**Федеральное государственное автономное образовательное**

**учреждение**

**Высшего образования**

«Национальный исследовательский университет

«Высшая школа экономики»

Информатика и Вычислительная техника

Московский институт электроники и математики им. А.Н. Тихонова НИУ ВШЭ

**Курс: Проектный семинар «Python в науке о данных»**

**ОТЧЕТ**

**ПО ГРУППОВОМУ ПРОЕКТУ**

Работу выполнили

студент группы БИВ-235 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.М. Ястребов

подпись, дата

студент группы БИВ-235 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.О. Саушкин

подпись, дата

студент группы БИВ-235 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.В. Петрунина

подпись, дата

Работу проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.В. Полякова

подпись, дата

# Оглавление

[**Технические требования**](#_dbhgpws7etg4) **3**

[**Версии библиотек**](#_6j5upvbvh9) **4**

[**Архитектура приложения**](#_msrqzwi7dgot) **5**

[**Структура директорий**](#_go6khryhsf8a) **6**

[**Листинг скриптов**](#_fdkfw05ng9e4) **7**

# Технические требования

Средняя мощность оперативной памяти компьютера - 8Gb.

Средняя мощность видеокарты - 2Gb.

Любая операционная система, на которую возможна установка интерпретатора Python 3.8.5

# Версии библиотек

Приложение использует некоторые библиотеки. Их список с указанием версий представлен ниже.

|  |  |
| --- | --- |
| **Библиотека** | **Версия** |
| Tkinter | 8.6 |
| pandas | 1.4.2 |
| sys | 3.8.5 |
| numpy | 1.22.2 |

таблица 1. Версии библиотек

# Архитектура приложения

Приложение содержит 4 модуля, которые находятся в директории

Work/Scripts.

Модуль filther\_by\_country\_whithprogram.py служит для создания графического интерфейса на tkinter для фильтрации данных из CSV-файла и сохранения отфильтрованных данных в новые файлы CSV.

Модуль graphics\_reports.py служит для загрузки данных о рейтингах университетов из нескольких CSV-файлов, создании различных типов графиков для визуализации этих данных и сохранении их.

Модуль pivot\_data.py нормализует данные о рейтингах университетов до первой, второй и третьей нормальных форм, создает отдельные таблицы для университетов, рейтингов и стран, и сохраняет эти таблицы в CSV-файлы.

Модуль text\_reports.py загружает данные о рейтингах университетов, определяет различные функции для анализа этих данных и сохраняет результаты в CSV-файлы в директории.

# Структура директорий

Приложение состоит из системы директорий, представленной ниже.

|  |  |  |
| --- | --- | --- |
| Первый уровень | Второй уровень | Описание |
| Work |  | Директория, которая  содержит все файлы,  необходимые для  разработчика и  обеспечивающие  корректную работу  приложения |
|  | Data | Директория, которая  содержит все файлы с  данными |
|  | Scripts | Директория, которая содержит библиотеку функций, разработанных бригадой, которые можно использовать для создания других приложений. |
|  | Output | Директория, в которой лежат результаты работы программы |
|  | Graphics | Директория, в которой лежат копии графических отчетов |
|  | Notes | Директория с документацией по проекту: Руководство пользователя и руководство разработчика. |

таблица 2. Структура директорий

# Листинг скриптов

Все имеющиеся функции приложения описаны ниже.

|  |  |
| --- | --- |
| **Модуль** | **Описание функций** |
| filther\_by\_country\_whithprogram.py | **def filter\_data\_and\_save\_by\_country(df, country, output\_filename):**  Цель: Фильтрация данных по стране и сохранение их в файл.  Вход: df : DataFrame, country : str, output\_filename : str Выход: Нет  **def filter\_data\_and\_save\_by\_score(df, min\_score, max\_score, output\_filename):**  Цель: Фильтрация данных по диапазону баллов и сохранение их в файл.  Вход: df : DataFrame, min\_score : float, max\_score : float, output\_filename : str  Выход: Нет  **def filter\_data\_and\_save\_by\_year(df, year, output\_filename):**  Цель: Фильтрация данных по году и сохранение их в файл.  Вход: df : DataFrame, year : int, output\_filename : str Выход: str - путь к файлу  **def get\_selected\_year(year\_listbox):**  Цель: Получение выбранного года и фильтрация данных по этому году.  Вход: year\_listbox : Listbox  Выход: Нет  **def filter\_data\_and\_save\_by\_patents(df, min\_patents, max\_patents, output\_filename):**  Цель: Фильтрация данных по диапазону патентов и сохранение их в файл.  Вход: df : DataFrame, min\_patents : int, max\_patents : int, output\_filename : str  Выход: Нет  **def create\_gui():**  Цель: Создание основного графического интерфейса приложения.  Вход: Нет  Выход: Нет  **def show\_country\_window():**  Цель: Создание окна для выбора страны.  Вход: Нет  Выход: Нет  **def get\_selected\_country(country\_listbox):**  Цель: Получение выбранной страны и фильтрация данных по этой стране.  Вход: country\_listbox : Listbox  Выход: Нет  **def show\_score\_range\_window():**  Цель: Создание окна для выбора диапазона баллов.  Вход: Нет  Выход: Нет  **def apply\_score\_range\_filter():**  Цель: Применение фильтрации по диапазону баллов и сохранение данных.  Вход: entry\_min\_score : Entry, entry\_max\_score : Entry  Выход: Нет  **def show\_year\_window():**  Цель: Создание окна для выбора года.  Вход: Нет  Выход: Нет  **def show\_patents\_window():**  Цель: Создание окна для выбора диапазона патентов.  Вход: Нет  Выход: Нет  **def apply\_patents\_filter(entry\_min\_patents, entry\_max\_patents):**  Цель: Применение фильтрации по диапазону патентов и сохранение данных.  Вход: entry\_min\_patents : Entry, entry\_max\_patents : Entry  Выход: Нет |
| graphics\_reports.py | **def clustered\_bar\_chart\_qual\_qual(universities\_subset):**  Цель: Построение кластеризованной столбчатой диаграммы для визуализации количества университетов по странам.  Вход: Данные о университетах  Выход: сохранение и отображение графика  **def pie\_chart\_qual\_qual(universities\_subset):**  Цель: Построение круговой диаграммы для визуализации распределения университетов по странам.  Вход: Данные о университетах  Выход: сохранение и отображение графика.  **def categorized\_histogram\_quant\_qual(rankings\_subset):**  Цель: Построение гистограммы для визуализации распределения оценок университетов.  Вход: Данные о рейтингах университетов  Выход: сохранение и отображение графика  **def categorized\_boxplot\_quality\_education\_by\_country(universities\_subset):**  Цель: Построение ящичной диаграммы для визуализации качества образования по странам.  Вход: Данные о университетах  Выход: Нет  **def quality\_of\_education\_vs\_alumni\_employment\_by\_country\_categorized\_scatterplot(rankings\_subset):**  Цель: Построение диаграммы рассеивания для визуализации качества образования и трудоустройства выпускников по странам.  Вход: Данные о университетах  Выход: сохранение и отображение графика  **def average\_university\_score\_by\_country(data):**  Цель: Построение столбчатой диаграммы для визуализации среднего рейтинга университетов по странам.  Вход: Данные о университетах  Выход: сохранение и отображение графика  **def scatter\_plot(rankings\_subset):**  Цель: Построение диаграммы рассеивания для визуализации мирового рейтинга и оценок университетов по странам.  Вход: Данные о университетах  Выход: сохранение и отображение графика  **def clustered\_bar\_citations(first\_country, second\_country):**  Цель: Построение кластеризованной столбчатой диаграммы для сравнения количества цитирований между двумя странами.  Вход: Данные о университетах  Выход: сохранение и отображение графика |
| text\_reports.py | **def get\_universities\_with\_highest\_ranking(data\_frame, country):**  Цель: Определение университетов с самым высоким рейтингом из определенной страны  Вход: Данные о университетах, название страны  Выход: датафрейм с университетами из заданной страны и их позициями  **def most\_popular\_country\_function(data\_frame):**  Цель: Определение самой популярной страны с университетами.  Вход: Данные о университетах  Выход: датафрейм с количеством университетов по каждой стране  **def score\_most\_popular\_country(data\_frame):**  Цель: Определение общего рейтинга для каждой страны  Вход: Данные о университетах  Выход: датафрейм с общим рейтингом для каждой страны  **def get\_top\_institutions(data\_frame, column\_name, n):**  Цель: Поиск топового заданного числа университетов по заданному критерию.  Вход: Данные о университетах, название столбца, количество топовых университетов  Выход: отсортированный датафрейм с топовыми университетами  **def get\_filtered\_with\_scores(df, min\_score, max\_score):**  Цель: Определение университетов с рейтингом между минимальным и максимальным, которые вводит пользователь.  Вход: Данные о университетах, max и min рейтинг  Выход: датафрейм с университетами в заданном диапазоне рейтинга  **def get\_filtered\_with\_years(df, min\_year, max\_year):**  Цель: Определение университетов с годами между.  Вход: Данные о университетах, min и max год  Выход: датафрейм с университетами в заданном диапазоне лет |

таблица 3. Листинг скриптов

Листинг основного скрипта:

import os

import tkinter as tk

from tkinter import messagebox, filedialog

import pandas as pd

import matplotlib.pyplot as plt

# Список для хранения истории действий

action\_history = []

# Вывод авторов

def show\_authors():

"""

Функция для отображения информации об авторах программы

author: Petrunina Maria

"""

messagebox.showinfo("Авторы", "Авторы:\nМария Петрунина\n"

"Ястребов Арсений\n"

"Саушкин Николай")

# Загружаем таблицу при нажатии кнопки "Работа с таблицами"

def load\_table():

"""

Функция для загрузки таблицы из файла и отображения опций работы с таблицей

author: Petrunina Maria

"""

file\_path = filedialog.askopenfilename(title="Выберите файл таблицы",

filetypes=(("CSV files", "\*.csv"), ("Excel files", "\*.xlsx")))

if file\_path:

try:

# Попытка загрузить файл таблицы

df = pd.read\_csv(file\_path) if file\_path.endswith('.csv') else pd.read\_excel(file\_path)

# Показать сообщение о загрузке таблицы

messagebox.showinfo("Загрузить таблицу", f"Файл {file\_path} загружен")

# Показать окно с кнопками для работы с таблицей

show\_table\_options(df)

# Добавить действие в историю

action\_history.append(load\_table)

except Exception as e:

# В случае ошибки показать сообщение об ошибке

messagebox.showerror("Ошибка", f"Не удалось загрузить файл: {e}")

# Кнопки, которые выводятся после загрузки таблицы

def show\_table\_options(df):

"""

Функция для отображения окна с опциями работы с таблицей

Parameters:

- df: pandas.DataFrame

Загруженная таблица данных

author: Saushkin Nikolay

"""

table\_window = tk.Toplevel(root)

table\_window.title("Работа с таблицами")

table\_window.geometry(f"{app\_width}x{app\_height}+{x}+{y}")

show\_button = tk.Button(table\_window, text="Вывести таблицу", command=lambda: show\_table\_content(df))

show\_button.place(x=125, y=50, width=140, height=35)

edit\_button = tk.Button(table\_window, text="Изменить таблицу", command=lambda: edit\_table(df))

edit\_button.place(x=125, y=150, width=140, height=35)

plot\_button = tk.Button(table\_window, text="Построить график", command=lambda: show\_graph\_options(df.head(100)))

plot\_button.place(x=125, y=250, width=140, height=35)

# Добавить кнопку "Назад"

back\_button = tk.Button(table\_window, text="Назад", command=go\_back)

back\_button.place(x=150, y=350, width=80, height=20)

# Функция для отображения содержимого таблицы

def show\_table\_content(df):

"""

Функция для отображения содержимого таблицы в новом окне

Parameters:

- df: pandas.DataFrame

Таблица с данными

author: Saushkin Nikolay

"""

# Создание нового окна для отображения содержимого загруженной таблицы

content\_window = tk.Toplevel(root)

content\_window.title("Содержимое загруженной таблицы")

content\_window.geometry("900x900")

# Создание виджета текста для отображения содержимого таблицы

text\_widget = tk.Text(content\_window, wrap="none", width=100, height=30)

text\_widget.insert("end", df.to\_string(index=False))

text\_widget.pack(expand=True, fill="both")

# Отключение возможности редактирования текста

text\_widget.config(state="disabled")

# Добавить кнопку "Назад"

back\_button = tk.Button(content\_window, text="Назад", command=go\_back)

back\_button.pack()

def save\_edited\_table(original\_df, edited\_text):

"""

Функция для сохранения отредактированной таблицы в файл

Parameters:

- original\_df: pandas.DataFrame

Исходная таблица данных

- edited\_text: str

Отредактированный текст таблицы

author: Saushkin Nikolay

"""

file\_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=(("CSV files", "\*.csv"), ("All files", "\*.\*")))

if file\_path:

try:

with open(file\_path, 'w') as file:

file.write(edited\_text)

messagebox.showinfo("Файл сохранен", f"Отредактированные данные были сохранены в:\n{file\_path}")

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось сохранить файл: {e}")

def edit\_table(df):

"""

Функция для отображения окна с опциями редактирования таблицы

Parameters:

- df: pandas.DataFrame

Таблица с данными

author: Saushkin Nikolay

"""

edit\_window = tk.Toplevel(root)

edit\_window.title("Изменить таблицу")

edit\_window.geometry("300x200")

button\_select\_country = tk.Button(edit\_window, text="Фильтровать по стране", command=lambda: show\_country\_window(df, get\_selected\_country))

button\_select\_country.pack()

button\_select\_score\_range = tk.Button(edit\_window, text="Фильтровать по диапазону баллов", command=lambda: show\_score\_range\_window(df))

button\_select\_score\_range.pack()

button\_select\_year = tk.Button(edit\_window, text="Фильтровать по году", command=lambda: show\_year\_window(df, get\_selected\_year))

button\_select\_year.pack()

button\_full\_edit = tk.Button(edit\_window, text="Полное редактирование", command=lambda: show\_full\_edit\_window(df))

button\_full\_edit.pack()

def get\_selected\_country(filtered\_df, country\_listbox):

"""

Функция для фильтрации данных по выбранной стране и сохранения отфильтрованных данных в файл

Parameters:

- filtered\_df: pandas.DataFrame

Таблица с данными

- country\_listbox: tkinter.Listbox

Виджет списка стран

author: Yastrebov Arseniy

"""

selected\_country\_index = country\_listbox.curselection()

if selected\_country\_index:

selected\_country = country\_listbox.get(selected\_country\_index[0])

# Создаем папку, если ее нет

folder\_path = os.path.join(os.getcwd(), selected\_country)

if not os.path.exists(folder\_path):

os.makedirs(folder\_path)

# Формируем путь к файлу

file\_path = os.path.join(folder\_path, f"{selected\_country}.csv")

# Сохраняем отфильтрованные данные в CSV файл

filtered\_df.to\_csv(file\_path, index=False)

messagebox.showinfo("Файл сохранен", f"Отфильтрованные данные были сохранены в:\n{file\_path}")

else:

messagebox.showinfo("Не выбрана страна", "Пожалуйста, выберите страну.")

def get\_selected\_year(df, year\_listbox):

"""

Функция для фильтрации данных по выбранному году и сохранения отфильтрованных данных в файл

Parameters:

- df: pandas.DataFrame

Таблица с данными

- year\_listbox: tkinter.Listbox

Виджет списка годов

author: Yastrebov Arseniy

"""

selected\_index = year\_listbox.curselection()

if selected\_index:

selected\_year = year\_listbox.get(selected\_index[0])

# Фильтруем данные по выбранному году

filtered\_df = df[df['year'] == int(selected\_year)]

if not filtered\_df.empty:

# Создаем папку, если ее нет

folder\_path = os.path.join(os.getcwd(), "Filtered Data")

if not os.path.exists(folder\_path):

os.makedirs(folder\_path)

# Формируем путь к файлу

file\_path = os.path.join(folder\_path, f"data\_{selected\_year}.csv")

# Сохраняем отфильтрованные данные в CSV файл

filtered\_df.to\_csv(file\_path, index=False)

messagebox.showinfo("Файл сохранен", f"Отфильтрованные данные были сохранены в:\n{file\_path}")

print("Selected year:", selected\_year)

else:

messagebox.showinfo("Пустые данные", "Нет данных для выбранного года")

else:

print("No year selected")

def show\_country\_window(df, callback\_function):

"""

Функция для отображения окна выбора страны для фильтрации данных

Parameters:

- df: pandas.DataFrame

Таблица с данными

- callback\_function: function

Функция для применения фильтрации

author: Yastrebov Arseniy

"""

def apply\_filter():

selected\_country\_index = country\_listbox.curselection()

if selected\_country\_index:

selected\_country = country\_listbox.get(selected\_country\_index[0])

filtered\_df = df[df['country'] == selected\_country]

callback\_function(filtered\_df, country\_listbox)

else:

messagebox.showinfo("Нет выбора", "Пожалуйста, выберите страну.")

country\_window = tk.Toplevel(root)

country\_window.title("Выберите страну")

country\_label = tk.Label(country\_window, text="Выберите страну:")

country\_label.pack()

country\_listbox = tk.Listbox(country\_window)

unique\_countries = df['country'].unique()

for country in unique\_countries:

country\_listbox.insert(tk.END, country)

country\_listbox.pack()

select\_button = tk.Button(country\_window, text="Выбрать", command=apply\_filter)

select\_button.pack()

def show\_full\_edit\_window(df):

"""

Функция для отображения окна полного редактирования таблицы

Parameters:

- df: pandas.DataFrame

Таблица с данными

author: Petrunina Maria

"""

def save\_changes():

edited\_text = text\_widget.get("1.0", "end")

save\_edited\_table(df, edited\_text)

full\_edit\_window = tk.Toplevel(root)

full\_edit\_window.title("Полное редактирование таблицы")

full\_edit\_window.geometry("400x300")

text\_widget = tk.Text(full\_edit\_window, wrap="none", width=60, height=15)

text\_widget.insert("end", df.to\_string(index=False))

text\_widget.pack(expand=True, fill="both")

save\_button = tk.Button(full\_edit\_window, text="Сохранить изменения", command=save\_changes)

save\_button.pack()

def show\_score\_range\_window(df):

"""

Функция для отображения окна выбора диапазона баллов для фильтрации данных

Parameters:

- df: pandas.DataFrame

Таблица с данными

author: Petrunina Maria

"""

score\_range\_window = tk.Toplevel(root)

score\_range\_window.title("Выбор диапазона баллов")

score\_range\_window.geometry("300x150")

min\_score\_label = tk.Label(score\_range\_window, text="Минимальный балл:")

min\_score\_label.pack()

min\_score\_entry = tk.Entry(score\_range\_window)

min\_score\_entry.pack()

max\_score\_label = tk.Label(score\_range\_window, text="Максимальный балл:")

max\_score\_label.pack()

max\_score\_entry = tk.Entry(score\_range\_window)

max\_score\_entry.pack()

apply\_button = tk.Button(score\_range\_window, text="Применить фильтр",

command=lambda: apply\_score\_filter(df, min\_score\_entry.get(), max\_score\_entry.get()))

apply\_button.pack()

def apply\_score\_filter(df, min\_score, max\_score):

try:

min\_score = float(min\_score)

max\_score = float(max\_score)

filtered\_df = df[(df['score'] >= min\_score) & (df['score'] <= max\_score)]

show\_table\_content(filtered\_df)

except ValueError:

messagebox.showerror("Ошибка", "Некорректные значения баллов")

def show\_year\_window(df, callback\_function):

"""

Функция для отображения окна выбора года для фильтрации данных

Parameters:

- df: pandas.DataFrame

Таблица с данными

- callback\_function: function

Функция для применения фильтрации

author: Petrunina Maria

"""

year\_window = tk.Toplevel()

year\_window.title("Select Year")

year\_label = tk.Label(year\_window, text="Select a year:")

year\_label.pack()

year\_listbox = tk.Listbox(year\_window)

unique\_years = df['year'].unique() # Corrected variable name

for year in unique\_years:

year\_listbox.insert(tk.END, year)

year\_listbox.pack()

select\_button = tk.Button(year\_window, text="Select", command=lambda: callback\_function(df, year\_listbox))

select\_button.pack()

def show\_patents\_window():

# ...

pass

# Функция для отображения окна с выбором графика

def show\_graph\_options(df):

"""

Функция для отображения окна с опциями построения графиков

Parameters:

- df: pandas.DataFrame

Таблица с данными

author: Saushkin Nikolay

"""

graph\_window = tk.Toplevel(root)

graph\_window.title("Выбор типа графика")

graph\_window.geometry("400x300")

clustered\_button = tk.Button(graph\_window, text="Clustered bar chart qual qual",

command=lambda: plot\_clustered\_bar\_chart(df))

clustered\_button.place(x=50, y=50, width=300, height=35)

categorized\_histogram\_button = tk.Button(graph\_window, text="Categorized histogram quant qual",

command=lambda: plot\_categorized\_histogram(df))

categorized\_histogram\_button.place(x=50, y=100, width=300, height=35)

categorized\_boxplot\_button = tk.Button(graph\_window, text="Categorized boxplot quality education by country",

command=lambda: plot\_categorized\_boxplot(df))

categorized\_boxplot\_button.place(x=50, y=150, width=300, height=35)

unknown\_graph1\_button = tk.Button(graph\_window, text="Average University Score by Country",

command=lambda: plot\_unknown\_graph1(df))

unknown\_graph1\_button.place(x=50, y=200, width=300, height=35)

unknown\_graph2\_button = tk.Button(graph\_window, text="Pie chart qual qual",

command=lambda: plot\_unknown\_graph2(df))

unknown\_graph2\_button.place(x=50, y=250, width=300, height=35)

scatter\_plot\_button = tk.Button(graph\_window, text="Scatter Plot",

command=lambda: plot\_scatter\_plot(df))

scatter\_plot\_button.place(x=50, y=300, width=300, height=35)

# Добавить кнопку "Назад"

back\_button = tk.Button(graph\_window, text="Назад", command=graph\_window.destroy)

back\_button.place(x=150, y=350, width=80, height=20)

# Функции для построения различных видов графиков

def plot\_clustered\_bar\_chart(df):

plt.figure(figsize=(10, 6))

df.groupby('country')['national\_rank'].count().plot(kind='bar')

plt.xlabel('Country')

plt.ylabel('Number of Universities')

plt.title('Clustered bar chart qual qual')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.savefig('clustered\_bar\_chart\_qual\_qual.png')

plt.show()

def plot\_categorized\_histogram(df):

plt.figure(figsize=(10, 6))

plt.hist(df['score'], bins=10, alpha=0.5, label='Score')

plt.xlabel('Score')

plt.ylabel('Frequency')

plt.title('Categorized histogram quant qual')

plt.legend()

plt.tight\_layout()

plt.savefig('categorized\_histogram\_quant\_qual.png')

plt.show()

def plot\_categorized\_boxplot(df):

plt.figure(figsize=(10, 6))

plt.boxplot(df['score'])

plt.xlabel('Score')

plt.ylabel('Score')

plt.title('Categorized boxplot quality education by country')

plt.xticks([1], ['Overall'])

plt.tight\_layout()

plt.savefig('categorized\_boxplot\_quality\_education\_by\_country.png')

plt.show()

def plot\_unknown\_graph1(df):

# Группировка данных по странам и вычисление среднего рейтинга в каждой стране

average\_score\_by\_country = df.groupby('country')['score'].mean()

# Вычисление минимального и максимального среднего

min\_score = average\_score\_by\_country.min() \* 0.9

max\_score = average\_score\_by\_country.max() \* 1.1

plt.figure(figsize=(12, 6))

plt.ylim(min\_score, max\_score) # Установка пределов по оси y, для лучшей видимости, можно убрать

average\_score\_by\_country.plot(kind='bar', color='skyblue')

plt.title('Average University Score by Country')

plt.xlabel('Country')

plt.ylabel('Average Score')

plt.xticks(rotation=90)

plt.grid(axis='y')

plt.tight\_layout()

plt.savefig('Average\_University\_Score\_by\_Country.png')

plt.show()

def plot\_unknown\_graph2(df):

plt.figure(figsize=(8, 8))

df['country'].value\_counts().plot(kind='pie', autopct='%1.1f%%', startangle=140, fontsize=8)

plt.axis('equal')

plt.title('Pie chart qual qual')

plt.tight\_layout()

plt.savefig('pie\_chart\_qual\_qual.png')

plt.show()

def plot\_scatter\_plot(df):

plt.figure(figsize=(10, 6))

plt.boxplot([df[df['country'] == country]['score'] for country in df['country'].unique()],

labels=df['country'].unique(), notch=True, patch\_artist=True)

plt.xlabel('Country')

plt.ylabel('Score')

plt.title('Scatter Plot')

plt.xticks(rotation=90)

plt.tick\_params(axis='both', which='major', labelsize=5)

plt.tight\_layout()

plt.savefig('scatter\_plot.png')

plt.show()

# Функция для возврата к предыдущему действию

def go\_back():

"""

Функция для возврата на предыдущий шаг в истории действий

author: Saushkin Nikolay

"""

if action\_history:

action = action\_history.pop()

action()

root = tk.Tk()

root.title("Python таблицы")

# Размеры экрана

screen\_width = root.winfo\_screenwidth()

screen\_height = root.winfo\_screenheight()

# Размеры окна

app\_width = 400

app\_height = 400

# Расчет координат для центрирования окна

x = (screen\_width - app\_width) // 2

y = (screen\_height - app\_height) // 2

root.geometry(f"{app\_width}x{app\_height}+{x}+{y}")

# Настройки кнопки "Авторы" на главном экране

authors\_button = tk.Button(root, text="Авторы", command=show\_authors)

authors\_button.place(x=150, y=350, width=120, height=35)

# Настройки кнопки "Работа с таблицами"

table\_button = tk.Button(root, text="Работа с таблицами", command=load\_table)

table\_button.place(x=125, y=150, width=170, height=35)

root.mainloop()