**Федеральное государственное автономное образовательное учреждение высшего образования   
"Национальный исследовательский университет   
"Высшая школа экономики"**

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

**Департамент компьютерной инженерии**

**Курс: Python в науке о данных**

**Руководство пользователя  
для итогового проекта по курсу**

**БИВ234 Бригада 2**

**Состав бригады:  
Бабенко Андрей   
(директор, +79257336990,** [**anvibabenko@edu.hse.ru**](mailto:anvibabenko@edu.hse.ru)**)**

**Осинцев Кирилл**

**Свистунов Андрей**

**Сорокин Глеб**

**Москва 2024**

**Оглавление**

[**1.** Описание приложения 3](#_Toc162697180)

[2. Описание структуры каталогов 4](#_Toc162697181)

[3. системные требования 5](#_Toc162697182)

[**4.** Описание архитектуры приложения 6](#_Toc162697183)

[5. Листинг 7](#_Toc162697184)

**Описание приложения**

Данное приложения создано для анализа футбольных матчей по всему миру. Основная задача приложения – дать возможность пользователю смотреть статистику футбольных матчей и добавлять данные про новые матчи и клубы, которых ещё нет в приложении.

**Описание структуры каталогов**

Все каталоги приложения хранятся в общем каталоге work, которая имеет такое дерево:  
**data** – содержит базу данных и фотографию на главном экране приложения

**notes** – содержит всю документацию, включая руководства разработчика и пользователя

**scripts** – содержит все файлы кода для приложения

**library –** содержит модуль library, в котором расположены 3 функции load\_data, save\_data и open\_file. Эти файлы могут быть использованы и в других проектах

**Системные требования**

Приложение разработано для операционных систем Windows (10 версия и выше), Linux, macOS.

Минимальные требования компьютера:

Оперативная память: 2Гб

Место на диске: 500 Мб свободного места

Процессор: 2 или более ядра частотой 1ГГц

Так как приложение не требует больших вычислительных мощностей, то оно не должно создавать неудобств во время эксплуатации на современных компьютерах.

**Описание архитектуры приложения**

Архитектура нашего приложения разработана следующим образом:

1. Пользовательский интерфейс (UI):

UI реализуется с помощью стандартной библиотеки для языка программирования Python – tkinter. Эта библиотека нужна для создания самого приложения, окон, кнопок, полей и других элементов интерфейса для конечного пользователя.

1. Данные:

Исходные данные про футбольные матчи хранятся в каталоге data в excel файле. С помощью такой библиотеки, как pandas разработчику удаётся работать с любыми данными .csv и .xlsx файлами.

1. Логика программы:

Нажатие пользователем на кнопки в приложении вызывает определённые функции, которые выполняют разные функции – от создания графиков с помощью библиотек matplotlib и seaborn до создания новых окон и открытия excel файла.

Такая архитектура приложения крайне понятна для человека, что позволяет легко исправить, заменить или добавить новые функции в приложение.

**Листинг**

main.py:

# -\*- coding: utf-8 -\*-  
*"""  
Spyder Editor  
  
This is a temporary script file.  
"""*# -\*- coding: utf-8 -\*-  
"""  
Создано 15 мая 2024 года  
  
Авторы: Бабенко А, Осинцев К  
"""  
  
import tkinter as tk  
from tkinter import messagebox, ttk  
import os  
import sys  
from PIL import Image, ImageTk  
from library import load\_data # Модуль для загрузки, сохранения и открытия файлов  
import filter # Модуль для фильтрации данных  
import sheet\_report # Модуль для создания текстовых отчетов  
import club\_operations # Импорт нового модуля  
from graph\_display import open\_view\_graphs, show\_graph  
sys.path.append("../../work")  
  
  
def open\_reports():  
 *"""  
 Открывает файл отчетов 'reports.xlsx', если он существует.  
  
 Исключения:  
 FileNotFoundError: Если файл отчетов не найден.  
 Exception: Если произошла ошибка при открытии файла.  
 """* try:  
 os.startfile("reports.xlsx")  
 except FileNotFoundError:  
 messagebox.showerror("Ошибка", "Файл отчетов не найден.")  
 except Exception as e:  
 messagebox.showerror(  
 "Ошибка", f"Произошла ошибка при открытии файла отчетов: {e}")  
  
  
def generate\_reports():  
 *"""  
 Генерирует отчеты, вызывая основную функцию из модуля sheet\_report,  
 и отображает сообщение об успешном создании.  
 """* sheet\_report.main()  
 messagebox.showinfo("Создать отчеты", "Отчеты успешно созданы.")  
  
  
class *Application*(tk.Tk):  
 *"""  
 Основной класс приложения для анализа данных.  
  
 Этот класс инициализирует главное окно, загружает данные и создает виджеты.  
  
 Атрибуты:  
 img (ImageTk.PhotoImage): Фоновое изображение для фрейма отображения.  
 display\_frame (tk.Frame): Фрейм, в котором отображается основной контент.  
 canvas (tk.Canvas): Канва для отображения фонового изображения.  
 clubs\_df (pd.DataFrame): DataFrame с данными о клубах.  
 matches\_df (pd.DataFrame): DataFrame с данными о матчах.  
 managers\_df (pd.DataFrame): DataFrame с данными о менеджерах.  
 data\_file\_path (str): Путь к файлу данных.  
 report\_file\_path (str): Путь к файлу отчетов.  
 picture1\_file\_path (str): Путь к фоновому изображению.  
 button\_width (int): Ширина кнопок.  
 button\_height (int): Высота кнопок.  
 """* def \_\_init\_\_(self):  
 *"""  
 Инициализирует приложение, загружает данные и настраивает главное окно и виджеты.  
 """* super().\_\_init\_\_()  
 self.img = None  
 self.display\_frame = None  
 self.canvas = None  
 self.entries = None  
 self.entry\_club\_position = None  
 self.entry\_manager\_name = None  
 self.entry\_club\_name = None  
 self.entry\_club\_id = None  
 self.entry\_club\_formation = None  
 self.open\_reports\_button = None  
 self.generate\_reports\_button = None  
 self.view\_excel\_button = None  
 self.view\_graphs\_button = None  
 self.view\_clubs\_button = None  
 self.add\_club\_button = None  
 self.title("Data Analyze")  
 self.geometry("1200x650")  
 self.configure(bg="white")  
  
 # Определение путей к файлам данных и отчетов относительно расположения скрипта  
 base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 self.data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 self.report\_file\_path = os.path.join(base\_dir, "..", "reports.xlsx")  
 self.picture1\_file\_path = os.path.join(  
 base\_dir, "..", "data", "picture1.png")  
  
 # Загрузка данных  
 try:  
 self.clubs\_df = load\_data(self.data\_file\_path, "clubs\_normalized")  
 self.matches\_df = load\_data(  
 self.data\_file\_path, "matches\_normalized")  
 self.managers\_df = load\_data(self.data\_file\_path, "club\_managers")  
 except FileNotFoundError:  
 messagebox.showerror("Ошибка", "Файл данных не найден.")  
 self.destroy()  
 return  
  
 self.button\_width = 20  
 self.button\_height = 2  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 *"""  
 Создает и настраивает виджеты в главном окне приложения.  
 """* label = ttk.Label(text='Добро пожаловать', font=('Open Sans Light', 22),  
 justify='center', foreground='black', background='white')  
  
 label.grid(row=0, column=0, padx=10, pady=10)  
 self.columnconfigure(index=0, weight=1)  
 self.columnconfigure(index=1, weight=2)  
  
 # Конфигурация кнопок  
 button\_config = {  
 'bg': "#9400D3",  
 'fg': "white",  
 '*padx*': 25,  
 '*pady*': 20,  
 'font': ("Times New Roman", 14),  
 'width': self.button\_width  
 }  
  
 button\_add\_clubs = tk.Button(text='Добавить клуб', command=lambda: club\_operations.add\_club(self),  
 \*\*button\_config)  
 button\_show\_clubs = tk.Button(text='Просмотреть клубы', command=lambda: club\_operations.view\_clubs(self),  
 \*\*button\_config)  
 button\_show\_graphs = tk.Button(text='Посмотреть графики', command=self.open\_view\_graphs,  
 \*\*button\_config)  
 button\_show\_excel = tk.Button(text='Открыть таблицу Excel', command=self.view\_excel\_table,  
 \*\*button\_config)  
 button\_generate\_reports = tk.Button(text='Создать отчеты', command=generate\_reports,  
 \*\*button\_config)  
 button\_open\_reports = tk.Button(text='Открыть отчеты', command=open\_reports,  
 \*\*button\_config)  
  
 button\_add\_clubs.grid(row=1, column=0, pady=5)  
 button\_show\_clubs.grid(row=2, column=0, pady=5)  
 button\_show\_graphs.grid(row=3, column=0, pady=5)  
 button\_show\_excel.grid(row=4, column=0, pady=5)  
 button\_generate\_reports.grid(row=5, column=0, pady=5)  
 button\_open\_reports.grid(row=6, column=0, pady=5)  
  
 self.display\_frame = tk.Frame(self, bg="white")  
 self.display\_frame.grid(  
 row=0, column=1, rowspan=7, padx=10, pady=25, sticky="nsew")  
  
 self.canvas = tk.Canvas(  
 self.display\_frame, bg="white", height=600, width=800)  
 self.canvas.pack(fill="both", expand=True)  
  
 try:  
 self.img = Image.open(self.picture1\_file\_path)  
 self.img = self.img.resize((800, 600))  
 self.img = ImageTk.PhotoImage(self.img)  
 self.canvas.create\_image(0, 0, anchor='nw', image=self.img)  
 except Exception as e:  
 messagebox.showerror(  
 "Ошибка", f"Не удалось загрузить изображение: {e}")  
  
 def view\_excel\_table(self):  
 *"""  
 Отображает таблицу Excel с данными о клубах в фрейме отображения и предоставляет  
 возможность фильтрации данных.  
 """* def reset\_filter():  
 *"""  
 Сбрасывает примененный фильтр и отображает исходные данные о клубах.  
 """* for item in tree.get\_children():  
 tree.delete(item)  
 for \_, row in self.clubs\_df.iterrows():  
 tree.insert("", tk.END, values=tuple(row))  
  
 for widget in self.display\_frame.winfo\_children():  
 widget.destroy()  
  
 tree = ttk.Treeview(self.display\_frame, columns=tuple(  
 self.clubs\_df.columns), show='headings')  
 for col in self.clubs\_df.columns:  
 tree.heading(col, text=col)  
  
 for \_, row in self.clubs\_df.iterrows():  
 tree.insert("", tk.END, values=tuple(row))  
  
 scrollbar = ttk.Scrollbar(  
 self.display\_frame, orient="vertical", command=tree.yview)  
 tree.configure(yscrollcommand=scrollbar.set)  
 tree.pack(fill="both", expand=True)  
 scrollbar.pack(side="right", fill="y")  
  
 filter\_frame = tk.Frame(self.display\_frame, bg="white")  
 filter\_frame.pack(pady=10)  
  
 label\_filter = tk.Label(  
 filter\_frame, text="Введите club\_id:", bg="white")  
 label\_filter.grid(row=0, column=0)  
  
 entry\_filter = tk.Entry(filter\_frame)  
 entry\_filter.grid(row=0, column=1)  
  
 filter\_button = tk.Button(filter\_frame, text="Применить фильтр",  
 command=lambda: filter.apply\_filter(tree, entry\_filter.get(), self.clubs\_df))  
 filter\_button.grid(row=0, column=2)  
  
 reset\_button = tk.Button(  
 filter\_frame, text="Сбросить фильтр", command=reset\_filter)  
 reset\_button.grid(row=0, column=3)  
  
 def show\_graph(self, graph\_function):  
 show\_graph(self.display\_frame, graph\_function)  
  
 def open\_view\_graphs(self):  
 open\_view\_graphs(self.display\_frame)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = Application()  
 app.mainloop()

club\_operations.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Осинцев К.  
"""*import tkinter as tk  
from tkinter import messagebox  
import pandas as pd  
from library import save\_data  
  
  
def add\_club(app, button\_color=None):  
 *"""  
 Отображает форму для добавления нового клуба.  
  
 Аргументы:  
 app (Application): Экземпляр главного приложения.  
 button\_color (str): Цвет кнопки для добавления клуба.  
  
 Действия:  
 Очищает текущий фрейм отображения и создает новую форму для ввода данных клуба.  
 При нажатии кнопки "Добавить клуб" вызывается функция success\_added\_club.  
 """* for widget in app.display\_frame.winfo\_children():  
 widget.destroy()  
  
 form\_frame = tk.Frame(app.display\_frame, bg="grey")  
 form\_frame.pack(fill="both", expand=True)  
  
 fields = [  
 ("ID клуба", app.entry\_club\_id),  
 ("Название клуба", app.entry\_club\_name),  
 ("Позиция клуба", app.entry\_club\_position),  
 ("Имя менеджера", app.entry\_manager\_name),  
 ("Стратегия клуба", app.entry\_club\_formation)  
 ]  
  
 app.entries = {}  
  
 for label\_text, entry\_var in fields:  
 label = tk.Label(form\_frame, text=label\_text, bg="white")  
 label.pack(pady=5)  
 entry\_var = tk.Entry(form\_frame)  
 entry\_var.pack(pady=5)  
 app.entries[label\_text] = entry\_var  
  
 submit\_button = tk.Button(form\_frame, text="Добавить клуб", command=lambda: success\_added\_club(app), bg=button\_color,  
 fg="#9400D3", padx=10, pady=5, font=("Times New Roman", 14))  
 submit\_button.pack(pady=10)  
  
  
def success\_added\_club(app):  
 *"""  
 Добавляет новый клуб в DataFrame и сохраняет обновленные данные.  
  
 Аргументы:  
 app (Application): Экземпляр главного приложения.  
  
 Действия:  
 Получает данные из полей ввода, добавляет новую строку в DataFrame клубов,  
 сохраняет обновленный DataFrame в файл и отображает сообщение о успешном добавлении клуба.  
 """* new\_row = pd.DataFrame([[  
 app.entries["ID клуба"].get(),  
 app.entries["Название клуба"].get(),  
 app.entries["Позиция клуба"].get(),  
 app.entries["Имя менеджера"].get(),  
 app.entries["Стратегия клуба"].get()  
 ]], columns=app.clubs\_df.columns)  
  
 app.clubs\_df = pd.concat([app.clubs\_df, new\_row], ignore\_index=True)  
  
 save\_data(app.data\_file\_path, clubs\_normalized=app.clubs\_df, matches\_normalized=app.matches\_df,  
 club\_managers=app.managers\_df)  
  
 messagebox.showinfo("Добавить клуб", "Клуб успешно добавлен.")  
  
  
def view\_clubs(app):  
 *"""  
 Отображает таблицу с данными о клубах.  
  
 Аргументы:  
 app (Application): Экземпляр главного приложения.  
  
 Действия:  
 Вызывает метод view\_excel\_table главного приложения для отображения данных о клубах.  
 """* app.view\_excel\_table()

excel\_table.py :

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Бабенко А.  
"""*import tkinter as tk  
from tkinter import ttk  
import filter  
  
  
def view\_excel\_table(clubs\_df):  
 *"""  
 Отображает окно для просмотра таблицы Excel с данными о клубах.  
  
 Аргументы:  
 clubs\_df (pandas.DataFrame): DataFrame, содержащий данные о клубах.  
  
 Действия:  
 Создает новое окно с Treeview для отображения данных о клубах.  
 Добавляет возможность фильтрации данных по club\_id.  
 """* excel\_window = tk.Toplevel()  
 excel\_window.title("Просмотр таблицы Excel")  
  
 tree = ttk.Treeview(excel\_window)  
  
 tree["columns"] = tuple(clubs\_df.columns)  
 for col in tree["columns"]:  
 tree.heading(col, text=col)  
  
 for index, row in clubs\_df.iterrows():  
 tree.insert("", tk.END, values=tuple(row))  
  
 scrollbar = ttk.Scrollbar(  
 excel\_window, orient="vertical", command=tree.yview)  
 scrollbar.pack(side="right", fill="y")  
 tree.configure(yscrollcommand=scrollbar.set)  
  
 tree.pack(fill="both", expand=True)  
  
 filter\_frame = tk.Frame(excel\_window)  
 filter\_frame.pack(pady=10)  
  
 label\_filter = tk.Label(filter\_frame, text="Введите club\_id:")  
 label\_filter.grid(row=0, column=0)  
  
 entry\_filter = tk.Entry(filter\_frame)  
 entry\_filter.grid(row=0, column=1)  
  
 filter\_button = tk.Button(filter\_frame, text="Применить фильтр",  
 command=lambda: filter.apply\_filter(tree, entry\_filter.get(), clubs\_df))  
 filter\_button.grid(row=0, column=2)

Filter.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Свистунов А.  
"""*import tkinter as tk  
  
  
def apply\_filter(tree, club\_id, clubs\_df):  
 *"""  
 Применяет фильтр к данным и обновляет отображение дерева.  
  
 Аргументы:  
 tree (ttk. Treeview): Виджет Treeview, который отображает данные.  
 club\_id (str): Значение club\_id для фильтрации данных.  
 clubs\_df (pandas.DataFrame): DataFrame, содержащий данные о клубах.  
  
 Действия:  
 Очищает текущее содержимое дерева и вставляет только те строки, которые  
 соответствуют указанному club\_id.  
 """* # Очищаем дерево перед применением фильтра  
 tree.delete(\*tree.get\_children())  
  
 # Применяем фильтр по club\_id  
 filtered\_data = clubs\_df.loc[clubs\_df['club\_id'] == int(club\_id)]  
  
 # Вставляем только отфильтрованные данные в дерево  
 for index, row in filtered\_data.iterrows():  
 tree.insert("", tk.END, values=tuple(row))  
  
  
def reset\_filter(tree, entry\_filter, clubs\_df):  
 *"""  
 Сбрасывает фильтр и возвращает полное отображение данных в дереве.  
  
 Аргументы:  
 tree (ttk. Treeview): Виджет Treeview, который отображает данные.  
 Entry\_filter (tk. Entry): Поле ввода для значения club\_id.  
 Clubs\_df (pandas. DataFrame): DataFrame, содержащий данные о клубах.  
  
 Действия:  
 Очищает поле ввода, очищает текущее содержимое дерева и вставляет  
 все строки из DataFrame обратно в дерево.  
 """* # Очищаем поле ввода для фильтрации  
 entry\_filter.delete(0, tk.END)  
  
 # Очищаем дерево  
 tree.delete(\*tree.get\_children())  
  
 # Вставляем все данные из DataFrame при загрузке  
 for index, row in clubs\_df.iterrows():  
 tree.insert("", tk.END, values=tuple(row))

graph\_display.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Бабенко А.  
"""*import os  
import sys  
import tkinter as tk  
from tkinter import messagebox  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
from PIL import Image, ImageTk  
import graph\_reports  
  
sys.path.append("../../work")  
  
  
def show\_graph(display\_frame, graph\_function):  
 *"""  
 Отображает график, сгенерированный указанной функцией, в указанном фрейме отображения.  
  
 Аргументы:  
 display\_frame (tk.Frame): Фрейм для отображения графика.  
 graph\_function (function): Функция, которая генерирует отображаемый график.  
 """* # Удаление всех виджетов из фрейма отображения  
 for widget in display\_frame.winfo\_children():  
 widget.destroy()  
  
 graphs\_frame = tk.Frame(display\_frame, bg="white")  
 graphs\_frame.pack(fill="both", expand=True)  
  
 # Генерация графика с помощью переданной функции  
 fig = graph\_function()  
  
 canvas = tk.Canvas(graphs\_frame, bg="white", height=600, width=800)  
 canvas.pack(fill="both", expand=True)  
  
 try:  
 global img  
 base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 img = Image.open(os.path.join(base\_dir, "outputs", fig))  
 img = img.resize((800, 600))  
 img = ImageTk.PhotoImage(img)  
 canvas.create\_image(0, 0, anchor='nw', image=img)  
 except Exception as e:  
 messagebox.showerror(  
 "Ошибка", f"Не удалось загрузить изображение: {e}")  
  
  
def open\_view\_graphs(display\_frame):  
 *"""  
 Открывает фрейм для выбора и отображения различных графиков.  
  
 Аргументы:  
 display\_frame (tk.Frame): Фрейм для отображения графиков.  
 """* # Удаление всех виджетов из фрейма отображения  
 for widget in display\_frame.winfo\_children():  
 widget.destroy()  
  
 # Создание фрейма для отображения графиков  
 graphs\_frame = tk.Frame(display\_frame, bg="white")  
 graphs\_frame.pack(fill="both", expand=True)  
  
 button\_color = "#9400D3"  
  
 # Список графиков с соответствующими функциями  
 graphs = [  
 ("График 1", graph\_reports.view\_graph\_1),  
 ("График 2", graph\_reports.view\_graph\_2),  
 ("График 3", graph\_reports.view\_graph\_3),  
 ("График 4", graph\_reports.view\_graph\_4),  
 ("График 5", graph\_reports.view\_graph\_5),  
 ("График 6", graph\_reports.view\_graph\_6),  
 ("График 7", graph\_reports.view\_graph\_7),  
 ]  
  
 # Создание кнопок для выбора графиков  
 for text, command in graphs:  
 button = tk.Button(graphs\_frame, text=text, command=lambda cmd=command: show\_graph(display\_frame, cmd),  
 bg=button\_color, fg="white", padx=10, pady=5,  
 font=("Times New Roman", 14), width=20)  
 button.pack(pady=10)

graph\_reports.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Бабенко А.  
"""*import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import os  
  
output\_folder = 'outputs'  
os.makedirs(output\_folder, exist\_ok=True)  
  
  
def view\_graph\_1():  
 *"""  
 Создает и отображает столбчатую диаграмму топ-30 команд по количеству матчей.  
  
 Действия:  
 1. Загружает данные из файла new\_normalized\_data.xlsx.  
 2. Группирует данные по количеству матчей для каждого клуба.  
 3. Строит столбчатую диаграмму для топ-30 клубов.  
 4. Сохраняет диаграмму в файл и отображает ее.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
  
 clubs\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='clubs\_normalized')  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
  
 matches\_count = matches\_normalized.groupby(  
 'home\_club\_id').size().reset\_index(name='matches\_count')  
 matches\_count = matches\_count.merge(clubs\_normalized[['club\_id', 'club\_name']], left\_on='home\_club\_id',  
 right\_on='club\_id')  
  
 top\_30\_matches\_count = matches\_count.sort\_values(  
 by='matches\_count', ascending=False).head(204)  
 colors = sns.color\_palette('Purples', n\_colors=30)[::-1]  
  
 plt.figure(figsize=(12, 8))  
 sns.barplot(x='club\_name', y='matches\_count', data=top\_30\_matches\_count, hue='club\_name', palette=colors,  
 legend=False)  
 plt.xlabel('Название клуба', fontsize=12)  
 plt.ylabel('Количество матчей', fontsize=12)  
 plt.title('Топ-30 команд по количеству матчей', fontsize=14)  
 plt.xticks(rotation=90, fontsize=10)  
 plt.yticks(range(0, top\_30\_matches\_count['matches\_count'].max() + 1, 5))  
 plt.tight\_layout()  
 plt.savefig(os.path.join(output\_folder,  
 'top\_30\_matches\_per\_club\_*barplot*.png'))  
  
 return 'top\_30\_matches\_per\_club\_*barplot*.png'  
  
  
def view\_graph\_2():  
 *"""  
 Создает и отображает столбчатую диаграмму топ-20 тренеров по количеству побед.  
  
 Действия:  
 1. Загружает данные из файла new\_normalized\_data.xlsx.  
 2. Группирует данные по количеству побед для каждого тренера.  
 3. Строит столбчатую диаграмму для топ-20 тренеров.  
 4. Сохраняет диаграмму в файл и отображает ее.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
  
 coach\_wins = matches\_normalized.groupby(  
 'home\_club\_manager\_name').size().reset\_index(name='wins')  
 top\_20\_coaches = coach\_wins.sort\_values(  
 by='wins', ascending=False).head(20)  
 colors = sns.color\_palette('Purples', n\_colors=20)[::-1]  
  
 plt.figure(figsize=(12, 8))  
 sns.barplot(x='home\_club\_manager\_name', y='wins', data=top\_20\_coaches, hue='home\_club\_manager\_name', palette=colors,  
 legend=False)  
 plt.xlabel('Имя тренера', fontsize=12)  
 plt.ylabel('Количество побед', fontsize=12)  
 plt.title('Топ 20 тренеров с наибольшим количеством побед', fontsize=14)  
 plt.xticks(rotation=90, fontsize=10)  
 plt.tight\_layout()  
 plt.savefig(os.path.join(output\_folder,  
 'top\_20\_coaches\_wins\_histogram\_purple\_gradient.png'))  
  
 return 'top\_20\_coaches\_wins\_histogram\_purple\_gradient.png'  
  
  
def view\_graph\_3():  
 *"""  
 Создает и отображает категоризированную диаграмму “box-and-whiskers”.  
  
 Действия:  
 1. Загружает данные из файла new\_normalized\_data.xlsx.  
 2. Строит boxplot диаграмму голов домашнего клуба по ID соревнования.  
 3. Сохраняет диаграмму в файл и отображает ее.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
  
 quant\_*qual*\_data\_box = matches\_normalized[[  
 'home\_club\_goals', 'competition\_id']]  
 colors = sns.color\_palette('Purples', n\_colors=29)[::-1]  
  
 plt.figure(figsize=(12, 8))  
 sns.boxplot(data=quant\_qual\_data\_box, x='competition\_id', y='home\_club\_goals', hue='competition\_id',  
 legend=False, palette=colors, showfliers=False)  
 plt.xlabel('ID соревнования', fontsize=13)  
 plt.ylabel('Голы домашнего клуба', fontsize=13)  
 plt.title('Категоризированная диаграмма “box-and-whiskers”', fontsize=15)  
 plt.tight\_layout()  
 plt.savefig(os.path.join(output\_folder, 'categorized\_boxplot.png'))  
  
 return 'categorized\_boxplot.png'  
  
  
def view\_graph\_4():  
 *"""  
 Создает и отображает категоризированную диаграмму рассеивания.  
  
 Действия:  
 1. Загружает данные из файла new\_normalized\_data.xlsx.  
 2. Строит scatterplot диаграмму голов домашнего клуба против голов гостевого клуба,  
 категоризированную по ID соревнования.  
 3. Сохраняет диаграмму в файл и отображает ее.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
  
 quant\_quant\_*qual*\_data = matches\_normalized[[  
 'home\_club\_goals', 'away\_club\_goals', 'competition\_id']]  
  
 plt.figure(figsize=(12, 8))  
 sns.scatterplot(data=quant\_quant\_qual\_data, x='home\_club\_goals', y='away\_club\_goals', hue='competition\_id',  
 palette='viridis', s=100)  
 plt.xlabel('Голы домашнего клуба', fontsize=13)  
 plt.ylabel('Голы гостевого клуба', fontsize=13)  
 plt.title('Категоризированная диаграмма рассеивания', fontsize=15)  
 plt.legend(title='ID соревнования', fontsize=10)  
 plt.tight\_layout()  
 plt.savefig(os.path.join(output\_folder, 'categorized\_scatterplot.png'))  
  
 return 'categorized\_scatterplot.png'  
  
  
def view\_graph\_5():  
 *"""  
 Создает и отображает гистограмму распределения количества матчей по сезонам.  
  
 Действия:  
 1. Загружает данные из файла new\_normalized\_data.xlsx.  
 2. Строит гистограмму количества матчей по сезонам.  
 3. Сохраняет гистограмму в файл и отображает ее.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
  
 plt.figure(figsize=(12, 8))  
 sns.histplot(data=matches\_normalized, x='season', bins=10, kde=True)  
 plt.xlabel('Сезон', fontsize=12)  
 plt.ylabel('Частота', fontsize=12)  
 plt.title('Распределение количества матчей по сезонам', fontsize=14)  
 plt.xticks(rotation=45, fontsize=10)  
 plt.tight\_layout()  
 plt.savefig(os.path.join(output\_folder,  
 'matches\_per\_season\_histogram.png'))  
  
 return 'matches\_per\_season\_histogram.png'  
  
  
def view\_graph\_6():  
 *"""  
 Создает и отображает диаграмму рассеивания зависимости голов домашней команды от позиции клуба.  
  
 Действия:  
 1. Загружает данные из файла new\_normalized\_data.xlsx.  
 2. Рассчитывает позиции клубов на основе количества побед.  
 3. Строит scatterplot диаграмму голов домашней команды против позиции клуба.  
 4. Сохраняет диаграмму в файл и отображает ее.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
  
 home\_club\_wins = matches\_normalized.groupby(  
 'home\_club\_id')['home\_club\_goals'].count().reset\_index(name='wins')  
 home\_club\_wins['club\_position'] = home\_club\_wins['wins'].rank(  
 ascending=False, method='min')  
  
 matches\_with\_positions = matches\_normalized.merge(home\_club\_wins[['home\_club\_id', 'club\_position']],  
 left\_on='home\_club\_id', right\_on='home\_club\_id')  
  
 plt.figure(figsize=(12, 8))  
 sns.scatterplot(data=matches\_with\_positions, x='club\_position', y='home\_club\_goals', hue='season',  
 palette='viridis', s=100)  
 plt.xlabel('Позиция клуба', fontsize=13)  
 plt.ylabel('Голы домашней команды', fontsize=13)  
 plt.title('Зависимость голов домашней команды от позиции клуба', fontsize=15)  
 plt.tight\_layout()  
 plt.savefig(os.path.join(output\_folder,  
 'home\_goals\_vs\_club\_position\_scatterplot.png'))  
  
 return 'home\_goals\_vs\_club\_position\_scatterplot.png'  
  
  
def view\_graph\_7():  
 *"""  
 Создает и отображает диаграмму рассеивания зависимости голов гостевой команды от позиции клуба.  
  
 Действия:  
 1. Загружает данные из файла new\_normalized\_data.xlsx.  
 2. Рассчитывает позиции клубов на основе количества побед.  
 3. Строит scatterplot диаграмму голов гостевой команды против позиции клуба.  
 4. Сохраняет диаграмму в файл и отображает ее.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 clubs\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='clubs\_normalized')  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
 club\_managers = pd.read\_excel(data\_file\_path, sheet\_name='club\_managers')  
  
 matches\_count = matches\_normalized.groupby(  
 'home\_club\_id').size().reset\_index(name='matches\_count')  
 matches\_count = matches\_count.merge(clubs\_normalized[['club\_id', 'club\_name']], left\_on='home\_club\_id',  
 right\_on='club\_id')  
  
 # Рассчитаем позиции клубов на основе количества побед  
 away\_club\_wins = matches\_normalized.groupby(  
 'away\_club\_id')['away\_club\_goals'].count().reset\_index(name='wins')  
 away\_club\_wins['club\_position'] = away\_club\_wins['wins'].rank(  
 ascending=False, method='min')  
  
 # Объединим информацию о позициях клубов с данными о матчах  
 matches\_with\_positions\_away = matches\_normalized.merge(away\_club\_wins[['away\_club\_id', 'club\_position']],  
 left\_on='away\_club\_id', right\_on='away\_club\_id')  
  
 # График зависимости количества голов гостевой команды от позиции в таблице  
 plt.figure(figsize=(12, 8))  
 sns.scatterplot(data=matches\_with\_positions\_away, x='club\_position', y='away\_club\_goals', hue='season',  
 palette='viridis', s=100)  
 plt.xlabel('Позиция в таблице', fontsize=13)  
 plt.ylabel('Голы гостевой команды', fontsize=13)  
 plt.title(  
 'Зависимость количества голов гостевой команды от позиции в таблице', fontsize=15)  
 plt.xticks(fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig(os.path.join(output\_folder,  
 'away\_goals\_vs\_club\_position\_scatterplot.png'))  
  
 return 'away\_goals\_vs\_club\_position\_scatterplot.png'

Graph\_window.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Сорокин Г.  
"""*import tkinter as tk  
import graph\_reports  
  
  
def open\_view\_graphs\_window():  
 *"""  
 Открывает новое окно с кнопками для просмотра различных графиков.  
  
 Кнопки:  
 - График 1: Отображает график 1.  
 - График 2: Отображает график 2.  
 - График 3: Отображает график 3.  
 - График 4: Отображает график 4.  
 - График 5: Отображает график 5.  
 - График 6: Отображает график 6.  
 - График 7: Отображает график 7.  
  
 Каждая кнопка вызывает соответствующую функцию из модуля graph\_reports для отображения графика.  
 """* clubs\_window = tk.Toplevel()  
 clubs\_window.title("View Graphs")  
 clubs\_window.geometry("500x500")  
  
 button\_color = "#9400D3"  
 button\_config = {  
 "bg": button\_color,  
 "fg": "white",  
 "*padx*": 10,  
 "*pady*": 5,  
 "font": ("Times new roman", 14)  
 }  
  
 buttons = [  
 ("График 1", graph\_reports.view\_graph\_1),  
 ("График 2", graph\_reports.view\_graph\_2),  
 ("График 3", graph\_reports.view\_graph\_3),  
 ("График 4", graph\_reports.view\_graph\_4),  
 ("График 5", graph\_reports.view\_graph\_5),  
 ("График 6", graph\_reports.view\_graph\_6),  
 ("График 7", graph\_reports.view\_graph\_7)  
 ]  
  
 for text, command in buttons:  
 tk.Button(clubs\_window, text=text, command=command,  
 \*\*button\_config).pack(pady=10)

library.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Свистунов А.  
"""*import pandas as pd  
import os  
  
  
def load\_data(file\_path, sheet\_name):  
 *"""  
 Загружает данные из файла Excel в объект DataFrame.  
  
 Параметры:  
 file\_path (str): Путь к файлу Excel.  
 sheet\_name (str): Название листа, из которого следует загрузить данные.  
  
 Возвращает:  
 pandas.DataFrame: DataFrame с загруженными данными из указанного листа файла Excel.  
 """* return pd.read\_excel(file\_path, sheet\_name=sheet\_name)  
  
  
def save\_data(file\_path, \*\*dataframes):  
 *"""  
 Сохраняет несколько DataFrame в один Excel-файл.  
  
 Параметры:  
 file\_path (str): Путь к файлу Excel.  
 \*\*dataframes: Набор именованных аргументов, где ключ - название листа, а значение - DataFrame.  
  
 Выход:  
 None  
 """* with pd.ExcelWriter(file\_path, engine='openpyxl') as writer:  
 for sheet\_name, df in dataframes.items():  
 df.to\_excel(writer, sheet\_name=sheet\_name, index=False)  
  
  
def open\_file(file\_path):  
 *"""  
 Открывает файл с помощью приложения по умолчанию в системе.  
  
 Параметры:  
 file\_path (str): Путь к файлу, который нужно открыть.  
  
 Выход:  
 None  
 """* try:  
 os.startfile(file\_path)  
 except FileNotFoundError:  
 raise FileNotFoundError(f"Файл не найден: {file\_path}")  
 except Exception as e:  
 raise Exception(f"Произошла ошибка при открытии файла: {e}")

sheet\_report.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Бабенко А. Осинцев К.  
"""*import pandas as pd  
import os  
  
  
def clean\_data(data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Очищает данные, удаляя ведущие и завершающие пробелы и преобразуя строки к нижнему регистру.  
  
 Parameters  
 ----------  
 data : pd.DataFrame  
 Входной DataFrame, который нужно очистить.  
  
 Returns  
 -------  
 pd.DataFrame  
 Очищенный DataFrame.  
  
 """* if 'home\_club\_manager\_name' in data.columns:  
 data['home\_club\_manager\_name'] = data['home\_club\_manager\_name'].str.strip(  
 ).str.lower()  
 return data  
  
  
def generate\_text\_reports(clubs\_normalized: pd.DataFrame, matches\_normalized: pd.DataFrame) -> dict:  
 *"""  
 Генерирует текстовые отчеты на основе предоставленных данных.  
  
 Parameters  
 ----------  
 clubs\_normalized : pd.DataFrame  
 Данные о клубах.  
 matches\_normalized : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 dict  
 Словарь, где ключ - название отчета, значение - сводная таблица.  
 """* reports = {}  
  
 try:  
 def matches\_per\_club\_report(matches\_data: pd.DataFrame, clubs\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о количестве матчей, проведенных каждым клубом.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
 clubs\_data : pd.DataFrame  
 Данные о клубах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Объединяем данные о матчах с данными о клубах для получения названий клубов  
 merged\_data = matches\_data.merge(clubs\_data[['club\_id', 'club\_name']], left\_on='home\_club\_id',  
 right\_on='club\_id', how='left')  
  
 # Строим сводную таблицу  
 pivot\_table = pd.pivot\_table(merged\_data, index='club\_name', columns='season', values='game\_id',  
 aggfunc='count', fill\_value=0)  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_club\_report': {error}")  
 return pd.DataFrame()  
  
 def matches\_per\_coach\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о всех матчах, проведенных каждым тренером.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Отбираем нужные столбцы и сортируем данные о матчах по тренеру  
 matches\_sorted = matches\_data[['home\_club\_manager\_name', 'game\_id', 'home\_club\_id', 'away\_club\_id',  
 'season', 'home\_club\_goals', 'away\_club\_goals']].sort\_values(by='home\_club\_manager\_name')  
  
 # Переименовываем столбцы для удобства  
 matches\_sorted.columns = ['Тренер', 'ID матча', 'ID домашнего клуба',  
 'ID гостевого клуба', 'Сезон', 'Голы домашнего клуба', 'Голы гостевого клуба']  
  
 # Группируем данные по тренеру  
 grouped = matches\_sorted.groupby('Тренер')  
  
 # Создаем пустой DataFrame для хранения результатов  
 result = pd.DataFrame()  
  
 # Заполняем результат  
 for name, group in grouped:  
 result = pd.concat([result, group])  
  
 return result  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_coach\_report': {error}")  
 return pd.DataFrame()  
  
 def average\_goals\_per\_match\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о среднем количестве голов в каждом матче.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Создаем сводную таблицу, показывающую среднее количество голов в домашних и выездных матчах для каждого клуба в каждом сезоне  
 pivot\_table = pd.pivot\_table(matches\_data, index=['home\_club\_id', 'away\_club\_id', 'season'],  
 values=['home\_club\_goals', 'away\_club\_goals'], aggfunc='mean', fill\_value=0)  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'average\_goals\_per\_match\_report': {error}")  
 return pd.DataFrame()  
  
 def matches\_per\_season\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о количестве матчей в разные сезоны.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Группируем данные о матчах по сезону и считаем количество матчей  
 matches\_per\_season = matches\_data.groupby(  
 'season').size().reset\_index(name='matches\_count')  
  
 # Строим сводную таблицу  
 pivot\_table = matches\_per\_season.set\_index('season')  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_season\_report': {error}")  
 return pd.DataFrame()  
  
 reports['Отчёт матчей по клубам'] = matches\_per\_club\_report(  
 matches\_normalized, clubs\_normalized)  
 reports['Отчёт матчей по тренерам'] = matches\_per\_coach\_report(  
 matches\_normalized)  
 reports['Отчёт среднее по голам'] = average\_goals\_per\_match\_report(  
 matches\_normalized)  
 reports['Отчёт матчи по сезонам'] = matches\_per\_season\_report(  
 matches\_normalized)  
  
 except Exception as e:  
 print(f"raise error: {e}")  
  
 return reports  
  
  
def main():  
 base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 clubs\_normalized = pd.read\_excel(data\_file\_path,  
 sheet\_name='clubs\_normalized')  
 matches\_normalized = pd.read\_excel(data\_file\_path,  
 sheet\_name='matches\_normalized')  
  
 if not clubs\_normalized.empty and not matches\_normalized.empty:  
 reports = generate\_text\_reports(clubs\_normalized, matches\_normalized)  
  
 with pd.ExcelWriter('reports.xlsx', engine='openpyxl') as writer:  
 for report\_name, report\_table in reports.items():  
 report\_table.to\_excel(writer, sheet\_name=report\_name)  
  
 print("Отчеты успешно записаны в файл Excel.")  
 else:  
 print("Не удалось загрузить данные для генерации отчетов.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

seaborn\_pictures.py:  
  
# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Бабенко А. Осинцев К.  
"""*import os  
import pandas as pd  
  
  
def clean\_data(data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Очищает данные, удаляя ведущие и завершающие пробелы и преобразуя строки к нижнему регистру.  
  
 Parameters  
 ----------  
 data : pd.DataFrame  
 Входной DataFrame, который нужно очистить.  
  
 Returns  
 -------  
 pd.DataFrame  
 Очищенный DataFrame.  
  
 """* if 'home\_club\_manager\_name' in data.columns:  
 data['home\_club\_manager\_name'] = data['home\_club\_manager\_name'].str.strip(  
 ).str.lower()  
 return data  
  
  
def generate\_text\_reports(clubs\_normalized: pd.DataFrame, matches\_normalized: pd.DataFrame) -> dict:  
 *"""  
 Генерирует текстовые отчеты на основе предоставленных данных.  
  
 Parameters  
 ----------  
 clubs\_normalized : pd.DataFrame  
 Данные о клубах.  
 matches\_normalized : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 dict  
 Словарь, где ключ - название отчета, значение - сводная таблица.  
 """* reports = {}  
  
 try:  
 def matches\_per\_club\_report(matches\_data: pd.DataFrame, clubs\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о количестве матчей, проведенных каждым клубом.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
 clubs\_data : pd.DataFrame  
 Данные о клубах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Объединяем данные о матчах с данными о клубах для получения названий клубов  
 merged\_data = matches\_data.merge(clubs\_data[['club\_id', 'club\_name']], left\_on='home\_club\_id',  
 right\_on='club\_id', how='left')  
  
 # Строим сводную таблицу  
 pivot\_table = pd.pivot\_table(merged\_data, index='club\_name', columns='season', values='game\_id',  
 aggfunc='count', fill\_value=0)  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_club\_report': {error}")  
 return pd.DataFrame()  
  
 def matches\_per\_coach\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о всех матчах, проведенных каждым тренером.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Отбираем нужные столбцы и сортируем данные о матчах по тренеру  
 matches\_sorted = matches\_data[['home\_club\_manager\_name', 'game\_id', 'home\_club\_id', 'away\_club\_id',  
 'season', 'home\_club\_goals', 'away\_club\_goals']].sort\_values(by='home\_club\_manager\_name')  
  
 # Переименовываем столбцы для удобства  
 matches\_sorted.columns = ['Тренер', 'ID матча', 'ID домашнего клуба',  
 'ID гостевого клуба', 'Сезон', 'Голы домашнего клуба', 'Голы гостевого клуба']  
  
 # Группируем данные по тренеру  
 grouped = matches\_sorted.groupby('Тренер')  
  
 # Создаем пустой DataFrame для хранения результатов  
 result = pd.DataFrame()  
  
 # Заполняем результат  
 for group in grouped:  
 result = pd.concat([result, group])  
  
 return result  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_coach\_report': {error}")  
 return pd.DataFrame()  
  
 def average\_goals\_per\_match\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о среднем количестве голов в каждом матче.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Создаем сводную таблицу, показывающую среднее количество голов в домашних и выездных матчах для каждого клуба в каждом сезоне  
 pivot\_table = pd.pivot\_table(matches\_data, index=['home\_club\_id', 'away\_club\_id', 'season'],  
 values=['home\_club\_goals', 'away\_club\_goals'], aggfunc='mean', fill\_value=0)  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'average\_goals\_per\_match\_report': {error}")  
 return pd.DataFrame()  
  
 def matches\_per\_season\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о количестве матчей в разные сезоны.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Группируем данные о матчах по сезону и считаем количество матчей  
 matches\_per\_season = matches\_data.groupby(  
 'season').size().reset\_index(name='matches\_count')  
  
 # Строим сводную таблицу  
 pivot\_table = matches\_per\_season.set\_index('season')  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_season\_report': {error}")  
 return pd.DataFrame()  
  
 reports['Отчёт матчей по клубам'] = matches\_per\_club\_report(  
 matches\_normalized, clubs\_normalized)  
 reports['Отчёт матчей по тренерам'] = matches\_per\_coach\_report(  
 matches\_normalized)  
 reports['Отчёт среднее по голам'] = average\_goals\_per\_match\_report(  
 matches\_normalized)  
 reports['Отчёт матчи по сезонам'] = matches\_per\_season\_report(  
 matches\_normalized)  
  
 except Exception as e:  
 print(f"Произошла ошибка при генерации отчетов: {e}")  
  
 return reports  
  
  
def main():  
 base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
 clubs\_normalized = pd.read\_excel(data\_file\_path,  
 sheet\_name='clubs\_normalized')  
 matches\_normalized = pd.read\_excel(data\_file\_path,  
 sheet\_name='matches\_normalized')  
  
 if not clubs\_normalized.empty and not matches\_normalized.empty:  
 reports = generate\_text\_reports(clubs\_normalized, matches\_normalized)  
  
 with pd.ExcelWriter('reports.xlsx', engine='openpyxl') as writer:  
 for report\_name, report\_table in reports.items():  
 report\_table.to\_excel(writer, sheet\_name=report\_name)  
  
 print("Отчеты успешно записаны в файл Excel.")  
 else:  
 print("Не удалось загрузить данные для генерации отчетов.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

matplotlib\_pictures.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Бабенко А.  
"""*import pandas as pd  
import matplotlib.pyplot as plt  
import os  
  
  
def load\_and\_process\_data():  
 *"""  
 Загружает и обрабатывает данные из файла Excel.  
  
 Параметры:  
 file\_path (str): Путь к файлу Excel.  
  
 Возвращает:  
 pandas.DataFrame: DataFrame с данными о клубах.  
 pandas.DataFrame: DataFrame с данными о матчах.  
 pandas.DataFrame: DataFrame с данными о менеджерах клубов.  
 """* base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
  
 clubs\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='clubs\_normalized')  
 matches\_normalized = pd.read\_excel(  
 data\_file\_path, sheet\_name='matches\_normalized')  
 club\_managers = pd.read\_excel(data\_file\_path, sheet\_name='club\_managers')  
  
 return clubs\_normalized, matches\_normalized, club\_managers  
  
  
# График топ-30 команд по количеству матчей  
def plot\_top\_30\_matches\_per\_club(clubs\_normalized, matches\_normalized):  
 *"""  
 График топ-30 команд по количеству матчей.  
  
 Параметры:  
 clubs\_normalized (pandas.DataFrame): DataFrame с данными о клубах.  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* matches\_count = matches\_normalized.groupby(  
 'home\_club\_id').size().reset\_index(name='matches\_count')  
 matches\_count = matches\_count.merge(clubs\_normalized[['club\_id', 'club\_name']], left\_on='home\_club\_id',  
 right\_on='club\_id')  
 top\_30\_matches\_count = matches\_count.sort\_values(  
 by='matches\_count', ascending=False).head(204)  
  
 plt.figure(figsize=(12, 8))  
 plt.bar(top\_30\_matches\_count['club\_name'],  
 top\_30\_matches\_count['matches\_count'], color='purple')  
 plt.xlabel('Название клуба', fontsize=12)  
 plt.ylabel('Количество матчей', fontsize=12)  
 plt.title('Топ-30 команд по количеству матчей', fontsize=14)  
 plt.xticks(rotation=90, fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('top\_30\_matches\_per\_club\_*barplot*.png')  
 plt.show()  
  
  
# График топ-20 тренеров с наибольшим количеством побед  
def plot\_top\_20\_coaches\_wins(matches\_normalized):  
 *"""  
 График топ-20 тренеров с наибольшим количеством побед.  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* coach\_wins = matches\_normalized.groupby(  
 'home\_club\_manager\_name').size().reset\_index(name='wins')  
 top\_20\_coaches = coach\_wins.sort\_values(  
 by='wins', ascending=False).head(20)  
  
 plt.figure(figsize=(12, 8))  
 plt.bar(top\_20\_coaches['home\_club\_manager\_name'],  
 top\_20\_coaches['wins'], color='purple')  
 plt.xlabel('Имя тренера', fontsize=12)  
 plt.ylabel('Количество побед', fontsize=12)  
 plt.title('Топ 20 тренеров с наибольшим количеством побед', fontsize=14)  
 plt.xticks(rotation=90, fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('top\_20\_coaches\_wins\_histogram\_purple.png')  
 plt.show()  
  
  
# Категоризированная диаграмма "box-and-whiskers"  
def plot\_categorized\_boxplot(matches\_normalized):  
 *"""  
 Категоризированная диаграмма "box-and-whiskers".  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* quant\_*qual*\_data\_box = matches\_normalized[[  
 'home\_club\_goals', 'competition\_id']]  
  
 plt.figure(figsize=(12, 8))  
 plt.boxplot([quant\_qual\_data\_box[quant\_qual\_data\_box['competition\_id'] == i]['home\_club\_goals'] for i in  
 quant\_qual\_data\_box['competition\_id'].unique()], patch\_artist=True)  
 plt.xlabel('ID соревнования', fontsize=13)  
 plt.ylabel('Голы домашнего клуба', fontsize=13)  
 plt.title('Категоризированная диаграмма “box-and-whiskers”', fontsize=15)  
 plt.tight\_layout()  
 plt.savefig('categorized\_boxplot.png')  
 plt.show()  
  
  
# Категоризированная диаграмма рассеивания  
def plot\_categorized\_scatterplot(matches\_normalized):  
 *"""  
 Категоризированная диаграмма рассеивания.  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* quant\_quant\_*qual*\_data = matches\_normalized[[  
 'home\_club\_goals', 'away\_club\_goals', 'competition\_id']]  
  
 plt.figure(figsize=(12, 8))  
 for competition\_id in quant\_quant\_qual\_data['competition\_id'].unique():  
 data = quant\_quant\_qual\_data[quant\_quant\_qual\_data['competition\_id']  
 == competition\_id]  
 plt.scatter(data['home\_club\_goals'], data['away\_club\_goals'],  
 label=f'Competition ID {competition\_id}', s=100)  
 plt.xlabel('Голы домашнего клуба', fontsize=13)  
 plt.ylabel('Голы гостевого клуба', fontsize=13)  
 plt.title('Категоризированная диаграмма рассеивания', fontsize=15)  
 plt.legend(title='ID соревнования', fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('categorized\_scatterplot.png')  
 plt.show()  
  
  
# Распределение количества матчей по сезонам  
def plot\_matches\_per\_season(matches\_normalized):  
 *"""  
 Распределение количества матчей по сезонам.  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* matches\_per\_season = matches\_normalized['season'].value\_counts(  
 ).sort\_index()  
  
 plt.figure(figsize=(12, 8))  
 plt.bar(matches\_per\_season.index, matches\_per\_season.values, color='purple')  
 plt.xlabel('Сезон', fontsize=12)  
 plt.ylabel('Частота', fontsize=12)  
 plt.title('Распределение количества матчей по сезонам', fontsize=14)  
 plt.xticks(rotation=45, fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('matches\_per\_season\_histogram.png')  
 plt.show()  
  
  
# Зависимость голов домашней команды от позиции клуба  
def plot\_home\_goals\_vs\_club\_position(matches\_normalized):  
 *"""  
 Зависимость голов домашней команды от позиции клуба.  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* home\_club\_wins = matches\_normalized.groupby(  
 'home\_club\_id')['home\_club\_goals'].count().reset\_index(name='wins')  
 home\_club\_wins['club\_position'] = home\_club\_wins['wins'].rank(  
 ascending=False, method='min')  
 matches\_with\_positions = matches\_normalized.merge(home\_club\_wins[['home\_club\_id', 'club\_position']],  
 left\_on='home\_club\_id', right\_on='home\_club\_id')  
  
 plt.figure(figsize=(12, 8))  
 plt.scatter(matches\_with\_positions['club\_position'], matches\_with\_positions['home\_club\_goals'],  
 c=matches\_with\_positions['season'], cmap='viridis', s=100)  
 plt.xlabel('Позиция клуба', fontsize=13)  
 plt.ylabel('Голы домашней команды', fontsize=13)  
 plt.title('Зависимость голов домашней команды от позиции клуба', fontsize=15)  
 plt.xticks(fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.colorbar(label='Сезон')  
 plt.tight\_layout()  
 plt.savefig('home\_goals\_vs\_club\_position\_scatterplot.png')  
 plt.show()  
  
  
# Зависимость голов гостевой команды от позиции в таблице  
def plot\_away\_goals\_vs\_club\_position(matches\_normalized):  
 *"""  
 Зависимость голов гостевой команды от позиции в таблице.  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* away\_club\_wins = matches\_normalized.groupby(  
 'away\_club\_id')['away\_club\_goals'].count().reset\_index(name='wins')  
 away\_club\_wins['club\_position'] = away\_club\_wins['wins'].rank(  
 ascending=False, method='min')  
 matches\_with\_positions\_away = matches\_normalized.merge(away\_club\_wins[['away\_club\_id', 'club\_position']],  
 left\_on='away\_club\_id', right\_on='away\_club\_id')  
 plt.figure(figsize=(12, 8))  
 plt.scatter(matches\_with\_positions\_away['club\_position'], matches\_with\_positions\_away['away\_club\_goals'],  
 c=matches\_with\_positions\_away['season'], cmap='viridis', s=100)  
 plt.xlabel('Позиция в таблице', fontsize=13)  
 plt.ylabel('Голы гостевой команды', fontsize=13)  
 plt.title(  
 'Зависимость количества голов гостевой команды от позиции в таблице', fontsize=15)  
 plt.xticks(fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.colorbar(label='Сезон')  
 plt.tight\_layout()  
 plt.savefig('away\_goals\_vs\_club\_position\_scatterplot.png')  
 plt.show()  
  
  
# Топ-10 расстановок клубов по количеству побед  
def plot\_top\_10\_formations\_wins(matches\_normalized, clubs\_normalized):  
 *"""  
 Топ-10 расстановок клубов по количеству побед.  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
 clubs\_normalized (pandas.DataFrame): DataFrame с данными о клубах.  
  
 Возвращает:  
 None  
 """* # Сгруппировать данные по типу формации и подсчитать количество побед для каждой формации  
 formation\_wins = matches\_normalized.merge(clubs\_normalized[['club\_id', 'club\_formation']], left\_on='home\_club\_id',  
 right\_on='club\_id')  
 formation\_wins = formation\_wins.groupby(  
 'club\_formation').size().reset\_index(name='wins')  
 formation\_wins = formation\_wins.sort\_values(by='wins', ascending=False)  
  
 # Выбрать топ-10 расстановок по победам  
 top\_10\_formations = formation\_wins.head(10)  
  
 plt.figure(figsize=(12, 8))  
 plt.bar(top\_10\_formations['club\_formation'],  
 top\_10\_formations['wins'], color='purple')  
 plt.xlabel('Тип формации клуба', fontsize=13)  
 plt.ylabel('Количество побед', fontsize=13)  
 plt.title('Топ-10 расстановок клубов по количеству побед', fontsize=14)  
 plt.xticks(rotation=45, ha='right', fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('top\_10\_formations\_wins\_*barplot*.png')  
 plt.show()  
  
  
# Диаграмма рассеивания: Зависимость количества голов домашней команды от голов гостевой команды  
def plot\_home\_goals\_vs\_away\_goals\_scatterplot(matches\_normalized):  
 *"""  
 Диаграмма рассеивания: Зависимость количества голов домашней команды от голов гостевой команды.  
  
 Параметры:  
 matches\_normalized (pandas.DataFrame): DataFrame с данными о матчах.  
  
 Возвращает:  
 None  
 """* plt.figure(figsize=(12, 8))  
 plt.scatter(matches\_normalized['home\_club\_goals'],  
 matches\_normalized['away\_club\_goals'], c='blue', alpha=0.5)  
 plt.xlabel('Голы домашней команды', fontsize=13)  
 plt.ylabel('Голы гостевой команды', fontsize=13)  
 plt.title(  
 'Зависимость количества голов домашней команды от голов гостевой команды', fontsize=15)  
 plt.xticks(fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('home\_goals\_vs\_away\_goals\_scatterplot.png')  
 plt.show()  
  
  
base\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
data\_file\_path = os.path.join(  
 base\_dir, "..", "data", "new\_normalized\_data.xlsx")  
# Загрузка и обработка данных  
clubs\_normalized, matches\_normalized, club\_managers = load\_and\_process\_data()  
  
# Графики  
plot\_top\_30\_matches\_per\_club(clubs\_normalized, matches\_normalized)  
plot\_top\_20\_coaches\_wins(matches\_normalized)  
plot\_categorized\_boxplot(matches\_normalized)  
plot\_categorized\_scatterplot(matches\_normalized)  
plot\_matches\_per\_season(matches\_normalized)  
plot\_home\_goals\_vs\_club\_position(matches\_normalized)  
plot\_away\_goals\_vs\_club\_position(matches\_normalized)  
plot\_top\_10\_formations\_wins(matches\_normalized, clubs\_normalized)  
plot\_home\_goals\_vs\_away\_goals\_scatterplot(matches\_normalized)

reports.py:  
  
# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Сорокин Г.  
"""*import os  
import sys  
from tkinter import messagebox  
import subprocess  
import sheet\_report  
  
  
def open\_reports():  
 *"""Открыть файл отчетов.  
  
 Пытается открыть файл "reports.xlsx" при помощи приложения по умолчанию в системе.  
 Если файл не найден, выводит сообщение об ошибке.  
  
 Raises:  
 FileNotFoundError: Если файл отчетов не найден.  
 Exception: Если произошла ошибка при открытии файла отчетов.  
 """* try:  
 if os.name == 'nt':  
 os.startfile("reports.xlsx")  
 elif os.name == 'posix':  
 opener = "open" if sys.platform == "darwin" else "xdg-open"  
 subprocess.call([opener, "reports.xlsx"])  
 except FileNotFoundError:  
 messagebox.showerror("Ошибка", "Файл отчетов не найден.")  
 except Exception as e:  
 messagebox.showerror(  
 "Ошибка", f"Произошла ошибка при открытии файла отчетов: {e}")  
  
  
def generate\_reports():  
 *"""Создать отчеты.  
  
 Запускает процесс создания отчетов при помощи функции main из модуля sheet\_report.  
 Выводит сообщение об успешном создании отчетов.  
 """* sheet\_report.main()  
 messagebox.showinfo("Создать отчеты", "Отчеты успешно созданы.")

report.py:

# -\*- coding: utf-8 -\*-  
*"""  
Created on Sun May 15 20:04:57 2024  
  
@author: Бабенко А.  
"""*import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
  
def clean\_data(data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Очищает данные, удаляя ведущие и завершающие пробелы и преобразуя строки к нижнему регистру.  
  
 Parameters  
 ----------  
 data : pd.DataFrame  
 Входной DataFrame, который нужно очистить.  
  
 Returns  
 -------  
 pd.DataFrame  
 Очищенный DataFrame.  
  
 """* if 'home\_club\_manager\_name' in data.columns:  
 data['home\_club\_manager\_name'] = data['home\_club\_manager\_name'].str.strip(  
 ).str.lower()  
 return data  
  
  
def generate\_text\_reports(clubs\_normalized: pd.DataFrame, matches\_normalized: pd.DataFrame) -> dict:  
 *"""  
 Генерирует текстовые отчеты на основе предоставленных данных.  
  
 Parameters  
 ----------  
 clubs\_normalized : pd.DataFrame  
 Данные о клубах.  
 matches\_normalized : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 dict  
 Словарь, где ключ - название отчета, значение - сводная таблица.  
 """* reports = {}  
  
 try:  
 def matches\_per\_club\_report(matches\_data: pd.DataFrame, clubs\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о количестве матчей, проведенных каждым клубом.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
 clubs\_data : pd.DataFrame  
 Данные о клубах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Объединяем данные о матчах с данными о клубах для получения названий клубов  
 merged\_data = matches\_data.merge(clubs\_data[['club\_id', 'club\_name']], left\_on='home\_club\_id',  
 right\_on='club\_id', how='left')  
  
 # Строим сводную таблицу  
 pivot\_table = pd.pivot\_table(merged\_data, index='club\_name', columns='season', values='game\_id',  
 aggfunc='count', fill\_value=0)  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_club\_report': {error}")  
 return pd.DataFrame()  
  
 def matches\_per\_coach\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о количестве матчей, проведенных каждым тренером.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Группируем данные о матчах по домашнему тренеру и считаем количество матчей  
 matches\_count\_per\_coach = matches\_data.groupby('home\_club\_manager\_name').size().reset\_index(  
 name='matches\_count')  
  
 # Строим сводную таблицу  
 pivot\_table = pd.pivot\_table(matches\_count\_per\_coach, index='home\_club\_manager\_name',  
 values='matches\_count', aggfunc='sum')  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_coach\_report': {error}")  
 return pd.DataFrame()  
  
 def average\_goals\_per\_match\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о среднем количестве голов в матче.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Считаем среднее количество голов в матче  
 avg\_goals\_per\_match = matches\_data[[  
 'home\_club\_goals', 'away\_club\_goals']].mean().mean()  
  
 # Строим сводную таблицу  
 pivot\_table = pd.DataFrame(  
 {'Среднее количество голов в матче': [avg\_goals\_per\_match]})  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'average\_goals\_per\_match\_report': {error}")  
 return pd.DataFrame()  
  
 def matches\_per\_season\_report(matches\_data: pd.DataFrame) -> pd.DataFrame:  
 *"""  
 Отчет о количестве матчей в разные сезоны.  
  
 Parameters  
 ----------  
 matches\_data : pd.DataFrame  
 Данные о матчах.  
  
 Returns  
 -------  
 pd.DataFrame  
 Сводная таблица.  
 """* try:  
 # Группируем данные о матчах по сезону и считаем количество матчей  
 matches\_per\_season = matches\_data.groupby(  
 'season').size().reset\_index(name='matches\_count')  
  
 # Строим сводную таблицу  
 pivot\_table = matches\_per\_season.set\_index('season')  
  
 return pivot\_table  
 except Exception as error:  
 print(  
 f"Произошла ошибка при создании отчета 'matches\_per\_season\_report': {error}")  
 return pd.DataFrame()  
  
 reports['Отчёт матчей по тренерам'] = (matches\_per\_club\_report  
 (matches\_normalized, clubs\_normalized))  
 reports['Отчёт матчей по клубам'] = matches\_per\_coach\_report(  
 matches\_normalized)  
 reports['Отчёт среднее по голам'] = average\_goals\_per\_match\_report(  
 matches\_normalized)  
 reports['Отчёт матчи по сезонам'] = matches\_per\_season\_report(  
 matches\_normalized)  
  
 except Exception as e:  
 print(f"Произошла ошибка при генерации отчетов: {e}")  
  
 return reports  
  
  
def generate\_graphical\_reports():  
 *"""  
 Генерирует графические отчеты на основе предоставленных данных.  
  
 Возвращает  
 -------  
 None  
 Сохраняет графики в файлы и отображает их.  
  
 """* try:  
 clubs\_normalized = pd.read\_excel('../data/new\_normalized\_data.xlsx',  
 sheet\_name='clubs\_normalized')  
 matches\_normalized = pd.read\_excel('../data/new\_normalized\_data.xlsx',  
 sheet\_name='matches\_normalized')  
 # club\_managers = pd.read\_excel('../data/new\_normalized\_data.xlsx',  
 # sheet\_name='club\_managers')  
  
 matches\_count = matches\_normalized.groupby(  
 'home\_club\_id').size().reset\_index(name='matches\_count')  
 matches\_count = matches\_count.merge(clubs\_normalized[['club\_id', 'club\_name']],  
 left\_on='home\_club\_id', right\_on='club\_id')  
  
 top\_30\_matches\_count = matches\_count.sort\_values(  
 by='matches\_count', ascending=False).head(204)  
  
 colors = sns.color\_palette('Purples', n\_colors=30)[  
 ::-1] # Выбор 30 цветов из палитры Purples  
  
 # столбчатая гистограмма  
 plt.figure(figsize=(12, 8))  
 sns.barplot(x='club\_name', y='matches\_count', data=top\_30\_matches\_count, hue='club\_name', palette=colors,  
 legend=False)  
 plt.xlabel('Название клуба', fontsize=12)  
 plt.ylabel('Количество матчей', fontsize=12)  
 plt.title('Топ-30 команд по количеству матчей', fontsize=14)  
 plt.xticks(rotation=90, fontsize=10)  
 plt.yticks(  
 range(0, top\_30\_matches\_count['matches\_count'].max() + 1, 5))  
 plt.tight\_layout()  
 plt.savefig('top\_30\_matches\_per\_club\_bar plot.png')  
 plt.show()  
  
 # Подсчет количества побед для каждого тренера  
 coach\_wins = matches\_normalized.groupby(  
 'home\_club\_manager\_name').size().reset\_index(name='wins')  
  
 # Выбор топ 20 тренеров с наибольшим количеством побед  
 top\_20\_coaches = coach\_wins.sort\_values(  
 by='wins', ascending=False).head(20)  
  
 colors = sns.color\_palette('Purples', n\_colors=20)[::-1]  
  
 # Категоризированная диаграмма  
 plt.figure(figsize=(12, 8)) # Увеличиваем размер графика  
 sns.barplot(x='home\_club\_manager\_name', y='wins', data=top\_20\_coaches,  
 hue='home\_club\_manager\_name', palette=colors,  
 saturation=1.5, legend=False)  
 plt.xlabel('Имя тренера', fontsize=12)  
 plt.ylabel('Количество побед', fontsize=12)  
 plt.title('Топ-20 тренеров по количеству побед', fontsize=14)  
 # Поворачиваем и смещаем подписи оси x  
 plt.xticks(rotation=45, ha='right', fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('top\_20\_coaches\_wins\_bar plot.png')  
 plt.show()  
  
 # для категоризированной диаграммы "box-and-whiskers"  
 quant\_quail\_data\_box = matches\_normalized[[  
 'home\_club\_goals', 'competition\_id']]  
 colors = sns.color\_palette('Purples', n\_colors=29)[  
 ::-1] # Выбор 29 цветов из палитры Purples  
  
 # категоризированная диаграмма "box-and-whiskers"  
 plt.figure(figsize=(12, 8))  
 sns.boxplot(data=quant\_quail\_data\_box, x='competition\_id', y='home\_club\_goals', hue='competition\_id',  
 palette=colors, legend=False)  
 plt.xlabel('ID соревнования', fontsize=13)  
 plt.ylabel('Голы домашнего клуба', fontsize=13)  
 plt.title('Категоризированная диаграмма “box-and-whiskers”', fontsize=15)  
 plt.tight\_layout()  
 plt.savefig('categorized\_boxplot.png')  
 plt.show()  
  
 quant\_quant\_quail\_data = matches\_normalized[[  
 'home\_club\_goals', 'away\_club\_goals', 'competition\_id']]  
  
 # категоризированная диаграмма рассеивания  
 plt.figure(figsize=(12, 8))  
 sns.scatterplot(data=quant\_quant\_quail\_data, x='home\_club\_goals', y='away\_club\_goals', hue='competition\_id',  
 palette='viridis', s=100)  
 plt.xlabel('Голы домашнего клуба', fontsize=13)  
 plt.ylabel('Голы гостевого клуба', fontsize=13)  
 plt.title('Категоризированная диаграмма рассеивания', fontsize=15)  
 plt.legend(title='ID соревнования', fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('categorized\_scatterplot.png')  
 plt.show()  
  
 # категоризированная гистограмма: распределение количества матчей по сезонам  
 plt.figure(figsize=(12, 8))  
 sns.histplot(data=matches\_normalized, x='season', bins=10, kde=True)  
 plt.xlabel('Сезон', fontsize=12)  
 plt.ylabel('Частота', fontsize=12)  
 plt.title('Распределение количества матчей по сезонам', fontsize=14)  
 plt.xticks(rotation=45, fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('matches\_per\_season\_histogram.png')  
 plt.show()  
  
 # рассчитаем позиции клубов на основе количества побед  
 home\_club\_wins = (matches\_normalized.groupby('home\_club\_id')['home\_club\_goals'].  
 count().reset\_index(name='wins'))  
 home\_club\_wins['club\_position'] = home\_club\_wins['wins'].rank(  
 ascending=False, method='min')  
  
 # Объединим информацию о позициях клубов с данными о матчах  
 matches\_with\_positions = matches\_normalized.merge(home\_club\_wins[['home\_club\_id', 'club\_position']],  
 left\_on='home\_club\_id', right\_on='home\_club\_id')  
  
 # Построим диаграмму рассеивания  
 plt.figure(figsize=(12, 8))  
 sns.scatterplot(data=matches\_with\_positions, x='club\_position', y='home\_club\_goals', hue='season',  
 palette='viridis', s=100)  
 plt.xlabel('Позиция клуба', fontsize=13)  
 plt.ylabel('Голы домашней команды', fontsize=13)  
 plt.title(  
 'Зависимость голов домашней команды от позиции клуба', fontsize=15)  
 plt.xticks(fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('home\_goals\_vs\_club\_position\_scatterplot.png')  
 plt.show()  
  
 # Рассчитаем позиции клубов на основе количества побед  
 away\_club\_wins = matches\_normalized.groupby(  
 'away\_club\_id')['away\_club\_goals'].count().reset\_index(name='wins')  
 away\_club\_wins['club\_position'] = away\_club\_wins['wins'].rank(  
 ascending=False, method='min')  
  
 # Объединим информацию о позициях клубов с данными о матчах  
 matches\_with\_positions\_away = matches\_normalized.merge(away\_club\_wins[['away\_club\_id', 'club\_position']],  
 left\_on='away\_club\_id', right\_on='away\_club\_id')  
  
 # График зависимости количества голов гостевой команды от позиции в таблице  
 plt.figure(figsize=(12, 8))  
 sns.scatterplot(data=matches\_with\_positions\_away, x='club\_position', y='away\_club\_goals', hue='season',  
 palette='viridis', s=100)  
 plt.xlabel('Позиция в таблице', fontsize=13)  
 plt.ylabel('Голы гостевой команды', fontsize=13)  
 plt.title(  
 'Зависимость количества голов гостевой команды от позиции в таблице', fontsize=15)  
 plt.xticks(fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('away\_goals\_vs\_club\_position\_scatterplot.png')  
 plt.show()  
  
 # сгруппируем данные по типу формации и подсчитаем количество побед для каждой формации  
 formation\_wins = matches\_normalized.merge(clubs\_normalized[['club\_id', 'club\_formation']],  
 left\_on='home\_club\_id',  
 right\_on='club\_id')  
 formation\_wins = formation\_wins.groupby(  
 'club\_formation').size().reset\_index(name='wins')  
 formation\_wins = formation\_wins.sort\_values(by='wins', ascending=False)  
  
 # выберем топ-10 расстановок по победам  
 top\_10\_formations = formation\_wins.head(10)  
  
 colors = sns.color\_palette('Purples', n\_colors=10)[  
 ::-1] # Выбор 10 цветов из палитры Purples  
  
 # топ-10 расстановок по победам  
 plt.figure(figsize=(12, 8))  
 sns.barplot(x='club\_formation', y='wins', data=top\_10\_formations, palette=colors, hue='club\_formation',  
 legend=False)  
 plt.xlabel('Тип формации клуба', fontsize=12)  
 plt.ylabel('Количество побед', fontsize=12)  
 plt.title('Топ-10 формаций клубов по количеству побед', fontsize=14)  
 plt.xticks(rotation=45, ha='right', fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('top\_10\_formations\_wins\_bar plot\_seaborn.png')  
 plt.show()  
  
 # диаграмма рассеивания количества голов домашней команды от голов гостевой команды  
 plt.figure(figsize=(12, 8))  
 sns.scatterplot(data=matches\_normalized, x='home\_club\_goals', y='away\_club\_goals', hue='home\_club\_goals',  
 palette='viridis', s=100)  
 plt.xlabel('Голы домашней команды', fontsize=13)  
 plt.ylabel('Голы гостевой команды', fontsize=13)  
 plt.title(  
 'Зависимость количества голов домашней команды от голов гостевой команды', fontsize=15)  
 plt.xticks(fontsize=10)  
 plt.yticks(fontsize=10)  
 plt.tight\_layout()  
 plt.savefig('home\_goals\_vs\_away\_goals\_scatterplot\_seaborn.png')  
 plt.show()  
  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
  
generate\_graphical\_reports()  
  
  
def main():  
 clubs\_normalized = pd.read\_excel('../data/new\_normalized\_data.xlsx',  
 sheet\_name='clubs\_normalized')  
 matches\_normalized = pd.read\_excel('../data/new\_normalized\_data.xlsx',  
 sheet\_name='matches\_normalized')  
  
 if not clubs\_normalized.empty and not matches\_normalized.empty:  
 reports = generate\_text\_reports(clubs\_normalized, matches\_normalized)  
  
 with pd.ExcelWriter('reports.xlsx', engine='openpyxl') as writer:  
 for report\_name, report\_table in reports.items():  
 report\_table.to\_excel(writer, sheet\_name=report\_name)  
  
 print("Отчеты успешно записаны в файл Excel.")  
 else:  
 print("Не удалось загрузить данные для генерации отчетов.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()