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

Характеристики ПК:

- процессор ×86 с тактовой частотой не менее 1 ГГц;
- оперативная память объемом, не менее 1 Гб;
- видеокарта, монитор, мышь, клавиатура;
- ОС 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 количествввенный
        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 == 'Сумма':  # 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: база 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
```