ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | С.Ю. Гуков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |
| СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ (VCS) |
| по курсу: ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4217 |  |  |  | И.М. Полозков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

**Цель работы**

Изучить предназначение и различные способы организаций систем контроля версий (Version Control System, VCS) Git. Познакомиться с операциями над файлами в репозитории и с приемами командной работы над проектом.

**Задание**

Задание можно выполнять в любой платформе, основанной на git (GitHub, GitLab, GitFlic, BitBucket и др.). Необходимо объединиться в команды по 2-4 человека. У каждого участника команды должен быть свой зарегистрированный аккаунт на выбранной платформе. Один из участников команды создает репозиторий и присоединяет к нему остальных участников. Необходимо придумать общий интерфейс программы (один из участников делает коммит набросков интерфейса в репозиторий, остальные обновляют у себя локальную копию репозитория). Далее каждый из участников обязательно в своей отдельной ветке выполняет свое задание по варианту (задания в команде должны различаться), периодически делая коммиты своих классов и изменений в коде в репозиторий и делая pull request в ветку dev, при этом обновляя (дополняя) свой локальный проект кодом этой ветки (с обновлениями коллег по команде). Ветки после слияния удалять не нужно. Также при pull request каждому необходимо сделать проверку (review) кода коллег – оставить несколько комментариев с советами и замечаниями по различным местам в коде. После того, как все участники команды сделают свое задание, ветка dev сливается в главную ветку master (main), и оформляется файл README.md с пояснениями о выполненных заданиях. В итоге должен получиться проект с единым интерфейсом, выполняющий несколько различных задач (по количеству участников команды). У каждого участника команды на компьютере должен находиться полный общий локальный проект (содержащий свое реализованное задание и код коллег по команде). В качестве проверки задания преподаватель также будет смотреть в онлайн репозитории созданные ветки и список коммитов – кто из участников, когда и какие сделал изменения в проекте. Проект обязательно должен иметь графический пользовательский интерфейс (User Interface, UI), а также может быть написан на любом языке программирования. Замечание: разрешается выполнять проект в одиночку, при условии имитации работы в команде (регистрации 2-3 аккаунтов и выполнении различного задания с каждого аккаунта).

**Вариант**

5. Пользователь открывает файл с данными о численности населения России за последние 15 лет. Вывести эту информацию на экран в удобном табличном формате. По этим 6 данным построить графики зависимости от года. Вычислить максимальный процент прироста и убыли населения за год. Реализовать статистическое прогнозирование методом экстраполяции по скользящей средней на последующие N лет, вывести эту информацию на отдельном графике либо закрасить другим цветом на том же.

17. Пользователь открывает файл с данными о проценте детей, рожденных вне брака за последние 15 лет. Вывести эту информацию на экран в удобном табличном формате. По этим данным построить графики зависимости от года. Вычислить максимальный и минимальный процент изменения этих данных за год. Реализовать статистическое прогнозирование методом экстраполяции по скользящей средней на последующие N лет, вывести эту информацию на отдельном графике либо закрасить другим цветом на том же.

**Решение**

Сначала был произведён вход в основной аккаунт. См. рис. 1.

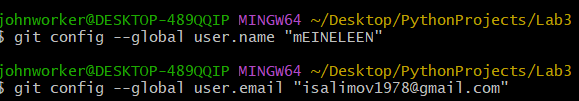


Рисунок 1 – Вход в основной аккаунт

Далее была произведена инициализация локального репозитория и переименование ветки в main, так как это рекомендовано GitHub. См. рис. 2.

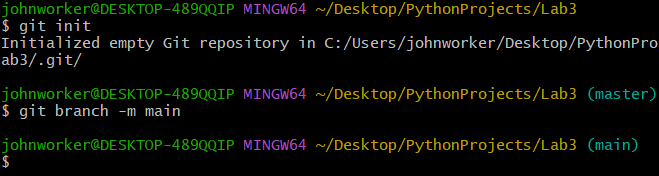


Рисунок 2 – Инициализация локального репозитория и переименование ветки

Далее локальный репозиторий был подключён к удалённому.



Рисунок 3 – Подключение локального репозитория к удалённому

Далее был создан и запушен первый коммит. См. рис. 4, 5.

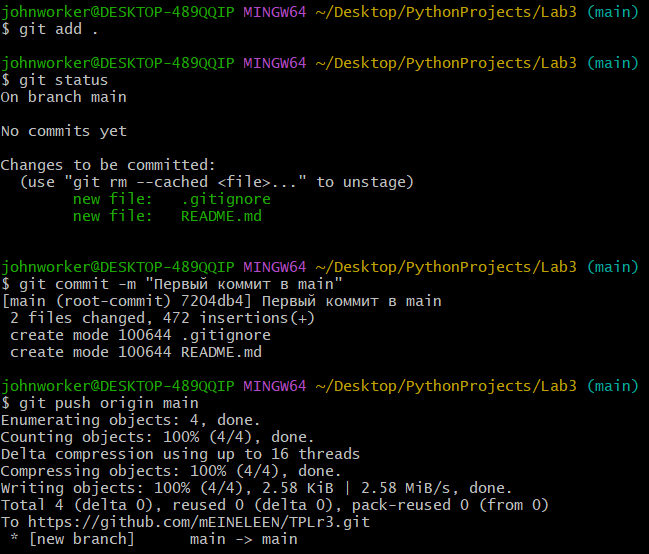


Рисунок 4 – Пуш коммита

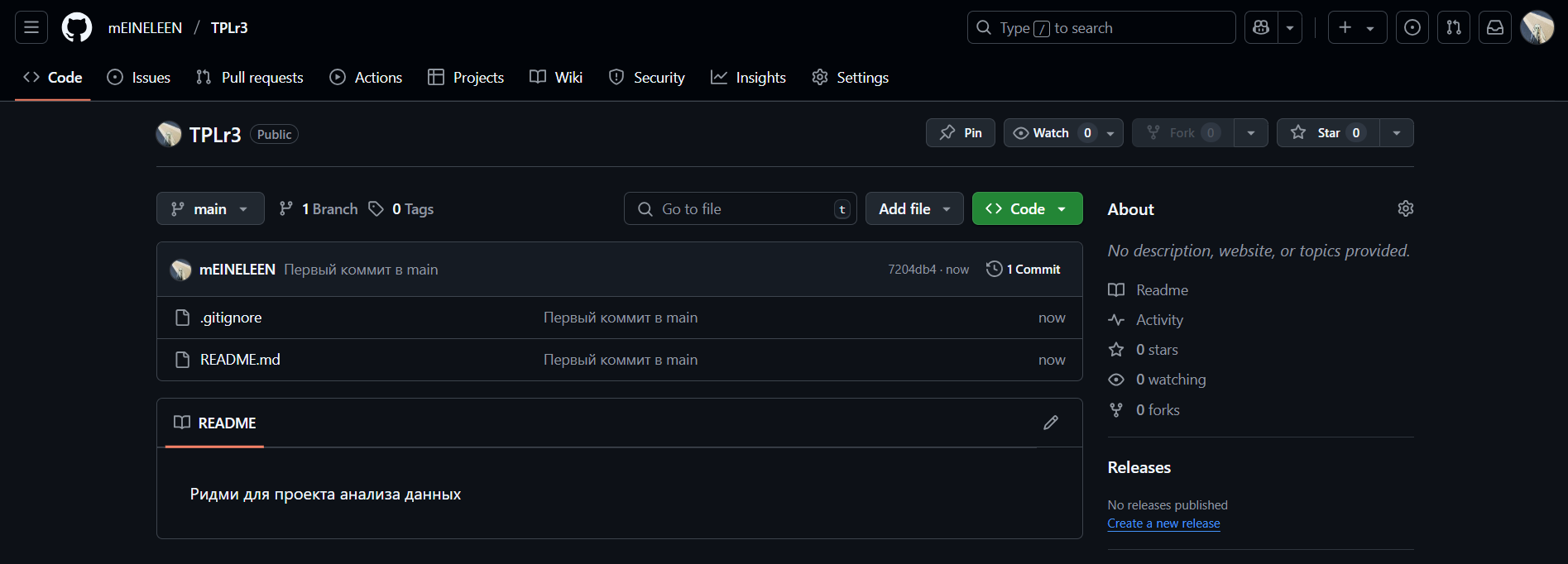


Рисунок 5 – Репозиторий после пуша

Далее были созданы ветки, необходимые для разработки по 17 варианту. А также произведён первый пуш, чтобы ветки появились на GitHub. См. рис. 5, 6.

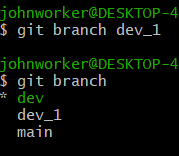


Рисунок 5 – Список созданных веток

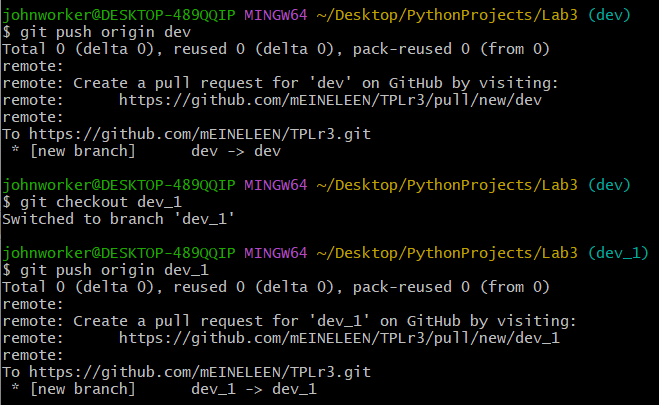


Рисунок 6 – Первый пуш на GitHub

Ветки отобразились на сайте GitHub. См. рис. 7.

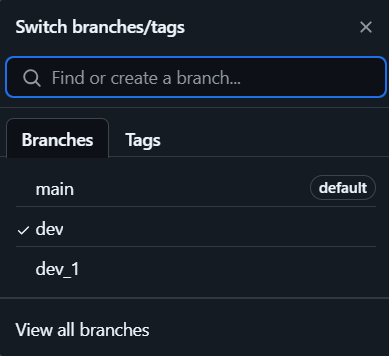


Рисунок 7 – Отображение ветки на GitHub

Для разработки интерфейса была использована библиотека PyQt5, для прорисовки графиков matplotlib, для работы с данными pandas. Программа в целом написана на языке Python. Полный код программы доступен в приложениях А, Б и В.

Сначала был разработан макет для лицевой страницы приложения. См. рис. 8.

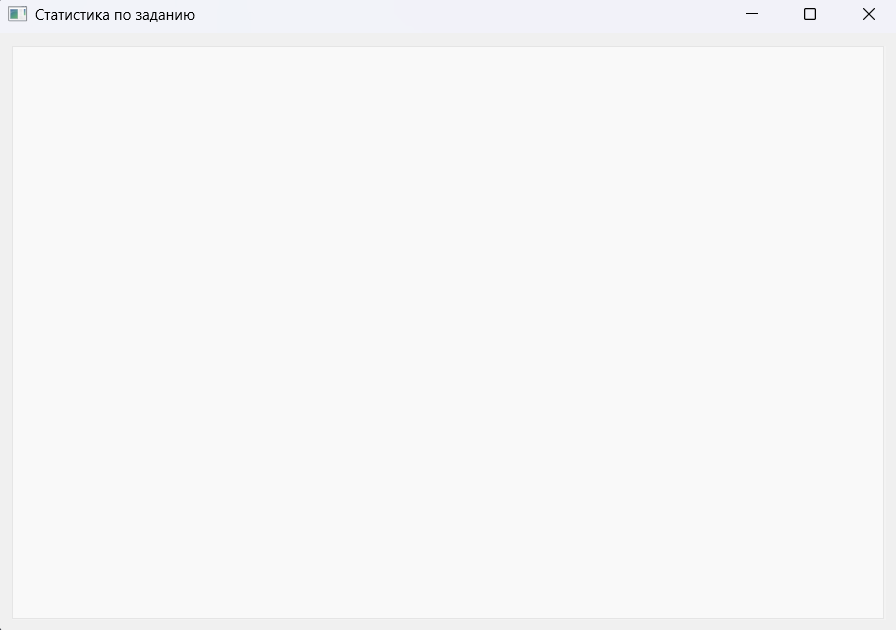


Рисунок 8 – Макет лицевой страницы приложения

Далее была добавлен вкладка со статистикой по детям, рождённым вне брака, вкладка показана на рисунках 9, 10.

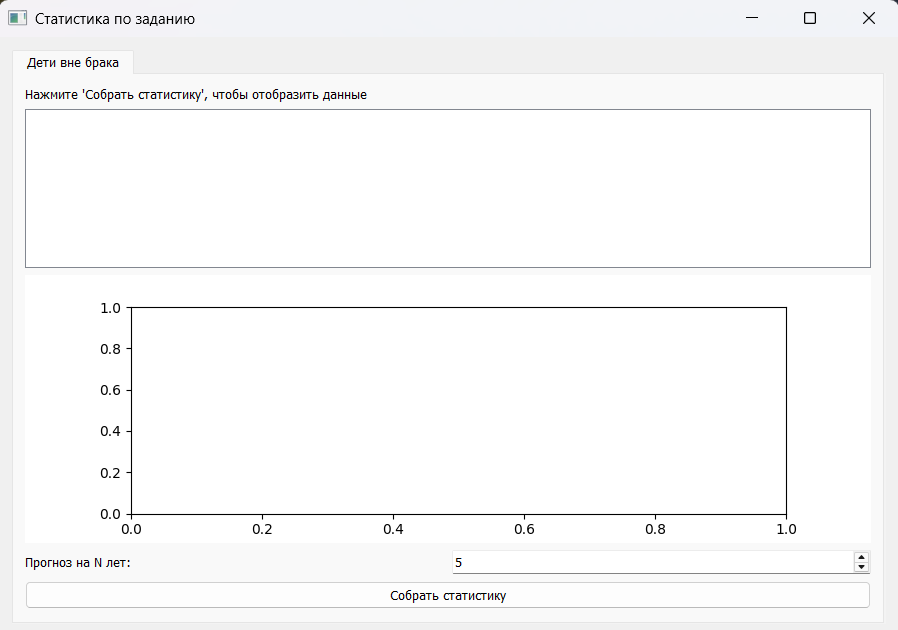


Рисунок 9 – Добавленная вкладка до нажатия на кнопку «Собрать статистику»

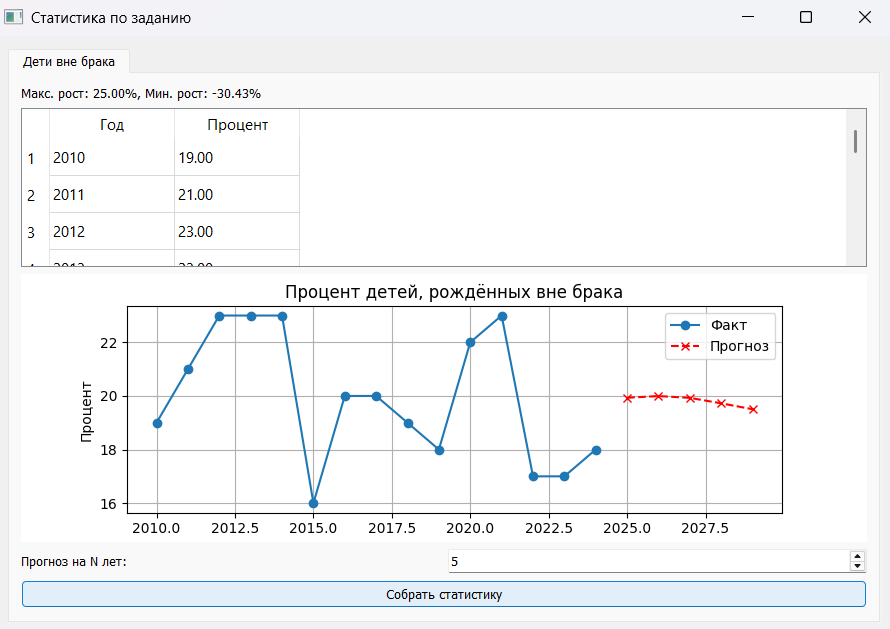


Рисунок 10 – Добавленная вкладка после нажатия на кнопку «Собрать статистику»

Далее показана работа скользящей средней. См. рис. 11.

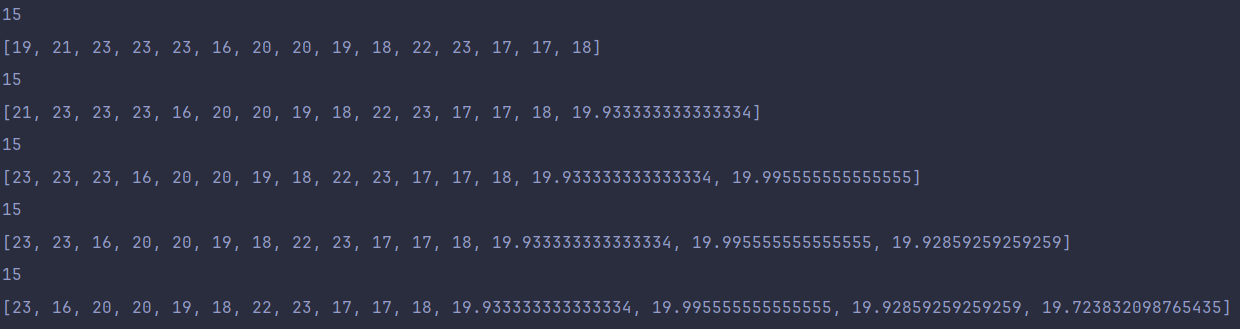


Рисунок 11 – Скользящая средняя в работе

После окончания разработки был произведён Pull Request и merge. См. рис. 12, 13.

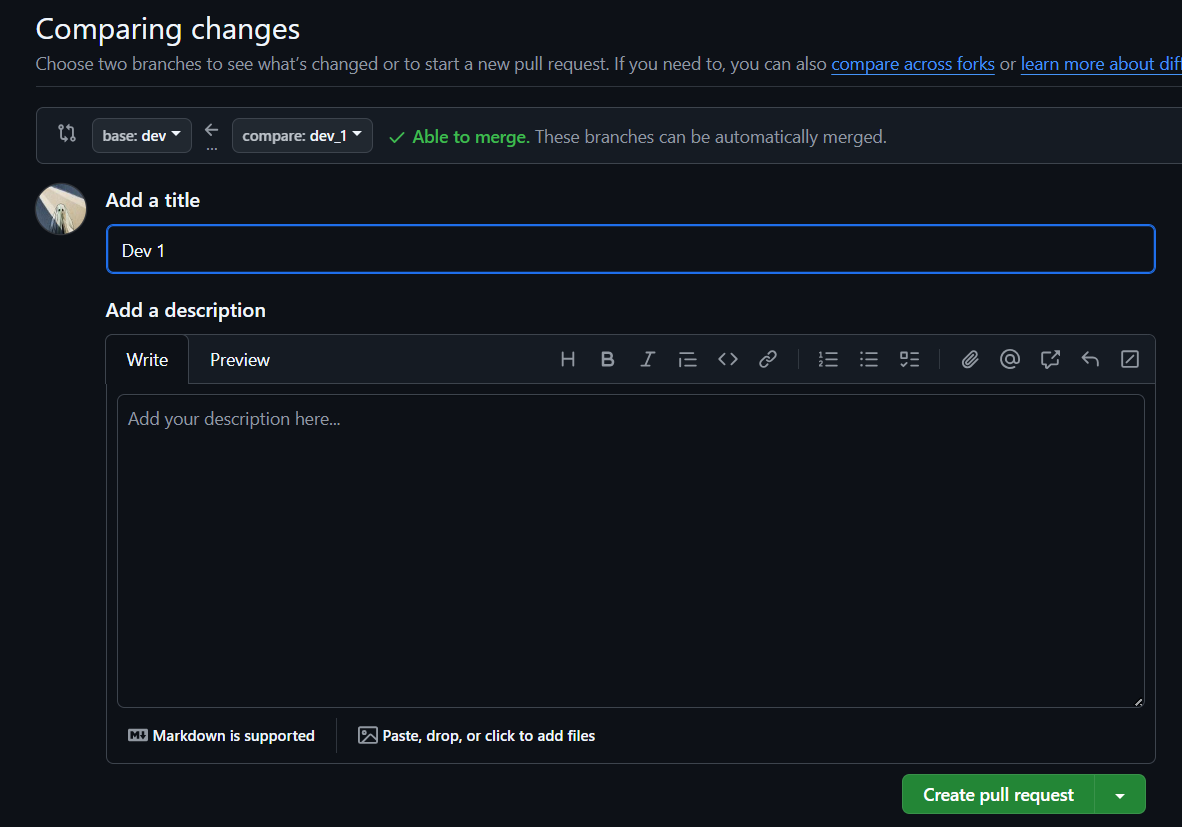


Рисунок 12 – Создание pull request

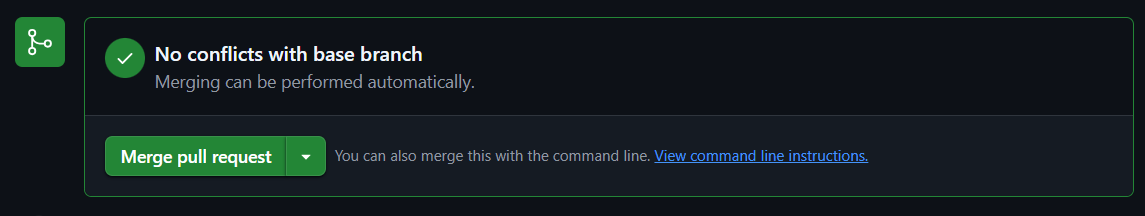


Рисунок 13 – Завершение pull request

Далее после слияния файлы были обновлены в локальном репозитории ветки main. См. рис. 14, 15.

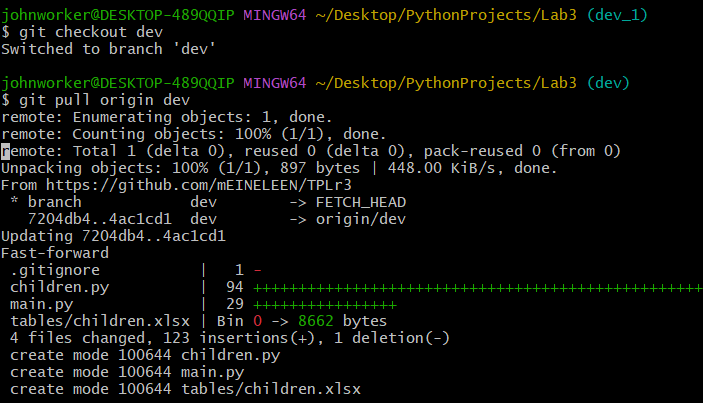


Рисунок 14 – Обновление ветки dev в локальном репозитории

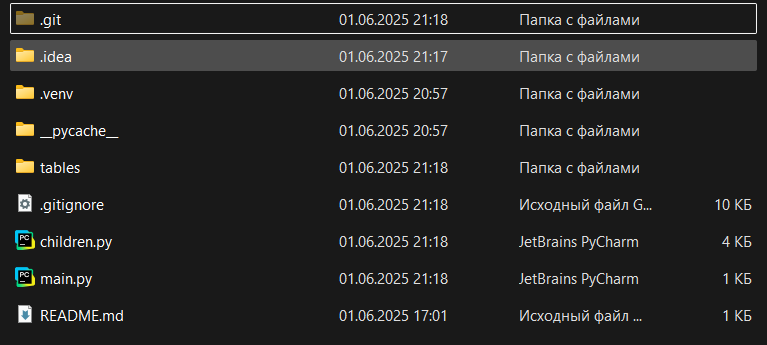


Рисунок 15 – Результат обновления

Далее был переключён пользователь и клонирован репозиторий, а далее создана ветка dev\_2 для выполнения задания по 5 варианту. См. рис. 16, 17.

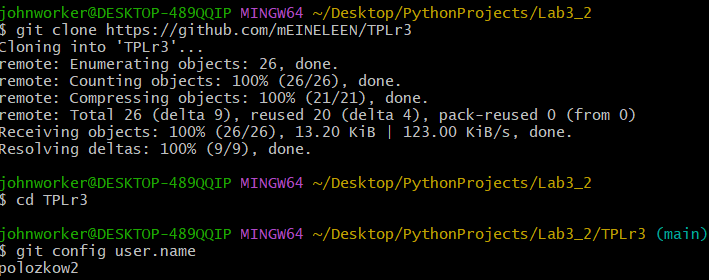


Рисунок 16 – Клонирование репозитория

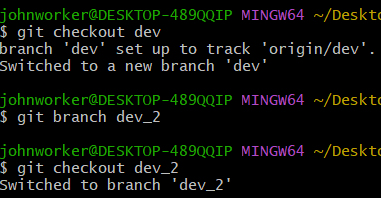


Рисунок 17 – Создание необходимой ветки

Далее показано разработанное приложение по стандартному макету интерфейса. См. рис. 18.

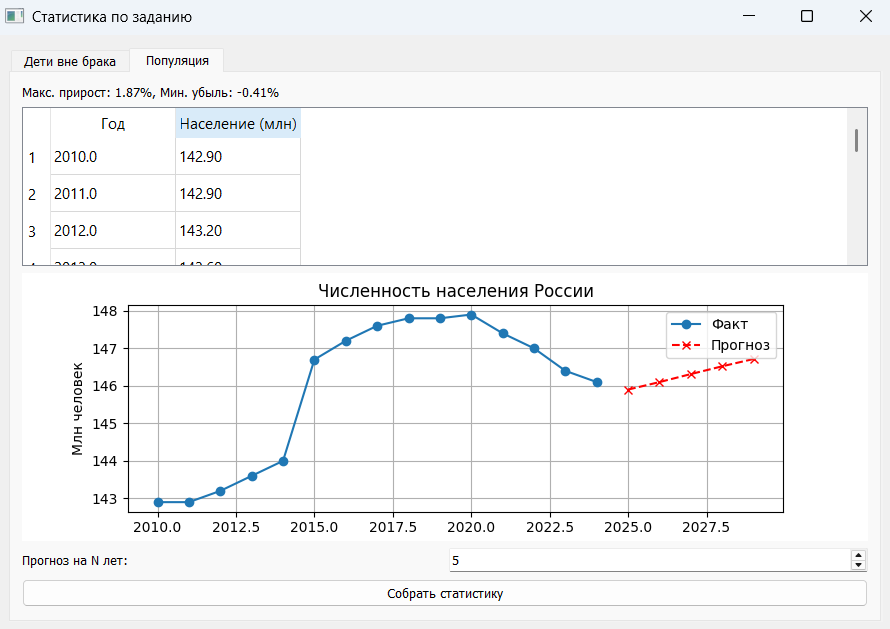


Рисунок 18 – Вторая разработанная вкладка с населением России

Далее показана работа скользящей средней во второй вкладке. См. рис. 19.

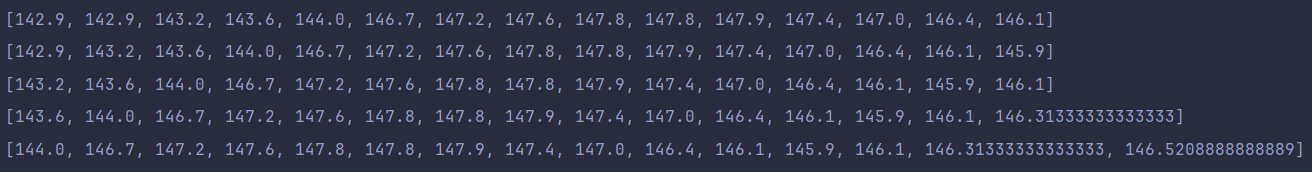


Рисунок 19 – Скользящая средняя во второй вкладке

Далее были оставлены комментарии к коду. См. рис. 20, 21.

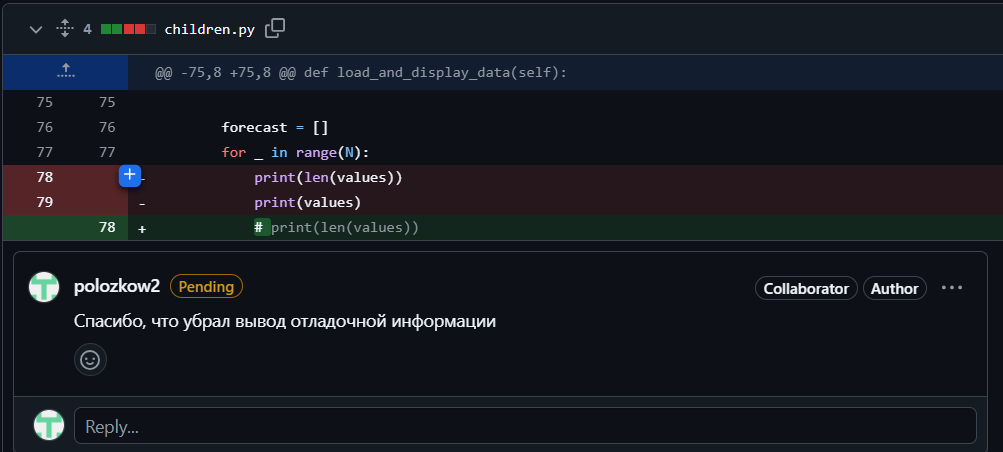


Рисунок 20 – Хвалебный комментарий к коду



Рисунок 21 – Второй комментарий к коду

Далее со второго аккаунта были оставлены комментарии к коду на код ревью. См. рис. 22, 23.

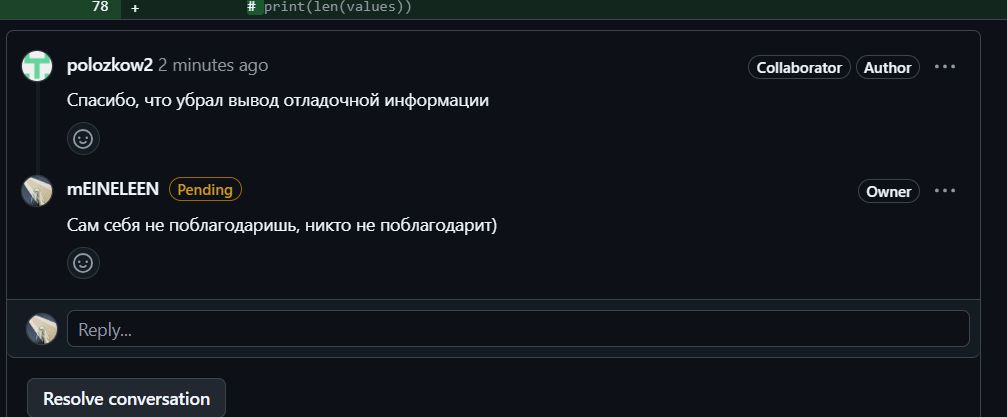


Рисунок 22 – Комментарии к коду со второго аккаунта

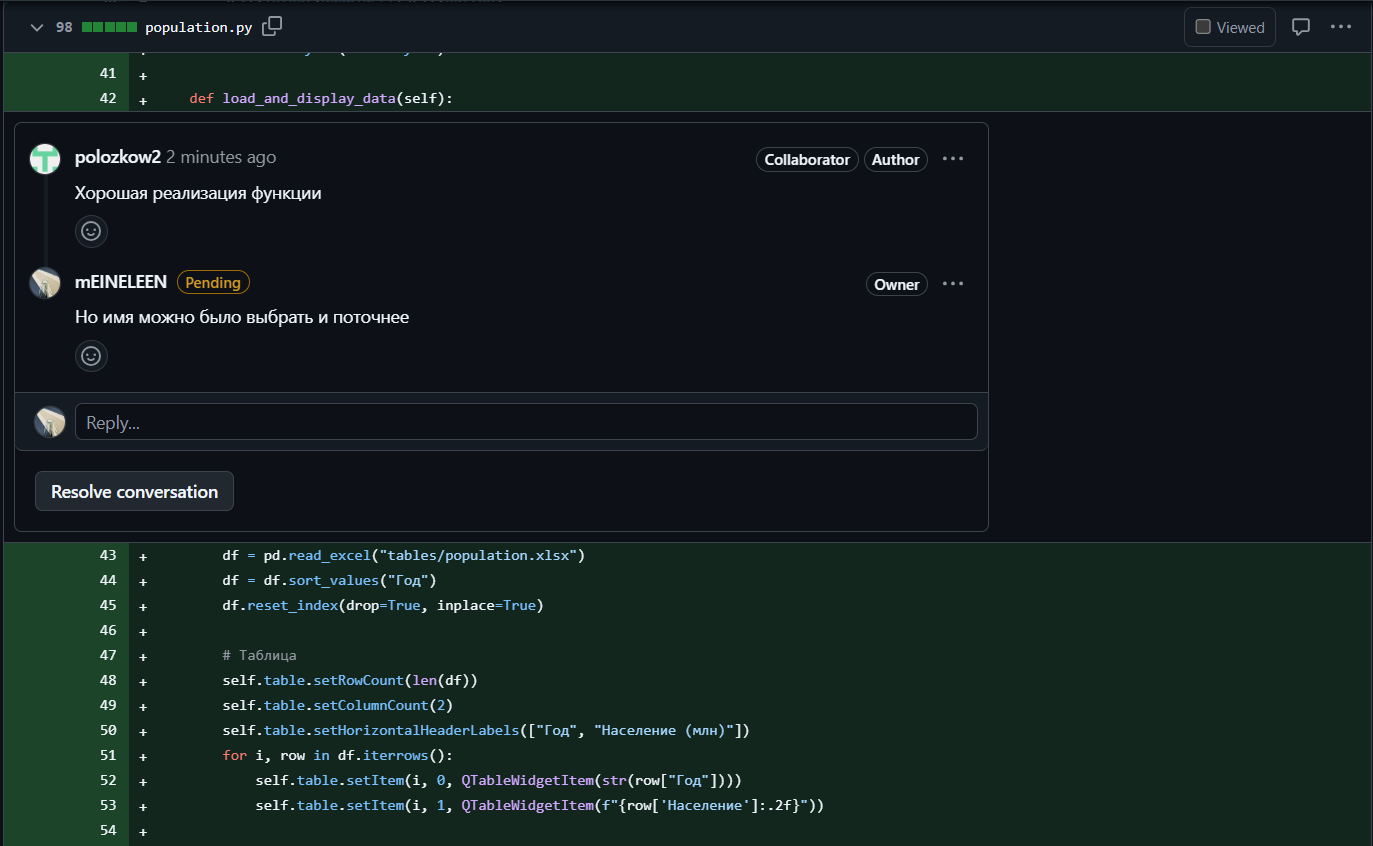


Рисунок 23 – Второй комментарий к коду со второго аккаунта

Далее показан результат объединения всех веток. См. рис. 24.

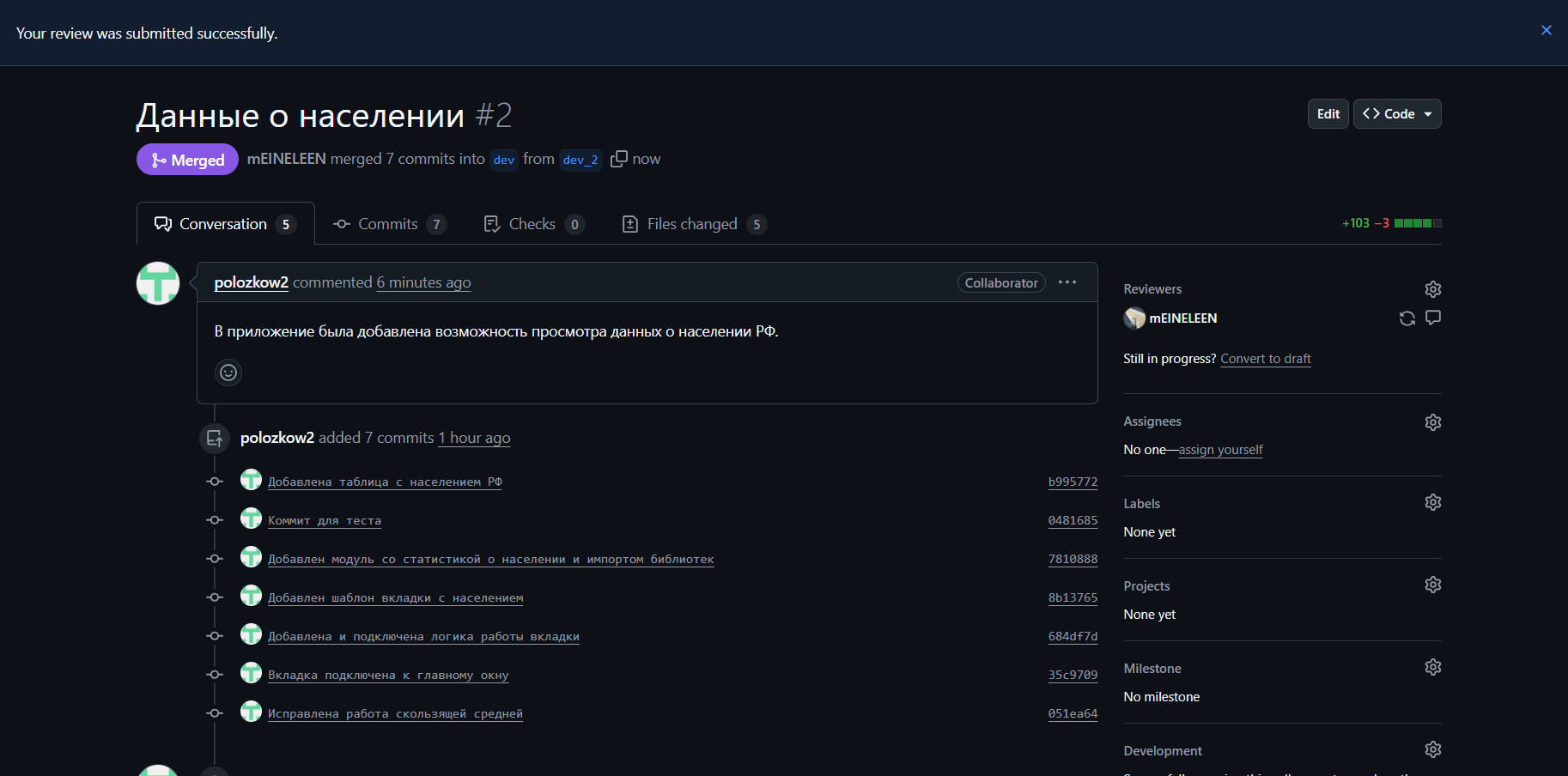


Рисунок 24 – Второе ветки dev\_2 и dev

Далее показан итоговый репозиторий и история коммитов. См. рис. 25, 26.

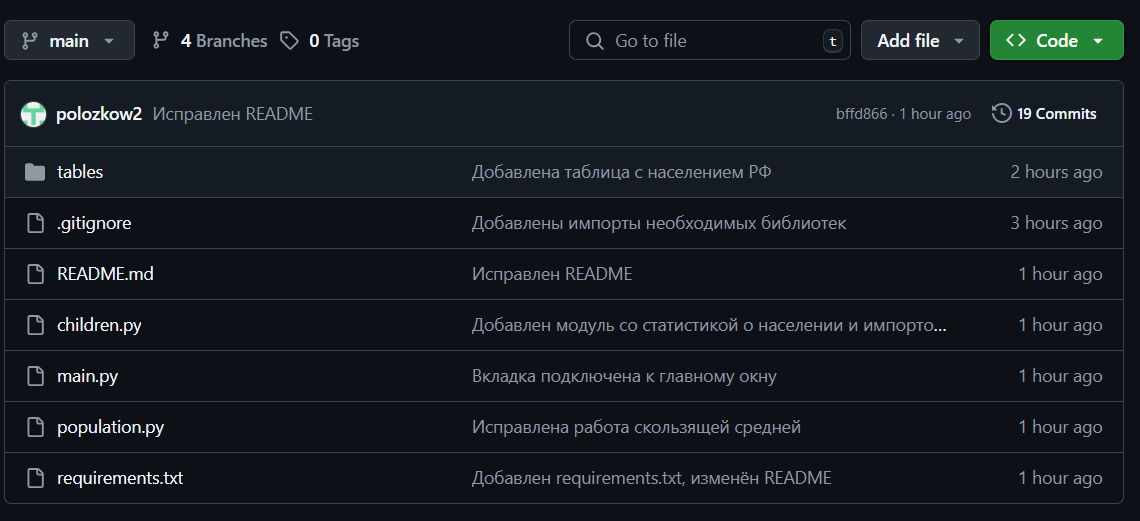


Рисунок 25 – Итоговый репозиторий

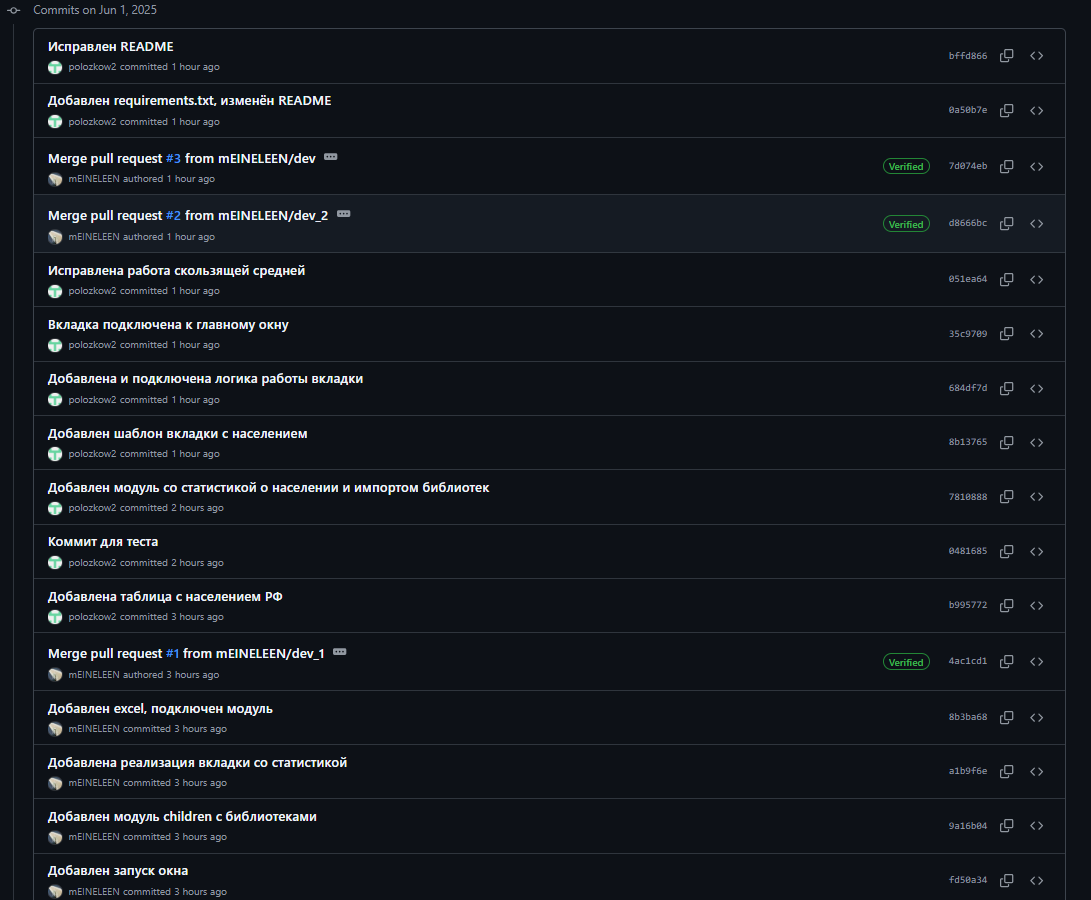


Рисунок 26 – Часть истории коммитов

**Ссылка на GitHub**

<https://github.com/mEINELEEN/TPLr3>

Для реализации задания по 17 варианту использовалась ветка dev\_1 (использовалась пользователем mEINELEEN), для реализации задания по 5 варианту использовалась ветка dev\_2 (использовалась пользователем polozkow2), для обобщения этих веток использовалась ветка dev и для применения финальных изменений использовалась ветка main.

**Вывод**

В ходе работы была реализована и протестирована информационная система, демонстрирующая основные возможности систем контроля версий, а также навыки командной разработки с использованием Git и GitHub. На основе двух заданий было создано приложение с графическим интерфейсом на Python с использованием PyQt5, которое визуализирует и анализирует демографические данные, предоставляя прогноз методом скользящей средней. В процессе разработки были созданы отдельные ветки для каждого участника, организованы pull request'ы и проведён взаимный code review, что обеспечило совместную работу и контроль изменений. Итоговый проект представляет собой программную систему с модульной архитектурой, графиками и табличными представлениями данных, оформленную в соответствии с требованиями командной разработки.

**Приложение А**

**«Код программы из модуля main.py»**

import sys  
from PyQt5.QtWidgets import (  
 QApplication, QWidget, QVBoxLayout, QTabWidget,  
)  
from children import ChildrenStatsTab  
from population import PopulationStatsTab  
  
  
class MainWindow(QWidget):  
 def \_\_init\_\_(*self*):  
 super().\_\_init\_\_()  
 *self*.setWindowTitle("Статистика по заданию")  
 *self*.resize(900, 600)  
  
 layout = QVBoxLayout()  
  
 *self*.tabs = QTabWidget()  
  
 *self*.tabs.addTab(ChildrenStatsTab(), "Дети вне брака")  
 *self*.tabs.addTab(PopulationStatsTab(), "Популяция")  
 # Сюда можно добавлять другие вкладки  
  
 layout.addWidget(*self*.tabs)  
 *self*.setLayout(layout)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 window = MainWindow()  
 window.show()  
 sys.exit(app.exec\_())

**Приложение Б**

**«Код программы из модуля children.py»**

import pandas as pd  
from PyQt5.QtWidgets import (  
 QWidget, QVBoxLayout, QPushButton,  
 QLabel, QTableWidget, QTableWidgetItem,  
 QSpinBox, QHBoxLayout  
)  
from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas  
from matplotlib.figure import Figure  
  
  
class ChildrenStatsTab(QWidget):  
 def \_\_init\_\_(*self*):  
 super().\_\_init\_\_()  
  
 *self*.layout = QVBoxLayout()  
  
 *self*.label = QLabel("Нажмите 'Собрать статистику', чтобы отобразить данные")  
 *self*.layout.addWidget(*self*.label)  
  
 *self*.table = QTableWidget()  
 *self*.layout.addWidget(*self*.table)  
  
 *self*.plot\_widget = FigureCanvas(Figure(figsize=(5, 3)))  
 *self*.ax = *self*.plot\_widget.figure.add\_subplot(111)  
 *self*.layout.addWidget(*self*.plot\_widget)  
  
 control\_layout = QHBoxLayout()  
 *self*.n\_label = QLabel("Прогноз на N лет:")  
 control\_layout.addWidget(*self*.n\_label)  
 *self*.n\_spinbox = QSpinBox()  
 *self*.n\_spinbox.setRange(1, 20)  
 *self*.n\_spinbox.setValue(5)  
 control\_layout.addWidget(*self*.n\_spinbox)  
 *self*.layout.addLayout(control\_layout)  
  
 *self*.button = QPushButton("Собрать статистику")  
 *self*.button.clicked.connect(*self*.load\_and\_display\_data)  
 *self*.layout.addWidget(*self*.button)  
  
 *self*.setLayout(*self*.layout)  
  
 def load\_and\_display\_data(*self*):  
 # Чтение файла  
 df = pd.read\_excel("tables/children.xlsx")  
 df = df.sort\_values("Год")  
 df.reset\_index(drop=True, inplace=True)  
  
 # Отображение таблицы  
 *self*.table.setRowCount(len(df))  
 *self*.table.setColumnCount(2)  
 *self*.table.setHorizontalHeaderLabels(["Год", "Процент"])  
 for i, row in df.iterrows():  
 *self*.table.setItem(i, 0, QTableWidgetItem(str(row["Год"])))  
 *self*.table.setItem(i, 1, QTableWidgetItem(f"{row["Процент"]:.2f}"))  
  
 # Построение графика  
 *self*.ax.clear()  
 *self*.ax.plot(df["Год"], df["Процент"], marker='o', label="Факт")  
  
 # Расчёт изменений  
 df['Δ%'] = df['Процент'].pct\_change() \* 100  
 max\_growth = df['Δ%'].max()  
 min\_growth = df['Δ%'].min()  
  
 *self*.label.setText(  
 f"Макс. рост: {max\_growth:.2f}%, Мин. рост: {min\_growth:.2f}%"  
 )  
  
 # Прогнозирование (скользящая средняя)  
 N = *self*.n\_spinbox.value()  
 k = 15 # длина окна скользящего среднего  
  
 # Начальные значения — последние k точек  
 values = df['Процент'].tolist()[-k:]  
  
 forecast = []  
 for \_ in range(N):  
 # print(len(values))  
 # print(values)  
 next\_value = sum(values[-k:]) / k  
 forecast.append(next\_value)  
 values = values[1:] + [next\_value] # удаляем старое, добавляем новое  
  
 last\_year = df['Год'].iloc[-1]  
 forecast\_years = [last\_year + i for i in range(1, N + 1)]  
 *self*.ax.plot(forecast\_years, forecast, linestyle='--', color='red', marker='x', label='Прогноз')  
  
 *self*.ax.set\_title("Процент детей, рождённых вне брака")  
 *self*.ax.set\_xlabel("Год")  
 *self*.ax.set\_ylabel("Процент")  
 *self*.ax.legend()  
 *self*.ax.grid(True)  
  
 *self*.plot\_widget.draw()

**Приложение В**

**«Код программы из модуля population.py»**

import pandas as pd  
import numpy as np  
from PyQt5.QtWidgets import (  
 QWidget, QVBoxLayout, QLabel, QPushButton, QTableWidget,  
 QTableWidgetItem, QHBoxLayout, QSpinBox  
)  
from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas  
from matplotlib.figure import Figure  
  
  
class PopulationStatsTab(QWidget):  
 def \_\_init\_\_(*self*):  
 super().\_\_init\_\_()  
  
 *self*.layout = QVBoxLayout()  
  
 *self*.label = QLabel("Нажмите 'Собрать статистику', чтобы отобразить данные")  
 *self*.layout.addWidget(*self*.label)  
  
 *self*.table = QTableWidget()  
 *self*.layout.addWidget(*self*.table)  
  
 *self*.plot\_widget = FigureCanvas(Figure(figsize=(5, 3)))  
 *self*.ax = *self*.plot\_widget.figure.add\_subplot(111)  
 *self*.layout.addWidget(*self*.plot\_widget)  
  
 control\_layout = QHBoxLayout()  
 *self*.n\_label = QLabel("Прогноз на N лет:")  
 control\_layout.addWidget(*self*.n\_label)  
 *self*.n\_spinbox = QSpinBox()  
 *self*.n\_spinbox.setRange(1, 20)  
 *self*.n\_spinbox.setValue(5)  
 control\_layout.addWidget(*self*.n\_spinbox)  
 *self*.layout.addLayout(control\_layout)  
  
 *self*.button = QPushButton("Собрать статистику")  
 *self*.button.clicked.connect(*self*.load\_and\_display\_data)  
 *self*.layout.addWidget(*self*.button)  
  
 *self*.setLayout(*self*.layout)  
  
 def load\_and\_display\_data(*self*):  
 df = pd.read\_excel("tables/population.xlsx")  
 df = df.sort\_values("Год")  
 df.reset\_index(drop=True, inplace=True)  
  
 # Таблица  
 *self*.table.setRowCount(len(df))  
 *self*.table.setColumnCount(2)  
 *self*.table.setHorizontalHeaderLabels(["Год", "Население (млн)"])  
 for i, row in df.iterrows():  
 *self*.table.setItem(i, 0, QTableWidgetItem(str(row["Год"])))  
 *self*.table.setItem(i, 1, QTableWidgetItem(f"{row['Население']:.2f}"))  
  
 # График  
 *self*.ax.clear()  
 *self*.ax.plot(df["Год"], df["Население"], marker='o', label="Факт")  
  
 # Процентные изменения  
 df["Δ%"] = df["Население"].pct\_change() \* 100  
 max\_growth = df["Δ%"].max()  
 min\_growth = df["Δ%"].min()  
  
 *self*.label.setText(  
 f"Макс. прирост: {max\_growth:.2f}%, Мин. убыль: {min\_growth:.2f}%"  
 )  
  
 # Прогнозирование (скользящее среднее с движущимся окном)  
 N = *self*.n\_spinbox.value()  
 k = 15 # длина окна, можно заменить на переменную  
  
 # Убедимся, что данных минимум k  
 if len(df) < k:  
 *self*.label.setText("Недостаточно данных для расчёта скользящей средней.")  
 return  
  
 # последние k значений  
 values = df['Население'].tolist()[-k:]  
 forecast = []  
  
 for \_ in range(N):  
 next\_value = sum(values) / k  
 forecast.append(next\_value)  
 print(values)  
 values = values[1:] + [next\_value]  
  
 last\_year = df['Год'].iloc[-1]  
 forecast\_years = [last\_year + i for i in range(1, N + 1)]  
  
 # Рисуем прогноз  
 *self*.ax.plot(forecast\_years, forecast, linestyle='--', color='red', marker='x', label='Прогноз')  
  
 *self*.ax.set\_title("Численность населения России")  
 *self*.ax.set\_xlabel("Год")  
 *self*.ax.set\_ylabel("Млн человек")  
 *self*.ax.grid(True)  
 *self*.ax.legend()  
 *self*.plot\_widget.draw()