ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ» МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ

Руководство разработчика по работе с приложением:

**«****Приложение для анализа данных потенциальных заемщиков»**

**Разработчики:**

Якушев Тимофей Павлович

Татаринова Полина Юрьевна

Пыжов Илья Игоревич

**Руководитель:**

Полякова М. В.

Москва 2023 г.

Оглавление

[Требования к характеристикам компьютера и операционной системе 3](#_Toc137408919)

[Версии интерпретатора и используемых библиотек 3](#_Toc137408920)

[Инструкция по запуску и настройке приложения 3](#_Toc137408921)

[Описание структуры БД 3](#_Toc137408922)

[Структура каталогов 4](#_Toc137408923)

[Архитектура приложения 4](#_Toc137408924)

[Листинг основного скрипта и всех модулей. 4](#_Toc137408925)

[main.py 4](#_Toc137408926)

[scripts.py 6](#_Toc137408927)

[libraries.py 40](#_Toc137408928)

# Требования к характеристикам компьютера и операционной системе

Характеристики ПК:

* процессор ×86 с тактовой частотой не менее 1 ГГц;
* оперативная память объемом, не менее 1 Гб;
* видеокарта, монитор, мышь, клавиатура;
* OC MS Windows 10, MacOS High Sierra

Наличие на компьютере интерпретатора «Python» (вне зависимости от среды разработки)

# Версии интерпретатора и используемых библиотек

**Интерпретатор** – Python 3.10

**Используемые библиотеки:**

|  |  |
| --- | --- |
| Библиотека | Версия |
| contourpy | 1.0.7 |
| cycler | 0.11.0 |
| et-xmlfile | 1.1.0 |
| fonttools | 4.38.0 |
| kiwisolver | 1.4.4 |
| matplotlib | 3.5.3 |
| numpy | 1.21.6 |
| openpy×l | 3.1.2 |
| packaging | 23.0 |
| pandas | 1.3.5 |
| Pillow | 9.4.0 |
| pyparsing | 3.0.9 |
| python-dateutil | 2.8.2 |
| pytz | 2022.7.1 |
| six | 1.16.0 |
| tzdata | 2023.3 |

# Инструкция по запуску и настройке приложения

Приложение запускается из командной строки: python <имя главного модуля>.py.

# Описание структуры БД

База данных имеет следующие поля:

CLIENTNUM – натуральное число  
Attrition\_Flag – строка  
Customer\_Age – натуральное число  
Gender – строка  
Dependent\_count – натуральное число  
Education\_Level – строка  
Marital\_Status – строка  
Income\_Category – строка  
Card\_Category – строка  
Months\_on\_book – натуральное число  
Total\_Relationship\_Count – натуральное число  
Months\_Inactive\_12\_mon – натуральное число  
Contacts\_Count\_12\_mon – натуральное число  
Credit\_Limit – натуральное число  
Total\_Revolving\_Bal – натуральное число   
Avg\_Open\_To\_Buy – натуральное число  
Total\_Amt\_Chng\_Q4\_Q1 – вещественное число  
Total\_Trans\_Amt – натуральное число  
Total\_Trans\_Ct – натуральное число   
Total\_Ct\_Chng\_Q4\_Q1 – вещественное число  
Avg\_Utilization\_Ratio – вещественное число

# Структура каталогов

Информационно–аналитическое приложение размещается в стандартной структуре каталогов:

**Work <-** основной каталог.

**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

|  |  |  |
| --- | --- | --- |
| Модуль | Местонахождение | Функция |
| main.py | Work/Scripts | Основной файл приложения |
| libraries.py | Work/Library | Библиотека универсальных функций |
| scripts.py | Work/Scripts | Специализированный модуль |

# Листинг основного скрипта и всех модулей.

## main.py

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
*"""  
Приложение для анализа данных кредитных историй заёмщиков  
"""*import sys  
import os  
import scripts  
import numpy as np  
  
  
ABS\_PATH = os.path.abspath('main.py')  
if '\\' in ABS\_PATH:  
 ABS\_PATH = '\\'.join(ABS\_PATH.split('\\')[:len(ABS\_PATH.split('\\')) - 2])  
else:  
 ABS\_PATH = '/'.join(ABS\_PATH.split('/')[:len(ABS\_PATH.split('/')) - 2])  
sys.path.append(ABS\_PATH)  
  
  
import Library.libraries  
  
  
if '\\' in ABS\_PATH:  
 FILE = ABS\_PATH + '\\Data\\Base\_edited.xlsx' # файл с исходной базой данных  
else:  
 FILE = ABS\_PATH + '/Data/Base\_edited.xlsx'  
data = Library.libraries.read\_from\_text\_file(FILE) # чтение файла  
data\_columns = data.columns # получение названий столбцов базы данных  
columns = []  
int\_columns = []  
string\_columns = []  
float\_columns = []  
for i, cols in enumerate(data\_columns):  
 columns.append(cols) # добавление названий столбцов в список  
 if isinstance(data.iloc[1, i], str): # если тип данных в столбце str  
 string\_columns.append(cols) # добавление названия столбца в список переменных  
 # типа str  
 elif isinstance(data.iloc[1, i], np.float64): # если тип данных в столбце float  
 float\_columns.append(cols) # добавление названия столбца в список переменных  
 # типа float  
 else: # если тип данных в столбце int  
 int\_columns.append(cols) # добавление названия столбца в список переменных типа int  
qualitative\_variables = ['Attrition\_Flag', 'Gender', 'Education\_Level',  
 'Marital\_Status', 'Income\_Category', 'Card\_Category'] # список  
# качественных переменных  
quantitative\_variables = ['Customer\_Age', 'Dependent\_count', 'Months\_on\_book',  
 'Total\_Relationship\_Count', 'Months\_Inactive\_12\_mon',  
 'Contacts\_Count\_12\_mon', 'Credit\_Limit',  
 'Total\_Revolving\_Bal', 'Avg\_Open\_To\_Buy',  
 'Total\_Amt\_Chng\_Q4\_Q1', 'Total\_Trans\_Amt',  
 'Total\_Trans\_Ct', 'Total\_Ct\_Chng\_Q4\_Q1',  
 'Avg\_Utilization\_Ratio'] # список количественных переменных  
scripts.interface(columns, data, qualitative\_variables, quantitative\_variables, string\_columns,  
 int\_columns, float\_columns) # вызов и запуск интерфейса

## scripts.py

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
*"""  
Специализированный модуль  
"""*import tkinter as tki  
from tkinter import ttk  
import configparser  
import sys  
import os  
import numpy as np  
import pandas as pd  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
from matplotlib.figure import Figure  
  
ABS\_PATH = os.path.abspath('main.py')  
if '\\' in ABS\_PATH:  
 ABS\_PATH = '\\'.join(ABS\_PATH.split('\\')[:len(ABS\_PATH.split('\\')) - 2])  
else:  
 ABS\_PATH = '/'.join(ABS\_PATH.split('/')[:len(ABS\_PATH.split('/')) - 2])  
sys.path.append(ABS\_PATH)  
  
import Library.libraries  
  
  
def adding\_entities(tree, data, columns, string\_columns, int\_columns, float\_columns):  
 *"""  
 Функция добавляет строку со значениями, введёнными пользователем, в базу данных  
 и таблицу treeview  
 Входные данные: таблица (ttk.Treeview)  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* def add\_click():  
 *"""  
 Функция срабатывает при нажатии на кнопку подтверждения выбора значений  
 для новой строки. Добавляет в базу данных новую строку с этими значениями,  
 добавляет эту строку в Treeview  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* entities = []  
 flag = 0  
 for i, cols in enumerate(columns):  
 if cols in string\_columns: # если столбец качественный  
 entities.append(combobox[i].get()) # в новую строку берется значение из выпадающего  
 # списка  
 elif cols in int\_columns: # если столбец количественный целочисленный  
 if Library.libraries.is\_numeric(spinbox[i].get()): # проверка на число  
 entities.append(np.int64(spinbox[i].get())) # в новую строку берется значение  
 # из поля ввода чисел  
 else:  
 flag = 1 # если было введено не число  
 else: # если столбец количественный вещественный  
 if Library.libraries.is\_numeric(spinbox[i].get()): # проверка на число  
 entities.append(np.float64(spinbox[i].get())) # в новую строку берется значение  
 # из поля ввода чисел  
 else:  
 flag = 1 # если было введено не число  
 add\_window.grab\_release()  
 add\_window.destroy() # закрытие окна, в котором вводились значения новой строки  
 if flag == 0: # ошибок нет  
 data.loc[len(data.index)] = entities # добавление новой строки  
 tree.insert("", tki.END, values=entities, iid=len(data)) # вывод строки в таблицу в  
 # главном окне  
 else: # если в поле ввода для чисел было введено не число, то появляется ошибка  
 tki.messagebox.showerror(title="Ошибка",  
 message="Была введена качественная переменная для"  
 " количественного столбца")  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 add\_window = tki.Toplevel() # создание нового окна  
 add\_window.title("Добавление строки") # название окна  
 add\_window.geometry(config['Adding\_menu']['x'] + 'x' + config['Adding\_menu']['y']) # размер  
 # окна  
 add\_window.resizable(False, False) # запрет на изменение размера окна  
 combobox = 21 \* [0]  
 spinbox = 21 \* [0]  
 for i in range(len(columns)):  
 add\_label = tki.Label(add\_window, text=columns[i]) # вывод названий столбцов  
 add\_label.grid(row=i, column=0)  
 for i, cols in enumerate(columns):  
 if cols in string\_columns: # если столбец качественный  
 characteristics = []  
 pd\_characteristics = pd.unique(data[cols]) # уникальные значения столбца  
 for chars in pd\_characteristics:  
 characteristics.append(chars) # добавление в список уникальных  
 # значений столбца  
 combobox[i] = ttk.Combobox(add\_window,  
 values=characteristics) # создание выпадающего списка с  
 # уникальными значениями  
 # столбца  
 combobox[i].current(0) # задаёт начальное значение в выпадающем списке  
 combobox[i].grid(row=i, column=1) # расположение выпадающего списка  
 if cols in int\_columns or cols in float\_columns: # если столбец количественный  
 characteristics = pd.unique(data[cols]) # уникальные значения столбца  
 spinbox[i] = tki.Spinbox(add\_window, from\_=min(characteristics),  
 to=max(characteristics)) # поле для ввода чисел  
 spinbox[i].grid(row=i, column=1) # расположение поля  
 add\_button = ttk.Button(add\_window, text="Добавить", command=add\_click) # кнопка, запускающая  
 # добавление строки  
 add\_button.grid(row=21, column=1, sticky="we") # расположение кнопки  
  
  
def deleting\_entities(tree, data\_local, columns):  
 *"""  
 Функция удаляет выбранную пользователем строку из базы данных и из таблицы  
 Treeview  
 Входные данные: таблица(ttk.Treeview)  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* def del\_select(event):  
 *"""  
 Функция срабатывает при нажатии на строку Treeview. Вызывает окно  
 подтверждения удаления выбранной строки  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* if not tree.selection(): # если не выбрана строка  
 return # выход из функции  
 check\_window = tki.Toplevel() # создание нового окна  
 check\_window.title("Подтверждение") # название окна  
 check\_window.geometry(config['Deleting\_menu']['x'] + 'x' + config['Deleting\_menu']['y']) #  
 # размеры окна  
 check\_window.resizable(False, False) # запрет на изменение размеров окна  
 label = ttk.Label(check\_window, text="Удалить выделенную строку?",  
 font=(config['Deleting\_menu']['font'],  
 int(config['Deleting\_menu']['font\_size'])),  
 foreground=config['Deleting\_menu']['foreground'])  
 label.pack(anchor="c")  
 button\_yes = ttk.Button(check\_window, text="Да", command=lambda: click\_yes(check\_window,  
 data\_local))  
 button\_yes.pack(side="left", padx=5) # кнопка согласия с удалением  
 button\_no = ttk.Button(check\_window, text="Нет", command=lambda: click\_no(check\_window))  
 button\_no.pack(side="right", padx=5) # кнопка несогласия с удалением  
  
 def click\_yes(check\_window, data\_local):  
 *"""  
 Функция срабатывает при нажатии на кнопку подтверждения удаления.  
 Удаляет выбранную пользователем строку из базы данных и из таблицы  
 Treeview  
 Входные данные: окно подтверждения(tki.Toplevel)  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* drop\_index = int(tree.selection()[0]) # индекс удаляемой строки  
 check\_window.grab\_release()  
 check\_window.destroy() # закрытие окна подтверждения  
 data\_local.drop(index=drop\_index, inplace=True) # удаление из базы данных выбранной строки  
 data\_local = data\_local.reset\_index(drop=True) # новая индексация базы данных с учетом  
 # удаления  
 tree.delete(\*tree.get\_children()) # удаление значений из таблицы в главном окне  
 for i in range(len(data\_local)):  
 values = []  
 for j in range(len(columns)):  
 values.append(data\_local.iloc[i, j])  
 tree.insert("", tki.END, values=values, iid=i) # вывод новой таблицы в главное окно с  
 # учётом удаления  
 tree.bind('<<TreeviewSelect>>', Library.libraries.plug) # при нажатии на строки таблицы не  
 # будет срабатывать функция удаления  
  
 def click\_no(check\_window):  
 *"""  
 Функция срабатывает при нажатии на кнопку отмены удаления. Закрывает  
 окно подтверждения  
 Входные данные: окно подтверждения(tki.Toplevel)  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* check\_window.grab\_release()  
 check\_window.destroy()  
 tree.bind('<<TreeviewSelect>>', Library.libraries.plug)  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 tki.messagebox.showinfo(title="Информация", message="Выберите строку для удаления")  
 tree.bind('<<TreeviewSelect>>', del\_select) # при нажатии на строку таблицы будет срабатывать  
 # функция  
  
  
def manual\_modification(tree, data, columns, string\_columns, int\_columns):  
 *"""  
 Функция осуществляет модификацию выбранной ячейки в базе данных и таблице  
 Treeview  
 Входные данные: таблица(ttk.Treeview)  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* def edit\_select(event):  
 *"""  
 Функция срабатывает при нажатии на строку Treeview. Вызывает окно  
 редактирования выбранной строки  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* if not tree.selection(): # если не выбрана строка  
 return # выход из функции  
 row\_num = int(tree.selection()[0]) # получение номера строки, в которой будет  
 # отредактирована ячейка  
 edit\_window = tki.Toplevel() # создание нового окна  
 edit\_window.title("Редактирование") # название окна  
 edit\_window.geometry(config['Modification\_menu']['x'] + 'x' +  
 config['Modification\_menu']['y']) # размер окна  
 edit\_window.resizable(False, False) # запрет на изменение размера окна  
 edit\_combobox = ttk.Combobox(edit\_window, values=columns) # добавление выпадающего списка  
 # из названий столбцов  
 edit\_combobox.current(0) # присваивание значения по умолчанию в выпадающем списке  
 edit\_combobox.grid(column=0, row=0, padx=10, pady=10) # размещение выпадающего списка  
 edit\_button1 = ttk.Button(edit\_window, text="Выбрать столбец",  
 command=lambda: edit\_click1(edit\_combobox.get(), edit\_window,  
 row\_num, edit\_button1))  
 edit\_button1.grid(column=0, row=1, padx=10, sticky="we") # кнопка для подтверждения выбора  
 # столбца  
  
 def edit\_click1(edit\_column, edit\_window, row\_num, edit\_button1):  
 *"""  
 Функция срабатывает при нажатии на кнопку подтверждения выбора столбца,  
 в котором нужно отредактировать ячейку. Вызывает поле для ввода нового  
 значения ячейки  
 Входные данные: выбранный столбец(str), окно редактирования(tki.Toplevel),  
 номер строки, в которой происходит редактирование(int), кнопка(ttk.Button)  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* edit\_button1.config(state='disabled')  
 if edit\_column in string\_columns: # если выбранный столбец качественный  
 characteristics = []  
 pd\_characteristics = pd.unique(data[edit\_column]) # уникальные значения из выбранного  
 # столбца  
 for chars in pd\_characteristics:  
 characteristics.append(chars) # добавление уникальных значений из  
 # столбца в список  
 value = ttk.Combobox(edit\_window, values=characteristics) # создание выпадающего списка  
 value.current(0) # значение по умолчанию в выпадающем списке  
 value.grid(column=1, row=0, padx=10, pady=10) # расположение выпадающего списка  
 else: # если выбранный столбец количестввенный  
 characteristics = pd.unique(data[edit\_column]) # уникальные значения из выбранного  
 # столбца  
 value = tki.Spinbox(edit\_window, from\_=min(characteristics),  
 to=max(characteristics)) # создание поля для ввода числа  
 value.grid(column=1, row=0, padx=10, pady=10) # расположение поля  
 edit\_button2 = ttk.Button(edit\_window, text="Редактировать",  
 command=lambda: edit\_click2(value.get(), edit\_column, row\_num,  
 edit\_window))  
 edit\_button2.grid(column=1, row=1, padx=10, sticky="we") # кнопка подтверждения  
 # редактирования  
  
 def edit\_click2(value, edit\_column, row\_num, edit\_window):  
 *"""  
 Функция срабатывает при нажатии на кнопку подтверждения редактирования  
 после введения нового значения ячейки. Осуществляет модификацию  
 выбранной ячейки в базе данных и таблице Treeview  
 Входные данные: новое значение ячейки(str), выбранный столбец(str),  
 номер строки, в которой происходит редактирование(int),окно  
 редактирования(tki.Toplevel)  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* flag = 0  
 if edit\_column in string\_columns: # если выбранный столбец качественный  
 new\_value = value  
 elif edit\_column in int\_columns: # если выбранный столбец количественный целочисленный  
 if Library.libraries.is\_numeric(value): # проверка на ввод числа  
 new\_value = np.int64(value)  
 else: # было введено не число  
 flag = 1  
 tki.messagebox.showerror(title="Ошибка",  
 message="Была введена качественная переменная для"  
 " количественного столбца")  
 else: # если выбранный столбец количественный вещественный  
 if Library.libraries.is\_numeric(value): # проверка на ввод числа  
 new\_value = np.float64(value)  
 else: # было введено не число  
 flag = 1  
 tki.messagebox.showerror(title="Ошибка",  
 message="Была введена качественная переменная для"  
 " количественного столбца")  
 edit\_window.grab\_release()  
 edit\_window.destroy() # закрытие окна  
 tree.bind('<<TreeviewSelect>>', Library.libraries.plug) # при нажатии на строки таблицы не  
 # будет срабатывать функция изменения  
 if flag == 0:  
 pd.options.mode.chained\_assignment = None # отключение предупреждения  
 data[edit\_column][row\_num] = new\_value # изменение значения ячейки  
 tree.delete(\*tree.get\_children()) # удаление таблицы из главного окна  
 for i in range(len(data)):  
 values = []  
 for j in range(len(columns)):  
 values.append(data.iloc[i, j])  
 tree.insert("", tki.END, values=values, iid=i) # добавление обновленной таблицы в  
 # главное окно  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 tki.messagebox.showinfo(title="Информация", message="Выберите строку для редактирования")  
 tree.bind('<<TreeviewSelect>>', edit\_select) # при нажатии на строку таблицы будет вызываться  
 # функция  
  
  
def data\_filter(data, columns, string\_columns, int\_columns, float\_columns):  
 *"""  
 Фильтрация базы данных по выбранным условиям, сохранение отфильтрованной базы данных  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* def checkbutton\_changed(i):  
 *"""  
 Функция добавляет поле для выбора условия фильтрации  
 Входные данные: номер столбца (int)  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* if checkbutton\_var[i].get() == 1: # если checkbutton отмечен галочкой  
 label3 = ttk.Label(window, text="Введите значения, по которым",  
 font=(config['Filter\_menu']['font'], int(config['Filter\_menu']  
 ['font\_size'])),  
 foreground=config['Filter\_menu']['foreground'])  
 label3.grid(row=0, column=1)  
 label4 = ttk.Label(window, text="нужно отсортировать данные",  
 font=(config['Filter\_menu']['font'], int(config['Filter\_menu']  
 ['font\_size'])),  
 foreground=config['Filter\_menu']['foreground'])  
 label4.grid(row=1, column=1)  
 if columns[i] in string\_columns: # если столбец качественный  
 characteristics = []  
 pd\_characteristics = pd.unique(data[columns[i]]) # уникальные значения столбца  
 for chars in pd\_characteristics:  
 characteristics.append(chars) # добавление уникальных значений  
 # столбца в список  
 combobox[i] = ttk.Combobox(window, values=characteristics) # выпадающий список из  
 # значений столбца  
 combobox[i].current(0) # значение по умолчанию в выпадающем списке  
 combobox[i].grid(row=i + 2, column=1) # расположение выпадающего списка  
 if columns[i] in int\_columns or columns[i] in float\_columns: # если столбец  
 # количественный  
 characteristics = pd.unique(data[columns[i]]) # уникальные значения столбца  
 spinbox[i] = tki.Spinbox(window, from\_=min(characteristics),  
 to=max(characteristics)) # поле для ввода числа  
 spinbox[i].grid(row=i + 2, column=1) # расположение поля для ввода числа  
  
 def click1(btn1):  
 *"""  
 Функция проверяет, заданы ли условия для фильтрации. Если да, то  
 выводит список столбцов для выбора тех, которые останутся в  
 отфильтрованной базе данных  
 Входные данные: кнопка(ttk.Button)  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* flag1 = 0  
 flag2 = 0  
 sel = True  
 for i in range(len(checkbutton)):  
 if checkbutton\_var[i].get() == 1: # если checkbutton отмечен галочкой  
 flag1 = 1  
 if columns[i] in string\_columns: # если столбец качественный  
 condition = combobox[i].get()  
 elif columns[i] in int\_columns: # если столбец количественный целочисленный  
 if Library.libraries.is\_numeric(spinbox[i].get()): # проверка на число  
 condition = np.int64(spinbox[i].get())  
 else: # если было введено не число  
 flag2 = 1  
 else: # если столбец количественный вещественный  
 if Library.libraries.is\_numeric(spinbox[i].get()): # проверка на число  
 condition = np.float64(spinbox[i].get())  
 else: # если было введено не число  
 flag2 = 1  
 if flag2 == 0:  
 sel = sel & (data[columns[i]] == condition) # проверка строк на соответствие  
 # введенным условиям  
 if flag1 == 0: # если не были выбраны checkbutton  
 tki.messagebox.showwarning(title="Предупреждение",  
 message="Не выбраны значения")  
 elif flag2 == 0:  
 btn1.config(state='disabled')  
 label1 = ttk.Label(window, text="Выберите столбцы,",  
 font=(config['Filter\_menu']['font'], int(config['Filter\_menu']  
 ['font\_size'])),  
 foreground=config['Filter\_menu']['foreground'])  
 label1.grid(row=0, column=2)  
 label2 = ttk.Label(window, text="которые нужно оставить",  
 font=(config['Filter\_menu']['font'], int(config['Filter\_menu']  
 ['font\_size'])),  
 foreground=config['Filter\_menu']['foreground'])  
 label2.grid(row=1, column=2)  
 # добавление checkbutton с названиями столбцов  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[0],  
 variable=checkbutton\_var1[0]))  
 checkbutton1[0].grid(row=2, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[1],  
 variable=checkbutton\_var1[1]))  
 checkbutton1[1].grid(row=3, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[2],  
 variable=checkbutton\_var1[2]))  
 checkbutton1[2].grid(row=4, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[3],  
 variable=checkbutton\_var1[3]))  
 checkbutton1[3].grid(row=5, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[4],  
 variable=checkbutton\_var1[4]))  
 checkbutton1[4].grid(row=6, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[5],  
 variable=checkbutton\_var1[5]))  
 checkbutton1[5].grid(row=7, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[6],  
 variable=checkbutton\_var1[6]))  
 checkbutton1[6].grid(row=8, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[7],  
 variable=checkbutton\_var1[7]))  
 checkbutton1[7].grid(row=9, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[8],  
 variable=checkbutton\_var1[8]))  
 checkbutton1[8].grid(row=10, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[9],  
 variable=checkbutton\_var1[9]))  
 checkbutton1[9].grid(row=11, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[10],  
 variable=checkbutton\_var1[10]))  
 checkbutton1[10].grid(row=12, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[11],  
 variable=checkbutton\_var1[11]))  
 checkbutton1[11].grid(row=13, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[12],  
 variable=checkbutton\_var1[12]))  
 checkbutton1[12].grid(row=14, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[13],  
 variable=checkbutton\_var1[13]))  
 checkbutton1[13].grid(row=15, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[14],  
 variable=checkbutton\_var1[14]))  
 checkbutton1[14].grid(row=16, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[15],  
 variable=checkbutton\_var1[15]))  
 checkbutton1[15].grid(row=17, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[16],  
 variable=checkbutton\_var1[16]))  
 checkbutton1[16].grid(row=18, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[17],  
 variable=checkbutton\_var1[17]))  
 checkbutton1[17].grid(row=19, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[18],  
 variable=checkbutton\_var1[18]))  
 checkbutton1[18].grid(row=20, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[19],  
 variable=checkbutton\_var1[19]))  
 checkbutton1[19].grid(row=21, column=2)  
 checkbutton\_var1.append(tki.IntVar())  
 checkbutton1.append(ttk.Checkbutton(window, text=columns[20],  
 variable=checkbutton\_var1[20]))  
 checkbutton1[20].grid(row=22, column=2)  
 btn2 = ttk.Button(window, text="Отфильтровать", command=lambda: click2(sel))  
 btn2.grid(row=24, column=1) # кнопка подтверждения фильтрации  
 elif flag2 == 1:  
 tki.messagebox.showerror(title="Ошибка",  
 message="Была введена качественная переменная для"  
 " количественного столбца")  
  
 def click2(sel):  
 *"""  
 Функция проверяет, заданы ли условия для фильтрации. Если да, то  
 выводит список столбцов для выбора тех, которые останутся в  
 отфильтрованной базе данных  
 Входные данные: условие фильтрации(pd.Series())  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* flag = 0  
 report\_columns = []  
 for i in range(len(checkbutton1)):  
 if checkbutton\_var1[i].get() == 1: # если столбец checkbutton выбран  
 flag = 1  
 report\_columns.append(  
 columns[i]) # добавление столбца в список столбцов, которые нужно оставить в  
 # отчёте  
 if flag == 1: # проверка на то, был ли выбран хотя бы один столбец  
 data\_filter\_local = data.loc[sel, report\_columns] # создание отфильтрованной базы  
 # данных  
 if data\_filter\_local.empty: # проверка на существование бзы данных с такими параметрами  
 tki.messagebox.showinfo(title="Информация",  
 message="Выбранным параметрам не соответствует ни одна"  
 " строка")  
 window.grab\_release()  
 window.destroy() # закрытие окна  
 else: # если такая база данных существует  
 window.grab\_release()  
 window.destroy() # закрытие окна  
 window1 = tki.Toplevel() # создание нового окна  
 window1.title("Отфильтрованная база данных") # название окна  
 window1.geometry(config['Filter\_menu']['a'] + 'x' + config['Filter\_menu']['b']) #  
 # размеры окна  
 menu1 = tki.Menu(window1) # создание меню сохранения  
 filter\_menu = tki.Menu(tearoff=0)  
 filter\_menu.add\_command(label="Excel файл",  
 command=lambda:  
 Library.libraries.save\_to\_excel(data\_filter\_local))  
 filter\_menu.add\_command(label="Бинарный файл", command=lambda:  
 Library.libraries.save\_to\_bin\_file(data\_filter\_local))  
 menu1.add\_cascade(label="Сохранить", menu=filter\_menu)  
 window1.config(menu=menu1)  
 tree1 = ttk.Treeview(window1, columns=report\_columns,  
 show="headings") # создание отфильтрованной таблицы  
 for cols in report\_columns:  
 tree1.heading(cols, text=cols)  
 for i in range(len(data\_filter\_local)):  
 values = []  
 for j in range(len(report\_columns)):  
 values.append(data\_filter\_local.iloc[i, j])  
 tree1.insert("", tki.END, values=values)  
 scrollbar11 = ttk.Scrollbar(window1, orient="horizontal", command=tree1.xview)  
 scrollbar11.pack(fill="x", side="bottom")  
 tree1["xscrollcommand"] = scrollbar11.set # добавление горизонтальной прокрутки  
 scrollbar21 = ttk.Scrollbar(window1, orient="vertical", command=tree1.yview, )  
 scrollbar21.pack(side="right", fill="y")  
 tree1["yscrollcommand"] = scrollbar21.set # добавление вертикальной прокрутки  
 tree1.pack(fill="both", expand=1)  
 else:  
 tki.messagebox.showwarning(title="Предупреждение", message="Не выбраны значения")  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 window = tki.Toplevel() # создание нового окна  
 window.title("Фильтр") # название окна  
 window.geometry(config['Filter\_menu']['x'] + 'x' + config['Filter\_menu']['y']) # размер окна  
 label1 = ttk.Label(window, text="Выберите столбцы, по которым",  
 font=(config['Filter\_menu']['font'], int(config['Filter\_menu']  
 ['font\_size'])),  
 foreground=config['Filter\_menu']['foreground'])  
 label1.grid(row=0, column=0)  
 label2 = ttk.Label(window, text="нужно отсортировать данные",  
 font=(config['Filter\_menu']['font'], int(config['Filter\_menu']  
 ['font\_size'])),  
 foreground=config['Filter\_menu']['foreground'])  
 label2.grid(row=1, column=0)  
 checkbutton\_var = [] # список состояний checkbutton  
 checkbutton = [] # список из checkbutton  
 checkbutton\_var1 = []  
 checkbutton1 = []  
 spinbox = 21 \* [0] # список из полей для ввода чисел  
 combobox = 21 \* [0] # список из выпадающих списков  
 # далее создание checkbutton, названия которых являются названиями столбцов  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[0], variable=checkbutton\_var[0], command=lambda:  
 checkbutton\_changed(0)))  
 checkbutton[0].grid(row=2, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[1], variable=checkbutton\_var[1], command=lambda:  
 checkbutton\_changed(1)))  
 checkbutton[1].grid(row=3, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[2], variable=checkbutton\_var[2], command=lambda:  
 checkbutton\_changed(2)))  
 checkbutton[2].grid(row=4, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[3], variable=checkbutton\_var[3], command=lambda:  
 checkbutton\_changed(3)))  
 checkbutton[3].grid(row=5, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[4], variable=checkbutton\_var[4], command=lambda:  
 checkbutton\_changed(4)))  
 checkbutton[4].grid(row=6, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[5], variable=checkbutton\_var[5], command=lambda:  
 checkbutton\_changed(5)))  
 checkbutton[5].grid(row=7, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[6], variable=checkbutton\_var[6], command=lambda:  
 checkbutton\_changed(6)))  
 checkbutton[6].grid(row=8, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[7], variable=checkbutton\_var[7], command=lambda:  
 checkbutton\_changed(7)))  
 checkbutton[7].grid(row=9, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[8], variable=checkbutton\_var[8], command=lambda:  
 checkbutton\_changed(8)))  
 checkbutton[8].grid(row=10, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(window, text=columns[9], variable=checkbutton\_var[9], command=lambda:  
 checkbutton\_changed(9)))  
 checkbutton[9].grid(row=11, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[10], variable=checkbutton\_var[10],  
 command=lambda: checkbutton\_changed(10)))  
 checkbutton[10].grid(row=12, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[11], variable=checkbutton\_var[11],  
 command=lambda: checkbutton\_changed(11)))  
 checkbutton[11].grid(row=13, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[12], variable=checkbutton\_var[12],  
 command=lambda: checkbutton\_changed(12)))  
 checkbutton[12].grid(row=14, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[13], variable=checkbutton\_var[13],  
 command=lambda: checkbutton\_changed(13)))  
 checkbutton[13].grid(row=15, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[14], variable=checkbutton\_var[14],  
 command=lambda: checkbutton\_changed(14)))  
 checkbutton[14].grid(row=16, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[15], variable=checkbutton\_var[15],  
 command=lambda: checkbutton\_changed(15)))  
 checkbutton[15].grid(row=17, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[16], variable=checkbutton\_var[16],  
 command=lambda: checkbutton\_changed(16)))  
 checkbutton[16].grid(row=18, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[17], variable=checkbutton\_var[17],  
 command=lambda: checkbutton\_changed(17)))  
 checkbutton[17].grid(row=19, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[18], variable=checkbutton\_var[18],  
 command=lambda: checkbutton\_changed(18)))  
 checkbutton[18].grid(row=20, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[19], variable=checkbutton\_var[19],  
 command=lambda: checkbutton\_changed(19)))  
 checkbutton[19].grid(row=21, column=0)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(ttk.Checkbutton(window, text=columns[20], variable=checkbutton\_var[20],  
 command=lambda: checkbutton\_changed(20)))  
 checkbutton[20].grid(row=22, column=0)  
 btn1 = ttk.Button(window,text="Закончить выбор",command=lambda:click1(btn1))  
 btn1.grid(row=23, column=1) # кнопка подтверждения выбора столбцов  
  
  
def statistic\_report(data, qualitative\_variables, quantitative\_variables):  
 *"""  
 Статистический отчёт по выбранным количественным или качественным переменным  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* def click\_1():  
 *"""  
 Вызывается при нажатии кнопки подтверждения выбора типа переменной,  
 даёт возможность пользователю выбрать столбцы для создания отчёта  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* button\_1.config(state='disabled') # запрет на нажатие кнопки  
 if combobox\_1.get() == "Качественная": # если выбран качественный тип переменных  
 label2 = ttk.Label(statistic\_window, text="Выберите столбец",  
 font=(config['Statistic\_menu']['font'], int(config['Statistic\_menu']  
 ['font\_size'])),  
 foreground=config['Statistic\_menu']['foreground'])  
 label2.grid(row=0, column=1)  
 combobox\_2 = ttk.Combobox(statistic\_window,  
 values=qualitative\_variables) # выпадающий список из  
 # качественных столбцов  
 combobox\_2.current(0) # значение по умолчанию в выпадающем списке  
 combobox\_2.grid(row=1, column=1) # расположение выпадающего списка  
 button\_2 = ttk.Button(statistic\_window, text="Закончить выбор",  
 command=lambda: click\_2(combobox\_2.get()))  
 button\_2.grid(row=2, column=1) # кнопка подтверждения выбора столбца  
 else: # если выбран количественный тип столбца  
 label2 = ttk.Label(statistic\_window, text="Выберите столбцы",  
 font=(config['Statistic\_menu']['font'],  
 int(config['Statistic\_menu']['font\_size'])),  
 foreground=config['Statistic\_menu']['foreground'])  
 label2.grid(row=0, column=1)  
 # создание checkbutton с названиями количественных столбцов  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[0],  
 variable=checkbutton\_var[0]))  
 checkbutton[0].grid(row=1, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[1],  
 variable=checkbutton\_var[1]))  
 checkbutton[1].grid(row=2, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[2],  
 variable=checkbutton\_var[2]))  
 checkbutton[2].grid(row=3, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[3],  
 variable=checkbutton\_var[3]))  
 checkbutton[3].grid(row=4, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[4],  
 variable=checkbutton\_var[4]))  
 checkbutton[4].grid(row=5, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[5],  
 variable=checkbutton\_var[5]))  
 checkbutton[5].grid(row=6, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[6],  
 variable=checkbutton\_var[6]))  
 checkbutton[6].grid(row=7, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[7],  
 variable=checkbutton\_var[7]))  
 checkbutton[7].grid(row=8, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[8],  
 variable=checkbutton\_var[8]))  
 checkbutton[8].grid(row=9, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[9],  
 variable=checkbutton\_var[9]))  
 checkbutton[9].grid(row=10, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[10],  
 variable=checkbutton\_var[10]))  
 checkbutton[10].grid(row=11, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[11],  
 variable=checkbutton\_var[11]))  
 checkbutton[11].grid(row=12, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[12],  
 variable=checkbutton\_var[12]))  
 checkbutton[12].grid(row=13, column=1)  
 checkbutton\_var.append(tki.IntVar())  
 checkbutton.append(  
 ttk.Checkbutton(statistic\_window, text=quantitative\_variables[13],  
 variable=checkbutton\_var[13]))  
 checkbutton[13].grid(row=14, column=1)  
 button\_3 = ttk.Button(statistic\_window, text="Закончить выбор", command=click\_3)  
 button\_3.grid(row=15, column=1) # кнопка заверешения выбора столбцов  
  
 def click\_2(var):  
 *"""  
 Вызывается, если пользователь создаёт отчёт по качественной переменной.  
 Формирует статистический отчёт, выводит его в отдельное окно с  
 возможностью сохранения  
 Входные данные: название переменной(str)  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* statistic\_window.grab\_release()  
 statistic\_window.destroy() # закрытие окна  
 # формирование статистической таблицы по качественной переменной  
 statistics = pd.crosstab(index=data[var], columns='Frequency')  
 parts = pd.crosstab(index=data[var], columns='Percentage')  
 parts = parts / parts.sum()  
 statistics = pd.concat([statistics, parts], axis=1)  
 statistics\_columns = statistics.columns # название столбцов таблицы  
 statistics\_rows = statistics.index.tolist() # индексы строк таблицы  
 col = []  
 col.append(var)  
 for cols in statistics\_columns:  
 col.append(cols) # название столбцов статистической таблицы  
 statistic\_window1 = tki.Toplevel() # создание нового окна  
 statistic\_window1.title("Статистический отчёт") # название окна  
 statistic\_window1.geometry(config['Statistic\_menu']['a'] + 'x' +  
 config['Statistic\_menu']['b']) # размеры окна  
 menu1 = tki.Menu(statistic\_window1) # создание меню сохранения  
 statistic\_menu = tki.Menu(tearoff=0)  
 statistic\_menu.add\_command(label="Excel файл",  
 command=lambda: Library.libraries.save\_to\_excel\_index(statistics))  
 statistic\_menu.add\_command(label="Бинарный файл",  
 command=lambda: Library.libraries.save\_to\_bin\_file(statistics))  
 menu1.add\_cascade(label="Сохранить", menu=statistic\_menu)  
 statistic\_window1.config(menu=menu1)  
 tree1 = ttk.Treeview(statistic\_window1, columns=col,  
 show="headings") # вывод статистической таблицы в окно приложения  
 for cols in col:  
 tree1.heading(cols, text=cols)  
 for i in range(len(statistics)):  
 values = []  
 values.append(statistics\_rows[i])  
 for j in range(len(col) - 1):  
 values.append(statistics.iloc[i, j])  
 tree1.insert("", tki.END, values=values)  
 tree1.pack(fill="both", expand=1)  
  
 def click\_3():  
 *"""  
 Вызывается, если пользователь создаёт отчёт по количественным переменным.  
 Формирует статистический отчёт, выводит его в отдельное окно с  
 возможностью сохранения  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* statistic\_window.grab\_release()  
 statistic\_window.destroy() # закрытие окна  
 var\_list = []  
 flag = 0  
 for i in range(len(checkbutton)):  
 if checkbutton\_var[i].get() == 1:  
 flag = 1  
 var\_list.append(quantitative\_variables[i]) # создание списка из выбранных  
 # количественных переменных  
 if flag == 0:  
 tki.messagebox.showwarning(title="Предупреждение", message="Не выбраны значения")  
 else: # если выбраны значения  
 # создание статистической таблицы по количественным переменным  
 statistics = pd.DataFrame({'': var\_list, 'max': [data[i].max() for i in var\_list],  
 'min': [data[i].min() for i in var\_list],  
 'mean': [data[i].mean() for i in var\_list],  
 'sample\_variance': [data[i].var() for i in var\_list],  
 'standard\_deviation': [data[i].std() for i in var\_list]})  
 statistics\_columns = statistics.columns # столбцы таблицы  
 col = []  
 for cols in statistics\_columns:  
 col.append(cols) # список из названий столбцов  
 statistic\_window1 = tki.Toplevel() # создание нового окна  
 statistic\_window1.title("Статистический отчёт") # название окна  
 statistic\_window1.geometry(  
 config['Statistic\_menu']['a'] + 'x' + config['Statistic\_menu']['b']) # размеры окна  
 menu1 = tki.Menu(statistic\_window1) # создание меню сохранения  
 statistic\_menu = tki.Menu(tearoff=0)  
 statistic\_menu.add\_command(label="Excel файл",  
 command=lambda: Library.libraries.save\_to\_excel(statistics))  
 statistic\_menu.add\_command(label="Бинарный файл",  
 command=lambda:  
 Library.libraries.save\_to\_bin\_file(statistics))  
 menu1.add\_cascade(label="Сохранить", menu=statistic\_menu)  
 statistic\_window1.config(menu=menu1)  
 tree1 = ttk.Treeview(statistic\_window1, columns=col, show="headings") # вывод  
 # статистической таблицы в окно приложения  
 for cols in col:  
 tree1.heading(cols, text=cols)  
 for i in range(len(statistics)):  
 values = []  
 for j in range(len(col)):  
 values.append(statistics.iloc[i, j])  
 tree1.insert("", tki.END, values=values)  
 tree1.pack(fill="both", expand=1)  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 statistic\_window = tki.Toplevel() # создание нового окна  
 statistic\_window.title("Статистический отчет") # название окна  
 statistic\_window.geometry(config['Statistic\_menu']['x'] + 'x' + config['Statistic\_menu']['y'])  
 # размеры окна  
 statistic\_window.resizable(False, False) # запрет изменения размера окна  
 label1 = ttk.Label(statistic\_window, text="Выберите тип переменных",  
 font=(config['Statistic\_menu']['font'],  
 int(config['Statistic\_menu']['font\_size'])),  
 foreground=config['Statistic\_menu']['foreground'])  
 label1.grid(row=0, column=0)  
 col = ["Количественные", "Качественная"]  
 combobox\_1 = ttk.Combobox(statistic\_window, values=col) # выпадающий список для выбора типа  
 # переменной  
 combobox\_1.current(0) # значение по умолчанию в выпадающем списке  
 combobox\_1.grid(row=1, column=0) # расположение выпадающего списка  
 button\_1 = ttk.Button(statistic\_window, text="Подтвердить", command=click\_1)  
 button\_1.grid(row=2, column=0) # кнопка подтверждения выбора типа столбца  
 checkbutton\_var = []  
 checkbutton = []  
  
  
def pivot\_table(data, qualitative\_variables, quantitative\_variables):  
 *"""  
 Создание сводной таблицы по паре выбранных качественных переменных  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* def click\_1():  
 *"""  
 Создание сводной таблицы по паре выбранных качественных переменных после  
 проверки на различие выбранных переменных, вывод таблицы в отдельное  
 окно с возможностью сохранения  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* if combobox\_5 == 'Cумма': # если выбран метод суммирование  
 method = 'count'  
 else: # если выбран метод средних  
 method = 'mean'  
 if combobox\_1.get() == combobox\_2.get() or combobox\_1.get() == combobox\_3.get() or \  
 combobox\_2.get() == \  
 combobox\_3.get():  
 # проверка на различность качественных значений  
 tki.messagebox.showwarning(title="Предупреждение",  
 message="Были выбраны одинаковые значения")  
 else:  
 # создание сводной таблицы  
 pivot\_table\_local = pd.pivot\_table(data, index=[combobox\_1.get(), combobox\_2.get()],  
 columns=combobox\_3.get(),  
 values=combobox\_4.get(), aggfunc=method)  
 pivot\_columns = pivot\_table\_local.columns  
 col = []  
 col.append(combobox\_1.get())  
 col.append(combobox\_2.get())  
 rows = pivot\_table\_local.index.tolist() # названия строк таблицы  
 for pivots in pivot\_columns:  
 col.append(pivots) # названия столбцов таблицы  
 pivot\_window1 = tki.Toplevel() # создание нового окна  
 pivot\_window1.title("Сводная таблица") # название окна  
 pivot\_window1.geometry(config['Pivot\_menu']['a'] + 'x' + config['Pivot\_menu']['b'])  
 # размеры окна  
 menu1 = tki.Menu(pivot\_window1) # создание меню сохранения  
 pivot\_menu = tki.Menu(tearoff=0)  
 pivot\_menu.add\_command(label="Excel файл",  
 command=lambda:  
 Library.libraries.save\_to\_excel\_index(pivot\_table\_local))  
 pivot\_menu.add\_command(label="Бинарный файл",  
 command=lambda:  
 Library.libraries.save\_to\_bin\_file(pivot\_table\_local))  
 menu1.add\_cascade(label="Сохранить", menu=pivot\_menu)  
 pivot\_window1.config(menu=menu1)  
 tree1 = ttk.Treeview(pivot\_window1, columns=col, show="headings")# вывод сводной таблицы  
 for cols in col:  
 tree1.heading(cols, text=cols)  
 for i in range(len(pivot\_table\_local)):  
 values = []  
 values.append(rows[i][0])  
 values.append(rows[i][1])  
 for j in range(len(col) - 2):  
 values.append(pivot\_table\_local.iloc[i, j])  
 tree1.insert("", tki.END, values=values)  
 # создание горизонтальной прокрутки  
 scrollbar1 = ttk.Scrollbar(pivot\_window1, orient="horizontal", command=tree1.xview)  
 scrollbar1.pack(fill="x", side="bottom")  
 tree1["xscrollcommand"] = scrollbar1.set  
 tree1.pack(fill="both", expand=1)  
 pivot\_window.grab\_release()  
 pivot\_window.destroy() # закрытие окна  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 pivot\_window = tki.Toplevel() # создание нового окна  
 pivot\_window.title("Сводная таблица") # название окна  
 pivot\_window.geometry(config['Pivot\_menu']['x'] + 'x' + config['Pivot\_menu']['y']) # размеры  
 # окна  
 pivot\_window.resizable(False, False) # запрет на изменение размера окна  
 methods = ['Сумма', 'Среднее'] # виды методов  
 label\_1 = ttk.Label(pivot\_window, text="Выберите качественные переменные",  
 font=(config['Pivot\_menu']['font'], int(config['Pivot\_menu']['font\_size'])),  
 foreground=config['Pivot\_menu']['foreground'])  
 label\_1.grid(row=0, column=1)  
 # создание выпадающих списков, в которых можно выбрать параметры сводной таблицы  
 combobox\_1 = ttk.Combobox(pivot\_window, values=qualitative\_variables)  
 combobox\_1.current(0)  
 combobox\_1.grid(row=1, column=0)  
 combobox\_2 = ttk.Combobox(pivot\_window, values=qualitative\_variables)  
 combobox\_2.current(1)  
 combobox\_2.grid(row=1, column=1)  
 combobox\_3 = ttk.Combobox(pivot\_window, values=qualitative\_variables)  
 combobox\_3.current(2)  
 combobox\_3.grid(row=1, column=2)  
 label\_2 = ttk.Label(pivot\_window, text="Выберите количественную переменную",  
 font=(config['Pivot\_menu']['font'], int(config['Pivot\_menu']['font\_size'])),  
 foreground=config['Pivot\_menu']['foreground'])  
 label\_2.grid(row=2, column=1)  
 combobox\_4 = ttk.Combobox(pivot\_window, values=quantitative\_variables)  
 combobox\_4.current(0)  
 combobox\_4.grid(row=3, column=1)  
 label\_3 = ttk.Label(pivot\_window, text="Выберите метод агрегации",  
 font=(config['Pivot\_menu']['font'], int(config['Pivot\_menu']['font\_size'])),  
 foreground=config['Pivot\_menu']['foreground'])  
 label\_3.grid(row=4, column=1)  
 combobox\_5 = ttk.Combobox(pivot\_window, values=methods)  
 combobox\_5.current(0)  
 combobox\_5.grid(row=5, column=1)  
 button\_1 = ttk.Button(pivot\_window, text="Создать таблицу", command=click\_1)  
 button\_1.grid(row=6, column=1) # кнопка подтверждения создания таблицы  
  
  
def clustered\_bar\_chart(data, qualitative\_variables):  
 *"""  
 Создание кластеризованной столбчатой диаграммы для пары 'качественная - качественная' переменных  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_1(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_2(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_3(event):  
 *"""  
 Создание и вывод графика на экран  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_4():  
 *"""  
 Сохранение графика в файл .png  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* Library.libraries.save\_graphics(fig) # сохранение в формате .png  
  
 fig = Figure(figsize=(15, 7), dpi=100) # создание экземпляра Figure для графиков  
 ax\_local = fig.add\_subplot(111) # создание поля для графика  
 x\_list = pd.unique(data[combobox\_1.get()]) # инициализация датафрейма из уровней  
 # первой переменной  
 y\_list = [sum(data[data[combobox\_2.get()] == combobox\_3.get()]  
 [combobox\_1.get()] == x) for x in x\_list] # создание списка частот  
 # для каждого уровня  
 color = list('rbgmcyk') # инициализация списка цветов для диаграммы  
 ax\_local.grid() # добавление сетки на график  
 ax\_local.bar(x\_list, y\_list, color=color) # построение диаграммы  
 canvas\_1 = FigureCanvasTkAgg(fig, master=window) # создание поля для двумерных  
 # изображений  
 canvas\_1.draw() # перерисовывание текущей фигуры  
 canvas\_1.get\_tk\_widget().pack(side=tki.TOP, fill=tki.NONE, expand=0) # расположение  
 # графика на виджете  
 menu1 = tki.Menu(window) # создание меню сохранения  
 menu1.add\_command(label="Сохранить", command=selected\_4)  
 window.config(menu=menu1)  
 window.after(200, None) # продолжение в фоновом режиме  
  
 label\_3 = ttk.Label(window, text='Выберите значение второй переменной',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_3.place(x=20, y=110)  
 selection = combobox\_2.get() # получение текущего выбранного значения из второго списка  
 a\_local = list(data[selection].unique()) # инициализация списка из уровней выбранной  
 # переменной  
 combobox\_3 = ttk.Combobox(window, values=a\_local, state='readonly') # создание третьего  
 # выпадающего списка  
 combobox\_3.place(x=20, y=130) # расположение выпадающего списка  
 combobox\_3.bind('<<ComboboxSelected>>', selected\_3) # отслеживание статуса списка  
  
 label\_2 = ttk.Label(window, text='Выберите 2-ю качественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_2.place(x=20, y=70)  
 drop = combobox\_1.get() # сохранение индекса использованной до этого переменной  
 combobox\_2 = ttk.Combobox(window, values=[x for x in qualitative\_variables if x != drop],  
 state='readonly') #  
 # создание второго выпадающего списка из оставшихся переменных  
 combobox\_2.place(x=20, y=90) # расположение выпадающего списка  
 combobox\_2.bind('<<ComboboxSelected>>', selected\_2) # отслеживание статуса списка  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 window = tki.Toplevel() # открывает новое диалоговое окно  
 window.title("Кластеризованная столбчатая диаграмма") # название окна  
 window.geometry(config['Graphic\_menu']['x'] + 'x' + config['Graphic\_menu']['y']) # размер окна  
 label\_1 = ttk.Label(window, text='Выберите 1-ю качественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_1.place(x=20, y=30)  
 combobox\_1 = ttk.Combobox(window, values=qualitative\_variables, state='readonly') # создание  
 # первого выпадающего  
 # списка  
 combobox\_1.place(x=20, y=50) # расположение выпадающего списка  
 combobox\_1.bind('<<ComboboxSelected>>', selected\_1) # отслеживание статуса списка  
  
  
def categorized\_bar\_chart(data, qualitative\_variables, quantitative\_variables):  
 *"""  
 Создание категоризированной гистограммы для пары 'количественная - качественная' переменных  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_1(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_2(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_3(event):  
 *"""  
 Создание и вывод графика на экран  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_4():  
 *"""  
 Сохранение графика в файл .png  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* Library.libraries.save\_graphics(fig) # сохранение в формате .png  
  
 fig = Figure(figsize=(15, 7), dpi=100) # создание экземпляра Figure для графиков  
 column\_size = len(data[data[combobox\_2.get()] == combobox\_3.get()]  
 [combobox\_1.get()]) # нахождение  
 # размера выборки  
 s\_dev = np.std(data[data[combobox\_2.get()] == combobox\_3.get()][combobox\_1.get()])  
 # нахождение  
 # среднего отклонения  
 iqr = np.subtract(\*np.percentile(data[data[combobox\_2.get()] == combobox\_3.get()]  
 [combobox\_1.get()],  
 [75, 25])) # нахождение межквартильного размаха  
 min\_max = max(data[data[combobox\_2.get()] == combobox\_3.get()]  
 [combobox\_1.get()]) - \  
 min(data[data[combobox\_2.get()] == combobox\_3.get()][combobox\_1.get()])  
 # нахождение размаха  
 # выборки  
 sturges = 1 + 3.322 \* np.log10(column\_size) # Sturges' formula  
 scott = min\_max \* np.power(column\_size, 1 / 3) / (3.5 \* s\_dev) # Scott's rule  
 freedman = min\_max \* np.power(column\_size, 1 / 3) / (2 \* iqr) # Freedman–Diaconis'  
 # choice  
 labels = ['Sturges', 'Scott', 'Freedman-Diaconis', 'Categories'] # инициализация  
 # списка названий для  
 # гистограмм  
 colors = ['#3e1ca8', '#ff3442', '#00e277', '#ffe4e1'] # инициализация списка цветов  
 # для гистограммы  
 n\_bins = list(map(round, [sturges, scott, freedman])) + [10] # список из количества  
 # интервалов на  
 # графиках  
 for i in range(4):  
 ax\_local = fig.add\_subplot(int('22' + str(i + 1))) # создание поля для графика  
 ax\_local.hist(data[data[combobox\_2.get()] == combobox\_3.get()]  
 [combobox\_1.get()],  
 bins=n\_bins[i],  
 color=colors[i]) # построение диаграммы  
 ax\_local.set\_title(labels[i]) # добавление названия  
 ax\_local.axvline(np.mean(data[data[combobox\_2.get()] == combobox\_3.get()]  
 [combobox\_1.get()]),  
 linestyle='dashed', color='black') # построение линии через среднее  
 # значение  
 canvas\_1 = FigureCanvasTkAgg(fig, master=window) # создание поля для двумерных  
 # изображений  
 canvas\_1.draw() # перерисовывание текущей фигуры  
 canvas\_1.get\_tk\_widget().pack(side=tki.TOP, fill=tki.NONE, expand=0) # расположение  
 # графиков на виджете  
 menu1 = tki.Menu(window) # создание меню сохранения  
 menu1.add\_command(label="Сохранить", command=selected\_4)  
 window.config(menu=menu1)  
 window.after(200, None) # продолжение в фоновом режиме  
  
 label\_3 = ttk.Label(window, text='Выберите значение качественной переменной',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_3.place(x=20, y=110)  
 selection = combobox\_2.get() # получение выбранного значения из второго списка  
 a\_local = list(data[selection].unique()) # инициализация списка из уровней выбранной  
 # переменной  
 combobox\_3 = ttk.Combobox(window, values=a\_local, state='readonly') # создание третьего  
 # выпадающего списка  
 combobox\_3.place(x=20, y=130) # расположение списка  
 combobox\_3.bind('<<ComboboxSelected>>', selected\_3) # отслеживание статуса списка  
  
 label\_2 = ttk.Label(window, text='Выберите качественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_2.place(x=20, y=70)  
 combobox\_2 = ttk.Combobox(window, values=qualitative\_variables, state='readonly')  
 # создание второго списка  
 combobox\_2.place(x=20, y=90) # расположение списка  
 combobox\_2.bind('<<ComboboxSelected>>', selected\_2) # отслеживание статуса списка  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 window = tki.Toplevel() # создание диалогового окна  
 window.title("Категоризированная гистограмма") # название окна  
 window.geometry(config['Graphic\_menu']['x'] + 'x' + config['Graphic\_menu']['y']) # размер окна  
 label\_1 = ttk.Label(window, text='Выберите количественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_1.place(x=20, y=30)  
 combobox\_1 = ttk.Combobox(window, values=quantitative\_variables, state='readonly') # создание  
 # первого выпадающего  
 # списка  
 combobox\_1.place(x=20, y=50) # расположение списка  
 combobox\_1.bind('<<ComboboxSelected>>', selected\_1) # отслеживание статуса списка  
  
  
def box\_and\_whiskers\_chart(data, qualitative\_variables, quantitative\_variables):  
 *"""  
 Создание категоризированной диаграммы Бокса-Вискера для пары 'количественная - качественная'  
 переменных  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* def selected\_1(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* def selected\_2(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* def selected\_3(event):  
 *"""  
 Создание и вывод графика на экран  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* def selected\_4():  
 *"""  
 Сохранение графика в файл .png  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* Library.libraries.save\_graphics(fig) # сохранение в формате .png  
  
 fig = Figure(figsize=(15, 7), dpi=100) # создание экземпляра Figure для графиков  
 ax\_local = fig.add\_subplot(111) # создание поля для графика  
 ax\_local.boxplot(data[data[combobox\_2.get()] == combobox\_3.get()][combobox\_1.get()],  
 vert=False) # построение  
 # диаграммы  
 canvas\_1 = FigureCanvasTkAgg(fig, master=window) # создание поля для двумерных  
 # изображений  
 canvas\_1.draw() # перерисовывание текущей фигуры  
 canvas\_1.get\_tk\_widget().pack(side=tki.TOP, fill=tki.NONE, expand=0) # расположение  
 # графика на виджете  
 menu1 = tki.Menu(window) # создание меню сохранения  
 menu1.add\_command(label="Сохранить", command=selected\_4)  
 window.config(menu=menu1)  
 window.after(200, None) # продолжение в фоновом режиме  
  
 label\_3 = ttk.Label(window, text='Выберите значение качественной переменной',  
 font=(config['Graphic\_menu']['font'], int(config['Graphic\_menu']  
 ['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_3.place(x=20, y=110)  
 selection = combobox\_2.get() # получение выбранного значения  
 a\_local = list(data[selection].unique()) # инициализация списка из уровней выбранной  
 # переменной  
 combobox\_3 = ttk.Combobox(window, values=a\_local, state='readonly') # создание  
 # выпадающего  
 # списка  
 combobox\_3.place(x=20, y=130) # расположение списка  
 combobox\_3.bind('<<ComboboxSelected>>', selected\_3) # отслеживание статуса списка  
  
 label\_2 = ttk.Label(window, text='Выберите качественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_2.place(x=20, y=70)  
 combobox\_2 = ttk.Combobox(window, values=qualitative\_variables, state='readonly')  
 # создание выпадающего списка  
 combobox\_2.place(x=20, y=90) # расположение списка  
 combobox\_2.bind('<<ComboboxSelected>>', selected\_2) # отслеживание списка  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 window = tki.Toplevel() # создание диалогового окна  
 window.title("Категоризированная диаграмма Бокса-Вискера") # название окна  
 window.geometry(config['Graphic\_menu']['x'] + 'x' + config['Graphic\_menu']['y']) # размер окна  
 label\_1 = ttk.Label(window, text='Выберите количественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_1.place(x=20, y=30)  
 combobox\_1 = ttk.Combobox(window, values=quantitative\_variables, state='readonly') # создание  
 # выпадающего списка  
 combobox\_1.place(x=20, y=50) # расположение списка  
 combobox\_1.bind('<<ComboboxSelected>>', selected\_1) # отслеживание статуса списка  
  
  
def scatter\_chart(data, qualitative\_variables, quantitative\_variables):  
 *"""  
 Создание категоризированной диаграммы рассеивания для пары количественных переменных и одной  
 качественной переменной  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_1(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_2(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_3(event):  
 *"""  
 Создание выпадающего списка  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_4(event):  
 *"""  
 Создание и вывод графика на экран  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_5():  
 *"""  
 Сохранение графика в файл .png  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* Library.libraries.save\_graphics(fig) # сохранение в формате .png  
  
 fig = Figure(figsize=(15, 7), dpi=100) # создание экземпляра Figure для  
 # графиков  
 ax\_local = fig.add\_subplot(111) # создание поля для графика  
 x\_list = data[data[combobox\_3.get()] == combobox\_4.get()][combobox\_1.get()]  
 # создание датафрейма  
 # для первой переменной  
 y\_list = data[data[combobox\_3.get()] == combobox\_4.get()][combobox\_2.get()]  
 # создание датафрейма  
 # для второй переменной  
 ax\_local.scatter(x\_list, y\_list, s=1) # создание графика  
 canvas\_1 = FigureCanvasTkAgg(fig, master=window) # создание поля для двумерных  
 # изображений  
 canvas\_1.draw() # перерисовывание текущей фигуры  
 canvas\_1.get\_tk\_widget().pack(side=tki.TOP, fill=tki.NONE, expand=0)  
 # расположение графика на  
 # виджете  
 menu1 = tki.Menu(window) # создание меню сохранения  
 menu1.add\_command(label="Сохранить", command=selected\_5)  
 window.config(menu=menu1)  
 window.after(200, None) # продолжение в фоновом режиме  
  
 label\_4 = ttk.Label(window, text='Выберите значение качественной переменной',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_4.place(x=20, y=150)  
 selection = combobox\_3.get() # получение выбранного значения  
 a\_local = list(data[selection].unique()) # инициализация списка из уровней  
 # выбранной  
 # переменной  
 combobox\_4 = ttk.Combobox(window, values=a\_local, state='readonly') # создание  
 # выпадающего списка  
 combobox\_4.place(x=20, y=170) # расположение списка  
 combobox\_4.bind('<<ComboboxSelected>>', selected\_4) # отслеживание статуса списка  
  
 label\_3 = ttk.Label(window, text='Выберите качественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_3.place(x=20, y=110)  
 combobox\_3 = ttk.Combobox(window, values=qualitative\_variables, state='readonly')  
 # создание выпадающего  
 # списка  
 combobox\_3.place(x=20, y=130) # расположение списка  
 combobox\_3.bind('<<ComboboxSelected>>', selected\_3) # отслеживание статуса списка  
  
 label\_2 = ttk.Label(window, text='Выберите 2-ю количественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_2.place(x=20, y=70)  
 drop = combobox\_1.get() # сохранения индекса выбранной переменной  
 combobox\_2 = ttk.Combobox(window, values=[x for x in quantitative\_variables if x != drop],  
 state='readonly') #  
 # создание выпадающего списка  
 combobox\_2.place(x=20, y=90) # расположение списка  
 combobox\_2.bind('<<ComboboxSelected>>', selected\_2) # отслеживание статуса списка  
  
 config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 window = tki.Toplevel() # создание диалогового окна  
 window.title("Категоризированная диаграмма рассеивания") # название окна  
 window.geometry(config['Graphic\_menu']['x'] + 'x' + config['Graphic\_menu']['y']) # размер окна  
 label\_1 = ttk.Label(window, text='Выберите 1-ю количественную переменную',  
 font=(config['Graphic\_menu']['font'],  
 int(config['Graphic\_menu']['font\_size'])),  
 foreground=config['Graphic\_menu']['foreground'])  
 label\_1.place(x=20, y=30)  
 combobox\_1 = ttk.Combobox(window, values=quantitative\_variables, state='readonly') # создание  
 # выпадающего списка  
 combobox\_1.place(x=20, y=50) # расположение списка  
 combobox\_1.bind('<<ComboboxSelected>>', selected\_1) # отслеживание статуса списка  
  
  
def settings\_editing():  
 *"""  
 Пользовательская настройка интерфейса  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
  
 def selected\_1(event):  
 *"""  
 Создание списка доступных для изменения настроек  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_2(event):  
 *"""  
 Создание списка доступных значений для изменения настройки  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* def selected\_font(event):  
 *"""  
 Изменение шрифта  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* config = configparser.RawConfigParser() # создание экземпляра RawConfigParser  
 config.optionxform = str # сохранение регистра файла  
 config.read(ABS\_PATH + '/Scripts/config.ini') # чтение текущих  
 # данных  
 config.set(dict\_1[combobox\_1.get()], 'font', combobox\_3.get()) # обновление  
 if '\\' in ABS\_PATH:  
 with open(ABS\_PATH + '\\Scripts\\config.ini', 'w', encoding='utf-8') as configfile:  
 config.write(configfile) # перезапись файлов  
 else:  
 with open(ABS\_PATH + '/Scripts/config.ini', 'w', encoding='utf-8') as configfile:  
 config.write(configfile) # перезапись файлов  
  
 def selected\_color(event):  
 *"""  
 Изменение цвета  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* config = configparser.RawConfigParser() # создание экземпляра RawConfigParser  
 config.optionxform = str # сохранение регистра файла  
 config.read(ABS\_PATH + '/Scripts/config.ini') # чтение текущих  
 # данных  
 config.set(dict\_1[combobox\_1.get()], 'foreground', dict\_3[combobox\_3.get()])  
 if '\\' in ABS\_PATH:  
 with open(ABS\_PATH + '\\Scripts\\config.ini', 'w', encoding='utf-8') as configfile:  
 config.write(configfile) # перезапись файлов  
 else:  
 with open(ABS\_PATH + '/Scripts/config.ini', 'w', encoding='utf-8') as configfile:  
 config.write(configfile) # перезапись файлов  
  
 def selected\_number():  
 *"""  
 Изменение размера окон  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Якушев Тимофей  
 """* config = configparser.RawConfigParser() # создание экземпляра RawConfigParser  
 config.optionxform = str # сохранение регистра файла  
 config.read(ABS\_PATH + '/Scripts/config.ini') # чтение текущих  
 # данных  
 config.set(dict\_1[combobox\_1.get()], dict\_2[combobox\_2.get()], str(spinbox.get()))  
 if '\\' in ABS\_PATH:  
 with open(ABS\_PATH + '\\Scripts\\config.ini', 'w', encoding='utf-8') as configfile:  
 config.write(configfile) # перезапись файлов  
 else:  
 with open(ABS\_PATH + '/Scripts/config.ini', 'w', encoding='utf-8') as configfile:  
 config.write(configfile) # перезапись файлов  
  
 selection\_2 = combobox\_2.get() # получение значения из выпадающего списка  
 if selection\_2 == 'Шрифт':  
 combobox\_3 = ttk.Combobox(window, values=['TkDefaultFont', 'TkTextFont',  
 'TkFixedFont', 'TkMenuFont',  
 'TkHeadingFont', 'TkCaptionFont',  
 'TkSmallCaptionFont',  
 'TkIconFont', 'TkTooltipFont'])  
 # создание списка  
 combobox\_3.place(x=250, y=100) # расположение списка  
 combobox\_3.bind('<<ComboboxSelected>>', selected\_font) # отслеживание статуса  
 # списка  
 elif selection\_2 == 'Цвет текста':  
 dict\_3 = {'Чёрный': '#000000', 'Красный': '#FF0000', 'Синий': '#0000FF', 'Зелёный':  
 '#008000',  
 'Жёлтый': '#FFFF00', 'Фиолетовый': '#8B00FF', 'Оранжевый': '#FFA500'}  
 combobox\_3 = ttk.Combobox(window, values=['Чёрный', 'Красный', 'Синий', 'Зелёный',  
 'Жёлтый',  
 'Фиолетовый', 'Оранжевый']) # создание  
 # списка  
 combobox\_3.place(x=250, y=100) # расположение списка  
 combobox\_3.bind('<<ComboboxSelected>>', selected\_color) # отслеживание статуса  
 # списка  
 elif selection\_2 == 'Размер шрифта':  
 spinbox = ttk.Spinbox(window, from\_=8, to=32, state='readonly', increment=1) # создание  
 # счётчика  
 spinbox.place(x=250, y=100) # расположение счётчика  
 btn = ttk.Button(window, text='Сохранить', command=selected\_number) # создание  
 # кнопки сохранения  
 btn.pack(anchor=tki.S) # расположение кнопки  
 else:  
 spinbox = ttk.Spinbox(window, from\_=100, to=900, state='readonly', increment=50) # создание  
 # счётчика  
 spinbox.place(x=250, y=100) # расположение счётчика  
 btn = ttk.Button(window, text='Сохранить', command=selected\_number) # создание  
 # кнопки сохранения  
 btn.pack(anchor=tki.S) # расположение кнопки  
  
 selection\_1 = combobox\_1.get() # получение значения из выпадающего списка  
 label\_2 = ttk.Label(window, text='Выберите настройку, который вы хотите изменить',  
 font=(config['Settings\_menu']['font'],  
 int(config['Settings\_menu']['font\_size'])),  
 foreground=config['Settings\_menu']['foreground']) # создание надписи  
 label\_2.place(x=20, y=70) # расположение надписи  
 dict\_2 = {'Длина окна': 'x', 'Ширина окна': 'y', 'Минимальная длина': 'x\_min',  
 'Минимальная ширина': 'y\_min',  
 'Размер шрифта': 'font\_size', 'Длина доп. окна': 'a', 'Ширина доп. окна': 'b'}  
 if selection\_1 == 'Основное меню':  
 a\_local = ['Длина окна', 'Ширина окна', 'Минимальная длина', 'Минимальная ширина']  
 elif selection\_1 == 'Меню модификации':  
 a\_local = ['Длина окна', 'Ширина окна']  
 elif selection\_1 in ('Статистический отчёт', 'Сводная таблица'):  
 a\_local = ['Длина окна', 'Ширина окна', 'Длина доп. окна', 'Ширина доп. окна', 'Шрифт',  
 'Размер шрифта',  
 'Цвет текста']  
 else:  
 a\_local = ['Длина окна', 'Ширина окна', 'Шрифт', 'Размер шрифта', 'Цвет текста']  
 combobox\_2 = ttk.Combobox(window, values=a\_local, state='readonly') # создание выпадающего  
 # списка  
 combobox\_2.place(x=20, y=100) # расположение списка  
 combobox\_2.bind('<<ComboboxSelected>>', selected\_2) # отслеживание статуса списка  
  
 window = tki.Toplevel() # создание диалогового окна  
 window.title('Настройки приложения') # название окна  
 window.geometry(config['Settings\_menu']['x'] + 'x' + config['Settings\_menu']['y']) # размер  
 # окна  
 label\_1 = ttk.Label(window, text='Выберите раздел настроек, который вы хотите изменить',  
 font=(config['Settings\_menu']['font'],  
 int(config['Settings\_menu']['font\_size'])),  
 foreground=config['Settings\_menu']['foreground']) # создание надписи  
 label\_1.place(x=20, y=30) # расположение надписи  
 dict\_1 = {'Основное меню': 'Main\_menu', 'Графические отчёты': 'Graphic\_menu', 'Меню добавления':  
 'Adding\_menu',  
 'Меню удаления': 'Deleting\_menu', 'Меню модификации': 'Modification\_menu', 'Фильтр':  
 'Filter\_menu',  
 'Статистический отчёт': 'Statistic\_menu', 'Сводная таблица': 'Pivot\_menu',  
 'Меню настроек': 'Settings\_menu'}  
 combobox\_1 = ttk.Combobox(window, values=['Основное меню', 'Графические отчёты',  
 'Меню добавления', 'Меню удаления',  
 'Меню модификации', 'Фильтр', 'Статистический отчёт',  
 'Сводная таблица',  
 'Меню настроек'], state='readonly') #  
 # создание выпадающего списка  
 combobox\_1.place(x=20, y=50) # расположение списка  
 combobox\_1.bind('<<ComboboxSelected>>', selected\_1) # отслеживание статуса списка  
  
  
def interface(columns, data, qualitative\_variables, quantitative\_variables, string\_columns,  
 int\_columns, float\_columns):  
 *"""  
 Интерфейс программы  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* config = configparser.ConfigParser() # создание экземпляра ConfigParser  
 if '\\' in ABS\_PATH:  
 config.read(  
 ABS\_PATH + '\\Scripts\\config.ini') # чтение конфигурационного файла  
 else:  
 config.read(  
 ABS\_PATH + '/Scripts/config.ini') # чтение конфигурационного файла  
 root = tki.Tk() # создание главного окна приложения  
 root.title('Приложение для анализа данных кредитных историй заёмщиков') # название окна  
 root.geometry(config['Main\_menu']['x'] + 'x' + config['Main\_menu']['y']) # размеры окна  
 root.minsize(int(config['Main\_menu']['x\_min']), int(config['Main\_menu']['y\_min']))  
 # минимальный размер окна  
 menu = tki.Menu(root) # создание меню  
 edit\_menu = tki.Menu(tearoff=0)  
 edit\_menu.add\_command(label="Редактировать ячейку", command=lambda: manual\_modification(tree,  
 data,  
 columns,  
 string\_columns,  
 int\_columns))  
 edit\_menu.add\_command(label="Удалить строку", command=lambda: deleting\_entities(tree, data,  
 columns))  
 edit\_menu.add\_command(label="Добавить строку", command=lambda: adding\_entities(tree, data,  
 columns,  
 string\_columns,  
 int\_columns,  
 float\_columns))  
 file\_menu = tki.Menu(tearoff=0)  
 save\_menu = tki.Menu(tearoff=0)  
 save\_menu.add\_command(label='Excel файл', command=lambda: Library.libraries.save\_to\_excel(data))  
 save\_menu.add\_command(label='Бинарный файл',  
 command=lambda: Library.libraries.save\_to\_bin\_file(data))  
 file\_menu.add\_cascade(label="Редактировать", menu=edit\_menu)  
 file\_menu.add\_cascade(label="Сохранить", menu=save\_menu)  
 report\_menu = tki.Menu(tearoff=0)  
 report\_menu.add\_command(label="Фильтр", command=lambda: data\_filter(data, columns,  
 string\_columns, int\_columns,  
 float\_columns))  
 report\_menu.add\_command(label="Статистический отчёт",  
 command=lambda: statistic\_report(data, qualitative\_variables,  
 quantitative\_variables))  
 report\_menu.add\_command(label="Сводная таблица",  
 command=lambda: pivot\_table(data, qualitative\_variables,  
 quantitative\_variables))  
 graphic\_menu = tki.Menu(tearoff=0)  
 graphic\_menu.add\_command(label='Кластеризованная столбчатая диаграмма',  
 command=lambda: clustered\_bar\_chart(data, qualitative\_variables))  
 graphic\_menu.add\_command(label='Категоризированная гистограмма',  
 command=lambda: categorized\_bar\_chart(data, qualitative\_variables,  
 quantitative\_variables))  
 graphic\_menu.add\_command(label='Категоризированная диаграмма Бокса-Вискера',  
 command=lambda: box\_and\_whiskers\_chart(data, qualitative\_variables,  
 quantitative\_variables))  
 graphic\_menu.add\_command(label='Категоризированная диаграмма рассеивания',  
 command=lambda: scatter\_chart(data, qualitative\_variables,  
 quantitative\_variables))  
 settings\_menu = tki.Menu(tearoff=0)  
 settings\_menu.add\_command(label='Настройки', command=settings\_editing)  
 menu.add\_cascade(label="Файл", menu=file\_menu)  
 menu.add\_cascade(label="Отчёт", menu=report\_menu)  
 menu.add\_cascade(label="Графические отчёты", menu=graphic\_menu)  
 menu.add\_cascade(label='Настройки', menu=settings\_menu)  
 root.config(menu=menu)  
 tree = ttk.Treeview(columns=columns, show="headings", height=500) # вывод базы данных в главное  
 # окно приложения  
 for i, cols in enumerate(columns):  
 tree.heading(cols, text=cols)  
 for i in range(len(data)):  
 values = []  
 for j in range(len(columns)):  
 values.append(data.iloc[i, j])  
 tree.insert("", tki.END, values=values, iid=i)  
 scrollbar1 = ttk.Scrollbar(orient="horizontal", command=tree.xview) # создание горизонтальной  
 # прокрутки  
 scrollbar1.pack(fill="x", side="bottom")  
 tree["xscrollcommand"] = scrollbar1.set  
 scrollbar2 = ttk.Scrollbar(orient="vertical", command=tree.yview) # создание вертикальной  
 # прокрутки  
 scrollbar2.pack(side="right", fill="y")  
 tree["yscrollcommand"] = scrollbar2.set  
 tree.pack()  
 root.mainloop()

## libraries.py

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
*"""  
Библиотека универсальных функций  
"""*from tkinter import filedialog as fld  
import pandas as pd  
import numpy as np  
  
  
def read\_from\_text\_file(file\_name):  
 *"""  
 Функция читает базу данных из файла формата .csv или .xlsx  
 Входные данные: имя файла (строка)  
 Выходные данные: датафрейм с базой данных (pd.DataFrame())  
 Автор: Якушев Тимофей  
 """* if '.csv' in file\_name: # если файл .csv  
 data\_local = pd.read\_csv(file\_name) # чтение из .csv  
 else: # если файл .xlsx или .xls  
 data\_local = pd.read\_excel(file\_name) # чтение из .xlsx или .xls  
 return data\_local # создание и возврат датафрейма  
  
  
def read\_from\_bin\_file(file\_name):  
 *"""  
 Функция читает базу данных из двоичного файла  
 Входные данные: имя файла  
 Выходные данные: базa данных (массив, кортеж, словарь и т. д.)  
 Автор: Якушев Тимофей  
 """* data\_local = np.load(file\_name) # загрузка базы данных из двоичного файла  
 return data\_local # возвращение базы данных  
  
  
def save\_to\_excel(data\_local):  
 *"""  
 Функция сохраняет базу данных в файл .xlsx  
 Входные данные: датафрейм с базой данных (pd.DataFrame())  
 Выходные данные: нет  
 Автор: Татаринова Полина   
 """* ftypes = [('Excel файлы', '\*.xlsx')] # в диалоговом окне могут отображаться только файлы .xlsx  
 dlg = fld.SaveAs(filetypes=ftypes) # вызов диалогового окна сохранения  
 path = dlg.show() + '.xlsx' # путь, выбранный пользователем  
 data\_local.to\_excel(path, index=False) # сохранение базы данных в формате .xlsx  
  
  
def save\_to\_excel\_index(data\_local):  
 *"""  
 Функция сохраняет базу данных в файл .xlsx, оставляя индекс  
 Входные данные: датафрейм с базой данных (pd.DataFrame())  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* ftypes = [('Excel файлы', '\*.xlsx')]  
 dlg = fld.SaveAs(filetypes=ftypes)  
 path = dlg.show() + '.xlsx'  
 data\_local.to\_excel(path) # сохранение базы данных в формате .xlsx без индекса  
  
  
def save\_to\_csv(file\_name, data\_local):  
 *"""  
 Функция сохраняет базу данных в файл .csv  
 Входные данные: имя файла (строка), датафрейм с базой данных (pd.DataFrame())  
 Выходные данные: нет  
 Автор: Якушев Тимофей   
 """* np.savetxt(file\_name, data\_local, fmt='%s', delimiter=';') # сохранение базы данных в формате  
 #.csv  
  
  
def save\_to\_bin\_file(data\_local):  
 *"""  
 Функция сохраняет базу данных в бинарный файл  
 Входные данные: датафрейм с базой данных (pd.DataFrame()), имя файла (строка)  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* dlg = fld.SaveAs()#вызов диалогового окна сохранения  
 path = dlg.show() #путь, выбранный пользователем  
 np.save(path, data\_local) # сохранение базы данных в бинарном файле  
  
  
def save\_graphics(figure):  
 *"""  
 Функция сохраняет построенный график в файл .png  
 Входные данные: имя файла (строка)  
 Выходные данные: нет  
 Автор: Пыжов Илья  
 """* ftypes = [('.png файлы', '\*.png')] # в диалоговом окне могут отображаться только файлы .png  
 dlg = fld.SaveAs(filetypes=ftypes) # вызов диалогового окна сохранения  
 path = dlg.show() + '.png' # путь, выбранный пользователем  
 figure.savefig(path) # сохранение графика в формате .png  
  
  
def is\_numeric(s\_local):  
 *"""  
 Функция для проверки было ли введено число  
 Входные данные: строка, которую нужно проверить(str)  
 Выходные данные: True или False  
 Автор: Пыжов Илья  
 """* try:  
 float(s\_local) # пробует конвертировать в float  
 return True # возвращает False  
 except ValueError: # если возникает ошибка  
 return False # возвращает False  
  
  
def plug(i):  
 *"""  
 Функция-"заглушка", чтобы при нажатии на строки treeview не срабатывали  
 другие функции  
 Входные данные: нет  
 Выходные данные: нет  
 Автор: Татаринова Полина  
 """* return i