Vj])
        for i in range(Ui.size):
            table.add_row(
                [xAxis[i], Ui[i], *matrix[i]])
        print(draw_latex(table))

        table = Texttable()
        table.set_cols_align(["l"] * 5)
        table.add_row(['n_{i}', 'n_{i}u_{i}', 'sum', 'a', 'a'])
        for i in range(Ui.size):
            table.add_row(
                [Ni[i], NiUi[i], sumNijVj[i], NiUipow2[i], UiSumNijVj[i]]
            )
        print(draw_latex(table))

        table = Texttable()
        table.set_cols_align(["l"] * 5)
        table.add_row(['m_{i}', 'm_{i}v_{i}', 'sum', 'a', 'a'])
        for i in range(Vj.size):
            table.add_row(
                [Vj[i], MjVj[i], sumNijUi[i], MjVjpow2[i], ViSumNijUi[i]]
            )
        print(draw_latex(table))

        table = Texttable()
        table.set_cols_align(["l"] * (xAxis.size + 1))
        table.add_row(['$X$', *xAxis])
        table.add_row(['$\\bar{x_y}$', *barYs])
        print(draw_latex(table))

        table = Texttable()
        table.set_cols_align(["l"] * (yAxis.size + 1))
```

```
        table.add_row(['$Y$', *yAxis])
        table.add_row(['$\\bar{y_x}$', *barXs])
        print(draw_latex(table))

    print_tables()

def create_axis(start: float, step: float, count: int) -> np.ndarray:
    result = np.zeros(count)
    for i in range(count):
        result[i] = start + step * i
    return result

if __name__ == "__main__":
    task()
```