МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

институт информационных технологий и технологического образования
кафедра информационных технологий и электронного обучения

Основная профессиональная образовательная программа
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль) «Технологии разработки программного обеспечения»
форма обучения – очная

Курсовая работа

по дисциплине «Пакеты прикладных программ для статистической обработки и анализа данных»

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КОРРЕЛЯЦИОННОГО АНАЛИЗА ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Обучающегося 3 курса
Стецук Максима Николаевича
_____

Руководитель:
д.п.н, профессор
_____ Власова Е. З.

«_____» _____ 2024 г.

Санкт-Петербург
2024

Оглавление

Введение

Актуальность данной темы заключается в стремительном росте объема данных используемого в современном обществе. В наше время многие организации, исследователи и обычные люди нередко сталкиваются с необходимостью анализировать большие объемы информации и делать выводы исходя из неё.

Данная тенденция роста объема данных привела к повсеместному использованию анализа данных для решения различных задач, таких как выявление тенденций, прогнозирование, принятие решений на основе данных, персонализация, оптимизация процессов, проведение различных исследований и многих других.

Анализ данных – это процесс обработки, интерпретации и моделирования данных с целью выявления закономерностей, тенденций и важной информации, которую в дальнейшем применяют в различных целях. Он основан на применении различных математических методов, таких как линейная алгебра, теория вероятностей и математическая статистика, к данным, требующим обработки.

Существует большое количество методов проведения анализа больших данных, из которых наиболее популярные и распространенные – корреляционный анализ и анализ дискретных вариационных рядов. Эти методы могут быть реализованы в различных табличных процессорах и специализированных программах, но их все объединяет одна проблема, заключающаяся в сложности расширения объема данных, который используемая программа может принять и проанализировать, без изменения ранее заготовленного шаблона.

Предмет исследования – программа реализации алгоритмов для проведения корреляционного анализа и анализа вариационных рядов.

Целью данного исследования является программная реализация алгоритмов для проведения корреляционного анализа и анализа вариационных рядов, без привязки к некоторому, заранее задаваемому объему данных.

Задачи исследования:

- изучение специальной литературы по теме данного исследования;
- рассмотрение статистических методов используемых в корреляционном анализе и анализе дискретных вариационных рядов;
- программная реализация ранговой корреляции Спирмена и алгоритма получения данных для построения вариационных рядов;
- построения графиков корреляционного поля, полигона, кумулянты и эмпирической функции распределения с использованием специализированной библиотеки;
- проведение анализа больших данных на примере конкретной задачи с использованием разработанной программы.

При выполнении курсовой работы были использованы учебная и справочная литература, а также электронные ресурсы и ресурсы сети Интернет.

Глава 1. Теоретический блок

1.1 Корреляционный анализ

Корреляционный анализ – это статистический метод, который измеряет степень взаимосвязи между двумя переменными [3]. Он позволяет определить, насколько изменение одной переменной сопровождается изменением другой, а также выявить направление (положительное или отрицательное) и силу этой взаимосвязи.

Для реализации мной был выбран один из способов проведения корреляционного анализа – ранговая корреляция.

Ранговая корреляция – это метод измерения силы и направления взаимосвязи между двумя переменными, основанный на рангах значений этих переменных, а не на самих значениях.

Ранг значения – это позиция, которую занимает значение в упорядоченном ряду данных.

Для оценки возможного типа взаимосвязи между анализируемыми диапазонами применяется коэффициент ранговой корреляции Спирмена, который рассчитывается по формуле

$$r_s = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}, \quad (1)$$

Где:

$\sum_{i=1}^{n} d_i^2$ – сумма квадратов разностей рангов,

n – количество пар данных.

Для оценки статистической значимости коэффициента ранговой корреляции Спирмена и проверки наличия линейной связи между двумя выборками используется t–статистика, которая для коэффициента ранговой корреляции Спирмена рассчитывается по формуле:

$$t = |r_s| \sqrt{\frac{n-2}{1-r_s^2}}, \quad (2)$$

Где:

r_s – коэффициент ранговой корреляции Спирмена,

n – количество пар данных.

Для проверки нулевой гипотезы об отсутствии корреляции между переменными, полученную t–статистику сравнивают с критическим значением, взятым из таблицы критических значений с требуемым уровнем значимости и количеством степеней свободы равным n–2.

Уровень значимости представляет собой предопределенную вероятность ошибки первого рода в статистическом тесте. Он определяет, какую долю случаев, когда нулевая гипотеза верна, мы готовы считать статистически значимыми.

Обычно используются стандартные уровни значимости, такие как 0.05 или 0.01. Уровень значимости 0.05, означает, что мы готовы принять ошибку первого рода (отвержение верной нулевой гипотезы) в 5% случаев, если нулевая гипотеза верна.

1.2 Дискретный вариационный ряд

Дискретным вариационным рядом или статистическим распределением выборки называют таблицу частот, которая в первой строке содержит значения $x_1, x_2, \dots, x_n$, а во второй – числа их повторений $m_1, m_2, \dots, m_n$ [4].

Для проведения анализа дискретных вариационных рядов в данном исследовании применяются графические методы визуализации данных. Исследование основывается на использовании следующих графиков:

1. Полигона распределения, который позволяет визуально оценить форму распределения и выделить особенности изменений частот;
2. Кумулянты распределения, представляющей собой график, отражающий сумму частот от начала вариационного ряда до каждого интервала;
3. Эмпирической функции распределения, график которой строится на основе накопленных относительных частот и дает визуальное представление распределения частот по значениям и скорости их накопления.

Для построения полигона распределения используют таблицу абсолютных частот, для построения кумулянты распределения используют таблицу накопления частот, а для построения графика эмпирической функции распределения используют таблицу накопления относительных частот.

Относительная частота – процентное количество повторений от общего числа, относящееся к конкретному значению.

Глава 2. Практический блок

2.1 Описание алгоритма разработанной программы

В ходе выполнения курсовой работы мной была разработана программа на языке программирования Python [2], которая вычисляет данные для корреляционного анализа и строит корреляционное поле, а затем строит полигон, кумулянту и график эмпирической функции распределения для дискретного вариационного ряда по выборке из исходных данных.

После запуска, программа последовательно выполняет следующий программно реализованный алгоритм [1]:

1. Считывает и сохраняет данные для дальнейшего использования из двух представленных пользователем файлов в формате csv;
2. Суммирует значения в полученных из файлов таблицах по столбцам и сохраняет полученные суммы;
3. Находит среднее значение для каждого столбца;
4. Ранжирует средние значения для каждой таблицы;
5. Получает значения квадратов разностей для каждой пары соответствующих рангов из двух таблиц;
6. По формуле (1) находит коэффициент ранговой корреляции Спирмена;
7. Используя формулу (2) и значение коэффициента ранговой корреляции Спирмена находит Т–критерий;
8. Из заранее подготовленного файла загружает таблицу критических значений для 3 уровней значимости: 0.05, 0.01 и 0.001.
9. Исходя из выбранного пользователем уровня значимости и ранее определенного количества атрибутов, по которым ведется анализ, определяет количество степеней свободы, а затем по этим параметрам находит критическое значение Т.

10. Выбирает данные, по которым будет строиться корреляционное поле, а также данные необходимые для проведения корреляционного анализа и формирует сводную таблицу, которую сохраняет в соответствующую папку в формате csv.
11. Строит корреляционное поле и сохраняет изображение полученного графика в соответствующую папку для графиков.
12. Переходит к построению вариационных рядов. Получает от пользователя информацию об атрибутах, по которым необходимо сделать выборки, а затем по ним отбирает данные из ранее полученных таблиц с полными данными.
13. Получает частоты появления значений для значений выборок.
14. Строит полигоны распределения для 2 выборок данных и сохраняет изображения полученных графиков в соответствующую папку для графиков.
15. Получает накопления частот появления значений для 2 выборок.
16. Строит кумулянты распределения для 2 выборок данных и сохраняет изображения полученных графиков в соответствующую папку.
17. Получает накопления относительных частот появления значений для 2 выборок.
18. Строит графики эмпирической функции распределения для 2 выборок и сохраняет их изображения в соответствующую папку.

Данный алгоритм – это общая последовательность действий, схожая с последовательностью действий, которую необходимо выполнять самостоятельно, при проведении корреляционного анализа и построении дискретных вариационных рядов с использованием электронных таблиц или специализированного ПО.

Подробное описание реализации шагов указанного алгоритма, с опорой на код, рассматривается в параграфе 2.2.

2.2 Программная реализация алгоритма

Рассмотрим реализацию алгоритма, указанного в параграфе 2.1 подробнее, уточнив, что каждый этап алгоритма реализован в программе в виде отдельной функции. Для его реализации мной были использованы две вспомогательные библиотеки: csv – для генерации файла с таблицей, содержащий данные для построения корреляционного поля, а так же коэффициент ранговой корреляции Спирмена, значение T–статистики и T критическое, а так же matplotlib [5], для создания графиков по полученным значениям.

ПОДРОБНОЕ ОПИСАНИЕ АЛГОРИТМА

1. На данном этапе программа дважды вызывает функцию get_data, в которую при вызове передаётся имя файла. Она итерируется по строкам файла и возвращает список, содержащий списки с элементами каждой строки, код описанной функции представлен на рисунке 2.1.

```
2  # чтение данных из csv файла
3  def get_data(file_name):
4      data_list = []
5      with open(f"data_files/{file_name}.csv") as file:
6          array = [row.strip() for row in file]
7      for el in array:
8          data_list.append(el.split(';'))
9      return data_list
```

Рисунок 2.1. Функция get_data

2. Программа дважды вызывает функцию sum_value, которая вычисляет сумму по столбцам для каждой таблицы. При вызове, на вход она получает список полученный после выполнения первого этапа, затем создаёт список необходимой длины, заполненный нулями и итерируясь по спискам внутри переданного списка (кроме первой строки) прибавляет значения к соответствующим значениям в созданном списке. Пройдя весь список,

программа возвращает список сумм. Код указанной функции представлен на рисунке 2.2.

```
11  # суммирование значений
12  def sum_value(my_double_list):
13      sum_list = [0 for x in range(len(my_double_list[0]))]
14      for i in range(1, len(my_double_list)):
15          for j in range(len(my_double_list[0])):
16              sum_list[j] = float(sum_list[j]) + float(my_double_list[i][j])
17      return sum_list
```

Рисунок 2.2. Функция sum_value

3. Программа дважды вызывает функцию averge_value, в которую передаётся список сумм и количество элементов в столбце, без учета первой строки. Функция создаёт пустой список. Итерируясь по элементам переданного списка сумм, функция делит каждое число на переданное количество и записывает полученное значение в созданный список. После выхода из цикла функция возвращает список средних значений, её код представлен на рисунке 2.3.

```
19  # нахождение средних
20  def averge_value(my_list, n):
21      av_list = []
22      for el in my_list:
23          av_list.append(round((el / n), 3))
24      return av_list
```

Рисунок 2.3. Функция averge_value

4. На данном этапе программа дважды вызывает функцию ranks, в которую передаются список средних значений и список атрибутов (первая строка исходного списка строк таблицы), с помощью создания двух вспомогательных словарей, а именно словарей вида {атрибут : среднее значение} и {значение : ранг}, и итерации по списку атрибутов, программа заполняет список вида {атрибут : ранг}, а затем возвращает его, тем самым проведя ранжирование средних значений для столбцов исходной таблицы данных. Реализация данной функции представлена на рисунке 2.4.

```python
#ранжирование
def ranks(avg_val, keys):
    my_dict_el_val = {}
    my_dict_el_val_rank = {}
    my_dict_el_rank = {}

    i = 0
    for key in keys:
        my_dict_el_val[key]=avg_val[i]
        i+=1

    sorted_avg_val = sorted(avg_val)
    i = 1
    for val in sorted_avg_val:
        my_dict_el_val_rank[val] = i
        i+=1

    for key in keys:
        value = my_dict_el_val[key]
        my_dict_el_rank[key] = my_dict_el_val_rank[value]

    return my_dict_el_rank
```

Рисунок 2.4. Функция ranks

5. Сначала программа дважды вызывает функцию get_X_Y, которой по очереди передаются словари, полученные на 4 этапе и список атрибутов. Она создаёт пустой список и итерируясь по списку атрибутов, получает ранги соответствующие каждому атрибуту и записывает их в список. Таким образом, после вызова данной функции, которая представлена на рисунке 2.5, программа получает 2 списка рангов Xi и Yi.

```python
# получение массивов для корреляционного поля
def get_X_Y(dict, keys):
    my_list = []
    for key in keys:
        my_list.append(dict[key])
    return my_list
```

Рисунок 2.5. Функция get_X_Y

На этом же этапе программа вызывает функцию get_mass_d2, которая на вход получает массивы значений Xi и Yi. Создаёт пустой массивы, а затем с помощью цикла для каждой пары значений xi и yi из переданных списков находит их разность и записывает её квадрат в созданный список, который после обхода всех пар она возвращает. Данные действия выполняются с помощью кода представленного на рисунке 2.6.

```
56  # квадраты разностей между соответствующими значениями
57  def get_mass_d2(mass_x, mass_y):
58      d2_list = []
59      for i in range(len(mass_x)):
60          d = mass_x[i] - mass_y[i]
61          d2_list.append(d**2)
62      return d2_list
```

Рисунок 2.6. Функция get_mass_d2

6. Теперь программа вызывает функцию rank_cor_Spir, которая получает на вход список значений квадратов разностей рангов атрибутов начальных списков. Используя формулу (1) она рассчитывает и возвращает коэффициент ранговой корреляции Спирмена. Реализация функции для подсчета коэффициента представлена на рисунке 2.7.

```
64  # ранговая корреляция Спирмена
65  def rank_cor_Spir(deltas_list):
66      l = len(deltas_list)
67      sum_d2 = 0
68      for el in deltas_list:
69          sum_d2 += el
70      Rs = 1 - (6 * sum_d2) / (l * (l**2 -1))
71      return Rs
```

Рисунок 2.7. Функция rank_cor_Spir

7. Получив на этапе 6, значение коэффициента ранговой корреляции Спирмена, программа вызывает функцию T_kof, которая получает на вход значение коэффициента корреляции и количество атрибутов (количество элементов в первых строках исходных таблиц). А затем по формуле (2) функция рассчитывает значение T–критерия, которое и возвращает. Ее реализация представлена на рисунке 2.8.

```
73  # T-критерий
74  def T_kof(Rs, n):
75      from math import sqrt
76      t = abs(Rs)*sqrt((n-2)/(1-Rs*Rs))
77      return t
```

Рисунок 2.8. Функция T_kof

8. Программа вызывает функцию get_T_crit_values, которая итерируясь по строкам файла с критическими значениями, записывает их в виде списков элементов в заранее созданный пустой список, а затем возвращает заполненный список. Программная реализация функции представлена на рисунке 2.9.

```
79  # загрузка критических значений из файла
80  def get_T_crit_values():
81      data_list = []
82      with open("data_files/T_crit_values.csv") as file:
83          array = [row.strip() for row in file]
84      for el in array:
85          data_list.append(el.split(';'))
86      return data_list
```

Рисунок 2.9. Функция get_T_crit_values

9. Теперь программа вызывает функцию find_crit, получающую на вход количество атрибутов, выбранный уровень значимости и список критических значений. Программа определяет, в каком столбце необходимо искать критическое значение (каждому столбцу соответствует своё критическое значение), а затем находит строку таблицы с требуемым количеством степеней свободы (n–2) и возвращает значение на пересечении требуемого столбца и данной строки. Код функции представлен на рисунке 2.10.

```
88  # получение критического значения
89  def find_crit(n, alpha, crit_list = get_T_crit_values()):
90      if alpha == 0.05:
91          var = 1
92      elif alpha == 0.01:
93          var = 2
94      elif alpha == 0.001:
95          var = 3
96      else:
97          print("Выбран недоступный уровень значимости!")
98      for i in range(1, len(crit_list)):
99          if int(crit_list[i][0]) == n - 2:
100             return crit_list[i][var]
```

Рисунок 2.10. Функция find_crit

10. На данном этапе программа вызывает функцию create_csv_table, на вход которой передаются такие параметры, как атрибуты, названия первой и

второй исходных выборок, списки значений Xi и Yi, список значений квадратов разницы xi – yi, количество атрибутов, коэффициент ранговой корреляции Спирмена, значение T–критерия для коэффициента ранговой корреляции и T–критическое. С помощью библиотеки csv функция построчно записывает эти данные в специальный файл с расширением csv. Реализация функции представлена на рисунке 2.11.

```
102  # csv таблица с данными
103  def create_csv_table(KEYS, name_1, name_2, X, Y, D2, n, rs, t, t_crit):
104      import csv
105      with open('cor_field/cor_field_data.csv', 'w', newline='') as csvfile:
106          spamwriter = csv.writer(csvfile, delimiter=',',
107                                  quotechar=',', quoting=csv.QUOTE_MINIMAL)
108
109          spamwriter.writerow(['Измерения'] + KEYS)
110          spamwriter.writerow([f'Ранг {name_1}'] + X)
111          spamwriter.writerow([f'Ранг {name_2}'] + Y)
112          spamwriter.writerow(['d^2'] + D2)
113          spamwriter.writerow(['n', n])
114          spamwriter.writerow(['rs', str(round(float(rs), 3))])
115          spamwriter.writerow(['t', str(round(float(t), 3))])
116          spamwriter.writerow(['t_crit', str(round(float(t_crit), 3))])
```

Рисунок 2.11. Функция create_csv_table

11. Программа вызывает функцию correlation_field, которая получает на вход списки Xi и Yi (списки рангов), а затем с помощью функционала библиотеки matplotlib строит и сохраняет в виде изображения точечную диаграмму, используя в качестве x и y ранги атрибутов двух исходных наборов данных. Код функции correlation_field представлен на рисунке 2.12.

```
120  # корреляционное поле
121  def correlation_field(x_mass, y_mass):
122      import matplotlib.pyplot as plt
123      import numpy as np
124      plt.figure("Корреляция")
125      plt.title('Корреляционное поле')
126      plt.xlabel("IT")
127      plt.ylabel("Мед")
128      plt.grid( axis = 'y')
129      #plt.grid(True)
130      x = np.array(x_mass)
131      y = np.array(y_mass)
132      plt.scatter(x, y)
133      z = np.polyfit(x, y, 1)
134      p = np.poly1d(z)
135      plt.plot(x, p(x), color = 'r')
136      #plt.show()
137      plt.savefig('graphics/cor_field.png')
```

Рисунок 2.12. Функция correlation_field

12. На данном шаге программа получает от пользователя список атрибутов, по которым необходимо сделать выборку результатов для построения графиков для вариационных рядов.

13. Программа дважды вызывает функцию get_Xi_mi, которая получает на вход исходные данные и список атрибутов для выборки. С помощью итераторов программа подсчитывает количество повторений в выборке каждого значения и возвращает словарь вида {значение : частота}. Реализация данной функции представлена на рисунке 2.13.

```
139  # получение данных для полигона распределение
140  def get_Xi_mi(data_list, selected):
141      selection = []
142      for el in selected:
143          for i in range(len(data_list[0])):
144              if data_list[0][i] == el:
145                  selection.append(i+1)
146      vals = [int(i) for i in range(1,len(data_list[0])+1)]
147      counter = {}
148      for el in vals:
149          counter[el] = 0
150      for i in range(1, len(data_list)):
151          for el in selection:
152              counter[int(data_list[i][el-1])]+=1
153      return counter
```

Рисунок 2.13. Функция get_Xi_mi

14. Получив необходимые словари на этапе 13, программа дважды вызывает функцию distribution_polygon, которой на вход передаётся словарь частот появления для выборки из начального набора данных и название этого набора. Функция преобразует, полученные данные и с помощью matplotlib строит полигоны распределения для двух выборок и сохраняет их графики в виде изображений. Код данной функции представлен на рисунке 2.14.

```
155  # полигон распределения
156  def distribution_polygon(my_data_dict:dict, name):
157      import matplotlib.pyplot as plt
158      import numpy as np
159      Xi = list(my_data_dict.keys())
160      Mi = []
161      for el in Xi:
162          Mi.append(my_data_dict[el])
163
164      plt.figure()
165      plt.title(f'Полигон распределения {name}')
166      plt.xlabel("Xi")
167      plt.ylabel("mi")
168      plt.grid(True)
169      x = np.array(Xi)
170      y = np.array(Mi)
171      plt.plot(x, y, 'o-')
172      #plt.show()
173      plt.savefig(f'graphics/poligon_{name}.png')
```

Рисунок 2.14. Функция distribution_polygon

15. Программа дважды вызывает функцию get_Xi_mxi, которая получает на словарь с информацией о частоте появления значений, создаёт пустой словарь, а затем с помощью цикла, в него добавляются значения в качестве ключей, хранящих информацию о накоплении частоты. Она возвращает словарь вида {значение : накопление частоты} Данная функция представлена на рисунке 2.15.

```
175  # получение данных для кумулянты
176  def get_Xi_mxi(dict_xi:dict):
177      n = len(list(dict_xi.keys()))
178      keys = [i for i in range(1, n+2)]
179      value = 0
180      dict_Xi_mxi = {}
181      dict_Xi_mxi[keys[0]] = value
182      for key in keys:
183          if key != keys[0]:
184              value += dict_xi[key-1]
185              dict_Xi_mxi[key] = value
186      return dict_Xi_mxi
```

Рисунок 2.15. Функция get_Xi_mxi

16. Дважды вызывается функция cumulant, которая аналогично функции distribution_polygon преобразует полученный словарь и с помощью matplotlib строит кумулянты распределения для двух выборок и сохраняет их графики в виде изображений. Код функции представлен на рисунке 2.16.

```
188  # кумулянта
189  def cumulant(my_data_dict:dict, name):
190      import matplotlib.pyplot as plt
191      import numpy as np
192      Xi = list(my_data_dict.keys())
193      Mxi = []
194      for el in Xi:
195          Mxi.append(my_data_dict[el])
196
197      plt.figure()
198      plt.title(f'Кумулянта {name}')
199      plt.xlabel("Xi")
200      plt.ylabel("Mxi")
201      plt.grid(True)
202      x = np.array(Xi)
203      y = np.array(Mxi)
204      plt.plot(x, y, 'o-')
205      #plt.show()
206      plt.savefig(f'graphics/cumulant_{name}.png')
```

Рисунок 2.16. Функция cumulant

17. Программа дважды вызывает функцию get_Xi_Wxi, которая получает на словарь с информацией о накоплении частот появления значений, с помощью циклов обходит его и получает списки значений и

накопления относительных частот. А затем возвращает два полученных списка. Реализация данной функции представлена на рисунке 2.17.

```
208  # получение данных для эмпирической функции распределения
209  def get_Xi_Wxi(dict_xi_mxi:dict):
210      x_keys = list(dict_xi_mxi.keys())
211      values = []
212      for key in x_keys:
213          values.append(dict_xi_mxi[key])
214      x_keys.pop(-1)
215      sum = values[len(values)-1]
216      values.pop(0)
217      wxi = []
218      for val in values:
219          wxi.append(round(val / sum, 3))
220      return x_keys, wxi
```

Рисунок 2.17. Функция get_Xi_Wxi

18. Дважды вызывает функцию empirical_distribution, в которую передаются списки полученные на этапе 17. С помощью matplotlib функция строит графики эмпирических функций распределения для двух выборок и сохраняет их в виде изображений. Код функции представлен на рисунке 2.18.

```
222  # эмпирическая функция распределения
223  def empirical_distribution(list_Xi, list_Wxi, name):
224      import matplotlib.pyplot as plt
225      import numpy as np
226      pairs_Xi_list = []
227      pairs_Wxi_list = []
228      for el in list_Xi:
229          pairs_Xi_list.append([el, el+1])
230      for el in list_Wxi:
231          pairs_Wxi_list.append([el, el])
232
233      plt.figure()
234      plt.title(f'Эмпирическая функция распределения ({name})')
235      plt.xlabel("Xi")
236      plt.ylabel("Wxi")
237      plt.grid(True)
238      for i in range(len(pairs_Xi_list)):
239          x = np.array(pairs_Xi_list[i])
240          y = np.array(pairs_Wxi_list[i])
241          plt.plot(x, y, 'blue')
242      plt.savefig(f'graphics/emp_dist_{name}.png')
```

Рисунок 2.18. Функция empirical_distribution

2.3 Анализ больших данных с помощью разработанной программы

Полный код программы, алгоритм которой был разобран в параграфах 2.1 и 2.2, начальные таблицы данных и полученные во время решения задачи графики расположены в Git репозитории, ссылка на него представлена в Приложении 1.

УСЛОВИЕ ЗАДАЧИ

Был проведён анонимный опрос среди студентов двух групп (по 50 человек в каждой), обучающихся в различных сферах подготовки. Первая группа студентов обучается в колледже в сфере информационных технологий, а вторая в колледже в медицинской сфере. В ходе опроса каждому студенту предоставлялся список из 10 наиболее развивающихся и известных ИИ, предназначенных для различного рода задач. Им необходимо было расположить представленные ИИ в порядке от наиболее полезного/известного для конкретного студента, до наименее известного (соответственно: значение 1 – очень полезный/известный для конкретного студента, например он использует данный ИИ с некоторой периодичностью, значение 10 – наименее известный, имеет минимальный опыт использования, никогда не использовал или даже не слышал о нём). Частичные результаты опросов для двух групп представлены в таблицах на рисунках 3.1 и 3.2:

ChatGPT	CodeGPT	BLOOM	Whisper	DALL-E 2	Craiyon	Stable Diffusion	Imagen	Make-A-Video	GitHub Copilot
1	3	4	8	7	5	6	10	9	2
1	2	10	5	4	6	7	8	9	3
3	2	9	8	7	4	10	5	6	1
2	3	9	7	10	8	6	1	4	5
2	1	4	10	5	6	9	7	8	3
1	3	6	5	9	8	4	10	7	2
2	4	8	7	1	6	3	9	10	5
3	4	6	8	10	9	7	1	2	5
4	3	8	5	9	1	2	10	7	6
1	4	3	2	9	7	8	6	10	5
5	10	7	6	1	4	9	2	3	8

Рисунок 3.1. Результаты опроса для IT группы

ChatGPT	CodeGPT	BLOOM	Whisper	DALL-E 2	Craiyon	Stable Diffusion	Imagen	Make-A-Video	GitHub Copilot
9	6	3	5	10	4	7	1	2	8
3	4	9	7	2	10	6	5	8	1
7	10	8	4	1	6	5	9	2	3
5	4	8	9	6	7	1	3	10	2
5	10	3	9	1	7	4	6	8	2
9	8	10	2	7	1	4	6	3	5
3	2	5	9	1	6	7	8	4	10
3	1	2	8	7	10	5	6	9	4
1	6	2	8	7	10	5	3	9	4
1	6	10	4	7	8	2	5	3	9
4	7	1	3	5	10	2	9	6	8

Рисунок 3.2. Результаты опроса для медицинской группы

Необходимо сравнить совокупности ответов студентов каждой из групп, а также построить и изобразить графически (полигон, кумулянта и эмпирическая функция распределения) дискретный вариационный ряд для каждой группы студентов по выборке оценок для 3 ИИ: ChatGPT, CodeGPT, GitHub Copilot. Проанализировать полученный результат, сделать выводы о востребованности прикладных ИИ в двух различных сферах подготовки студентов.

РЕШЕНИЕ ЗАДАЧИ

Файлы с таблицами результатов переименованы в data_1 и data_2 для группы IT студентов и студентов медиков соответственно, а затем переданы на вход программе.

После запуска, программа просит указать названия выборок, ввожу соответственно IT и Med. После ввода она просит выбрать уровень значимости для нахождения критического значения T. Выбираю уровень значимости 0.05. На рисунке 3.3 представлен консольный вывод программы.

```
Введите название первой выборки: IT
Введите название второй выборки: Med
Коэффициент ранговой корреляции Пирсона: 0.115
T-критерий: 0.328
Выберите уровень значимости (0.05, 0.01 или 0.001): 0.05
T критическое: 2.306
```

Рисунок 3.3. Консольный вывод для корреляционного анализа

Программа нашла средние значения для столбцов каждой таблицы, провела их ранжирование, по этим данным построила корреляционное поле, которое представлено на рисунке 3.4. Затем определила разности рангов и их квадраты. Используя формулу (1) нашла коэффициент ранговой корреляции Спирмена, после чего используя T–статистику, нашла T–критерий для двух начальных таблиц значений. И по таблице критических значений нашла T–критическое для данной задачи.

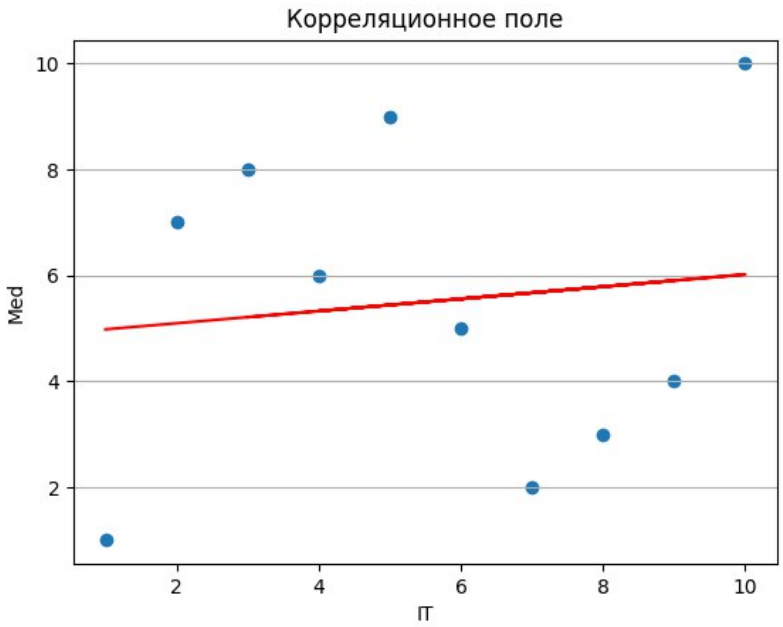


Рисунок 3.4. Корреляционное поле

Помимо консольного вывода, программа сформировала таблицу с полученными значениями, которая представлена на рисунке 3.5.

cor_field_data

Измерения	ChatGPT	CodeGPT	BLOOM	Whisper	DALL-E 2	Craiyon	Stable Diffusion	Imagen	Make-A-Video	GitHub Copilot
Ранг IT	1	2	8	6	9	5	4	7	10	3
Ранг Med	1	7	3	5	4	9	6	2	10	8
d^2	0	25	25	1	25	16	4	25	0	25
n	10									
rs	0.115									
t	0.328									
t_crit	2.306									

Рисунок 3.5. Сводная таблица значений

Из полученного вывода программы и сводной таблицы видим, что T = 0.328, а T–критическое = 2.306, получается, что $t_{\text{расч}} < t_{\text{кр}}$, а значит, нулевая гипотеза не отвергается. И поэтому можно сказать, что между двумя результатами опросов отсутствует статистически значимая связь.

Таким образом, можно сделать вывод, что мнения студентов двух групп (в совокупности) про 10 представленных в опросе ИИ сильно различаются, и данное различие может быть вызвано непосредственно направлением их подготовки и различием потребности в определённых инструментах у каждого студента. Однако выбор, сделанный одной группой студентов, никаким образом не влияет на выбор другой группы студентов.

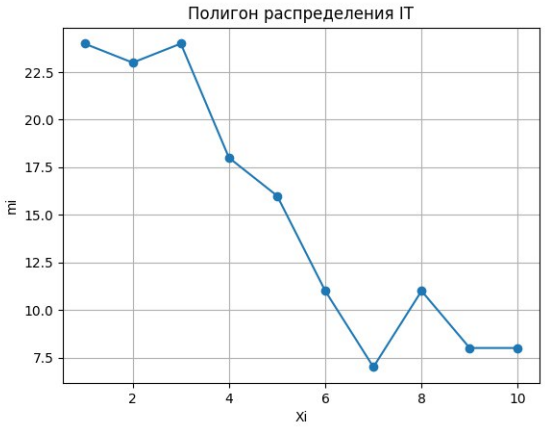
Теперь проанализируем потребность студентов двух групп в прикладных ИИ, таких как ChatGPT, CodeGPT и GitHub Copilot.

После построения корреляционного поля и создания сводной таблицы, программа просит ввести атрибуты, по которым будут сформированы выборки результатов. Поэтому ввожу 3 атрибута: ChatGPT, CodeGPT и GitHub Copilot, консольный вывод для данного действия представлен на рисунке 3.6.

```
Выберите по каким атрибутам будем строить вариационные ряды (введите через запятую)
 ['ChatGPT', 'CodeGPT', 'BLOOM', 'Whisper', 'DALL-E 2', 'Craiyon', 'Stable Diffusion
Атрибуты: ChatGPT,CodeGPT,GitHub Copilot
['ChatGPT', 'CodeGPT', 'GitHub Copilot']
Graphics saved to /graphics!
```

Рисунок 3.6. Ввод атрибутов для выборок

После выбора атрибутов, программа подсчитывает необходимые значения и строит графики полигона, кумулянты и эмпирической функции распределения. На рисунке 3.7 представлены полигоны распределения для обеих выборок, на рисунке 3.8 представлены кумулянты, а на рисунке 3.9 представлены графики эмпирической функции распределения.



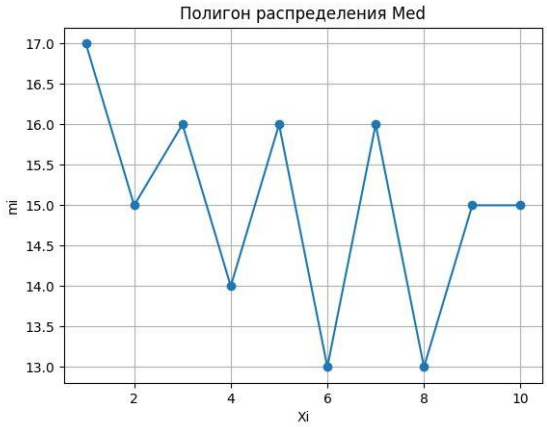
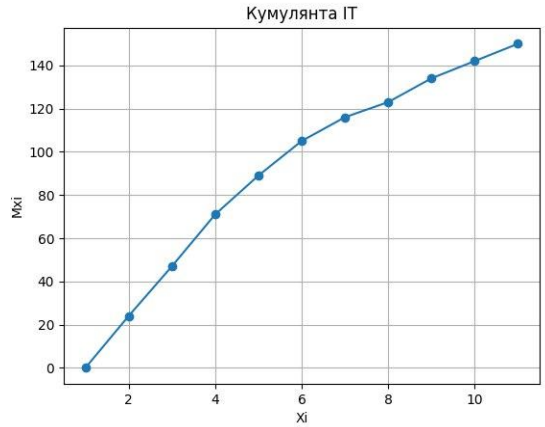


Рисунок 3.7. Полигоны



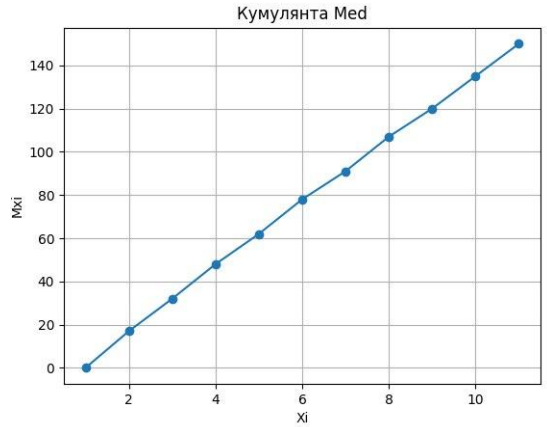
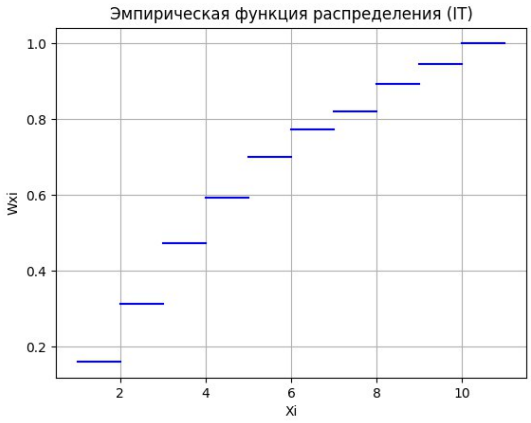


Рисунок 3.8. Кумулянты



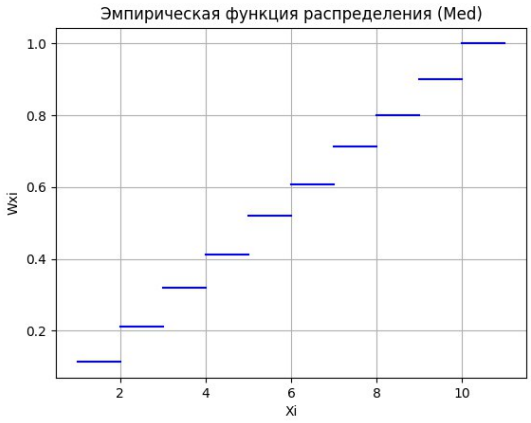


Рисунок 3.9. Эмпирические функции распределения

Исходя из полученных полигонов распределений, кумулянт и эмпирических функций распределения видно, что у студентов, обучающихся по направлению информационных технологий, ChatGPT, CodeGPT и GitHub Copilot занимают лидирующую позицию среди остальных ИИ по известности/полезности, т.к. большая часть участников опроса указали оценку (ранг) 1, 2 или 3, ведь именно эти ИИ имеют большие возможности и спектр применений при программировании и т.п. В то же время у студентов, обучающихся на направлении связанном с медициной, оценки (ранги) сильно варьируются от 1 до 10.

Заключение

При написании курсовой работы по представленной теме была изучена специальная литература, включавшая справочные материалы по корреляционному анализу, анализу и построению вариационных рядов, а также учебные пособия и ресурсы сети Интернет посвящённые программированию на языке Python и построению графиков с помощью библиотеки matplotlib.

В теоретической части курсовой работы раскрыты основные определения и понятия, необходимые для понимания принципа проведения корреляционного анализа и построения вариационных рядов. Указаны основные использованные формулы и пояснения к ним. В практической части работы представлен алгоритм, разработанный на основе изученных материалов. Представлена программная реализация данного алгоритма. А так же демонстрируется решение задачи на анализ больших данных с помощью разработанной программы. Стоит отметить, что с помощью разработанной программы можно проводить корреляционный анализ опросов, в которых опрашиваемых просят отранжировать атрибуты по некоторому признаку. А так же строить вариационные ряды по выборке атрибутов, на усмотрение пользователя. Программа является универсальной, благодаря тому, что не требует особых действий для расширения анализируемого диапазона. Достаточно загрузить в нее таблицы в формате csv с новыми результатами опросов (с иным количеством ранжируемых атрибутов или иным количеством опрашиваемых) и она сама определит размеры выборки данных и количество атрибутов, а затем рассчитает искомые значения и построит требуемые графики.

Таким образом, было проведено исследование по теме курсовой работы, а также был изучен материал, необходимый для проведения

разработки и анализа по заданной теме. Была разработана программа, примененная при проведении анализа больших данных.

Список литературы

1. Алгоритмы. Построение и анализ / К. Томас, Л. Чарльз, Р. Рональд, Ш. Клиффорд.: 2015. – 1328 с.

2. Марк Лутц Изучаем Python. Том 1 / Лутц Марк. – 5–е изд. – М.: Вильямс, 2019. – 832 с.

3. Лапач, С. Н. Статистика в науке и бизнесе / С. Н. Лапач, А. В. Чубенко, П. Н. Бабич. – К. : МОРИОН, 2002. – 176 с.

4. Usmonov M. Дискретный вариационный ряд. Полигон частот и эмпирическая функция распределения / Scienceweb academic papers collection – 2022.

5. Абдрахманов М. И. Devpractice Team. Библиотека Matplotlib / М. И. Абдрахманов: devpractice.ru, 2019 – 100 с.

Приложение 1

Ссылка на Git репозиторий с полным листингом кода программы и исходными данными, использованными при проведении анализа больших данных: https://clck.ru/36xMDW

QR код с ссылкой на Git репозиторий: