**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

**Московский институт электроники и математики**

**Руководство разработчика программного обеспечения** **«Исследование количественного и национального состава субъектов РФ».**

**Студентов образовательной программы «Информатика и вычислительная техника».**

**Группы БИВ 235**

**Номер бригады: 7**

**Состав бригады:**

Папиной Анжеликой Владимировной,

Чапановой Мадиной Руслановной,

Румянцевой Кирой Сергеевной.

**Контактные данные:**

ksrumiantseva@edu.hse.ru

89875366451

Оглавление

[**Введение** 3](#_Toc169129906)

[**Техническое задание** 3](#_Toc169129907)

[**Требования к автоматизированному рабочему месту** 3](#_Toc169129908)

[**Технические требования к автоматизированному рабочему месту:** 3](#_Toc169129909)

[**Работа с данными** 4](#_Toc169129910)

[**Версии языка и используемые библиотеки** 6](#_Toc169129911)

[**Функции построения графиков** 7](#_Toc169129912)

[**Функции построения отчетов** 14](#_Toc169129913)

[**Интерфейс** 18](#_Toc169129914)

[**Листинг основного скрипта** 20](#_Toc169129915)

[**Листинг модулей графиков** 24](#_Toc169129916)

[**Листинги отчетов** 40](#_Toc169129917)

### **Введение**

Настоящее руководство разработчика распространяется на программное обеспечение (далее ПО) «Исследование количественного и национального состава субъектов РФ» версии 1.0. Данное руководство включает в себя информацию о требованиях к характеристикам компьютера и операционной системы, указания версий языка и библиотек, использованных при разработке приложения, описание архитектуры приложения, описание структуры каталогов, листинг основного скрипта и всех модулей.

## **Техническое задание**

Разработать программное обеспечение позволяющее анализировать данные, создавать графическое представление информации и отчеты.

## **Требования к автоматизированному рабочему месту**

Для получения качественного результата с использованием программного обеспечения «Исследование количественного и национального состава субъектов РФ», необходимо автоматизированное рабочее место с предустановленной операционной системой (Windows или Linux).

## **Технические требования к автоматизированному рабочему месту:**

Процессор: 1 гигагерц (ГГц) или быстрее с двумя или более ядрами на

совместимом 64-разрядном процессоре;

ОЗУ: 4 гигабайта (ГБ) или больше;

ПЗУ: 1 ГБ или больше свободного места на диске;

Графическая карта: совместим с DirectX 12 или более поздней версии с драйвером

WDDM 2.0;

Дисплей: дисплей с высоким разрешением (720p), монитор с разрешением 9

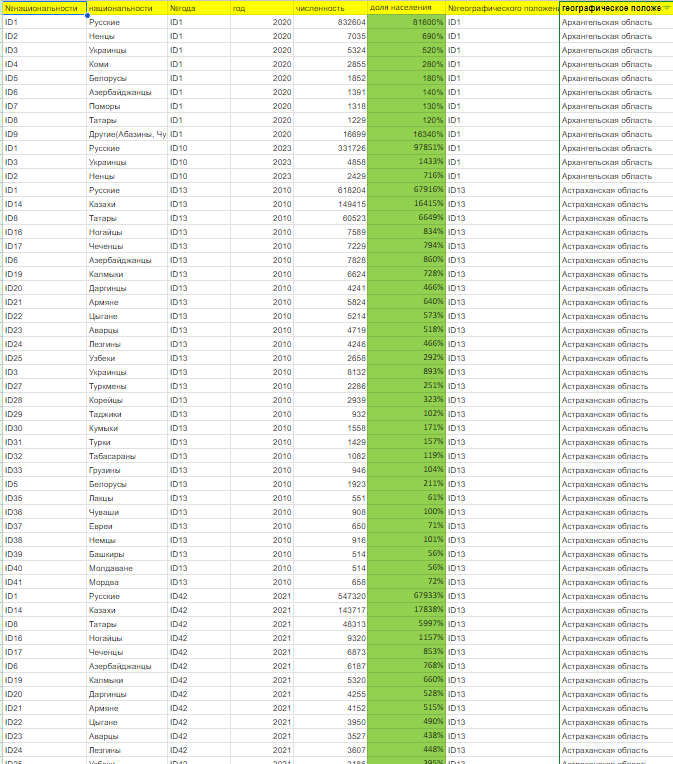
дюймов или более, 8 бит на цветовый канал;

Устройства ввода: клавиатура, компьютерная мышь.

## **Работа с данными**

Данные для реализации данного проекта были собраны вручную из официальных источников Росстата. Дополнены информацией, размещённых на сайтах со статистическими исследованиями численностей национальностей.

Первая часть таблицы представлена на Рис. 1.

Рис. 1.

Она включает в себя данные о национальности, численности, доле населения, годе и географическом положении. Для упрощения дальнейшей работы с датасетом и анализом введены следующие параметры: ID национальности, ID года, ID географического положения.

Вторая часть таблицы представлена на рис. 2.

Рис. 2.

Она включает в себя данные о субъектах Российской Федерации, периоде и объеме населения.

## **Версии языка и используемые библиотеки**

Программное обеспечение «Исследование количественного и национального состава субъектов РФ» версии 1.0. разработано на языке Python. Применена версия Python 3.12.4.

Для создания графиков используются такие библиотеки как numpy и matplotlib.

Для написания графического интерфейса используется библиотека PyQt6.

Для работы с базой данных используется библиотека pandas.

Для совместной работы разных функций кода используется библиотека sys

## **Функции построения графиков**

График Box-and-Whiskers

def create\_report(self):

        # Получаем текущее географическое положение из выпадающего списка

        geografic\_position = self.comboBox.currentText()

        # Фильтруем данные по выбранному географическому положению

        data = self.df[self.df['географическое положение'] == geografic\_position]

        # Группируем данные по национальностям и суммируем численность

        report\_df = data.groupby('национальности')['численность'].sum().reset\_index()

        # Сортируем данные по национальностям

        report\_df = report\_df.sort\_values(by='национальности')

        # Обновляем данные в модели для отображения в таблице

        self.model.\_data = report\_df

        self.model.layoutChanged.emit()

        # Очищаем фигуру перед построением нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Строим Box-and-Whiskers график для данных по национальностям

        ax.boxplot([data[data['национальности'] == nationality]['численность'] for nationality in report\_df['национальности']], patch\_artist=True, widths=0.7)

        ax.set\_xlabel('Национальность')

        ax.set\_ylabel('Численность')

        ax.set\_title(f'Box-and-Whiskers график для города {geografic\_position}.')

        ax.set\_xticks(range(1, len(report\_df) + 1))

        ax.set\_xticklabels(report\_df['национальности'], rotation=90)

        ax.grid(True)

        self.canvas.draw()

Ф. 1 (график Box and Whiskers)

Программа составляет отчет по необходимым данным, в данном случае по региону, и на основе этого отчета создает график. Данный отчет можно редактировать, и благодаря этому удалять и создавать отчет по выбранным национальностям.

def update\_plot(self):

        # Очистка фигуры перед построением нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Сортировка данных по национальностям

        sorted\_data = self.model.\_data.sort\_values(by='национальности')

        # Построение Box-and-Whiskers графика для всех национальностей

        ax.boxplot([self.df[(self.df['географическое положение'] == self.comboBox.currentText()) & (self.df['национальности'] == nationality)]['численность'] for nationality in sorted\_data['национальности']], patch\_artist=True, widths=0.7)

        ax.set\_xlabel('Национальность')

        ax.set\_ylabel('Численность')

        ax.set\_title(f'Box-and-Whiskers график для города {self.comboBox.currentText()}')

        ax.set\_xticks(range(1, len(sorted\_data) + 1))

        ax.set\_xticklabels(sorted\_data['национальности'], rotation=90)

        ax.grid(True)

        self.canvas.draw()

Ф. 2(функция для удаления национальности и обновления графика)

def update\_plot\_2(self):

        # Очистка фигуры перед построением нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Получение выбранных национальностей из таблицы

        selected\_nationalities = [index.data() for index in self.tableView.selectionModel().selectedRows(column=0)]

        if selected\_nationalities:

            # Построение Box-and-Whiskers графика для выбранных национальностей

            ax.boxplot([self.df[(self.df['географическое положение'] == self.comboBox.currentText()) & (self.df['национальности'] == nationality)]['численность'] for nationality in selected\_nationalities], patch\_artist=True, widths=0.7)

            ax.set\_xlabel('Национальность')

            ax.set\_ylabel('Численность')

            ax.set\_title(f'Box-and-Whiskers график для города {self.comboBox.currentText()}')

            ax.set\_xticks(range(1, len(selected\_nationalities) + 1))

            ax.set\_xticklabels(selected\_nationalities, rotation=90)

            ax.grid(True)

            self.canvas.draw()

        else:

            # В случае отсутствия выбранных национальностей выводим сообщение

            ax.axis('off')

            ax.text(0.5, 0.5, "Выберите национальность в таблице", horizontalalignment='center', verticalalignment='center', transform=ax.transAxes)

            self.canvas.draw()

Ф. 3(функция для создания графика по выбранным национальностям)

Категоризированная гистограмма

def create\_report(self):

        # Получаем текущее географическое положение и выбранные года из выпадающих списков

        geografic\_position = self.comboBox.currentText()

        year1 = int(self.comboBox\_2.currentText())

        year2 = int(self.comboBox\_3.currentText())

        # Загружаем данные из Excel файла и создаем DataFrame

        data = pd.read\_excel('python.xlsx')

        df = pd.DataFrame(data)

        # Фильтруем данные по выбранному географическому положению

        df = df[df['географическое положение'] == geografic\_position]

        # Фильтруем данные по выбранным годам

        df\_year1 = df[(df['год'] == year1))]

        df\_year2 = df[(df['год'] == year2))]

        # Получаем уникальные национальности

        nationalities = set(df\_year1['национальности']).union(set(df\_year2['национальности']))

        # Создаем объединенный DataFrame

        df\_combined = pd.DataFrame({'национальности': list(nationalities)})

        df\_combined = df\_combined.merge(df\_year1, on='национальности', how='left')

        df\_combined = df\_combined.merge(df\_year2, on='национальности', how='left')

        # Создаем модель данных и устанавливаем ее для отображения в таблице

        self.model = PandasModel(df\_combined[['национальности', 'численность\_x', 'численность\_y']])

        self.tableView.setModel(self.model)

        # Параметры для построения столбчатой диаграммы

        bar\_width = 0.35

        index = np.arange(len(df\_combined['национальности']))

        # Очищаем предыдущий график и строим новую столбчатую диаграмму

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        ax.clear()

        ax.bar(index, df\_combined['численность\_x'].fillna(0), bar\_width, color='blue', label=str(year1))

        ax.bar(index + bar\_width, df\_combined['численность\_y'].fillna(0), bar\_width, color='red', label=str(year2))

        # Настройки осей и заголовка графика

        ax.set\_xlabel('Национальности')

        ax.set\_ylabel('Численность')

        ax.set\_title(f'Численность самых крупных национальностей в {year1} и {year2} годах')

        ax.set\_xticks(index + bar\_width / 2)

        ax.set\_xticklabels(df\_combined['национальности'], rotation=90)

        ax.legend()

        # Обновляем график на холсте

        self.canvas.draw()

Ф. 4 (Функция для создания категоризированной гистограммы)

Программа пишет отчет, создает DataFrame по выбранному региону и годам. Далее на основе этого отчет создает график, это позволяет нам удалять или создавать график по выбранным регионам. Функции работают на приведенном подобии выше. (См. ф2. и ф3)

Круговая диаграмма(Пайчарт)

def create\_report(self):

        # Функция для создания отчета, вызывает функцию grafic4 для построения круговой диаграммы

        print("Функция create\_report вызвана")

        geografic\_position = self.comboBox.currentText()

        year = int(self.comboBox\_2.currentText())  # Преобразуем год из строки в число

        self.grafic4(geografic\_position, year)

    def grafic4(self, geografic\_position, year):

        """

        Функция для построения круговой диаграммы на основе отфильтрованных данных

        :param geografic\_position: выбранное географическое положение

        :param year: выбранный год

        """

        data = self.df

        # Фильтрация данных по географическому положению и году

        filtered\_data = data[(data["географическое положение"] == geografic\_position) & (data["год"] == year)]

        colors = plt.cm.tab20b.colors

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Построение круговой диаграммы

        wedges, texts = ax.pie(filtered\_data["доля населения"], colors=colors, startangle=140)

        ax.axis('equal')

        ax.set\_title(f"Круговая диаграмма национальностей по численности за {year} год")

        # Добавление легенды к диаграмме

        ax.legend(wedges, filtered\_data["национальности"], title="Национальности", loc="upper right", bbox\_to\_anchor=(1.05, 1))

        self.canvas.draw()

    ##### Отчет

    def create\_report\_1(self):

        # Функция для создания отчета 1, отображает данные в виде табличного представления

        # Использует выбранное географическое положение и год для формирования отчета

        geografic\_position = self.comboBox.currentText()

        year = int(self.comboBox\_2.currentText())

        filtered\_data = self.df[(self.df["географическое положение"] == geografic\_position) & (self.df["год"] == year)]

        report\_df = filtered\_data[["национальности", "доля населения"]].copy()

        report\_df["доля населения"] = report\_df["доля населения"].apply(lambda x: "{:.2%}".format((x/100)\*100))

        report\_df.rename(columns={"доля населения": "численность"}, inplace=True)

        self.model = PandasModel(report\_df)

        self.tableView.setModel(self.model)

Ф. 5 (Круговая диаграмма)

Программа создает отчет и на его основе составляет круговую диаграмму. Вводят регион и год. Программа использует национальности и доли населения для вывода информации. Также crete\_report\_1 предоставляет отчет о долях, выведенных на пайчарт национальностей.

Линейный график

class MplCanvas(FigureCanvas):

    def \_\_init\_\_(self, parent=None, width=7.6, height=5.8, dpi=100):

        # Создание фигуры и осей для графика

        fig, self.ax = plt.subplots(figsize=(width, height), dpi=dpi)

        super(MplCanvas, self).\_\_init\_\_(fig)

        self.setParent(parent)

    def plot\_data(self, oblasts, sheet\_name="Лист1"):

        # Очистка текущего графика

        self.ax.clear()

        # Загрузка данных из Excel файла

        data = pd.read\_excel('python.xlsx', sheet\_name=sheet\_name)

        df = pd.DataFrame(data)

        # Построение графиков для выбранных регионов

        for oblast in oblasts:

            # Фильтрация данных по выбранному региону

            df\_oblast = df[df['субъекты'] == oblast]

            # Преобразование столбца 'население' в числовой формат

            df\_oblast['население'] = pd.to\_numeric(df\_oblast['население'], errors='coerce')

            # Удаление строк с отсутствующими значениями в столбце 'население'

            df\_oblast = df\_oblast.dropna(subset=['население'])

            # Построение графика населения по годам для выбранного региона

            self.ax.plot(df\_oblast['период'], df\_oblast['население'], label=oblast)

        # Настройка подписей осей и заголовка графика

        self.ax.set\_xlabel('Год')

        self.ax.set\_ylabel('Количество населения')

        self.ax.set\_title('Сравнение регионов по населению')

        # Форматирование значений осей

        self.ax.ticklabel\_format(style='plain')

        # Добавление легенды к графику

        self.ax.legend()

        # Обновление отображения графика

        self.figure.canvas.draw()

Ф. 6 (линейный график)

Вызывается функция, которая собирает все введенные данные пользователем. Собирается список всех выбранных регионов. Создается экземпляр класса MPLCanvas, который представляет собой виджет для отображения графиков. Для каждого выбранного региона строится график населения по годам.

Категоризированная диаграмма рассеивания

def create\_report(self):

        # Получение выбранного географического положения из комбо-бокса

        geografic\_position = self.comboBox.currentText()

        # Фильтрация данных по выбранному географическому положению

        data = self.df[self.df['географическое положение'] == geografic\_position]

        # Группировка данных по национальности, доле населения и году, и подсчет суммы

        report\_df = data.groupby(['национальности', 'доля населения', 'год']).sum().reset\_index()

        # Преобразование столбца 'доля населения' в процентный формат

        report\_df["доля населения"] = report\_df["доля населения"].apply(lambda x: "{:.2%}".format((x/1000)))

        # Сортировка данных по национальности

        report\_df = report\_df.sort\_values(by='национальности')

        # Отображение данных с наибольшей долей населения

        report\_df = report\_df.sort\_values(by='доля населения')

        # Обновление модели данных для отображения в таблице

        self.model.\_data = report\_df

        self.model.layoutChanged.emit()

        # Очистка текущего графика и добавление нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Построение диаграммы рассеивания

        scatter = ax.scatter(report\_df['доля населения'], report\_df['национальности'], c=report\_df['год'])

        ax.set\_xlabel('Доля населения')

        ax.set\_ylabel('Национальность')

        ax.set\_title('Категоризированная диаграмма рассеивания')

        ax.xaxis.set\_tick\_params(rotation=90)

        ax.grid(True)

        # Добавление легенды к графику

        legend = ax.legend(\*scatter.legend\_elements(), title="Годы")

        ax.add\_artist(legend)

        # Скрытие лишних столбцов в таблице

        self.tableView.setColumnHidden(3, True)

        self.tableView.setColumnHidden(4, True)

        self.tableView.setColumnHidden(5, True)

        self.tableView.setColumnHidden(6, True)

        self.tableView.setColumnHidden(7, True)

        # Обновление отображения графика

        self.canvas.draw()

Ф. 7 (Категоризированная диаграмма рассеивания)

Пользователь выбирает регион. Составляется отчет по выбранным национальностям и выводится на экран, по этому отчету рисуется график, для которого необходимы такие данные как национальность, доля населения и год.

Пользователь может удалять и создавать график по своим предпочтениям благодаря созданию графика на основе создаваемого нового отчета. Функции реализации подобны ранее представленным функциям. (См. Ф. 2 и Ф. 3)

## **Функции построения отчетов**

Отчет о численности для национальности

def chisl(self, df: pd.DataFrame, cndname: str, cndval: str, intname: str, intval: int) -> pd.DataFrame:

        """

        Функция фильтрует данные DataFrame по заданным условиям и возвращает новый DataFrame.

        Параметры:

        - df: pd.DataFrame - исходный DataFrame с данными

        - cndname: str - название столбца для условия фильтрации

        - cndval: str - значение условия фильтрации

        - intname: str - название столбца для сравнения с числовым значением

        - intval: int - числовое значение для фильтрации

        Возвращает:

        - new\_df: pd.DataFrame - новый DataFrame после фильтрации

        """

        # Фильтрация данных по заданным условиям

        new\_df = df[(df[cndname] == cndval) & (df[intname] < intval)][['численность', 'национальности', 'географическое положение', 'год']]

        # Возврат нового DataFrame после фильтрации

        return new\_df

Ф. 8 (Отчет о численности для национальности)

Программа запрашивает национальности и просит ввести максимальное число для населения. Данные далее фильтруются и выводится отчет о численности, самой национальности, его регионе и годе.

Отчет о национальном составе региона в определенный год

def chisl(self, df: pd.DataFrame, cndname\_1: str, cndval\_1: str, cndname\_2: str, cndval\_2: str) -> pd.DataFrame:

        """

        Функция фильтрует данные DataFrame по двум заданным условиям и возвращает новый DataFrame.

        Параметры:

        - df: pd.DataFrame - исходный DataFrame с данными

        - cndname\_1: str - название первого столбца для условия фильтрации

        - cndval\_1: str - значение первого условия фильтрации

        - cndname\_2: str - название второго столбца для условия фильтрации

        - cndval\_2: str - значение второго условия фильтрации

        Возвращает:

        - new\_df: pd.DataFrame - новый DataFrame после фильтрации

        """

        # Фильтрация данных по двум заданным условиям

        new\_df = df[(df[cndname\_1] == cndval\_1) & (df[cndname\_2].astype(str) == cndval\_2)][['национальности', 'год', 'географическое положение', 'численность']]

        # Возврат нового DataFrame после фильтрации

        return new\_df

Ф. 8(Отчет о национальном составе региона в определенный год)

Программа на ввод запрашивает регион и год. Далее строит отчет о национальности, непосредственно годе, субъекте и численности.

Численный состав национальности в пределах

def report4(self, df: pd.DataFrame, cndname\_1: str, intname: int, cndname\_2: str, cndval\_2: int) -> pd.DataFrame:

        """

        Функция формирует отчет на основе данных DataFrame, фильтруя их по заданным условиям.

        Параметры:

        - df: pd.DataFrame - исходный DataFrame с данными

        - cndname\_1: str - название первого столбца для фильтрации по значению больше или равно intname

        - intname: int - числовое значение для фильтрации по cndname\_1

        - cndname\_2: str - название второго столбца для фильтрации по значению меньше или равно cndval\_2

        - cndval\_2: int - числовое значение для фильтрации по cndname\_2

        Возвращает:

        - new\_df: pd.DataFrame - новый DataFrame после применения фильтрации

        """

        # Фильтрация данных по заданным условиям

        new\_df = df[(df[cndname\_1] >= intname) & (df[cndname\_2] <= int(cndval\_2))][['численность', 'национальности', 'географическое положение', 'год']]

        # Возврат нового DataFrame после фильтрации для формирования отчета

        return new\_df

Ф. 9 (функция по созданию отчета о численном составе)

На вход программа запрашивает минимальное и максимальное число. Далее пользователю выводится отчет с непосредственно численностью, национальностью, субъектом и годом.

Сводная таблица количества населения субъектов РФ

class MyApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Загрузка пользовательского интерфейса из файла svodntable2.ui

        loadUi("otcheti\svodntable2.ui", self)

        # Чтение данных из файла Excel 'python.xlsx' из листа с индексом 1

        df = pd.read\_excel("python.xlsx", sheet\_name=1)

        # Преобразование столбцов 'население' и 'период' в числовой формат

        df['население'] = pd.to\_numeric(df['население'], errors='coerce').fillna(0).astype(int)

        df['период'] = pd.to\_numeric(df['период'], errors='coerce').fillna(0).astype(int)

        # Создание сводной таблицы на основе данных DataFrame

        pivot\_table = pd.pivot\_table(df, values='население', index='субъекты', columns='период')

        # Создание модели данных PandasModel на основе сводной таблицы

        model = PandasModel(pivot\_table)

        # Установка модели данных для отображения в tableView

        self.tableView.setModel(model)

        # Установка режима изменения размеров колонок таблицы по содержимому

        self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.ResizeToContents)

Ф. 10 (Сводная таблица регионов)

Пользователю выводится отчет о количестве населения для каждого региона по разным годам.

Сводная таблица национального состава РФ

class MyApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Загрузка пользовательского интерфейса из файла svodntable.ui

        loadUi("otcheti\svodntable.ui", self)

        # Чтение данных из файла Excel 'python.xlsx'

        df = pd.read\_excel("python.xlsx")

        # Преобразование столбцов 'численность' и 'год' в числовой формат

        df['численность'] = pd.to\_numeric(df['численность'], errors='coerce').fillna(0).astype(int)

        df['год'] = pd.to\_numeric(df['год'], errors='coerce').fillna(0).astype(int)

        # Создание сводной таблицы на основе данных DataFrame

        pivot\_table = pd.pivot\_table(df, values='численность', index='национальности', columns='год', aggfunc='sum')

        # Создание модели данных PandasModel на основе сводной таблицы

        model = PandasModel(pivot\_table)

        # Установка модели данных для отображения в tableView

        self.tableView.setModel(model)

        # Установка режима изменения размеров колонок таблицы по содержимому

        self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.ResizeToContents)

Ф. 11 (Функция по созданию сводной таблицы)

Пользователю выводится таблица о каждой национальности в разные года. Сколько их было в сумме по данным Росстата в России по разным годам.

### **Интерфейс**

Интерфейс реализован с помощью библиотеки PyQt6. Страницы собирались в редакторе Qt Designer. Обращение ко всем созданные виджетам осуществляется обращением к папкам xxxx.ui.

В интерфейсе реализованы функции проверки вводимых данных и способность редактировать графики.

 try:

            # Попытка преобразовать текущий текст из выпадающего списка в целое число для минимальной и максимальной численности населения

            min\_population = int(self.comboBox.currentText())

            max\_population = int(self.comboBox\_2.currentText())

            # Проверка, что минимальное значение населения не больше максимального

            if min\_population > max\_population:

                # Вывод предупреждения об ошибке, если условие не выполняется

                QMessageBox.warning(self, "Ошибка", "Минимальное значение не может быть больше Максимального")

                return

        except ValueError:

            # В случае исключения ValueError (некорректные данные) выводится предупреждение

            QMessageBox.warning(self, "Выберите корректные данные")

            return

Часть программы. 1 (проверка того, что количество выбрано корректно)

Данная функция проверяет что минимальное и максимальное значения соответствуют свои определениям.

Необходима для отчета о численном составе национальностей в пределах.

 # Проверка на наличие корректного значения для количества

        if self.kol is None:

            # Вывод предупреждения, если количества нет или некорректное

            QMessageBox.warning(self, "Предупреждение", "Введите корректное значение для количества")

            return

Часть программы 2. (Проверка на непустоту строки)

Проверяет что введены корректные числовые данные.

Необходим для отчета для численности национальности.

Имеется навигационная панель для пользователя.

Navigation Toolbar добавлена к холсту Figure Canvas, благодаря чему пользователь может масштабировать, перемещать и использовать другие инструменты.

Программа имеет основной файл face.py, где имеются пути к каждому Widgety. Тоесть к каждой программе графика или отчета.

За связь между ними отвечают функции, которые создают и настраивают окна с графиками.

def opeт\_first\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_report1])

def opeт\_second\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_report2])

def opeт\_3\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_report3])

def opeт\_4\_report():

    import subprocess

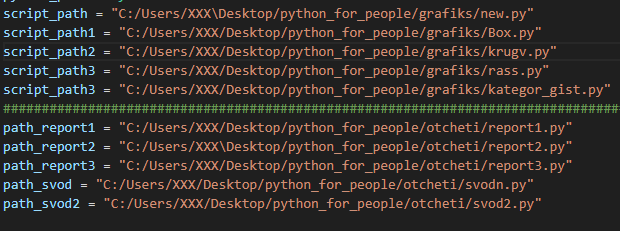
    subprocess.Popen([python\_path, path\_svod])

def opeт\_5\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_svod2])

Ф. 12 (Функции для окон)



Часть программы 3. (пути ко всем виджетам)

## **Листинг основного скрипта**

from PyQt6 import uic

from PyQt6.QtWidgets import QApplication, QMainWindow,  QDialog, QPushButton, QVBoxLayout, QLineEdit, QLabel

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import sys

Form, Window = uic.loadUiType("pril.ui")

GraficsForm, GraficsWindow = uic.loadUiType("grafics.ui")

ReportsForm, ReportsWindow = uic.loadUiType("reports.ui")

app = QApplication([])

window = Window()

form = Form()

form.setupUi(window)

grafics\_window = None

reports\_window = None

python\_path = sys.executable

script\_path = "C:/Users/XXX\Desktop/python\_for\_people/grafiks/new.py"

script\_path1 = "C:/Users/XXX/Desktop/python\_for\_people/grafiks/Box.py"

script\_path2 = "C:/Users/XXX/Desktop/python\_for\_people/grafiks/krugv.py"

script\_path3 = "C:/Users/XXX/Desktop/python\_for\_people/grafiks/rass.py"

script\_path3 = "C:/Users/XXX/Desktop/python\_for\_people/grafiks/kategor\_gist.py"

################################################################################################################################################################################################

path\_report1 = "C:/Users/XXX/Desktop/python\_for\_people/otcheti/report1.py"

path\_report2 = "C:/Users/XXX\Desktop/python\_for\_people/otcheti/report2.py"

path\_report3 = "C:/Users/XXX/Desktop/python\_for\_people/otcheti/report3.py"

path\_svod = "C:/Users/XXX/Desktop/python\_for\_people/otcheti/svodn.py"

path\_svod2 = "C:/Users/XXX/Desktop/python\_for\_people/otcheti/svod2.py"

########################################################################################################################################################################################################################

##################################################################################################################################################################################################################

def create\_second\_window():

    global reports\_window

    reports\_window = ReportsWindow()

    reports\_form = ReportsForm()

    reports\_form.setupUi(reports\_window)

    button11 = QPushButton("Отчет о численности для национальности")

    button11.setObjectName("pushbutton\_report1")

    button11.clicked.connect(opeт\_first\_report)

    button12 = QPushButton("Отчет о национальном составе региона в определенный год")

    button12.setObjectName("pushbutton\_report2")

    button12.clicked.connect(opeт\_second\_report)

    button13 = QPushButton("Численный состав национальности в пределах")

    button13.setObjectName("pushbutton\_report3")

    button13.clicked.connect(opeт\_3\_report)

    button14 = QPushButton("Сводная таблица национального состава РФ")

    button14.setObjectName("pushbutton\_report4")

    button14.clicked.connect(opeт\_4\_report)

    button15 = QPushButton("Сводная таблица количества населения субъектов РФ")

    button15.setObjectName("pushbutton\_report5")

    button15.clicked.connect(opeт\_5\_report)

    layout = reports\_form.verticalLayout

    layout.addWidget(button11)

    layout.addWidget(button12)

    layout.addWidget(button13)

    layout.addWidget(button14)

    layout.addWidget(button15)

    reports\_window.show()

def opeт\_first\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_report1])

def opeт\_second\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_report2])

def opeт\_3\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_report3])

def opeт\_4\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_svod])

def opeт\_5\_report():

    import subprocess

    subprocess.Popen([python\_path, path\_svod2])

###################################################################################################################################################################################################################

def create\_new\_window():

    global grafics\_window

    grafics\_window = GraficsWindow()

    grafics\_form = GraficsForm()

    grafics\_form.setupUi(grafics\_window)

    button0 = QPushButton("Линейный график")

    button0.setObjectName("pushButton\_line")

    button0.clicked.connect(open\_new\_window)

    button1 = QPushButton("График Bow-and-Whisker")

    button1.setObjectName("pushButton\_line0")

    button1.clicked.connect(open\_new\_window\_1)

    button22 = QPushButton("Круговая диаграмма")

    button22.setObjectName("pushButton\_grafic1")

    button22.clicked.connect(open\_new\_window\_2)

    button3 = QPushButton("Категоризированная диаграмма рассеивания")

    button3.setObjectName("pushButton\_grafic2")

    button3.clicked.connect(open\_new\_window\_3)

    button2 = QPushButton("Категоризированная гистограмма")

    button2.setObjectName("pushButton\_grafic1")

    button2.clicked.connect(create\_klaster2)

    layout = grafics\_form.verticalLayout

    layout.addWidget(button0)

    layout.addWidget(button1)

    layout.addWidget(button22)

    layout.addWidget(button3)

    layout.addWidget(button2)

    grafics\_window.show()

def open\_new\_window():

    import subprocess

    subprocess.Popen([python\_path, script\_path])

def open\_new\_window\_1():

    import subprocess

    subprocess.Popen([python\_path, script\_path1])

def open\_new\_window\_2():

    import subprocess

    subprocess.Popen([python\_path, script\_path2])

def open\_new\_window\_3():

    import subprocess

    subprocess.Popen([python\_path, script\_path3])

form.pushButton.clicked.connect(create\_new\_window)

form.toolButton\_2.clicked.connect(create\_second\_window)

window.show()

app.exec()

Программа 1

### **Листинг модулей графиков**

import sys

import pandas as pd

import matplotlib.pyplot as plt

from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QComboBox, QPushButton, QAbstractItemView

from PyQt6.QtCore import Qt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.backends.backend\_qt5agg import NavigationToolbar2QT as NavigationToolbar

from matplotlib.figure import Figure

from PyQt6.QtCore import QAbstractTableModel, QVariant, QModelIndex

from PyQt6.uic import loadUi

# Класс для модели данных Pandas

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent=QModelIndex()):

        return len(self.\_data.index)

    def columnCount(self, parent=QModelIndex()):

        return len(self.\_data.columns)

    def data(self, index, role=Qt.ItemDataRole.DisplayRole):

        if role == Qt.ItemDataRole.DisplayRole:

            return str(self.\_data.iloc[index.row(), index.column()])

        return QVariant()

# Главное окно приложения

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        loadUi("grafiks/Box.ui", self)  # Загрузка пользовательского интерфейса

        self.df = pd.read\_excel("python.xlsx")  # Загрузка данных из Excel файла

        # Добавление уникальных значений географического положения в комбо-бокс

        self.items1 = sorted(self.df["географическое положение"].astype(str).unique().tolist())

        self.comboBox.addItems(self.items1)

        # Подключение кнопок к соответствующим методам

        self.pushButton.clicked.connect(self.create\_report)

        self.pushButton\_2.clicked.connect(self.remove\_nationality)

        self.pushButton\_3.clicked.connect(self.remove\_nationality\_2)

        # Создание модели данных для отчета и привязка к таблице

        self.model = PandasModel(pd.DataFrame(columns=['национальности', 'численность']))

        self.tableView.setModel(self.model)

        self.tableView.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)

        # Создание графического окна и добавление элементов

        self.figure = Figure()

        self.canvas = FigureCanvas(self.figure)

        layout = QVBoxLayout()

        layout.addWidget(self.canvas)

        self.toolbar = NavigationToolbar(self.canvas, self)

        layout.addWidget(self.toolbar)

        self.graphicsView.setLayout(layout)

    # Метод для удаления строки с выбранной национальностью из таблицы и обновления графика

    def remove\_nationality(self):

        selected\_rows = self.tableView.selectionModel().selectedRows()

        if selected\_rows:

            row = selected\_rows[0].row()

            nationality = self.model.\_data.iloc[row]['национальности']

            self.model.\_data = self.model.\_data[self.model.\_data['национальности'] != nationality]

            self.model.layoutChanged.emit()

            self.update\_plot()

    # Метод для выбора строк для построения графика

    def remove\_nationality\_2(self):

        selected\_rows = self.tableView.selectionModel().selectedRows()

        if selected\_rows:

            row = selected\_rows[0].row()

            nationality = self.model.\_data.iloc[row]['национальности']

            self.model.\_data = self.model.\_data[self.model.\_data['национальности'] != nationality]

            self.model.layoutChanged.emit()

            self.update\_plot\_2()

    def update\_plot\_2(self):

        # Очистка фигуры перед построением нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Получение выбранных национальностей из таблицы

        selected\_nationalities = [index.data() for index in self.tableView.selectionModel().selectedRows(column=0)]

        if selected\_nationalities:

            # Построение Box-and-Whiskers графика для выбранных национальностей

            ax.boxplot([self.df[(self.df['географическое положение'] == self.comboBox.currentText()) & (self.df['национальности'] == nationality)]['численность'] for nationality in selected\_nationalities], patch\_artist=True, widths=0.7)

            ax.set\_xlabel('Национальность')

            ax.set\_ylabel('Численность')

            ax.set\_title(f'Box-and-Whiskers график для города {self.comboBox.currentText()}')

            ax.set\_xticks(range(1, len(selected\_nationalities) + 1))

            ax.set\_xticklabels(selected\_nationalities, rotation=90)

            ax.grid(True)

            self.canvas.draw()

        else:

            # В случае отсутствия выбранных национальностей выводим сообщение

            ax.axis('off')

            ax.text(0.5, 0.5, "Выберите национальность в таблице", horizontalalignment='center', verticalalignment='center', transform=ax.transAxes)

            self.canvas.draw()

    def update\_plot(self):

        # Очистка фигуры перед построением нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Сортировка данных по национальностям

        sorted\_data = self.model.\_data.sort\_values(by='национальности')

        # Построение Box-and-Whiskers графика для всех национальностей

        ax.boxplot([self.df[(self.df['географическое положение'] == self.comboBox.currentText()) & (self.df['национальности'] == nationality)]['численность'] for nationality in sorted\_data['национальности']], patch\_artist=True, widths=0.7)

        ax.set\_xlabel('Национальность')

        ax.set\_ylabel('Численность')

        ax.set\_title(f'Box-and-Whiskers график для города {self.comboBox.currentText()}')

        ax.set\_xticks(range(1, len(sorted\_data) + 1))

        ax.set\_xticklabels(sorted\_data['национальности'], rotation=90)

        ax.grid(True)

        self.canvas.draw()

    def create\_report(self):

        # Получаем текущее географическое положение из выпадающего списка

        geografic\_position = self.comboBox.currentText()

        # Фильтруем данные по выбранному географическому положению

        data = self.df[self.df['географическое положение'] == geografic\_position]

        # Группируем данные по национальностям и суммируем численность

        report\_df = data.groupby('национальности')['численность'].sum().reset\_index()

        # Сортируем данные по национальностям

        report\_df = report\_df.sort\_values(by='национальности')

        # Обновляем данные в модели для отображения в таблице

        self.model.\_data = report\_df

        self.model.layoutChanged.emit()

        # Очищаем фигуру перед построением нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Строим Box-and-Whiskers график для данных по национальностям

        ax.boxplot([data[data['национальности'] == nationality]['численность'] for nationality in report\_df['национальности']], patch\_artist=True, widths=0.7)

        ax.set\_xlabel('Национальность')

        ax.set\_ylabel('Численность')

        ax.set\_title(f'Box-and-Whiskers график для города {geografic\_position}.')

        ax.set\_xticks(range(1, len(report\_df) + 1))

        ax.set\_xticklabels(report\_df['национальности'], rotation=90)

        ax.grid(True)

        self.canvas.draw()

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MainWindow()

    window.show()

    sys.exit(app.exec())

Программа 2(Box-and-Whisker)

import sys

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QComboBox, QPushButton, QAbstractItemView, QMessageBox

from PyQt6.QtCore import Qt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.backends.backend\_qt5agg import NavigationToolbar2QT as NavigationToolbar

from matplotlib.figure import Figure

from PyQt6.QtCore import QAbstractTableModel, QVariant, QModelIndex

from PyQt6.uic import loadUi

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent=QModelIndex()):

        return len(self.\_data.index)

    def columnCount(self, parent=QModelIndex()):

        return len(self.\_data.columns)

    def data(self, index, role=Qt.ItemDataRole.DisplayRole):

        if role == Qt.ItemDataRole.DisplayRole:

            return str(self.\_data.iloc[index.row(), index.column()])

        return QVariant()

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        loadUi("grafiks/Kategor.ui", self)

        self.df = pd.read\_excel("python.xlsx")

        self.items1 = sorted(self.df["географическое положение"].astype(str).unique().tolist())

        self.comboBox.addItems(self.items1)

        self.comboBox.currentIndexChanged.connect(self.update\_items2)

        self.kol = None

        self.update\_items2()

        ########################

        self.pushButton.clicked.connect(self.create\_report)

        self.pushButton\_2.clicked.connect(lambda: self.remove\_nationality(self.comboBox\_2.currentText(), self.comboBox\_3.currentText()))

        self.pushButton\_3.clicked.connect(lambda: self.remove\_nationality\_2(self.comboBox\_2.currentText(), self.comboBox\_3.currentText()))

        self.comboBox.currentIndexChanged.connect(self.create\_report)

    def update\_items2(self):

        geografic\_position = self.comboBox.currentText()

        df = pd.read\_excel("python.xlsx")

        filtered\_df = df[df["географическое положение"] == geografic\_position]

        self.items2 = sorted(filtered\_df["год"].unique().astype(str).tolist())

        self.comboBox\_2.clear()

        self.comboBox\_2.addItems(self.items2)

        self.comboBox\_3.clear()

        self.comboBox\_3.addItems(self.items2)

        self.model = PandasModel(pd.DataFrame(columns=['национальности', 'численность', ]))

        self.tableView.setModel(self.model)

        self.tableView.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)

        self.figure = Figure()

        self.canvas = FigureCanvas(self.figure)

        self.toolbar = NavigationToolbar(self.canvas, self)

        # Очищаем существующий layout и создаем новый

        layout = QVBoxLayout()

        layout.addWidget(self.canvas)

        layout.addWidget(self.toolbar)

        # Устанавливаем layout в graphicsView

        self.graphicsView.setLayout(layout)

        self.create\_report()

    def create\_report(self):

        # Получаем текущее географическое положение и выбранные года из выпадающих списков

        geografic\_position = self.comboBox.currentText()

        year1 = int(self.comboBox\_2.currentText())

        year2 = int(self.comboBox\_3.currentText())

        # Загружаем данные из Excel файла и создаем DataFrame

        data = pd.read\_excel('python.xlsx')

        df = pd.DataFrame(data)

        # Фильтруем данные по выбранному географическому положению

        df = df[df['географическое положение'] == geografic\_position]

        # Фильтруем данные по выбранным годам и исключаем национальность 'Русские'

        df\_year1 = df[(df['год'] == year1) & (df['национальности'] != 'Русские')]

        df\_year2 = df[(df['год'] == year2) & (df['национальности'] != 'Русские')]

        # Получаем уникальные национальности

        nationalities = set(df\_year1['национальности']).union(set(df\_year2['национальности']))

        # Создаем объединенный DataFrame

        df\_combined = pd.DataFrame({'национальности': list(nationalities)})

        df\_combined = df\_combined.merge(df\_year1, on='национальности', how='left')

        df\_combined = df\_combined.merge(df\_year2, on='национальности', how='left')

        # Создаем модель данных и устанавливаем ее для отображения в таблице

        self.model = PandasModel(df\_combined[['национальности', 'численность\_x', 'численность\_y']])

        self.tableView.setModel(self.model)

        # Параметры для построения столбчатой диаграммы

        bar\_width = 0.35

        index = np.arange(len(df\_combined['национальности']))

        # Очищаем предыдущий график и строим новую столбчатую диаграмму

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        ax.clear()

        ax.bar(index, df\_combined['численность\_x'].fillna(0), bar\_width, color='blue', label=str(year1))

        ax.bar(index + bar\_width, df\_combined['численность\_y'].fillna(0), bar\_width, color='red', label=str(year2))

        # Настройки осей и заголовка графика

        ax.set\_xlabel('Национальности')

        ax.set\_ylabel('Численность')

        ax.set\_title(f'Численность самых крупных национальностей в {year1} и {year2} годах')

        ax.set\_xticks(index + bar\_width / 2)

        ax.set\_xticklabels(df\_combined['национальности'], rotation=90)

        ax.legend()

        # Обновляем график на холсте

        self.canvas.draw()

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MainWindow()

    window.show()

    sys.exit(app.exec())

программа 3(Категоризированная гистограмма)

import sys

import pandas as pd

import matplotlib.pyplot as plt

from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QComboBox, QPushButton, QAbstractItemView

from PyQt6.QtCore import Qt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.backends.backend\_qt5agg import NavigationToolbar2QT as NavigationToolbar

from matplotlib.figure import Figure

from PyQt6.QtCore import QAbstractTableModel, QVariant, QModelIndex

from PyQt6.uic import loadUi

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent=QModelIndex()):

        return len(self.\_data.index)

    def columnCount(self, parent=QModelIndex()):

        return len(self.\_data.columns)

    def data(self, index, role=Qt.ItemDataRole.DisplayRole):

        if role == Qt.ItemDataRole.DisplayRole:

            return str(self.\_data.iloc[index.row(), index.column()])

        return QVariant()

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.figure = Figure()

        self.canvas = FigureCanvas(self.figure)  # Добавьте эту строку

        loadUi("grafiks/krug.ui", self)

        self.df = pd.read\_excel("python.xlsx")

        self.items1 = sorted(self.df["географическое положение"].astype(str).unique().tolist())

        self.comboBox.addItems(self.items1)

        self.pushButton.clicked.connect(self.create\_report)

        self.pushButton.clicked.connect(self.create\_report\_1)

        self.comboBox.currentIndexChanged.connect(self.update\_items2)

        self.kol = None

        self.update\_items2()

        self.model = PandasModel(pd.DataFrame(columns=['национальности', 'численность']))  # Создаем модель для отчета

        self.tableView.setModel(self.model)

        self.tableView.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)

        self.figure = Figure()

        self.canvas = FigureCanvas(self.figure)

        layout = QVBoxLayout()

        layout.addWidget(self.canvas)

        self.toolbar = NavigationToolbar(self.canvas, self)

        layout.addWidget(self.toolbar)

        self.graphicsView.setLayout(layout)

    def update\_items2(self):

        geografic\_position = self.comboBox.currentText()

        df = pd.read\_excel("python.xlsx")

        filtered\_df = df[df["географическое положение"] == geografic\_position]

        self.items2 = sorted(filtered\_df["год"].unique().astype(str).tolist())

        self.comboBox\_2.clear()

        self.comboBox\_2.addItems(self.items2)

        self.model = PandasModel(pd.DataFrame(columns=['национальности', 'численность', ]))

        self.tableView.setModel(self.model)

        self.tableView.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)

        self.figure.clear()  # Очищаем старый график

        self.canvas.draw()

    def create\_report(self):

        # Функция для создания отчета, вызывает функцию grafic4 для построения круговой диаграммы

        print("Функция create\_report вызвана")

        geografic\_position = self.comboBox.currentText()

        year = int(self.comboBox\_2.currentText())  # Преобразуем год из строки в число

        self.grafic4(geografic\_position, year)

    def grafic4(self, geografic\_position, year):

        """

        Функция для построения круговой диаграммы на основе отфильтрованных данных

        :param geografic\_position: выбранное географическое положение

        :param year: выбранный год

        """

        data = self.df

        # Фильтрация данных по географическому положению и году

        filtered\_data = data[(data["географическое положение"] == geografic\_position) & (data["год"] == year)]

        colors = plt.cm.tab20b.colors

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Построение круговой диаграммы

        wedges, texts = ax.pie(filtered\_data["доля населения"], colors=colors, startangle=140)

        ax.axis('equal')

        ax.set\_title(f"Круговая диаграмма национальностей по численности за {year} год")

        # Добавление легенды к диаграмме

        ax.legend(wedges, filtered\_data["национальности"], title="Национальности", loc="upper right", bbox\_to\_anchor=(1.05, 1))

        self.canvas.draw()

    ##### Отчет

    def create\_report\_1(self):

        # Функция для создания отчета 1, отображает данные в виде табличного представления

        # Использует выбранное географическое положение и год для формирования отчета

        geografic\_position = self.comboBox.currentText()

        year = int(self.comboBox\_2.currentText())

        filtered\_data = self.df[(self.df["географическое положение"] == geografic\_position) & (self.df["год"] == year)]

        report\_df = filtered\_data[["национальности", "доля населения"]].copy()

        report\_df["доля населения"] = report\_df["доля населения"].apply(lambda x: "{:.2%}".format((x/100)\*100))

        report\_df.rename(columns={"доля населения": "численность"}, inplace=True)

        self.model = PandasModel(report\_df)

        self.tableView.setModel(self.model)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MainWindow()

    window.show()

    sys.exit(app.exec())

Программа 4 (Круговая диаграмма)

import sys

import pandas as pd

import matplotlib.pyplot as plt

from PyQt6.QtWidgets import QApplication, QMainWindow

from PyQt6.QtGui import QPixmap

from PyQt6.QtCore import Qt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from PyQt6.uic import loadUi

class MplCanvas(FigureCanvas):

    def \_\_init\_\_(self, parent=None, width=7.6, height=5.8, dpi=100):

        # Создание фигуры и осей для графика

        fig, self.ax = plt.subplots(figsize=(width, height), dpi=dpi)

        super(MplCanvas, self).\_\_init\_\_(fig)

        self.setParent(parent)

    def plot\_data(self, oblasts, sheet\_name="Лист1"):

        # Очистка текущего графика

        self.ax.clear()

        # Загрузка данных из Excel файла

        data = pd.read\_excel('python.xlsx', sheet\_name=sheet\_name)

        df = pd.DataFrame(data)

        # Построение графиков для выбранных регионов

        for oblast in oblasts:

            # Фильтрация данных по выбранному региону

            df\_oblast = df[df['субъекты'] == oblast]

            # Преобразование столбца 'население' в числовой формат

            df\_oblast['население'] = pd.to\_numeric(df\_oblast['население'], errors='coerce')

            # Удаление строк с отсутствующими значениями в столбце 'население'

            df\_oblast = df\_oblast.dropna(subset=['население'])

            # Построение графика населения по годам для выбранного региона

            self.ax.plot(df\_oblast['период'], df\_oblast['население'], label=oblast)

        # Настройка подписей осей и заголовка графика

        self.ax.set\_xlabel('Год')

        self.ax.set\_ylabel('Количество населения')

        self.ax.set\_title('Сравнение регионов по населению')

        # Форматирование значений осей

        self.ax.ticklabel\_format(style='plain')

        # Добавление легенды к графику

        self.ax.legend()

        # Обновление отображения графика

        self.figure.canvas.draw()

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Загрузка интерфейса из файла .ui

        loadUi("grafiks/new.ui", self)

        # Загрузка данных из Excel файла и создание списка уникальных значений для комбо-боксов

        df = pd.read\_excel("python.xlsx", sheet\_name="Лист1")

        self.items1 = sorted(df["субъекты"].astype(str).unique().tolist())

        # Добавление уникальных значений в комбо-боксы для выбора

        self.comboBox\_2.addItems(self.items1)

        self.comboBox\_3.addItems(self.items1)

        self.comboBox\_4.addItems(self.items1)

        self.comboBox\_5.addItems(self.items1)

        self.comboBox\_6.addItems(self.items1)

        self.comboBox\_7.addItems(self.items1)

        self.comboBox\_14.addItems(self.items1)

        self.comboBox\_15.addItems(self.items1)

        self.comboBox\_16.addItems(self.items1)

        self.comboBox\_17.addItems(self.items1)

        # Подключение кнопки к функции создания графика

        self.pushButton.clicked.connect(self.create\_plot)

        # Создание экземпляра класса MplCanvas для отображения графиков

        self.canvas = MplCanvas(self.graphicsView)

    def create\_plot(self):

        # Получение выбранных значений из комбо-боксов

        oblasts = [

            self.comboBox\_2.currentText(),

            self.comboBox\_3.currentText(),

            self.comboBox\_4.currentText(),

            self.comboBox\_5.currentText(),

            self.comboBox\_6.currentText(),

            self.comboBox\_7.currentText(),

            self.comboBox\_14.currentText(),

            self.comboBox\_15.currentText(),

            self.comboBox\_16.currentText(),

            self.comboBox\_17.currentText(),

        ]

        # Фильтрация значений, исключение 'nan'

        oblasts = [oblast for oblast in oblasts if oblast != 'nan']

        # Передача отфильтрованных данных для построения графика

        self.canvas.plot\_data(oblasts)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MainWindow()

    window.show()

    sys.exit(app.exec())

Программа 5(Линейная диаграмма)

import sys

import pandas as pd

import matplotlib.pyplot as plt

from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QComboBox, QPushButton, QAbstractItemView

from PyQt6.QtCore import Qt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.backends.backend\_qt5agg import NavigationToolbar2QT as NavigationToolbar

from matplotlib.figure import Figure

from PyQt6.QtCore import QAbstractTableModel, QVariant, QModelIndex

from PyQt6.uic import loadUi

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent=QModelIndex()):

        return len(self.\_data.index)

    def columnCount(self, parent=QModelIndex()):

        return len(self.\_data.columns)

    def data(self, index, role=Qt.ItemDataRole.DisplayRole):

        if role == Qt.ItemDataRole.DisplayRole:

            return str(self.\_data.iloc[index.row(), index.column()])

        return QVariant()

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        loadUi("grafiks/rass.ui", self)

        self.df = pd.read\_excel("python.xlsx")

        self.items1 = sorted(self.df["географическое положение"].astype(str).unique().tolist())

        self.comboBox.addItems(self.items1)

        self.pushButton.clicked.connect(self.create\_report)

        self.pushButton\_2.clicked.connect(self.remove\_nationality)

        self.pushButton\_3.clicked.connect(self.remove\_nationality\_2)

        self.model = PandasModel(pd.DataFrame(columns=['национальности', 'численность']))

        self.tableView.setModel(self.model)

        self.tableView.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)

        self.figure = Figure()

        self.canvas = FigureCanvas(self.figure)

        layout = QVBoxLayout()

        layout.addWidget(self.canvas)

        self.toolbar = NavigationToolbar(self.canvas, self)

        layout.addWidget(self.toolbar)

        self.graphicsView.setLayout(layout)

    def remove\_nationality(self):

        selected\_rows = self.tableView.selectionModel().selectedRows()

        if selected\_rows:

            row = selected\_rows[0].row()

            nationality = self.model.\_data.iloc[row]['национальности']

            self.model.\_data = self.model.\_data[self.model.\_data['национальности'] != nationality]

            self.model.layoutChanged.emit()

            self.update\_plot()

    def remove\_nationality\_2(self):

        selected\_rows = self.tableView.selectionModel().selectedRows()

        if selected\_rows:

            row = selected\_rows[0].row()

            nationality = self.model.\_data.iloc[row]['национальности']

            self.model.\_data = self.model.\_data[self.model.\_data['национальности'] != nationality]

            self.model.layoutChanged.emit()

            self.update\_plot\_2()

    def update\_plot\_2(self):

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        selected\_nationalities = [index.data() for index in self.tableView.selectionModel().selectedRows(column=0)]

        if selected\_nationalities:

            data = self.df[(self.df['географическое положение'] == self.comboBox.currentText()) & (self.df['национальности'].isin(selected\_nationalities))]

            data = data.sort\_values(by='доля населения')

            scatter = ax.scatter(data['доля населения'].apply(lambda x: "{:.2%}".format((x/1000))), data['национальности'], c=data['год'])

            ax.set\_xlabel('Доля населения')

            ax.set\_ylabel('Национальность')

            ax.set\_title('Категоризированная диаграмма рассеивания')

            ax.xaxis.set\_tick\_params(rotation=90)

            ax.grid(True)

            legend = ax.legend(\*scatter.legend\_elements(), title="Годы")

            ax.add\_artist(legend)

            self.canvas.draw()

        else:

            ax.axis('off')

            ax.text(0.5, 0.5, "Выберите национальность в таблице", horizontalalignment='center', verticalalignment='center', transform=ax.transAxes)

            self.canvas.draw()

    def update\_plot(self):

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        sorted\_data = self.model.\_data.sort\_values(by='национальности')

        data = self.df[(self.df['географическое положение'] == self.comboBox.currentText()) & (self.df['национальности'].isin(sorted\_data['национальности']))]

        data = data.sort\_values(by='доля населения')

        scatter = ax.scatter(data['доля населения'].apply(lambda x: "{:.2%}".format((x/1000))), data['национальности'], c=data['год'])

        ax.set\_xlabel('Доля населения')

        ax.set\_ylabel('Национальность')

        ax.set\_title('Категоризированная диаграмма рассеивания')

        ax.xaxis.set\_tick\_params(rotation=90)

        ax.grid(True)

        legend = ax.legend(\*scatter.legend\_elements(), title="Годы")

        ax.add\_artist(legend)

        self.canvas.draw()

    def create\_report(self):

        # Получение выбранного географического положения из комбо-бокса

        geografic\_position = self.comboBox.currentText()

        # Фильтрация данных по выбранному географическому положению

        data = self.df[self.df['географическое положение'] == geografic\_position]

        # Группировка данных по национальности, доле населения и году, и подсчет суммы

        report\_df = data.groupby(['национальности', 'доля населения', 'год']).sum().reset\_index()

        # Преобразование столбца 'доля населения' в процентный формат

        report\_df["доля населения"] = report\_df["доля населения"].apply(lambda x: "{:.2%}".format((x/1000)))

        # Сортировка данных по национальности

        report\_df = report\_df.sort\_values(by='национальности')

        # Отображение данных с наибольшей долей населения

        report\_df = report\_df.sort\_values(by='доля населения')

        # Обновление модели данных для отображения в таблице

        self.model.\_data = report\_df

        self.model.layoutChanged.emit()

        # Очистка текущего графика и добавление нового графика

        self.figure.clear()

        ax = self.figure.add\_subplot(111)

        # Построение диаграммы рассеивания

        scatter = ax.scatter(report\_df['доля населения'], report\_df['национальности'], c=report\_df['год'])

        ax.set\_xlabel('Доля населения')

        ax.set\_ylabel('Национальность')

        ax.set\_title('Категоризированная диаграмма рассеивания')

        ax.xaxis.set\_tick\_params(rotation=90)

        ax.grid(True)

        # Добавление легенды к графику

        legend = ax.legend(\*scatter.legend\_elements(), title="Годы")

        ax.add\_artist(legend)

        # Скрытие лишних столбцов в таблице

        self.tableView.setColumnHidden(3, True)

        self.tableView.setColumnHidden(4, True)

        self.tableView.setColumnHidden(5, True)

        self.tableView.setColumnHidden(6, True)

        self.tableView.setColumnHidden(7, True)

        # Обновление отображения графика

        self.canvas.draw()

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MainWindow()

    window.show()

    sys.exit(app.exec())

Программа 5(диаграмма рассеивания)

### **Листинги отчетов**

import sys

import pandas as pd

from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QTableView, QTableWidgetItem, QHeaderView

from PyQt6.uic import loadUi

from PyQt6.QtWidgets import QMessageBox, QTableView

from PyQt6.QtCore import QAbstractTableModel, Qt

number\_to\_word = {

    0: "Численность",

    1: "Национальность",

    2: "Субъект",

    3: "Год"

}

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent):

        return self.\_data.shape[0]

    def columnCount(self, parent):

        return self.\_data.shape[1]

    def data(self, index, role):

        if role == Qt.ItemDataRole.DisplayRole:

            value = self.\_data.iloc[index.row(), index.column()]

            if isinstance(value, int) and value in number\_to\_word:

                return number\_to\_word[value]

            else:

                return str(value)

    def headerData(self, section, orientation, role):

        if role == Qt.ItemDataRole.DisplayRole:

            if orientation == Qt.Orientation.Horizontal:

                if section in number\_to\_word:

                    return number\_to\_word[section]

                else:

                    return str(self.\_data.columns[section])

            elif orientation == Qt.Orientation.Vertical:

                return str(section + 1)

        return None

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        loadUi("otcheti\otchet1.ui", self)

        df = pd.read\_excel("python.xlsx")

        self.items1 = sorted([str(x) for x in df["национальности"].unique()])

        self.comboBox.addItems(self.items1)

        self.pushButton.clicked.connect(self.create\_report)

        self.lineEdit.textChanged.connect(self.update\_kol)

        self.kol = None

    def update\_kol(self):

        text = self.lineEdit.text()

        try:

            self.kol = int(text)

        except ValueError:

            self.kol = None

    def create\_report(self):

        # Получение выбранной национальности из комбо-бокса

        nationalities = self.comboBox.currentText()

        # Проверка на пустое значение выбранной национальности

        if nationalities == "nan":

            # Вывод предупреждения, если национальность не выбрана

            QMessageBox.warning(self, "Предупреждение", "Выберите национальность")

            return

        # Проверка на наличие корректного значения для количества

        if self.kol is None:

            # Вывод предупреждения, если количества нет или некорректное

            QMessageBox.warning(self, "Предупреждение", "Введите корректное значение для количества")

            return

        # Загрузка данных из Excel файла 'python.xlsx' в DataFrame

        df = pd.read\_excel('python.xlsx')

        # Преобразование столбцов 'численность' и 'год' в числовой формат

        df['численность'] = pd.to\_numeric(df['численность'], errors='coerce').fillna(0).astype(int)

        df['год'] = pd.to\_numeric(df['год'], errors='coerce').fillna(0).astype(int)

        # Сортировка данных по столбцу 'численность'

        df.sort\_values(by='численность', inplace=True)

        # Вызов функции self.chisl для создания отчета по выбранной национальности и количеству

        report\_df = self.chisl(df, 'национальности', nationalities, 'численность', self.kol)

        # Создание модели данных PandasModel на основе отчета report\_df

        model = PandasModel(report\_df)

        # Установка созданной модели данных в таблицу для отображения

        self.tableView.setModel(model)

    def chisl(self, df: pd.DataFrame, cndname: str, cndval: str, intname: str, intval: int) -> pd.DataFrame:

        """

        Функция фильтрует данные DataFrame по заданным условиям и возвращает новый DataFrame.

        Параметры:

        - df: pd.DataFrame - исходный DataFrame с данными

        - cndname: str - название столбца для условия фильтрации

        - cndval: str - значение условия фильтрации

        - intname: str - название столбца для сравнения с числовым значением

        - intval: int - числовое значение для фильтрации

        Возвращает:

        - new\_df: pd.DataFrame - новый DataFrame после фильтрации

        """

        # Фильтрация данных по заданным условиям

        new\_df = df[(df[cndname] == cndval) & (df[intname] < intval)][['численность', 'национальности', 'географическое положение', 'год']]

        # Возврат нового DataFrame после фильтрации

        return new\_df

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MainWindow()

    window.show()

    sys.exit(app.exec())

Программа 6 (Отчет о численности для национальности)

import sys

import pandas as pd

from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QTableView, QTableWidgetItem, QHeaderView

from PyQt6.uic import loadUi

from PyQt6.QtWidgets import QMessageBox, QTableView

from PyQt6.QtCore import QAbstractTableModel, Qt

number\_to\_word = {

    0: "Национальность",

    1: "Год",

    2: "Субъект",

    3: "Численность"

}

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent):

        return self.\_data.shape[0]

    def columnCount(self, parent):

        return self.\_data.shape[1]

    def data(self, index, role):

        if role == Qt.ItemDataRole.DisplayRole:

            value = self.\_data.iloc[index.row(), index.column()]

            if isinstance(value, int) and value in number\_to\_word:

                return number\_to\_word[value]

            else:

                return str(value)

    def headerData(self, section, orientation, role):

        if role == Qt.ItemDataRole.DisplayRole:

            if orientation == Qt.Orientation.Horizontal:

                if section in number\_to\_word:

                    return number\_to\_word[section]

                else:

                    return str(self.\_data.columns[section])

            elif orientation == Qt.Orientation.Vertical:

                return str(section + 1)

        return None

class MyApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        loadUi("otcheti\otchet2.ui", self)

        df = pd.read\_excel("python.xlsx")

        self.items1 = sorted(df["географическое положение"].unique().tolist())

        self.comboBox.addItems(self.items1)

        self.comboBox.currentIndexChanged.connect(self.update\_items2)

        self.pushButton.clicked.connect(self.create\_report)

        self.kol = None

        self.update\_items2()

    def update\_items2(self):

        selected\_subject = self.comboBox.currentText()

        df = pd.read\_excel("python.xlsx")

        filtered\_df = df[df["географическое положение"] == selected\_subject]

        self.items2 = sorted(filtered\_df["год"].unique().astype(str).tolist())

        self.comboBox\_2.clear()

        self.comboBox\_2.addItems(self.items2)

    def create\_report(self):

        position = self.comboBox.currentText()

        year = self.comboBox\_2.currentText()

        df = pd.read\_excel('python.xlsx')

        report\_df = self.chisl(df, 'географическое положение', position, 'год', year)

        model = PandasModel(report\_df)

        self.tableView.setModel(model)

    def chisl(self, df: pd.DataFrame, cndname\_1: str, cndval\_1: str, cndname\_2: str, cndval\_2: str) -> pd.DataFrame:

        """

        Функция фильтрует данные DataFrame по двум заданным условиям и возвращает новый DataFrame.

        Параметры:

        - df: pd.DataFrame - исходный DataFrame с данными

        - cndname\_1: str - название первого столбца для условия фильтрации

        - cndval\_1: str - значение первого условия фильтрации

        - cndname\_2: str - название второго столбца для условия фильтрации

        - cndval\_2: str - значение второго условия фильтрации

        Возвращает:

        - new\_df: pd.DataFrame - новый DataFrame после фильтрации

        """

        # Фильтрация данных по двум заданным условиям

        new\_df = df[(df[cndname\_1] == cndval\_1) & (df[cndname\_2].astype(str) == cndval\_2)][['национальности', 'год', 'географическое положение', 'численность']]

        # Возврат нового DataFrame после фильтрации

        return new\_df

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MyApp()

    window.show()

    sys.exit(app.exec())

Программа 7(Отчет о национальном составе региона в определенный год)

import sys

import pandas as pd

from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QTableView, QTableWidgetItem, QHeaderView

from PyQt6.uic import loadUi

from PyQt6.QtWidgets import QMessageBox, QTableView

from PyQt6.QtCore import QAbstractTableModel, Qt

number\_to\_word = {

    0: "Численность",

    1: "Национальность",

    2: "Субъект",

    3: "Год"

}

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent):

        return self.\_data.shape[0]

    def columnCount(self, parent):

        return self.\_data.shape[1]

    def data(self, index, role):

        if role == Qt.ItemDataRole.DisplayRole:

            value = self.\_data.iloc[index.row(), index.column()]

            if isinstance(value, int) and value in number\_to\_word:

                return number\_to\_word[value]

            else:

                return str(value)

    def headerData(self, section, orientation, role):

        if role == Qt.ItemDataRole.DisplayRole:

            if orientation == Qt.Orientation.Horizontal:

                if section in number\_to\_word:

                    return number\_to\_word[section]

                else:

                    return str(self.\_data.columns[section])

            elif orientation == Qt.Orientation.Vertical:

                return str(section + 1)

        return None

class MyApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        loadUi("otcheti\otcheti3.ui", self)

        df = pd.read\_excel("python.xlsx")

        df['численность'] = pd.to\_numeric(df['численность'], errors='coerce')

        df = df.dropna(subset=['численность'])

        self.items1 = sorted(df['численность'].astype(int).unique().tolist())

        self.comboBox.addItems([str(x) for x in self.items1])

        self.comboBox\_2.addItems([str(x) for x in self.items1])

        self.pushButton.clicked.connect(self.create\_report)

    def create\_report(self):

        try:

            # Попытка преобразовать текущий текст из выпадающего списка в целое число для минимальной и максимальной численности населения

            min\_population = int(self.comboBox.currentText())

            max\_population = int(self.comboBox\_2.currentText())

            # Проверка, что минимальное значение населения не больше максимального

            if min\_population > max\_population:

                # Вывод предупреждения об ошибке, если условие не выполняется

                QMessageBox.warning(self, "Ошибка", "Минимальное значение не может быть больше Максимального")

                return

        except ValueError:

            # В случае исключения ValueError (некорректные данные) выводится предупреждение

            QMessageBox.warning(self, "Выберите корректные данные")

            return

        df = pd.read\_excel('python.xlsx')

        df['численность'] = pd.to\_numeric(df['численность'], errors= 'coerce').fillna(0).astype(int)

        df['год'] = pd.to\_numeric(df['год'], errors= 'coerce').fillna(0).astype(int)

        df.sort\_values(by = 'численность', inplace=True)

        report\_df = self.report4(df, 'численность', min\_population, 'численность', max\_population)

        model = PandasModel(report\_df)

        self.tableView.setModel(model)

    def report4(self, df: pd.DataFrame, cndname\_1: str, intname: int, cndname\_2: str, cndval\_2: int) -> pd.DataFrame:

        """

        Функция формирует отчет на основе данных DataFrame, фильтруя их по заданным условиям.

        Параметры:

        - df: pd.DataFrame - исходный DataFrame с данными

        - cndname\_1: str - название первого столбца для фильтрации по значению больше или равно intname

        - intname: int - числовое значение для фильтрации по cndname\_1

        - cndname\_2: str - название второго столбца для фильтрации по значению меньше или равно cndval\_2

        - cndval\_2: int - числовое значение для фильтрации по cndname\_2

        Возвращает:

        - new\_df: pd.DataFrame - новый DataFrame после применения фильтрации

        """

        # Фильтрация данных по заданным условиям

        new\_df = df[(df[cndname\_1] >= intname) & (df[cndname\_2] <= int(cndval\_2))][['численность', 'национальности', 'географическое положение', 'год']]

        # Возврат нового DataFrame после фильтрации для формирования отчета

        return new\_df

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MyApp()

    window.show()

    sys.exit(app.exec())

Программа 8(численный состав национальности в пределах)

import sys

import pandas as pd

from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QTableView, QHeaderView

from PyQt6.uic import loadUi

from PyQt6.QtCore import QAbstractTableModel, Qt, QVariant

from PyQt6.QtGui import QColor

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent):

        return self.\_data.shape[0]

    def columnCount(self, parent):

        return self.\_data.shape[1]

    def data(self, index, role):

        if role == Qt.ItemDataRole.DisplayRole:

            value = self.\_data.iloc[index.row(), index.column()]

            if pd.isna(value):

                return ('')

            else:

                return str(value)

        elif role == Qt.ItemDataRole.BackgroundRole:

            value = self.\_data.iloc[index.row(), index.column()]

            if pd.isna(value):

                return QColor('red')

            else:

                return QColor('white')

    def headerData(self, section, orientation, role):

        if role == Qt.ItemDataRole.DisplayRole:

            if orientation == Qt.Orientation.Horizontal:

                return QVariant(str(self.\_data.columns[section]))

            elif orientation == Qt.Orientation.Vertical:

                return QVariant(str(self.\_data.index[section]))

class MyApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Загрузка пользовательского интерфейса из файла svodntable2.ui

        loadUi("otcheti\svodntable2.ui", self)

        # Чтение данных из файла Excel 'python.xlsx' из листа с индексом 1

        df = pd.read\_excel("python.xlsx", sheet\_name=1)

        # Преобразование столбцов 'население' и 'период' в числовой формат

        df['население'] = pd.to\_numeric(df['население'], errors='coerce').fillna(0).astype(int)

        df['период'] = pd.to\_numeric(df['период'], errors='coerce').fillna(0).astype(int)

        # Создание сводной таблицы на основе данных DataFrame

        pivot\_table = pd.pivot\_table(df, values='население', index='субъекты', columns='период')

        # Создание модели данных PandasModel на основе сводной таблицы

        model = PandasModel(pivot\_table)

        # Установка модели данных для отображения в tableView

        self.tableView.setModel(model)

        # Установка режима изменения размеров колонок таблицы по содержимому

        self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.ResizeToContents)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MyApp()

    window.show()

    sys.exit(app.exec())

программа 9 (сводная таблица о населении регионов)

import sys

import pandas as pd

from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QTableView, QHeaderView

from PyQt6.uic import loadUi

from PyQt6.QtCore import QAbstractTableModel, Qt, QVariant

from PyQt6.QtGui import QColor

class PandasModel(QAbstractTableModel):

    def \_\_init\_\_(self, data):

        super().\_\_init\_\_()

        self.\_data = data

    def rowCount(self, parent):

        return self.\_data.shape[0]

    def columnCount(self, parent):

        return self.\_data.shape[1]

    def data(self, index, role):

        if role == Qt.ItemDataRole.DisplayRole:

            value = self.\_data.iloc[index.row(), index.column()]

            if pd.isna(value):

                return ('')

            else:

                return str(value)

        elif role == Qt.ItemDataRole.BackgroundRole:

            value = self.\_data.iloc[index.row(), index.column()]

            if pd.isna(value):

                return QColor('red')

            else:

                return QColor('white')

    def headerData(self, section, orientation, role):

        if role == Qt.ItemDataRole.DisplayRole:

            if orientation == Qt.Orientation.Horizontal:

                return QVariant(str(self.\_data.columns[section]))

            elif orientation == Qt.Orientation.Vertical:

                return QVariant(str(self.\_data.index[section]))

class MyApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Загрузка пользовательского интерфейса из файла svodntable.ui

        loadUi("otcheti\svodntable.ui", self)

        # Чтение данных из файла Excel 'python.xlsx'

        df = pd.read\_excel("python.xlsx")

        # Преобразование столбцов 'численность' и 'год' в числовой формат

        df['численность'] = pd.to\_numeric(df['численность'], errors='coerce').fillna(0).astype(int)

        df['год'] = pd.to\_numeric(df['год'], errors='coerce').fillna(0).astype(int)

        # Создание сводной таблицы на основе данных DataFrame

        pivot\_table = pd.pivot\_table(df, values='численность', index='национальности', columns='год', aggfunc='sum')

        # Создание модели данных PandasModel на основе сводной таблицы

        model = PandasModel(pivot\_table)

        # Установка модели данных для отображения в tableView

        self.tableView.setModel(model)

        # Установка режима изменения размеров колонок таблицы по содержимому

        self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.ResizeToContents)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = MyApp()

    window.show()

    sys.exit(app.exec())

Программа 10(Сводная таблица о национальностях)