Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ

Н. Г. ЧЕРНЫШЕВСКОГО»

УТВЕРЖДАЮ

Зав.кафедрой,

к. ф.-м. н.

\_\_\_\_\_\_\_\_\_\_ М. В. Огнева

**ОТЧЕТ О ПРАКТИКЕ**

Студентки 2 курса 241 группы факультета КНиИТ

Хрустицкой Софьи Кирилловны

вид практики: производственно-технологическая

кафедра: информатики и программирования

курс: 1

семестр: 2

продолжительность: 2 нед., с 24.06.2024 г. по 07.07.2024 г.

Руководитель практики от университета,

ст. пр.                                   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_                          А. А. Казачкова

ст. пр.                                   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_                          Е. Е. Лапшева

Тема практики: «Библиотеки предобработки данных»

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc176017673)

[1. Основные сведения о библиотеках Python для обработки данных 7](#_Toc176017674)

[1.1 Библиотека Random 7](#_Toc176017675)

[1.2 Библиотека Math 8](#_Toc176017676)

[1.3 Библиотека NumPy 9](#_Toc176017677)

[1.4 Библиотека Pandas 11](#_Toc176017678)

[1.5 Библиотека Matplotlib 12](#_Toc176017679)

[1.6 Библиотека Plotly 13](#_Toc176017680)

[2 Решение задач с использованием изученных библиотек 15](#_Toc176017681)

[2.1 Встроенные модули 15](#_Toc176017682)

[2.1.1 Модуль Random 15](#_Toc176017683)

[2.1.2 Измерение времени выполнения 19](#_Toc176017684)

[2.2 Модуль NumPy 24](#_Toc176017685)

[2.2.1 NumPy знакомство 25](#_Toc176017686)

[2.2.2 NumPy обработка матрицы 27](#_Toc176017687)

[2.2.3 NumPy вектор и матрицы 28](#_Toc176017688)

[2.2.4 NumPy задачка 29](#_Toc176017689)

[2.3 Модуль Pandas 32](#_Toc176017690)

[2.3.1 Модуль Pandas 33](#_Toc176017691)

[2.4 Визуализация 37](#_Toc176017692)

[2.4.1 Перевозки (статистика) 37](#_Toc176017693)

[2.5 Итоговое задание 42](#_Toc176017694)

[2.5.1 Анализ результатов экзаменов по информатике 42](#_Toc176017695)

[ЗАКЛЮЧЕНИЕ 67](#_Toc176017696)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 68](#_Toc176017697)

[ПРИЛОЖЕНИЕ А Модуль Random 69](#_Toc176017698)

[ПРИЛОЖЕНИЕ Б Измерение времени выполнения 72](#_Toc176017699)

[ПРИЛОЖЕНИЕ В NumPy знакомство 81](#_Toc176017700)

[ПРИЛОЖЕНИЕ Г NumPy Обработка матрицы 83](#_Toc176017701)

[ПРИЛОЖЕНИЕ Д NumPy вектор и матрица 84](#_Toc176017702)

[ПРИЛОЖЕНИЕ Ё NumPy загадка 85](#_Toc176017703)

[ПРИЛОЖЕНИЕ Ж NumPy сравнение эффективности 87](#_Toc176017704)

[ПРИЛОЖЕНИЕ И Библиотека Pandas 89](#_Toc176017705)

[ПРИЛОЖЕНИЕ К Визуализация 92](#_Toc176017706)

[ПРИЛОЖЕНИЕ Л Итоговое задание 96](#_Toc176017707)

# **ВВЕДЕНИЕ**

В современном мире, предобработка данных играет ключевую роль в различных областях науки, технологий и бизнеса. С быстрым ростом объема данных, доступных из различных источников, таких как социальные сети, транзакционные системы и многое другое, необходимость в эффективной предобработке данных становится все более очевидной.

Важность предобработки данных подтверждается различными исследованиями. Например, одно из исследований показывает, что около 80% времени, затрачиваемого на проекты анализа данных, уходит на этапы сбора, очистки и подготовки данных. Это подчеркивает критическую важность эффективного управления данными для достижения успешных результатов в аналитике. [1]

**Актуальность** инструментов, изучаемых на практике, заключается в том, что современные специалисты в анализе данных сталкиваются с огромными объемами информации, которые необходимо эффективно обрабатывать и интерпретировать. Библиотеки предоставляют мощные возможности для манипуляции, анализа и визуализации данных. Без этих инструментов специалист было бы гораздо сложнее выполнять задачи по подготовке данных, выявлению закономерностей и представлению результатов анализа в наглядной форме.

Основной целью практики является ознакомление с возможностями обработки данных с использованием языка Python и его стандартных модулей и библиотек Random, Math, NumPy, Pandas и библиотек визуализации.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить возможности и инструменты программной среды Google Colab;
2. Изучить стандартные модули Math и Random языка Python;

3. Изучить объекты и методы библиотек NumPy, Pandas и библиотек визуализации;

4. Применить полученные знания для решения поставленных задач, а также для самостоятельной обработки и анализа набора данных из реальной жизни.

# **1. Основные сведения о библиотеках Python для обработки данных**

Python является одним из самых популярных языков программирования в области анализа данных и научных вычислений благодаря своей простоте, читаемости и обширной экосистеме библиотек.

Среди множества библиотек, доступных в Python, особенно выделяются те, которые предназначены для работы с данными и их визуализации. Эти библиотеки значительно облегчают выполнение сложных вычислений, обработку больших объемов данных и создание наглядных визуализаций.

Кроме того, среда Google Colab представляет собой облачную платформу для выполнения Python-скриптов. Colab предоставляет доступ к вычислительным ресурсам Google, что позволяет выполнять сложные вычислительные задачи без необходимости устанавливать локальные инструменты, что делает Colab идеальным инструментом для обучения и экспериментов в области анализа данных. Google Colab поддерживает все популярные библиотеки Python, такие как NumPy и Pandas, что делает его мощной и удобной платформой для работы с данными.

Далее мы рассмотрим основные библиотеки Python, среди которых есть те, что используются для предобработки данных: Random, Math, Numpy и Pandas, а также библиотеки для визуализации данных: Matplotlib и Plotly.

## **1.1 Библиотека Random**

Библиотека Random предоставляет функции для генерации случайных чисел, выборки элементов из последовательностей и выполнения других задач, связанных со случайными данными. Эти функции широко используются в различных областях, таких как игры, моделирование, тестирование и безопасность.

Часто используемые методы библиотеки Random:

1. random.random()

Метод возвращает случайное число с плавающей точкой в диапазоне [0.0, 1.0). Этот метод часто используется для генерации случайных дробных чисел, которые можно масштабировать к нужному диапазону.

2. random.randint(a, b)

Метод возвращает случайное целое число N такое, что a <= N <= b. Этот метод часто используется для генерации случайных целых чисел в заданном диапазоне.

3. random.choice(seq)

Метод возвращает случайный элемент из непустой последовательности seq. Этот метод удобен для выбора случайного элемента из списка, кортежа или строки.

4. random.shuffle(seq)

Метод перемешивает последовательность seq на месте. Этот метод часто используется для случайного перемешивания элементов списка.

5. random.sample(seq, k)

Метод возвращает новый список длины k, состоящий из уникальных элементов, случайным образом выбранных из последовательности seq. Этот метод полезен для случайного выбора подмножества элементов. [2]

Эти методы популярны благодаря своей простоте использования, широкому спектру задач и гибкости. Они подходят для различных задач, от простого выбора случайных элементов до сложного моделирования и тестирования, что позволяет применять их в самых разнообразных областях. Кроме того, методы предоставляют возможности для генерации как целых, так и дробных чисел, а также для работы с последовательностями различных типов, что делает их очень гибкими и универсальными в использовании.

## **1.2 Библиотека Math**

Библиотека Math в Python является стандартной библиотекой для выполнения различных математических операций. Она предоставляет функции для работы с числами, включая основные арифметические операции, вычисления тригонометрических функций, логарифмов, экспонент и многое другое. Эта библиотека широко используется в различных приложениях, от простых вычислений до сложных научных исследований.

Примеры популярных методов:

1. math.sqrt(x)

Метод возвращает квадратный корень из числа x. Этот метод часто используется в инженерных и научных вычислениях, где необходимо найти квадратный корень числа.

2. math.pow(x, y)

Другой важный метод, который возвращает значение x в степени y. Этот метод используется для возведения числа в степень, что необходимо в различных математических задачах.

3. math.sin(x)

Метод, возвращающий синус угла x, где x задан в радианах. Он широко используется в тригонометрических вычислениях и при работе с периодическими функциями.

4. math.log(x, base)

Метод возвращает логарифм числа x по основанию base. Если основание не указано, используется натуральный логарифм.

5. math.factorial(x)

Метод возвращает факториал числа x. Этот метод используется в комбинаторике, теории вероятностей и других областях математики. [3]

Эти методы популярны из-за их эффективности и простоты использования. Они предоставляют надежные и точные результаты, что делает их незаменимыми в различных приложениях, включая научные исследования, инженерные расчеты, экономический анализ и разработку алгоритмов.

## **1.3 Библиотека NumPy**

Библиотека NumPy (Numerical Python) является одной из самых популярных и широко используемых библиотек для научных вычислений в Python. Она предоставляет поддержку для многомерных массивов и матриц, а также содержит множество высокоуровневых математических функций. NumPy является основой для многих других библиотек и инструментов в экосистеме Python для анализа данных и машинного обучения.

Одной из ключевых особенностей NumPy является его способность обрабатывать большие массивы данных и выполнять векторизованные операции, что значительно повышает производительность по сравнению с традиционными методами, основанными на циклах. Это делает NumPy незаменимым инструментом для научных исследований, анализа данных и машинного обучения.

Примеры часто используемых методов:

1. numpy.array()

Метод создает массив NumPy из списка или кортежа.

2. numpy.arange()

Метод генерирует массив, содержащий числа в заданном диапазоне с определенным шагом. Часто используется для создания числовых последовательностей, необходимых для различных вычислений и графиков.

3. numpy.zeros()

Метод создает массив заданной формы, заполненный нулями. Используется для инициализации массивов и матриц перед выполнением вычислений.

4. numpy.ones(shape)

Метод создает массив заданной формы, заполненный единицами. Используется для инициализации массивов и матриц, где требуется начальное значение единицы.

5. numpy.linspace(start, stop, num)

Метод создает массив из num равномерно разнесенных значений между start и stop. Полезен для создания массивов значений, используемых в численных методах и графиках. [4]

Эти методы популярны благодаря их простоте и эффективности в создании и манипулировании массивами. Они предоставляют удобные средства для выполнения сложных математических операций, анализа данных и построения моделей. Векторизованные операции в NumPy позволяют значительно ускорить вычисления по сравнению с традиционными методами, что делает эту библиотеку незаменимой для ученых и инженеров.

## **1.4 Библиотека Pandas**

Библиотека Pandas является одной из самых популярных и мощных библиотек для анализа и манипуляции данными в Python. Она предоставляет высокоуровневые структуры данных и функции, предназначенные для работы с табличными данными, временными рядами и другими видами данных. Pandas позволяет эффективно и удобно выполнять операции по фильтрации, агрегации, объединению и очистке данных.

Примеры популярных методов:

1. pd.DataFrame()

Метод является одним из часто используемых, который создает DataFrame из словаря, списка или другой структуры данных. Этот метод предоставляет удобный способ для представления и работы с данными в табличной форме.

2. pd.read\_csv()

Метод с помощью которого осуществляется чтение данных из CSV файлов , который загружает данные из CSV файла в DataFrame, что позволяет легко импортировать и анализировать данные из внешних источников.

3. df.loc[]

Метод осуществляет фильтрацию данных в DataFrame по маскам. Это особенно полезно для выбора определенных подмножеств данных на основе условий.

4. df.groupby()

Метод используется для группировки данных по указанному столбцу и применения агрегирующих функций, таких как среднее значение, сумма и так далее. Это позволяет легко выполнять сводный анализ данных.

5. pd.merge()

Метод используется для объединения двух DataFrame по указанному ключу, позволяя соединять данные из разных источников в один DataFrame. [5]

Эти методы востребованы благодаря их эффективности и удобству в обработке данных. Они позволяют быстро и легко выполнять сложные операции, такие как фильтрация, агрегирование и объединение данных, что делает библиотеку Pandas незаменимым инструментом для анализа данных.

## **1.5 Библиотека Matplotlib**

Библиотека Matplotlib является одной из самых популярных библиотек для визуализации данных в Python. Она предоставляет широкие возможности для создания различных типов графиков и диаграмм, что делает ее незаменимым инструментом для аналитиков данных, ученых и инженеров.

Matplotlib позволяет создавать статические, анимированные и интерактивные визуализации, предоставляя гибкость и мощные инструменты для отображения данных.

Основные методы и функции:

1. plt.plot()

Метод используется для создания линейных графиков. Этот метод принимает в качестве аргументов списки или массивы значений по оси X и Y.

2. plt.hist()

Метод используется для построения гистограмм, которые показывают распределение данных.

3. plt.scatter()

Метод позволяет создавать графики рассеяния, отображающие зависимость между двумя переменными. Этот метод полезен для визуализации корреляций.

4. plt.bar()

Метод используется для создания столбчатых диаграмм, которые показывают значения для различных категорий.

5. plt.pie()

Метод используется для создания круговых диаграмм для визуализации пропорций.

Matplotlib также предоставляет возможности для настройки графиков, такие как добавление легенд, аннотаций и стилизации. Например, вы можете добавить легенду к вашему графику с помощью метода plt.legend(). [6]

Эти методы востребованы благодаря их простоте и возможности создавать различные типы визуализаций для анализа и представления данных. Графики и диаграммы, создаваемые с помощью Matplotlib, помогают лучше понять данные, выявить тренды и закономерности, а также эффективно представлять результаты анализа.

## **1.6 Библиотека Plotly**

Библиотека Plotly является мощным инструментом для создания интерактивных графиков и визуализаций данных в Python. Она позволяет создавать интерактивные графики, которые могут быть встроены в веб-приложения или опубликованы онлайн. Plotly поддерживает различные типы графиков, включая линейные графики, гистограммы, графики рассеяния, тепловые карты и многое другое.

Примеры популярных методов и функций:

1. plotly.express.line()

Метод используется для создания линейных графиков. Он принимает данные в виде DataFrame или словаря и позволяет легко настроить оси, заголовок и другие параметры.

2. plotly.express.histogram()

Метод используется для построения гистограмм, показывающих распределение данных по категориям или числовым значениям.

3. plotly.express.scatter()

Метод позволяет создавать графики рассеяния, которые отображают зависимость между двумя переменными. Графики рассеяния полезны для визуализации корреляций.

4. plotly.express.density\_heatmap()

Метод используется для создания тепловых карт, которые показывают плотность точек в области графика. Это полезно для визуализации концентраций данных.

5. plotly.express.pie()

Метод позволяет создавать круговые диаграммы для визуализации пропорций данных по категориям [7].

Эти методы востребованы благодаря их простоте и возможности создавать интерактивные визуализации данных. Графики и диаграммы, создаваемые с помощью Plotly, помогают лучше понять данные, выявить тренды и закономерности, а также эффективно представлять результаты анализа. Интерактивные элементы позволяют пользователям исследовать данные более глубоко, предоставляя мощный инструмент для анализа и визуализации данных.

# **2 Решение задач с использованием изученных библиотек**

В рамках летней практики мы не только изучали теоретические аспекты различных модулей Python, но и активно применяли их на практике, решая конкретные задачи. В этой части отчета будут рассмотрены задачи и методы их решения с использованием модулей random, numpy, math, pandas, matplotlib и plotly.

## **2.1 Встроенные модули**

В этом разделе будут рассмотрены задачи на использование библиотеки Random и экспериментальные задачи на сравнение времени выполнения некоторых функций.

### **2.1.1 Модуль Random**

В этом подразделе будет рассмотрена задача генерации пользовательских данных с использованием библиотеки Random.

Задача: id — случайное пятизначное число, логин — случайная последовательность из 6 маленьких английских букв, пароль — случайная последовательность из 10 неповторяющихся больших и маленьких английских букв и цифр.

Создайте функцию генерации id, функцию генерации логина и функцию генерации пароля. С использованием этих трёх функций напишите функцию, генерирующую список из N троек вида (id, логин, пароль), в которых ни id, ни логины, ни пароли НЕ повторяются (при этом повторный вызов функции не должен учитывать результаты предыдущей генерации).

Также должны выполняться следующие условия: (гласными считаем «aeiou», \* можно учитывать «y» по правилу "если буква «y» является первой буквой в слове, или перед ней стоит согласная буква, то считаем её гласной, иначе согласной").

Дополнительное условие: в id есть хотя бы одна «7», в логине ровно три согласные, в пароле есть хотя бы одна большая буква и хотя бы одна нечётная цифра.

Решение:

Была создана функция, генерирующая строку идентификатора заданной длины, состоящую из случайных цифр, с определенным количеством заданных цифр в случайных позициях:

def generate\_id(n=5, digit=7):

# Определяем количество цифр 'digit', которые будут включены в идентификатор.

# Количество выбирается случайным образом от 1 до 3.

count\_digit = random.randint(1, 3)

# Генерируем уникальные индексы для размещения цифры 'digit' в пределах длины идентификатора n.

index\_digits = set(random.randint(0, n - 1) for i in range(count\_digit))

# Инициализируем пустой список для хранения частей идентификатора.

res = []

# Проходимся по всем позициям от 0 до n-1.

for i in range(n):

if i in index\_digits:

# Если текущий индекс находится в index\_digits, добавляем 'digit' в результат.

res += [digit]

else:

# В противном случае добавляем случайную цифру от 0 до 9, исключая 'digit'.

res += [random.choice([int(elem) for elem in string.digits if elem != str(digit)])]

# Объединяем список цифр в строку.

res = ''.join(map(str, res))

# Возвращаем сгенерированный идентификатор.

return res

После была создана функция нужная для генерации логина, которая определяет, является ли заданный символ гласной буквой в английском языке, с учетом особого правила для буквы 'y':

def is\_vowel(char, previous\_char=None):

# Определяем строку гласных букв.

vowels = "aeiou"

# Обрабатываем случай, когда текущий символ 'y'.

if char == 'y':

# Если предыдущий символ отсутствует или является согласной буквой,

# считаем 'y' гласной.

if previous\_char is None or previous\_char in "bcdfghjklmnpqrstvwxyz":

return True

else:

# В противном случае считаем 'y' согласной.

return False

# Возвращаем True, если текущий символ является одной из гласных букв (кроме 'y').

return char in vowels

Далее была создана функция, генерирующая строку (логин) заданной длины, состоящую из случайных строчных букв английского алфавита, с заданным количеством согласных букв:

def generate\_login(n=6, cons=3):

# Бесконечный цикл для генерации логина, пока не будут выполнены условия

while True:

# Генерируем случайную последовательность из n маленьких английских букв

login = ''.join(random.choices(string.ascii\_lowercase, k=n))

# Считаем количество согласных в сгенерированной последовательности

consonants\_count = sum(1 for i in range(n) if not is\_vowel(login[i], login[i-1] if i > 0 else None))

# Если количество согласных равно заданному количеству (cons), возвращаем логин

if consonants\_count == cons:

return login

Была создана функция генерации случайного пароля заданной длины, содержащий как минимум одну заглавную букву и одну нечетную цифру:

# Определяем список строковых представлений нечетных цифр от 1 до 9.

odd\_digits = [str(i) for i in range(1, 10, 2)]

def generate\_password(n=10):

# Бесконечный цикл для генерации пароля, пока не будут выполнены условия

while True:

# Создаем список всех возможных символов: большие и маленькие английские буквы, а также цифры.

characters = list(string.ascii\_letters + string.digits)

# Генерируем случайную последовательность из n символов без повторений.

password = ''.join(random.sample(characters, n))

# Проверяем, содержит ли пароль хотя бы одну большую букву и хотя бы одну нечетную цифру.

if any(c.isupper() for c in password) and any(c in odd\_digits for c in password):

# Если условия выполнены, возвращаем сгенерированный пароль.

return password

В конце была создана функция, генерирующая список уникальных пользователей, каждый из которых имеет уникальные идентификатор, логин и пароль:

def generate\_user(n=3):

# Инициализируем множества для хранения уникальных идентификаторов, логинов и паролей.

existing\_ids = set()

existing\_logins = set()

existing\_passwords = set()

# Инициализируем список для хранения троек (id, логин, пароль).

tuples\_list = []

# Цикл для генерации n пользователей.

for \_ in range(n):

# Генерируем уникальный идентификатор.

while True:

id = generate\_id()

if id not in existing\_ids:

# Если id уникален, добавляем его в множество existing\_ids и выходим из цикла.

existing\_ids.add(id)

break

# Генерируем уникальный логин.

while True:

login = generate\_login()

if login not in existing\_logins:

# Если логин уникален, добавляем его в множество existing\_logins и выходим из цикла.

existing\_logins.add(login)

break

# Генерируем уникальный пароль.

while True:

password = generate\_password()

if password not in existing\_passwords:

# Если пароль уникален, добавляем его в множество existing\_passwords и выходим из цикла.

existing\_passwords.add(password)

break

# Добавляем тройку (id, логин, пароль) в список tuples\_list.

tuples\_list.append((id, login, password))

# Возвращаем список сгенерированных троек.

return tuples\_list

В результате выполнения функции generate\_user() будут созданы n уникальных пользователя, каждый из которых представлен тройкой (id, логин, пароль).

Вывод: были созданы функции generate\_id, generate\_login, generate\_password, которые создают id, логин и пароль соответственно. Функция is\_vowel является дополнительной функцией проверки букв на гласную или согласную. Итоговая функция generate\_user объединяет все эти функции для генерации n пользователей с уникальными данными.

### **2.1.2 Измерение времени выполнения**

В этом подразделе будут рассмотрены экспериментальные задачи на использование библиотеки Random для сравнения эффективности различных функций.

Задача 1: сравните время выполнения алгоритмов на различных размерах входных данных:

1. проверка на простоту перебором до N, N // 2 и до корня,

2. поиск элемента в списке и во множестве,

3. ответ на запросы о количестве вхождений элементов в список или букв в строку с помощью вызова метода count и с использованием Counter'а,

4. квадратичная сортировка, быстрая сортировка, встроенная сортировка на числах,

5. квадратичная сортировка, быстрая сортировка, встроенная сортировка на строках,

6. квадратичная сортировка, быстрая сортировка, встроенная сортировка на "пользователях", для которых сгенерированы id, логины и пароли,

7. квадратичная сортировка, быстрая сортировка, встроенная сортировка на "вакансиях".

Решение:

1. Проверка на простоту перебором до N, N/2, корень из N.

Вывод: по таблице 1 видно, что функция isprime\_n\_2 выполняется примерно в два раза быстрее чем функция isprime\_n, но isprime\_sqrt\_n выполняется быстрее isprime\_n\_2 примерно в 88 раз. Это значит, что самая эффективная по времени выполнения среди этих трёх функций это isprime\_sqrt\_n.

Таблица 1 – Время выполнения функций проверки на простоту

|  |  |
| --- | --- |
| Название функции | Время выполнения на 99 900 чисел |
| isprime\_n | 42.2 секунды |
| isprime\_n\_2 | 20.6 секунды |
| isprime\_sqrt\_n | 234 миллисекунды |

2. Поиск элемента в списке.

Вывод: по таблице 2 видно, что поиск элемента в списке работает в 25 раз медленнее, чем поиск элемента во множестве, поэтому асимптотически лучше выбирать множество для поиска элемента.

Таблица 2 – Время выполнения поиска элемента по списку и множеству

|  |  |
| --- | --- |
| Функция хранения элементов | Время выполнения поиска числа среди 1000000 чисел |
| list | 24.3 миллисекунды |
| set | 951 микросекунда |

3. Количество вхождений элементов в список с помощью метода count и Counter.

Вывод: по таблице 3 видно, что Counter работает быстрее практически в два раза чем метод count, поэтому для подсчета количества вхождений элементов в список лучше использовать Counter из встроенной библиотеки collections.

Таблица 3 – Время выполнения подсчета количества вхождений элементов в список с помощью count и Counter

|  |  |
| --- | --- |
| Счётчик | Время выполнения на 1000000 чисел |
| count | 164 миллисекунды |
| Counter | 68.8 миллисекунды |

4. Сортировка пузырьком, быстрая сортировка и встроенная сортировка на числах.

Вывод: по таблице 4 видно, что самая медленная по времени это квадратичная сортировка bubble\_sort, она медленее в 49 раз по сравнению с быстрой сортировкой quicksort, но также быстрая сортировка медленее sorted в 73 раза, но бывают такие случаи, что быстрая сортировка будет иметь такое же время выполнения как sorted, но при разных условия, например маленький размер данных.

Таблица 4 – Время выполнения квадратичной сортировки, быстрой сортировки и встроенной сортировки на числах

|  |  |
| --- | --- |
| Название функции | Время выполнения на 10000 числах |
| bubble\_sort | 11.4 секунды |
| quicksort | 23.2 миллисекунды |
| sorted | 314 микросекунды |

5. Сортировка пузырьком, быстрая сортировка и встроенная сортировка на строках.

Вывод: по таблице 5 видно, что самая медленная по времени это квадратичная сортировка bubble\_sort\_string, она медленее в 814 раз по сравнению с быстрой сортировкой quicksort\_string, но также быстрая сортировка медленее sorted в 10 раза, но бывает, что быстрая сортировка будет иметь такое же время выполнения как sorted при разных условия, например маленький размер данных. Также можно заметить, что работа со строками немного больше занимает времени чем работа с числами.

Таблица 5 – Время выполнения квадратичной сортировки, быстрой сортировки и встроенной сортировки на строках

|  |  |
| --- | --- |
| Название функции | Время выполнения на 10000 строках |
| bubble\_sort\_string | 13.6 секунды |
| quicksort\_string | 16.7 миллисекунды |
| sorted | 1.62 миллисекунды |

6. Сортировка пузырьком, быстрая сортировка и встроенная сортировка на "пользователях".

Вывод: по таблице 6 видно, что самая медленная по времени это квадратичная сортировка bubble\_sort\_users, она медленее в 307 раз по сравнению с быстрой сортировкой quicksort\_users, но также быстрая сортировка медленее sorted в 14 раза, но бывает, что быстрая сортировка будет иметь такое же время выполнения как sorted при разных условия, например маленький размер данных. Также можно заметить, что время выполнения медленне чем со строками и числами.

Таблица 6 – Время выполнения квадратичной сортировки, быстрой сортировки и встроенной сортировки на “пользователя”, для которых сгенерированы id, логины и пароли

|  |  |
| --- | --- |
| Название функции | Время выполнения на 10000 “пользователях” |
| bubble\_sort\_users | 16.5 секунды |
| quicksort\_users | 53.6 миллисекунды |
| sorted | 3.62 миллисекунды |

7. Сортировка пузырьком, быстрая сортировка и встроенная сортировка на "вакансиях".

Вывод: по таблице 7 видно, что самая медленная по времени это квадратичная сортировка bubble\_sort\_jobs, она медленее в 240 раз по сравнению с быстрой сортировкой quicksort\_jobs, но также быстрая сортировка медленее sorted в 10 раза, но бывает, что быстрая сортировка будет иметь такое же время выполнения как sorted при разных условия, например маленький размер данных. Также можно заметить, что время увеличилось по сравнению с числами, строками и объектами User.

Таблица 7 – Время выполнения квадратичной сортировки, быстрой сортировки и встроенной сортировки на “вакансиях”, для которых сгенерированы id, название и описание

|  |  |
| --- | --- |
| Название функции | Время выполнения на 10000 “вакансиях” |
| bubble\_sort\_users | 15.9 секунды |
| quicksort\_users | 66.2 миллисекунды |
| sorted | 6.47 миллисекунды |

Вывод по задаче 1: время выполнения различных алгоритмов сортировки изменяется в зависимости от данных, что указывает на наличие определённых констант. Тем не менее, можно наблюдать, что время выполнения пузырьковой сортировки всегда больше или равно времени выполнения быстрой сортировки, а время выполнения быстрой сортировки больше или равно времени выполнения встроенной функции сортировки sorted (bubble\_sort >= quicksort >= sorted).

Задача 2: Проверьте экспериментально, подходит ли коллекция list как основа для абстрактных структур данных стек и очередь. Для этого определите характер зависимости времени выполнения операций (добавления и извлечения элементов) от размера коллекции. Сравните с использованием коллекции deque.

Решение:

Сначала была создана функция stack\_push\_list, которая добавляет в стек, который является структурой list, элемент и создадим функцию stack\_pop\_list, которая удаляет из стека последний элемент:

def stack\_push\_list(stack):

# Добавляем случайное число в стек (список)

stack.append(random.randint(1000, 9999))

def stack\_pop\_list(stack):

# Проверяем, не пуст ли стек, и если нет, то удаляем последний элемент

if stack:

stack.pop()

Далее была создана функция queue\_push\_list, которая добавляет в очередь, которая является структурой list, с конца элемент и создадим функцию queue\_pop\_list, которая удаляет из очереди с начала первый элемент:

def queue\_push\_list(queue):

# Добавляем случайное число от 1000 до 9999 в конец очереди

queue.append(random.randint(1000, 9999))

def queue\_pop\_list(queue):

# Удаляем и возвращаем первый элемент из очереди, если она не пуста

if queue:

queue.pop(0)

Такие же функции используются для структуры list и deque.

Таблица 8 – Время выполнения стека и очереди на структуре list и deque.

|  |  |  |  |
| --- | --- | --- | --- |
| Абстрактный тип данных | Последовательный тип данных | Действие | Время выполнения на 1000000 элементах |
| Стек | list | добавление | 1.2 секунды |
| удаление | 222 миллисекунды |
| deque | добавление | 1.71 секунда |
| удаление | 220 миллисекунды |
| Очередь | list | добавление | 1.17 секунды |
| удаление | 3 минуты 21 секунда |
| deque | добавление | 1.19 секунды |
| удаление | 218 миллисекунды |

Вывод по задаче 2: время выполнения добавления в стек и время выполнения удаления из стека с использование list и deque примерно равны между собой. Время выполнения добавления в очередь с использованием list работает также как с использование deque, а удаление с использование list работает в 922 раз медленее чем с использованием deque. Таким образом делаем вывод, что для работы c данными, которые имеют две точки доступа, лучше использовать deque.

## **2.2 Модуль NumPy**

В этом разделе будут рассмотрены задачи, связанные с использованием библиотеки NumPy, а также экспериментальные задачи по сравнению времени выполнения различных функций и методов. Задания из хэндбука, касающиеся использования библиотек NumPy и math, не будут включены в данный раздел, как и их решения.

### **2.2.1 NumPy знакомство**

В этом подразделе будет рассмотрена ознакомительная задача на использование библиотеки NumPy.

Задача:

1. Сгенерировать матрицу размером 12 × 6, заполненную целыми числами.

2. Вывести на экран элемент с индексами [7, 5].

3. Вывести на экран четвертую строку матрицы.

4. Вывести на экран шестую строку матрицы в обратном порядке.

5. Изменить форму матрицы с 12 × 6 на 6 × 12.

6. Разделить каждый элемент матрицы на заданное число.

7. Найти максимум в каждой строке.

8. Найди минимальный элемент в первой строке.

9. Дан одномерный массив. Определить процент четных и нечетных

чисел.

10. В заданной матрице подсчитать количество двузначных чисел.

Решение:

1. Была сгенерирована матрица размером 12 × 6, заполненная целыми числами. Для этого количество строк в матрице было установлено равным 12, а количество столбцов — равным 6:

n = 12 # Количество строк в матрице

m = 6 # Количество столбцов в матрице

# Создаем одномерный массив чисел от 1 до n \* m и преобразуем его в матрицу размером n x m

matrix = np.arange(1, n \* m + 1).reshape(n, m)

2. На экран был выведен элемент матрицы с индексами [7, 5], соответствующий значению в указанной позиции:

matrix[7, 5]

3. На экран была выведена четвертая строка матрицы, что позволило ознакомиться с элементами, содержащимися в данной строке:

matrix[3]

4. На экран была выведена шестая строка матрицы в обратном порядке, что дало возможность увидеть элементы строки в противоположной последовательности:

matrix[5, ::-1]

5. Форма матрицы была изменена с 12 × 6 на 6 × 12, что позволило переупорядочить элементы матрицы в новом формате:

matrix = matrix.reshape(m, n)

6. Каждый элемент матрицы был разделен на заданное число, что привело к изменению значений всех элементов матрицы:

matrix = matrix // int(input())

7. Был найден максимальный элемент в каждой строке матрицы, что позволило выявить наибольшие значения в каждой строке:

matrix.max(axis=1)

8. Был определен минимальный элемент в первой строке матрицы, что дало информацию о наименьшем значении в этой строке:

matrix[0].min()

9. В одномерном массиве, состоящем из чисел был определен процент четных и нечетных чисел. Процентное соотношение было вычислено на основе количества четных и нечетных чисел в массиве:

# Создаем одномерный массив чисел от 1 до 55 (включительно)

matrix\_1 = np.arange(1, 56)

# Считаем количество четных чисел в массиве

even\_numbers = len(matrix\_1[matrix\_1 % 2 == 0])

# Считаем количество нечетных чисел в массиве

odd\_numbers = len(matrix\_1[matrix\_1 % 2 != 0])

# Общее количество чисел в массиве

all\_numbers = len(matrix\_1)

# Выводим процент четных чисел в массиве

print(f'Процент четных чисел: {(even\_numbers / all\_numbers):.0%}%')

# Выводим процент нечетных чисел в массиве

print(f'Процент нечетных чисел: {(odd\_numbers / all\_numbers):.0%}%')

10. В заданной матрице было подсчитано количество двухзначных чисел:

n = 12 # Количество строк в матрице

m = 6 # Количество столбцов в матрице

# Создаем одномерный массив чисел от 1 до n \* m и преобразуем его в матрицу размером n x m

matrix = np.arange(1, n \* m + 1).reshape(n, m)

# Создаем булеву матрицу, где True обозначает элементы, абсолютное значение которых в диапазоне от 10 до 99 (включительно)

matrix\_2 = (np.absolute(matrix) >= 10) & (np.absolute(matrix) <= 99)

# Считаем количество элементов в matrix\_2, соответствующих условию (где True)

matrix\_2.sum()

Вывод: был изучен базовый функционал библиотеки Numpy.

### **2.2.2 NumPy обработка матрицы**

В этом подразделе будет рассмотрена задача на эффективную работу с матрицами с помощью библиотеки NumPy.

Задача:

Выполнить обработку элементов прямоугольной матрицы 𝐴, имеющей 𝑛 строк и 𝑚 столбцов. Найти наименьший элемент столбца матрицы, для которого сумма абсолютных значений элементов максимальна.

Решение:

Сначала была создана матрица А, имеющая n строк и m столбцов:

# Получаем значения n и m от пользователя, преобразуя их в целые числа

n, m = int(input()), int(input())

# Создаем одномерный массив чисел от 1 до n \* m и преобразуем его в матрицу размером n x m

A = np.arange(1, n \* m + 1).reshape(n, m)

Для каждого столбца матрицы A рассчитывается сумма абсолютных значений элементов:

# Вычисляем сумму абсолютных значений элементов матрицы A по каждому столбцу

sum\_abs = np.absolute(A).sum(axis=0)

# Находим индекс столбца с максимальной суммой абсолютных значений

sum\_abs\_index = np.argmax(sum\_abs)

На основе рассчитанных сумм абсолютных значений для каждого столбца определяется индекс столбца, сумма которого является максимальной. Используя ранее найденный индекс столбца, определяется наименьший элемент в этом столбце:

A[:, sum\_abs\_index].min()

Вывод: в ходе работы был изучен процесс обработки и анализа матрицы с использованием библиотеки NumPy.

### **2.2.3 NumPy вектор и матрицы**

В этом подразделе будет рассмотрена задача с использованием операций над векторами и матрицами с помощью NumPy.

Задача:

Дана матрица 𝐵 размерности 𝑁 × 2 и матрица 𝐴 размерности 𝑁 × 2 ⋅ 𝑁. Условно разделить матрицу 𝐴 пополам на левую и правую части. Умножить столбцы правой половины матрицы 𝐴 на первый столбец матрицы 𝐵. Строки левой половины матрицы 𝐴 умножить на второй столбец матрицы 𝐵 в обратном порядке. Результат записать в две новые матрицы и посчитать определитель их произведения.

Решение:

Была создана матрицы A и B по условию задачи:

# Создаем одномерный массив чисел от 1 до n \* 2 и преобразуем его в матрицу размером B\_n x B\_m

B = np.arange(1, n \* 2 + 1).reshape(B\_n, B\_m)

# Создаем одномерный массив чисел от 1 до n \* 2 \* n и преобразуем его в матрицу размером A\_n x A\_m

A = np.arange(1, n \* 2 \* n + 1).reshape(A\_n, A\_m)

Матрица A была разделена на две части:

# A\_left содержит все строки, но только первые половину столбцов

A\_left = A[:, :A\_m // 2]

# A\_right содержит все строки, но только вторую половину столбцов

A\_right = A[:, A\_m // 2:]

Правая часть матрицы A​ была поэлементно умножена на первый столбец матрицы B, после чего получилась новая матрица matrix\_new\_1:

matrix\_new\_1 = A\_right \* B[:, 0].reshape(-1, 1)

Левая часть матрицы A была поэлементно умножена на второй столбец матрицы B в обратном порядке, в результате чего получилась матрица matrix\_new\_2:

matrix\_new\_2 = A\_left \* B[::-1, 1]

Было вычислено произведение двух новых матриц matrix\_new\_1 и matrix\_new\_2, в результате чего была получена матрица prod\_matrix:

prod\_matrix = matrix\_new\_1 @ matrix\_new\_2

Определитель матрицы prod\_matrix был вычислен с использованием функции np.linalg.det:

np.linalg.det(prod\_matrix)

Вывод: данное исследование продемонстрировало основные принципы работы с матрицами и векторами с помощью библиотеки NumPy, а также методы выполнения элементарных линейных операций, таких как поэлементное умножение и вычисление определителя матрицы.

### **2.2.4 NumPy задачка**

В этом подразделе будет рассмотрена задача Открытой олимпиады школьников «Информационные технологии» для 11 класса 2017 года, которую можно решить с помощью библиотеки NumPy.

Задача:

Двумерные символьные массивы строятся по следующему алгоритму:

1. Берется копия массива, полученного на предыдущем шаге (исходный массив для первой итерации), строится её зеркальное отражение относительно правой границы и присоединяется справа. Получается новый массив с таким же количеством строк и с вдвое большим количеством столбцов.
2. Берется копия массива, полученного на предыдущем шаге, строится её зеркальное отражение относительно верхней границы и присоединяется сверху. Получается новый массив с таким же количеством столбцов и с вдвое большим количеством строк.
3. Если получился массив размером N×N, то алгоритм завершается, в противном случае переход на шаг 1. Пусть был взят исходный массив, размером 3 на 3 элемента:

[A, B, C

B, C, A

C, A, B]

и принято N=192.

В построенном в результате выполнения алгоритма массиве R, размером 192×192 элемента определите значения элемента с индексами [x, y].

Решение:

Была создана функция two\_dimensional для создания двумерного символьного массива определенного размера:

def two\_dimensional(array, N):

# Создаем копию исходного массива, чтобы избежать изменения оригинала

array = np.copy(array)

# Создаем зеркальное отражение массива по горизонтали (отражаем столбцы)

# Соединяем оригинальный массив и его отражение по горизонтали

array\_right\_reflect = np.hstack((array, array[:, ::-1]))

# Создаем копию нового массива для дальнейшей работы

array = np.copy(array\_right\_reflect)

# Создаем зеркальное отражение массива по вертикали (отражаем строки)

# Соединяем отраженный массив и оригинальный по вертикали

array\_top\_reflect = np.vstack((array[::-1, :], array))

# Проверяем, если размер нового массива соответствует требуемому размеру N x N

if np.shape(array\_top\_reflect) == (N, N):

# Если да, то возвращаем его как результат

return array\_top\_reflect

# Если нет, рекурсивно вызываем функцию для нового массива, чтобы продолжить процесс

return two\_dimensional(array\_top\_reflect, N)

Был создан двумерный символьный массив:

# Исходный массив 3x3

array\_start = np.array([['A', 'B', 'C'], ['B', 'C', 'A'], ['C', 'A', 'B']])

# Задаем желаемый размер итогового массива

N = 192

# Функция для расширения массива до нужного размера N x N

array = two\_dimensional(array\_start, N)

Для определения значений элемента с индексами [x, y] был использован следующий код:

# Получаем значения x и y от пользователя и преобразуем их в целые числа

x, y = int(input()), int(input())

# Определяем размеры массива, используя функцию np.shape, которая возвращает кортеж (число строк, число столбцов)

array\_shape = np.shape(array)

# Проверяем, входят ли x и y в допустимые диапазоны индексов массива

# Индексы должны быть не больше, чем соответствующие размеры массива минус 1

if x <= array\_shape[0] - 1 and y <= array\_shape[1] - 1:

# Если индексы допустимы, выводим значение элемента массива с индексами (x, y)

print(array[x, y])

else:

# Если хотя бы один из индексов выходит за пределы размеров массива, выбрасываем исключение с сообщением об ошибке

raise ValueError(f'x: {x} и/или y: {y} не входят в размеры массива: {array\_shape}')

Вывод: в ходе работы была изучена и успешно реализована техника симметричного отражения матриц с использованием библиотеки NumPy.

**2.2.5 NumPy сравнение эффективности**

В этом подразделе будет рассмотрена экспериментальная задача для сравнения эффективности с помощью библиотеки NumPy и без её использования.

Задача:

Написать программу, реализующую алгоритм нахождения медианы элементов матрицы с использованием list, замерить время выполнения и сравнить с готовой реализацией алгоритма из библиотеки NumPy. Для более полного анализа проводить сравнения на разных размерностях входных данных. Например, для матриц можно взять три варианта реализации: 10 × 10, 100 × 100, 500 × 500.

Решение:

Была создана функция find\_median для нахождения медианы элементов в матрице:

def find\_median(matrix):

    # Преобразуем двумерный массив (матрицу) в одномерный список элементов

    if not isinstance(matrix, np.ndarray):

        elements = [item for row in matrix for item in row]

    else:

        elements = matrix.flatten()

    # Сортируем список элементов по возрастанию

    elements.sort()

    # Определяем количество элементов в списке

    n = len(elements)

    # Проверяем, четное или нечетное количество элементов

    if n % 2 != 0:

        # Если количество элементов нечетное, медианой является средний элемент

        median = elements[n // 2]

    else:

        # Если количество элементов четное, медианой является среднее значение двух средних элементов

        median = (elements[n // 2 - 1] + elements[n // 2]) / 2

    # Возвращаем найденную медиану

    return median

Для нахождения медианы с использованием библиотеки NumPy можно воспользоваться следующим методом:

np.median(np.array(matrix))

Вывод: как видно из данных в таблице 9, использование метода median из библиотеки NumPy для нахождения медианы элементов матрицы оказывается в 5 раз эффективнее по сравнению с функцией find\_median, когда элементы матрицы представлены в виде структуры list. В случае, когда элементы матрицы представлены в виде структуры numpy.ndarray, метод median из NumPy демонстрирует эффективность в 6 раз выше по сравнению с функцией find\_median.

Таблица 9 – Время выполнения нахождения медианы элементов матрицы с помощью метода median и с использованием функции find\_median.

|  |  |  |
| --- | --- | --- |
| Нахождение медианы | Тип данных элементов матрицы | Время выполнения на 3 размерах – 10 × 10, 100 × 100, 500 × 500 |
| np.median | list | 19.3 миллисекунды |
| numpy.ndarray | 3.37 миллисекунды |
| find\_median | list | 99.3 миллисекунды |
| numpy.ndarray | 21.4 миллисекунды |

## **2.3 Модуль Pandas**

В этом разделе будут рассмотрены задачи, связанные с использованием библиотеки Pandas. Задания из хэндбука, касающиеся использования библиотеки Pandas, а также их решения, в данном разделе рассматриваться не будут.

### **2.3.1 Модуль Pandas**

В этом подразделе будут рассмотрены задачи, связанные с использованием библиотеки Pandas, включая анализ и визуализацию данных.

Задача:

Вам дан фрагмент базы данных, содержащий информацию о продажах туров в различные города России. База данных состоит из трёх таблиц в формате csv с разделителем "точка с запятой".

Файл travel\_agents.csv содержит информацию о туроператорах. Заголовок таблицы имеет следующий вид: ID туроператора;Название;Адрес

Файл travels.csv содержит информацию об основных характеристиках каждого тура. Заголовок таблицы имеет следующий вид: ID тура;Город;Продолжительность, дней;Стоимость, на 1 чел

Файл sale\_of\_tour\_packages.csv содержит информацию о проданных турах за первый квартал 2022 года. Заголовок таблицы имеет следующий вид: ID операции;Дата;ID тура;ID туроператора;Количество проданных путёвок;

1. Найти количество человек, совершивших путешествие, воспользовавшись услугами туроператоров, чьи названия начинаются на букву "В".

2. Построить круговую диаграмму, отображающую общую стоимость путёвок и стоимость путёвок по каждому городу Архангельской области.

3. Построить диаграмму, показывающую стоимость и количество проданных путёвок в Москву каждым из туроператоров.

Решение:

1. Сначала из файлов travels.csv, travel\_agents.csv и sale\_of\_tour\_packages.csv были загружены соответствующие DataFrame. Далее из DataFrame с туроператорами (travel\_agents\_df) были отобраны только те туроператоры, названия которых начинаются на букву "В":

travel\_agents\_B = travel\_agents\_df[travel\_agents\_df['Название'].str.startswith('В')]

Затем из таблицы sale\_tour\_df были выбраны все строки, где ID туроператора соответствует одному из туроператоров, выбранных на предыдущем шаге:

travel\_B = sale\_tour\_df[sale\_tour\_df['ID туроператора'].isin(travel\_agents\_B['ID туроператора'])]

После фильтрации данных был произведен подсчёт общего количества проданных путёвок (и, соответственно, количества человек), совершивших путешествие с помощью туроператоров, названия которых начинаются на букву "В":

travel\_B['Количество проданных путёвок'].sum()

В результате этого подсчета было получено значение **12 859.**

Вывод: был изучен базовый функционал библиотеки Pandas, включая фильтрацию данных и агрегирование.

2. Был составлен список городов Архангельской области и преобразован в объект Series для дальнейшего использования в фильтрации данных:

# Создаем список городов Архангельской области

cities\_arkh = [

'Архангельск', 'Вельск', 'Каргополь', 'Мезень', 'Няндома', 'Северодвинск',

'Новодвинск', 'Мирный', 'Каргополь', 'Котлас', 'Коряжма', 'Онега', 'Шенкурск'

]

# Преобразуем список городов в объект Series из библиотеки pandas и присваиваем имя серии

cities\_arkh\_series = pd.Series(cities\_arkh, name='Города Архангельской области')

Затем были отобраны данные о путёвках только в города из списка cities\_arkh\_series. Из исходного DataFrame travels\_df оставлены только те строки, где город совпадает с одним из городов Архангельской области:

# Фильтруем строки в DataFrame travels\_df, оставляя только те, где значения в столбце 'Город' совпадают с городами из списка cities\_arkh\_series

travels\_arkh = travels\_df[travels\_df['Город'].isin(cities\_arkh\_series)].loc[:, ['Город', 'Стоимость, на 1 чел']]

# Устанавливаем столбец 'Город' в качестве индекса для DataFrame travels\_arkh

travels\_arkh.set\_index('Город', inplace=True)

Была рассчитана общая стоимость путёвок во все города Архангельской области:

total\_cost = travels\_arkh['Стоимость, на 1 чел'].sum()

На основе полученных данных была построена круговая диаграмма, отображающая распределение стоимости путёвок по каждому городу Архангельской области, а также общую стоимость путёвок:

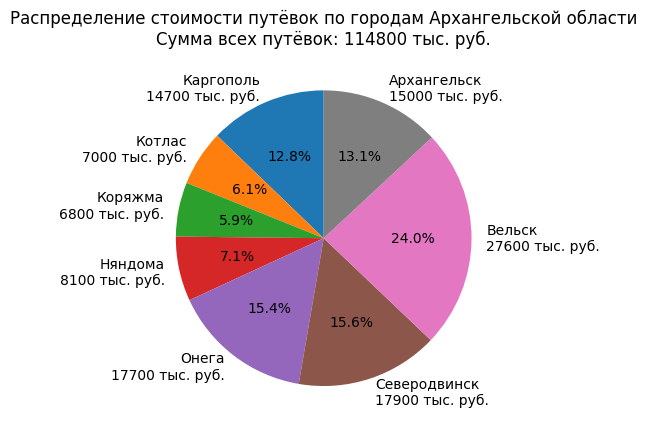


Рис. 1 Круговая диаграмма распределения стоимости путёвок по городам Архангельской области

Вывод: была изучена возможность визуализации данных с помощью библиотеки Pandas.

3. В исходной таблице travels\_df были отобраны все туры, которые имели пункт назначения в Москве:

moscow\_df = travels\_df[travels\_df['Город'] == 'Москва']

В таблице sale\_tour\_df были найдены все строки, где ID тура совпадает с ID туров в Москву. Это позволило выделить только те продажи, которые связаны с поездками в Москву:

moscow\_travel = sale\_tour\_df[sale\_tour\_df['ID тура'].isin(moscow\_df['ID тура'])]

Для того чтобы привязать продажи к конкретным туроператорам, данные о проданных турах в Москву были объединены с таблицей travel\_agents\_df по столбцу "ID туроператора":

moscow\_travel\_agents = moscow\_travel.merge(travel\_agents\_df, left\_on='ID туроператора', right\_on='ID туроператора')

Для каждого туроператора была рассчитана общая стоимость проданных путёвок в Москву, умножив количество проданных путёвок на стоимость путёвки для одного человека:

moscow\_travel\_agents['Стоимость путёвок'] = moscow\_travel\_agents['Количество проданных путёвок'] \*  moscow\_df.loc[0, 'Стоимость, на 1 чел']

Для определения общего количества проданных путёвок по каждому туроператору, данные были сгруппированы по названию туроператора и суммированы:

moscow\_count = moscow\_travel\_agents.groupby('Название')['Количество проданных путёвок'].sum()

Аналогично, была рассчитана общая стоимость проданных путёвок для каждого туроператора:

moscow\_sum = moscow\_travel\_agents.groupby('Название')['Стоимость путёвок'].sum()

На основе сгруппированных данных была построена диаграмма, отображающая как количество проданных путёвок, так и их общую стоимость для каждого туроператора:

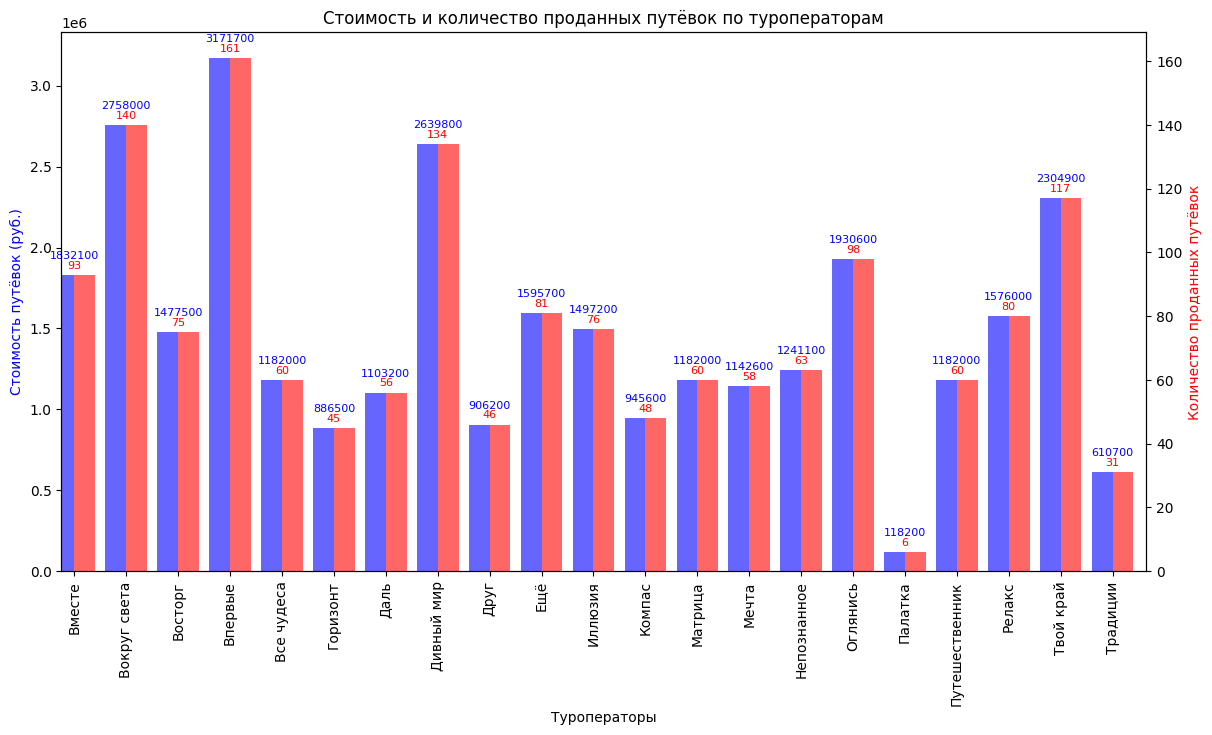


Рис. 2 Совмещенная столбчатая диаграмма стоимости и количества проданных путёвок по туроператорам

Вывод: были исследованы возможности библиотеки Pandas фильтровать данные, объединять таблицы, группировать и визуализировать результаты.

## **2.4 Визуализация**

В этом разделе будет проведено исследование с использованием библиотеки Pandas и Plotly, при этом доклад по библиотеке визуализации Plotly не будет рассмотрен.

### **2.4.1 Перевозки (статистика)**

В этом подразделе будет проведено статистическое исследование с использованием библиотек Pandas и Plotly, охватывающее анализ и визуализацию данных.

Задача:

Возьмите данные о перевозках между посёлками.

Проведите небольшое статистическое исследование, для визуализации результатов постройте не менее трех диаграмм и графиков (например, круговую диаграмму, показывающую суммарный расход бензина по датам по всем направлениям).

Решение:

Всего будет построено 4 диаграммы.

1. Был создан новый столбец "Направление", который представляет собой комбинацию пунктов отправления и назначения. Это позволяет отобразить каждое направление движения как отдельную категорию:

df['Направление'] = df['Пункт отправления'] + '-' + df['Пункт назначения']

Для упрощения анализа из таблицы были удалены столбцы, содержащие пункты отправления и назначения, оставив только необходимые для анализа данные:

df = df.loc[:, ['Дата', 'Направление', 'Расстояние', 'Расход бензина', 'Масса груза']]

Данные были сгруппированы по направлениям и датам. Для каждого направления на каждую дату был рассчитан суммарный расход бензина:

group\_df = df.groupby(['Направление', 'Дата'])['Расход бензина'].sum()

На основе сгруппированных данных была построена интерактивная столбчатая диаграмма. Диаграмма отображает суммарный расход бензина по каждому направлению в разрезе дат. Это позволяет визуализировать и сравнивать, как расход топлива меняется по разным направлениям и в разные даты:

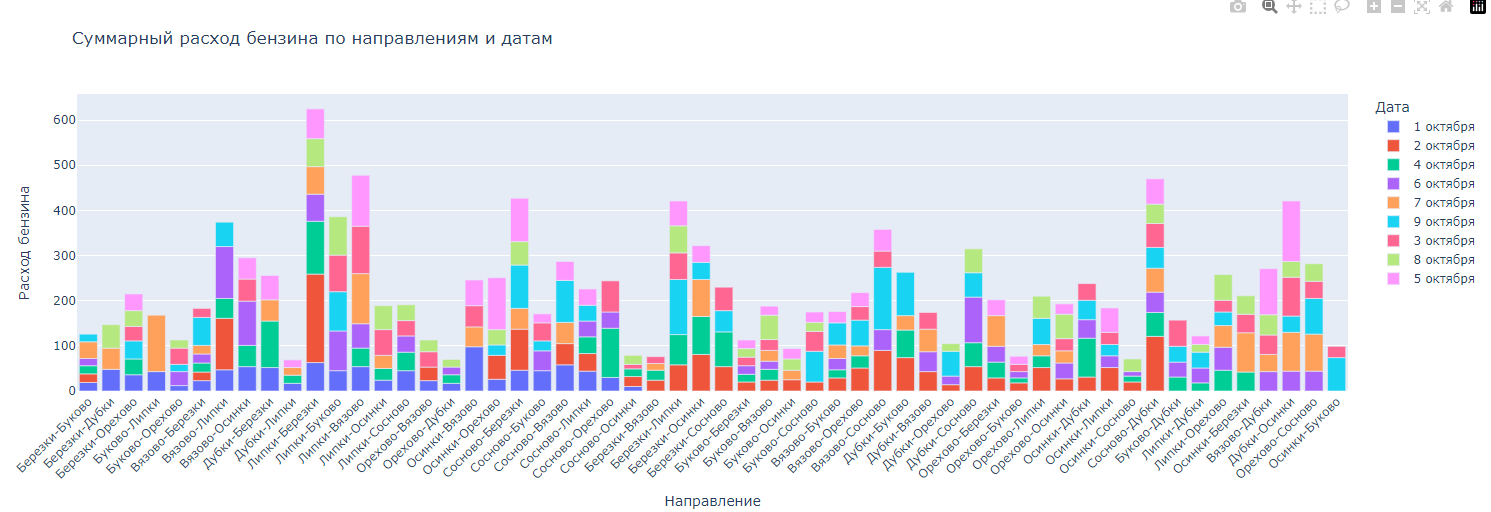


Рис. 3 Сложенная столбчатая диаграмма суммарного расхода бензина по направлениям и по датам

Вывод: в ходе исследования было установлено, что больше всего бензина расходует направление Липки-Березки, что значит, что они находятся далеко друг от друга, а самое близкое направление это Дубки-Липки. Также видно, что обратные направления довольно сильно различаются по расходу бензина, либо это зависит от других факторов (например водитель ехал другой дорогой), либо это искусственные данные.

2. На основе данных из файла task19.csv был рассчитан показатель эффективности. Эффективность определялась как отношение расхода бензина к пройденному расстоянию для каждой записи:

df['Эффективность'] = df['Расход бензина'] / df['Расстояние']

В ходе анализа было выявлено, что расстояния от пункта отправления до пункта назначения и обратно могут различаться. Для более точного расчета были сгруппированы данные по парам "Пункт отправления" и "Пункт назначения", а также по обратным маршрутам. Рассчитаны суммарные значения расстояний и расхода бензина для каждого направления:

group\_from\_to = pd.DataFrame(df.groupby(['Пункт отправления', 'Пункт назначения']).agg({'Расстояние': 'sum', 'Расход бензина': 'sum'})).reset\_index()

group\_to\_from = pd.DataFrame(df.groupby(['Пункт назначения', 'Пункт отправления']).agg({'Расстояние': 'sum', 'Расход бензина': 'sum'})).reset\_index()

Чтобы получить более точные данные для анализа, таблицы с суммарными расстояниями и расходом бензина для прямого и обратного маршрутов были объединены. Для объединения использовались названия столбцов с суффиксами для обозначения направления маршрута:

merged\_df = pd.merge(group\_from\_to, group\_to\_from, on=['Пункт назначения', 'Пункт отправления'], suffixes=('\_1', '\_2'))

Далее были рассчитаны средние значения расстояния и расхода бензина для каждого маршрута:

merged\_df['Расстояние (среднее)'] = merged\_df[['Расстояние\_1', 'Расстояние\_2']].mean(axis=1)

merged\_df['Расход бензина (средний)'] = merged\_df[['Расход бензина\_1', 'Расход бензина\_2']].mean(axis=1)

После удаления ненужных столбцов в итоговой таблице была рассчитана эффективность, основанная на средних значениях:

df['Эффективность'] = df['Расход бензина (средний)'] / df['Расстояние (среднее)']

На основе полученных данных была построена тепловая карта, которая наглядно отображает эффективность расхода бензина по разным маршрутам. Такая визуализация позволяет легко выявить маршруты с наибольшей и наименьшей эффективностью, что может быть полезно для оптимизации логистики и управления расходом топлива:

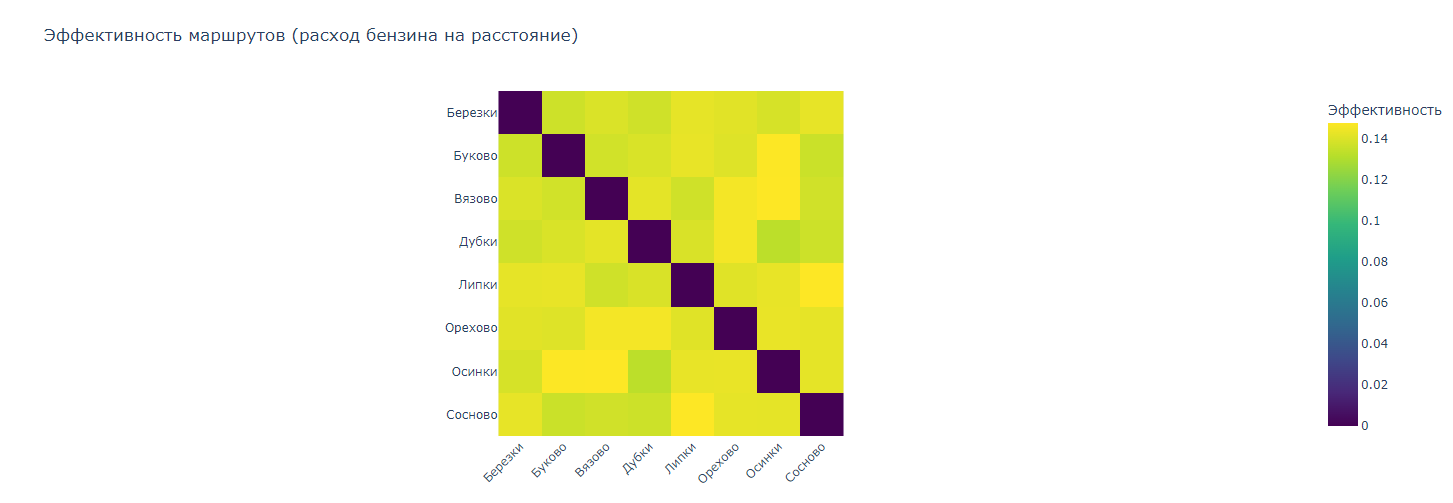


Рис. 4 Тепловая карта эффективности маршрутов

Вывод: в ходе исследования было выяснено, что практически во всех случаях эффективность расхода бензина одинакова, что указывает на искусственный характер данных.

3. Данные из файла task19.csv были предварительно обработаны. В исходном наборе данных названия месяцев были представлены на русском языке, поэтому была выполнена замена русскоязычных названий месяцев на их английские эквиваленты с использованием следующего словаря:

months\_mapping = {

"января": "January",

"февраля": "February",

"марта": "March",

"апреля": "April",

"мая": "May",

"июня": "June",

"июля": "July",

"августа": "August",

"сентября": "September",

"октября": "October",

"ноября": "November",

"декабря": "December"

}

Замена была произведена для всех значений столбца "Дата":

for rus, eng in months\_mapping.items():

df['Дата'] = df['Дата'].str.replace(rus, eng)

После замены названий месяцев, столбец Дата был преобразован в формат datetime для корректной работы с временными рядами. Формат даты был установлен так, чтобы сначала шёл день, а затем месяц:

df['Дата'] = pd.to\_datetime(df['Дата'], format='%d %B', dayfirst=True)

На основе подготовленных данных была построена диаграмма рассеяния, которая отображает изменение расхода бензина во времени. Данная визуализация позволяет легко отслеживать тенденции расхода топлива в разные даты, выявляя возможные закономерности или аномалии в данных:

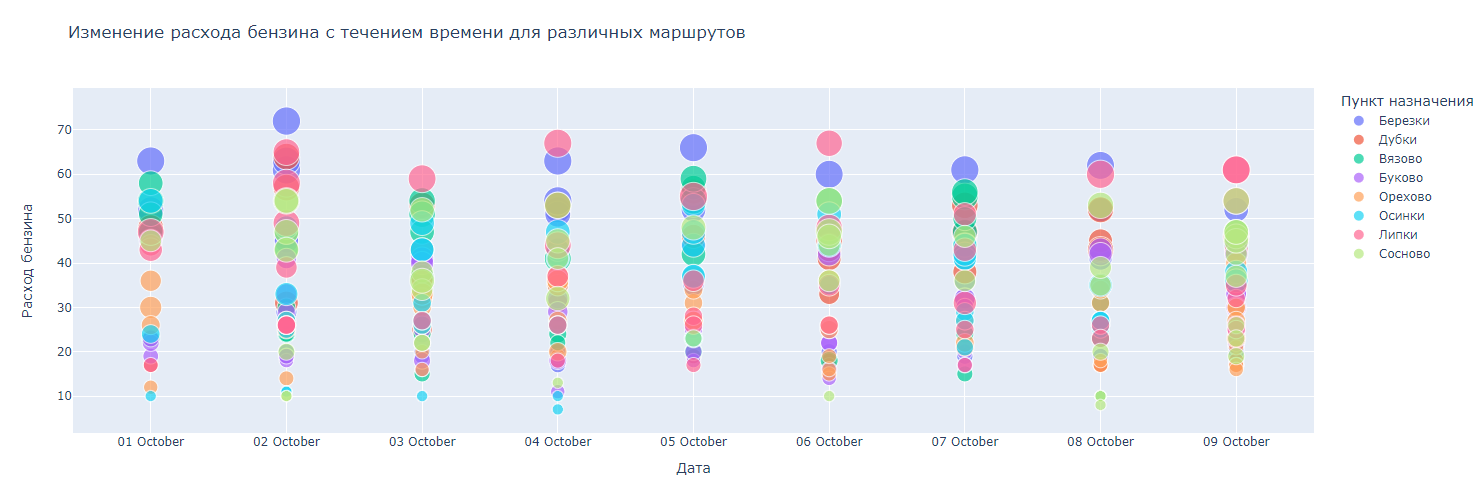


Рис. 5 Диаграмма рассеяния изменения расхода бензина с течением время для различных маршрутов

Вывод: в ходе исследования было установлено, что пункты назначения Березки и Липки потребляют наибольшее количество бензина в день, тогда как Осинки — наименьшее. Это позволяет предположить, что Осинки находятся в непосредственной близости ко всем другим пунктам, что объясняет их низкий расход топлива. В то же время, Березки и Липки, вероятно, находятся дальше всего от других пунктов (за исключением Осинок), что объясняет их высокий расход бензина.

4. Исходные данные из файла task19.csv были загружены и преобразованы. Данные были сгруппированы по столбцу "Дата" и рассчитано суммарное значение столбца "Расстояние" для каждой группы. Затем полученная таблица была отсортирована по убыванию суммарного расстояния:

group\_df = df.groupby('Дата')['Расстояние'].sum()

group\_df = pd.DataFrame(group\_df).sort\_values('Расстояние', ascending=False).reset\_index()

На основе подготовленной таблицы была построена диаграмма воронки, которая позволяет наглядно отобразить распределение суммарного расстояния по каждой дате. Визуализация помогает выявить даты с наибольшими значениями суммарного расстояния и проследить тенденции распределения:

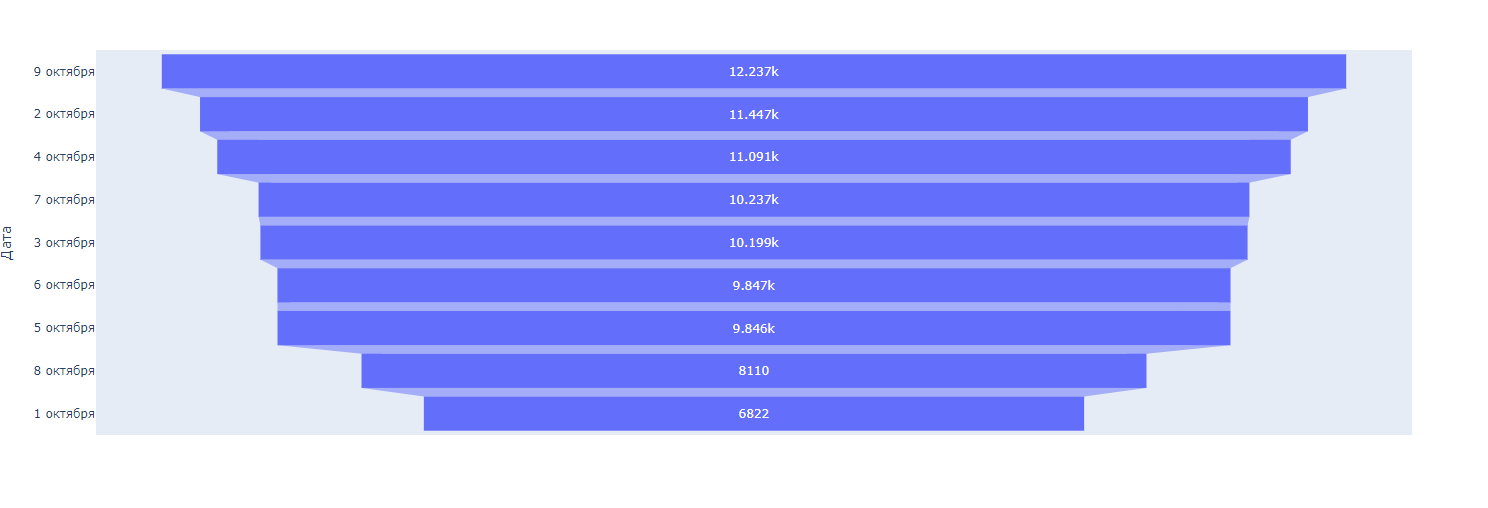


Рис. 6 Диаграмма воронки, отображающая суммарное расстояние по каждой дате

Вывод: в ходе расследования было выяснено, что в начале месяца расстояние, пройденное транспортными средствами, было меньше, но по мере увеличения числа дней оно возрастало. Это может указывать на зависимость пройденного расстояния от внешних факторов, таких как изменение маршрутов водителями, проведение мероприятий с перекрытием улиц, ремонт дорог и другие причины, которые не отражены в представленных данных. Альтернативно, такой тренд может свидетельствовать о наличии искусственных данных.

## **2.5 Итоговое задание**

В этом разделе будет проведен анализ с помощью библиотек Pandas и Plotly.

### **2.5.1 Анализ результатов экзаменов по информатике**

В этом подразделе будет проведен анализ результатов экзаменов по информатике с использованием библиотек Pandas и Plotly. В ходе анализа будут проверены выдвинутые гипотезы, которые будут либо подтверждены, либо опровергнуты.

Задача:

Изучите данные результатов экзаменов. Визуализируйте распределения значений признаков.

Обобщите результаты по районам и видам школ (гимназии, лицеи и т.п.). Найдите лучшие и худшие школы и районы.

Найдите закономерности в данных. Придумайте гипотезы, подтвердите или опровергните их. Наглядно демонстрируйте результаты с помощью визуализации.

Гипотеза 1: школы с разным типом (гимназии, лицеи, средние общеобразовательные школы и т.п.) показывают различия в результатах экзаменов.

Гипотеза 2: предполагается, что лицеи и гимназии имеют лучшие средние результаты по сравнению с обычными школами.

Гипотеза 3: результаты экзаменов (ОГЭ и ЕГЭ) зависят от расстояния школы до Саратова.

Решение:

1. Предобработка данных.

В начале будет выполнена предобработка данных из файлов oge.csv, ege 2022.csv, geo.xlsx и codificator.xlsx. Поскольку этот этап составляет треть проекта, в этом пункте будут рассмотрены ключевые аспекты предобработки данных.

1.1 ОГЭ

После загрузки данных из файла oge.csv сразу были исправлены грамматические ошибки в названиях столбцов:

oge\_df = oge\_df.rename(columns={'Оцека по 100': 'Оценка (100)', 'Оцека по 5': 'Оценка (5)'})

Затем были проверены данные на корректность, начиная с проверки баллов части B. Для начала было подсчитано количество плюсов и минусов в столбце "Часть B":

oge\_df\_copy = oge\_df.copy()

oge\_df\_copy['+\_Часть В'] = oge\_df\_copy['Часть В'].str.count(r'\+')

oge\_df\_copy['-\_Часть В'] = oge\_df\_copy['Часть В'].str.count('-')

После этого было проверено наличие значений NaN в столбцах, относящихся к части B:

oge\_df.loc[:, '1\_Часть В':'12\_Часть В'].isna().sum(axis=1).sum()

В результате проверки не было обнаружено значений NaN.

Для удобства подсчёта была проведена замена символов + на 1 и – на 0:

oge\_df.loc[:, '1\_Часть В':'12\_Часть В'] = oge\_df.loc[:, '1\_Часть В':'12\_Часть В'].replace({'+': 1, '-': 0})

Затем было подсчитано количество плюсов и минусов в столбце "Часть B" и проверено соответствие этих значений со значениями в отдельных столбцах частей B:

(oge\_df\_copy['+\_Часть В'] != plus\_columns).sum()

(oge\_df\_copy['-\_Часть В'] != minus\_column).sum()

В результате проверки было подтверждено, что баллы в части B были корректно выставлены.

Поскольку столбец "Часть В" не требуется для дальнейшего анализа, он был удалён:

oge\_df = oge\_df.drop(columns=['Часть В'])

Также были удалены столбцы "Предмет", "Код предмета", "Тип ОО", "Часть D", "Оценка (100)", так как они содержат только одно уникальное значение и не несут информативной ценности.

Аналогично, как и для столбца "Часть В", была проведена проверка корректности выставленных баллов в части C, после чего столбец "Часть C" также был удалён.

Для удобства дальнейшего анализа были добавлены столбцы, содержащие сумму баллов по частям B и C:

oge\_inf\_oo\_df['Сумма баллов части В'] = oge\_inf\_oo\_df.loc[:, '1\_Часть В':'12\_Часть В'].sum(axis=1)

oge\_inf\_oo\_df['Сумма баллов части С'] = oge\_inf\_oo\_df.loc[:, '1\_Часть С':'3\_Часть С'].sum(axis=1)

Далее все наименования столбцов были переведены на английский язык и приведены к стилю "змеиный регистр":

oge\_inf\_oo\_df = oge\_inf\_oo\_df.rename(columns={'Код ОО': 'code\_oo', 'Краткое наименование ОО': 'short\_name\_oo', 'Вид ОО': 'type\_oo',

'Выпускник': 'graduate', 'Вариант': 'variant', 'Сумма баллов части В': 'sum\_b', 'Сумма баллов части С': 'sum\_c',

'Первичный балл': 'primary\_score', '% Оценка': 'score\_%', 'Оценка (5)': 'score\_5',

'1\_Часть В': 'ex\_1\_b', '2\_Часть В': 'ex\_2\_b', '3\_Часть В': 'ex\_3\_b', '4\_Часть В': 'ex\_4\_b', '5\_Часть В': 'ex\_5\_b',

'6\_Часть В': 'ex\_6\_b', '7\_Часть В': 'ex\_7\_b', '8\_Часть В': 'ex\_8\_b', '9\_Часть В': 'ex\_9\_b',

'10\_Часть В': 'ex\_10\_b', '11\_Часть В': 'ex\_11\_b', '12\_Часть В': 'ex\_12\_b', '1\_Часть С': 'ex\_13\_c',

'2\_Часть С': 'ex\_14\_c', '3\_Часть С': 'ex\_15\_c'})

Так как в столбцах частей B изначально были строковые данные, они были преобразованы в целочисленные:

columns\_b = oge\_inf\_oo\_df.filter(regex=r'^ex\_\d+\_b$').columns

oge\_inf\_oo\_df[columns\_b] = oge\_inf\_oo\_df[columns\_b].astype(int)

Также была проведена проверка соответствия первичного балла сумме баллов за части B и C, и все данные оказались корректными.

В завершение был добавлен столбец с кодом региона, который был извлечен из кода ОО:

oge\_inf\_oo\_df['code\_region'] = oge\_inf\_oo\_df['code\_oo'].astype(str).str[:3].astype(int)

1.2 ЕГЭ

После загрузки данных из файла ege 2022.csv были заменены русскоязычные названия столбцов на английские и приведены к стилю "змеиный регистр":

ege\_df = ege\_df.rename(columns={'Код ОО': 'code\_oo', 'Первичный балл': 'primary\_score', 'Балл': 'score'})

Затем была проверена корректность баллов по каждому заданию в сравнении с первичным баллом, а также изменены названия соответствующих столбцов:

(ege\_df.primary\_score != ege\_df.loc[:, 'Task1':'Task27'].sum(axis=1)).sum()

rename\_columns = {f'Task{i}': f'ex\_{i}' for i in range(1, 28)}

rename\_columns = {f'Task{i}': f'ex\_{i}' for i in range(1, 28)}

ege\_df.rename(columns=rename\_columns, inplace=True)

Был удален столбец "Задания с кратким ответом", поскольку он не требовался для дальнейшего анализа:

ege\_df.drop(columns='Задания с кратким ответом', inplace=True)

Для удобства был добавлен столбец с коротким названием образовательной организации (ОО):

oge\_df\_code\_oo = oge\_inf\_oo\_df.set\_index('code\_oo')['short\_name\_oo'].to\_dict()

ege\_df['short\_name\_oo'] = ege\_df['code\_oo'].map(oge\_df\_code\_oo)

Поскольку в коротких названиях некоторых ОО присутствовали значения NaN, была проведена предобработка файла codificator.xlsx в пункте 1.3. После этого недостающие названия были заполнены соответствующими значениями из codificator.xlsx:

merged\_df = ege\_df\_copy.merge(codificator\_df[['code\_oo', 'short\_name\_oo']], on='code\_oo', how='left', suffixes=('', '\_y'))

merged\_df['short\_name\_oo'] = merged\_df['short\_name\_oo'].fillna(merged\_df['short\_name\_oo\_y'])

Далее был добавлен столбец с кодом региона, который извлекался из кода ОО:

ege\_df['code\_region'] = ege\_df['code\_oo'].astype(str).str[:3].astype(int)

Также был добавлен столбец с оценкой в пятибалльной системе:

def categorize(value):

    if value < 39:

        return 2

    elif 40 <= value < 56:

        return 3

    elif 57 <= value <= 72:

        return 4

    else:

        return 5

ege\_df['score\_5'] = ege\_df['score'].apply(categorize)

Наконец, был проверен тип данных всех столбцов, и было установлено, что все типы данных соответствуют требованиям.

1.3 Кодификатор

В файле codificator.xlsx были переименованы столбцы, заменив русскоязычные названия на английские и приведя их к стилю "змеиный регистр".

1.4 Регионы

После загрузки данных из файла geo.xlsx были проверены названия столбцов. Русскоязычные названия столбцов были заменены на английские и приведены к стилю "змеиный регистр":

geo\_df = geo\_df.rename(columns={'Код района': 'code\_region', 'Район': 'region', 'Расстояние до Саратова': 'distance\_to\_saratov'})

Были обнаружены лишние пробелы в названиях регионов в столбце region, поэтому они были удалены:

geo\_df = geo\_df.applymap(lambda x: x.strip() if isinstance(x, str) else x)

2. Визуализация распределения значений признаков результатов экзаменов.

Будут рассмотрены графики, сделанные на основе предоставленных данных, результатов ОГЭ и ЕГЭ, которые показывают распределение баллов.

2.1 ОГЭ

Гистограмма, изображенная на рисунке 7, показывает, что большинство учеников получили 8 баллов, которые соответствуют оценке "3". Таким образом, график показывает, что в условиях экзамена большинство учеников демонстрируют базовый уровень подготовки.

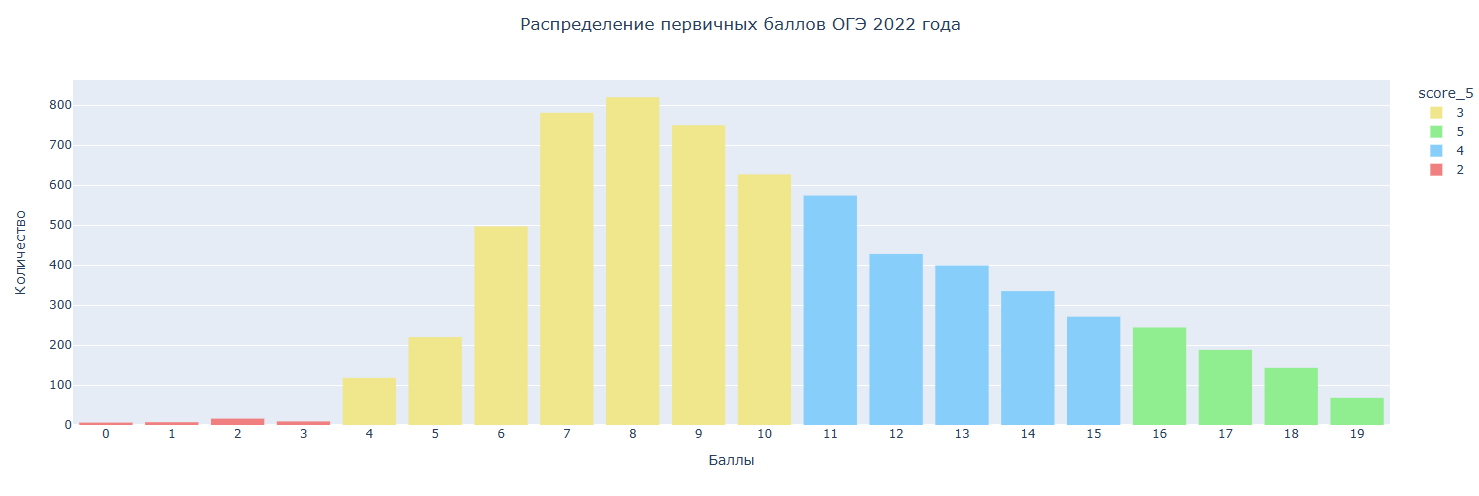


Рис. 7 Гистограмма распределения первичных баллов ОГЭ с цветами, которые отвечают за оценку по пятибалльной шкале

График на рисунке 8 показывает, что большинство учеников набрали средние баллы (от 7 до 13), что указывает на средний уровень успеваемости. Распределение баллов относительно широкое (от 0 до 19), но медиана в 10 баллов отражает, что половина учеников получила результаты выше или ниже этого значения. Это может означать, что уровень подготовки большинства учеников соответствует базовым требованиям экзамена или сам экзамен был качественно составлен. 

Рис. 8 "Ящик с усами", показывающий распределения первичных баллов ОГЭ

2.2 ЕГЭ

Гистограмма на рисунке 9 показывает, что большинство учеников набрали средние баллы на ЕГЭ. Это говорит о нормальном распределении результатов, где большинство учеников показали средний уровень успеваемости и о качественно составленном экзамене.



Рис. 9 Гистограмма, распределения первичных баллов ЕГЭ

График на рисунке 10 показывает, что большинство учеников сдали экзамен на средние и выше среднего баллы, что может свидетельствовать о хорошей подготовке учащихся и адекватной сложности самого экзамена

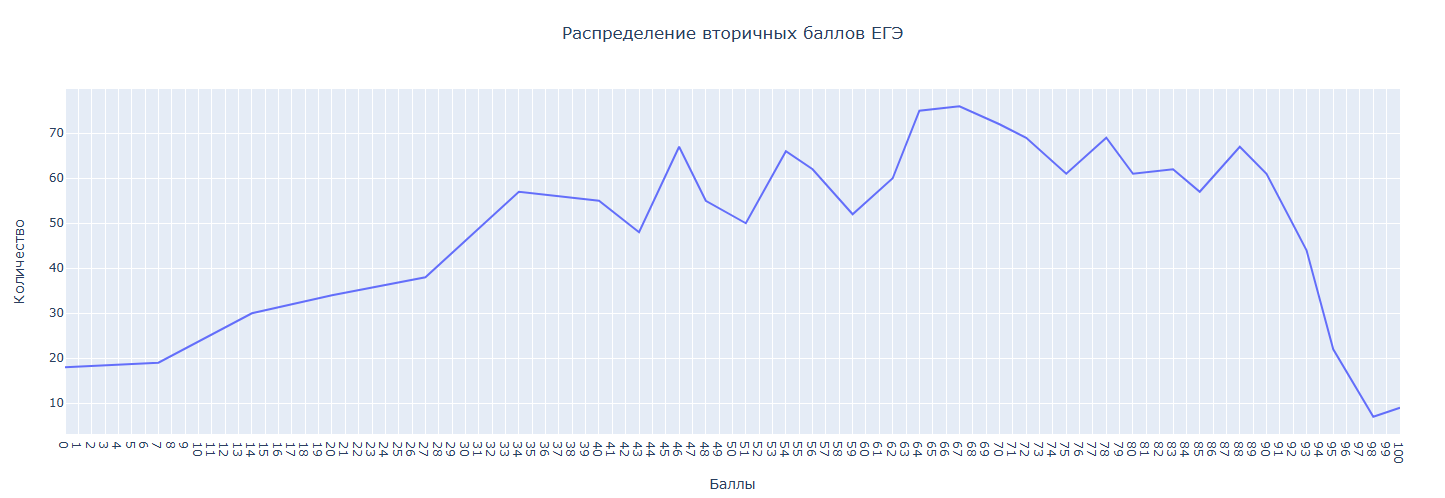


Рис. 10 Линейный график распределения вторичных баллов ЕГЭ

График на рисунке 11 иллюстрирует, что ученики в основном успешно сдали экзамен, с преобладанием результатов в диапазоне средних и выше среднего баллов, что также подтверждает хорошо составленный экзамен и хорошую подготовку учеников.

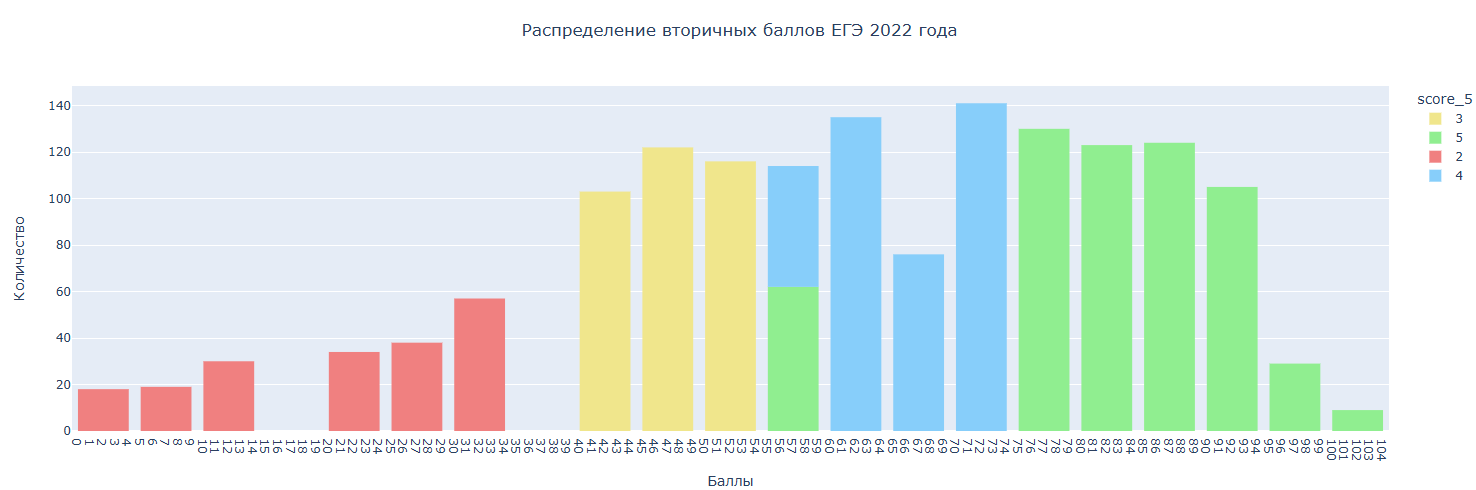


Рис. 11 Гистограмма распределения вторичных баллов ЕГЭ с цветами, которые отвечают за оценку по пятибалльной шкале

Для лучшей наглядности были также сделаны графики "ящики с усами" распределения первичных и вторичных баллов ЕГЭ, которые подтверждают всё вышесказанное.

3. Визуализация результатов ОГЭ и ЕГЭ по районам и видам школ.

Будут также рассмотрены графики, сделанные на основе предоставленных данных, результатов ОГЭ и ЕГЭ, которые показывают распределение баллов по районам и видам школ. Также будут найдены худшие и лучшие школы и районы.

3.1 ОГЭ

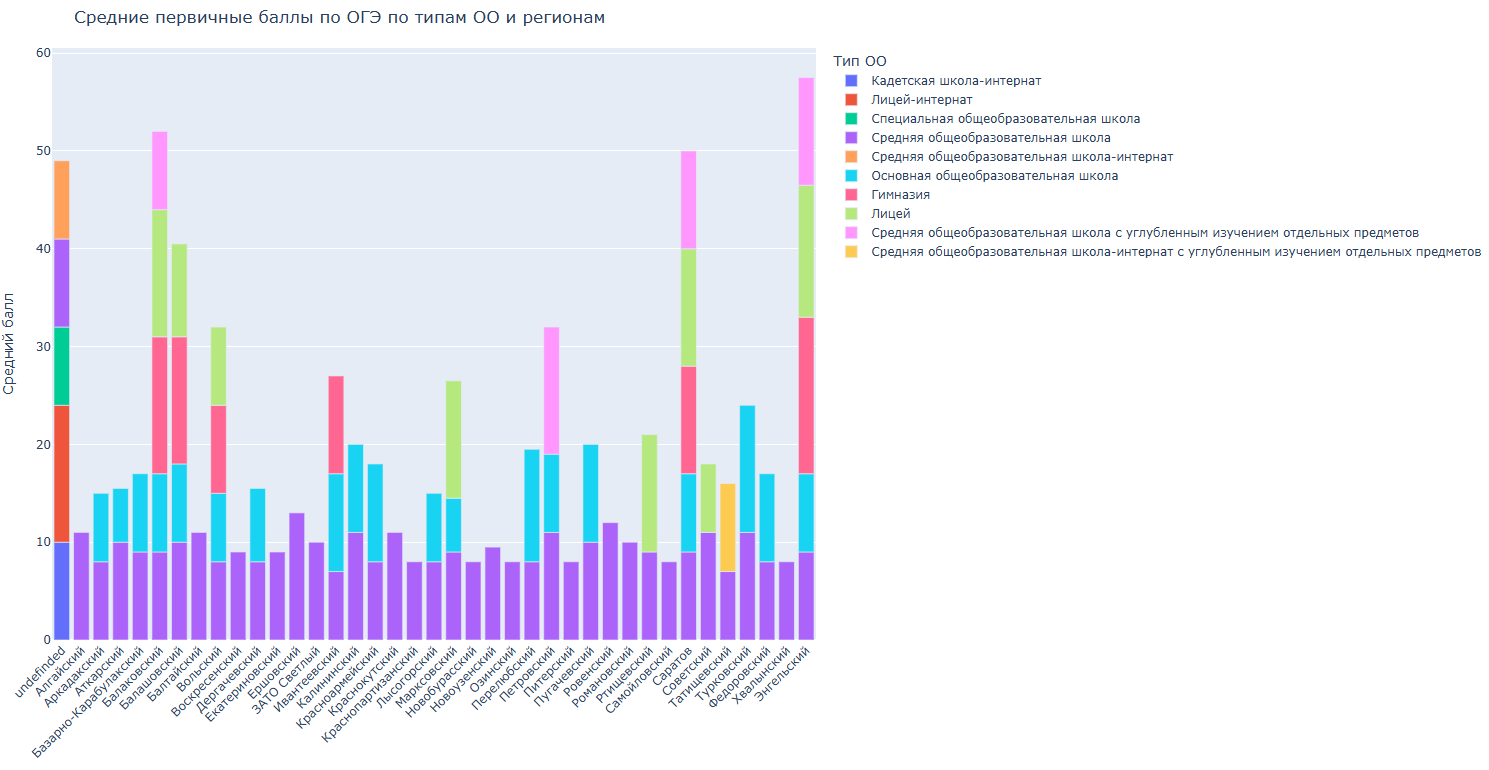
По гистограмме можно сделать вывод, что результаты ОГЭ зависят как от типа образовательного учреждения, так и от региона. Лицеи и гимназии показывают высокие результаты в большинстве регионов, особенно в Москве, Республике Башкортостан и Чеченской Республике. В этих регионах заметны большие различия в баллах между типами школ. В других регионах, таких как Свердловская область и Республика Татарстан, результаты распределены более равномерно, и разница между школами менее выражена. Таким образом, регионы с высоким уровнем образования чаще имеют специализированные школы, показывающие лучшие результаты.

Рис. 12 Гистограмма средних первичных баллов ОГЭ по типам образовательных организаций и регионам

График на рисунке 13 показывает, что наибольшие средние баллы у "Лицеев-интернатов" (14) и "Гимназий" (12), что указывает на высокий уровень подготовки в этих типах школ. "Лицеи" (12) и "Кадетские школы-интернаты" (10) также показывают хорошие результаты. Наименьшие средние баллы у "Средних общеобразовательных школ" с углубленным изучением отдельных предметов (8) и "Специальных общеобразовательных школ" (8). Это говорит о том, что специализированные школы, такие как лицеи и гимназии, обеспечивают лучшие результаты по сравнению с обычными и специализированными школами.

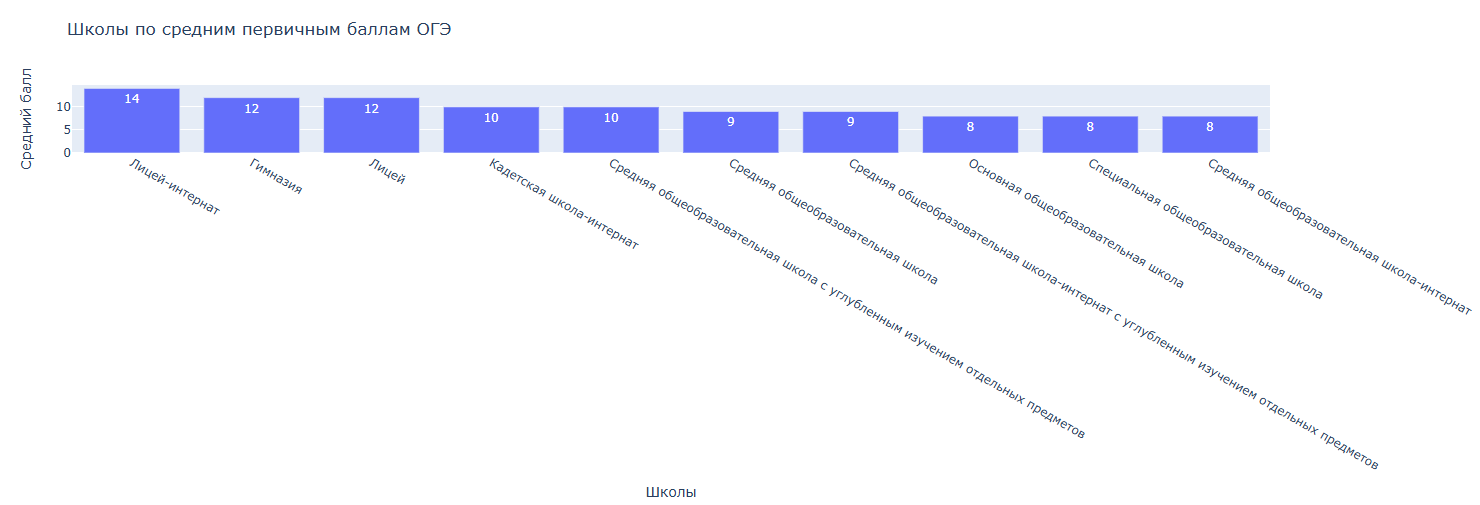


Рис. 13 График, показывающий средние первичные баллы ОГЭ для разных типов школ

По графику на рисунке 14 видно, что лидирует МАОУ "ФТЛ № 1" с наибольшим средним баллом — 17. За ней следуют "МАОУ Гимназия № 1", "МАОУ Гимназия № 3" с баллами 16 и так далее. Это говорит о том, что школы с наибольшим средним баллом ОГЭ часто являются гимназиями и лицеями, но не всегда, так как в график попала и школа с баллом 14.5.



Рис. 14 График топ 10 лучших школ по среднему первичному баллу ОГЭ с учётом количества учеников

По графику на рисунке 15 ясно, что наименьшие средние баллы у школ МОУ ВМР "СОШ № 47 р.п. Сенной", МОУ "СОШ № 1", и МОУ "СОШ № 38" — по 6 баллов. Эти данные указывают на то, что данные школы, возможно, испытывают трудности с подготовкой учащихся к ОГЭ по сравнению с другими образовательными учреждениями и также доказывает, что у общеобразовательных школ подготовка к ОГЭ намного ниже по сравнению с лицеями и гимназиями.

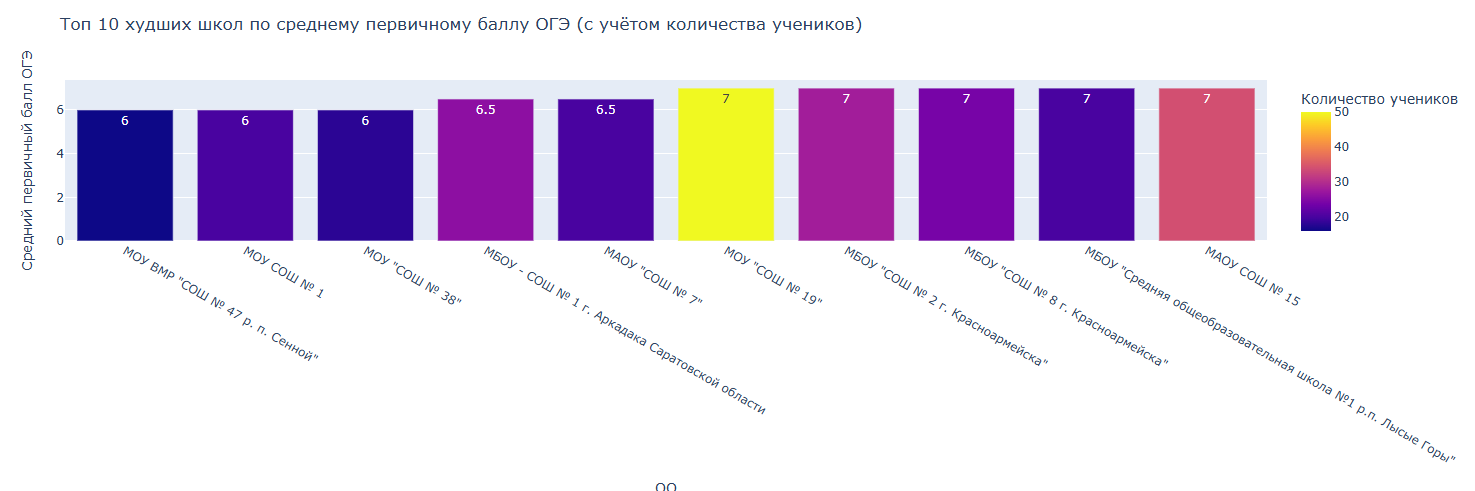


Рис. 15 График топ 10 худших школ по среднему первичному баллу ОГЭ с учётом количества учеников

3.2 ЕГЭ

График, изображенный на рисунке 16, показывает, что самые высокие средние баллы ЕГЭ у гимназий, лицеев и лицеев-интернатов — по 18 баллов. Средние общеобразовательные школы с углубленным изучением отдельных предметов также показывают высокие показатели. Напротив, кадетские школы-интернаты и средние общеобразовательные школы-интернаты с углубленным изучением отдельных предметов демонстрируют самые низкие результаты. Эти данные указывают на то, что специализированные школы, такие как гимназии и лицеи, обеспечивают более высокий уровень подготовки к ЕГЭ, тогда как школы-интернаты показывают значительно более низкие результаты.

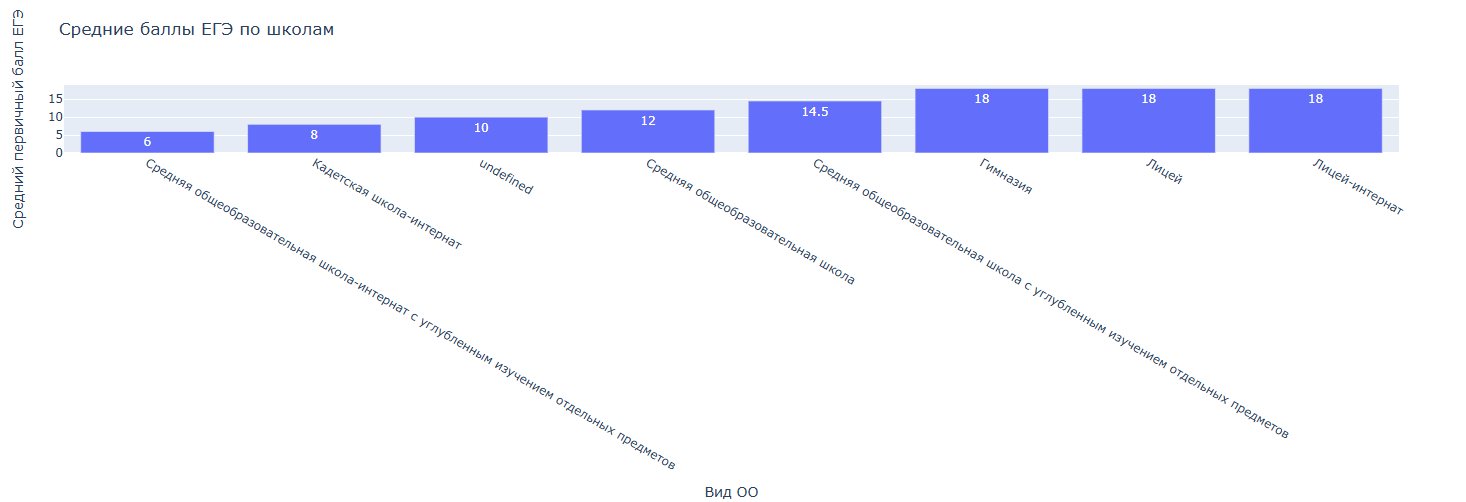


Рис. 16 График средних первичных баллов ЕГЭ по школам

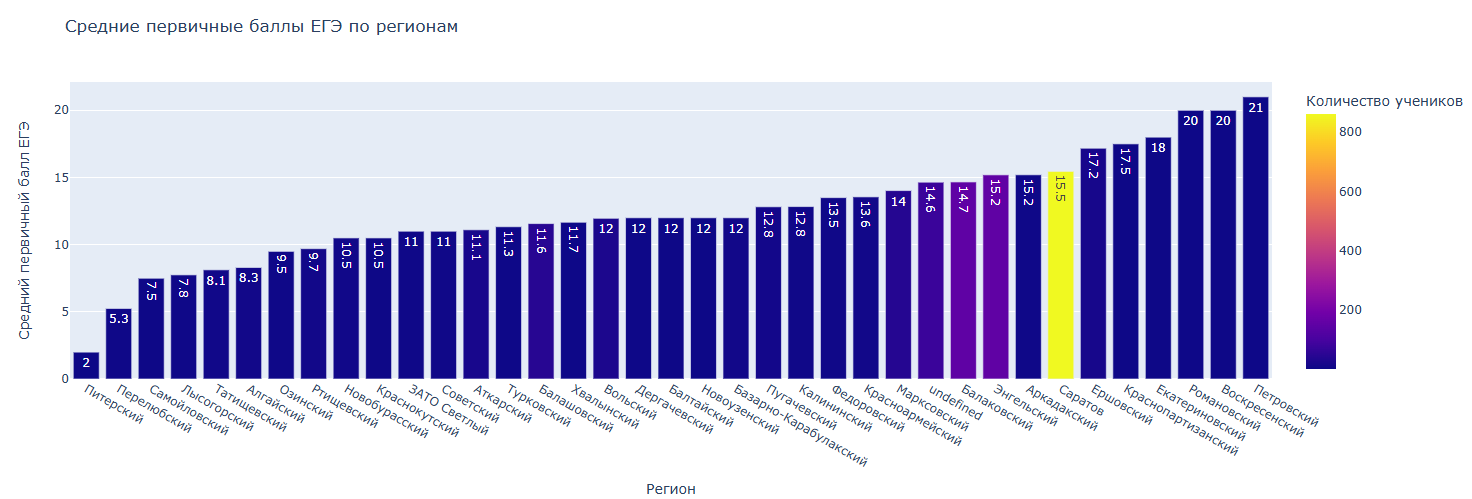
График на рисунке 17 показывает, что Саратов выделяется не только большим числом учеников, сдававших ЕГЭ, но и высокими баллами, что свидетельствует о лучшей подготовке в региональном центре. Также хорошие результаты у Балаковского и Энгельского региона, где сдавало много учеников. Балашовский район, с 44 сдававшими и средним баллом 11.6, показывает более низкий уровень подготовки. Высокие результаты в регионах с малым числом сдававших говорят о самостоятельной подготовке учеников, но не отражают общего качества образования в этих регионах.

Рис. 17 График средних первичных баллов ЕГЭ по регионам

График на рисунке 18 подчеркивает тенденцию к более высокому уровню подготовки в специализированных школах, которые часто имеют более строгий отбор учащихся и фокусируются на углубленном изучении предметов. Такие учреждения, как правило, предоставляют качественное образование и создают условия для достижения высоких результатов на ЕГЭ.

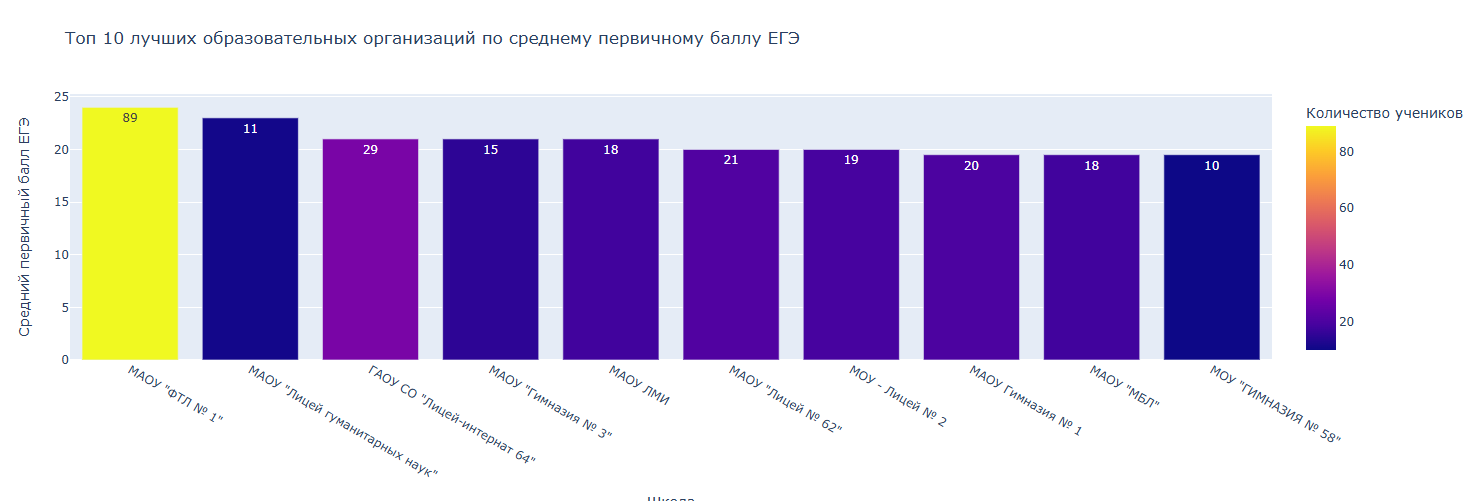


Рис. 18 График топ 10 лучших образовательных организаций по среднему первичному баллу ЕГЭ

График на рисунке 19 подчеркивает, что среди худших по результатам ЕГЭ в основном находятся общеобразовательные школы, но встречаются и лицеи, что может свидетельствовать о разной эффективности подготовки к ЕГЭ даже в специализированных учебных заведениях.



Рис. 19 График топ 10 худших образовательных организаций по среднему первичному баллу ЕГЭ

Также были построены два аналогичных графика для вторичных баллов, которые показали те же образовательные организации. Это свидетельствует о том, что перевод первичных баллов в вторичные был выполнен корректно, и таблица перевода баллов была составлена правильно.

4. Профили отличников, хорошистов и троечников.

Будут рассмотрены графики, которые показывает какие задания выполняют отличники, хорошисты и троечники и их количество в каждом регионе.

4.1 ОГЭ

4.1.1 Отличники

График на рисунке 20 демонстрирует, что отличники успешно справились почти со всеми заданиями, однако задания под номерами 6 и 12 оказались для них самыми сложными. Это может указывать на необходимость их пересмотра в будущем.



Рис. 20 График количества отличников, выполнивших каждое задание

По графику на рисунке 21 понятно, что самые тяжелые задания в экзамене были хорошо сделаны и выполнены практически всеми отличниками.

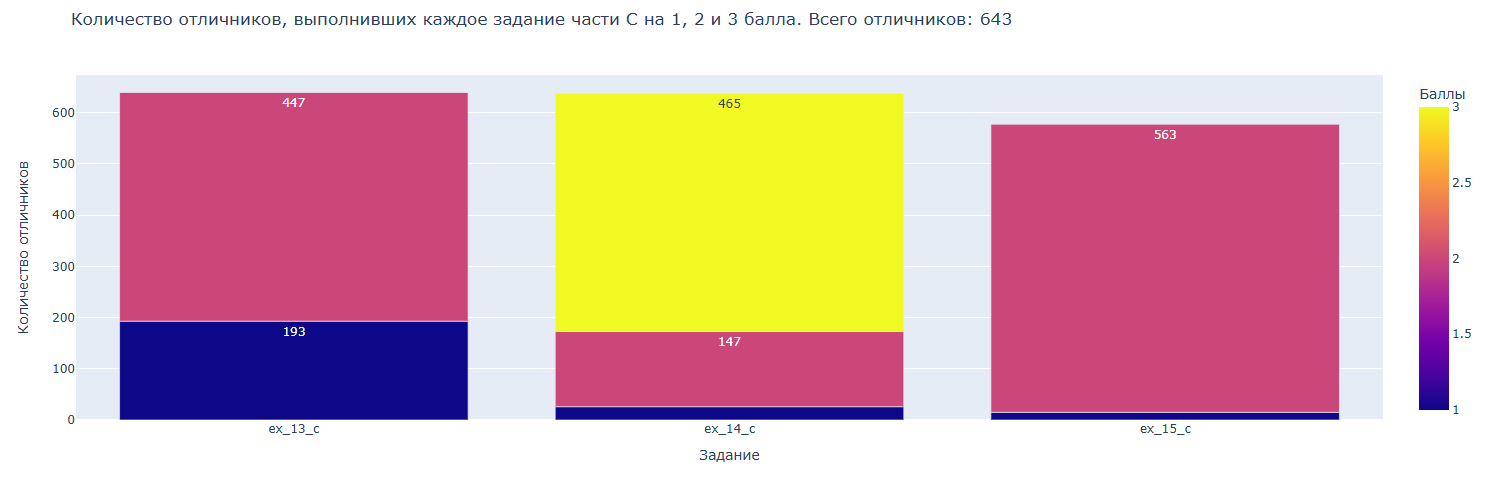


Рис. 21 График количества отличников, выполнивших каждое задание части C на 1, 2 и 3 балла

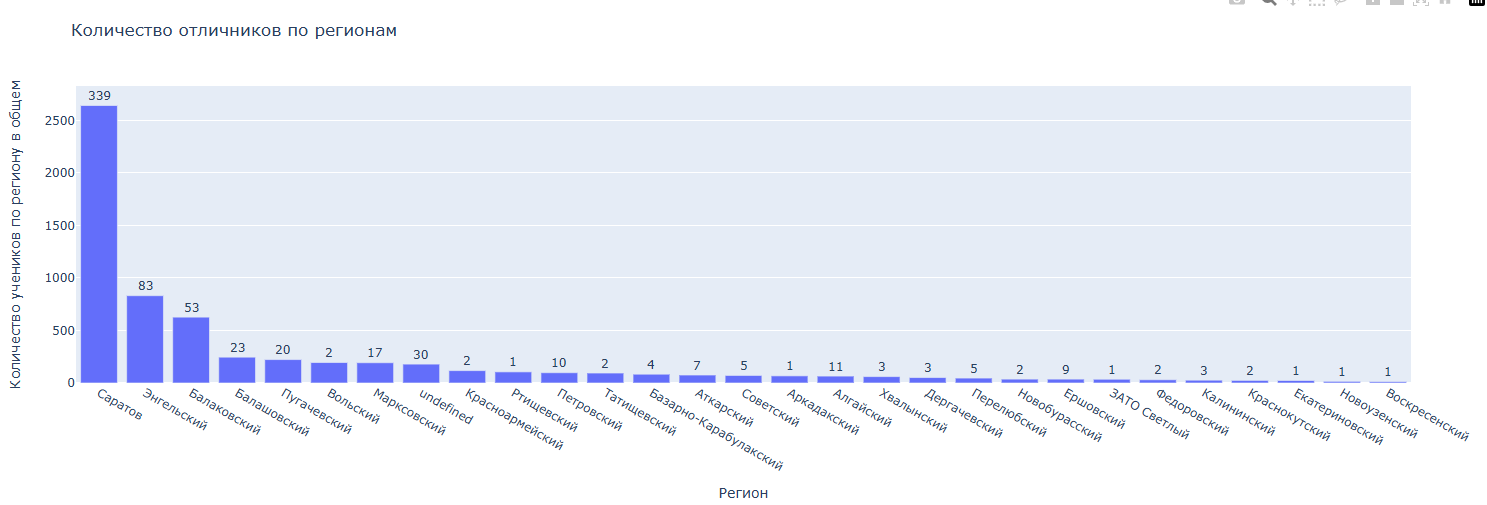
График на рисунке 22 показывает, что наибольшее количество отличников ожидается в региональном центре и крупных городах области, тогда как в малонаселённых пунктах количество отличников будет значительно ниже.

Рис. 22 График количества отличников по регионам

4.1.2 Хорошисты

График на рисунке 23 показывает, что хорошисты успешно справились со многими заданиями, однако, как и для отличников, задания под номерами 6 и 12 из первой части оказались для них самыми сложными, что подтверждает их высокую степень сложности. Также не все ученики смогли выполнить трудные задания из второй части экзамена, возможно, не успев дойти до них или не подготовившись должным образом. Тем не менее, значительное количество учеников все же решали их.

Рис. 23 График количества хорошистов, выполнивших каждое задание

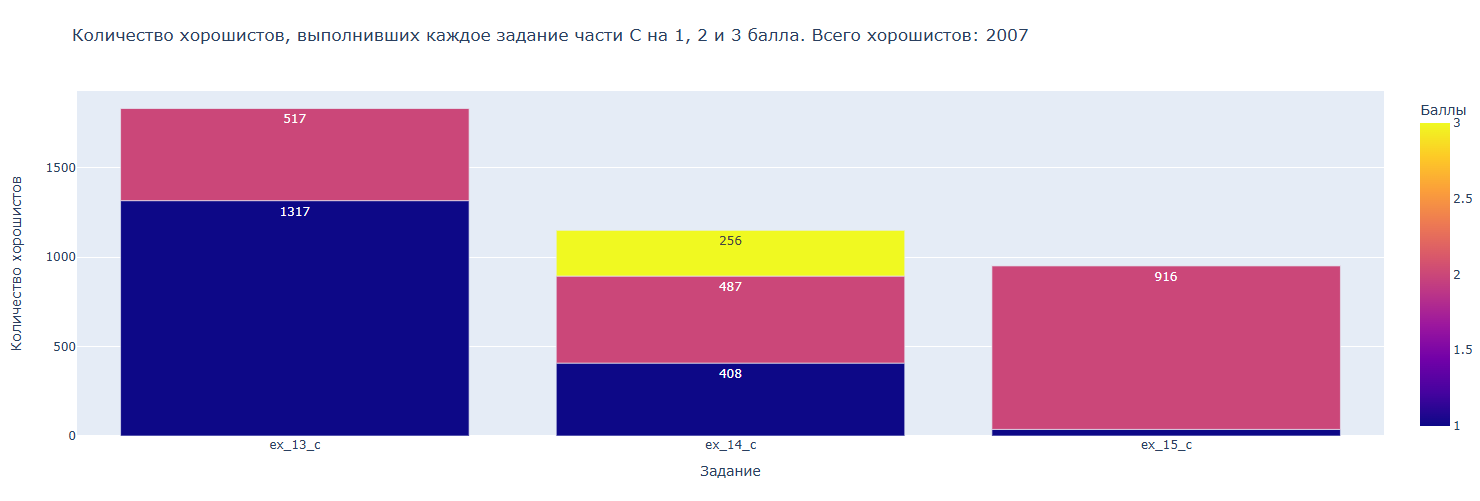
График на рисунке 24 показывает, что хорошисты справлялись с заданиями второй части хуже, чем отличники, но всё же выполняли их.

Рис. 24 График количества хорошистов, выполнивших каждое задание части C на 1, 2 и 3 балла

Также был построен график, на котором видно, что количество хорошистов естественно превышает количество отличников.

4.1.3 Троечники

График на рисунке 25 показывает, что троечники также плохо справились с заданиями 6 и 12, что вновь подтверждает их сложность и возможную необходимость пересмотра этих заданий в будущем. Также видно, что задание 13 во второй части хорошо выполняют все три группы учеников, что указывает на его относительную простоту по сравнению с другими заданиями второй части и может служить основанием для его пересмотра в будущем.

Рис. 25 График количества троечников, выполнивших каждое задание

Также был представлен график, показывающий количество троечников, выполнивших каждое задание части C на 1, 2 и 3 балла. Из графика было видно, что задание 13 выполнила треть троечников, что является довольно высоким показателем и подтверждает сказанное выше. Кроме того, имеется график, отображающий количество троечников по регионам, и их количество превышает количество отличников и хорошистов, что также является закономерным.

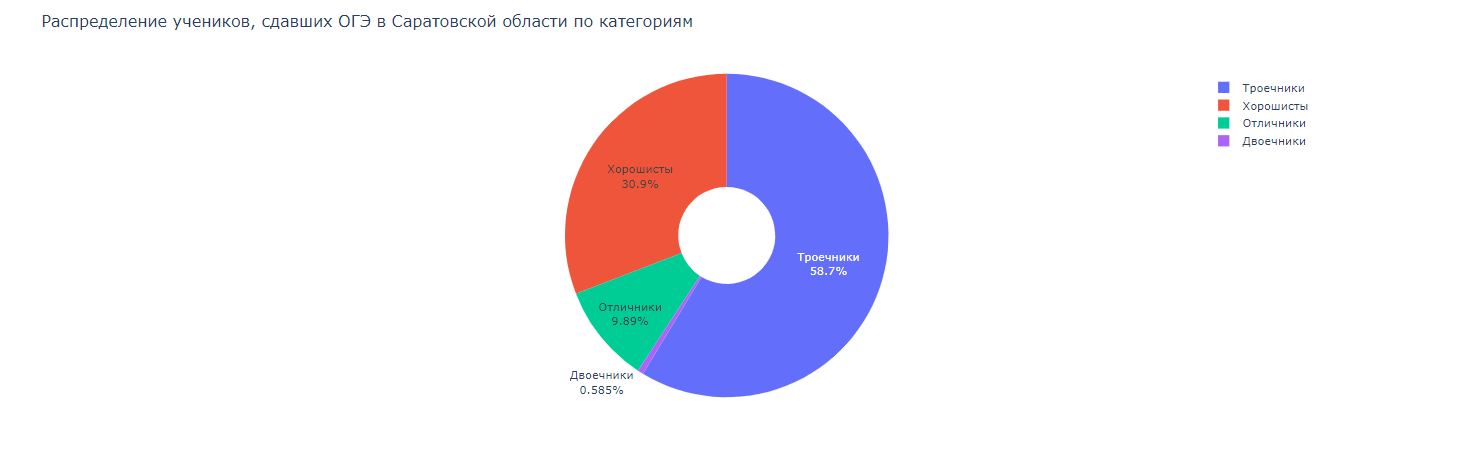
В итоге на графике на рисунке 26 показано, что большинство учеников, сдавших ОГЭ, относятся к категории троечников. Это может свидетельствовать о том, что они выбирают данный предмет, не осознавая его реальной сложности, или недостаточно готовятся к экзамену, считая его лёгким. Также возможно, что учащиеся предполагают успешную сдачу экзамена по информатике, исходя из частого использования компьютерной техники в повседневной жизни.

Рис. 26 График распределения учеников, сдавших ОГЭ в Саратовской области

4.2 ЕГЭ

4.2.1 Отличники

График на рисунке 27 показывает, что отличники успешно справились почти со всеми заданиями, однако задание 27 вызвало у них значительные трудности, что указывает на его высокую сложность и возможную необходимость пересмотра в будущем. Также заметно, что задания 7, 8 и 9 из первой части выполнило меньшее количество учеников по сравнению с другими заданиями из первой части, что также указывает на возможный пересмотр этих заданий в будущем. 

Рис. 27 График количества отличников, выполнивших каждое задание

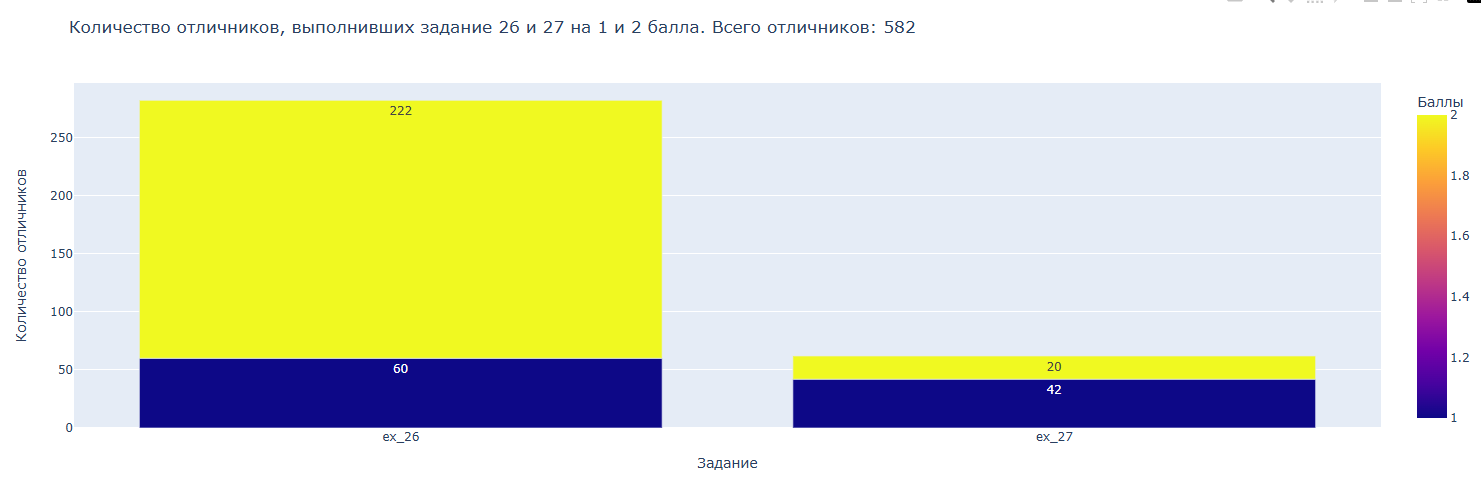
График на рисунке 28 показывает, что задание 27 является довольно сложным, так как из 582 учеников только 42 смогли решить его на 1 балл, и лишь 20 — на 2 балла.

Рис. 28 График количества отличников, выполнивших задание 26 и 27 на 1 и 2 балла

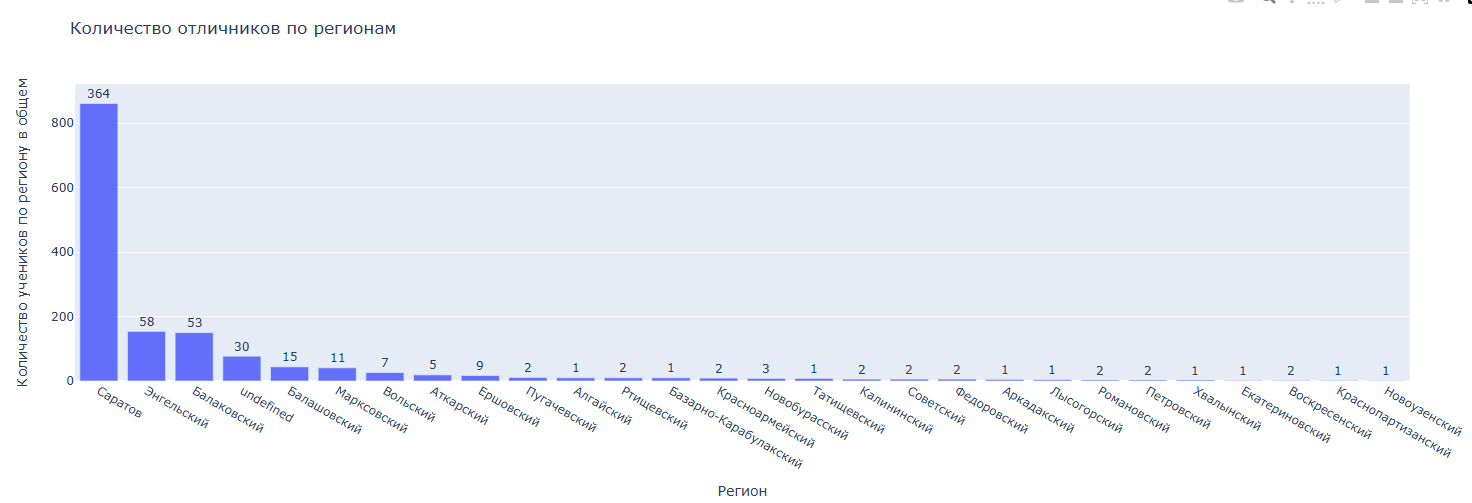
График на рисунке 29 показывает, что наибольшее количество отличников по ЕГЭ, как и по ОГЭ, сосредоточено в региональном центре и крупных городах области. В малонаселённых пунктах учеников меньше, и уровень подготовки зачастую ниже.

Рис. 29 График количества отличников, выполнивших задание 26 и 27 на 1 и 2 балла

4.2.2 Хорошисты

График на рисунке 30 показывает, что учащиеся с оценкой "хорошо" успешно решили большое количество задач. Однако задание 27 вызвало у них значительные трудности, что указывает на его высокую сложность и возможную необходимость пересмотра в будущем. Также видно, что задания 7, 8 и 9 из первой части выполнило меньше учеников по сравнению с другими заданиями этого раздела, что также может свидетельствовать о необходимости их пересмотра.



Рис. 30 График количества хорошистов, выполнивших каждое задание

На графике 31 видно, что задание 27 смогли решить только два учащихся с оценкой "хорошо", и оба получили за него лишь 1 балл, что подтверждает его чрезвычайную сложность.

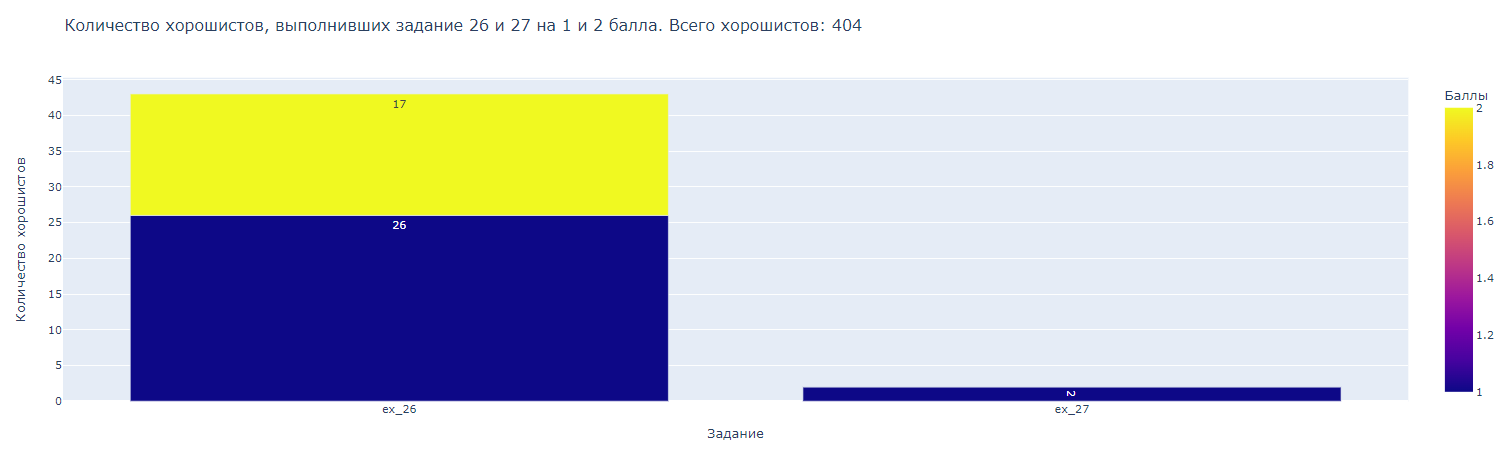


Рис. 31 График количества хорошистов, выполнивших задание 26 и 27 на 1 и 2 балла

На графике на рисунке 32 видно, что количество хорошистов меньше, чем отличников, что свидетельствует о том, что учащиеся основательно готовятся по этому предмету и стремятся к высоким результатам. Это также указывает на то, что предмет не является обязательным.

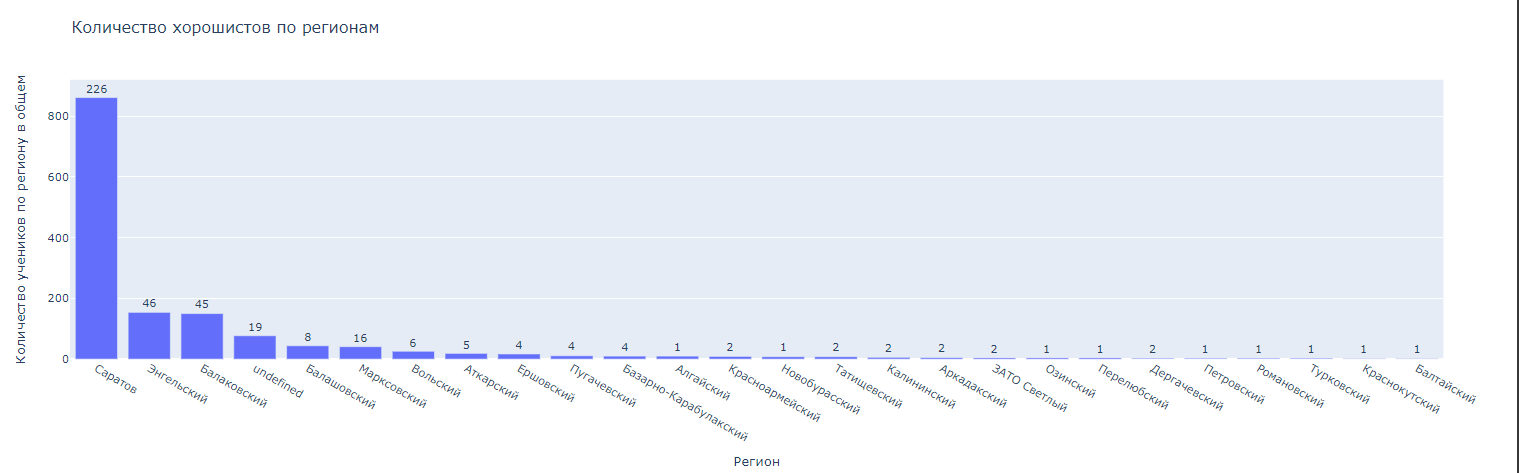


Рис. 32 График количества хорошистов по регионам

4.2.3 Троечники

На графике на рисунке 33 видно, что задания ближе ко второй части оказываются слишком сложными для троечников, либо они просто к ним не готовятся. Однако некоторые задания из первой части решены практически всеми троечниками. Также заметно, что несмотря на то, что задания 7, 8 и 9 кажутся трудными для хорошистов и отличников на других графиках, некоторые троечники все же смогли их решить. Это указывает на то, что данные задания не являются сверхсложными, а скорее средней сложности.

Рис. 33 График количества троечников, выполнивших каждое задание

Нет смысла смотреть график количества троечников, выполнивших 26 и 27 задания на 1 и 2 балла, так как 27 задания троечники просто не решали, либо у них неверные ответы, а 26 смогли решить только 6 учеников.

Также на графике на рисунке 34, видно, что их меньше, чем хорошистов и отличников. Это также указывает на то, что информатику выбирают в основном те ученики, которые заинтересованы в успешной сдаче этого предмета.

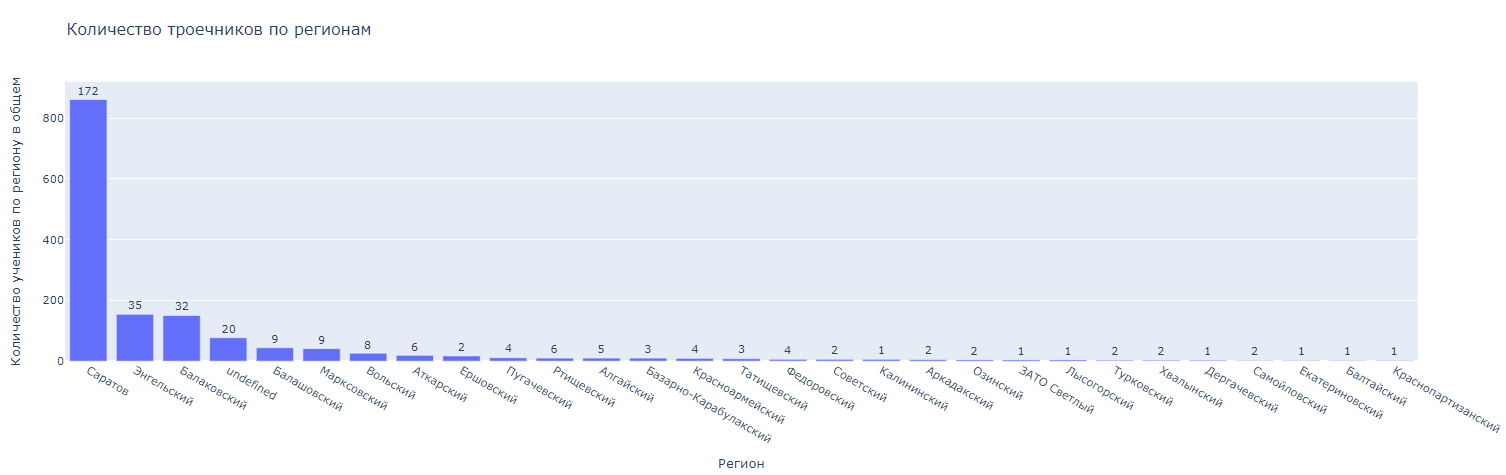


Рис. 34 График количества хорошистов по регионам

В конце на графике на рисунке 35 видно, что количество отличников наибольшее, что вновь подтверждает: большинство учеников выбирают информатику исходя из своих интересов и целенаправленно готовятся к высоким результатам.

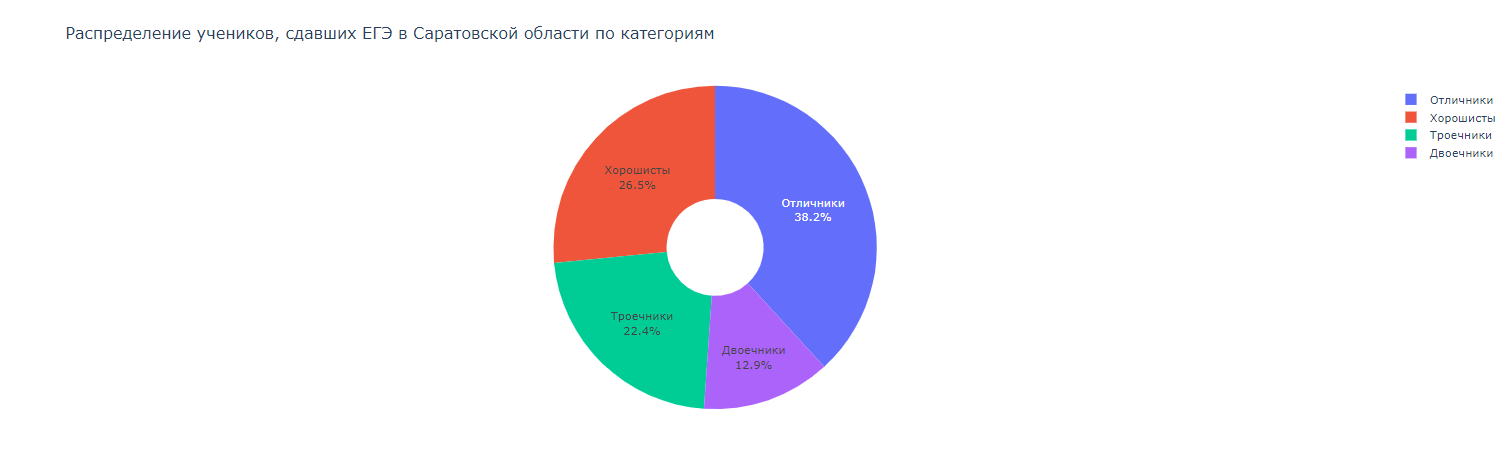


Рис. 35 График распределения учеников, сдавших ЕГЭ по информатике в Саратовской области

5. Гипотезы

5.1 Гипотеза 1

На графиках на рисунке 34 видно, что лицеи и гимназии, как правило, обеспечивают более высокие и стабильные баллы, что может свидетельствовать о более высоком уровне подготовки, качественном преподавании и отборе мотивированных учащихся. В то же время средние общеобразовательные школы демонстрируют больший разброс баллов, что указывает на разнообразие уровня подготовки учеников. Школы-интернаты, специальные школы и кадетские школы-интернаты имеют более низкие медианные баллы и узкие распределения, что может отражать либо специфику контингента учащихся, либо ограничения в учебных программах и ресурсах.

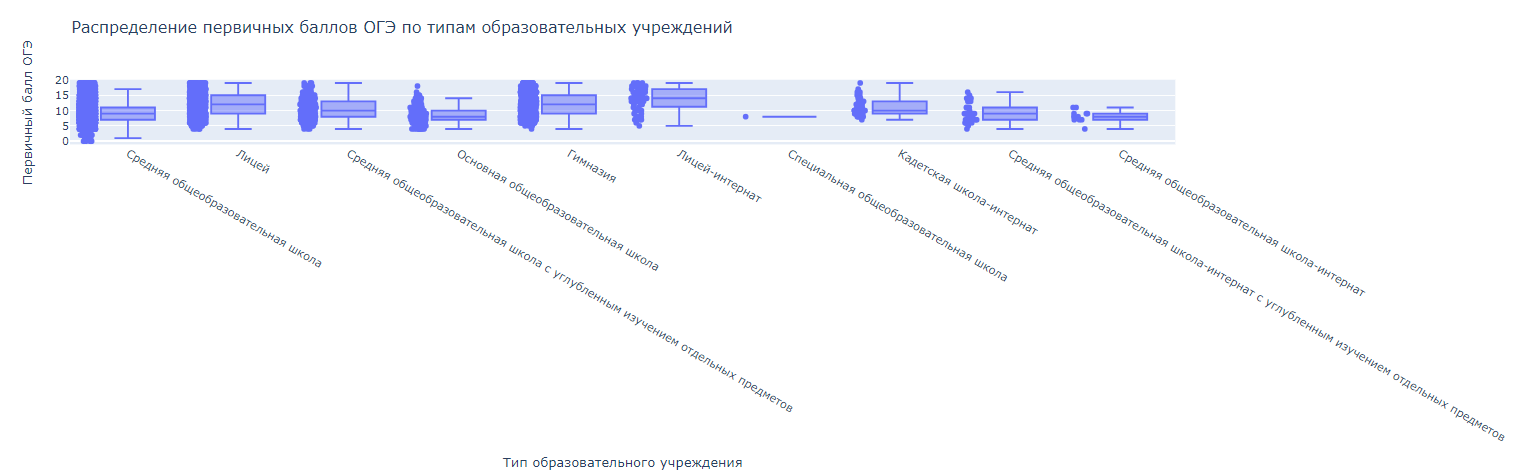


Рис. 36 График распределения первичных баллов ОГЭ по типам образовательных учреждений

На графике на рисунке 35 видно, что лицеи и гимназии достигают более высоких медианных результатов с меньшим разбросом баллов, что свидетельствует о высоком уровне подготовки учеников и качественном преподавании. В то же время средние общеобразовательные школы имеют более широкий разброс баллов, указывая на разный уровень подготовки учащихся. Школы-интернаты и кадетские школы показывают более низкие медианные баллы и узкие распределения, что может отражать ограниченные ресурсы и специфику учебного процесса. Наличие выбросов в некоторых категориях указывает на аномальные результаты, требующие дополнительного анализа.

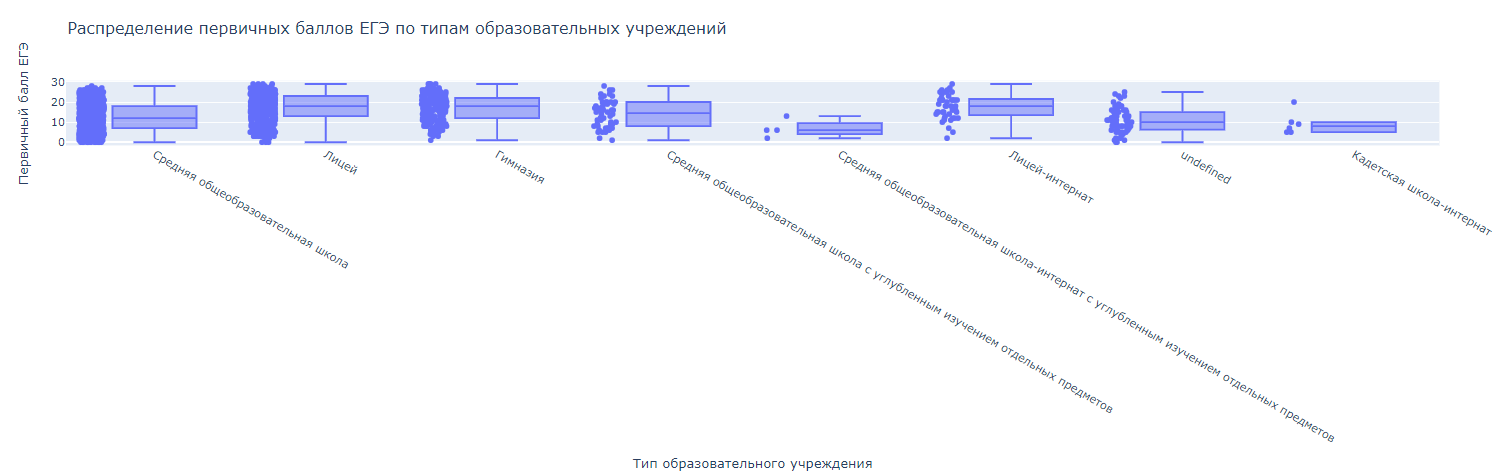


Рис. 37 График распределения первичных баллов ЕГЭ по типам образовательных учреждений

Вывод: по двум графикам, показывающим распределение первичных баллов ОГЭ и ЕГЭ по типам образовательных учреждений, можно сделать вывод, что тип учебного заведения существенно влияет на результаты учеников, что доказывает гипотезу, что школы с разным типом (гимназии, лицеи, средние общеобразовательные школы и т.п.) показывают различия в результатах экзаменов.

5.2 Гипотеза 2

На графике на рисунке 36 понятно, что лицеи и гимназии имеют более высокий средний первичный балл ЕГЭ (около 12), чем обычные школы (около 9). Это может свидетельствовать о более высоком уровне подготовки учеников в лицеях и гимназиях по сравнению с обычными школами, что, в свою очередь, может быть связано с более высоким уровнем учебных программ или более интенсивной подготовкой учеников.

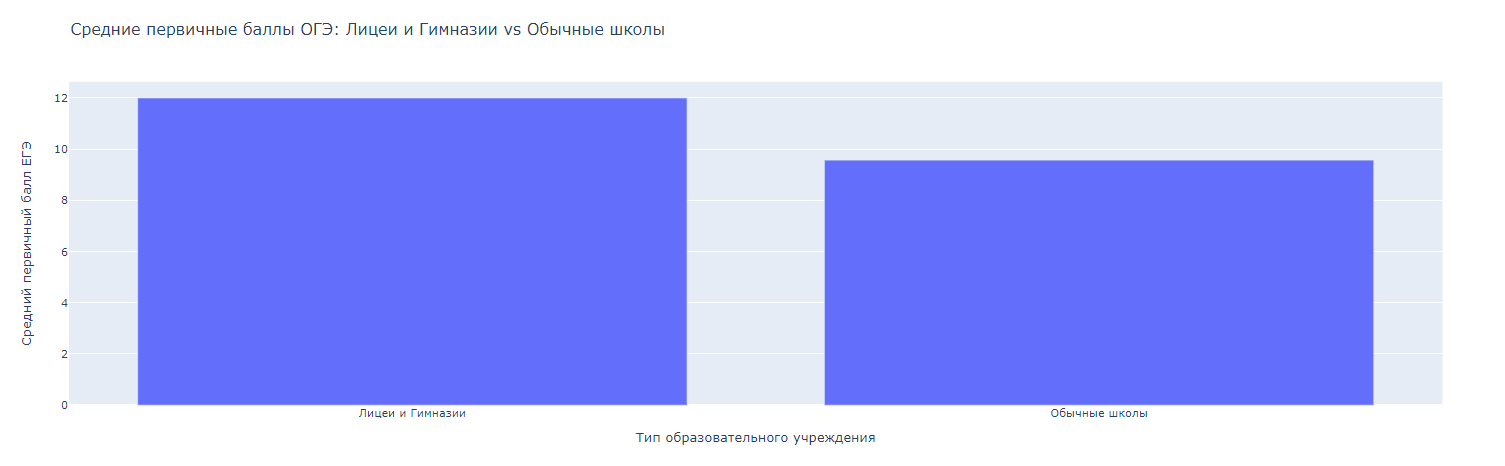


Рис. 38 График сравнения средних первичных баллов ОГЭ между лицеями и гимназиями и обычными школами

На графике на рисунке 37 понятно, что лицеи и гимназии демонстрируют более высокий средний первичный балл ЕГЭ, достигающий 18 баллов, тогда как в обычных школах этот показатель составляет примерно 14 баллов. Это может свидетельствовать о более высоком уровне подготовки к экзамену в лицеях и гимназиях, что может быть связано с более строгими учебными программами, квалифицированными преподавателями или дополнительной подготовкой учащихся.

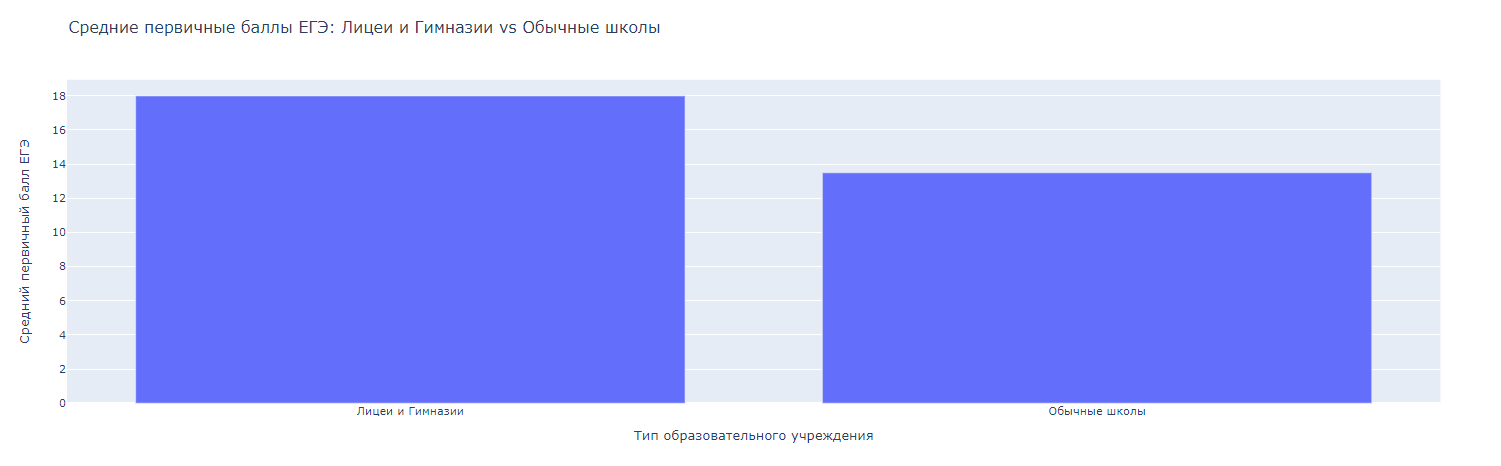


Рис. 39 График сравнения средних первичных баллов ЕГЭ между лицеями и гимназиями и обычными школами

Вывод: анализ двух графиков показывает, что образовательные учреждения с углубленной программой обучения (лицеи и гимназии) обеспечивают более высокие результаты учеников по сравнению с обычными школами на всех уровнях экзаменов, что доказывает гипотезу, которая предполагала, что лицеи и гимназии имеют лучшие средние результаты по сравнению с обычными школами.

5.3 Гипотеза 3

На графике на рисунке 38 видно, что о между двумя переменными наблюдается слабая отрицательная корреляция: с увеличением расстояния до Саратова результаты экзаменов слегка снижаются. Однако, эта тенденция не выражена явно, что можно увидеть по разбросу данных, который достаточно велик. Это указывает на то, что расстояние до Саратова не является основным фактором, влияющим на результаты ОГЭ.

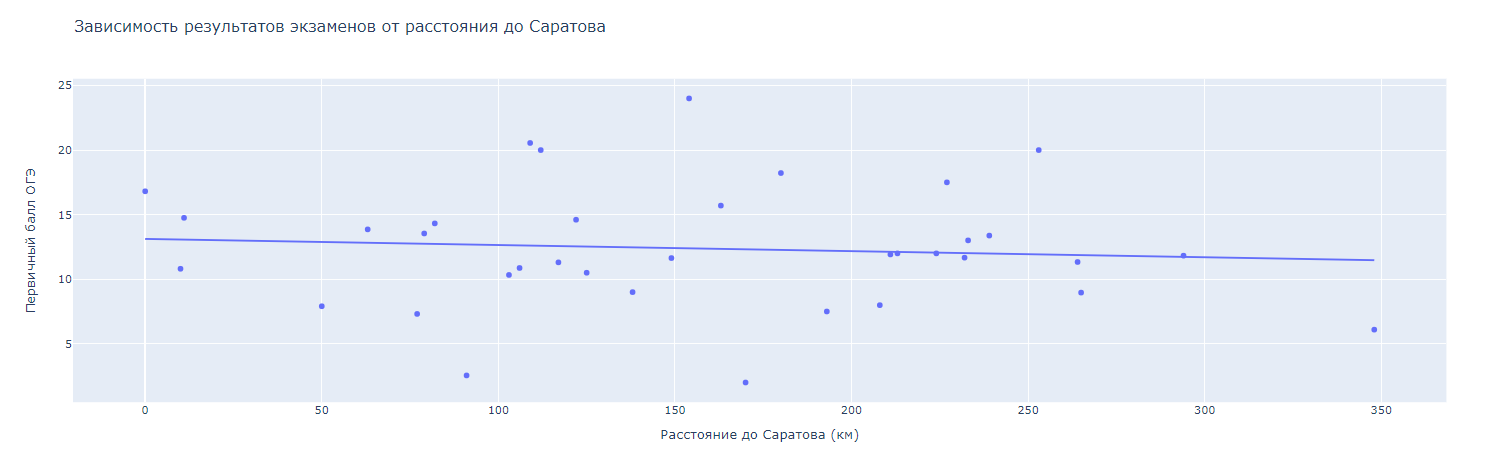


Рис. 40 График зависимости результатов ОГЭ от расстояния до Саратова

На графике на рисунке 39 становится также понятно, что видна слабая отрицательная корреляция, что означает небольшое снижение среднего балла ЕГЭ по мере увеличения расстояния до Саратова. Однако, как и в первом графике, разброс данных значительный, что свидетельствует о том, что расстояние до Саратова не является ключевым фактором, влияющим на результаты экзаменов, и другие факторы могут играть более значимую роль.

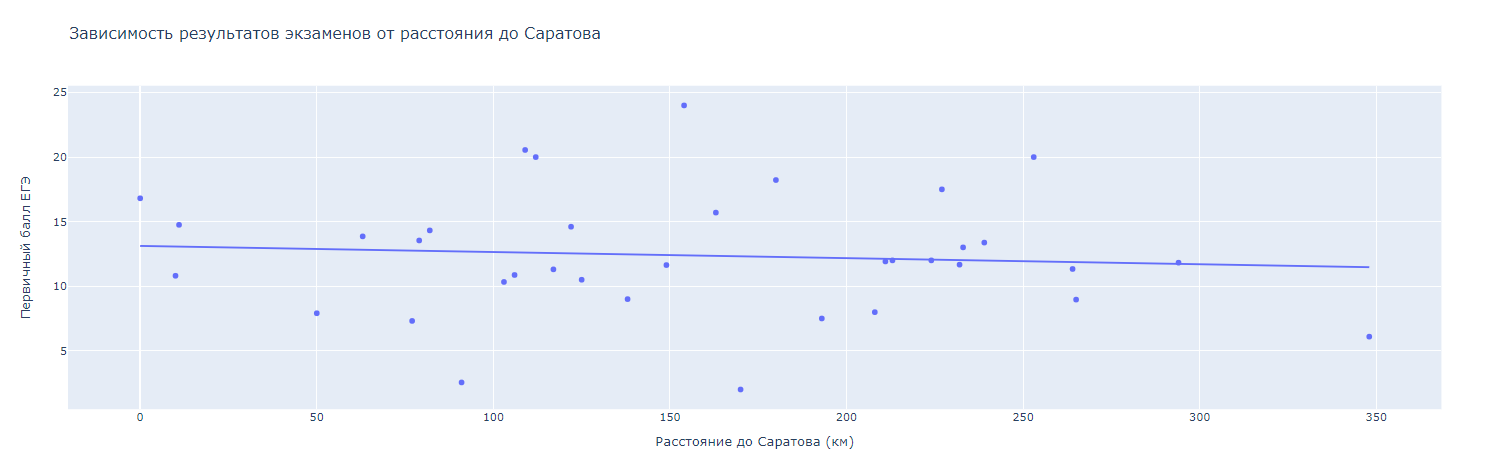


Рис. 41 График зависимости результатов ЕГЭ от расстояния до Саратова

Вывод: два графика указывают на то, что расстояние до Саратова не является определяющим фактором, влияющим на результаты экзаменов. Вероятно, на успеваемость учащихся влияют другие факторы. Таким образом, хотя существует слабая тенденция к снижению результатов с увеличением расстояния до Саратова, она не является статистически значимой и требует более глубокого анализа с учётом других переменных, что опровергает гипотезу, что результаты экзаменов (ОГЭ и ЕГЭ) зависят от расстояния школы до Саратова.

# **ЗАКЛЮЧЕНИЕ**

В ходе практики были изучены основные библиотеки языка программирования Python, предназначенные для предобработки данных, такие как Random, Math, NumPy, Pandas, Matplotlib, Plotly. Было решено множество задач с использованием данных библиотек, что позволило на практике закрепить полученные теоретические знания.

Практическая работа включала изучение встроенных модулей языка Python и применение библиотек для выполнения задач обработки данных. Особое внимание было уделено исследованию эффективности выполнения различных алгоритмов на разных размерностях входных данных. В частности, проводилось сравнение времени выполнения таких операций, как сортировка, поиск элементов и обработка данных, с использованием как встроенных возможностей Python, так и специализированных библиотек (например, NumPy и Pandas).

В качестве финального задания было проведено исследование результатов экзаменов (ОГЭ и ЕГЭ) по информатике с использованием библиотек Pandas и Plotly. В ходе анализа были выдвинуты гипотезы и проведена их проверка, что позволило выявить закономерности в данных и сделать обоснованные выводы о зависимости результатов от типа образовательного учреждения и других факторов.

Таким образом, практика позволила приобрести ценные навыки работы с данными в Python, научиться анализировать и визуализировать большие объемы информации, а также эффективно использовать различные библиотеки для решения реальных задач анализа данных.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Survey Shows Data Scientists Spend Time Cleaning Data [Электронный ресурс] : [сайт] - URL: https://www.dataversity.net/survey-shows-data-scientists-spend-time-cleaning-data/ (дата обращения: 07.07.2024)

2. Random — Generate pseudo-random numbers [Электронный ресурс] : [сайт] - URL: https://docs.python.org/3/library/random.html (дата обращения: 08.07.2024)

3. Math — Mathematical functions [Электронный ресурс] : [сайт] - URL: https://docs.python.org/3/library/math.html (дата обращения: 09.07.2024)

4. Идрис И. NumPy Beginner's Guide / И. Идрис; 3-е изд. — Packt Publishing, 2015. — 262 с. (дата обращения: 09.07.2024).

5. МакКинни У. Python и анализ данных / 2-е изд. пер. с анг. А. А. Слинкина – ДМК Пресс, 2020. – 540 с. (дата обращения: 09.07.2024)

6. Matplotlib — Using Matplotlib [Электронный ресурс] : [сайт] - URL: https://matplotlib.org/stable/users/index.html (дата обращения: 09.07.2024)

7. Plotly Open Source Graphing Library for Python [Электронный ресурс] : [сайт] - URL: https://plotly.com/python/ (дата обращения: 22.07.2024)

# **ПРИЛОЖЕНИЕ А Модуль Random**

26 вариант

# Подключение библиотек random и string

import random

import string

def generate\_id(n=5, digit=7):

    # Определяем количество цифр 'digit', которые будут включены в идентификатор.

    # Количество выбирается случайным образом от 1 до 3.

    count\_digit = random.randint(1, 3)

    # Генерируем уникальные индексы для размещения цифры 'digit' в пределах длины идентификатора n.

    index\_digits = set(random.randint(0, n - 1) for i in range(count\_digit))

    # Инициализируем пустой список для хранения частей идентификатора.

    res = []

    # Проходимся по всем позициям от 0 до n-1.

    for i in range(n):

        if i in index\_digits:

            # Если текущий индекс находится в index\_digits, добавляем 'digit' в результат.

            res += [digit]

        else:

            # В противном случае добавляем случайную цифру от 0 до 9, исключая 'digit'.

            res += [random.choice([int(elem) for elem in string.digits if elem != str(digit)])]

    # Объединяем список цифр в строку.

    res = ''.join(map(str, res))

    # Возвращаем сгенерированный идентификатор.

    return res

def is\_vowel(char, previous\_char=None):

    # Определяем строку гласных букв.

    vowels = "aeiou"

    # Обрабатываем случай, когда текущий символ 'y'.

    if char == 'y':

        # Если предыдущий символ отсутствует или является согласной буквой,

        # считаем 'y' гласной.

        if previous\_char is None or previous\_char in "bcdfghjklmnpqrstvwxyz":

            return True

        else:

            # В противном случае считаем 'y' согласной.

            return False

    # Возвращаем True, если текущий символ является одной из гласных букв (кроме 'y').

    return char in vowels

def generate\_login(n=6, cons=3):

    # Бесконечный цикл для генерации логина, пока не будут выполнены условия

    while True:

        # Генерируем случайную последовательность из n маленьких английских букв

        login = ''.join(random.choices(string.ascii\_lowercase, k=n))

        # Считаем количество согласных в сгенерированной последовательности

        consonants\_count = sum(1 for i in range(n) if not is\_vowel(login[i], login[i-1] if i > 0 else None))

        # Если количество согласных равно заданному количеству (cons), возвращаем логин

        if consonants\_count == cons:

            return login

# Определяем список строковых представлений нечетных цифр от 1 до 9.

odd\_digits = [str(i) for i in range(1, 10, 2)]

def generate\_password(n=10):

    # Бесконечный цикл для генерации пароля, пока не будут выполнены условия

    while True:

        # Создаем список всех возможных символов: большие и маленькие английские буквы, а также цифры.

        characters = list(string.ascii\_letters + string.digits)

        # Генерируем случайную последовательность из n символов без повторений.

        password = ''.join(random.sample(characters, n))

        # Проверяем, содержит ли пароль хотя бы одну большую букву и хотя бы одну нечетную цифру.

        if any(c.isupper() for c in password) and any(c in odd\_digits for c in password):

            # Если условия выполнены, возвращаем сгенерированный пароль.

            return password

def generate\_user(n=3):

    # Инициализируем множества для хранения уникальных идентификаторов, логинов и паролей.

    existing\_ids = set()

    existing\_logins = set()

    existing\_passwords = set()

    # Инициализируем список для хранения троек (id, логин, пароль).

    tuples\_list = []

    # Цикл для генерации n пользователей.

    for \_ in range(n):

        # Генерируем уникальный идентификатор.

        while True:

            id = generate\_id()

            if id not in existing\_ids:

                # Если id уникален, добавляем его в множество existing\_ids и выходим из цикла.

                existing\_ids.add(id)

                break

        # Генерируем уникальный логин.

        while True:

            login = generate\_login()

            if login not in existing\_logins:

                # Если логин уникален, добавляем его в множество existing\_logins и выходим из цикла.

                existing\_logins.add(login)

                break

        # Генерируем уникальный пароль.

        while True:

            password = generate\_password()

            if password not in existing\_passwords:

                # Если пароль уникален, добавляем его в множество existing\_passwords и выходим из цикла.

                existing\_passwords.add(password)

                break

        # Добавляем тройку (id, логин, пароль) в список tuples\_list.

        tuples\_list.append((id, login, password))

    # Возвращаем список сгенерированных троек.

    return tuples\_list

# Пример создания тройки пользователей

for elem in generate\_user():

    print(elem)

# **ПРИЛОЖЕНИЕ Б Измерение времени выполнения**

# Подключение библиотек random, string, collections

import random

import string

from collections import deque, Counter

def isprime\_n(n):

# Проверка на то, что число меньше или равно 1

if n <= 1:

return False # Числа меньше или равные 1 не являются простыми

# Перебор чисел от 2 до n-1

for i in range(2, n):

# Если n делится на i без остатка, значит n составное число

if n % i == 0:

return False # Возвращаем False, так как найден делитель

return True # Если не найдено ни одного делителя, возвращаем True

%%time

for i in range(100, 100000):

isprime\_n(i)

def isprime\_n\_2(n):

# Проверка на то, что число меньше или равно 1

if n <= 1:

return False # Числа меньше или равные 1 не являются простыми

# Перебор чисел от 2 до n // 2 + 1

for i in range(2, n // 2 + 1):

# Если n делится на i без остатка, значит n составное число

if n % i == 0:

return False # Возвращаем False, так как найден делитель

return True # Если не найдено ни одного делителя, возвращаем True

%%time

for i in range(100, 100000):

isprime\_n\_2(i)

def isprime\_sqrt\_n(n):

# Проверка на то, что число меньше или равно 1

if n <= 1:

return False # Числа меньше или равные 1 не являются простыми

# Перебор чисел от 2 до квадратного корня из n включительно

for i in range(2, int(n \*\* 0.5) + 1):

# Если n делится на i без остатка, значит n составное число

if n % i == 0:

return False # Возвращаем False, так как найден делитель

return True # Если не найдено ни одного делителя, возвращаем True

%%time

for i in range(100, 100000):

isprime\_sqrt\_n(i)

s1 = list(range(1000000)) # Создаём список из 1000000 чисел

%%time

item = 999999 # берем любой элемент

print(item in s1) # находим элемент в списке

s2 = set(range(1000000)) # Создаём множество из 1000000 чисел

%%time

item = 999999 # берем любой элемент

print(item in s2) # находим элемент во множестве

large\_list = [i % 10 for i in

range(1000000)] # Создаем большой список, где каждый элемент является остатком от деления индекса на 10

%%time

# Создаем множество уникальных элементов из large\_list

unique\_characters = set(large\_list)

# Инициализируем пустой словарь для подсчета количества каждого уникального элемента

character\_count = {}

# Проходим по каждому уникальному элементу и считаем его количество в large\_list

for char in unique\_characters:

character\_count[char] = large\_list.count(char) # Добавляем элемент и его количество в словарь character\_count

print(character\_count)

from collections import Counter

%%time

print(Counter(large\_list))

def bubble\_sort(arr):

# Получаем длину массива

n = len(arr)

# Проходим по всем элементам массива

for i in range(n):

# Внутренний цикл для сравнения и обмена элементов

for j in range(0, n - i - 1):

# Если текущий элемент больше следующего, меняем их местами

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j] # Перестановка элементов

def quicksort(arr):

# Базовый случай: если массив содержит 0 или 1 элемент, он уже отсортирован

if len(arr) <= 1:

return arr

# Выбираем опорный элемент (pivot) - элемент посередине массива

pivot = arr[len(arr) // 2]

# Подмассивы, содержащие элементы меньше, равные и больше опорного

left = [x for x in arr if x < pivot]

middle = [x for x in arr if x == pivot]

right = [x for x in arr if x > pivot]

# Рекурсивно сортируем подмассивы и объединяем их с опорным элементом

return quicksort(left) + middle + quicksort(right)

numbers = [random.randint(0, 10000) for \_ in range(10000)] # Генерация списка случайных чисел

%%time

sort\_list = bubble\_sort(numbers) # Сортировка пузырьком

%%time

sort\_list = quicksort(numbers) # Быстрая сортировка

%%time

sort\_list = sorted(numbers) # Встроенная сортировка

def bubble\_sort\_string(s):

# Преобразуем строку в список символов для сортировки

arr = list(s)

n = len(arr)

# Внешний цикл для прохождения по всем элементам списка

for i in range(n):

# Внутренний цикл для сравнения и обмена элементов

for j in range(0, n - i - 1):

# Если текущий элемент больше следующего, меняем их местами

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j] # Перестановка элементов

# Преобразуем отсортированный список обратно в строку

return ''.join(arr)

def quicksort\_string(s):

# Преобразуем строку в список символов для сортировки

arr = list(s)

# Базовый случай: если длина массива 1 или меньше, возвращаем строку как есть

if len(arr) <= 1:

return s

else:

# Выбираем опорный элемент (pivot) - элемент посередине массива

pivot = arr[len(arr) // 2]

# Создаем подмассивы:

# left - элементы меньше опорного

# middle - элементы равные опорному

# right - элементы больше опорного

left = [x for x in arr if x < pivot]

middle = [x for x in arr if x == pivot]

right = [x for x in arr if x > pivot]

# Рекурсивно сортируем подмассивы, объединяем их и возвращаем как строку

return ''.join(quicksort\_string(''.join(left)) + ''.join(middle) + quicksort\_string(''.join(right)))

symbols = ''.join(random.choice((string.ascii\_letters + string.digits) \* (i + 1)) for i in range(10000)) # Генерация случайной строки

%%time

sort\_symbols = bubble\_sort\_string(symbols) # Сортировка пузырьком для строки

%%time

sort\_symbols = quicksort\_string(symbols) # Быстрая сортировка для строки

%%time

sort\_symbols = sorted(symbols) # Встроенная сортировка для строки

class User:

def \_\_init\_\_(self, user\_id, username, password):

self.id = user\_id # Уникальный идентификатор пользователя

self.username = username # Имя пользователя

self.password = password # Пароль пользователя

def \_\_repr\_\_(self):

# Метод для удобного отображения информации о пользователе

return f"User(id={self.id}, username={self.username}, password={self.password})"

# Функция для создания списка случайных пользователей

def create\_random\_users(num\_users):

users = []

for \_ in range(num\_users):

# Генерация случайного id пользователя в диапазоне от 1000 до 9999

user\_id = random.randint(1000, 9999)

# Генерация случайного имени пользователя из 8 букв

username = ''.join(random.choices(string.ascii\_letters, k=8))

# Генерация случайного пароля из букв и цифр длиной 8 символов

password = ''.join(random.choices(string.ascii\_letters + string.digits, k=8))

# Добавление созданного пользователя в список

users.append(User(user\_id, username, password))

return users

def bubble\_sort\_users(users):

# Получаем количество пользователей в списке

n = len(users)

# Внешний цикл для прохождения по всем элементам списка

for i in range(n):

# Внутренний цикл для сравнения и обмена элементов

for j in range(0, n - i - 1):

# Сравниваем поле 'id' текущего пользователя с полем 'id' следующего пользователя

if users[j].id > users[j + 1].id:

# Если 'id' текущего пользователя больше, чем 'id' следующего, меняем их местами

users[j], users[j + 1] = users[j + 1], users[j]

# Возвращаем отсортированный список пользователей

return users

def quicksort\_users(users):

# Базовый случай: если список содержит 0 или 1 элемент, он уже отсортирован

if len(users) <= 1:

return users

else:

# Выбираем опорного пользователя (pivot) - элемент посередине списка

pivot = users[len(users) // 2]

# Создаем подсписки:

# left - пользователи с id меньше, чем у опорного

# middle - пользователи с id равным id опорного

# right - пользователи с id больше, чем у опорного

left = [x for x in users if x.id < pivot.id]

middle = [x for x in users if x.id == pivot.id]

right = [x for x in users if x.id > pivot.id]

# Рекурсивно сортируем подсписки и объединяем их

return quicksort\_users(left) + middle + quicksort\_users(right)

num\_users = 10000

random\_users = create\_random\_users(num\_users) # Создание случайных пользователей

%%time

sort\_users = bubble\_sort\_users(random\_users) # Сортировка пузырьком для пользователей

%%time

sort\_users = quicksort\_users(random\_users) # Быстрая сортировка для пользователей

%%time

sort\_users = sorted(random\_users, key=lambda user: user.id) # Встроенная сортировка для пользователей

# Класс Job для представления информации о вакансии

class Job:

def \_\_init\_\_(self, job\_id, title, description):

self.id = job\_id # Уникальный идентификатор вакансии

self.title = title # Название вакансии

self.description = description # Описание вакансии

def \_\_repr\_\_(self):

# Метод для удобного отображения информации о вакансии

return f"Job(id={self.id}, title={self.title}, description={self.description})"

# Функция для создания списка случайных вакансий

def create\_random\_jobs(num\_jobs):

jobs = []

for \_ in range(num\_jobs):

# Генерация случайного id вакансии в диапазоне от 1000 до 9999

job\_id = random.randint(1000, 9999)

# Генерация случайного названия вакансии из 12 символов (буквы и пробелы)

title = ''.join(random.choices(string.ascii\_letters + ' ', k=12)).strip()

# Генерация случайного описания вакансии из 20 символов (буквы и пробелы)

description = ''.join(random.choices(string.ascii\_letters + ' ', k=20)).strip()

# Добавление созданной вакансии в список

jobs.append(Job(job\_id, title, description))

return jobs

def bubble\_sort\_jobs(jobs):

# Получаем количество вакансий в списке

n = len(jobs)

# Внешний цикл для прохождения по всем элементам списка

for i in range(n):

# Внутренний цикл для сравнения и обмена элементов

for j in range(0, n - i - 1):

# Сравниваем поле 'id' текущей вакансии с полем 'id' следующей вакансии

if jobs[j].id > jobs[j + 1].id:

# Если 'id' текущей вакансии больше, чем 'id' следующей, меняем их местами

jobs[j], jobs[j + 1] = jobs[j + 1], jobs[j]

# Возвращаем отсортированный список вакансий

return jobs

def quicksort\_jobs(jobs):

# Базовый случай: если список содержит 0 или 1 элемент, он уже отсортирован

if len(jobs) <= 1:

return jobs

else:

# Выбираем опорный элемент (pivot) - вакансия посередине списка

pivot = jobs[len(jobs) // 2]

# Создаем подсписки:

# left - вакансии с id меньше, чем у опорной

# middle - вакансии с id равным id опорной

# right - вакансии с id больше, чем у опорной

left = [x for x in jobs if x.id < pivot.id]

middle = [x for x in jobs if x.id == pivot.id]

right = [x for x in jobs if x.id > pivot.id]

# Рекурсивно сортируем подсписки и объединяем их

return quicksort\_jobs(left) + middle + quicksort\_jobs(right)

num\_jobs = 10000

random\_jobs = create\_random\_jobs(num\_jobs) # Создание случайных вакансий

%%time

sort\_jobs = bubble\_sort\_jobs(random\_jobs) # Сортировка пузырьком для вакансий

%%time

sort\_jobs = quicksort\_jobs(random\_jobs) # Быстрая сортировка для вакансий

%%time

sort\_jobs = sorted(random\_jobs, key=lambda job: job.id) # Встроенная сортировка для вакансий

# Экспериментальная проверка: подходит ли коллекция list как основа для абстрактных структур данных стек и очередь.

def stack\_push\_list(stack):

stack.append(random.randint(1000, 9999)) # Добавление элемента в конец списка

def stack\_pop\_list(stack):

if stack:

stack.pop() # Удаление последнего элемента списка (элемент верхушки стека)

def stack\_push\_deque(stack):

stack.append(random.randint(1000, 9999)) # Добавление элемента в конец дека

def stack\_pop\_deque(stack):

if stack:

stack.pop() # Удаление последнего элемента дека (элемент верхушки стека)

def queue\_push\_list(queue):

queue.append(random.randint(1000, 9999)) # Добавление элемента в конец списка (очередь)

def queue\_pop\_list(queue):

if queue:

queue.pop(0) # Удаление первого элемента списка (элемент начала очереди)

def queue\_push\_deque(queue):

queue.append(random.randint(1000, 9999)) # Добавление элемента в конец дека (очередь)

def queue\_pop\_deque(queue):

if queue:

queue.popleft() # Удаление первого элемента дека (элемент начала очереди)

sizes = [1000000]

stack\_example = []

%%time

for num\_operations in sizes:

for i in range(num\_operations):

stack\_push\_list(stack\_example) # Добавление элементов в стек (список)

%%time

for num\_operations in sizes:

for i in range(num\_operations):

stack\_pop\_list(stack\_example) # Удаление элементов из стека (список)

sizes = [1000000]

stack\_example = deque()

%%time

for num\_operations in sizes:

for i in range(num\_operations):

stack\_push\_deque(stack\_example) # Добавление элементов в стек (дек)

%%time

for num\_operations in sizes:

for i in range(num\_operations):

stack\_pop\_deque(stack\_example) # Удаление элементов из стека (дек)

sizes = [1000000]

queue\_example = []

%%time

for num\_operations in sizes:

for i in range(num\_operations):

queue\_push\_list(queue\_example) # Добавление элементов в очередь (список)

%%time

for num\_operations in sizes:

for i in range(num\_operations):

queue\_pop\_list(queue\_example) # Удаление элементов из очереди (список)

sizes = [1000000]

queue\_example = deque()

%%time

for num\_operations in sizes:

for i in range(num\_operations):

queue\_push\_deque(queue\_example) # Добавление элементов в очередь (дек)

%%time

for num\_operations in sizes:

for i in range(num\_operations):

queue\_pop\_deque(queue\_example) # Удаление элементов из очереди (дек)

# **ПРИЛОЖЕНИЕ В NumPy знакомство**

23 вариант

# Импорт используемой библиотеки numpy

import numpy as np

# Определяем размеры матрицы

n = 12 # Количество строк

m = 6 # Количество столбцов

# Создаем матрицу размером n x m с элементами от 1 до n\*m

matrix = np.arange(1, n \* m + 1).reshape(n, m)

# Доступ к элементу в 8-й строке и 6-м столбце (индексация с нуля)

matrix[7, 5]

# Получаем всю 4-ю строку матрицы

matrix[3]

# Получаем 6-ю строку в обратном порядке

matrix[5, ::-1]

# Изменяем форму матрицы на m x n (транспонирование)

matrix = matrix.reshape(m, n)

# Делим все элементы матрицы на число, введенное пользователем

matrix = matrix // int(input("Введите число для деления элементов матрицы: "))

# Вычисляем максимальное значение в каждой строке матрицы

matrix.max(axis=1)

# Находим минимальный элемент в первой строке матрицы

matrix[0].min()

# Создаем одномерный массив от 1 до 55 включительно

matrix\_1 = np.arange(1, 56)

# Считаем количество четных чисел в массиве

even\_numbers = len(matrix\_1[matrix\_1 % 2 == 0])

# Считаем количество нечетных чисел в массиве

odd\_numbers = len(matrix\_1[matrix\_1 % 2 != 0])

# Определяем общее количество чисел в массиве

all\_numbers = len(matrix\_1)

# Выводим процентное соотношение четных и нечетных чисел в массиве

print(f'Процент четных чисел: {(even\_numbers / all\_numbers):.0%}%')

print(f'Процент нечетных чисел: {(odd\_numbers / all\_numbers):.0%}%')

# Создаем матрицу размером n x m с элементами от 1 до n\*m

n = 12

m = 6

matrix = np.arange(1, n \* m + 1).reshape(n, m)

# Создаем логическую матрицу, где элементы находятся в диапазоне от 10 до 99 включительно

matrix\_2 = (np.absolute(matrix) >= 10) & (np.absolute(matrix) <= 99)

# Считаем количество элементов, удовлетворяющих условию (от 10 до 99)

matrix\_2.sum()

# **ПРИЛОЖЕНИЕ Г NumPy Обработка матрицы**

23 вариант

# Вводим размеры матрицы n и m от пользователя

n, m = int(input("Введите количество строк (n): ")), int(input("Введите количество столбцов (m): "))

# Создаем матрицу размером n x m с элементами от 1 до n\*m

A = np.arange(1, n \* m + 1).reshape(n, m)

# Вычисляем сумму абсолютных значений элементов по каждому столбцу

sum\_abs = np.absolute(A).sum(axis=0)

# Находим индекс столбца с наибольшей суммой абсолютных значений

sum\_abs\_index = np.argmax(sum\_abs)

# Находим минимальный элемент в столбце с наибольшей суммой абсолютных значений

min\_element = A[:, sum\_abs\_index].min()

# **ПРИЛОЖЕНИЕ Д NumPy вектор и матрица**

23 вариант

# Импорт используемой библиотеки numpy

import numpy as np

# Вводим значение n от пользователя

n = int(input("Введите значение n: "))

# Определяем размеры матриц A и B

A\_n = n # Количество строк для матрицы A

A\_m = 2 \* n # Количество столбцов для матрицы A

B\_n = n # Количество строк для матрицы B

B\_m = 2 # Количество столбцов для матрицы B

# Создаем матрицу B размером n x 2 с элементами от 1 до n \* 2

B = np.arange(1, n \* 2 + 1).reshape(B\_n, B\_m)

# Создаем матрицу A размером n x (2 \* n) с элементами от 1 до n \* 2 \* n

A = np.arange(1, n \* 2 \* n + 1).reshape(A\_n, A\_m)

# Разделяем матрицу A на две части: левую и правую

A\_left = A[:, :A\_m // 2] # Левая часть матрицы A (первые n столбцов)

A\_right = A[:, A\_m // 2:] # Правая часть матрицы A (последние n столбцов)

# Умножаем правую часть матрицы A на первый столбец матрицы B, приведенный к нужной форме

matrix\_new\_1 = A\_right \* B[:, 0].reshape(-1, 1)

# Умножаем левую часть матрицы A на второй столбец матрицы B в обратном порядке

matrix\_new\_2 = A\_left \* B[::-1, 1]

# Перемножаем две полученные матрицы

prod\_matrix = matrix\_new\_1 @ matrix\_new\_2

# Вычисляем определитель полученной произведенной матрицы

determinant = np.linalg.det(prod\_matrix)

# **ПРИЛОЖЕНИЕ Ё NumPy загадка**

# Импорт используемой библиотеки numpy

import numpy as np

def two\_dimensional(array, N):

# Создаем копию исходного массива, чтобы избежать изменения оригинала

array = np.copy(array)

# Создаем зеркальное отражение массива по горизонтали (отражаем столбцы)

# Соединяем оригинальный массив и его отражение по горизонтали

array\_right\_reflect = np.hstack((array, array[:, ::-1]))

# Создаем копию нового массива для дальнейшей работы

array = np.copy(array\_right\_reflect)

# Создаем зеркальное отражение массива по вертикали (отражаем строки)

# Соединяем отраженный массив и оригинальный по вертикали

array\_top\_reflect = np.vstack((array[::-1, :], array))

# Проверяем, если размер нового массива соответствует требуемому размеру N x N

if np.shape(array\_top\_reflect) == (N, N):

# Если да, то возвращаем его как результат

return array\_top\_reflect

# Если нет, рекурсивно вызываем функцию для нового массива, чтобы продолжить процесс

return two\_dimensional(array\_top\_reflect, N)

# Исходный массив 3x3

array\_start = np.array([['A', 'B', 'C'], ['B', 'C', 'A'], ['C', 'A', 'B']])

# Задаем желаемый размер итогового массива

N = 192

# Функция для расширения массива до нужного размера N x N

array = two\_dimensional(array\_start, N)

# Получаем значения x и y от пользователя и преобразуем их в целые числа

x, y = int(input()), int(input())

# Определяем размеры массива, используя функцию np.shape, которая возвращает кортеж (число строк, число столбцов)

array\_shape = np.shape(array)

# Проверяем, входят ли x и y в допустимые диапазоны индексов массива

# Индексы должны быть не больше, чем соответствующие размеры массива минус 1

if x <= array\_shape[0] - 1 and y <= array\_shape[1] - 1:

# Если индексы допустимы, выводим значение элемента массива с индексами (x, y)

print(array[x, y])

else:

# Если хотя бы один из индексов выходит за пределы размеров массива, выбрасываем исключение с сообщением об ошибке

raise ValueError(f'x: {x} и/или y: {y} не входят в размеры массива: {array\_shape}')

# **ПРИЛОЖЕНИЕ Ж NumPy сравнение эффективности**

# Импорт используемых библиотек numpy и random

import numpy as np

import random

def find\_median(matrix):

# Функция для нахождения медианы в матрице

# Преобразуем двумерный массив (матрицу) в одномерный список элементов

if not isinstance(matrix, np.ndarray): # Проверяем, является ли входная матрица объектом numpy ndarray

elements = [item for row in matrix for item in row] # Преобразуем обычный список списков в одномерный список

else:

elements = matrix.flatten() # Если это numpy ndarray, используем метод flatten() для преобразования в одномерный массив

# Сортируем список элементов по возрастанию

elements.sort()

# Определяем количество элементов в списке

n = len(elements)

# Проверяем, четное или нечетное количество элементов

if n % 2 != 0:

# Если количество элементов нечетное, медианой является средний элемент

median = elements[n // 2]

else:

# Если количество элементов четное, медианой является среднее значение двух средних элементов

median = (elements[n // 2 - 1] + elements[n // 2]) / 2

# Возвращаем найденную медиану

return median

# Задаем размеры матриц для тестирования

sizes = [(10, 10), (100, 100), (500, 500)]

# Генерация списков списков (обычных матриц) с случайными числами в диапазоне от 1 до 10000

matrixs = []

for size in sizes:

matrix = [[random.randint(1, 10000) for \_\_ in range(size[0])] for \_ in range(size[1])]

matrixs.append(matrix)

# Измеряем время выполнения пользовательской функции find\_median для списка матриц

%%timeit

for matrix in matrixs:

find\_median(matrix)

# Измеряем время выполнения встроенной функции numpy для нахождения медианы для списка матриц

%%timeit

for matrix in matrixs:

np.median(np.array(matrix))

# Генерация numpy массивов (матриц) с случайными числами в диапазоне от 0 до 10000

matrixs = [np.random.randint(0, 10000, size=size) for size in sizes]

# Измеряем время выполнения встроенной функции numpy для нахождения медианы для numpy массивов

%%timeit

for matrix in matrixs:

np.median(np.array(matrix))

# Измеряем время выполнения пользовательской функции find\_median для numpy массивов

%%timeit

for matrix in matrixs:

find\_median(matrix)

# **ПРИЛОЖЕНИЕ И Библиотека Pandas**

# Импорт используемых библиотек pandas и matplotlib

import pandas as pd

import matplotlib.pyplot as plt

# Чтение данных из CSV файлов

travels\_df = pd.read\_csv('travels.csv', delimiter=';') # Данные о путешествиях

travel\_agents\_df = pd.read\_csv('travel\_agents.csv', delimiter=';') # Данные о туроператорах

sale\_tour\_df = pd.read\_csv('sale\_of\_tour\_packages.csv', delimiter=';') # Данные о продажах турпакетов

# Фильтрация туроператоров, названия которых начинаются с буквы "В"

travel\_agents\_B = travel\_agents\_df[travel\_agents\_df['Название'].str.startswith('В')]

# Выборка продаж туров у туроператоров, отфильтрованных выше

travel\_B = sale\_tour\_df[sale\_tour\_df['ID туроператора'].isin(travel\_agents\_B['ID туроператора'])]

# Подсчет общего количества проданных путевок у туроператоров с названиями на "В"

total\_sold\_packages = travel\_B['Количество проданных путёвок'].sum()

print(f"Общее количество проданных путевок у туроператоров на 'В': {total\_sold\_packages}")

# Список городов Архангельской области

cities\_arkh = [

'Архангельск', 'Вельск', 'Каргополь', 'Мезень', 'Няндома', 'Северодвинск',

'Новодвинск', 'Мирный', 'Каргополь', 'Котлас', 'Коряжма', 'Онега', 'Шенкурск'

]

# Создаем Series для городов Архангельской области

cities\_arkh\_series = pd.Series(cities\_arkh, name='Города Архангельской области')

# Фильтрация туров по городам Архангельской области и выбор колонок Город и Стоимость

travels\_arkh = travels\_df[travels\_df['Город'].isin(cities\_arkh\_series)].loc[:, ['Город', 'Стоимость, на 1 чел']]

travels\_arkh.set\_index('Город', inplace=True) # Устанавливаем индекс по городу

# Подсчет общей стоимости путевок

total\_cost = travels\_arkh['Стоимость, на 1 чел'].sum()

# Создание меток для круговой диаграммы

labels = [f'{city}\n{cost} тыс. руб.' for city, cost in zip(travels\_arkh.index, travels\_arkh['Стоимость, на 1 чел'])]

# Построение круговой диаграммы распределения стоимости путевок по городам Архангельской области

fig, ax = plt.subplots()

ax.pie(travels\_arkh['Стоимость, на 1 чел'], labels=labels, autopct='%1.1f%%', startangle=90)

# Настройка заголовка и отображение диаграммы

plt.title(

f'Распределение стоимости путёвок по городам Архангельской области\nСумма всех путёвок: {total\_cost} тыс. руб.')

plt.ylabel('')

plt.show()

# Фильтрация данных по турам в Москву

moscow\_df = travels\_df[travels\_df['Город'] == 'Москва']

# Выборка данных о продажах туров в Москву

moscow\_travel = sale\_tour\_df[sale\_tour\_df['ID тура'].isin(moscow\_df['ID тура'])]

# Объединение данных о продажах туров в Москву с данными о туроператорах

moscow\_travel\_agents = moscow\_travel.merge(travel\_agents\_df, left\_on='ID туроператора', right\_on='ID туроператора')

# Расчет стоимости путевок для каждого туроператора

moscow\_travel\_agents['Стоимость путёвок'] = moscow\_travel\_agents['Количество проданных путёвок'] \* moscow\_df.loc[0, 'Стоимость, на 1 чел']

# Группировка данных по названию туроператора и подсчет количества проданных путевок

moscow\_count = moscow\_travel\_agents.groupby('Название')['Количество проданных путёвок'].sum()

moscow\_count = pd.DataFrame(moscow\_count)

# Группировка данных по названию туроператора и подсчет стоимости проданных путевок

moscow\_sum = moscow\_travel\_agents.groupby('Название')['Стоимость путёвок'].sum()

moscow\_sum = pd.DataFrame(moscow\_sum)

# Объединение данных о количестве и стоимости проданных путевок по туроператорам

moscow\_res\_df = pd.merge(moscow\_count, moscow\_sum, on='Название')

# Построение диаграммы для отображения стоимости и количества проданных путевок по туроператорам

fig, ax1 = plt.subplots(figsize=(14, 7))

# График стоимости путевок

moscow\_res\_df['Стоимость путёвок'].plot(kind='bar', ax=ax1, color='b', alpha=0.6, position=1, width=0.4, align='center')

ax1.set\_ylabel('Стоимость путёвок (руб.)', color='b')

ax1.set\_xlabel('Туроператоры')

ax1.set\_title('Стоимость и количество проданных путёвок по туроператорам')

# Добавление значений на график стоимости путевок

for index, value in enumerate(moscow\_res\_df['Стоимость путёвок']):

ax1.text(index, value + 100000, f'{value}', ha='center', color='b', fontsize=8)

# Создание второго графика для отображения количества проданных путевок

ax2 = ax1.twinx()

moscow\_res\_df['Количество проданных путёвок'].plot(kind='bar', ax=ax2, color='r', alpha=0.6, position=0, width=0.4, align='center')

ax2.set\_ylabel('Количество проданных путёвок', color='r')

# Добавление значений на график количества проданных путевок

for index, value in enumerate(moscow\_res\_df['Количество проданных путёвок']):

ax2.text(index, value + 2, f'{value}', ha='center', color='r', fontsize=8)

# Отображение графиков

plt.show()

# **ПРИЛОЖЕНИЕ К Визуализация**

# Импорт используемых библиотек pandas и plotly

import pandas as pd

import plotly.express as px

# Загрузка данных из CSV файла

df = pd.read\_csv('task19.csv', delimiter=';', encoding='utf-8')

# Создание нового столбца 'Направление', который объединяет 'Пункт отправления' и 'Пункт назначения'

df['Направление'] = df['Пункт отправления'] + '-' + df['Пункт назначения']

# Оставляем только необходимые столбцы в датафрейме

df = df.loc[:, ['Дата', 'Направление', 'Расстояние', 'Расход бензина', 'Масса груза']]

# Группируем данные по 'Направление' и 'Дата' и суммируем 'Расход бензина'

group\_df = df.groupby(['Направление', 'Дата'])['Расход бензина'].sum()

group\_df = pd.DataFrame(group\_df).reset\_index()

# Создание индекса и сброс его для получения форматированного датафрейма

group\_df = group\_df.reset\_index()

# Создание столбчатой диаграммы для отображения суммарного расхода бензина по направлениям и датам

fig = px.bar(group\_df, x='Направление', y='Расход бензина', color='Дата',

hover\_data=['Направление'],

labels={'Расход бензина': 'Расход бензина', 'Направление': 'Направление', 'Дата': 'Дата'},

title='Суммарный расход бензина по направлениям и датам')

# Поворот меток оси X на -45 градусов для лучшей читаемости

fig.update\_layout(xaxis\_tickangle=-45)

# Отображение графика

fig.show()

# Повторная загрузка данных из CSV файла

df = pd.read\_csv('task19.csv', delimiter=';', encoding='utf-8')

# Расчет эффективности расхода бензина (расход бензина на единицу расстояния)

df['Эффективность'] = df['Расход бензина'] / df['Расстояние']

# Поиск максимальной эффективности

df['Эффективность'].max()

# Получение строки с максимальной эффективностью

df.loc[df['Эффективность'].argmax(), :]

# Поиск минимальной эффективности

df['Эффективность'].min()

# Получение строки с минимальной эффективностью

df.loc[df['Эффективность'].argmin(), :]

# Группировка данных по 'Пункт отправления' и 'Пункт назначения' для расчета суммарного расстояния и расхода бензина

group\_from\_to = pd.DataFrame(df.groupby(['Пункт отправления', 'Пункт назначения']).agg(

{'Расстояние': 'sum', 'Расход бензина': 'sum'})).reset\_index()

# Группировка данных по 'Пункт назначения' и 'Пункт отправления' для расчета суммарного расстояния и расхода бензина

group\_to\_from = pd.DataFrame(df.groupby(['Пункт назначения', 'Пункт отправления']).agg(

{'Расстояние': 'sum', 'Расход бензина': 'sum'})).reset\_index()

# Переименование столбцов для объединения данных

group\_from\_to = group\_to\_from.rename(

columns={'Пункт отправления': 'Пункт назначения', 'Пункт назначения': 'Пункт отправления'})

# Объединение двух датафреймов по направлению (туда и обратно)

merged\_df = pd.merge(group\_from\_to, group\_to\_from, on=['Пункт назначения', 'Пункт отправления'], suffixes=('\_1', '\_2'))

# Расчет среднего расстояния и среднего расхода бензина для каждого направления

merged\_df['Расстояние (среднее)'] = merged\_df[['Расстояние\_1', 'Расстояние\_2']].mean(axis=1)

merged\_df['Расход бензина (средний)'] = merged\_df[['Расход бензина\_1', 'Расход бензина\_2']].mean(axis=1)

# Создание нового датафрейма с нужными столбцами

df = merged\_df.loc[:, ['Пункт назначения', 'Пункт отправления', 'Расстояние (среднее)', 'Расход бензина (средний)']]

# Расчет эффективности для среднего расхода бензина и расстояния

df['Эффективность'] = df['Расход бензина (средний)'] / df['Расстояние (среднее)']

# Получение уникальных значений пунктов назначения для создания тепловой карты

unique\_points = sorted(df['Пункт назначения'].unique())

# Создание пустой таблицы для данных тепловой карты

heatmap\_data = pd.DataFrame(index=unique\_points, columns=unique\_points)

# Заполнение данных тепловой карты эффективностью маршрутов

for index, row in df.iterrows():

heatmap\_data.loc[row['Пункт отправления'], row['Пункт назначения']] = row['Эффективность']

# Замена значений NaN на 0

heatmap\_data = heatmap\_data.fillna(0)

# Создание тепловой карты для визуализации эффективности маршрутов

fig = px.imshow(

heatmap\_data,

color\_continuous\_scale='Viridis',

labels={'color': 'Эффективность'},

title='Эффективность маршрутов (расход бензина на расстояние)'

)

# Поворот меток оси X на -45 градусов для лучшей читаемости

fig.update\_layout(xaxis\_tickangle=-45)

# Отображение графика

fig.show()

# Повторная загрузка данных из CSV файла

df = pd.read\_csv('task19.csv', delimiter=';', encoding='utf-8')

# Словарь для замены русских названий месяцев на английские

months\_mapping = {

"января": "January",

"февраля": "February",

"марта": "March",

"апреля": "April",

"мая": "May",

"июня": "June",

"июля": "July",

"августа": "August",

"сентября": "September",

"октября": "October",

"ноября": "November",

"декабря": "December"

}

# Замена русских названий месяцев на английские

for rus, eng in months\_mapping.items():

df['Дата'] = df['Дата'].str.replace(rus, eng)

# Преобразование строковых данных с датами в формат datetime

df['Дата'] = pd.to\_datetime(df['Дата'], format='%d %B', dayfirst=True)

# Создание диаграммы рассеяния для отображения изменения расхода бензина с течением времени для различных маршрутов

fig = px.scatter(

df,

x='Дата',

y='Расход бензина',

color='Пункт назначения',

size='Расстояние',

hover\_data=['Пункт отправления', 'Масса груза'],

labels={'Расход бензина': 'Расход бензина', 'Дата': 'Дата', 'Пункт назначения': 'Пункт назначения'},

title='Изменение расхода бензина с течением времени для различных маршрутов'

)

# Настройка формата отображения дат на оси X

fig.update\_layout(xaxis\_tickformat='%d %B')

# Отображение графика

fig.show()

# Повторная загрузка данных из CSV файла

df = pd.read\_csv('task19.csv', delimiter=';', encoding='utf-8')

# Группировка данных по дате и суммирование расстояний

group\_df = df.groupby('Дата')['Расстояние'].sum()

group\_df = pd.DataFrame(group\_df).sort\_values('Расстояние', ascending=False).reset\_index()

# Создание графика воронки для визуализации расстояний по датам

fig = px.funnel(group\_df, x='Расстояние', y='Дата')

# Отображение графика

fig.show()

# **ПРИЛОЖЕНИЕ Л Итоговое задание**

# Импорт используемых библиотек pandas и plotly  
import pandas as pd  
import plotly.express as px  
import plotly.graph\_objects as go  
  
# Чтение данных из CSV файла ОГЭ и переименование колонок для удобства  
oge\_df = pd.read\_csv('oge.csv', delimiter=';', encoding='utf-8')  
oge\_df = oge\_df.rename(columns={'Оцека по 100': 'Оценка (100)', 'Оцека по 5': 'Оценка (5)'})  
  
# Вывод информации о DataFrame (структура данных, типы данных и количество значений)  
oge\_df.info()  
  
# Создание копии исходного DataFrame для работы с новыми столбцами  
oge\_df\_copy = oge\_df.copy()  
  
# Подсчет количества правильных ('+') и неправильных ('-') ответов в части В  
oge\_df\_copy['+\_Часть В'] = oge\_df\_copy['Часть В'].str.count(r'\+')  
oge\_df\_copy['-\_Часть В'] = oge\_df\_copy['Часть В'].str.count('-')  
  
# Подсчет количества пропущенных значений в колонках от '1\_Часть В' до '12\_Часть В'  
oge\_df.loc[:, '1\_Часть В':'12\_Часть В'].isna().sum(axis=1).sum()  
  
# Замена '+' на 1 и '-' на 0 для колонок части В  
oge\_df.loc[:, '1\_Часть В':'12\_Часть В'] = oge\_df.loc[:, '1\_Часть В':'12\_Часть В'].replace({'+': 1, '-': 0})  
  
# Подсчет количества '+' и '-' после преобразования и проверка совпадений с исходным подсчетом  
plus\_columns = oge\_df.loc[:, '1\_Часть В':'12\_Часть В'].eq(1).sum(axis=1)  
minus\_column = oge\_df.loc[:, '1\_Часть В':'12\_Часть В'].eq(0).sum(axis=1)  
  
# Проверка правильности подсчетов  
(oge\_df\_copy['+\_Часть В'] != plus\_columns).sum()  
(oge\_df\_copy['-\_Часть В'] != minus\_column).sum()  
  
# Удаление колонки 'Часть В', так как ее данные преобразованы в отдельные колонки  
oge\_df = oge\_df.drop(columns=['Часть В'])  
  
# Получение уникальных значений из столбца 'Предмет'  
oge\_df['Предмет'].unique()  
  
# Получение уникальных значений из столбца 'Код предмета'  
oge\_df['Код предмета'].unique()  
  
# Удаление колонок 'Предмет' и 'Код предмета' для создания нового DataFrame  
oge\_inf\_df = oge\_df.drop(columns=['Предмет', 'Код предмета'])  
  
# Получение уникальных значений из столбца 'Тип ОО'  
oge\_inf\_df['Тип ОО'].unique()  
  
# Удаление колонки 'Тип ОО' для создания нового DataFrame  
oge\_inf\_oo\_df = oge\_inf\_df.drop(columns=['Тип ОО'])  
  
# Проверка наличия значений в колонке 'Часть D' и удаление этой колонки  
oge\_inf\_oo\_df['Часть D'].isna().sum() == oge\_inf\_oo\_df.shape[0]  
oge\_inf\_oo\_df = oge\_inf\_oo\_df.drop(columns=['Часть D'])  
  
# Суммирование значений в колонке 'Оценка (100)' и удаление этой колонки  
oge\_inf\_oo\_df['Оценка (100)'].sum()  
oge\_inf\_oo\_df = oge\_inf\_oo\_df.drop(columns=['Оценка (100)'])  
  
# Получение уникальных значений из столбца 'Выпускник'  
oge\_inf\_oo\_df['Выпускник'].unique()  
  
# Проверка совпадения значений в части С с соответствующими колонками  
(oge\_inf\_oo\_df['Часть С'].str[0].astype(int) != oge\_inf\_oo\_df['1\_Часть С']).sum()  
(oge\_inf\_oo\_df['Часть С'].str[4].astype(int) != oge\_inf\_oo\_df['2\_Часть С']).sum()  
(oge\_inf\_oo\_df['Часть С'].str[-4].astype(int) != oge\_inf\_oo\_df['3\_Часть С']).sum()  
  
# Удаление колонки 'Часть С' после проверки  
oge\_inf\_oo\_df = oge\_inf\_oo\_df.drop(columns=['Часть С'])  
  
# Создание новых колонок с суммой баллов по частям В и С  
oge\_inf\_oo\_df['Сумма баллов части В'] = oge\_inf\_oo\_df.loc[:, '1\_Часть В':'12\_Часть В'].sum(axis=1)  
oge\_inf\_oo\_df['Сумма баллов части С'] = oge\_inf\_oo\_df.loc[:, '1\_Часть С':'3\_Часть С'].sum(axis=1)  
  
# Перемещение колонок в нужный порядок  
col = oge\_inf\_oo\_df.pop('Сумма баллов части В')  
oge\_inf\_oo\_df.insert(20, 'Сумма баллов части В', col)  
  
col = oge\_inf\_oo\_df.pop('Сумма баллов части С')  
oge\_inf\_oo\_df.insert(21, 'Сумма баллов части С', col)  
  
# Переименование колонок для удобства использования  
oge\_inf\_oo\_df = oge\_inf\_oo\_df.rename(  
 columns={'Код ОО': 'code\_oo', 'Краткое наименование ОО': 'short\_name\_oo', 'Вид ОО': 'type\_oo',  
 'Выпускник': 'graduate', 'Вариант': 'variant', 'Сумма баллов части В': 'sum\_b',  
 'Сумма баллов части С': 'sum\_c',  
 'Первичный балл': 'primary\_score', '% Оценка': 'score\_%', 'Оценка (5)': 'score\_5',  
 '1\_Часть В': 'ex\_1\_b', '2\_Часть В': 'ex\_2\_b', '3\_Часть В': 'ex\_3\_b', '4\_Часть В': 'ex\_4\_b',  
 '5\_Часть В': 'ex\_5\_b', '6\_Часть В': 'ex\_6\_b', '7\_Часть В': 'ex\_7\_b', '8\_Часть В': 'ex\_8\_b',  
 '9\_Часть В': 'ex\_9\_b', '10\_Часть В': 'ex\_10\_b', '11\_Часть В': 'ex\_11\_b', '12\_Часть В': 'ex\_12\_b',  
 '1\_Часть С': 'ex\_13\_c', '2\_Часть С': 'ex\_14\_c', '3\_Часть С': 'ex\_15\_c'})  
  
# Проверка типов данных после переименования и преобразования колонок  
oge\_inf\_oo\_df.info()  
  
# Определение колонок, относящихся к заданиям части В, и преобразование их типов данных в целочисленный формат  
columns\_b = oge\_inf\_oo\_df.filter(regex=r'^ex\_\d+\_b$').columns  
oge\_inf\_oo\_df[columns\_b] = oge\_inf\_oo\_df[columns\_b].astype(int)  
  
# Проверка типов данных после преобразования  
oge\_inf\_oo\_df.info()  
  
# Проверка совпадения суммы баллов частей В и С с первичным баллом  
((oge\_inf\_oo\_df.sum\_b + oge\_inf\_oo\_df.sum\_c) != oge\_inf\_oo\_df.primary\_score).sum()  
  
# Добавление колонки с кодом региона, выделенного из первых трех цифр кода ОО  
oge\_inf\_oo\_df['code\_region'] = oge\_inf\_oo\_df['code\_oo'].astype(str).str[:3].astype(int)  
  
# Фильтрация данных по названию ОО с № 64  
oge\_inf\_oo\_df[oge\_inf\_oo\_df.short\_name\_oo.str.contains('№ 64', na=False)]  
  
# Чтение данных из CSV файла ЕГЭ и переименование некоторых колонок для удобства  
ege\_df = pd.read\_csv('ege 2022.csv', delimiter=';', encoding='utf-8')  
ege\_df = ege\_df.rename(columns={'Код ОО': 'code\_oo', 'Первичный балл': 'primary\_score', 'Балл': 'score'})  
  
# Удаление ненужной колонки 'id'  
ege\_df = ege\_df.drop(columns=['id'])  
  
# Проверка длины значения в колонке 'Задания с кратким ответом' для первой строки  
len(ege\_df.loc[0, 'Задания с кратким ответом'])  
  
# Проверка совпадения суммы баллов по всем заданиям (Task1-Task27) с первичным баллом  
(ege\_df.primary\_score != ege\_df.loc[:, 'Task1':'Task27'].sum(axis=1)).sum()  
  
# Переименование колонок Task1-Task27 для более удобного использования  
rename\_columns = {f'Task{i}': f'ex\_{i}' for i in range(1, 28)}  
ege\_df.rename(columns=rename\_columns, inplace=True)  
  
# Удаление колонки 'Задания с кратким ответом' после преобразования  
ege\_df.drop(columns='Задания с кратким ответом', inplace=True)  
  
# Создание словаря соответствий для кодов ОО и сокращенных наименований из данных ОГЭ  
oge\_df\_code\_oo = oge\_inf\_oo\_df.set\_index('code\_oo')['short\_name\_oo'].to\_dict()  
  
# Добавление сокращенных наименований ОО в данные ЕГЭ на основе соответствий из ОГЭ  
ege\_df['short\_name\_oo'] = ege\_df['code\_oo'].map(oge\_df\_code\_oo)  
  
# Чтение данных из Excel файла с кодировками  
codificator\_df = pd.read\_excel('codificator.xlsx')  
  
# Проверка названий колонок после чтения данных  
codificator\_df.columns  
  
# Удаление ненужной колонки '№'  
codificator\_df = codificator\_df.drop(columns=['№'])  
  
# Переименование колонок для удобства использования  
codificator\_df = codificator\_df.rename(columns={  
 'Код ОО': 'code\_oo',  
 'Наименование образовательной организации согласно Уставу образовательной организации ': 'name\_oo',  
 'Сокращенное наименование образовательной организации согласно Уставу образовательной организации ': 'short\_name\_oo',  
 'Код АТЕ': 'code\_region'  
})  
  
# Проверка количества пропущенных значений в данных  
codificator\_df.isna().sum()  
  
# Проверка информации о типах данных после переименования колонок  
codificator\_df.info()  
  
# Создание копии данных ЕГЭ для дальнейшего объединения  
ege\_df\_copy = ege\_df.copy()  
  
# Объединение данных ЕГЭ с сокращенными наименованиями ОО из файла кодировок (если они не совпадают, замена)  
merged\_df = ege\_df\_copy.merge(codificator\_df[['code\_oo', 'short\_name\_oo']], on='code\_oo', how='left',  
 suffixes=('', '\_y'))  
merged\_df['short\_name\_oo'] = merged\_df['short\_name\_oo'].fillna(merged\_df['short\_name\_oo\_y'])  
merged\_df.drop(columns=['short\_name\_oo\_y'], inplace=True)  
  
# Присваивание объединенного DataFrame переменной ege\_df  
ege\_df = merged\_df  
  
# Добавление колонки с кодом региона, выделенного из первых трех цифр кода ОО  
ege\_df['code\_region'] = ege\_df['code\_oo'].astype(str).str[:3].astype(int)  
  
  
# Функция для категоризации значений баллов в пятибалльной шкале  
def categorize(value):  
 if value < 39:  
 return 2  
 elif 40 <= value < 56:  
 return 3  
 elif 57 <= value <= 72:  
 return 4  
 else:  
 return 5  
  
  
# Применение функции для добавления новой колонки с оценками в пятибалльной шкале  
ege\_df['score\_5'] = ege\_df['score'].apply(categorize)  
  
# Проверка информации о типах данных после добавления новой колонки  
ege\_df.info()  
  
# Чтение данных из Excel файла с географической информацией  
geo\_df = pd.read\_excel('geo.xlsx')  
  
# Проверка названий колонок после чтения данных  
geo\_df.columns  
  
# Переименование колонок для удобства использования  
geo\_df = geo\_df.rename(  
 columns={'Код района': 'code\_region', 'Район': 'region', 'Расстояние до Саратова': 'distance\_to\_saratov'})  
  
# Применение функции удаления пробелов к строковым значениям в данных географии  
geo\_df = geo\_df.applymap(lambda x: x.strip() if isinstance(x, str) else x)  
  
# Проверка информации о типах данных после переименования колонок и преобразования  
geo\_df.info()  
  
# Создание гистограммы распределения первичных баллов ОГЭ 2022 года с разбивкой по оценкам в пятибалльной шкале  
fig = px.histogram(  
 oge\_inf\_oo\_df,  
 x='primary\_score',  
 nbins=30,  
 title='Распределение первичных баллов ОГЭ 2022 года',  
 color='score\_5',  
 color\_discrete\_map={  
 2: 'lightcoral',  
 3: 'khaki',  
 4: 'lightskyblue',  
 5: 'lightgreen'  
 }  
)  
  
# Настройка осей и отображения гистограммы  
fig.update\_layout(  
 xaxis\_title='Баллы',  
 yaxis\_title='Количество',  
 bargap=0.1,  
 bargroupgap=0.1,  
 title\_x=0.5,  
 xaxis=dict(tickmode='linear'),  
)  
  
# Отображение графика  
fig.show()  
  
# Построение коробчатой диаграммы (box plot) распределения первичных баллов ОГЭ 2022 года  
fig = px.box(  
 oge\_inf\_oo\_df,  
 y='primary\_score',  
 title='Распределение первичных баллов ОГЭ 2022 года'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis\_title='Баллы',  
 title\_x=0.5,  
 xaxis=dict(  
 tickmode='linear',  
 dtick=1  
 )  
)  
  
# Отображение графика  
fig.show()  
  
# Построение гистограммы распределения первичных баллов ЕГЭ 2022 года  
fig = px.histogram(ege\_df, x='primary\_score', nbins=30, title='Распределение первичных баллов ЕГЭ 2022 года')  
  
# Настройка осей и отображения гистограммы  
fig.update\_layout(  
 xaxis\_title='Баллы',  
 yaxis\_title='Количество',  
 bargap=0.1,  
 bargroupgap=0.1,  
 title\_x=0.5,  
 xaxis=dict(tickmode='linear'),  
)  
  
# Отображение графика  
fig.show()  
  
# Создание DataFrame для распределения вторичных баллов ЕГЭ и построение линейного графика распределения  
frequency\_df = ege\_df['score'].value\_counts().reset\_index()  
frequency\_df.columns = ['score', 'count']  
frequency\_df = frequency\_df.sort\_values(by='score')  
  
fig = px.line(  
 frequency\_df,  
 x='score',  
 y='count',  
 title='Распределение вторичных баллов ЕГЭ'  
)  
  
# Настройка осей и отображения графика  
fig.update\_layout(  
 xaxis\_title='Баллы',  
 yaxis\_title='Количество',  
 title\_x=0.5,  
 xaxis=dict(tickmode='linear', range=[0, 100]),  
)  
  
# Отображение графика  
fig.show()  
  
# Построение гистограммы распределения вторичных баллов ЕГЭ 2022 года с разбивкой по оценкам в пятибалльной шкале  
fig = px.histogram(  
 ege\_df,  
 x='score',  
 nbins=30,  
 title='Распределение вторичных баллов ЕГЭ 2022 года',  
 color='score\_5',  
 color\_discrete\_map={  
 2: 'lightcoral',  
 3: 'khaki',  
 4: 'lightskyblue',  
 5: 'lightgreen'  
 }  
)  
  
# Настройка осей и отображения гистограммы  
fig.update\_layout(  
 xaxis\_title='Баллы',  
 yaxis\_title='Количество',  
 bargap=0.1,  
 bargroupgap=0.1,  
 title\_x=0.5,  
 xaxis=dict(tickmode='linear'),  
)  
  
# Отображение графика  
fig.show()  
  
# Построение коробчатой диаграммы (box plot) распределения первичных баллов ЕГЭ 2022 года  
fig = px.box(  
 ege\_df,  
 y='primary\_score',  
 title='Распределение первичных баллов ЕГЭ 2022 года'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis\_title='Баллы',  
 title\_x=0.5,  
 xaxis=dict(  
 tickmode='linear',  
 dtick=1  
 )  
)  
  
# Отображение графика  
fig.show()  
  
# Построение коробчатой диаграммы (box plot) распределения вторичных баллов ЕГЭ 2022 года  
fig = px.box(  
 ege\_df,  
 y='score',  
 title='Распределение вторичных баллов ЕГЭ 2022 года'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis\_title='Баллы',  
 title\_x=0.5,  
 xaxis=dict(  
 tickmode='linear',  
 dtick=1  
 )  
)  
  
# Отображение графика  
fig.show()  
  
# Объединение данных ОГЭ с данными о регионах и их типами  
oge\_inf = oge\_inf\_oo\_df.merge(geo\_df[['code\_region', 'region']], on='code\_region', how='left').fillna('undefinded')  
  
# Объединение данных по регионам для районов Саратова  
oge\_inf.loc[oge\_inf.region.isin(  
 ['Кировский', 'Ленинский', 'Волжский', 'Заводской', 'Саратовский', 'Октябрьский', 'Юбилейный', 'Фрунзенский']),  
'region'] = 'Саратов'  
  
# Рассчет средних первичных баллов по ОГЭ по типам образовательных организаций (ОО) и регионам  
oge\_inf\_average = oge\_inf.groupby(['region', 'type\_oo'])['primary\_score'].median().reset\_index()  
oge\_inf\_average.rename(columns={'primary\_score': 'average\_score'}, inplace=True)  
  
# Построение столбчатой диаграммы (bar plot) для отображения средних баллов по регионам и типам ОО  
fig\_oge\_avg = px.bar(  
 oge\_inf\_average,  
 x='region',  
 y='average\_score',  
 color='type\_oo',  
 title='Средние первичные баллы по ОГЭ по типам ОО и регионам',  
 labels={'region': 'Регион', 'average\_score': 'Средний балл', 'type\_oo': 'Тип ОО'}  
)  
  
# Настройка осей и отображения диаграммы  
fig\_oge\_avg.update\_layout(  
 xaxis\_tickangle=-45,  
 xaxis\_title='Регион',  
 yaxis\_title='Средний балл',  
 height=800,  
 width=1500,  
 margin=dict(l=20, r=20, t=50, b=100)  
)  
  
# Построение диаграммы для средних первичных баллов ОГЭ по типам образовательных учреждений  
average\_scores = oge\_inf\_oo\_df.groupby('type\_oo')['primary\_score'].median().reset\_index()  
  
# Сортировка данных для построения диаграммы  
average\_scores\_sort = average\_scores.sort\_values(by=['primary\_score'], ascending=False)  
  
# Построение столбчатой диаграммы (bar plot) для школ по средним первичным баллам ОГЭ  
fig = px.bar(  
 average\_scores\_sort,  
 x='type\_oo',  
 y='primary\_score',  
 labels={'primary\_score': 'Средний балл', 'type\_oo': 'Школы'},  
 title='Школы по средним первичным баллам ОГЭ',  
 text='primary\_score'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:.0f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Добавление колонки с количеством учеников в каждой образовательной организации (ОО)  
oge\_inf\_oo\_df['student\_count'] = oge\_inf\_oo\_df.groupby('short\_name\_oo')['short\_name\_oo'].transform('count')  
  
# Вычисление медианных значений первичных баллов и количества учеников для каждой ОО  
average\_scores = oge\_inf\_oo\_df.groupby('short\_name\_oo').agg(  
 {'primary\_score': 'median', 'student\_count': 'first'}).reset\_index()  
  
# Определение минимального количества учеников для фильтрации  
min\_students = average\_scores['student\_count'].mean()  
  
# Фильтрация школ с количеством учеников больше или равно минимальному  
filtered\_schools = average\_scores[average\_scores['student\_count'] >= min\_students]  
  
# Получение топ-10 лучших школ по среднему первичному баллу ОГЭ  
best\_schools = filtered\_schools.sort\_values(by='primary\_score', ascending=False).head(10)  
  
# Построение столбчатой диаграммы (bar plot) для топ-10 лучших школ по среднему первичному баллу ОГЭ  
fig = px.bar(  
 best\_schools,  
 x='short\_name\_oo',  
 y='primary\_score',  
 color='student\_count',  
 labels={'primary\_score': 'Средний первичный балл ОГЭ', 'short\_name\_oo': 'ОО',  
 'student\_count': 'Количество учеников'},  
 title='Топ 10 школ по среднему первичному баллу ОГЭ (с учётом количества учеников)',  
 text='primary\_score'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Получение топ-10 худших школ по среднему первичному баллу ОГЭ  
worst\_schools = filtered\_schools.sort\_values(by='primary\_score').head(10)  
  
# Построение столбчатой диаграммы (bar plot) для топ-10 худших школ по среднему первичному баллу ОГЭ  
fig = px.bar(  
 worst\_schools,  
 x='short\_name\_oo',  
 y='primary\_score',  
 color='student\_count',  
 labels={'primary\_score': 'Средний первичный балл ОГЭ', 'short\_name\_oo': 'ОО',  
 'student\_count': 'Количество учеников'},  
 title='Топ 10 худших школ по среднему первичному баллу ОГЭ (с учётом количества учеников)',  
 text='primary\_score'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Группировка данных по коду ОО и получению их типов  
oge\_inf\_oo\_df\_grouped = oge\_inf\_oo\_df.groupby('code\_oo').agg({'type\_oo': 'first'}).reset\_index()  
  
# Объединение данных ЕГЭ с типами ОО и регионами  
ege\_merged\_df = ege\_df.merge(oge\_inf\_oo\_df\_grouped[['code\_oo', 'type\_oo']], how='left', on='code\_oo').fillna(  
 'undefined')  
ege\_merged\_df = ege\_merged\_df.merge(geo\_df[['code\_region', 'region']], how='left', on='code\_region').fillna('undefined')  
  
# Объединение данных по регионам для районов Саратова  
ege\_merged\_df.loc[ege\_merged\_df.region.isin(  
 ['Кировский', 'Ленинский', 'Волжский', 'Заводской', 'Саратовский', 'Октябрьский', 'Юбилейный', 'Фрунзенский']),  
'region'] = 'Саратов'  
  
# Добавление колонки с количеством учеников в каждой образовательной организации (ОО) в данных ЕГЭ  
ege\_merged\_df['student\_count'] = ege\_merged\_df.groupby('short\_name\_oo')['short\_name\_oo'].transform('count')  
  
# Добавление колонки с количеством учеников в каждом регионе в данных ЕГЭ  
ege\_merged\_df['student\_count\_region'] = ege\_merged\_df.groupby('region')['region'].transform('count')  
  
# Рассчет медианных значений первичных баллов по ЕГЭ по типам образовательных учреждений  
average\_scores\_ege = ege\_merged\_df.groupby(['type\_oo'])['primary\_score'].median().reset\_index().sort\_values(  
 by=['primary\_score'])  
  
# Построение столбчатой диаграммы (bar plot) для отображения средних баллов ЕГЭ по школам  
fig = px.bar(  
 average\_scores\_ege,  
 x='type\_oo',  
 y='primary\_score',  
 labels={'primary\_score': 'Средний первичный балл ЕГЭ', 'type\_oo': 'Вид ОО'},  
 title='Средние баллы ЕГЭ по школам',  
 text='primary\_score'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Рассчет средних значений первичных баллов по ЕГЭ по регионам и количеству учеников в каждом регионе  
average\_scores\_ege = ege\_merged\_df.groupby('region').agg(  
 {'primary\_score': 'mean', 'student\_count\_region': 'first'}).reset\_index().reset\_index().sort\_values(  
 by=['primary\_score'])  
  
# Построение столбчатой диаграммы (bar plot) для отображения средних первичных баллов ЕГЭ по регионам  
fig = px.bar(  
 average\_scores\_ege,  
 x='region',  
 y='primary\_score',  
 color='student\_count\_region',  
 barmode='group',  
 labels={'primary\_score': 'Средний первичный балл ЕГЭ', 'region': 'Регион',  
 'student\_count\_region': 'Количество учеников'},  
 title='Средние первичные баллы ЕГЭ по регионам',  
 text='primary\_score'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide',  
 coloraxis\_colorbar=dict(  
 title='Количество учеников',  
 orientation='v'  
 )  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Рассчет средних значений оценок в пятибалльной шкале по типам ОО  
average\_scores\_ege = ege\_merged\_df.groupby(['type\_oo'])['score\_5'].mean().reset\_index().sort\_values(by=['score\_5'])  
  
# Построение столбчатой диаграммы (bar plot) для отображения средних оценок ЕГЭ по типам образовательных учреждений  
fig = px.bar(  
 average\_scores\_ege,  
 x='type\_oo',  
 y='score\_5',  
 barmode='group',  
 labels={'type\_oo': 'Школа', 'score\_5': 'Оценка в пятибалльной шкале'},  
 title='Средняя оценка ЕГЭ по образовательным учреждениям (школа)',  
 text='score\_5'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Рассчет средних значений оценок в пятибалльной шкале по регионам и количеству учеников в каждом регионе  
average\_scores\_ege = ege\_merged\_df.groupby(['region']).agg(  
 {'score\_5': 'mean', 'student\_count\_region': 'first'}).reset\_index().reset\_index().sort\_values(by=['score\_5'])  
  
# Построение столбчатой диаграммы (bar plot) для отображения средних оценок ЕГЭ по регионам  
fig = px.bar(  
 average\_scores\_ege,  
 x='region',  
 y='score\_5',  
 color='student\_count\_region',  
 barmode='group',  
 labels={'score\_5': 'Средние оценки ЕГЭ', 'region': 'Регион', 'student\_count\_region': 'Количество учеников'},  
 title='Средние оценки ЕГЭ по регионам',  
 text='score\_5'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide',  
 coloraxis\_colorbar=dict(  
 title='Количество учеников',  
 orientation='v'  
 )  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Рассчет медианных значений первичных баллов по ЕГЭ для каждой образовательной организации (ОО) с количеством учеников  
average\_primary\_scores = ege\_merged\_df.groupby('short\_name\_oo').agg(  
 {'primary\_score': 'median', 'student\_count': 'first'}).reset\_index()  
  
# Установка минимального количества учеников для фильтрации школ  
min\_students = 10  
  
# Фильтрация школ с количеством учеников больше или равно минимальному  
filtered\_schools = average\_primary\_scores[average\_primary\_scores['student\_count'] >= min\_students]  
  
# Получение топ-10 лучших школ по среднему первичному баллу ЕГЭ  
best\_schools = filtered\_schools.sort\_values(by='primary\_score', ascending=False).head(10)  
  
# Построение столбчатой диаграммы (bar plot) для топ-10 лучших образовательных организаций по среднему первичному баллу ЕГЭ  
fig = px.bar(  
 best\_schools,  
 x='short\_name\_oo',  
 y='primary\_score',  
 color='student\_count',  
 labels={'primary\_score': 'Средний первичный балл ЕГЭ', 'short\_name\_oo': 'Школа',  
 'student\_count': 'Количество учеников'},  
 title='Топ 10 лучших образовательных организаций по среднему первичному баллу ЕГЭ',  
 text='student\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Получение топ-10 худших школ по среднему первичному баллу ЕГЭ  
worst\_schools = filtered\_schools.sort\_values(by='primary\_score').head(10)  
  
# Построение столбчатой диаграммы (bar plot) для топ-10 худших образовательных организаций по среднему первичному баллу ЕГЭ  
fig = px.bar(  
 worst\_schools,  
 x='short\_name\_oo',  
 y='primary\_score',  
 color='student\_count',  
 labels={'primary\_score': 'Средний первичный балл ЕГЭ', 'short\_name\_oo': 'Школа',  
 'student\_count': 'Количество учеников'},  
 title='Топ 10 худших образовательных организаций по среднему первичному баллу ЕГЭ',  
 text='student\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Рассчет медианных значений вторичных баллов по ЕГЭ для каждой образовательной организации (ОО) с количеством учеников  
average\_secondary\_scores = ege\_merged\_df.groupby('short\_name\_oo').agg(  
 {'score': 'median', 'student\_count': 'first'}).reset\_index()  
  
# Фильтрация школ с количеством учеников больше или равно минимальному  
filtered\_schools = average\_secondary\_scores[average\_secondary\_scores['student\_count'] >= min\_students]  
  
# Получение топ-10 лучших школ по среднему вторичному баллу ЕГЭ  
best\_schools = filtered\_schools.sort\_values(by='score', ascending=False).head(10)  
  
# Построение столбчатой диаграммы (bar plot) для топ-10 лучших образовательных организаций по среднему вторичному баллу ЕГЭ  
fig = px.bar(  
 best\_schools,  
 x='short\_name\_oo',  
 y='score',  
 color='student\_count',  
 labels={'score': 'Средний вторичный балл ЕГЭ', 'short\_name\_oo': 'Школа', 'student\_count': 'Количество учеников'},  
 title='Топ 10 лучших образовательных организаций по среднему вторичному баллу ЕГЭ',  
 text='student\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Получение топ-10 худших школ по среднему вторичному баллу ЕГЭ  
worst\_schools = filtered\_schools.sort\_values(by='score').head(10)  
  
# Построение столбчатой диаграммы (bar plot) для топ-10 худших образовательных организаций по среднему вторичному баллу ЕГЭ  
fig = px.bar(  
 worst\_schools,  
 x='short\_name\_oo',  
 y='score',  
 color='student\_count',  
 labels={'score': 'Средний вторичный балл ЕГЭ', 'short\_name\_oo': 'Школа', 'student\_count': 'Количество учеников'},  
 title='Топ 10 худших образовательных организаций по среднему вторичному баллу ЕГЭ',  
 text='student\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text:,.1~f}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Добавление колонки с количеством учеников в каждом регионе в данных ОГЭ  
oge\_inf\_oo\_df['student\_count\_region'] = oge\_inf\_oo\_df.groupby('code\_region')['code\_region'].transform('count')  
  
# Объединение данных ОГЭ с данными о регионах  
oge\_inf\_oo\_df = oge\_inf\_oo\_df.merge(geo\_df[['code\_region', 'region']], how='left', on='code\_region').fillna('undefined')  
  
# Объединение данных по регионам для районов Саратова  
oge\_inf\_oo\_df.loc[oge\_inf\_oo\_df.region.isin(  
 ['Кировский', 'Ленинский', 'Волжский', 'Заводской', 'Саратовский', 'Октябрьский', 'Юбилейный', 'Фрунзенский']),  
'region'] = 'Саратов'  
  
# Фильтрация данных по ученикам, получившим оценку 5 по ОГЭ  
oge\_5 = oge\_inf\_oo\_df[oge\_inf\_oo\_df.score\_5 == 5].reset\_index(drop=True)  
  
# Создание списка колонок заданий части В  
task\_columns = [col for col in oge\_5.columns if col.startswith('ex\_')]  
  
# Подсчет количества учеников, выполнивших каждое задание из части В (не нулевое значение)  
excellent\_students = oge\_5[task\_columns].apply(lambda x: (x != 0).sum()).reset\_index()  
excellent\_students.columns = ['task', 'students\_count']  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества отличников, выполнивших каждое задание  
fig = px.bar(  
 excellent\_students,  
 x='task',  
 y='students\_count',  
 labels={'task': 'Задание', 'students\_count': 'Количество учеников'},  
 title=f'Количество отличников, выполнивших каждое задание. Всего отличников: {oge\_5.shape[0]}',  
 text='students\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Создание списка колонок заданий части С  
task\_columns\_c = [col for col in oge\_5.columns if col.startswith('ex') and col.endswith('c')]  
  
# Подсчет количества отличников, выполнивших каждое задание части C на 1, 2 и 3 балла  
results = []  
for col in task\_columns\_c:  
 for score in [1, 2, 3]:  
 count = (oge\_5[col] == score).sum()  
 results.append({'task': col, 'score': score, 'count': count})  
  
results\_df = pd.DataFrame(results)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества отличников, выполнивших каждое задание части C  
fig = px.bar(  
 results\_df,  
 x='task',  
 y='count',  
 color='score',  
 barmode='group',  
 labels={'task': 'Задание', 'count': 'Количество отличников', 'score': 'Баллы'},  
 title=f'Количество отличников, выполнивших каждое задание части C на 1, 2 и 3 балла. Всего отличников: {oge\_5.shape[0]}',  
 text='count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Подсчет количества учеников по регионам и количества отличников по регионам  
oge\_region = oge\_inf\_oo\_df['region'].value\_counts().reset\_index()  
oge\_region.columns = ['region', 'total\_count']  
  
oge\_5\_region = oge\_5['region'].value\_counts().reset\_index()  
oge\_5\_region.columns = ['region', 'excellent\_count']  
  
# Объединение данных по регионам для отображения количества отличников и общего количества учеников  
merged\_counts = pd.merge(oge\_5\_region, oge\_region, on='region').sort\_values(by='total\_count', ascending=False)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества отличников по регионам  
fig\_bar = px.bar(  
 merged\_counts,  
 x='region',  
 y='total\_count',  
 labels={'region': 'Регион', 'total\_count': 'Количество учеников по региону в общем'},  
 title='Количество отличников по регионам',  
)  
  
# Добавление аннотаций на диаграмму с количеством отличников  
for i, row in merged\_counts.iterrows():  
 fig\_bar.add\_annotation(  
 x=row['region'],  
 y=row['total\_count'],  
 text=f"{row['excellent\_count']}",  
 showarrow=False,  
 yshift=10  
 )  
  
# Настройка осей и отображения диаграммы  
fig\_bar.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
)  
  
# Отображение графика  
fig\_bar.show()  
  
# Фильтрация данных по ученикам, получившим оценку 4 по ОГЭ  
oge\_4 = oge\_inf\_oo\_df[oge\_inf\_oo\_df.score\_5 == 4].reset\_index(drop=True)  
  
# Подсчет количества учеников, выполнивших каждое задание (не нулевое значение) из части В, для хорошистов  
task\_columns = [col for col in oge\_4.columns if col.startswith('ex\_')]  
good\_students = oge\_4[task\_columns].apply(lambda x: (x != 0).sum()).reset\_index()  
good\_students.columns = ['task', 'students\_count']  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества хорошистов, выполнивших каждое задание  
fig = px.bar(  
 good\_students,  
 x='task',  
 y='students\_count',  
 labels={'task': 'Задание', 'students\_count': 'Количество учеников'},  
 title=f'Количество хорошистов, выполнивших каждое задание. Всего хорошистов: {oge\_4.shape[0]}',  
 text='students\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Создание списка колонок заданий части C для хорошистов  
task\_columns\_c = [col for col in oge\_4.columns if col.startswith('ex') and col.endswith('c')]  
  
# Подсчет количества хорошистов, выполнивших каждое задание части C на 1, 2 и 3 балла  
results = []  
for col in task\_columns\_c:  
 for score in [1, 2, 3]:  
 count = (oge\_4[col] == score).sum()  
 results.append({'task': col, 'score': score, 'count': count})  
  
results\_df = pd.DataFrame(results)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества хорошистов, выполнивших каждое задание части C  
fig = px.bar(  
 results\_df,  
 x='task',  
 y='count',  
 color='score',  
 barmode='group',  
 labels={'task': 'Задание', 'count': 'Количество хорошистов', 'score': 'Баллы'},  
 title=f'Количество хорошистов, выполнивших каждое задание части C на 1, 2 и 3 балла. Всего хорошистов: {oge\_4.shape[0]}',  
 text='count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Подсчет количества хорошистов по регионам  
oge\_4\_region = oge\_4['region'].value\_counts().reset\_index()  
oge\_4\_region.columns = ['region', 'good\_count']  
  
# Объединение данных по регионам для отображения количества хорошистов и общего количества учеников  
merged\_counts = pd.merge(oge\_4\_region, oge\_region, on='region').sort\_values(by='total\_count', ascending=False)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества хорошистов по регионам  
fig\_bar = px.bar(  
 merged\_counts,  
 x='region',  
 y='total\_count',  
 labels={'region': 'Регион', 'total\_count': 'Количество учеников по региону в общем'},  
 title='Количество хорошистов по регионам',  
)  
  
# Добавление аннотаций на диаграмму с количеством хорошистов  
for i, row in merged\_counts.iterrows():  
 fig\_bar.add\_annotation(  
 x=row['region'],  
 y=row['total\_count'],  
 text=f"{row['good\_count']}",  
 showarrow=False,  
 yshift=10  
 )  
  
# Настройка осей и отображения диаграммы  
fig\_bar.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
)  
  
# Отображение графика  
fig\_bar.show()  
  
# Фильтрация данных по ученикам, получившим оценку 3 по ОГЭ  
oge\_3 = oge\_inf\_oo\_df[oge\_inf\_oo\_df.score\_5 == 3].reset\_index(drop=True)  
  
# Подсчет количества учеников, выполнивших каждое задание (не нулевое значение) из части В, для троечников  
task\_columns = [col for col in oge\_3.columns if col.startswith('ex\_')]  
bad\_students = oge\_3[task\_columns].apply(lambda x: (x != 0).sum()).reset\_index()  
bad\_students.columns = ['task', 'students\_count']  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества троечников, выполнивших каждое задание  
fig = px.bar(  
 bad\_students,  
 x='task',  
 y='students\_count',  
 labels={'task': 'Задание', 'students\_count': 'Количество учеников'},  
 title=f'Количество троечников, выполнивших каждое задание. Всего троечников: {oge\_3.shape[0]}',  
 text='students\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Создание списка колонок заданий части C для троечников  
task\_columns\_c = [col for col in oge\_3.columns if col.startswith('ex') and col.endswith('c')]  
  
# Подсчет количества троечников, выполнивших каждое задание части C на 1, 2 и 3 балла  
results = []  
for col in task\_columns\_c:  
 for score in [1, 2, 3]:  
 count = (oge\_3[col] == score).sum()  
 results.append({'task': col, 'score': score, 'count': count})  
  
results\_df = pd.DataFrame(results)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества троечников, выполнивших каждое задание части C  
fig = px.bar(  
 results\_df,  
 x='task',  
 y='count',  
 color='score',  
 barmode='group',  
 labels={'task': 'Задание', 'count': 'Количество троечников', 'score': 'Баллы'},  
 title=f'Количество троечников, выполнивших каждое задание части C на 1, 2 и 3 балла. Всего троечников: {oge\_3.shape[0]}',  
 text='count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Подсчет количества троечников по регионам  
oge\_3\_region = oge\_3['region'].value\_counts().reset\_index()  
oge\_3\_region.columns = ['region', 'bad\_count']  
  
# Объединение данных по регионам для отображения количества троечников и общего количества учеников  
merged\_counts = pd.merge(oge\_3\_region, oge\_region, on='region').sort\_values(by='total\_count', ascending=False)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества троечников по регионам  
fig\_bar = px.bar(  
 merged\_counts,  
 x='region',  
 y='total\_count',  
 labels={'region': 'Регион', 'total\_count': 'Количество учеников по региону в общем'},  
 title='Количество троечников по регионам',  
)  
  
# Добавление аннотаций на диаграмму с количеством троечников  
for i, row in merged\_counts.iterrows():  
 fig\_bar.add\_annotation(  
 x=row['region'],  
 y=row['total\_count'],  
 text=f"{row['bad\_count']}",  
 showarrow=False,  
 yshift=10  
 )  
  
# Настройка осей и отображения диаграммы  
fig\_bar.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
)  
  
# Отображение графика  
fig\_bar.show()  
  
# Фильтрация данных по ученикам, получившим оценку 5, 4, 3 и 2 по ОГЭ  
excellent\_students = oge\_inf\_oo\_df[oge\_inf\_oo\_df.score\_5 == 5]  
good\_students = oge\_inf\_oo\_df[oge\_inf\_oo\_df.score\_5 == 4]  
bad\_students = oge\_inf\_oo\_df[oge\_inf\_oo\_df.score\_5 == 3]  
poor\_students = oge\_inf\_oo\_df[oge\_inf\_oo\_df.score\_5 == 2]  
  
# Подсчет количества учеников по категориям (отличники, хорошисты, троечники, двоечники)  
counts = {  
 'Отличники': excellent\_students.shape[0],  
 'Хорошисты': good\_students.shape[0],  
 'Троечники': bad\_students.shape[0],  
 'Двоечники': poor\_students.shape[0]  
}  
  
# Создание DataFrame для распределения учеников по категориям и их процентного соотношения  
counts\_df = pd.DataFrame(list(counts.items()), columns=['category', 'count'])  
counts\_df['percent'] = (counts\_df['count'] / oge\_inf\_oo\_df.shape[0]) \* 100  
  
# Построение круговой диаграммы (pie chart) для распределения учеников по категориям  
fig\_pie = px.pie(  
 counts\_df,  
 values='count',  
 names='category',  
 title='Распределение учеников, сдавших ОГЭ в Саратовской области по категориям',  
 labels={'category': 'Категория', 'count': 'Количество учеников'},  
 hole=0.3  
)  
  
# Настройка отображения данных на диаграмме  
fig\_pie.update\_traces(textinfo='percent+label')  
  
# Отображение графика  
fig\_pie.show()  
  
# Фильтрация данных по ученикам, получившим оценку 5 по ЕГЭ  
ege\_5 = ege\_merged\_df[ege\_merged\_df.score\_5 == 5].reset\_index(drop=True)  
  
# Получение уникальных значений регионов и количества учеников по регионам в данных ЕГЭ  
ege\_region = ege\_merged\_df[['region', 'student\_count\_region']].drop\_duplicates(subset=['region']).reset\_index(drop=True)  
  
# Создание списка колонок заданий для анализа отличников  
task\_columns = [col for col in ege\_5.columns if col.startswith('ex\_')]  
  
# Подсчет количества отличников, выполнивших каждое задание (не нулевое значение)  
excellent\_students = ege\_5[task\_columns].apply(lambda x: (x != 0).sum()).reset\_index()  
excellent\_students.columns = ['task', 'students\_count']  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества отличников, выполнивших каждое задание  
fig = px.bar(  
 excellent\_students,  
 x='task',  
 y='students\_count',  
 labels={'task': 'Задание', 'students\_count': 'Количество учеников'},  
 title=f'Количество отличников, выполнивших каждое задание. Всего отличников: {ege\_5.shape[0]}',  
 text='students\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Создание списка колонок для заданий 26 и 27  
task\_columns\_26\_27 = [col for col in ege\_5.columns if  
 col.startswith('ex') and (col.endswith('27') or col.endswith('26'))]  
  
# Подсчет количества отличников, выполнивших задания 26 и 27 на 1 и 2 балла  
results = []  
for col in task\_columns\_26\_27:  
 for score in [1, 2]:  
 count = (ege\_5[col] == score).sum()  
 results.append({'task': col, 'score': score, 'count': count})  
  
results\_df = pd.DataFrame(results)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества отличников, выполнивших задания 26 и 27  
fig = px.bar(  
 results\_df,  
 x='task',  
 y='count',  
 color='score',  
 barmode='group',  
 labels={'task': 'Задание', 'count': 'Количество отличников', 'score': 'Баллы'},  
 title=f'Количество отличников, выполнивших задание 26 и 27 на 1 и 2 балла. Всего отличников: {ege\_5.shape[0]}',  
 text='count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Подсчет количества отличников по регионам в данных ЕГЭ  
ege\_5\_region = ege\_5['region'].value\_counts().reset\_index()  
ege\_5\_region.columns = ['region', 'excellent\_count']  
  
# Объединение данных по регионам для отображения количества отличников и общего количества учеников  
merged\_counts = pd.merge(ege\_5\_region, ege\_region, on='region').sort\_values(by='student\_count\_region', ascending=False)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества отличников по регионам в ЕГЭ  
fig\_bar = px.bar(  
 merged\_counts,  
 x='region',  
 y='student\_count\_region',  
 labels={'region': 'Регион', 'student\_count\_region': 'Количество учеников по региону в общем'},  
 title='Количество отличников по регионам',  
)  
  
# Добавление аннотаций на диаграмму с количеством отличников  
for i, row in merged\_counts.iterrows():  
 fig\_bar.add\_annotation(  
 x=row['region'],  
 y=row['student\_count\_region'],  
 text=f"{row['excellent\_count']}",  
 showarrow=False,  
 yshift=10  
 )  
  
# Настройка осей и отображения диаграммы  
fig\_bar.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
)  
  
# Отображение графика  
fig\_bar.show()  
  
# Фильтрация данных по ученикам, получившим оценку 4 по ЕГЭ  
ege\_4 = ege\_merged\_df[ege\_merged\_df.score\_5 == 4]  
  
# Подсчет количества учеников, выполнивших каждое задание (не нулевое значение) для хорошистов по ЕГЭ  
task\_columns = [col for col in ege\_4.columns if col.startswith('ex\_')]  
good\_students = ege\_4[task\_columns].apply(lambda x: (x != 0).sum()).reset\_index()  
good\_students.columns = ['task', 'students\_count']  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества хорошистов, выполнивших каждое задание по ЕГЭ  
fig = px.bar(  
 good\_students,  
 x='task',  
 y='students\_count',  
 labels={'task': 'Задание', 'students\_count': 'Количество учеников'},  
 title=f'Количество хорошистов, выполнивших каждое задание. Всего хорошистов: {ege\_4.shape[0]}',  
 text='students\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Создание списка колонок для заданий 26 и 27 для хорошистов по ЕГЭ  
task\_columns\_26\_27 = [col for col in ege\_4.columns if  
 col.startswith('ex') and (col.endswith('27') or col.endswith('26'))]  
  
# Подсчет количества хорошистов, выполнивших задания 26 и 27 на 1 и 2 балла  
results = []  
for col in task\_columns\_26\_27:  
 for score in [1, 2]:  
 count = (ege\_4[col] == score).sum()  
 results.append({'task': col, 'score': score, 'count': count})  
  
results\_df = pd.DataFrame(results)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества хорошистов, выполнивших задания 26 и 27  
fig = px.bar(  
 results\_df,  
 x='task',  
 y='count',  
 color='score',  
 barmode='group',  
 labels={'task': 'Задание', 'count': 'Количество хорошистов', 'score': 'Баллы'},  
 title=f'Количество хорошистов, выполнивших задание 26 и 27 на 1 и 2 балла. Всего хорошистов: {ege\_4.shape[0]}',  
 text='count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Подсчет количества хорошистов по регионам в данных ЕГЭ  
ege\_4\_region = ege\_4['region'].value\_counts().reset\_index()  
ege\_4\_region.columns = ['region', 'good\_count']  
  
# Объединение данных по регионам для отображения количества хорошистов и общего количества учеников  
merged\_counts = pd.merge(ege\_4\_region, ege\_region, on='region').sort\_values(by='student\_count\_region', ascending=False)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества хорошистов по регионам  
fig\_bar = px.bar(  
 merged\_counts,  
 x='region',  
 y='student\_count\_region',  
 labels={'region': 'Регион', 'student\_count\_region': 'Количество учеников по региону в общем'},  
 title='Количество хорошистов по регионам',  
)  
  
# Добавление аннотаций на диаграмму с количеством хорошистов  
for i, row in merged\_counts.iterrows():  
 fig\_bar.add\_annotation(  
 x=row['region'],  
 y=row['student\_count\_region'],  
 text=f"{row['good\_count']}",  
 showarrow=False,  
 yshift=10  
 )  
  
# Настройка осей и отображения диаграммы  
fig\_bar.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
)  
  
# Отображение графика  
fig\_bar.show()  
  
# Фильтрация данных по ученикам, получившим оценку 3 по ЕГЭ  
ege\_3 = ege\_merged\_df[ege\_merged\_df.score\_5 == 3].reset\_index(drop=True)  
  
# Подсчет количества учеников, выполнивших каждое задание (не нулевое значение) для троечников по ЕГЭ  
task\_columns = [col for col in ege\_3.columns if col.startswith('ex\_')]  
bad\_students = ege\_3[task\_columns].apply(lambda x: (x != 0).sum()).reset\_index()  
bad\_students.columns = ['task', 'students\_count']  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества троечников, выполнивших каждое задание по ЕГЭ  
fig = px.bar(  
 bad\_students,  
 x='task',  
 y='students\_count',  
 labels={'task': 'Задание', 'students\_count': 'Количество учеников'},  
 title=f'Количество троечников, выполнивших каждое задание. Всего троечников: {ege\_3.shape[0]}',  
 text='students\_count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Создание списка колонок для заданий 26 и 27 для троечников по ЕГЭ  
task\_columns\_26\_27 = [col for col in ege\_3.columns if  
 col.startswith('ex') and (col.endswith('27') or col.endswith('26'))]  
  
# Подсчет количества троечников, выполнивших задания 26 и 27 на 1 и 2 балла  
results = []  
for col in task\_columns\_26\_27:  
 for score in [1, 2]:  
 count = (ege\_3[col] == score).sum()  
 results.append({'task': col, 'score': score, 'count': count})  
  
results\_df = pd.DataFrame(results)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества троечников, выполнивших задания 26 и 27  
fig = px.bar(  
 results\_df,  
 x='task',  
 y='count',  
 color='score',  
 barmode='group',  
 labels={'task': 'Задание', 'count': 'Количество троечников', 'score': 'Баллы'},  
 title=f'Количество троечников, выполнивших задание 26 и 27 на 1 и 2 балла. Всего троечников: {ege\_3.shape[0]}',  
 text='count'  
)  
  
# Настройка осей и отображения диаграммы  
fig.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
 uniformtext\_minsize=8,  
 uniformtext\_mode='hide'  
)  
fig.update\_traces(texttemplate='%{text}', textposition='inside')  
  
# Отображение графика  
fig.show()  
  
# Подсчет количества троечников по регионам в данных ЕГЭ  
ege\_3\_region = ege\_3['region'].value\_counts().reset\_index()  
ege\_3\_region.columns = ['region', 'bad\_count']  
  
# Объединение данных по регионам для отображения количества троечников и общего количества учеников  
merged\_counts = pd.merge(ege\_3\_region, ege\_region, on='region').sort\_values(by='student\_count\_region', ascending=False)  
  
# Построение столбчатой диаграммы (bar plot) для отображения количества троечников по регионам  
fig\_bar = px.bar(  
 merged\_counts,  
 x='region',  
 y='student\_count\_region',  
 labels={'region': 'Регион', 'student\_count\_region': 'Количество учеников по региону в общем'},  
 title='Количество троечников по регионам',  
)  
  
# Добавление аннотаций на диаграмму с количеством троечников  
for i, row in merged\_counts.iterrows():  
 fig\_bar.add\_annotation(  
 x=row['region'],  
 y=row['student\_count\_region'],  
 text=f"{row['bad\_count']}",  
 showarrow=False,  
 yshift=10  
 )  
  
# Настройка осей и отображения диаграммы  
fig\_bar.update\_layout(  
 yaxis={'categoryorder': 'total ascending'},  
)  
  
# Отображение графика  
fig\_bar.show()  
  
# Фильтрация данных по ученикам, получившим оценки 5, 4, 3 и 2 по ЕГЭ  
excellent\_students = ege\_merged\_df[ege\_merged\_df.score\_5 == 5]  
good\_students = ege\_merged\_df[ege\_merged\_df.score\_5 == 4]  
bad\_students = ege\_merged\_df[ege\_merged\_df.score\_5 == 3]  
poor\_students = ege\_merged\_df[ege\_merged\_df.score\_5 == 2]  
  
# Подсчет количества учеников по категориям (отличники, хорошисты, троечники, двоечники)  
counts = {  
 'Отличники': excellent\_students.shape[0],  
 'Хорошисты': good\_students.shape[0],  
 'Троечники': bad\_students.shape[0],  
 'Двоечники': poor\_students.shape[0]  
}  
  
# Создание DataFrame для распределения учеников по категориям и их процентного соотношения  
counts\_df = pd.DataFrame(list(counts.items()), columns=['category', 'count'])  
counts\_df['percent'] = (counts\_df['count'] / ege\_merged\_df.shape[0]) \* 100  
  
# Построение круговой диаграммы (pie chart) для распределения учеников по категориям  
fig\_pie = px.pie(  
 counts\_df,  
 values='count',  
 names='category',  
 title='Распределение учеников, сдавших ЕГЭ в Саратовской области по категориям',  
 labels={'category': 'Категория', 'count': 'Количество учеников'},  
 hole=0.3  
)  
  
# Настройка отображения данных на диаграмме  
fig\_pie.update\_traces(textinfo='percent+label')  
  
# Отображение графика  
fig\_pie.show()  
  
# Построение коробчатой диаграммы (box plot) для распределения первичных баллов ЕГЭ по типам образовательных учреждений  
fig = px.box(  
 ege\_merged\_df,  
 x='type\_oo',  
 y='primary\_score',  
 points='all',  
 labels={'primary\_score': 'Первичный балл ЕГЭ', 'type\_oo': 'Тип образовательного учреждения'},  
 title='Распределение первичных баллов ЕГЭ по типам образовательных учреждений'  
)  
  
# Отображение графика  
fig.show()  
  
# Построение коробчатой диаграммы (box plot) для распределения первичных баллов ОГЭ по типам образовательных учреждений  
fig = px.box(  
 oge\_inf\_oo\_df,  
 x='type\_oo',  
 y='primary\_score',  
 points='all',  
 labels={'primary\_score': 'Первичный балл ОГЭ', 'type\_oo': 'Тип образовательного учреждения'},  
 title='Распределение первичных баллов ОГЭ по типам образовательных учреждений'  
)  
  
# Отображение графика  
fig.show()  
  
# Объединение данных ЕГЭ с данными ОГЭ по кодам ОО и типам ОО  
ege\_merged\_df = ege\_merged\_df.merge(oge\_inf\_oo\_df[['code\_oo', 'type\_oo']], how='left').fillna('undefined')  
  
# Разделение данных на лицеи и гимназии, а также обычные школы  
lyceums\_and\_gymnasiums = ege\_merged\_df[ege\_merged\_df['type\_oo'].isin(['Лицей', 'Гимназия'])]  
ordinary\_schools = ege\_merged\_df[~ege\_merged\_df['type\_oo'].isin(['Лицей', 'Гимназия'])]  
  
# Рассчет среднего первичного балла ЕГЭ для лицеев и гимназий, а также для обычных школ  
mean\_lyceums\_and\_gymnasiums = lyceums\_and\_gymnasiums['primary\_score'].mean()  
mean\_ordinary\_schools = ordinary\_schools['primary\_score'].mean()  
  
# Создание DataFrame для сравнения средних баллов ЕГЭ между лицеями, гимназиями и обычными школами  
mean\_scores = pd.DataFrame({  
 'type': ['Лицеи и Гимназии', 'Обычные школы'],  
 'mean\_primary\_score': [mean\_lyceums\_and\_gymnasiums, mean\_ordinary\_schools]  
})  
  
# Построение столбчатой диаграммы (bar plot) для сравнения средних баллов ЕГЭ между лицеями, гимназиями и обычными школами  
fig = px.bar(  
 mean\_scores,  
 x='type',  
 y='mean\_primary\_score',  
 labels={'mean\_primary\_score': 'Средний первичный балл ЕГЭ', 'type': 'Тип образовательного учреждения'},  
 title='Средние первичные баллы ЕГЭ: Лицеи и Гимназии vs Обычные школы'  
)  
  
# Отображение графика  
fig.show()  
  
# Разделение данных на лицеи и гимназии, а также обычные школы для данных ОГЭ  
lyceums\_and\_gymnasiums = oge\_inf\_oo\_df[oge\_inf\_oo\_df['type\_oo'].isin(['Лицей', 'Гимназия'])]  
ordinary\_schools = oge\_inf\_oo\_df[~oge\_inf\_oo\_df['type\_oo'].isin(['Лицей', 'Гимназия'])]  
  
# Рассчет среднего первичного балла ОГЭ для лицеев и гимназий, а также для обычных школ  
mean\_lyceums\_and\_gymnasiums = lyceums\_and\_gymnasiums['primary\_score'].mean()  
mean\_ordinary\_schools = ordinary\_schools['primary\_score'].mean()  
  
# Создание DataFrame для сравнения средних баллов ОГЭ между лицеями, гимназиями и обычными школами  
mean\_scores = pd.DataFrame({  
 'type': ['Лицеи и Гимназии', 'Обычные школы'],  
 'mean\_primary\_score': [mean\_lyceums\_and\_gymnasiums, mean\_ordinary\_schools]  
})  
  
# Построение столбчатой диаграммы (bar plot) для сравнения средних баллов ОГЭ между лицеями, гимназиями и обычными школами  
fig = px.bar(  
 mean\_scores,  
 x='type',  
 y='mean\_primary\_score',  
 labels={'mean\_primary\_score': 'Средний первичный балл ОГЭ', 'type': 'Тип образовательного учреждения'},  
 title='Средние первичные баллы ОГЭ: Лицеи и Гимназии vs Обычные школы'  
)  
  
# Отображение графика  
fig.show()  
  
# Объединение данных ЕГЭ с данными о географии (расстояние до Саратова)  
ege\_merged\_df = ege\_merged\_df.merge(geo\_df[['code\_region', 'distance\_to\_saratov']], how='left').dropna()  
  
# Группировка данных ЕГЭ по расстоянию до Саратова и расчет среднего первичного балла  
grouped\_ege = ege\_merged\_df.groupby('distance\_to\_saratov')['primary\_score'].mean().reset\_index()  
  
# Построение диаграммы рассеяния (scatter plot) для отображения зависимости результатов ЕГЭ от расстояния до Саратова  
fig = px.scatter(  
 grouped\_ege,  
 x='distance\_to\_saratov',  
 y='primary\_score',  
 trendline='ols',  
 labels={'distance\_to\_saratov': 'Расстояние до Саратова (км)', 'primary\_score': 'Первичный балл ЕГЭ'},  
 title='Зависимость результатов экзаменов от расстояния до Саратова'  
)  
  
# Отображение графика  
fig.show()  
  
# Объединение данных ОГЭ с данными о географии (расстояние до Саратова)  
oge\_merged\_df = oge\_inf\_oo\_df.merge(geo\_df[['code\_region', 'distance\_to\_saratov']], how='left').dropna()  
  
# Группировка данных ОГЭ по расстоянию до Саратова и расчет среднего первичного балла  
grouped\_oge = oge\_merged\_df.groupby('distance\_to\_saratov')['primary\_score'].mean().reset\_index()  
  
# Построение диаграммы рассеяния (scatter plot) для отображения зависимости результатов ОГЭ от расстояния до Саратова  
fig = px.scatter(  
 grouped\_oge,  
 x='distance\_to\_saratov',  
 y='primary\_score',  
 trendline='ols',  
 labels={'distance\_to\_saratov': 'Расстояние до Саратова (км)', 'primary\_score': 'Первичный балл ОГЭ'},  
 title='Зависимость результатов экзаменов от расстояния до Саратова'  
)  
  
# Отображение графика  
fig.show()