**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

**Руководство разработчика по работе с приложением:  
«Анализ данных музыкального сервиса».**

Разработчики:

Сальник Максим Алексеевич

Муратов Айнур Фуатович

Кучерявая Элина Романовна

Руководитель:

Поляков К. Л.

2023 г.

## 

**Оглавление**

[Требования к характеристикам компьютера и операционной системе 2](#_Toc137425767)

[Версии интерпретатора и используемых библиотек 2](#_Toc137425768)

[Инструкция по запуску и настройке приложения 2](#_Toc137425769)

[Описание структуры БД 2](#_Toc137425770)

[Структура каталогов 2](#_Toc137425771)

[Архитектура приложения 3](#_Toc137425772)

[Листинг основного скрипта и всех модулей 4](#_Toc137425773)

## Требования к характеристикам компьютера и операционной системе

* Наличие на компьютере интерпретатора «Python» (вне зависимости от среды разработки);
* операционная система: Windows, macOS и Linux;
* процессор ×86 с тактовой частотой не менее 1 ГГц;
* оперативная память объемом, не менее 1 Гб.

## Версии интерпретатора и используемых библиотек

Интерпретатор – Python 3.7+.

Используемые библиотеки:

|  |  |
| --- | --- |
| **Библиотека** | **Версия** |
| pandas | 1.3.5 |
| matplotlib | 3.7.1 |
| tkinter | 0.09 |
| numpy | 1.21.6 |

## Инструкция по запуску и настройке приложения

Приложение запускается из командной строки: python main.py.

## Описание структуры БД

Все базы данных имеют поля:

Любимый исполнитель – строка

Число слушателей за месяц – натуральное число

Жанр любимого исполнителя – строка

Самый популярный трек любимого исполнителя – строка

Язык любимого исполнителя – строка

Идентификатор пользователя – натуральное число

Имя – строка

Фамилия – строка

Возраст – натуральное число

Любимый исполнитель – строка

Любимый жанр музыки – строка

## Структура каталогов

**Work <-** основной каталог.

**Data** — содержит базу данных.

**Graphics** — содержит копии графических отчетов.

**Library** — содержит библиотеку стандартных (универсальных) функций, разработанных бригадой, которые могут использоваться для создания других приложений, например функции чтения файлов.

**Notes** — содержит документацию, в нем размещается Руководства пользователя и разработчика.

**Output** — содержит копии текстовых отчетов.

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

## Архитектура приложения

|  |  |  |
| --- | --- | --- |
| **Модуль** | **Местонахождение** | **Функция и ее пояснение** |
| main.py | Work/Scripts | Основной файл приложения |
| describing.py | Work/Scripts | Специализированный модуль. Библиотека функции создания текстового отчёта в виде базы данных, содержащей сведения о возрасте людей |
| genre\_match.py | Work/Scripts | Специализированный модуль. Библиотека функции создания текстового отчёта в виде базы данных, содержащей информацию о людях, чей любимый жанр совпадает с жанром артиста, а также рекомендуемую для них песню |
| language\_sing.py | Work/Scripts | Специализированный модуль. Библиотека функции создания текстового отчёта в виде базы данных, содержащей сведения об исполнителях и их песнях, в данном жанре и на данном языке |
| made\_table\_whole.py | Work/Library | Библиотека функции вывода справочника в окно приложения |
| piv\_t.py | Work/Scripts | Специализированный модуль. Библиотека функции создания текстового отчёта в виде базы данных, представляющей собой сводную таблицу, в которой видна зависимость возраста слушателя от языка и жанра музыки |
| read\_table.py | Work/Library | Библиотека функции вывода справочника в окно приложения |
| save\_file.py | Work/Library | Библиотека функции сохранения файла отчёта или справочника на компьютер |
| singer\_match.py | Work/Scripts | Специализированный модуль. Библиотека функции создания текстового отчёта в виде базы данных, содержащей информацию о людях, слушающих данного исполнителя |

## Листинг основного скрипта и всех модулей

"""

Cамостоятельная работа по курсу —

«Проектный семинар «Python в науке о данных»

Авторы:

Муратов Айнур

Кучерявая Элина

Сальник Максим

"""

from tkinter import ttk

from tkinter import messagebox

from tkinter.messagebox import askyesno

from tkinter.ttk import Combobox

import os

import sys

import tkinter as tk

import pandas as pd

import matplotlib.pyplot as plt

os.chdir("D:\work")

sys.path.append("D:\work")

from library.read\_table import make\_table

from library.save\_file import save\_file

from library.genre\_match import genre\_matches

from library.singer\_match import singers\_listeners

from library.language\_sing import language\_singer

from library.describing import text\_describe

from library.made\_table\_whole import make\_table\_wholetable

from library.piv\_t import piv\_table

BACKGR = '#745e8a'

def change\_color(color):

"""

Изменение фонового цвета

Args:

color (string): Цвет нового фона

Автор: Сальник Максим

"""

result = askyesno(title='Подтверждение операции',

message='Для выполнения операции'+

' необходимо закрыть все окна и'+

' несохраненные изменения пропадут.\nПродолжить?')

if result:

for widget in root.winfo\_children():

if isinstance(widget, tk.Toplevel):

widget.destroy()

button1.config(state='normal')

button2.config(state='normal')

button3.config(state='normal')

root["bg"] = color

global BACKGR

BACKGR = color

style.configure("Custom.TLabel", background=color, foreground="black")

def update\_data():

"""

Функция обновления справочников

Автор: Муратов Айнур

"""

global TABLE\_ONE

TABLE\_ONE = pd.read\_csv('./data/Tab1.csv', delimiter=';', encoding='cp1251')

global TABLE\_TWO

TABLE\_TWO = pd.read\_csv('./data/Tab2.csv', delimiter=';', encoding='cp1251')

global ITOG

ITOG = pd.merge(TABLE\_ONE, TABLE\_TWO, on="Любимый исполнитель")

#Прогрузка справочников

update\_data()

def add\_entity():

"""

Функция добавления сущности в справочник

Автор: Муратов Айнур

"""

add\_window = tk.Toplevel(root)

add\_window.title("Добавить сущность")

labels = ["ID пользователя:", "Имя:", "Фамилия:",

"Возраст:", "Любимый исполнитель:", "Любимый жанр музыки:",

"Число слушателей за месяц:", "Жанр любимого исполнителя:",

"Самый популярный трек:", "Язык исполнителя:"]

entries = []

for i, label in enumerate(labels):

lbl = tk.Label(add\_window, text=label)

lbl.grid(row=i, column=0, padx=10, pady=5)

entry = tk.Entry(add\_window)

entry.grid(row=i, column=1, padx=10, pady=5)

entries.append(entry)

def save\_entity():

"""

Функция сохранения новых данных

Автор: Муратов Айнур

"""

new\_entity = {}

for i, entry in enumerate(entries):

new\_entity[labels[i]] = entry.get()

table1\_path = './data/Tab1.csv'

table2\_path = './data/Tab2.csv'

# Сохранение данных в Table1

table1\_data = [new\_entity[label] for label in labels[:6]]

table1 = pd.DataFrame([table1\_data], columns=labels[:6])

file\_exists = os.path.isfile(table1\_path)

table1.to\_csv(table1\_path, index=False, mode='a',

header=not file\_exists, sep=';', encoding='cp1251')

# Проверка наличия Любимого исполнителя в Table2.csv

favorite\_artist = new\_entity["Любимый исполнитель:"]

table2\_exists = os.path.isfile(table2\_path)

if table2\_exists:

table2 = pd.read\_csv(table2\_path, delimiter=';', encoding='cp1251')

if not table2['Любимый исполнитель'].str.contains(favorite\_artist).any():

table2\_data = [new\_entity[label] for label in (labels[4:5] + labels[6:10])]

table2\_data = pd.DataFrame([table2\_data],

columns=(labels[4:5] + labels[6:10]))

table2\_data.to\_csv(table2\_path, index=False,

mode='a', header=False, sep=';',

encoding='cp1251')

else:

table2\_data = pd.DataFrame(

[new\_entity[label] for label in (labels[4:5] + labels[6:10])],

columns=(labels[4:5] + labels[6:10])

)

table2\_data.to\_csv(table2\_path, index=False, sep=';', encoding='cp1251')

update\_data()

add\_window.destroy()

messagebox.showinfo("Отлично", "Сущность успешно добавлена!")

save\_button = tk.Button(add\_window, text="Сохранить", command=save\_entity)

save\_button.grid(row=len(labels), columnspan=2, padx=10, pady=5)

# Функция-обработчик для кнопки со справочником с пользователями

def show\_table\_one(bgr):

"""

Функция показа окна с первым справочниклм

Args:

bgr (string): Цвет фона

Автор: Муратов Айнур

"""

# Блокировка кнопки повторного вызова таблицы

button1.config(state='disabled')

# Ф-я разблокирования кнопки и закртыия окна

def on\_button():

"""

Функция разблокирования кнопки после закрытия окна

Автор: Муратов Айнур

"""

window.destroy()

button1.config(state='')

window = tk.Toplevel(root)

window.title("Справочник с пользователями")

window.configure(bg=BACKGR)

window.minsize(800, 520)

window.focus()

colum\_names = ["ID пользователя", "Имя", "Фамилия",

"Возраст", "Любимый исполнитель", "Любимый жанр музыки"]

menubar = tk.Menu(window)

# Создание меню "Файл"

file\_menu = tk.Menu(menubar, tearoff=0)

file\_menu.add\_command(label="Сохранить", command = lambda:save\_file(TABLE\_ONE))

file\_menu.add\_separator()

file\_menu.add\_command(label="Выход", command=on\_button)

menubar.add\_cascade(label="Файл", menu=file\_menu)

window.config(menu=menubar)

top = tk.LabelFrame(window, text="Справочник с пользователями", bg=bgr)

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

make\_table(TABLE\_ONE, top, colum\_names)

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

def show\_itog(bgr):

"""

Функция показа окна с общим справочником

Args:

bgr (string): Цвет фона

Автор: Муратов Айнур

"""

# Блокировка кнопки повторного вызова таблицы

button2.config(state='disabled')

# Ф-я разблокирования кнопки и закртыия окна

def on\_button():

"""

Функция разблокирования кнопки после закрытия окна

Автор: Муратов Айнур

"""

window.destroy()

button2.config(state='normal')

window = tk.Toplevel(root)

window.title("Общий справочник")

window.focus()

window.minsize(800, 520)

window.configure(bg=BACKGR)

colum\_names = ["ID пользователя", "Имя", "Фамилия", "Возраст",

"Любимый исполнитель", "Любимый жанр музыки",

"Число слушателей за месяц", "Жанр любимого исполнителя",

"Самый популярный трек", "Язык исполнителя"]

menubar = tk.Menu(window)

# Создание меню Файл

file\_menu = tk.Menu(menubar, tearoff=0)

file\_menu.add\_command(label="Добавить сущность", command= add\_entity)

file\_menu.add\_command(label="Сохранить", command = lambda:save\_file(ITOG))

file\_menu.add\_separator()

file\_menu.add\_command(label="Выход", command=on\_button)

# Создание меню Графические отчеты

graph\_reports\_menu = tk.Menu(menubar, tearoff=0)

graph\_reports\_menu.add\_command(label="Гистограмма жанровых совпадений",

command= lambda: create\_genre\_histogram(menubar))

graph\_reports\_menu.add\_command(label="Популярность артистов",

command= lambda: create\_popularity\_chart(menubar))

graph\_reports\_menu.add\_command(label="Диаграмма Бокс-Вискера",

command= lambda: create\_boxplot(menubar))

graph\_reports\_menu.add\_command(label="Диаграмма рассеивания",

command= lambda: create\_scatterplot(menubar))

graph\_reports\_menu.add\_command(label='Кластерная гистограмма',

command= lambda: create\_clastar(menubar))

# Создание меню Сводная таблица

pivot\_table\_menu = tk.Menu(menubar, tearoff=0)

pivot\_table\_menu.add\_command(label="Генерировать сводную таблицу возрастов",

command= lambda: text\_pivot(menubar, BACKGR))

# Создание меню смены цыета фона

color\_menu = tk.Menu(menubar, tearoff=0)

color\_menu.add\_command(label="По умолчанию", command= lambda: change\_color('#745e8a'))

color\_menu.add\_command(label="Зеленый", command= lambda: change\_color('#598a34') )

color\_menu.add\_command(label="Синий", command= lambda:change\_color('#4b5c99') )

text\_reports\_menu = tk.Menu(menubar, tearoff=0)

menubar.add\_cascade(label="Файл", menu=file\_menu)

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

menubar.add\_cascade(label="Графические отчеты", menu=graph\_reports\_menu)

menubar.add\_cascade(label="Сводная таблица", menu=pivot\_table\_menu)

menubar.add\_cascade(label="Смена фона", menu= color\_menu )

# Создание меню Текстовые отчеты

text\_reports\_menu.add\_command(label="Возрастные сведения",

command = lambda: text\_describer(menubar, BACKGR))

text\_reports\_menu.add\_command(label="Соответствие жанров",

command= lambda: text\_genre(menubar, BACKGR))

text\_reports\_menu.add\_command(label="Поиск по языку и жанру",

command= lambda: text\_lang(menubar, BACKGR))

text\_reports\_menu.add\_command(label="Посик слушателей нужного исполнителя",

command= lambda: text\_singer(menubar, BACKGR))

window.config(menu=menubar)

top = tk.LabelFrame(window, text="Общий справочник", bg=bgr)

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

make\_table(ITOG, top, colum\_names)

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

window.mainloop()

def show\_table\_two(bgr):

"""

Функция показа окна со вторым справочником

Args:

bgr (string): Цвет фона

Автор: Муратов Айнур

"""

# Блокировка кнопки повторного вызова таблицы

button3.config(state='disabled')

# Ф-я разблокирования кнопки и закртыия окна

def on\_button():

"""

Функция разблокирования кнопки после закрытия окна

Автор: Муратов Айнур

"""

window.destroy()

button3.config(state='')

window = tk.Toplevel(root)

window.title("Справочник с исполнителями")

window.minsize(800, 520)

window.focus()

window.configure(bg=BACKGR)

colum\_names = ["Любимый исполнитель", "Число слушателей за месяц",

"Жанр любимого исполнителя", "Самый популярный трек",

"Язык исполнителя"]

menubar = tk.Menu(window)

# Создание меню "Файл"

file\_menu = tk.Menu(menubar, tearoff=0)

file\_menu.add\_command(label="Сохранить", command = lambda:save\_file(TABLE\_TWO))

file\_menu.add\_separator()

file\_menu.add\_command(label="Выход", command=on\_button)

menubar.add\_cascade(label="Файл", menu=file\_menu)

window.config(menu=menubar)

top = tk.LabelFrame(window, text="Справочник с исполнителями", bg=bgr)

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

make\_table(TABLE\_TWO, top, colum\_names)

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

def text\_describer(men, bgr):

"""

Функция составяления текстового отчёта в виде возрастной статистике

Args:

men (tk.Menu): меню, из которого вызвалась функция

bgr (string): цвет фона

Автор: Сальник Максим

"""

def generate\_second():

"""

Функция формирования отчёта

Автор: Сальник Максим

"""

for widgets in frame\_rep.winfo\_children():

widgets.destroy()

txt = text\_describe(ITOG, entry.get())

if entry.get() == 'Жанрам':

colum\_names = ['Количество людей', 'Средний возраст',

'Минимальный возраст','Медиана возраста',

'Максимальный возраст']

index\_names = txt.index

make\_table\_wholetable(txt, frame\_rep, colum\_names, index\_names)

elif entry.get() == 'Всей таблице':

colum\_names = ["Значение"]

index\_names = ['Количество людей', 'Средний возраст',

'Минимальный возраст','Медиана возраста',

'Максимальный возраст']

make\_table\_wholetable(txt, frame\_rep, colum\_names, index\_names)

def on\_button():

"""

Функция закрытия окна и разблокировки кнопки

Автор: Сальник Максим

"""

window.destroy()

men.entryconfig("Текстовые отчеты", state='normal')

window = tk.Toplevel(root)

window.configure(bg=bgr)

window.minsize(800, 520)

window.title("Возрастные сведения")

frame\_rep = tk.Frame(window)

frame\_rep.pack(side='left')

frame\_panel = tk.Frame(window, background=bgr)

frame\_panel.pack(expand=1)

lbl = tk.Label(frame\_panel, text='Возрастная статистика по:', bg= bgr, font=("Times", 20))

lbl.pack(padx=10, pady=10)

entry = Combobox(frame\_panel, state='readonly', font=("Times", 20))

entry['values'] = ('Всей таблице', 'Жанрам')

entry.current(0)

entry.pack(padx=10, pady=10)

button\_create = ttk.Button(frame\_panel, text="Сгенерировать",

style="Custom.TButton", command= generate\_second)

button\_create.pack(padx=10, pady=10)

button\_save = ttk.Button(frame\_panel, text="Сохранить",

style="Custom.TButton",

command= lambda: save\_file(text\_describe(ITOG, entry.get())))

button\_save.pack(padx=10, pady=10)

men.entryconfig("Текстовые отчеты", state='disabled')

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

# Функция текстового отчёта для сопоставления жанра

def text\_genre(men, bgr):

"""

Функция составяления текстового отчёта о пользователе и рекомендованной для него песне,

если его любимый жанр совпал с жанром исполнителя

Args:

men (tk.Menu): меню, из которого вызвалась функция

bgr (string): цвет фона

Автор: Сальник Максим

"""

def on\_button():

"""

Функция закрытия окна и разблокировки кнопки

Автор: Сальник Максим

"""

window.destroy()

men.entryconfig("Текстовые отчеты", state='normal')

def generate\_second():

"""

Функция формирования отчёта

Автор: Сальник Максим

"""

for widgets in frame\_rep.winfo\_children():

widgets.destroy()

txt = genre\_matches(ITOG, entry.get())

if txt.empty:

lbl = tk.Label(frame\_rep, text='Совпадений нет', font=("Times", 25))

lbl.pack(padx=10, pady=10)

else:

make\_table(txt, frame\_rep, colum\_names)

colum\_names = ["Идентификатор пользователя","Имя","Фамилия",

"Самый популярный трек любимого исполнителя"]

window = tk.Toplevel(root)

window.minsize(800, 520)

window.title("Соответствие жанров")

window.configure(bg=bgr)

frame\_rep = tk.Frame(window)

frame\_rep.pack(side='left')

#make\_table(TABLE\_TWO, frame\_rep)

frame\_panel = tk.Frame(window, bg=bgr)

frame\_panel.pack(expand=1)

lbl = tk.Label(frame\_panel, text='Совпадения по жанру:', bg= bgr, font=("Times", 20))

lbl.pack(padx=10, pady=10)

entry = Combobox(frame\_panel, values = list(TABLE\_ONE['Любимый жанр музыки'].unique())[:-1],

state='readonly', font=("Times", 20))

entry['values'] = (list(TABLE\_ONE['Любимый жанр музыки'].unique())[:-1])

entry.current(0)

entry.pack(padx=10, pady=10)

button\_create = ttk.Button(frame\_panel, text="Сгенерировать",

style="Custom.TButton", command= generate\_second)

button\_create.pack(padx=10, pady=10)

button\_save = ttk.Button(frame\_panel, text="Сохранить",

style="Custom.TButton",

command= lambda: save\_file(genre\_matches(ITOG, entry.get())))

button\_save.pack(padx=10, pady=10)

men.entryconfig("Текстовые отчеты", state='disabled')

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

# Функция текстового отчёта для поиска слушателей исполнителя

def text\_singer(men, bgr):

"""

Функция составяления текстового отчёта о пользователях и их любимом жанре музыки,

если они слушают данного исполнителя

Args:

men (tk.Menu): меню, из которого вызвалась функция

bgr (string): цвет фона

Автор: Сальник Максим

"""

def on\_button():

"""

Функция закрытия окна и разблокировки кнопки

Автор: Сальник Максим

"""

window.destroy()

men.entryconfig("Текстовые отчеты", state='normal')

def generate\_second():

"""

Функция формирования отчёта

Автор: Сальник Максим

"""

for widgets in frame\_rep.winfo\_children():

widgets.destroy()

txt = singers\_listeners(ITOG, entry.get())

if txt.empty:

lbl = tk.Label(frame\_rep, text='Совпадений нет', font=("Times", 25))

lbl.pack(padx=10, pady=10)

else:

make\_table(txt, frame\_rep, colum\_names)

colum\_names = ["Идентификатор пользователя","Имя","Фамилия", "Любимый жанр музыки"]

window = tk.Toplevel(root)

window.minsize(800, 520)

window.title("Поиск слушателей нужного исполнителя")

window.configure(bg=bgr)

frame\_rep = tk.Frame(window)

frame\_rep.pack(side='left')

frame\_panel = tk.Frame(window, bg=bgr)

frame\_panel.pack(expand=1)

lbl = tk.Label(frame\_panel, text='Введите исполнителя:', bg= bgr, font=("Times", 20))

lbl.pack(padx=10, pady=10)

entry = Combobox(frame\_panel, values = list(TABLE\_TWO['Любимый исполнитель'].unique())[:-1],

state='readonly', font=("Times", 20))

entry['values']=(list(TABLE\_TWO['Любимый исполнитель'].unique())[:-1])

entry.current(0)

entry.pack(padx=10, pady=10)

button\_create = ttk.Button(frame\_panel, text="Сгенерировать", style="Custom.TButton",

command= generate\_second)

button\_create.pack(padx=10, pady=10)

button\_save = ttk.Button(frame\_panel, text="Сохранить", style="Custom.TButton",

command= lambda: save\_file(singers\_listeners(ITOG, entry.get())))

button\_save.pack(padx=10, pady=10)

men.entryconfig("Текстовые отчеты", state='disabled')

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

# Функция текстового отчёта для поиска певца по языку

def text\_lang(men, bgr):

"""

Функция составяления текстового отчёта об исполнителе,

если он делает музыку в данном жанре и на данном языке

Args:

men (tk.Menu): меню, из которого вызвалась функция

bgr (string): цвет фона

Автор: Сальник Максим

"""

def on\_button():

"""

Функция закрытия окна и разблокировки кнопки

Автор: Сальник Максим

"""

window.destroy()

men.entryconfig("Текстовые отчеты", state='normal')

def generate\_second():

"""

Функция формирования отчёта

Автор: Сальник Максим

"""

for widgets in frame\_rep.winfo\_children():

widgets.destroy()

txt = language\_singer(TABLE\_TWO, entry.get(), entry2.get())

if txt.empty:

lbl = tk.Label(frame\_rep, text='Совпадений нет')

lbl.pack(padx=10, pady=10)

else:

make\_table(txt, frame\_rep, colum\_names)

colum\_names = ["Исполнитель", "Самый популярный трек"]

window = tk.Toplevel(root)

window.minsize(800, 520)

window.title("Поиск по языку и жанру")

window.configure(bg=bgr)

frame\_rep = tk.Frame(window)

frame\_rep.pack(side='left')

frame\_panel = tk.Frame(window, bg=bgr)

frame\_panel.pack(expand=1)

lbl = tk.Label(frame\_panel, text='Введите жанр исполнителя:', bg= bgr, font=("Times", 20))

lbl.pack(padx=10, pady=10)

entry = Combobox(frame\_panel, values = list(TABLE\_ONE['Любимый жанр музыки'].unique())[:-1],

state='readonly', font=("Times", 20))

entry['values']= (list(TABLE\_ONE['Любимый жанр музыки'].unique())[:-1])

entry.current(0)

entry.pack(padx=10, pady=10)

lbl2 = tk.Label(frame\_panel, text='Введите язык исполнителя:', bg= bgr, font=("Times", 20))

lbl2.pack(padx=10, pady=10)

entry2 = Combobox(frame\_panel,

values = list(TABLE\_TWO['Язык любимого исполнителя'].unique())[:-1],

state='readonly', font=("Times", 20))

entry2['values']=(list(TABLE\_TWO['Язык любимого исполнителя'].unique())[:-1])

entry2.current(0)

entry2.pack(padx=10, pady=10)

button\_create = ttk.Button(frame\_panel, text="Сгенерировать", style="Custom.TButton",

command= generate\_second)

button\_create.pack(padx=10, pady=10)

button\_save = ttk.Button(frame\_panel, text="Сохранить", style="Custom.TButton",

command= lambda: save\_file(

language\_singer(TABLE\_TWO, entry.get(), entry2.get()))

)

button\_save.pack(padx=10, pady=10)

men.entryconfig("Текстовые отчеты", state='disabled')

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

def text\_pivot(men, bgr):

"""

Функция составяления текстового отчёта в виде сводной таблице возрастов

слушателей для рассмотрения зависимости от жанра и языка музыки

Args:

men (tk.Menu): меню, из которого вызвалась функция

bgr (string): цвет фона

Автор: Сальник Максим

"""

def on\_button():

"""

Функция закрытия окна и разблокировки кнопки

Автор: Сальник Максим

"""

window.destroy()

men.entryconfig("Сводная таблица", state='normal')

window = tk.Toplevel(root)

window.minsize(800, 520)

window.title("Сводная таблица возрастов")

window.configure(bg=bgr)

window.columnconfigure(0, weight=1)

frame\_rep = tk.Frame(window)

frame\_rep.grid(row=0, column=0, sticky='nsew')

frame\_panel = tk.Frame(window, bg =bgr)

frame\_panel.grid(row=1, column=0, sticky='nsew')

button\_save = ttk.Button(frame\_panel, text="Сохранить", style="Custom.TButton",

command= lambda: save\_file(piv\_table(ITOG)))

button\_save.pack(padx=10, pady=10)

piv = piv\_table(ITOG)

colum\_names = piv.columns

ind\_names = piv.index

make\_table\_wholetable(piv, frame\_rep, colum\_names, ind\_names)

men.entryconfig("Сводная таблица", state='disabled')

# Создание протокла действий при попытке пользователя закрыть окно

window.protocol("WM\_DELETE\_WINDOW", on\_button)

def create\_genre\_histogram(men):

"""

Функция составления графического отчёта в виде гистограммы, показывающей,

как много пользователей слушают преимущественно один жанр.

Args:

men (tk.Menu): меню, из которого вызвалась функцияа

Автор: Кучерявая Элина

"""

men.entryconfig("Графические отчеты", state='disabled')

# Загрузка данных из таблицы 1 и таблицы 2

table1 = TABLE\_ONE

table2 = TABLE\_TWO

# Создание словаря для хранения количества соответствующих пользователей

genre\_counts = {}

# Подсчет количества соответствующих пользователей для каждой категории и столбца

for index, row in table2.iterrows():

genre = row['Жанр любимого исполнителя']

favorite\_artist = row['Любимый исполнитель']

matching\_users = table1[

(table1['Любимый жанр музыки'] == genre) & (table1['Любимый исполнитель'] == favorite\_artist)]

count = len(matching\_users)

if genre in genre\_counts:

genre\_counts[genre].append(count)

else:

genre\_counts[genre] = [count]

# Создание списков для категорий и соответствующих столбцов

categories = []

columns = []

# Суммирование количества пользователей в каждой категории

for genre, counts in genre\_counts.items():

categories.append(genre)

columns.append(sum(counts))

# Построение гистограммы

plt.bar(categories, columns)

plt.xlabel('Категория жанра')

plt.ylabel('Количество пользователей')

plt.title('Гистограмма жанров')

plt.xticks(rotation=90)

plt.show()

try:

men.entryconfig("Графические отчеты", state='normal')

except:

pass

def create\_popularity\_chart(men):

"""

Функция составления графического отчёта, показывающего количество людей,

слушающих конкретных исполнителей

Args:

men (tk.Menu): меню, из которого вызвалась функцияа

Автор: Кучерявая Элина

"""

men.entryconfig("Графические отчеты", state='disabled')

# Считываем данные из CSV-файла в DataFrame

data = TABLE\_ONE

# Группируем данные по исполнителю и подсчитываем количество пользователей

artist\_counts = data['Любимый исполнитель'].value\_counts()

# Получаем список исполнителей и соответствующие им количество слушателей

artists = artist\_counts.index

listeners = artist\_counts.values

# Создаем фигуру и оси для диаграммы

fig, ax = plt.subplots()

# Строим столбчатую диаграмму

ax.bar(artists, listeners)

# Настраиваем подписи осей и заголовок диаграммы

ax.set\_xlabel('Исполнитель')

ax.set\_ylabel('Количество слушателей')

ax.set\_title('Популярность исполнителей')

# Поворачиваем подписи на оси X для лучшей читаемости

plt.xticks(rotation=90)

# Отображаем диаграмму

plt.show()

try:

men.entryconfig("Графические отчеты", state='normal')

except:

pass

def create\_boxplot(men):

"""

Функция составления графического отчёта в виде Бокса-Вискера.

Args:

men (tk.Menu): меню, из которого вызвалась функцияа

Автор: Кучерявая Элина

"""

men.entryconfig("Графические отчеты", state='disabled')

table1 = TABLE\_ONE

table2 = TABLE\_TWO

# Объединение таблиц по общему столбцу 'Любимый исполнитель'

ITOG = pd.merge(table1, table2, on='Любимый исполнитель')

ITOG['Число слушателей за месяц'] = ITOG['Число слушателей за месяц'].map(lambda x: x.replace(" ", ""))

data\_types\_dict = {'Число слушателей за месяц': int}

ITOG = ITOG.astype(data\_types\_dict)

# Объединение таблиц по общему столбцу 'Любимый исполнитель'

merged\_data = ITOG

# Категоризация данных по столбцу 'Любимый жанр музыки'

categories = merged\_data['Любимый жанр музыки'].unique()

data\_by\_category = []

for category in categories:

data\_by\_category.append(merged\_data.loc[merged\_data['Любимый жанр музыки'] == category, 'Число слушателей за месяц'])

# Построение категоризированной диаграммы Бокса-Вискера

plt.boxplot(data\_by\_category, labels=categories)

plt.xlabel('Любимый жанр музыки')

plt.ylabel('Число слушателей за месяц')

plt.title('Категоризированная диаграмма Бокса-Вискера')

plt.xticks(rotation=90)

# Отображение диаграммы

plt.show()

try:

men.entryconfig("Графические отчеты", state='normal')

except:

pass

def create\_scatterplot(men):

"""

Функция составления графического отчёта в виде Диаграммы рассеивания.

Args:

men (tk.Menu): меню, из которого вызвалась функцияа

Автор: Кучерявая Элина

"""

men.entryconfig("Графические отчеты", state='disabled')

# Загрузка данных из таблицы 1 и таблицы 2

table1 = TABLE\_ONE

table2 = TABLE\_TWO

# Атрибуты для диаграммы рассеивания

x\_attribute = 'Возраст' # Количественный атрибут 1

y\_attribute = 'Число слушателей за месяц' # Количественный атрибут 2

category\_attribute = 'Любимый жанр музыки' # Качественный атрибут

# Объединение таблиц по общему столбцу 'Любимый исполнитель'

merged\_data = pd.merge(table1, table2, on='Любимый исполнитель')

# Сортировка данных по возрастанию значения столбца 'Число слушателей за месяц'

merged\_data\_sorted = merged\_data.sort\_values(by=y\_attribute)

# Получение уникальных значений категорий

categories = merged\_data\_sorted[category\_attribute].unique()

# Создание цветовой палитры для категорий

color\_palette = plt.cm.get\_cmap('tab10', len(categories))

# Построение категоризированной диаграммы рассеивания

for i, category in enumerate(categories):

category\_data = merged\_data\_sorted[merged\_data\_sorted[category\_attribute] == category]

x = category\_data[x\_attribute]

y = category\_data[y\_attribute]

plt.scatter(x, y, color=color\_palette(i), label=category)

# Настройка подписей осей и легенды

plt.xlabel(x\_attribute)

plt.ylabel(y\_attribute)

plt.title('Категоризированная диаграмма рассеивания')

plt.legend()

# Отображение диаграммы рассеивания

plt.show()

try:

men.entryconfig("Графические отчеты", state='normal')

except:

pass

def create\_clastar(men):

"""

Функция составления графического отчёта в виде Кластерной гистограммы.

Args:

men (tk.Menu): меню, из которого вызвалась функцияа

Автор: Кучерявая Элина

"""

men.entryconfig("Графические отчеты", state='disabled')

# Объединение таблиц по общему столбцу 'Любимый исполнитель'

merged\_data = pd.merge(TABLE\_ONE, TABLE\_TWO, on='Любимый исполнитель')

# Группировка данных по категориям 'Жанр любимого исполнителя' и 'Язык любимого исполнителя'

grouped\_data = merged\_data.groupby(['Жанр любимого исполнителя', 'Язык любимого исполнителя']).size().unstack().fillna(0)

# Построение кластеризованной столбчатой диаграммы

grouped\_data.plot(kind='bar', stacked=True)

# Настройка подписей осей и заголовка

plt.xlabel('Жанр любимого исполнителя')

plt.ylabel('Количество исполнителей')

plt.title('Кластеризованная столбчатая диаграмма')

# Отображение диаграммы

plt.show()

try:

men.entryconfig("Графические отчеты", state='normal')

except:

pass

# Создание главного окна

root = tk.Tk()

root.title("Мое приложение")

root.configure(bg=BACKGR) # Задний фон окна

root.minsize(800, 520)

def closing():

"""

Функция закрытия графических отчётов вместе с закрытием главного окна

Автор: Кучерявая Элина

"""

plt.close()

root.destroy()

# Создание кнопки "Закрыть"

close\_button = ttk.Button(root, text="Закрыть", style="Custom.TButton", command=closing)

close\_button.place(relx=0.5, rely=0.7, anchor="center")

# Создание стиля для метки

style = ttk.Style()

style.configure("Custom.TLabel", background=BACKGR, foreground="black")

# Создание метки с текстом "Выберите справочник"

label = ttk.Label(root, text="Выберите справочник", font=("Times", 30), style="Custom.TLabel")

label.pack(pady=50)

# Создание кнопок

button1 = ttk.Button(root, text="Справочник с пользователями", style="Custom.TButton",

command=lambda: show\_table\_one(BACKGR))

button2 = ttk.Button(root, text="Общий справочник", style="Custom.TButton",

command=lambda: show\_itog(BACKGR))

button3 = ttk.Button(root, text="Справочник с исполнителями", style="Custom.TButton",

command=lambda: show\_table\_two(BACKGR))

# Расстановка кнопок

button2.place(relx=0.5, rely=0.3, anchor="center")

button1.place(relx=0.3, rely=0.5, anchor="center")

button3.place(relx=0.7, rely=0.5, anchor="center")

# Создание стиля для кнопок

style.configure("Custom.TButton", background=BACKGR, padding=(10, 20), font=("Times", 14))

root.protocol("WM\_DELETE\_WINDOW", closing)

root.mainloop()

def text\_describe(ITOG, how):

"""

Функция создания текстого отчёта в виде базы данных,

содержащей сведения о возрасте людей

Args:

ITOG (pd.DataFrame): общий справочник

how (string): указатель на то, по каким данным делать отчёт

Returns:

pd.DataFrame: итоговый отчёт

Автор: Сальник Максим

"""

if how == 'Жанрам':

DES = ITOG.groupby(["Любимый жанр музыки"]).describe()

DES = DES.loc[:,"Возраст"]

DES = DES.drop(columns = ['std', '25%', '75%'])

DES.columns = ['Количество людей', 'Средний возраст', 'Минимальный возраст','Медиана возраста', 'Максимальный возраст']

elif how == 'Всей таблице':

DES = ITOG[["Возраст"]].describe()

DES = DES.drop(labels = ['std', '25%', '75%'])

DES.index = ['Количество людей', 'Средний возраст', 'Минимальный возраст','Медиана возраста', 'Максимальный возраст']

return DES

def genre\_matches (ITOG, genre): #Пользователи, чей любимый жанр совпадает с жанром артиста. Им можно предложить в рекомендациях самый популярный трек этого исполнителя

"""

Функция создания текстового отчёта в виде базы данных,

содержащей информацию о людях, чей любимый жанр совпадает с жанром артиста,

а также рекомендуемуе для них песню

Args:

ITOG (pd.DataFrame): общий справочник

genre (string): жанр, по которому следует искать совпадения

Returns:

pd.DataFrame: итоговый отчёт

Автор: Сальник Максим

"""

SEL = (ITOG["Жанр любимого исполнителя"] == genre) & (ITOG["Жанр любимого исполнителя"] == genre)

W1 = ITOG.loc[SEL, ["Идентификатор пользователя","Имя","Фамилия", "Самый популярный трек любимого исполнителя"]]

return W1

def language\_singer (ITOG, genre, language): #Получаем исполнителей и их любимый трек

"""

Функция создания текстого отчёта в виде базы данных,

содержащей сведения об исполнителях и их песнях,

в данном жанре и на данном языке

Args:

ITOG (pd.DataFrame): общий справочник

genre (string): жанр, по которому следует искать совпадения

language (string): язык, по которому следует искать совпадения

Returns:

pd.DataFrame: итоговый отчёт

Автор: Кучерявая Элина

"""

SEL = (ITOG["Жанр любимого исполнителя"] == genre) & (ITOG["Язык любимого исполнителя"] == language)

W1 = ITOG.loc[SEL, ["Любимый исполнитель", "Самый популярный трек любимого исполнителя"]]

return W1

def make\_table\_wholetable(name, master, column\_names, index\_names):

"""

Функция вывода справочника в окно приложения

Args:

name (pd.DataFrame): справочник, который нужно вывести

master (Tk): окно или фрейм, в который выводится справочник

column\_names (list): список с названиями столбцов

index\_names (list): список с названиями строк

Автор: Сальник Максим

"""

name = name.copy() #Копирование, чтобы избежать изменение исходных данных

name.columns = column\_names

name.index = index\_names

height = name.shape[0]

width = name.shape[1]

# Создание виджета Canvas

canvas = tk.Canvas(master)

canvas.pack(side="left", fill="both", expand=True)

# Создание Scrollbar по вертикали

y\_scrollbar = ttk.Scrollbar(master, orient="vertical", command=canvas.yview)

y\_scrollbar.pack(side="right", fill="y")

canvas.configure(yscrollcommand=y\_scrollbar.set)

# Создание Scrollbar по горизонтали

x\_scrollbar = ttk.Scrollbar(master, orient="horizontal", command=canvas.xview)

x\_scrollbar.pack(side="bottom", fill="x")

canvas.configure(xscrollcommand=x\_scrollbar.set)

# Создание вложенного фрейма для размещения таблицы

table\_frame = ttk.Frame(canvas)

canvas.create\_window((0, 0), window=table\_frame, anchor="nw")

# Добавление заголовков столбцов

for j, col\_name in enumerate(column\_names):

label = ttk.Label(table\_frame, text=col\_name)

label.grid(row=0, column=j+1, sticky="nsew")

# Добавление заголовков строк

for i, ind\_name in enumerate(index\_names):

label = ttk.Label(table\_frame, text=ind\_name)

label.grid(column=0, row=i+1, sticky="nsew")

# Создание и заполнение таблицы Entry

entry\_widgets = []

for i in range(height):

row\_widgets = []

for j in range(width):

entry = ttk.Entry(table\_frame)

entry.grid(row=i+1, column=j+1, sticky="nsew")

value = name.iloc[i, j]

entry.insert(0, value)

row\_widgets.append(entry)

entry\_widgets.append(row\_widgets)

# Растягивание ячеек таблицы при изменении размеров окна

for i in range(height + 1):

table\_frame.rowconfigure(i, weight=1)

for j in range(width):

table\_frame.columnconfigure(j, weight=1)

canvas.bind("<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all")))

def piv\_table(ITOG):

"""

Функция создания текстого отчёта в виде базы данных,

представляющей собой сводную таблицу, в которой

видна зависимость возраста слушателя от языка и жанра музыки

Args:

ITOG (pd.DataFrame): общий справочник

Returns:

pd.DataFrame: итоговый отчёт

Автор: Сальник Максим

"""

REP= pd.pivot\_table(ITOG, index='Язык любимого исполнителя', columns='Жанр любимого исполнителя', values='Возраст', aggfunc='mean')

return REP

def make\_table(name, master, column\_names):

"""

Функция вывода справочника в окно приложения

Args:

name (pd.DataFrame): справочник, который нужно вывести

master (Tk): окно или фрейм, в который выводится справочник

column\_names (list): список с названиями столбцов

Автор: Муратов Айнур

"""

name = name.copy() #Копирование, чтобы избежать изменение исходных данных

name.columns = column\_names

height = name.shape[0]

width = name.shape[1]

# Создание виджета Canvas

canvas = tk.Canvas(master)

canvas.pack(side="left", fill="both", expand=True)

# Создание Scrollbar по вертикали

y\_scrollbar = ttk.Scrollbar(master, orient="vertical", command=canvas.yview)

y\_scrollbar.pack(side="right", fill="y")

canvas.configure(yscrollcommand=y\_scrollbar.set)

# Создание Scrollbar по горизонтали

x\_scrollbar = ttk.Scrollbar(master, orient="horizontal", command=canvas.xview)

x\_scrollbar.pack(side="bottom", fill="x")

canvas.configure(xscrollcommand=x\_scrollbar.set)

# Создание вложенного фрейма для размещения таблицы

table\_frame = ttk.Frame(canvas)

canvas.create\_window((0, 0), window=table\_frame, anchor="nw")

# Добавление заголовков столбцов

for j, col\_name in enumerate(column\_names):

label = ttk.Label(table\_frame, text=col\_name)

label.grid(row=0, column=j, sticky="nsew")

# Создание и заполнение таблицы Entry

entry\_widgets = []

for i in range(height):

row\_widgets = []

for j in range(width):

entry = ttk.Entry(table\_frame)

entry.grid(row=i+1, column=j, sticky="nsew")

value = name.iloc[i, j]

entry.insert(0, value)

row\_widgets.append(entry)

entry\_widgets.append(row\_widgets)

# Растягивание ячеек таблицы при изменении размеров окна

for i in range(height + 1):

table\_frame.rowconfigure(i, weight=1)

for j in range(width):

table\_frame.columnconfigure(j, weight=1)

canvas.bind("<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all")))

def save\_file(name):

"""

Функция сохранения файла отчёта или справоника на компьютер

Args:

name (pd.DataFrame): справочник или отчёт, который нужно сохранить

Автор: Муратов Айнур

"""

filetypes = [

("CSV файл", "\*.csv"),

("Текстовый файл", "\*.txt"),

("Эксель файл", "\*.xlsx"),

("Все файлы", "\*.\*")]

file\_path = filedialog.asksaveasfilename(filetypes=filetypes)

if file\_path:

name.to\_csv(file\_path, encoding = 'cp1251')

elif file\_path.endswith(".xlsx"):

name.to\_excel(file\_path, delimiter = ';', encoding = 'cp1251')

def singers\_listeners (ITOG, singer): #Пользователи, которые слушают конкретного исполнителя

"""

Функция создания текстового отчёта в виде базы данных,

содержащей информацию о людях, слушающих данного исполнителя

Args:

ITOG (pd.DataFrame): общий справочник

singer (string): имя исполнителя

Returns:

pd.DataFrame: итоговый отчёт

Автор: Муратов Айнур

"""

SEL = (ITOG["Любимый исполнитель"] == singer)

W1 = ITOG.loc[SEL, ["Идентификатор пользователя","Имя","Фамилия", "Любимый жанр музыки"]]

return W1