ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

*Московский институт электроники и математики им. А.Н. Тихонова*

# Руководство разработчика по работе с приложением: «HR for students».

Разработчик:

Давтян Э.Д., БИВ225

Телефон: 89996310612  
Почта: eddavtyan@edu.hse.ru

Руководитель:

Полякова М. В.

Москва 2023 г.

Оглавление

[Руководство разработчика по работе с приложением: «HR for students». 1](#_Toc137232125)

[Требования к характеристикам компьютера и операционной системе 3](#_Toc137232126)

[Версии интерпретатора и используемых библиотек 3](#_Toc137232127)

[Инструкция по установке приложения 3](#_Toc137232128)

[Инструкция по запуску и настройке приложения 4](#_Toc137232129)

[Описание структуры БД 5](#_Toc137232130)

[Структура каталогов 6](#_Toc137232131)

[Архитектура приложения 6](#_Toc137232132)

[Листинг основного скрипта и всех модулей. 7](#_Toc137232133)

## Требования к характеристикам компьютера и операционной системе

* 64-битная версия Microsoft Windows 10, 8, 7 (SP1);
* RAM: не менее 4 ГБ, рекомендуется 8 ГБ;
* 1,5 ГБ сводного места на жестком диске + не менее 1 ГБ для кэша;
* разрешение экрана — не менее 1024×768 пикселей;
* Python 2.7, Python 3.5 или более поздняя версия.

## Версии интерпретатора и используемых библиотек

Интерпретатор – Python 2.7+.

Используемые библиотеки:

Таблица 1

|  |  |
| --- | --- |
| **Библиотека** | **Версия** |
| pandas | 2.0.1 |
| matplotlib | 3.7.1 |
| numpy | 1.24.3 |
| pandastable | 0.13.1 |
| tkinter | 8.6 |

# Инструкция по установке приложения

1. Убедитесь, что у вас установлен Python версии 3.x. Если его нет, загрузите и установите Python с официального сайта: <https://www.python.org/downloads/>.
2. Загрузите все необходимые файлы приложения, включая **libraries.py**, **scripts.py** и другие зависимости, также библиотеки, указанные в документе «Руководство разработчика».
3. Создайте новую папку для проекта и скопируйте загруженные файлы в эту папку.
4. Откройте командную строку (терминал) и перейдите в папку проекта, используя команду **cd <путь\_к\_папке\_проекта>**.
5. Установите все необходимые зависимости и библиотеки, выполнив следующую команду:

**pip install <название библиотеки>**

1. Установите переменную окружения **PYTHONPATH** для указания пути к вашей папке проекта. Это позволит Python найти и импортировать модули из вашего проекта. Для этого выполните следующую команду:
   * На Windows:

**setx PYTHONPATH "<путь\_к\_папке\_проекта>"**

* + На macOS/Linux:

**export PYTHONPATH="<путь\_к\_папке\_проекта>"**

1. Замените **<путь\_к\_папке\_проекта>** на фактический путь к вашей папке проекта.
2. После установки зависимостей и настройки переменной окружения приложение готово к запуску и использованию.

## Инструкция по запуску и настройке приложения

1. Откройте командную строку (терминал) и перейдите в папку проекта, используя команду **cd <путь\_к\_папке\_проекта>**.
2. Запустите приложение, выполнив следующую команду:

**python <имя\_главного\_модуля>.py**

Замените **<имя\_главного\_модуля>** на имя файла, содержащего код приложения.

1. После запуска приложения откроется графический интерфейс, готовый для работы с таблицами данных HR.
2. Вы можете использовать различные функциональные кнопки и действия, предоставляемые приложением, для работы с таблицами, импорта, экспорта, поиска, создания отчетов и т.д.
3. Настройте приложение в соответствии с вашими потребностями, используя функциональные возможности, предоставленные в интерфейсе.
4. По окончании работы с приложением закройте его, нажав на кнопку закрытия окна.

Примечания:

* Убедитесь, что у вас установлена библиотека **tkinter**, так как она является одной из зависимостей приложения.
* При необходимости установите другие зависимости, указанные в коде приложения или в его требованиях.
* Убедитесь, что в пути к запускаемому файлу нет ошибок и вы указываете правильное имя главного модуля для запуска приложения.
* Приложение предоставляет графический интерфейс для работы с таблицами данных HR и имеет различные функции, описанные в коде.

## Описание структуры БД

Все базы данных имеют поля:

1. StID: Уникальный идентификатор студента (Student ID).
2. YearsOfExp: Количество лет опыта работы (Years of Experience).
3. FuncCompSc: Оценка компьютерных навыков (Functional Computer Science).
4. MathScore: Оценка по математике (Math Score).
5. ReadingScore: Оценка по чтению (Reading Score).
6. WritingScore: Оценка по письму (Writing Score).
7. Top1SkillsScore: Оценка лучшего навыка (Top 1 Skills Score).
8. Top2SkillsScore: Оценка второго лучшего навыка (Top 2 Skills Score).
9. Top3SkillsScore: Оценка третьего лучшего навыка (Top 3 Skills Score).
10. BehCompScore: Оценка компетенций в поведении (Behavioral Competence Score).
11. Top1BehaviorSkillScore: Оценка лучшего навыка в поведении (Top 1 Behavioral Skill Score).
12. Top2BehaviorSkillScore: Оценка второго лучшего навыка в поведении (Top 2 Behavioral Skill Score).
13. Top3BehaviorSkillScore: Оценка третьего лучшего навыка в поведении (Top 3 Behavioral Skill Score).
14. Gender: Пол студента.
15. EthnicGroup: Этническая принадлежность студента.
16. ParentEduc: Образование родителей.
17. TestsPrep: Подготовка к тестам.
18. Romantic: Состояние отношений (состоит в них или нет).
19. ParentMaritalStatus: Семейное положение родителей.
20. PracticeSport: Занятие спортом.
21. IsFirstChild: Является ли студент старшим ребенком в семье.
22. NrSiblings: Количество братьев и сестер.
23. Transport: Вид транспорта.
24. studytime: Время, затрачиваемое на учебу.
25. CallForInterview: Результат собеседования.

## Структура каталогов

Таблица 2

|  |  |  |
| --- | --- | --- |
| **Первый уровень** | **Второй уровень** | **Объяснение** |
| Work |  | Основной каталог с подкаталогами |
|  | Data | Содержит базу данных. |
|  | Graphics | Содержит копии графических отчетов. |
|  | Library | Содержит библиотеку стандартных (универсальных) функций, разработанных бригадой, которые могут использоваться для создания других приложений, например функции чтения файлов. |
|  | Notes | Содержит документацию, в нем размещается Руководства пользователя и разработчика. |
|  | Output | Содержит копии текстовых отчетов. |
|  | Scripts | Содержит специализированный модуль и файл с определением параметров настройки приложения. |

## Архитектура приложения

Таблица 3

|  |  |  |
| --- | --- | --- |
| **Модуль** | **Местонахождение** | **Функция** |
| main.py | Work/Scripts | Основной код, основной файл интерфейса |
| libraries.py | Work/Library | Построение базовой статистики |
| scripts.py | Work/Scripts | Построение графиков |

## Листинг основного скрипта и всех модулей.

Таблица 4

|  |  |
| --- | --- |
| **Модуль** | **Функции с докстрингами** |
| main.py """  Описание программы: Графический интерфейс для работы с таблицами данных HR.  Автор: Давтян Эдмон  """ | if \_\_name\_\_ == '\_\_main\_\_':  """  Основная функция программы.  """  def import\_table\_wrapper():  """  Функция-обертка для импорта таблицы.  Вызывает функцию import\_tab с переданным объектом приложения.  """  def save\_table\_wrapper():  """  Функция-обертка для сохранения таблицы.  Вызывает функцию save\_tab с переданным объектом приложения.  """  def add\_table\_wrapper():  """  Функция-обертка для добавления таблицы.  Вызывает функцию add\_tab с переданным объектом приложения.  """  def pivot\_table\_wrapper():  """  Функция-обертка для объединения таблиц.  Вызывает функцию pivot\_tab с переданным объектом приложения.  """  def search\_wrapper():  """  Функция-обертка для поиска в таблице.  Вызывает функцию search с переданным объектом приложения.  """  def add\_rows\_wrapper():  """  Функция-обертка для добавления строк в таблицу.  Вызывает функцию add\_rows с переданным объектом приложения.  """  def add\_cols\_wrapper():  """  Функция-обертка для добавления столбцов в таблицу.  Вызывает функцию add\_cols с переданным объектом приложения.  """  def delete\_rows\_wrapper():  """  Функция-обертка для удаления строк из таблицы.  Вызывает функцию delete\_rows с переданным объектом приложения.  """  def delete\_cols\_wrapper():  """  Функция-обертка для удаления столбцов из таблицы.  Вызывает функцию delete\_cols с переданным объектом приложения.  """  def open\_image\_wrapper():  """  Функция-обертка для открытия изображения.  Вызывает функцию open\_image с переданным объектом приложения.  """  def graphics():  """  Функция-обертка для генерации графиков.  Вызывает функцию grap с переданным объектом приложения.  """  def stat():  """  Функция-обертка для генерации статистического отчета.  Вызывает функцию statistical\_report с переданным объектом приложения.  """  def simple():  """  Функция-обертка для генерации простого отчета.  Вызывает функцию generate\_report с переданным объектом приложения.  ""  def merge\_tab\_wrapper():  """  Функция-обертка для объединения таблиц.  Вызывает функцию merge\_tab с переданным объектом приложения.  """ |
| libraries.py  """  Этот скрипт создает приложение Tkinter для выбора и анализа зависимостей.  Автор: Давтян Эдмон  """ | def import\_tab(app\_local):  """  Импортирует таблицу из файла в приложение.  Предлагает пользователю выбрать файл и определяет его расширение. В зависимости от расширения,  функция загружает таблицу из файла Excel (xls или xlsx) или импортирует ее из файла CSV.  Args:  app\_local (Tk): Экземпляр приложения Tkinter.  Returns:  None  """  def add\_tab(app\_local):  """  Добавляет новую вкладку с дочерней таблицей в приложение.  Функция создает новую дочернюю таблицу в основной таблице приложения. Пользователю предлагается  выбрать файл и определить его расширение. Если расширение файла - 'csv', пользователю предлагается  ввести разделитель и данные из файла CSV импортируются в дочернюю таблицу. Если расширение файла  - 'xls' или 'xlsx', данные загружаются из файла Excel в дочернюю таблицу. Если расширение файла  не 'csv', 'xls' или 'xlsx', отображается сообщение об ошибке.  Args:  app\_local (Tk): Экземпляр приложения Tkinter.  Returns:  None  """  def pivot\_tab(app\_local):  """  Генерирует отчет сводной таблицы на основе выбранных атрибутов и метода агрегации.  Эта функция открывает окно, позволяющее пользователю выбрать два атрибута из данных таблицы  и указать метод агрегации. Затем она генерирует сводную таблицу на основе выбранных атрибутов  и отображает результат в новом окне. Пользователь может сохранить отчет в виде текстового файла.  Аргументы:  app\_local (Tk): Экземпляр приложения Tkinter.  Возвращает:  None  """  def save\_tab(app\_local):  """  Сохраняет данные таблицы в файл.  Параметры:  app\_local (object): Объект приложения, содержащий таблицу.  Возвращает:  None  """  def merge\_tab(app\_local):  """  Объединяет таблицы.  Параметры:  app\_local (object): Объект приложения, содержащий таблицы.  Возвращает:  None  """  def search(app\_local):  """  Открывает строку поиска в приложении для выполнения запросов к таблице.  Параметры:  app\_local (object): Объект приложения, в котором будет открыта строка поиска.  Возвращает:  None  """  def add\_rows(app\_local):  """  Добавляет строки в таблицу в приложении.  Параметры:  - app\_local (object): Объект приложения, в котором находится таблица.  Возвращает:  None  """  def add\_cols(app\_local):  """  Добавляет столбцы в таблицу в приложении.  Параметры:  - app\_local (object): Объект приложения, в котором находится таблица.  Возвращает:  None  """  def delete\_rows(app):  """  Удаляет строки из таблицы в приложении.  Параметры:  - app (object): Объект приложения, в котором находится таблица.  Возвращает:  None  """  def delete\_cols(app):  """  Удаляет столбцы из таблицы в приложении.  Параметры:  - app (object): Объект приложения, в котором находится таблица.  Возвращает:  None  """  def statistical\_report(app\_local):  """  Генерирует статистический отчет на основе выбранного атрибута в таблице.  Параметры:  - app\_local (object): Объект приложения, содержащий таблицу.  Возвращает:  None  """  def generate\_report(app\_local):  """  Генерирует отчет на основе выбранных столбцов и отфильтрованных строк.  Параметры:  - app\_local: Объект приложения, содержащий таблицу.  """  class TestApp(tk.Frame):  """  TestApp - пользовательский класс на основе фрейма Tkinter, представляющий основное приложение.  Предоставляет графический интерфейс пользователя (GUI) для отображения и взаимодействия с таблицами.  Атрибуты:  - parent: Родительский виджет.  Методы:  - \_\_init\_\_(self, parent): Инициализирует класс TestApp.  """ |
| scripts.py """  Скрипт создает приложение Tkinter для выбора и анализа зависимостей.  Автор: Давтян Эдмон  """ | def check(variables):  """  Устанавливает значение первых шести переменных равным 1.  Аргументы:  variables (список): Список переменных.  Возвращает:  None  """  def grap(app\_local):  """  Создает окно с флажками и кнопками для выбора и анализа зависимостей.  Аргументы:  app\_local (tk.Tk): Объект приложения Tkinter.  Возвращает:  None  """  def create\_grap(app\_local, vars\_list):  """  Создает график на основе выбранных переменных.  Аргументы:  app\_local (object): Объект приложения.  vars\_list (list): Список выбранных переменных.  Возвращает:  None  """  def open\_image(app\_local):  """  Открывает и отображает изображение в новом окне.  Args:  app\_local (Tk): Экземпляр приложения Tkinter.  Returns:  None  """ |

Листинг основного скрипта и всех модулей  
Листинг основного скрипта main.py:  
*"""  
Описание программы: Графический интерфейс для работы с таблицами данных HR.  
Автор: Давтян Эдмон  
"""*import tkinter as tk  
import libraries  
import scripts  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 """  
 Основная функция программы.  
 """  
 root = tk.Tk()  
 root.geometry('1280x720+10+10')  
 root.title('HR for students')  
  
 topframe = tk.Frame(root, bg='#006666')  
 topframe.pack(side=tk.TOP)  
  
 app\_local = libraries.TestApp(root)  
 app\_local.pack(fill=tk.BOTH, expand=1)  
  
 button\_bg = '#80c9b3'  
 button\_fg = 'white'  
  
 def import\_table\_wrapper():  
 *"""  
 Функция-обертка для импорта таблицы.  
  
 Вызывает функцию import\_tab с переданным объектом приложения.  
 """* libraries.import\_tab(app\_local)  
  
  
 def save\_table\_wrapper():  
 *"""  
 Функция-обертка для сохранения таблицы.  
  
 Вызывает функцию save\_tab с переданным объектом приложения.  
 """* libraries.save\_tab(app\_local)  
  
  
 def add\_table\_wrapper():  
 *"""  
 Функция-обертка для добавления таблицы.  
  
 Вызывает функцию add\_tab с переданным объектом приложения.  
 """* libraries.add\_tab(app\_local)  
  
  
 def pivot\_table\_wrapper():  
 *"""  
 Функция-обертка для объединения таблиц.  
  
 Вызывает функцию pivot\_tab с переданным объектом приложения.  
 """* libraries.pivot\_tab(app\_local)  
  
  
 def search\_wrapper():  
 *"""  
 Функция-обертка для поиска в таблице.  
  
 Вызывает функцию search с переданным объектом приложения.  
 """* libraries.search(app\_local)  
  
  
 def add\_rows\_wrapper():  
 *"""  
 Функция-обертка для добавления строк в таблицу.  
  
 Вызывает функцию add\_rows с переданным объектом приложения.  
 """* libraries.add\_rows(app\_local)  
  
  
 def add\_cols\_wrapper():  
 *"""  
 Функция-обертка для добавления столбцов в таблицу.  
  
 Вызывает функцию add\_cols с переданным объектом приложения.  
 """* libraries.add\_cols(app\_local)  
  
  
 def delete\_rows\_wrapper():  
 *"""  
 Функция-обертка для удаления строк из таблицы.  
  
 Вызывает функцию delete\_rows с переданным объектом приложения.  
 """* libraries.delete\_rows(app\_local)  
  
  
 def delete\_cols\_wrapper():  
 *"""  
 Функция-обертка для удаления столбцов из таблицы.  
  
 Вызывает функцию delete\_cols с переданным объектом приложения.  
 """* libraries.delete\_cols(app\_local)  
  
  
 def open\_image\_wrapper():  
 *"""  
 Функция-обертка для открытия изображения.  
  
 Вызывает функцию open\_image с переданным объектом приложения.  
 """* scripts.open\_image(app\_local)  
  
  
 def graphics():  
 *"""  
 Функция-обертка для генерации графиков.  
  
 Вызывает функцию grap с переданным объектом приложения.  
 """* scripts.grap(app\_local)  
  
  
 def stat():  
 *"""  
 Функция-обертка для генерации статистического отчета.  
  
 Вызывает функцию statistical\_report с переданным объектом приложения.  
 """* libraries.statistical\_report(app\_local)  
  
  
 def simple():  
 *"""  
 Функция-обертка для генерации простого отчета.  
  
 Вызывает функцию generate\_report с переданным объектом приложения.  
 """* libraries.generate\_report(app\_local)  
  
  
 def merge\_tab\_wrapper():  
 *"""  
 Функция-обертка для объединения таблиц.  
  
 Вызывает функцию merge\_tab с переданным объектом приложения.  
 """* libraries.merge\_tab(app\_local)  
  
 b1 = tk.Button(topframe, text="Открыть", command=import\_table\_wrapper, bg=button\_bg, fg=button\_fg)  
 b1.pack(side=tk.LEFT, padx=5, pady=5)  
 b2 = tk.Button(topframe, text="Найти", command=search\_wrapper, bg=button\_bg, fg=button\_fg)  
 b2.pack(side=tk.LEFT, padx=5, pady=5)  
 b3 = tk.Button(topframe, text="Графики", command=graphics, bg=button\_bg, fg=button\_fg)  
 b3.pack(side=tk.LEFT, padx=5, pady=5)  
 b4 = tk.Button(topframe, text="Названия столбцов", command=open\_image\_wrapper, bg=button\_bg, fg=button\_fg)  
 b4.pack(side=tk.LEFT, padx=5, pady=5)  
 b5 = tk.Button(topframe, text="Сохранить как...", command=save\_table\_wrapper, bg=button\_bg, fg=button\_fg)  
 b5.pack(side=tk.LEFT, padx=5, pady=5)  
 b6 = tk.Button(topframe, text="Сводная таблица", command=pivot\_table\_wrapper, bg=button\_bg, fg=button\_fg)  
 b6.pack(side=tk.LEFT, padx=5, pady=5)  
 b7 = tk.Button(topframe, text="Статистический отчет", command=stat, bg=button\_bg, fg=button\_fg)  
 b7.pack(side=tk.LEFT, padx=5, pady=5)  
 b8 = tk.Button(topframe, text="Простой отчет", command=simple, bg=button\_bg, fg=button\_fg)  
 b8.pack(side=tk.LEFT, padx=5, pady=5)  
 b9 = tk.Button(topframe, text="+Объект", command=add\_rows\_wrapper, bg=button\_bg, fg=button\_fg)  
 b9.pack(side=tk.LEFT, padx=5, pady=5)  
 b10 = tk.Button(topframe, text="+Параметр", command=add\_cols\_wrapper, bg=button\_bg, fg=button\_fg)  
 b10.pack(side=tk.LEFT, padx=5, pady=5)  
 b11 = tk.Button(topframe, text="-Объект", command=delete\_rows\_wrapper, bg=button\_bg, fg=button\_fg)  
 b11.pack(side=tk.LEFT, padx=5, pady=5)  
 b12 = tk.Button(topframe, text="-Параметр", command=delete\_cols\_wrapper, bg=button\_bg, fg=button\_fg)  
 b12.pack(side=tk.LEFT, padx=5, pady=5)  
 b13 = tk.Button(topframe, text="Добавить", command=add\_table\_wrapper, bg=button\_bg, fg=button\_fg)  
 b13.pack(side=tk.LEFT, padx=5, pady=5)  
 b14 = tk.Button(topframe, text="Объединить", command=merge\_tab\_wrapper, bg=button\_bg, fg=button\_fg)  
 b14.pack(side=tk.LEFT, padx=5, pady=5)  
  
 root.mainloop()

Листинг скрипта scripts.py  
  
*"""  
Скрипт создает приложение Tkinter для выбора и анализа зависимостей.  
  
Автор: Давтян Эдмон  
"""*from tkinter import Label, Button, Checkbutton, IntVar  
from tkinter.constants import LEFT  
from PIL import ImageTk, Image  
import matplotlib.pyplot as plt  
import tkinter as tk  
import numpy as np  
from tkinter import messagebox as mbx  
  
  
def check(variables):  
 *"""  
 Устанавливает значение первых шести переменных равным 1.  
  
 Аргументы:  
 variables (список): Список переменных.  
  
 Возвращает:  
 None  
 """* for var in variables[:6]:  
 var.set(1)  
  
  
def grap(app\_local):  
 *"""  
 Создает окно с флажками и кнопками для выбора и анализа зависимостей.  
  
 Аргументы:  
 app\_local (tk.Tk): Объект приложения Tkinter.  
  
 Возвращает:  
 None  
 """* # Создаем список переменных для флажков  
 vars\_list = [IntVar() for i in range(6)]  
  
 # Создаем новое окно  
 newWindow = tk.Toplevel(app\_local)  
 newWindow.geometry('540x360+100+100')  
  
 # Создаем метку  
 labelExample = tk.Label(newWindow, text="Какие зависимости вы бы хотели проанализировать?")  
 labelExample.pack()  
  
 # Устанавливаем значения флажков по умолчанию  
 for i in range(6):  
 vars\_list[i].set(-1)  
  
 # Создаем флажки  
 c0 = Checkbutton(newWindow, text="Распределение периода занятости среди юношей",  
 variable=vars\_list[0], onvalue=1, offvalue=0)  
  
 c1 = Checkbutton(newWindow, text="Распределение периода занятости среди девушек",  
 variable=vars\_list[1], onvalue=1, offvalue=0)  
  
 c2 = Checkbutton(newWindow, text="Распределение студентов по часам самоподготовки в неделю",  
 variable=vars\_list[2], onvalue=1, offvalue=0)  
  
 c3 = Checkbutton(newWindow, text="Соотношение результата технического теста к количеству часов самоподготовки",  
 variable=vars\_list[3], onvalue=1, offvalue=0)  
  
 c4 = Checkbutton(newWindow, text="Распределение принятия на работу от этнической группы",  
 variable=vars\_list[4], onvalue=1, offvalue=0)  
  
 c5 = Checkbutton(newWindow, text="Распределение итоговых оценок в зависимости от статуса отношений",  
 variable=vars\_list[5], onvalue=1, offvalue=0)  
  
 # Создаем кнопки  
 all\_button = Button(newWindow, text="Выбрать все", command=lambda: check(vars\_list))  
 plot\_button = Button(newWindow, text="Построить графики", command=lambda: create\_grap(app\_local, vars\_list))  
  
 # Размещаем флажки и кнопки в окне  
 c0.pack()  
 c1.pack()  
 c2.pack()  
 c3.pack()  
 c4.pack()  
 c5.pack()  
 all\_button.pack()  
 plot\_button.pack()  
  
  
def create\_grap(app\_local, vars\_list):  
 *"""  
 Создает график на основе выбранных переменных.  
  
 Аргументы:  
 app\_local (object): Объект приложения.  
 vars\_list (list): Список выбранных переменных.  
  
 Возвращает:  
 None  
 """* # Получаем данные из модели таблицы приложения  
 data = app\_local.table.model.df  
  
 # Создаем столбец 'st\_time' и заполняем его значениями np.nan  
 data['st\_time'] = np.nan  
 df = [data]  
  
 # Преобразуем значения столбца 'studytime' в текстовые метки  
 if 'studytime' in data:  
 for col in df:  
 col.loc[col['studytime'] == 1, 'st\_time'] = 'меньше 2'  
 col.loc[col['studytime'] == 2, 'st\_time'] = 'от 2 до 5'  
 col.loc[col['studytime'] == 3, 'st\_time'] = 'от 5 до 10'  
 col.loc[col['studytime'] == 4, 'st\_time'] = 'больше 10'  
  
 # Подсчитываем количество выбранных переменных  
 counter = sum([1 if var.get() == 1 else 0 for var in vars\_list])  
  
 print(counter)  
 if counter == 0:  
 return  
 if counter == 1:  
 # Создаем график с одним подграфиком  
 plt.rcParams['font.size'] = '20'  
 fig, ax = plt.subplots()  
 elif counter == 2:  
 # Создаем график с двумя подграфиками  
 plt.rcParams['font.size'] = '10'  
 fig, ax = plt.subplots(1, 2)  
 elif counter <= 4:  
 # Создаем график с четырьмя подграфиками  
 plt.rcParams['font.size'] = '10'  
 fig, ax = plt.subplots(2, 2)  
 maxj = 2  
 elif counter <= 6:  
 # Создаем график с шестью подграфиками  
 plt.rcParams['font.size'] = '5'  
 fig, ax = plt.subplots(2, 3)  
 maxj = 3  
 else:  
 # Создаем график с восемью подграфиками  
 plt.rcParams['font.size'] = '5'  
 fig, ax = plt.subplots(2, 4)  
 maxj = 4  
  
 error = set()  
 i, j = 0, 0  
 if vars\_list[0].get() == 1:  
 if 'YearsOfExp' in data and 'Gender' in data:  
 x = np.arange(3)  
 grades = list(data[(data.Gender == "male")]["YearsOfExp"])  
 y = np.array([grades.count(i) / len(grades) for i in range(3)])  
 if counter < 2:  
 ax.bar(x, y)  
 ax.set\_title("Распределение периода занятости среди юношей")  
 ax.set\_xlabel("Количество лет")  
 ax.set\_ylabel("Доля юношей")  
 ax.set\_xticks(x)  
 ax.set\_xticklabels(range(3))  
 ax.set\_xlim([0, 2])  
 elif counter == 2:  
 ax[j].bar(x, y)  
 ax[j].set\_title("Распределение периода занятости среди юношей")  
 ax[j].set\_xlabel("Количество лет")  
 ax[j].set\_ylabel("Доля юношей")  
 ax[j].set\_xticks(x)  
 ax[j].set\_xticklabels(range(3))  
 ax[j].set\_xlim([0, 2])  
 else:  
 ax[i, j].bar(x, y)  
 ax[i, j].set\_title("Распределение периода занятости среди юношей", fontsize=5)  
 ax[i, j].set\_xlabel("Количество лет", fontsize=5)  
 ax[i, j].set\_ylabel("Доля юношей", fontsize=5)  
 ax[i, j].set\_xticks(x)  
 ax[i, j].set\_xticklabels(range(3))  
 ax[i, j].set\_xlim([0, 2])  
 j += 1  
 else:  
 if 'YearsOfExp' not in data:  
 error.add('YearsOfExp')  
 if 'Gender' not in data:  
 error.add('Gender')  
  
 if vars\_list[1].get() == 1:  
 if 'YearsOfExp' in data and 'Gender' in data:  
 x = np.arange(3)  
 grades = list(data[(data.Gender == "female")]["YearsOfExp"])  
 y = np.array([grades.count(i) / len(grades) for i in range(3)])  
 if counter < 2:  
 ax.bar(x, y)  
 ax.set\_title("Распределение периода занятости среди девушек")  
 ax.set\_xlabel("Количество лет")  
 ax.set\_ylabel("Доля девушек")  
 ax.set\_xticks(x)  
 ax.set\_xticklabels(range(3))  
 ax.set\_xlim([0, 2])  
 elif counter == 2:  
 ax[j].bar(x, y)  
 ax[j].set\_title("Распределение периода занятости среди девушек")  
 ax[j].set\_xlabel("Количество лет")  
 ax[j].set\_ylabel("Доля девушек")  
 ax[j].set\_xticks(x)  
 ax[j].set\_xticklabels(range(3))  
 ax[j].set\_xlim([0, 2])  
 else:  
 ax[i, j].bar(x, y)  
 ax[i, j].set\_title("Распределение периода занятости среди женщин", fontsize=5)  
 ax[i, j].set\_xlabel("Количество лет", fontsize=5)  
 ax[i, j].set\_ylabel("Доля женщин", fontsize=5)  
 ax[i, j].set\_xticks(x)  
 ax[i, j].set\_xticklabels(range(3))  
 ax[i, j].set\_xlim([0, 2])  
 j += 1  
 else:  
 if 'YearsOfExp' not in data:  
 error.add('YearsOfExp')  
 if 'Gender' not in data:  
 error.add('Gender')  
  
 if vars\_list[2].get() == 1:  
 if 'studytime' in data:  
 if counter > 2 and j == maxj:  
 i += 1  
 j = 0  
 if counter < 2:  
 ax.hist(data["st\_time"])  
 ax.set\_title("Распределение студентов по часам самоподготовки в неделю")  
 ax.set\_xlabel("Количество часов самоподготовки")  
 ax.set\_ylabel("Количество студентов")  
 elif counter == 2:  
 ax[j].hist(data["st\_time"])  
 ax[j].set\_title("Распределение студентов по часам самоподготовки в неделю")  
 ax[j].set\_xlabel("Количество часов самоподготовки")  
 ax[j].set\_ylabel("Количество студентов")  
 else:  
 ax[i, j].hist(data["st\_time"])  
 ax[i, j].set\_title("Распределение студентов по часам самоподготовки в неделю", fontsize=5)  
 ax[i, j].set\_xlabel("Количество часов самоподготовки", fontsize=5)  
 ax[i, j].set\_ylabel("Количество студентов", fontsize=5)  
  
 j += 1  
 else:  
 if 'studytime' not in data:  
 error.add('studytime')  
  
 if vars\_list[3].get() == 1:  
 if 'studytime' in data and 'FuncCompSc' in data:  
 if counter > 2 and j == maxj:  
 i += 1  
 j = 0  
  
 studytime = [list(data[(data.st\_time == i)]["FuncCompSc"]) for i in  
 ['меньше 2', 'от 2 до 5', 'от 5 до 10', 'больше 10']]  
  
 if counter < 2:  
 ax.boxplot(studytime, vert=0, labels=['меньше 2', 'от 2 до 5', 'от 5 до 10', 'больше 10'])  
 ax.set\_title("Соотношение результата к к/ч самоподготовки")  
 ax.set\_xlabel("Итоговый результат")  
 ax.set\_ylabel("Количество часов самоподготовки")  
 elif counter == 2:  
 ax[j].boxplot(studytime, vert=0, labels=['меньше 2', 'от 2 до 5', 'от 5 до 10', 'больше 10'])  
 ax[j].set\_title("Соотношение результата к к/ч самоподготовки")  
 ax[j].set\_xlabel("Итоговый результат")  
 ax[j].set\_ylabel("Количество часов самоподготовки")  
 else:  
 ax[i, j].boxplot(studytime, vert=0, labels=['меньше 2', 'от 2 до 5', 'от 5 до 10', 'больше 10'])  
 ax[i, j].set\_title("Соотношение результата к к/ч самоподготовки", fontsize=5)  
 ax[i, j].set\_xlabel("Итоговый результат", fontsize=5)  
 ax[i, j].set\_ylabel("Количество часов самоподготовки", fontsize=5)  
  
 j += 1  
 else:  
 if 'FuncCompSc' not in data:  
 error.add('FuncCompSc')  
 if 'studytime' not in data:  
 error.add('studytime')  
  
 if vars\_list[4].get() == 1:  
 if 'EthnicGroup' in data and 'CallForInterview' in data:  
 x = np.array([0, 1])  
 grades1 = list(data[(data.EthnicGroup == "group A")]["CallForInterview"])  
 y1 = np.array([grades1.count(0) / len(grades1), grades1.count(1) / len(grades1)])  
 grades2 = list(data[(data.EthnicGroup == "group B")]["CallForInterview"])  
 y2 = np.array([grades2.count(0) / len(grades2), grades2.count(1) / len(grades2)])  
 grades3 = list(data[(data.EthnicGroup == "group C")]["CallForInterview"])  
 y3 = np.array([grades3.count(0) / len(grades3), grades3.count(1) / len(grades3)])  
 grades4 = list(data[(data.EthnicGroup == "group D")]["CallForInterview"])  
 y4 = np.array([grades4.count(0) / len(grades4), grades4.count(1) / len(grades4)])  
 grades5 = list(data[(data.EthnicGroup == "group E")]["CallForInterview"])  
 y5 = np.array([grades5.count(0) / len(grades5), grades5.count(1) / len(grades5)])  
 if counter < 2:  
 ax.bar(x - 0.2, y1, width=0.4, label="Group A")  
 ax.bar(x - 0.1, y2, width=0.4, label="Group B")  
 ax.bar(x, y3, width=0.4, label="Group C")  
 ax.bar(x + 0.1, y4, width=0.4, label="Group D")  
 ax.bar(x + 0.2, y5, width=0.4, label="Group E")  
 ax.set\_title("Распределение принятия на работу от этнической группы")  
 ax.set\_xlabel("Итог отбора")  
 ax.set\_ylabel("Этническая группа")  
 ax.set\_xticks(x)  
 ax.set\_xticklabels(["Rejected", "Accepted"])  
 ax.legend()  
 elif counter == 2:  
 ax.bar(x - 0.2, y1, width=0.4, label="Group A")  
 ax.bar(x - 0.1, y2, width=0.4, label="Group B")  
 ax.bar(x, y3, width=0.4, label="Group C")  
 ax.bar(x + 0.1, y4, width=0.4, label="Group D")  
 ax.bar(x + 0.2, y5, width=0.4, label="Group E")  
 ax[j].set\_title("Распределение принятия на работу от этнической группы")  
 ax[j].set\_xlabel("Итог отбора")  
 ax[j].set\_ylabel("Этническая группа")  
 ax[j].set\_xticks(x)  
 ax[j].set\_xticklabels(["Rejected", "Accepted"])  
 ax[j].legend()  
 else:  
 ax[i, j].bar(x - 0.2, y1, width=0.4, label="Group A")  
 ax[i, j].bar(x - 0.1, y2, width=0.4, label="Group B")  
 ax[i, j].bar(x, y3, width=0.4, label="Group C")  
 ax[i, j].bar(x + 0.1, y4, width=0.4, label="Group D")  
 ax[i, j].bar(x + 0.2, y5, width=0.4, label="Group E")  
 ax[i, j].set\_title("Распределение принятия на работу от этнической группы", fontsize=5)  
 ax[i, j].set\_xlabel("Итог отбора", fontsize=5)  
 ax[i, j].set\_ylabel("Этническая группа", fontsize=5)  
 ax[i, j].set\_xticks(x)  
 ax[i, j].set\_xticklabels(["Rejected", "Accepted"])  
 ax[i, j].legend()  
  
 j += 1  
 else:  
 if 'CallForInterview' not in data:  
 error.add('CallForInterview')  
 if 'EthnicGroup' not in data:  
 error.add('EthnicGroup')  
  
 if vars\_list[5].get() == 1:  
 if 'Romantic' in data and 'FuncCompSc' in data and 'BehCompScore' in data:  
 if counter > 2 and j == maxj:  
 i += 1  
 j = 0  
  
 if counter < 2:  
 colors = np.where(data['Romantic'] == 'yes', 1, 0)  
 plt.scatter(data['FuncCompSc'], data['BehCompScore'], c=colors, cmap='tab10')  
 plt.title('Распределение итоговых оценок в зависимости от статуса отношений', fontsize=15)  
 plt.xlabel("FuncCompSc")  
 plt.ylabel("BehCompScore")  
 elif counter == 2:  
 colors = np.where(data['Romantic'] == 'yes', 1, 0)  
 ax[j].scatter(data['FuncCompSc'], data['BehCompScore'], c=colors, cmap='tab10')  
 ax[j].set\_title('Распределение итоговых оценок в зависимости от статуса отношений', fontsize=15)  
 ax[j].set\_xlabel("FuncCompSc")  
 ax[j].set\_ylabel("BehCompScore")  
 else:  
 colors = np.where(data['Romantic'] == 'yes', 1, 0)  
 ax[i, j].scatter(data['FuncCompSc'], data['BehCompScore'], c=colors, cmap='tab10')  
 ax[i, j].set\_title('Распределение итоговых оценок в зависимости от статуса отношений', fontsize=5)  
 ax[i, j].set\_xlabel("FuncCompSc")  
 ax[i, j].set\_ylabel("BehCompScore")  
  
 j += 1  
 else:  
 if 'Romantic' not in data:  
 error.add('Romantic')  
 if 'FuncCompSc' not in data:  
 error.add('FuncCompSc')  
 if 'BehCompScore' not in data:  
 error.add('BehCompScore')  
  
 if not error:  
 fig.set\_figwidth(15)  
 fig.set\_figheight(10)  
 if counter <= 4:  
 fig.subplots\_adjust(wspace=0.7, hspace=0.5)  
 else:  
 fig.subplots\_adjust(wspace=0.5, hspace=0.3)  
  
 plt.show()  
 else:  
 mbx.showerror("Ошибка", "Нет необходимых параметров " + ', '.join(error))  
  
  
# Открывает и отображает изображение в новом окне  
def open\_image(app\_local):  
 *"""  
 Открывает и отображает изображение в новом окне.  
  
 Args:  
 app\_local (Tk): Экземпляр приложения Tkinter.  
  
 Returns:  
 None  
 """* # Открытие изображения  
 img = Image.open('../output/HR for students.png')  
 # Изменение размера изображения  
 img = img.resize((1280, 720), Image.ANTIALIAS)  
 # Создание объекта ImageTk для отображения в Tkinter  
 img = ImageTk.PhotoImage(img)  
 # Создание нового окна  
 new = tk.Toplevel(app\_local)  
 # Создание метки для отображения изображения  
 label = Label(new, image=img)  
 # Привязка изображения к метке  
 label.image = img  
 # Размещение метки в окне  
 label.pack(side=LEFT)

Листинг скрипта linraries.py:  
*"""  
Этот скрипт создает приложение Tkinter для выбора и анализа зависимостей.  
  
Автор: Давтян Эдмон  
"""*import os  
import tkinter.filedialog as fl  
import tkinter.messagebox as mbx  
import tkinter.simpledialog as sl  
import tkinter as tk  
import pandas as pd  
import numpy as np  
from pandastable import Table, MultipleValDialog  
  
  
# Импортирует таблицу из файла в приложение  
def import\_tab(app\_local):  
 *"""  
 Импортирует таблицу из файла в приложение.  
  
 Предлагает пользователю выбрать файл и определяет его расширение. В зависимости от расширения,  
 функция загружает таблицу из файла Excel (xls или xlsx) или импортирует ее из файла CSV.  
  
 Args:  
 app\_local (Tk): Экземпляр приложения Tkinter.  
  
 Returns:  
 None  
 """* # Запрос выбора файла у пользователя  
 file\_name = fl.askopenfilename()  
 # Определение расширения файла  
 file\_extension = file\_name.split('.')[-1]  
 # Загрузка таблицы из файла Excel  
 if file\_extension == 'xls' or file\_extension == 'xlsx':  
 app\_local.table.loadExcel(file\_name)  
 # Импорт таблицы из файла CSV  
 elif file\_extension == 'csv':  
 ans = sl.askstring("Разделитель", "Введите разделитель:")  
 app\_local.table.importCSV(file\_name, sep=ans)  
 else:  
 # Ошибка при неподдерживаемом формате файла  
 mbx.showerror("Ошибка", "Файл должен быть в формате csv или xls (xlsx)")  
  
  
def add\_tab(app\_local):  
 *"""  
 Добавляет новую вкладку с дочерней таблицей в приложение.  
  
 Функция создает новую дочернюю таблицу в основной таблице приложения. Пользователю предлагается  
 выбрать файл и определить его расширение. Если расширение файла - 'csv', пользователю предлагается  
 ввести разделитель и данные из файла CSV импортируются в дочернюю таблицу. Если расширение файла  
 - 'xls' или 'xlsx', данные загружаются из файла Excel в дочернюю таблицу. Если расширение файла  
 не 'csv', 'xls' или 'xlsx', отображается сообщение об ошибке.  
  
 Args:  
 app\_local (Tk): Экземпляр приложения Tkinter.  
  
 Returns:  
 None  
 """* # Создание дочерней таблицы в основной таблице  
 app\_local.table.createChildTable(None)  
  
 # Запрос выбора файла у пользователя  
 file\_name = fl.askopenfilename()  
 # Определение расширения файла  
 file\_extension = file\_name.split('.')[-1]  
  
 # Импорт данных из файла CSV  
 if file\_extension == 'csv':  
 ans = sl.askstring("Разделитель", "Введите разделитель:")  
 app\_local.table.child.importCSV(file\_name, sep=ans)  
 # Загрузка данных из файла Excel  
 elif file\_extension == 'xls' or file\_extension == 'xlsx':  
 app\_local.table.child.loadExcel(file\_name)  
 else:  
 # Ошибка при неподдерживаемом формате файла  
 mbx.showerror("Ошибка", "Файл должен быть в формате csv или xls (xlsx)")  
  
  
def pivot\_tab(app\_local):  
 *"""  
 Генерирует отчет сводной таблицы на основе выбранных атрибутов и метода агрегации.  
  
 Эта функция открывает окно, позволяющее пользователю выбрать два атрибута из данных таблицы  
 и указать метод агрегации. Затем она генерирует сводную таблицу на основе выбранных атрибутов  
 и отображает результат в новом окне. Пользователь может сохранить отчет в виде текстового файла.  
  
 Аргументы:  
 app\_local (Tk): Экземпляр приложения Tkinter.  
  
 Возвращает:  
 None  
 """* # Запрос названий атрибутов и метода агрегации  
 attributes = app\_local.table.model.df.columns.tolist()  
  
 # Создание окна выбора атрибутов  
 attribute\_window = tk.Toplevel()  
 attribute\_window.title("Выбор атрибутов")  
 attribute\_window.geometry("300x200")  
  
 # Создание переменных для хранения выбранных атрибутов  
 attribute1\_var = tk.StringVar()  
 attribute2\_var = tk.StringVar()  
  
 # Создание выпадающего списка для первого атрибута  
 attribute1\_label = tk.Label(attribute\_window, text="Первый атрибут:")  
 attribute1\_label.pack()  
 attribute1\_menu = tk.OptionMenu(attribute\_window, attribute1\_var, \*attributes)  
 attribute1\_menu.pack()  
  
 # Создание выпадающего списка для второго атрибута  
 attribute2\_label = tk.Label(attribute\_window, text="Второй атрибут:")  
 attribute2\_label.pack()  
 attribute2\_menu = tk.OptionMenu(attribute\_window, attribute2\_var, \*attributes)  
 attribute2\_menu.pack()  
  
 def generate\_report():  
 *"""  
 Генерирует отчет сводной таблицы на основе выбранных атрибутов и метода агрегации.  
 """* attribute1 = attribute1\_var.get()  
 attribute2 = attribute2\_var.get()  
 aggregation = sl.askstring("Метод агрегации", "Введите метод агрегации (например, sum, mean, max, min):",  
 parent=app\_local.table.parentframe)  
  
 if not attribute1 or not attribute2 or not aggregation:  
 mbx.showwarning("Ошибка", "Необходимо выбрать атрибуты и ввести метод агрегации.",  
 parent=app\_local.table.parentframe)  
 return  
  
 # Создание окна отчета  
 report\_window = tk.Toplevel()  
 report\_window.title("Сводная таблица")  
 report\_window.geometry("750x300")  
  
 # Создание текстового поля для отчета  
 report\_text = tk.Text(report\_window)  
 report\_text.pack(fill=tk.BOTH, expand=True)  
  
 # Создание сводной таблицы  
 pivot\_table = pd.pivot\_table(app\_local.table.model.df, index=[attribute1, attribute2], aggfunc=aggregation)  
  
 # Добавление содержимого сводной таблицы в текстовое поле  
 report\_text.insert(tk.END, str(pivot\_table))  
  
 def save\_report():  
 *"""  
 Сохраняет отчет сводной таблицы в виде текстового файла.  
 """* # Запрашиваем у пользователя путь для сохранения файла  
 file\_path = fl.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "\*.txt")])  
  
 if file\_path:  
 # Сохраняем отчет в указанный файл  
 with open(file\_path, "w") as f:  
 f.write(str(pivot\_table))  
  
 # Создание кнопки для сохранения отчета  
 save\_button = tk.Button(report\_window, text="Сохранить отчет", command=save\_report)  
 save\_button.pack()  
  
 # Запрашиваем у пользователя выбор сохранить отчет или нет при закрытии окна  
 report\_window.protocol("WM\_DELETE\_WINDOW", lambda: handle\_close\_report(report\_window))  
  
 def handle\_close\_report(window):  
 *"""  
 Обрабатывает закрытие окна отчета.  
  
 Эта функция вызывается при закрытии окна отчета. Она запрашивает у пользователя подтверждение  
 сохранения отчета перед закрытием окна. Если пользователь соглашается, отчет сохраняется  
 в виде текстового файла.  
  
 Аргументы:  
 window (Tk): Экземпляр окна отчета.  
  
 Возвращает:  
 None  
 """* save\_confirm = mbx.askyesno("Сохранить отчет", "Хотите сохранить отчет перед закрытием?",  
 parent=app\_local.table.parentframe)  
 if save\_confirm:  
 save\_report()  
 window.destroy()  
  
 # Создание кнопки для генерации отчета  
 generate\_button = tk.Button(attribute\_window, text="Сгенерировать отчет", command=generate\_report)  
 generate\_button.pack()  
  
 # Запуск главного цикла обработки событий окна выбора атрибутов  
 attribute\_window.mainloop()  
  
  
def save\_tab(app\_local):  
 *"""  
 Сохраняет данные таблицы в файл.  
  
 Параметры:  
 app\_local (object): Объект приложения, содержащий таблицу.  
  
 Возвращает:  
 None  
 """* # Запрос пути и имени файла для сохранения  
 filename = fl.asksaveasfilename(parent=app\_local.table.master,  
 defaultextension='.csv',  
 initialdir=os.getcwd(),  
 filetypes=[("csv", "\*.csv"),  
 ("excel", "\*.xls"),  
 ("html", "\*.html"),  
 ("All files", "\*.\*")])  
  
 if filename:  
 # Сохранение данных таблицы в выбранный файл  
 app\_local.table.model.save(filename)  
  
  
def merge\_tab(app\_local):  
 *"""  
 Объединяет таблицы.  
  
 Параметры:  
 app\_local (object): Объект приложения, содержащий таблицы.  
  
 Возвращает:  
 None  
 """* if app\_local.table.child is None:  
 # Проверка на наличие добавочного справочника  
 mbx.showwarning("Нет добавочного справочника", 'Вам необходимо загрузить справочник',  
 parent=app\_local.table.parentframe)  
 return  
  
 # Объединение таблиц  
 merged\_table = pd.merge(app\_local.table.model.df, app\_local.table.child.model.df)  
 print(merged\_table)  
  
 # Обновление данных таблицы  
 app\_local.table.model.df = merged\_table  
 app\_local.table.update()  
 app\_local.table.redraw()  
 app\_local.table.tableChanged()  
  
  
def search(app\_local):  
 *"""  
 Открывает строку поиска в приложении для выполнения запросов к таблице.  
  
 Параметры:  
 app\_local (object): Объект приложения, в котором будет открыта строка поиска.  
  
 Возвращает:  
 None  
 """* app\_local.table.queryBar()  
  
  
def add\_rows(app\_local):  
 *"""  
 Добавляет строки в таблицу в приложении.  
  
 Параметры:  
 - app\_local (object): Объект приложения, в котором находится таблица.  
  
 Возвращает:  
 None  
 """* # Запрос количества объектов для добавления  
 num = sl.askinteger("Сколько объектов добавить?", "Количество объектов:",  
 initialvalue=1, parent=app\_local.table.parentframe)  
 if not num:  
 return  
  
 # Сохранение текущего состояния таблицы  
 app\_local.table.storeCurrent()  
 df = app\_local.table.model.df  
  
 if len(df) == 0:  
 # Если таблица пуста, создание нового DataFrame с указанным количеством строк  
 app\_local.table.model.df = pd.DataFrame(pd.Series(range(num)))  
 return  
  
 try:  
 ind = app\_local.table.model.df.index.max() + 1  
 except:  
 ind = len(df) + 1  
  
 # Создание нового DataFrame с пустыми значениями и указанным количеством строк  
 new = pd.DataFrame(np.nan, index=range(ind, ind + num), columns=df.columns)  
 app\_local.table.model.df = pd.concat([df, new])  
 app\_local.table.model.df = app\_local.table.model.df.convert\_dtypes()  
  
 # Обновление цветов строк таблицы  
 app\_local.table.update\_rowcolors()  
  
 # Перерисовка таблицы  
 app\_local.table.redraw()  
  
 # Изменение таблицы (обновление внутреннего состояния)  
 app\_local.table.tableChanged()  
  
  
def add\_cols(app\_local):  
 *"""  
 Добавляет столбцы в таблицу в приложении.  
  
 Параметры:  
 - app\_local (object): Объект приложения, в котором находится таблица.  
  
 Возвращает:  
 None  
 """* # Задание типов столбцов  
 coltypes = ['object', 'int64']  
  
 # Открытие диалогового окна для ввода типов и имен столбцов  
 d = MultipleValDialog(title='Новый параметр', initialvalues=(coltypes, ''),  
 labels=('Тип параметра', 'Имя параметра'),  
 types=('combobox', 'string'), parent=app\_local.table.parentframe)  
  
 if d.result is None:  
 return  
 else:  
 dtype = d.results[0]  
 newname = d.results[1]  
  
 if newname is not None:  
 if newname in app\_local.table.model.df.columns:  
 # Проверка наличия столбца с таким же именем  
 mbx.showwarning("Ошибка", "Такой параметр уже существует!", parent=app\_local.table.parentframe)  
 else:  
 # Сохранение текущего состояния таблицы  
 app\_local.table.storeCurrent()  
  
 # Создание нового столбца с указанным именем и типом данных  
 data = pd.Series(dtype=dtype)  
 app\_local.table.model.df[newname] = data  
 app\_local.table.model.df = app\_local.table.model.df.convert\_dtypes()  
  
 # Настройка ширины родительского окна таблицы  
 app\_local.table.parentframe.configure(width=app\_local.table.width)  
  
 # Обновление цветов строк таблицы  
 app\_local.table.update\_rowcolors()  
  
 # Перерисовка таблицы  
 app\_local.table.redraw()  
  
 # Изменение таблицы (обновление внутреннего состояния)  
 app\_local.table.tableChanged()  
  
  
def delete\_rows(app):  
 *"""  
 Удаляет строки из таблицы в приложении.  
  
 Параметры:  
 - app (object): Объект приложения, в котором находится таблица.  
  
 Возвращает:  
 None  
 """* # Запрос на количество удаляемых объектов  
 num = sl.askinteger("Сколько объектов удалить?", "Количество объектов:",  
 initialvalue=1, parent=app.table.parentframe)  
 if not num:  
 return  
  
 # Сохранение текущего состояния таблицы  
 app.table.storeCurrent()  
 df = app.table.model.df  
  
 # Проверка на наличие данных в таблице  
 if len(df) == 0:  
 return  
  
 indices = []  
 for \_ in range(num):  
 # Запрос на индекс строки для удаления  
 ind = sl.askinteger("Введите индекс строки для удаления", "Индекс строки:",  
 initialvalue=1, minvalue=1, maxvalue=len(df), parent=app.table.parentframe)  
 if ind is None:  
 return  
 indices.append(ind)  
  
 # Удаление выбранных строк из таблицы  
 app.table.model.df = df.drop(indices)  
 app.table.model.df = app.table.model.df.convert\_dtypes()  
  
 # Обновление цветов строк таблицы  
 app.table.update\_rowcolors()  
  
 # Перерисовка таблицы  
 app.table.redraw()  
  
 # Изменение таблицы (обновление внутреннего состояния)  
 app.table.tableChanged()  
  
  
def delete\_cols(app):  
 *"""  
 Удаляет столбцы из таблицы в приложении.  
  
 Параметры:  
 - app (object): Объект приложения, в котором находится таблица.  
  
 Возвращает:  
 None  
 """* # Запрос имени удаляемого столбца  
 colname = sl.askstring("Удаление столбца", "Введите имя столбца для удаления:",  
 parent=app.table.parentframe)  
 if colname is None:  
 return  
  
 # Проверка наличия столбца с указанным именем  
 if colname not in app.table.model.df.columns:  
 mbx.showwarning("Ошибка",  
 "Столбец с таким именем не существует!",  
 parent=app.table.parentframe)  
 else:  
 # Сохранение текущего состояния таблицы  
 app.table.storeCurrent()  
  
 # Удаление указанного столбца из таблицы  
 app.table.model.df = app.table.model.df.drop(columns=[colname])  
 app.table.model.df = app.table.model.df.convert\_dtypes()  
  
 # Настройка ширины родительского окна таблицы  
 app.table.parentframe.configure(width=app.table.width)  
  
 # Обновление цветов строк таблицы  
 app.table.update\_rowcolors()  
  
 # Перерисовка таблицы  
 app.table.redraw()  
  
 # Изменение таблицы (обновление внутреннего состояния)  
 app.table.tableChanged()  
  
  
def statistical\_report(app\_local):  
 *"""  
 Генерирует статистический отчет на основе выбранного атрибута в таблице.  
  
 Параметры:  
 - app\_local (object): Объект приложения, содержащий таблицу.  
  
 Возвращает:  
 None  
 """* window = tk.Tk()  
 report\_window = None  
 frequency\_table = [None]  
 statistics\_table = [None]  
  
 def save\_report():  
 *"""  
 Сохраняет отчет в файл, выбранный пользователем.  
 """* nonlocal report\_window, frequency\_table, statistics\_table  
  
 # Запрашиваем у пользователя путь для сохранения файла  
 file\_path = fl.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "\*.txt")])  
  
 if file\_path:  
 # Сохраняем отчет в указанный файл  
 with open(file\_path, "w") as f:  
 if frequency\_table[0] is not None:  
 f.write(str(frequency\_table[0]))  
 elif statistics\_table[0] is not None:  
 f.write(str(statistics\_table[0]))  
  
 def handle\_selection(attribute):  
 *"""  
 Обрабатывает выбор атрибута и генерирует соответствующий отчет.  
  
 Параметры:  
 - attribute: Выбранный атрибут.  
 """* nonlocal report\_window, frequency\_table, statistics\_table  
  
 window.destroy() # Закрываем окно выбора атрибута  
  
 if not attribute:  
 mbx.showwarning("Ошибка", "Необходимо выбрать атрибут.", parent=app\_local.table.parentframe)  
 return  
  
 attribute\_data = app\_local.table.model.df[attribute]  
  
 if attribute\_data.dtype == 'object':  
 # Генерация отчета для качественного атрибута  
 frequency\_table[0] = attribute\_data.value\_counts().reset\_index()  
 frequency\_table[0].columns = ['Значение', 'Частота']  
 frequency\_table[0]['Процент'] = frequency\_table[0]['Частота'] / len(attribute\_data) \* 100  
  
 # Создаем окно отчета  
 report\_window = tk.Toplevel()  
 report\_window.title("Отчет")  
 report\_window.geometry("500x300")  
  
 # Создаем текстовое поле  
 report\_text = tk.Text(report\_window)  
 report\_text.pack(fill=tk.BOTH, expand=True)  
  
 # Добавляем содержимое отчета в текстовое поле  
 report\_text.insert(tk.END, str(frequency\_table[0]))  
  
 # Создаем кнопку для сохранения отчета  
 save\_button = tk.Button(report\_window, text="Сохранить отчет", command=save\_report)  
 save\_button.pack()  
  
 # Запрашиваем у пользователя выбор сохранить отчет или нет при закрытии окна  
 report\_window.protocol("WM\_DELETE\_WINDOW", handle\_close\_report)  
  
 elif pd.api.types.is\_numeric\_dtype(attribute\_data):  
 # Генерация отчета для количественного атрибута  
 statistics\_table[0] = pd.DataFrame({  
 'Переменная': [attribute],  
 'Максимум': [attribute\_data.max()],  
 'Минимум': [attribute\_data.min()],  
 'Среднее': [attribute\_data.mean()],  
 'Дисперсия': [attribute\_data.var()],  
 'Стандартное отклонение': [attribute\_data.std()]  
 })  
  
 # Создаем окно отчета  
 report\_window = tk.Toplevel()  
 report\_window.title("Отчет")  
 report\_window.geometry("750x300")  
  
 # Создаем текстовое поле  
 report\_text = tk.Text(report\_window)  
 report\_text.pack(fill=tk.BOTH, expand=True)  
  
 # Добавляем содержимое отчета в текстовое поле  
 report\_text.insert(tk.END, str(statistics\_table[0]))  
  
 # Создаем кнопку для сохранения отчета  
 save\_button = tk.Button(report\_window, text="Сохранить отчет", command=save\_report)  
 save\_button.pack()  
  
 # Запрашиваем у пользователя выбор сохранить отчет или нет при закрытии окна  
 report\_window.protocol("WM\_DELETE\_WINDOW", handle\_close\_report)  
  
 else:  
 mbx.showwarning("Ошибка", "Атрибут должен быть либо качественным, либо количественным.",  
 parent=app\_local.table.parentframe)  
 return  
  
 def handle\_close\_report():  
 *"""  
 Обрабатывает закрытие окна отчета и предлагает пользователю сохранить отчет.  
 """* nonlocal report\_window  
  
 close\_report = mbx.askyesno("Сохранить отчет", "Хотите сохранить отчет перед закрытием?", parent=report\_window)  
  
 if close\_report:  
 save\_report() # Сохраняем отчет  
 report\_window.destroy() # Закрываем окно отчета  
  
 # Создание окна выбора атрибута  
 window.title("Выберите атрибут")  
 window.geometry("300x100")  
  
 # Создание выпадающего списка  
 attribute\_var = tk.StringVar(window)  
 attribute\_var.set(app\_local.table.model.df.columns[0])  
 attribute\_optionmenu = tk.OptionMenu(window, attribute\_var, \*app\_local.table.model.df.columns)  
 attribute\_optionmenu.pack()  
  
 # Создание кнопки для подтверждения выбора  
 select\_button = tk.Button(window, text="Выбрать", command=lambda: handle\_selection(attribute\_var.get()))  
 select\_button.pack()  
  
 # Запуск главного цикла окна  
 window.mainloop()  
  
  
def generate\_report(app\_local):  
 *"""  
 Генерирует отчет на основе выбранных столбцов и отфильтрованных строк.  
  
 Параметры:  
 - app\_local: Объект приложения, содержащий таблицу.  
 """* # Создание окна выбора столбцов  
 column\_window = tk.Toplevel()  
 column\_window.title("Выберите атрибуты")  
 column\_window.geometry("300x200")  
  
 # Создание списка доступных столбцов  
 available\_columns = app\_local.table.model.df.columns  
  
 # Создание Listbox для выбора столбцов  
 selected\_columns = tk.StringVar(value=available\_columns)  
 columns\_listbox = tk.Listbox(column\_window, listvariable=selected\_columns, selectmode=tk.MULTIPLE)  
 columns\_listbox.pack()  
  
 def handle\_selection():  
 *"""  
 Обрабатывает выбор столбцов и фильтрацию строк.  
  
 Эта функция вызывается, когда пользователь подтверждает выбор столбцов.  
  
 Применяется операция проекции для выбранных столбцов и операция фильтрации для строк.  
 Затем создается окно отчета для отображения отфильтрованного DataFrame и предоставления опции сохранения отчета.  
  
 Если не указаны условия фильтрации строк, отображается предупреждающее сообщение.  
  
 Если окно отчета закрывается, пользователю предлагается сохранить отчет перед закрытием.  
 """* selected\_indices = columns\_listbox.curselection()  
 selected\_columns = [available\_columns[index] for index in selected\_indices]  
  
 # Закрытие окна выбора столбцов  
 column\_window.destroy()  
  
 # Запрос условия фильтрации строк  
 rows = sl.askstring("Строки", "Введите условие фильтрации строк:", parent=app\_local.table.parentframe)  
  
 if not rows:  
 mbx.showwarning("Ошибка", "Пожалуйста, введите условие фильтрации строк", parent=app\_local.table.parentframe)  
 return  
  
 # Применение операции проекции для выбранных столбцов  
 projected\_df = app\_local.table.model.df[selected\_columns]  
  
 # Применение операции фильтрации для строк  
 filtered\_df = projected\_df.query(rows)  
  
 # Создание окна отчета  
 report\_window = tk.Toplevel()  
 report\_window.title("Отчет")  
 report\_window.geometry("750x300")  
  
 # Создание текстового поля  
 report\_text = tk.Text(report\_window)  
 report\_text.pack(fill=tk.BOTH, expand=True)  
  
 # Добавление содержимого отчета в текстовое поле  
 report\_text.insert(tk.END, str(filtered\_df))  
  
 def save\_report():  
 *"""  
 Сохраняет отчет в выбранный пользователем файл.  
  
 Эта функция вызывается, когда пользователь нажимает кнопку "Сохранить отчет".  
  
 Пользователю предлагается выбрать путь файла для сохранения отчета в формате текстового файла.  
 Если указан путь файла, отчет сохраняется в указанный файл.  
 """* # Запрос пути файла для сохранения отчета  
 file\_path = fl.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "\*.txt")])  
  
 if file\_path:  
 # Сохранение отчета в указанный файл  
 with open(file\_path, "w") as f:  
 f.write(str(filtered\_df))  
  
 # Создание кнопки для сохранения отчета  
 save\_button = tk.Button(report\_window, text="Сохранить отчет", command=save\_report)  
 save\_button.pack()  
  
 # Предложение пользователю подтвердить сохранение отчета при закрытии окна  
 report\_window.protocol("WM\_DELETE\_WINDOW", lambda: handle\_close\_report(report\_window))  
  
 def handle\_close\_report(window):  
 *"""  
 Обрабатывает закрытие окна отчета.  
  
 Эта функция вызывается при закрытии окна отчета.  
  
 Пользователю предлагается подтвердить сохранение отчета перед закрытием окна.  
 Если пользователь выбирает сохранить отчет, вызывается функция save\_report для сохранения отчета.  
 Затем окно отчета закрывается.  
 """* close\_report = mbx.askyesno("Сохранить отчет", "Вы хотите сохранить отчет перед закрытием?",  
 parent=window)  
  
 if close\_report:  
 save\_report() # Сохранение отчета  
 window.destroy() # Закрытие окна отчета  
  
 # Запуск основного цикла окна отчета  
 report\_window.mainloop()  
  
 # Создание кнопки для подтверждения выбора столбцов  
 confirm\_button = tk.Button(column\_window, text="Создать", command=handle\_selection)  
 confirm\_button.pack()  
  
 # Запуск основного цикла окна выбора столбцов  
 column\_window.mainloop()  
  
  
class TestApp(tk.Frame):  
 *"""  
 TestApp - пользовательский класс на основе фрейма Tkinter, представляющий основное приложение.  
  
 Предоставляет графический интерфейс пользователя (GUI) для отображения и взаимодействия с таблицами.  
  
 Атрибуты:  
 - parent: Родительский виджет.  
  
 Методы:  
 - \_\_init\_\_(self, parent): Инициализирует класс TestApp.  
 """* def \_\_init\_\_(self, parent):  
 *"""  
 Инициализирует класс TestApp.  
  
 Параметры:  
 - parent: Родительский виджет.  
 """* super().\_\_init\_\_(parent)  
 self.table = Table(self, showstatusbar=True)  
 self.table.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app\_local = TestApp(root)  
 app\_local.pack()  
 root.mainloop()