ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

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

Бадажков Владислав, Галаев Рамазан, БИВ243, Бригада 11

**Руководство разработчика**

Проект по дисциплине «Проектный семинар ‘Python в науке о данных’»

студентов образовательной программы бакалавриата  
«Информатика и вычислительная техника»

Директор: Бадажков Владислав Денисович

Телефон: +79771325792

Адрес: vdbadazhkov@edu.hse.ru

Москва 2025 г.

Оглавление

[1. Требования к системе и компьютеру 3](#_Toc200717663)

[2. Версии языка и библиотек 3](#_Toc200717664)

[3. Описание архитектуры 3](#_Toc200717665)

[4. Структура каталогов 4](#_Toc200717666)

[5. Листинг основного скрипта и модулей 4](#_Toc200717667)

# 1. Требования к системе и компьютеру

Для работы приложения необходимы:

* **Операционная система**: Windows 10/11, macOS, Linux.
* **Минимальные характеристики**:

Процессор: 1 ГГц или выше.

Оперативная память: 4 ГБ.

Свободное место на диске: 200 МБ.

* **Установленное ПО**:

Python 3.8 или новее.

Библиотеки: tkinter, pandas, matplotlib, seaborn

# 2. Версии языка и библиотек

Язык: Python 3.11.4

matplotlib==3.7.1

pandas==1.5.3

seaborn==0.13.2

# 3. Описание архитектуры

Архитектура приложения:

1. Каталог scripts:

**main.py** – главный модуль, использующий функции domain.py и common.py. Отвечает за запуск основного окна приложения

**settings.ini** - настройка параметров конфигурации приложения

**domain.py** - модуль, содержащий специализированные функции (создание баз данных, текстовых и графических отчетов и т.д.)

1. Каталог library:

**common.py** - сохранение файлов разных форматов, заполнение виджета Treeview и т.д.

# 4. Структура каталогов

Информационно–аналитическое приложение размещается в следующей 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 размещаются руководства пользователя и разработчика.

Output — содержит копии текстовых отчетов.

Scripts — содержит специализированный модуль и файл с определением параметров настройки приложения.

# 5. Листинг основного скрипта и модулей

**main.py**

import tkinter as tk

import sys

import os

import pandas as pd

from pathlib import Path

sys.path.append(str(Path(\_\_file\_\_).parent.parent))

from scripts.domain import load\_config, type\_of\_pay, orders\_by\_hour, price\_boxplot, sum\_scatter, orders\_by\_hour\_text, orders\_by\_cashiers, pivot\_analyze, cashiers\_database, client\_types\_database, menu\_database, order\_database, orders\_database

from library.common import save\_csv, save\_excel, save\_png, save\_pickle, fill\_treeview, refresh\_tree

os.chdir("C:/work")

cashiers = pd.read\_pickle("C:/work/data/cashiers.pick")

client\_type = pd.read\_pickle("C:/work/data/client\_type.pick")

menu = pd.read\_pickle("C:/work/data/menu.pick")

order = pd.read\_pickle("C:/work/data/order.pick")

orders = pd.read\_pickle("C:/work/data/orders.pick")

def create\_menu(root):

    #Привязываем главное меню

    menu\_bar = tk.Menu(root)

    #Создаем подменю

    graphic\_menu = tk.Menu(menu\_bar, tearoff=0)

    text\_menu = tk.Menu(menu\_bar, tearoff=0)

    databases = tk.Menu(menu\_bar, tearoff=0)

    app\_info = tk.Menu(menu\_bar, tearoff=0)

    #Иниициализируем пункты меню графических отчетов

    graphic\_menu.add\_command(label="Тип оплаты по категориям клиентов", command=lambda: type\_of\_pay(root))

    graphic\_menu.add\_command(label="Распределение заказов по часам", command=lambda: orders\_by\_hour(root))

    graphic\_menu.add\_command(label="Разброс цен по категориям блюд", command=lambda: price\_boxplot(root))

    graphic\_menu.add\_command(label="Распределение суммы заказа по времени и типам клиентов", command=lambda: sum\_scatter(root))

    #Иниициализируем пункты меню текстовых отчетов

    text\_menu.add\_command(label="Количество заказов по часам", command=lambda: orders\_by\_hour\_text(root))

    text\_menu.add\_command(label="Количество заказов по кассирам", command=lambda: orders\_by\_cashiers(root))

    text\_menu.add\_command(label="Количество блюд разных категорий по часам", command=lambda: pivot\_analyze(root))

    #Иниициализируем пункты меню баз данных

    databases.add\_command(label="Кассиры", command=lambda: cashiers\_database(root))

    databases.add\_command(label="Типы клиентов", command=lambda: client\_types\_database(root))

    databases.add\_command(label="Меню", command=lambda: menu\_database(root))

    databases.add\_command(label="Состав заказов", command=lambda: order\_database(root))

    databases.add\_command(label="Заказы", command=lambda: orders\_database(root))

    #Добавляем подменю в главное меню с заголовком

    menu\_bar.add\_cascade(label="Графические отчеты", menu=graphic\_menu)

    menu\_bar.add\_cascade(label="Текстовые отчеты", menu=text\_menu)

    menu\_bar.add\_cascade(label="Базы данных", menu=databases)

    menu\_bar.add\_cascade(label="Справка", menu=app\_info)

    #Прикрепляем главное меню к корню

    root.config(menu=menu\_bar)

#создаем главное окно приложения

root = tk.Tk()

#Создаем главное меню

create\_menu(root)

#Окно работает пока пользователь его не закроет

root.mainloop()

**common.py**

import pandas as pd

import matplotlib.pyplot as plt

import tkinter as tk

#Сохранение файлов в форматах png, excel, csv

def save\_png(data, name):

    plt.savefig(f"C:/work/graphics/{name}.png")

def save\_excel(data, name):

    if isinstance(data, pd.DataFrame):

        data.to\_excel(f"C:/work/output/{name}.xlsx")

    else:

        df = pd.DataFrame({"Данные": [data]})

        df.to\_excel(f"C:/work/output/{name}.xlsx")

def save\_csv(data, name):

    if isinstance(data, pd.DataFrame):

        data.to\_csv(f"C:/work/output/{name}.csv", index=False, sep=";", encoding="utf-8-sig")

    else:

        df = pd.DataFrame({"Данные": [data]})

        df.to\_csv(f"C:/work/output/{name}.csv", index=False, sep=";", encoding="utf-8-sig")

#Заполнение виджета Treeview

def fill\_treeview(new\_window, data, tree):

    for col in data.columns:

        tree.heading(col, text=col)

    for idx, (\_, row) in enumerate(data.iterrows()):

        tree.insert("", "end", iid=str(idx), values=list(row))

    # Прокрутка к последней строке

    if tree.get\_children():

        tree.see(tree.get\_children()[-1])

    #Инициируем прокрутку

    scrollbar = tk.Scrollbar(new\_window, command=tree.yview)

    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

    tree.config(yscrollcommand=scrollbar.set)

#Обновление таблицы (виджет Treeview)

def refresh\_tree(tree, database):

    tree.delete(\*tree.get\_children())

    for idx, row in database.iterrows():

        tree.insert("", "end", iid=str(idx), values=list(row))

    # Показать последнюю строку

    if tree.get\_children():

        tree.see(tree.get\_children()[-1])

#Сохранение в файл формата pickle

def save\_pickle(data, name):

    data.to\_pickle(f"C:/work/data/{name}.pick")

**domain.py**

import tkinter as tk

from tkinter import ttk

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import os

import configparser

from library.common import fill\_treeview, refresh\_tree, save\_csv, save\_excel, save\_pickle, save\_png

#Загрузка параметров конфигурационного файла

def load\_config():

    config = configparser.ConfigParser()

    config.read("C:/Users/vlad/Desktop/ПС\_питон/HW3\_11/settings.ini")

    return {

        "bg\_color": config.get("UI", "bg\_color", fallback="#ffffff"),

        "font\_family": config.get("UI", "font\_family", fallback="Arial"),

        "font\_size": config.getint("UI", "font\_size", fallback=12)

    }

config = load\_config()

#Загрузка данных из баз

os.chdir("C:/work")

cashiers = pd.read\_pickle("C:/work/data/cashiers.pick")

client\_type = pd.read\_pickle("C:/work/data/client\_type.pick")

menu = pd.read\_pickle("C:/work/data/menu.pick")

order = pd.read\_pickle("C:/work/data/order.pick")

orders = pd.read\_pickle("C:/work/data/orders.pick")

#Обновление данных справочников

def reload\_data():

    global cashiers, client\_type, menu, order, orders

    cashiers = pd.read\_pickle("C:/work/data/cashiers.pick")

    client\_type = pd.read\_pickle("C:/work/data/client\_type.pick")

    menu = pd.read\_pickle("C:/work/data/menu.pick")

    order = pd.read\_pickle("C:/work/data/order.pick")

    orders = pd.read\_pickle("C:/work/data/orders.pick")

#Добавление новой строки в базу данных

def add\_database\_row(database, tree):

    next\_id = database[database.columns[0]].max() + 1 if not database.empty else 1

    types = list(map(type, database.iloc[0]))

    def types\_first(t):

        if t == pd.Timestamp:

            return pd.Timestamp.now()

        try:

            return t()

        except TypeError:

            return None

    if types[0] == pd.Timestamp:

        database.loc[len(database)] = [pd.Timestamp.now()] + [types\_first(t) for t in types[1:]]

    else:

        database.loc[len(database)] = [next\_id] + [types\_first(t) for t in types[1:]]

    refresh\_tree(tree, database)

#Удаление строки из базы данных

def delete\_database\_row(database, tree):

    selected\_item = tree.selection()[0]

    if not selected\_item:

        return

    database.drop(index=int(selected\_item), inplace=True)

    database.reset\_index(drop=True, inplace=True)

    refresh\_tree(tree, database)

#Редактирование данных справочника

def edit\_by\_click(event, tree, database, new\_window):

    #Защита окна от создания двойного виджета ввода

    if hasattr(tree, 'active\_entry') and tree.active\_entry is not None:

        return

    #Редактирование ячейки с защитой от реадктирования первой колонки

    selection = tree.selection()

    if not selection:

        return

    item = selection[0]

    col = tree.identify\_column(event.x)

    col\_idx = int(col[1:]) - 1

    if col\_idx == 0 and not database.equals(order):

        return

    # Получаем координаты ячейки относительно дерева

    x, y, width, height = tree.bbox(item, col)

    # Преобразуем координаты в абсолютные в пределах окна

    abs\_x = tree.winfo\_rootx() + x - new\_window.winfo\_rootx()

    abs\_y = tree.winfo\_rooty() + y - new\_window.winfo\_rooty()

    entry = ttk.Entry(new\_window)

    tree.active\_entry = entry

    entry.insert(0, tree.item(item, "values")[col\_idx])

    entry.place(x=abs\_x, y=abs\_y, width=width, height=height)

    #Обновление таблицы с учетом новых данных

    def save\_edit():

        row\_idx = int(item)

        try:

            value = type(database.iloc[row\_idx, col\_idx])(entry.get())

        except Exception:

            value = entry.get()

        database.iloc[row\_idx, col\_idx] = value

        refresh\_tree(tree, database)

        entry.destroy()

        tree.active\_entry = None

    entry.bind("<Return>", lambda e: save\_edit())

    entry.bind("<FocusOut>", lambda e: save\_edit())

#Создание окон редактирования баз данных (название\_database)

def cashiers\_database(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Кассиры", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    current\_df = cashiers

    #Создание таблицы

    tree = ttk.Treeview(new\_window, columns=list(current\_df.columns), show="headings")

    fill\_treeview(new\_window, current\_df, tree)

    tree.pack(expand=True, fill="both")

    tree.bind("<Double-1>", lambda e: edit\_by\_click(e, tree, current\_df, new\_window))

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(padx=110, pady=5, anchor="nw")

    #Создаем кнопку добавления

    button = tk.Button(frame,

                   text="Добавить",

                   command=lambda: add\_database\_row(current\_df, tree))

    button.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку удаления

    btn = tk.Button(frame, text="Удалить", command=lambda: delete\_database\_row(current\_df, tree))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в pickle

    btn = tk.Button(frame, text="Сохранить", command=lambda: save\_pickle(current\_df, "cashiers"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в Excel

    btn = tk.Button(frame, text="Экспорт в Excel", command=lambda: save\_excel(current\_df, "cashiers"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

def client\_types\_database(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Типы клиентов", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    current\_df = client\_type

    #Создание таблицы

    tree = ttk.Treeview(new\_window, columns=list(current\_df.columns), show="headings")

    fill\_treeview(new\_window, current\_df, tree)

    tree.pack(expand=True, fill="both")

    tree.bind("<Double-1>", lambda e: edit\_by\_click(e, tree, current\_df, new\_window))

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(padx=110, pady=5, anchor="nw")

    #Создаем кнопку добавления

    button = tk.Button(frame,

                   text="Добавить",

                   command=lambda: add\_database\_row(current\_df, tree))

    button.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку удаления

    btn = tk.Button(frame, text="Удалить", command=lambda: delete\_database\_row(current\_df, tree))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в pickle

    btn = tk.Button(frame, text="Сохранить", command=lambda: save\_pickle(current\_df, "client\_type"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в Excel

    btn = tk.Button(frame, text="Экспорт в Excel", command=lambda: save\_excel(current\_df, "client\_types"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

def menu\_database(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Меню", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    current\_df = menu

    #Создание таблицы

    tree = ttk.Treeview(new\_window, columns=list(current\_df.columns), show="headings")

    fill\_treeview(new\_window, current\_df, tree)

    tree.pack(expand=True, fill="both")

    tree.bind("<Double-1>", lambda e: edit\_by\_click(e, tree, current\_df, new\_window))

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(padx=110, pady=5, anchor="nw")

    #Создаем кнопку добавления

    button = tk.Button(frame,

                   text="Добавить",

                   command=lambda: add\_database\_row(current\_df, tree))

    button.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку удаления

    btn = tk.Button(frame, text="Удалить", command=lambda: delete\_database\_row(current\_df, tree))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в pickle

    btn = tk.Button(frame, text="Сохранить", command=lambda: save\_pickle(current\_df, "menu"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в Excel

    btn = tk.Button(frame, text="Экспорт в Excel", command=lambda: save\_excel(current\_df, "menu"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

def order\_database(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Состав заказов", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    current\_df = order

    #Создание таблицы

    tree = ttk.Treeview(new\_window, columns=list(current\_df.columns), show="headings")

    fill\_treeview(new\_window, current\_df, tree)

    tree.pack(expand=True, fill="both")

    tree.bind("<Double-1>", lambda e: edit\_by\_click(e, tree, current\_df, new\_window))

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(padx=110, pady=5, anchor="nw")

    #Создаем кнопку добавления

    button = tk.Button(frame,

                   text="Добавить",

                   command=lambda: add\_database\_row(current\_df, tree))

    button.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку удаления

    btn = tk.Button(frame, text="Удалить", command=lambda: delete\_database\_row(current\_df, tree))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в pickle

    btn = tk.Button(frame, text="Сохранить", command=lambda: save\_pickle(current\_df, "order"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в Excel

    btn = tk.Button(frame, text="Экспорт в Excel", command=lambda: save\_excel(current\_df, "order"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

def orders\_database(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Заказы", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    current\_df = orders

    #Создание таблицы

    tree = ttk.Treeview(new\_window, columns=list(current\_df.columns), show="headings")

    fill\_treeview(new\_window, current\_df, tree)

    tree.pack(expand=True, fill="both")

    tree.bind("<Double-1>", lambda e: edit\_by\_click(e, tree, current\_df, new\_window))

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(padx=110, pady=5, anchor="nw")

    #Создаем кнопку добавления

    button = tk.Button(frame,

                   text="Добавить",

                   command=lambda: add\_database\_row(current\_df, tree))

    button.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку удаления

    btn = tk.Button(frame, text="Удалить", command=lambda: delete\_database\_row(current\_df, tree))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в pickle

    btn = tk.Button(frame, text="Сохранить", command=lambda: save\_pickle(current\_df, "orders"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в Excel

    btn = tk.Button(frame, text="Экспорт в Excel", command=lambda: save\_excel(current\_df, "orders"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

def type\_of\_pay(root):

    #Прикручиваем график и кнопку

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Тип оплаты по категориям клиентов", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    #Заимствуем логику из прошлого дз

    fig, ax = plt.subplots(figsize=(12, 6))

    mg\_1 = orders.merge(client\_type, left\_on="тип клиента", right\_on="type\_id", how="inner")

    sns.countplot(data=mg\_1, x="тип", hue="тип оплаты ")

    plt.title("Типы оплаты по категориям клиентов")

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(anchor="n")

    #Создаем кнопку обновления

    button = tk.Button(frame,

                   text="Обновить",

                   command=lambda: upd\_type\_of\_pay(ax, canvas))

    button.pack(side="left", padx=5, pady=5, anchor="n")

    #Создаем кнопку сохранения в png

    btn = tk.Button(frame, text="Экспорт в png", command=lambda: save\_png(fig, "type\_of\_pay"))

    btn.pack(side="left", padx=5, pady=5, anchor="n")

    #Встраиваем график в ткинтер окно

    canvas = FigureCanvasTkAgg(fig, master=new\_window)

    canvas.draw()

    canvas.get\_tk\_widget().pack(pady=10)

    #ссылки

    new\_window.fig = fig

    new\_window.ax = ax

    new\_window.canvas = canvas

def orders\_by\_hour(root):

    #Также прикручиваем график и кнопку

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Распределение заказов по часам", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    #Заимствуем логику из прошлого дз

    fig, ax = plt.subplots(figsize=(12, 6))

    sns.histplot(data=orders, x="дата\_время", bins=20, kde=True, color="purple")

    plt.title("Распределение заказов по часам")

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(anchor="n")

    #Создаем кнопку обновления

    button = tk.Button(frame,

                   text="Обновить",

                   command=lambda: upd\_orders\_by\_hour(ax, canvas))

    button.pack(side="left", pady=5, padx=5)

    #Создаем кнопку сохранения в png

    btn = tk.Button(frame, text="Экспорт в png", command=lambda: save\_png(fig, "orders\_by\_hour"))

    btn.pack(side="left", padx=5, pady=5)

    #Встраиваем график в ткинтер окно

    canvas = FigureCanvasTkAgg(fig, master=new\_window)

    canvas.draw()

    canvas.get\_tk\_widget().pack(pady=10)

    #ссылки

    new\_window.fig = fig

    new\_window.ax = ax

    new\_window.canvas = canvas

def price\_boxplot(root):

    #Также прикручиваем график и кнопку

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Разброс цен по категориям блюд", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    #Заимствуем логику из прошлого дз

    fig, ax = plt.subplots(figsize=(16, 6))

    sns.boxplot(data=menu, y="цена", x="категория")

    plt.title("Разброс цен по категориям блюд")

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(anchor="n")

    #Создаем кнопку

    button = tk.Button(frame,

                   text="Обновить",

                   command=lambda: upd\_price\_boxplot(ax, canvas))

    button.pack(side="left", padx=5, pady=5)

    #Создаем кнопку сохранения в png

    btn = tk.Button(frame, text="Экспорт в png", command=lambda: save\_png(fig, "price\_boxplot"))

    btn.pack(side="left", padx=5, pady=5)

    #Встраиваем график в ткинтер окно

    canvas = FigureCanvasTkAgg(fig, master=new\_window)

    canvas.draw()

    canvas.get\_tk\_widget().pack(pady=10)

    #ссылки

    new\_window.fig = fig

    new\_window.ax = ax

    new\_window.canvas = canvas

def sum\_scatter(root):

    #Также прикручиваем график и кнопку

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Распределение суммы заказа по времени и типам клиентов", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    #Заимствуем логику из прошлого дз

    order\_sum = order.merge(menu, left\_on="позиция", right\_on="dish\_id", how="inner")

    order\_sum["сумма позиции"] = order\_sum["количество"] \* order\_sum["цена"]

    order\_totals = order\_sum.groupby("order\_id", as\_index=False)["сумма позиции"].sum()

    order\_totals = order\_totals.rename(columns={"сумма позиции": "сумма заказа"})

    orders\_loc = orders.drop(columns=["сумма заказа"], errors='ignore')

    orders\_loc = orders.merge(order\_totals, on="order\_id", how="left")

    fig, ax = plt.subplots(figsize=(12, 6))

    sns.scatterplot(data=orders\_loc, y="сумма заказа", x="дата\_время", hue="тип клиента", palette="viridis")

    plt.title("Распределение суммы заказа по времени и типам клиентов")

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(anchor="n")

    #Создаем кнопку

    button = tk.Button(frame,

                   text="Обновить",

                   command=lambda: upd\_sum\_scatter(ax, canvas))

    button.pack(side="left", pady=5, padx=5)

    #Создаем кнопку сохранения в png

    btn = tk.Button(frame, text="Экспорт в png", command=lambda: save\_png(fig, "scatterplot"))

    btn.pack(side="left", padx=5, pady=5)

    #Встраиваем график в ткинтер окно

    canvas = FigureCanvasTkAgg(fig, master=new\_window)

    canvas.draw()

    canvas.get\_tk\_widget().pack(pady=10)

    #ссылки

    new\_window.fig = fig

    new\_window.ax = ax

    new\_window.canvas = canvas

def orders\_by\_hour\_text(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Количество заказов по часам", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    label = tk.Label(new\_window, text="Введите час для подсчета количества заказов:", font=(config["font\_family"], config["font\_size"]), bg=config["bg\_color"])

    label.pack(pady=10)

    #Выбор часа

    spinbox = tk.Spinbox(new\_window, from\_=0.0, to=23.0)

    spinbox.pack(pady=10)

    #Считать

    btn = tk.Button(new\_window, text="Считать", command=lambda: count\_orders\_by\_hour(new\_window, spinbox))

    btn.pack(padx=6, pady=6)

def count\_orders\_by\_hour(new\_window, spinbox):

    #Подгрузка свежих данных

    reload\_data()

    #Удаление старых виджетов

    for widget in new\_window.winfo\_children()[4:]:

        widget.destroy()

    hour = int(spinbox.get())

    amount = len(orders[orders["дата\_время"].dt.hour == hour])

    label = tk.Label(new\_window, text="Отчет 1", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label = tk.Label(new\_window, text=f"Количество заказов за {hour}-й час: {amount}", font=(config["font\_family"], config["font\_size"]), bg=config["bg\_color"])

    label.pack(pady=10)

    data = f"Количество заказов за {hour}-й час: {amount}"

    #Сохранение в excel

    btn = tk.Button(new\_window, text="Экспорт в Excel", command=save\_excel(data, "orders\_by\_hour"))

    btn.pack(side="left", padx=10)

    #Сохранение в csv

    btn = tk.Button(new\_window, text="Экспорт в CSV", command=save\_csv(data, "orders\_by\_hour"))

    btn.pack(side="left", padx=10)

def orders\_by\_cashiers(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Количество клиентов, обслуженных каждым кассиром", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    label = tk.Label(new\_window, text="Введите ФИО кассира для подсчета количества заказов:", font=(config["font\_family"], config["font\_size"]), bg=config["bg\_color"])

    label.pack(pady=10)

    #Поле ввода

    entry = tk.Entry(new\_window)

    entry.pack(padx=6, pady=6)

    #Вывод статистики

    btn = tk.Button(new\_window, text="Считать", command=lambda: count\_orders\_by\_cashiers(entry, new\_window))

    btn.pack(padx=6, pady=6)

def count\_orders\_by\_cashiers(entry, new\_window):

    #Подгрузка свежих данных

    reload\_data()

    #Удаление старых виджетов

    for widget in new\_window.winfo\_children()[4:]:

        widget.destroy()

    #Формирование и заполнение Treeview

    cashier = entry.get()

    mg = orders.merge(cashiers, left\_on="кассир", right\_on="cashier\_id", how="left")

    mg\_new = mg.groupby("ФИО").count().reset\_index()

    mg\_new["процент заказов"] = round((mg\_new["кассир"] / 250) \* 100, 2)

    mg\_show = mg\_new[["ФИО", "кассир", "процент заказов"]]

    tree = ttk.Treeview(new\_window, columns=list(mg\_show.columns), show="headings")

    for col in mg\_show.columns:

        tree.heading(col, text=col)

    for \_, row in mg\_show.iterrows():

        tree.insert("", "end", values=list(row))

    tree.pack(expand=True, fill="both")

    #Сатитстика заказов по кассиру

    res = mg\_new[mg\_new["ФИО"] == cashier]

    if not res.empty:

        label = tk.Label(new\_window, text=f"количество заказов принятых кассиром: {res.iloc[0]['кассир']}", font=(config["font\_family"], config["font\_size"]), bg=config["bg\_color"])

    else:

        label = tk.Label(new\_window, text=f"кассир не найден", font=(config["font\_family"], config["font\_size"]), bg=config["bg\_color"])

    label.pack(pady=10)

    data = mg\_show

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(anchor="n", pady=20)

    #Сохранение в excel

    btn = tk.Button(frame, text="Экспорт в Excel", command=save\_excel(data, "orders\_by\_cahiers"))

    btn.pack(side="left", padx=10)

    #Сохранение в csv

    btn = tk.Button(frame, text="Экспорт в CSV", command=save\_csv(data, "orders\_by\_cahiers"))

    btn.pack(side="left", padx=10)

def pivot\_analyze(root):

    new\_window = tk.Toplevel(root)

    new\_window.configure(bg=config["bg\_color"])

    #Создаем заголовок

    label = tk.Label(new\_window, text="Количество блюд разных категорий по часам", font=(config["font\_family"], config["font\_size"], "bold"), bg=config["bg\_color"])

    label.pack(pady=10)

    btn = tk.Button(new\_window, text="Обновить", command=lambda: upd\_pivot\_analyze(new\_window))

    btn.pack(padx=6, pady=6)

def upd\_pivot\_analyze(new\_window):

    #Подгрузка свежих данных

    reload\_data()

    #Создание Treeview

    orders['час заказа'] = orders['дата\_время'].dt.hour

    order\_sum = order.merge(menu, left\_on="позиция", right\_on="dish\_id", how="inner")

    order\_sum = order\_sum.merge(orders[['order\_id', 'час заказа']], on='order\_id', how='left')

    pivot = pd.pivot\_table(order\_sum, index='час заказа', columns='категория', values='позиция', aggfunc='count', fill\_value=0)

    pivot = pivot.reset\_index()

    tree = ttk.Treeview(new\_window, columns=list(pivot.columns), show="headings")

    for col in pivot.columns:

        tree.heading(col, text=col)

    for \_, row in pivot.iterrows():

        tree.insert("", "end", values=list(row))

    tree.pack(expand=True, fill="both")

    data = pivot

    #Создаем группу кнопок

    frame = tk.Frame(new\_window, bg=config["bg\_color"])

    frame.pack(anchor="n", pady=20)

    #Сохранение в excel

    btn = tk.Button(frame, text="Экспорт в Excel", command=save\_excel(data, "pivot"))

    btn.pack(side="left", padx=10)

    #Сохранение в csv

    btn = tk.Button(frame, text="Экспорт в CSV", command=save\_csv(data, "pivot"))

    btn.pack(side="left", padx=10)

def upd\_price\_boxplot(ax, canvas):

    #Подгрузка свежих данных

    reload\_data()

    # Очищаем график

    ax.clear()

    # Рисуем новые данные

    sns.boxplot(data=menu, y="цена", x="категория", ax=ax)

    ax.set\_title("Разброс цен по категориям блюд")

    # Перерисовываем

    canvas.draw()

def upd\_sum\_scatter(ax, canvas):

    #Подгрузка свежих данных

    reload\_data()

    # Обновляем данные

    order\_sum = order.merge(menu, left\_on="позиция", right\_on="dish\_id", how="inner")

    order\_sum["сумма позиции"] = order\_sum["количество"] \* order\_sum["цена"]

    order\_totals = order\_sum.groupby("order\_id", as\_index=False)["сумма позиции"].sum()

    order\_totals = order\_totals.rename(columns={"сумма позиции": "сумма заказа"})

    orders\_loc = orders.drop(columns=["сумма заказа"], errors='ignore')

    orders\_loc = orders.merge(order\_totals, on="order\_id", how="left")

    # Очищаем график

    ax.clear()

    # Рисуем новые данные

    sns.scatterplot(data=orders\_loc, y="сумма заказа", x="дата\_время", hue="тип клиента", palette="viridis")

    ax.set\_title("Распределение суммы заказа по времени и типам клиентов")

    # Перерисовываем

    canvas.draw()

def upd\_type\_of\_pay(ax, canvas):

    #Подгрузка свежих данных

    reload\_data()

    # Обновляем данные

    mg\_1 = orders.merge(client\_type, left\_on="тип клиента", right\_on="type\_id", how="inner")

    # Очищаем график

    ax.clear()

    # Рисуем новые данные

    sns.countplot(data=mg\_1, x="тип", hue="тип оплаты ", ax=ax)

    ax.set\_title("Типы оплаты по категориям клиентов")

    # Перерисовываем

    canvas.draw()

def upd\_orders\_by\_hour(ax, canvas):

    #Подгрузка свежих данных

    reload\_data()

    # Очищаем график

    ax.clear()

    # Рисуем новые данные

    sns.histplot(data=orders, x="дата\_время", bins=20, kde=True, color="purple", ax=ax)

    ax.set\_title("Распределение заказов по часам")

    # Перерисовываем

    canvas.draw()