**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики - процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Методы заполнения пропусков»**

**Студент гр. 23Б15-пу**

**Кубякин Н.А.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

***Оглавление***

[1. Цель работы 3](#_Toc178797423)

[2. Описание задачи (формализация задачи) 3](#_Toc178797424)

[3. Теоретическая часть 3](#_Toc178797431)

[4. Основные шаги программы 4](#_Toc178797432)

[5. Блок-схема программы 5](#_Toc178797433)

[6. Описание программы 6](#_Toc178797442)

[7. Рекомендации пользователя 7](#_Toc178797443)

[8. Рекомендации программиста 7](#_Toc178797444)

[9. Исходный код программы 7](#_Toc178797445)

[10. Контрольный пример 8](#_Toc178797447)

[11. Вывод 25](#_Toc178797448)

[12. Источники 26](#_Toc178797448)

[13. Листинг 27](#_Toc178797450)

# Цель работы

# Разработать программное обеспечение, которое генерирует датасет по модели 1-ой лабораторной работы и выбрасывает и восстанавливает некоторый процент данных в нём.

# Описание задачи (формализация задачи)

# Сформировать датасеты с помощью программы из 1 лабораторной прошлого семестра разных величин (малый ~ 5000, средний ~ 50000, большой ~ 150000)

# Вычислить среднее значение, значение медианы и моды, построить распределение

# Удалить в столбцах 3, 5, 10, 20, 30 % значений, притом делать это выбросами.

# Выполнить заполнение пропусков методами (Хот-Дек, медиана, стохастическая линейная регрессия)

# Оценить результаты: вычислить среднее значение, значение медианы и моды, построить распределение.

# Сделать выводы.

# Теоретическая часть

* Хот-Дек:  
  Заполняет пропуски случайным значением из того же столбца.
* Метод заполнения медианой:  
  Вычисляет медиану (серединное значение) для каждого столбца и заполняет ей все пропуски.
* Метод заполнения пропусков стохастической линейной регрессией:  
  В отличие от обычной линейной регрессии, которая вычисляет градиент с использованием всего набора данных, стохастическая регрессия обновляет параметры модели итеративно, используя каждый пример.

# Основные шаги программы

1. Создать/загрузить датасет, задав необходимые величины.
2. Задать процент выбросов и убрать это количество данных (замена на nan).
3. Выбрать метод заполнения пропусков.
4. Выполнить заполнение выбранным методом.

# Блок-схема программы

# На рисунке представлена блок-схема алгоритма.

# 

# Рис. 5 Блок-схема алгоритма.

# Описание программы

# Алгоритм реализован на языке Python 3.10 с использованием библиотек tkinter [[1]](#tk), networkx [[2]](#nx), math [[3]](#mt), pandas [[4]](#pd), random [[5]](#rn), sklearn [[6]](#sk). Логика и интерфейс программы реализованы в следующих классах и функциях:

# Алгоритм реализован через модуль data\_processor.py, содержащий следующие функции:

Таблица 6.1 Описание модуля расчёта

|  |  |
| --- | --- |
| Имя функции | Назначение |
| detect\_categorical\_columns | Определяет категориальные переменные. |
| create\_mappings | Создаёт карты для этих признаков. |
| apply\_mappings | Применяет карты к датасету. |
| reverse\_mappings | Возвращает значения по словарю карт. |
| inject\_missing\_values | Внедряет пропуски в датасет. |
| impute\_missing\_values | Восстанавливает пустые значения. |

Интерфейс реализован через класс DataProcessorAppUI, cодержащий следующие функции:

Таблица 6.2 Описание класса интерфейса

|  |  |
| --- | --- |
| Имя функции | Назначение |
| load\_dataset | Загружает датасет. |
| generate\_dataset | Создает датасет по заданным значениям. |
| inject\_missing\_values | Отображает создание пропусков. |
| impute\_missing\_values | Отображает заполнение пропусков. |
| populate\_treeview | Отображает датасет в интерфейсе. |

# Рекомендации пользователя

1. Запустить main.py для запуска графического интерфейса программы.
2. Загрузить готовый датасет или задать значения количества строк и вероятностей значений для генерации датасета с нуля и создать, датасет отобразится в поле справа.
3. Задать процент выбросов и нажать на соответствующую кнопку.
4. Выбрать метод заполнения пропусков и нажать на соответствующую кнопку.

# Рекомендации программиста

Для запуска программы необходима 64-битная операционная система Windows, Linux или macOS. Для работы с кодом необходима среда разработки, совместимая с python 3.1 и библиотеки tkinter [[1]](#tk), networkx [[2]](#nx), math [[3]](#mt), pandas [[4]](#pd), random [[5]](#rn), sklearn [[6]](#sk).

Для запуска программы необходимо установить Python версии не ниже 3.10, а так же библиотеки tkinter [[1]](#tk), networkx [[2]](#nx), math [[3]](#mt), pandas [[4]](#pd), random [[5]](#rn), sklearn [[6]](#sk).

# Исходный код программы

# Исходный код программы и необходимые текстовые файлы доступны по ссылке: <https://github.com/NikiTaku1/spbu_alg_2/tree/main/lab4>

# Контрольный пример

В данном разделе представлен пример, демонстрирующий работу программы на 45 сгенерированных датасетах.

1) Запуск main.py

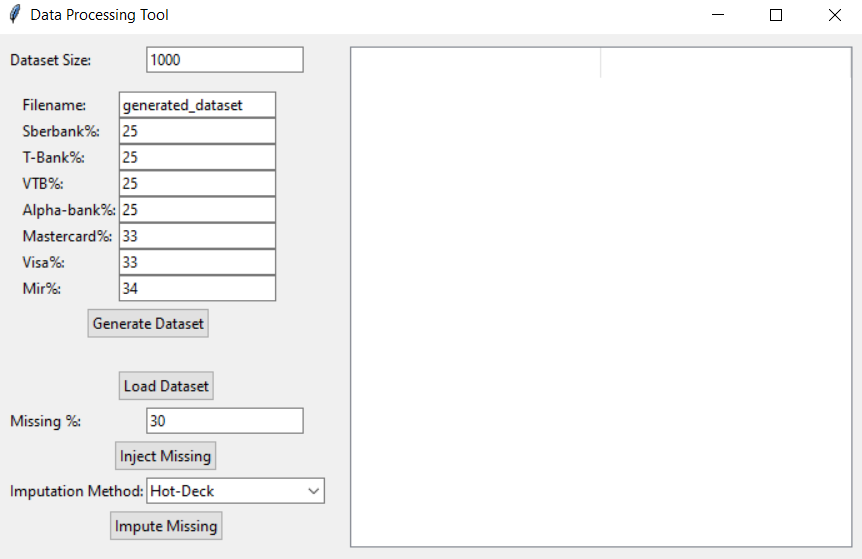


Рис 10.1 Графический интерфейс программы.

2) Генерация датасета

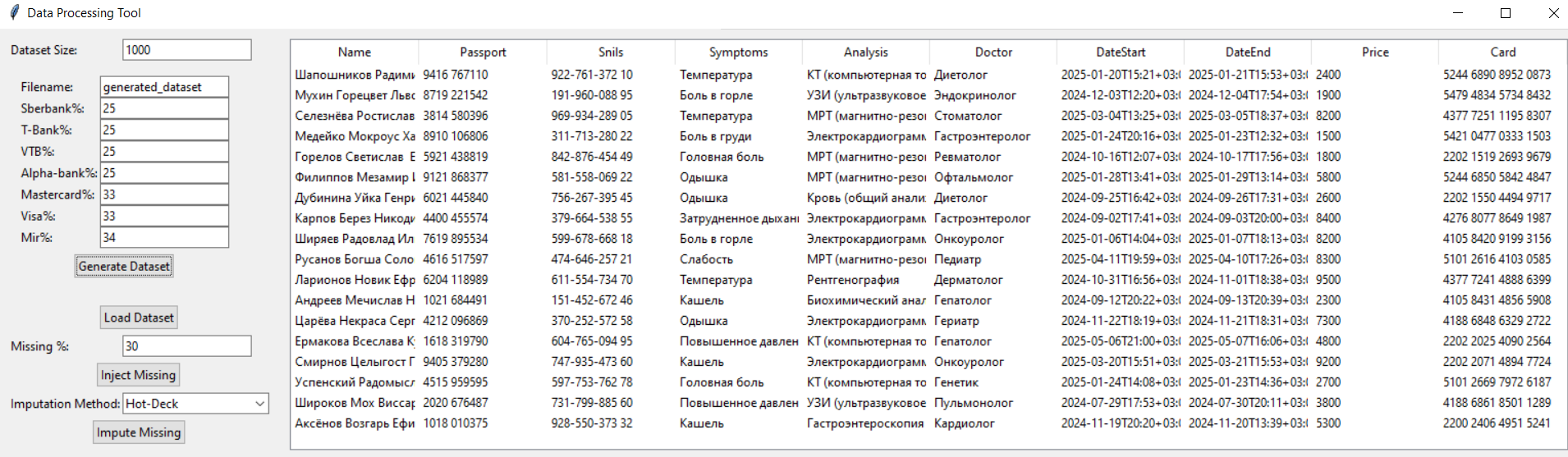


Рис 10.2 Графический интерфейс с созданным датасетом.

3) После выбросов

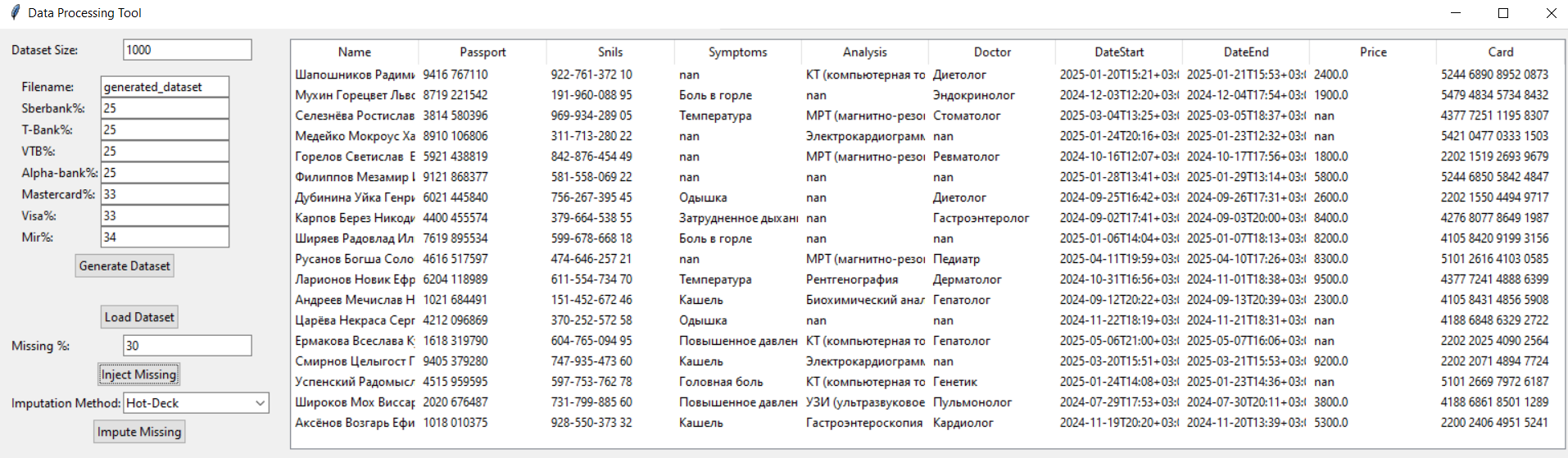


Рис 10.3 Датасет после выбросов.

4) После восстановления

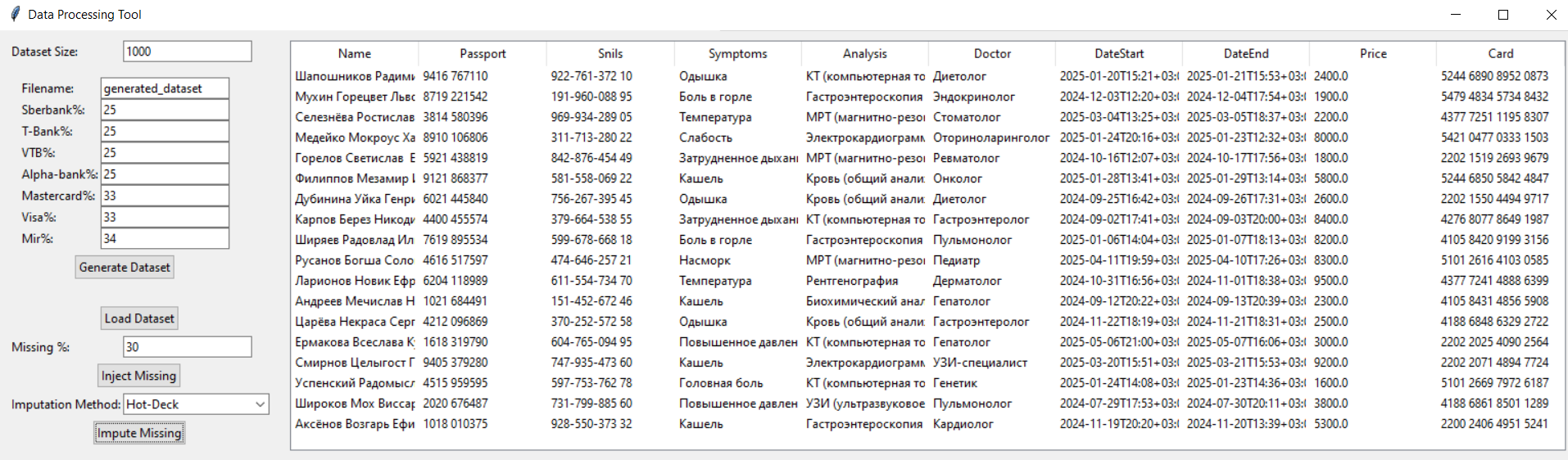


Рис 10.4 Датасет после восстановления данных.

5) Сравнение различных комбинаций начальных значений и методов восстановления

Таблица 10.5.1 Сравнение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество строк | Процент выброса | Метод восстановления | Ошибка (%) | Среднее, мода, медиана |
| 5000 | 3 | - | 0 | Symptoms\_mode: Слабость  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5504.98  Price\_median: 5500.0  Price\_mode: 5500 |
| HD | 2.44 | Symptoms\_mode: Слабость  Analysis\_mode: Рентгенография  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5494.92  Price\_median: 5500.0  Price\_mode: 5500.0 |
| Med | 2.24 | Symptoms\_mode: Боль в горле  Analysis\_mode: УЗИ (ультразвуковое исследование)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5501.98  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 2.20 | Symptoms\_mode: Боль в горле  Analysis\_mode: УЗИ (ультразвуковое исследование)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5505.38  Price\_median: 5500.0  Price\_mode: 5500.0 |
| 5 | - | 0 | Symptoms\_mode: Головная боль  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5490.66  Price\_median: 5500.0  Price\_mode: 3700 |
| HD | 4.11 | Symptoms\_mode: Головная боль  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5468.54  Price\_median: 5450.0  Price\_mode: 2200.0 |
| Med | 3.75 | Symptoms\_mode: Температура  Analysis\_mode: МРТ (магнитно-резонансная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5493.02  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 3.77 | Symptoms\_mode: Слабость  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5490.87  Price\_median: 5500.0  Price\_mode: 3700.0 |
| 10 | - | 0 | Symptoms\_mode: Температура  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5484.4  Price\_median: 5500.0  Price\_mode: 5300 |
| HD | 7.96 | Symptoms\_mode: Температура  Analysis\_mode: Гастроэнтероскопия  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5476.18  Price\_median: 5400.0  Price\_mode: 5300.0 |
| Med | 7.48 | Symptoms\_mode: Повышенное давление  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5494.76  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 7.34 | Symptoms\_mode: Повышенное давление  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Пульмонолог  Price\_mean: 5482.46  Price\_median: 5498.5  Price\_mode: 5300.0 |
| 20 | - | 0 | Symptoms\_mode: Затрудненное дыхание  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5521.24  Price\_median: 5500.0  Price\_mode: 7000 |
| HD | 15.96 | Symptoms\_mode: Затрудненное дыхание  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5536.12  Price\_median: 5500.0  Price\_mode: 6700.0 |
| Med | 15.45 | Symptoms\_mode: Кашель  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Ревматолог  Price\_mean: 5570.68  Price\_median: 5600.0  Price\_mode: 5600.0 |
| SLR | 14.86 | Symptoms\_mode: Головная боль  Analysis\_mode: МРТ (магнитно-резонансная томография)  Doctor\_mode: Ревматолог  Price\_mean: 5528.2  Price\_median: 5515.0  Price\_mode: 5515.0 |
| 30 | - | 0 | Symptoms\_mode: Кашель  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5551.42  Price\_median: 5600.0  Price\_mode: 8100 |
| HD | 24.06 | Symptoms\_mode: Насморк  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5542.82  Price\_median: 5600.0  Price\_mode: 5400.0 |
| Med | 22.46 | Symptoms\_mode: Боль в груди  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Педиатр  Price\_mean: 5575.34  Price\_median: 5600.0  Price\_mode: 5600.0 |
| SLR | 22.41 | Symptoms\_mode: Повышенное давление  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Онкодерматолог  Price\_mean: 5576.41  Price\_median: 5577.0  Price\_mode: 8300.0 |
| 50000 | 3 | - | 0 | Symptoms\_mode: Температура  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5500.92  Price\_median: 5500.0  Price\_mode: 5800 |
| HD | 2.38 | Symptoms\_mode: Температура  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5500.0  Price\_median: 5500.0  Price\_mode: 5800.0 |
| Med | 2.24 | Symptoms\_mode: Насморк  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5501.35  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 2.25 | Symptoms\_mode: Насморк  Analysis\_mode: Гастроэнтероскопия  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5502.52  Price\_median: 5543.0  Price\_mode: 6000.0 |
| 5 | - | 0 | Symptoms\_mode: Повышенное давление  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5512.47  Price\_median: 5500.0  Price\_mode: 6500 |
| HD | 3.99 | Symptoms\_mode: Повышенное давление  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5520.24  Price\_median: 5500.0  Price\_mode: 9400.0 |
| Med | 3.73 | Symptoms\_mode: Повышенное давление  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5509.63  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 3.71 | Symptoms\_mode: Слабость  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5508.9  Price\_median: 5500.0  Price\_mode: 9400.0 |
| 10 | - | 0 | Symptoms\_mode: Головная боль  Analysis\_mode: УЗИ (ультразвуковое исследование)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5511.64  Price\_median: 5500.0  Price\_mode: 2300 |
| HD | 8.00 | Symptoms\_mode: Кашель  Analysis\_mode: МРТ (магнитно-резонансная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5509.32  Price\_median: 5500.0  Price\_mode: 1300.0 |
| Med | 7.43 | Symptoms\_mode: Боль в горле  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Диетолог  Price\_mean: 5511.14  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 7.42 | Symptoms\_mode: Боль в горле  Analysis\_mode: Рентгенография  Doctor\_mode: Онколог  Price\_mean: 5511.59  Price\_median: 5545.0  Price\_mode: 5600.0 |
| 20 | - | 0 | Symptoms\_mode: Кашель  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5496.6  Price\_median: 5500.0  Price\_mode: 3400 |
| HD | 16.04 | Symptoms\_mode: Кашель  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5495.22  Price\_median: 5500.0  Price\_mode: 3400.0 |
| Med | 14.83 | Symptoms\_mode: Слабость  Analysis\_mode: Рентгенография  Doctor\_mode: Неонатолог  Price\_mean: 5497.53  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 14.83 | Symptoms\_mode: Слабость  Analysis\_mode: УЗИ (ультразвуковое исследование)  Doctor\_mode: Неонатолог  Price\_mean: 5478.72  Price\_median: 5448.0  Price\_mode: 5500.0 |
| 30 | - | 0 | Symptoms\_mode: Одышка  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5494.31  Price\_median: 5500.0  Price\_mode: 1100 |
| HD | 24.04 | Symptoms\_mode: Боль в груди  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5492.1  Price\_median: 5500.0  Price\_mode: 3700.0 |
| Med | 22.18 | Symptoms\_mode: Боль в груди  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Пульмонолог  Price\_mean: 5500.64  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 22.28 | Symptoms\_mode: Боль в груди  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Пульмонолог  Price\_mean: 5487.42  Price\_median: 5471.0  Price\_mode: 5500.0 |
| 150000 | 3 | - | 0 | Symptoms\_mode: Затрудненное дыхание  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5504.2  Price\_median: 5500.0  Price\_mode: 2800 |
| HD | 2.40 | Symptoms\_mode: Затрудненное дыхание  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5501.8  Price\_median: 5500.0  Price\_mode: 3100.0 |
| Med | 2.24 | Symptoms\_mode: Одышка  Analysis\_mode: МРТ (магнитно-резонансная томография)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5505.42  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 2.24 | Symptoms\_mode: Повышенное давление  Analysis\_mode: Рентгенография  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5503.73  Price\_median: 5500.0  Price\_mode: 2800.0 |
| 5 | - | 0 | Symptoms\_mode: Боль в горле  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5491.48  Price\_median: 5500.0  Price\_mode: 9700 |
| HD | 3.99 | Symptoms\_mode: Боль в горле  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5491.1  Price\_median: 5500.0  Price\_mode: 9700.0 |
| Med | 3.70 | Symptoms\_mode: Кашель  Analysis\_mode: КТ (компьютерная томография)  Doctor\_mode: Кардиолог  Price\_mean: 5493.44  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 3.73 | Symptoms\_mode: Одышка  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5491.99  Price\_median: 5500.0  Price\_mode: 9700.0 |
| 10 | - | 0 | Symptoms\_mode: Насморк  Analysis\_mode: Гастроэнтероскопия  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5494.02  Price\_median: 5500.0  Price\_mode: 5700 |
| HD | 8.00 | Symptoms\_mode: Насморк  Analysis\_mode: УЗИ (ультразвуковое исследование)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5493.4  Price\_median: 5500.0  Price\_mode: 5700.0 |
| Med | 7.47 | Symptoms\_mode: Боль в горле  Analysis\_mode: Электрокардиограмма (ЭКГ)  Doctor\_mode: Психотерапевт  Price\_mean: 5492.75  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 7.47 | Symptoms\_mode: Температура  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5494.93  Price\_median: 5499.0  Price\_mode: 5500.0 |
| 20 | - | 0 | Symptoms\_mode: Температура  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5500.95  Price\_median: 5500.0  Price\_mode: 3900 |
| HD | 16.01 | Symptoms\_mode: Температура  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5502.76  Price\_median: 5500.0  Price\_mode: 3900.0 |
| Med | 14.96 | Symptoms\_mode: Затрудненное дыхание  Analysis\_mode: Кровь (общий анализ)  Doctor\_mode: Психотерапевт  Price\_mean: 5496.75  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 14.88 | Symptoms\_mode: Кашель  Analysis\_mode: Гастроэнтероскопия  Doctor\_mode: Фтизиатр  Price\_mean: 5513.21  Price\_median: 5550.0  Price\_mode: 5500.0 |
| 30 | - | 0 | Symptoms\_mode: Одышка  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5504.33  Price\_median: 5500.0  Price\_mode: 4600 |
| HD | 23.97 | Symptoms\_mode: Боль в груди  Analysis\_mode: Биохимический анализ крови  Doctor\_mode: Гастроэнтеролог  Price\_mean: 5497.57  Price\_median: 5500.0  Price\_mode: 4600.0 |
| Med | 22.23 | Symptoms\_mode: Повышенное давление  Analysis\_mode: УЗИ (ультразвуковое исследование)  Doctor\_mode: Эндоскопист  Price\_mean: 5507.78  Price\_median: 5500.0  Price\_mode: 5500.0 |
| SLR | 22.35 | Symptoms\_mode: Повышенное давление  Analysis\_mode: Рентгенография  Doctor\_mode: Аллерголог  Price\_mean: 5506.61  Price\_median: 5512.0  Price\_mode: 5600.0 |

Таким образом из трёх используемых методов лучше всего себя показала стохастическая линейная регрессия, хотя заполнение медианой даёт результат либо похожий либо немного хуже, хот-дек же сильно хуже остальных двух методов. Так же можно заметить что процент ошибки с одинаковым процентом выбросов сохраняется примерно одинаковым независимо от размера датасета.

# Вывод

В процессе исследования методов заполнения пропусков в датасетах была написана программа, создающая датасет, позволяющая выбрасывать заданный процент данных и восстанавливать их в нём различными методами. Проведён анализ эффективности алгоритмов при различных начальных датасетах и процентах выбросов.

# Источники

# tkinter — tkinter documentation // Documentation URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 25.04.2025).

# networkx — networkx documentation // Documentation URL: [https://networkx.org/documentation/latest/](https://networkx.org/documentation/latest/%20) (дата обращения: 25.04.2025).

# math — math documentation // Documentation URL: <https://docs.python.org/3/library/math.html> (дата обращения: 25.04.2025).

# pandas — pandas documentation // Documentation URL: [https://pandas.pydata.org/docs/](https://pandas.pydata.org/docs/%20) (дата обращения: 25.04.2025).

# random — random documentation // Documentation URL: <https://docs.python.org/3/library/random.html> (дата обращения: 25.04.2025).

# sklearn — sklearn documentation // Documentation URL: <https://scikit-learn.org/stable/index.html> (дата обращения: 25.04.2025).

# Листинг

# 1) main.py

import tkinter as tk

import ui

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app\_ui = ui.DataProcessorAppUI(root)

    root.mainloop()

# 2) ui.py

import tkinter as tk

from tkinter import ttk, filedialog, messagebox

import pandas as pd

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import matplotlib.pyplot as plt

import data\_processor as dp

import dataset\_generator as data\_gen

import threading

import os

class DataProcessorAppUI:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Data Processing Tool")

        self.df = None

        self.df\_display = None

        self.df\_missing = None

        self.file\_path = None

        self.categorical\_cols = []

        self.mappings = {}

        self.numeric\_cols = []

        # --- Left Frame: Controls ---

        self.left\_frame = ttk.Frame(self.root, padding=10)

        self.left\_frame.grid(row=0, column=0, sticky=(tk.N, tk.S, tk.E, tk.W))

        # Dataset Generation

        ttk.Label(self.left\_frame, text="Dataset Size:").grid(row=0, column=0, sticky=tk.W)

        self.dataset\_size\_var = tk.StringVar(value="1000")

        self.count\_entry = ttk.Entry(self.left\_frame, textvariable=self.dataset\_size\_var)

        self.count\_entry.grid(row=0, column=1, sticky=tk.W)

        # Load Dataset Button

        ttk.Button(self.left\_frame, text="Load Dataset", command=self.load\_dataset).grid(row=2, column=0, columnspan=2, pady=5)

        # Missing Value Injection

        ttk.Label(self.left\_frame, text="Missing %:").grid(row=3, column=0, sticky=tk.W)

        self.missing\_percent\_var = tk.StringVar(value="30")

        self.missing\_entry = ttk.Entry(self.left\_frame, textvariable=self.missing\_percent\_var)

        self.missing\_entry.grid(row=3, column=1, sticky=tk.W)

        ttk.Button(self.left\_frame, text="Inject Missing", command=self.inject\_missing\_values).grid(row=4, column=0, columnspan=2, pady=5)

        # Imputation Method Selection

        ttk.Label(self.left\_frame, text="Imputation Method:").grid(row=5, column=0, sticky=tk.W)

        self.imputation\_method\_var = tk.StringVar(value="Hot-Deck")

        imputation\_methods = ["Hot-Deck", "Median", "Linear Regression"]

        self.imputation\_dropdown = ttk.Combobox(self.left\_frame, textvariable=self.imputation\_method\_var, values=imputation\_methods, state="readonly")

        self.imputation\_dropdown.grid(row=5, column=1, sticky=tk.W)

        ttk.Button(self.left\_frame, text="Impute Missing", command=self.impute\_missing\_values).grid(row=6, column=0, columnspan=2, pady=5)

        # --- Right Frame: Data Display ---

        self.right\_frame = ttk.Frame(self.root, padding=10)

        self.right\_frame.grid(row=0, column=1, sticky=(tk.N, tk.S, tk.E, tk.W))

        self.tree = ttk.Treeview(self.right\_frame, columns=("col1", "col2"), show="headings")  # Example columns

        self.tree.pack(expand=True, fill=tk.BOTH)

        # --- Configure Grid Weights ---

        self.root.columnconfigure(1, weight=1)

        self.root.rowconfigure(0, weight=1)

        # Dataset Generation Frame

        self.dataset\_generation\_frame = ttk.Frame(self.left\_frame, padding=10)

        self.dataset\_generation\_frame.grid(row=1, column=0, columnspan=2, pady=5, sticky=tk.W)

        ttk.Label(self.dataset\_generation\_frame, text="Filename:").grid(row=0, column=0, sticky=tk.W)

        self.output\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.output\_entry.grid(row=0, column=1, sticky=tk.W)

        self.output\_entry.insert(0, "generated\_dataset")

        ttk.Label(self.dataset\_generation\_frame, text="Sberbank%:").grid(row=1, column=0, sticky=tk.W)

        self.a\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.a\_entry.grid(row=1, column=1, sticky=tk.W)

        self.a\_entry.insert(0, "25")

        ttk.Label(self.dataset\_generation\_frame, text="T-Bank%:").grid(row=2, column=0, sticky=tk.W)

        self.b\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.b\_entry.grid(row=2, column=1, sticky=tk.W)

        self.b\_entry.insert(0, "25")

        ttk.Label(self.dataset\_generation\_frame, text="VTB%:").grid(row=3, column=0, sticky=tk.W)

        self.c\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.c\_entry.grid(row=3, column=1, sticky=tk.W)

        self.c\_entry.insert(0, "25")

        ttk.Label(self.dataset\_generation\_frame, text="Alpha-bank%:").grid(row=4, column=0, sticky=tk.W)

        self.d\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.d\_entry.grid(row=4, column=1, sticky=tk.W)

        self.d\_entry.insert(0, "25")

        ttk.Label(self.dataset\_generation\_frame, text="Mastercard%:").grid(row=5, column=0, sticky=tk.W)

        self.x\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.x\_entry.grid(row=5, column=1, sticky=tk.W)

        self.x\_entry.insert(0, "33")

        ttk.Label(self.dataset\_generation\_frame, text="Visa%:").grid(row=6, column=0, sticky=tk.W)

        self.y\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.y\_entry.grid(row=6, column=1, sticky=tk.W)

        self.y\_entry.insert(0, "33")

        ttk.Label(self.dataset\_generation\_frame, text="Mir%:").grid(row=7, column=0, sticky=tk.W)

        self.z\_entry = ttk.Entry(self.dataset\_generation\_frame)

        self.z\_entry.grid(row=7, column=1, sticky=tk.W)

        self.z\_entry.insert(0, "34")

        ttk.Button(self.dataset\_generation\_frame, text="Generate Dataset", command=lambda: self.generate\_dataset(self.a\_entry, self.b\_entry, self.c\_entry, self.d\_entry, self.x\_entry, self.y\_entry, self.z\_entry, self.count\_entry, self.output\_entry)).grid(row=8, column=0, columnspan=2, pady=5)

    def load\_dataset(self):

        self.file\_path = filedialog.askopenfilename(title="Select CSV File", filetypes=[("CSV files", "\*.csv")])

        if self.file\_path:

            try:

                self.df = pd.read\_csv(self.file\_path)

                self.df\_display = self.df.copy()

                self.categorical\_cols = dp.detect\_categorical\_columns(self.df)

                self.mappings = dp.create\_mappings(self.df, self.categorical\_cols)

                self.numeric\_cols = [col for col in self.df.columns if pd.api.types.is\_numeric\_dtype(self.df[col]) and col not in ['Name', 'Passport', 'Snils', 'Card']]

                self.populate\_treeview()

            except Exception as e:

                messagebox.showerror("Error", f"Failed to load dataset: {e}")

    def generate\_dataset(self, a\_entry, b\_entry, c\_entry, d\_entry, x\_entry, y\_entry, z\_entry, count\_entry, output\_entry):

        size = int(self.dataset\_size\_var.get())

        banks\_p = {

            "sberbank": int(a\_entry.get()),

            "tbank": int(b\_entry.get()),

            "vtb": int(c\_entry.get()),

            "alphabank": int(d\_entry.get()),

        }

        systems\_p = {

            "mastercard": int(x\_entry.get()),

            "visa": int(y\_entry.get()),

            "mir": int(z\_entry.get()),

        }

        try:

            data\_gen.compute(size, output\_entry.get(), banks\_p, systems\_p)

            self.file\_path = output\_entry.get() + ".csv"

            self.df = pd.read\_csv(self.file\_path)

            self.df\_display = self.df.copy()

            self.categorical\_cols = dp.detect\_categorical\_columns(self.df)

            self.mappings = dp.create\_mappings(self.df, self.categorical\_cols)

            self.numeric\_cols = [col for col in self.df.columns if pd.api.types.is\_numeric\_dtype(self.df[col]) and col not in ['Name', 'Passport', 'Snils', 'Card']]

            self.populate\_treeview()

            messagebox.showinfo("Success", "Dataset generated successfully!")

        except Exception as e:

            messagebox.showerror("Error", f"Failed to generate dataset: {e}")

    def inject\_missing\_values(self):

        if self.df is None:

            messagebox.showinfo("Info", "Load or generate a dataset first.")

            return

        # Define non-modifiable columns

        non\_modifiable\_cols = ['Passport', 'Snils', 'Card', 'DateStart', 'DateEnd']

        # Columns to modify (excluding 'Name' and non-modifiable columns)

        cols\_to\_modify = [col for col in self.df.columns if col not in ['Name'] + non\_modifiable\_cols]

        percent = float(self.missing\_percent\_var.get()) / 100

        # Apply to the df before and displayed for UI

        self.df\_missing = self.df.copy()

        df\_missing\_injected = dp.inject\_missing\_values(self.df\_missing, percent, cols\_to\_modify)

        self.df\_display = df\_missing\_injected

        df\_mapped = dp.apply\_mappings(df\_missing\_injected, self.mappings)

        self.df = df\_mapped

        self.numeric\_cols = [col for col in self.df.columns if pd.api.types.is\_numeric\_dtype(self.df[col]) and col not in ['Name', 'Passport', 'Snils', 'Card']] #check which are not numeric

        self.populate\_treeview()

        messagebox.showinfo("Success", "Missing values injected.")

    def impute\_missing\_values(self):

        if self.df is None:

            messagebox.showinfo("Info", "Load or generate a dataset first.")

            return

        method = self.imputation\_method\_var.get()

        df\_imputed\_mapped = dp.impute\_missing\_values(self.df, method, self.numeric\_cols)

        df\_display\_reverted = self.df.copy()

        mappings\_to\_use = self.mappings

        for col in self.categorical\_cols:

          if col in mappings\_to\_use:

            df\_display\_reverted[col] = df\_display\_reverted[col].map({v: k for k, v in mappings\_to\_use[col].items()})

        self.df\_display = df\_display\_reverted

        self.df = df\_imputed\_mapped

        if self.file\_path:

            output\_dir = os.path.join(os.path.dirname(self.file\_path), "imputed")

            os.makedirs(output\_dir, exist\_ok=True)

            # Create the output file path within the "imputed" directory

            file\_name, file\_extension = os.path.splitext(os.path.basename(self.file\_path))

            output\_path = os.path.join(output\_dir, f"{file\_name}\_imputed\_{method}{file\_extension}")

            # Save the DataFrame to the output path

            try:

                self.df\_display.to\_csv(output\_path, index=False)

                messagebox.showinfo("Success", f"Missing values imputed using {method}. Dataset saved to {output\_path}")

            except Exception as e:

                messagebox.showerror("Error", f"Failed to save imputed dataset: {e}")

        else:

            messagebox.showinfo("Info", "No file loaded, cant save it. Load or generate a dataset first.")

        # Refresh Treeview

        self.populate\_treeview()

    def populate\_treeview(self):

        """Populates the Treeview with the DataFrame's data."""

        for item in self.tree.get\_children():

            self.tree.delete(item)

        if self.df\_display is not None:

            self.tree["columns"] = list(self.df\_display.columns)

            self.tree["show"] = "headings"

            for col in self.df\_display.columns:

                self.tree.heading(col, text=col)

                self.tree.column(col, width=100)

            for index, row in self.df\_display.iterrows():

                self.tree.insert("", tk.END, values=list(row))

    @staticmethod

    def is\_numeric\_dtype\_string(dtype):

        """Helper function to check if a dtype string represents a numeric type."""

        numeric\_types = ['int', 'float']

        return any(numeric\_type in dtype.name for numeric\_type in numeric\_types)

3) data\_processor.py

import pandas as pd

import numpy as np

import random

from sklearn.impute import SimpleImputer

from sklearn.linear\_model import SGDRegressor

from sklearn.preprocessing import StandardScaler

def detect\_categorical\_columns(df):

    """Identifies categorical columns based on data type."""

    categorical\_cols = []

    if df is not None:

        for col in df.columns:

            if pd.api.types.is\_object\_dtype(df[col]):  # Check if object (string)

                categorical\_cols.append(col)

    return categorical\_cols

def create\_mappings(df, categorical\_cols):

    """Creates mappings (dictionaries) for categorical columns."""

    mappings = {}

    for col in categorical\_cols:

        unique\_values = df[col].unique()

        mappings[col] = {val: i for i, val in enumerate(unique\_values)}

    return mappings

def apply\_mappings(df, mappings):

    """Applies the mappings to convert categorical columns to numerical."""

    df\_mapped = df.copy()

    for col, mapping in mappings.items():

        df\_mapped[col] = df\_mapped[col].map(mapping)

    return df\_mapped

def reverse\_mappings(df, mappings):

    """Reverts the mappings (numerical to original labels)."""

    df\_reverted = df.copy()

    for col, mapping in mappings.items():

        # Invert the mapping dictionary

        inverse\_mapping = {v: k for k, v in mapping.items()}

        if col in df\_reverted.columns:  # Handle potential missing columns

            df\_reverted[col] = df\_reverted[col].map(inverse\_mapping)

    return df\_reverted

def inject\_missing\_values(df, percent, cols\_to\_modify):

    """Injects missing values as NaN into the specified columns."""

    df\_missing = df.copy()

    for col in cols\_to\_modify:

        num\_missing = int(len(df) \* percent)

        indices = random.sample(range(len(df)), num\_missing)

        df\_missing.loc[indices, col] = np.nan  # Introduce NaN

    return df\_missing

def impute\_missing\_values(df, method, numeric\_cols):

    """Imputes missing values using the specified method."""

    if method == "Median":

        imputer = SimpleImputer(strategy='median')

        for col in numeric\_cols:

            if pd.api.types.is\_numeric\_dtype(df[col]):

                df[[col]] = imputer.fit\_transform(df[[col]]) #try imputing each value alone

    elif method == "Linear Regression":

        for col in numeric\_cols:

            if df[col].isnull().any() and len(df) > 0:

                if len(df[[col]].dropna()) < 2:

                    continue

                df\_temp = df[[col]].copy()

                df\_temp['index'] = df.index  # Use index as a feature

                known = df\_temp[df\_temp[col].notnull()]

                unknown = df\_temp[df\_temp[col].isnull()]

                if len(known) > 2 and len(unknown) > 0:

                    # Train the model

                    try:

                        # Scale the features

                        scaler = StandardScaler()

                        known[['index']] = scaler.fit\_transform(known[['index']])

                        # Train the model

                        model = SGDRegressor(loss='squared\_error', penalty=None, max\_iter=1000, tol=1e-3)

                        model.fit(known[['index']], known[col])

                        # Predict missing values

                        unknown[['index']] = scaler.transform(unknown[['index']])

                        predicted = model.predict(unknown[['index']])

                        #Get max min values

                        min\_val = df[col].min() #get min value

                        max\_val = df[col].max() #get max value

                        predicted = np.clip(predicted, min\_val, max\_val) #Clip to set value

                        df.loc[unknown.index, col] = predicted.astype(int) #Округляем значения

                    except Exception as e:

                        print(e)

    # Hot-Deck Imputation to fill any remaining NaN values

    for col in numeric\_cols:

        valid\_values = df[col].dropna().values  # Remove NaN values

        if len(valid\_values) > 0:  # Check data before imputing values

            df[col] = df[col].fillna(pd.Series(np.random.choice(valid\_values, size=len(df))))  # Set if data is correct

    return df

3) dataset\_generator.py

import random

import pandas

def compute(n, out\_file, banks\_p, systems\_p):

    dictPath = "dictionary"

    names1 = [

        i.replace("\n", "")

        for i in (open(dictPath + "/names\_1.txt", "r", encoding="utf-8").readlines())

    ]

    names2 = [

        i.replace("\n", "")

        for i in (open(dictPath + "/names\_2.txt", "r", encoding="utf-8").readlines())

    ]

    lastnames1 = [

        i.replace("\n", "")

        for i in (

            open(dictPath + "/lastnames\_1.txt", "r", encoding="utf-8").readlines()

        )

    ]

    lastnames2 = [

        i.replace("\n", "")

        for i in (

            open(dictPath + "/lastnames\_2.txt", "r", encoding="utf-8").readlines()

        )

    ]

    patronymics1 = [

        i.replace("\n", "")

        for i in (

            open(dictPath + "/patronymics\_1.txt", "r", encoding="utf-8").readlines()

        )

    ]

    patronymics2 = [

        i.replace("\n", "")

        for i in (

            open(dictPath + "/patronymics\_2.txt", "r", encoding="utf-8").readlines()

        )

    ]

    doctors = [

        i.replace("\n", "")

        for i in (open(dictPath + "/doctors.txt", "r", encoding="utf-8").readlines())

    ]

    symptoms = [

        i.replace("\n", "")

        for i in (open(dictPath + "/symptoms.txt", "r", encoding="utf-8").readlines())

    ]

    analysis = [

        i.replace("\n", "")

        for i in (open(dictPath + "/analysis.txt", "r", encoding="utf-8").readlines())

    ]

    card\_keys = {}

    for i in open(dictPath + "/card\_keys.txt", "r", encoding="utf-8").readlines():

        row = i.replace("\n", "").split(" ")

        card\_keys[row[0] + "\_" + row[1]] = row[2]

    namesGen = [

        NamesGenerator(names1, lastnames1, patronymics1),

        NamesGenerator(names2, lastnames2, patronymics2),

    ]

    passportGen = PassportGenerator()

    snilsGen = SnilsGenerator()

    sympGen = SamplesGenerator(symptoms, 1)

    analysisGen = SamplesGenerator(analysis, 1)

    dateGen = DatetimeGenerator()

    cardGen = CardGenerator(banks\_p, systems\_p, card\_keys)

    data = {

        "Name": [],

        "Passport": [],

        "Snils": [],

        "Symptoms": [],

        "Analysis": [],

        "Doctor": [],

        "DateStart": [],

        "DateEnd": [],

        "Price": [],

        "Card": [],

    }

    datacheck = {

        "Name": [],

        "Passport": [],

        "Snils": [],

        "Card" : [],

    }

    for i in range(n):

        data["Name"].append(namesGen[random.randrange(2)].generate())

        data["Passport"].append(passportGen.generate())

        snilscheck = snilsGen.generate()

        while snilscheck in data["Snils"]:

            snilscheck = snilsGen.generate()

        data["Snils"].append(snilscheck)

    for i in range(n):

        if data["Name"][i] in datacheck["Name"]:

            ind = datacheck["Name"].index(data["Name"][i])

            datacheck["Name"].append(data["Name"][ind])

            datacheck["Passport"].append(data["Passport"][ind])

            datacheck["Snils"].append(data["Snils"][ind])

        else:

            datacheck["Name"].append(data["Name"][i])

            datacheck["Passport"].append(data["Passport"][i])

            datacheck["Snils"].append(data["Snils"][i])

    data["Name"] = datacheck["Name"]

    data["Passport"] = datacheck["Passport"]

    data["Snils"] = datacheck["Snils"]

    for i in range(n):

        num = random.randint(0, 49)

        data["Symptoms"].append(sympGen.generate())

        data["Analysis"].append(analysisGen.generate())

        data["Doctor"].append(doctors[num])

        data["DateStart"].append(dateGen.generate())

        data["DateEnd"].append(dateGen.generate())

        data["Price"].append(str(random.randint(10, 100) \* 100))

    for i in range(n):

        data["Card"].append(cardGen.generate())

    for i in range(n):

        if (data["Name"][i] in datacheck["Name"]):

            ind1 = datacheck["Name"].index(data["Name"][i])

            datacheck["Card"].append(data["Card"][ind1])

        else:

            datacheck["Card"].append(data["Card"][i])

    data["Card"] = datacheck["Card"]

    df = pandas.DataFrame(data)

    df.to\_csv(f"{out\_file}.csv", index=False)

import random

import datetime

class PassportGenerator:

    def \_\_init\_\_(self):

        self.used\_keys = {}

    def generate(self):

        key = self.\_generate\_unique\_key()

        self.used\_keys[key] = (random.randrange(10\*\*6 - 1))

        if key in self.used\_keys:

            self.used\_keys[key] += 1

        return "{:04d}".format(key) + " {:06d}".format(self.used\_keys[key])

    def \_generate\_unique\_key(self):

        key = (random.randrange(10\*\*2 - 1) + 1) \* 100 + (random.randrange(24))

        return key

class SnilsGenerator:

    def \_\_init\_\_(self):

        self.nums = ''

    def generate(self):

        self.nums = [

            random.randint(0, 9) if x != 3 and x != 7 and x != 11

            else '-' if x == 3 or x == 7

            else ' '

            for x in range(0, 12)

        ]

        cont = self.nums[0] \* 9 + self.nums[1] \* 8 + self.nums[2] \* 7 + self.nums[4] \* 6 + self.nums[5] \* 5 + self.nums[6] \* 4 + self.nums[8] \* 3 + self.nums[9] \* 2 + self.nums[10] \* 1

        if cont in (100, 101):

            cont = '00'

        elif cont > 101:

            cont = cont % 101

            if cont in (100, 101):

                cont = '00'

            elif cont < 10:

                cont = '0' + str(cont)

        elif cont < 10:

            cont = '0' + str(cont)

        self.nums.append(str(cont))

        return ''.join([str(x) for x in self.nums])

class NamesGenerator:

    def \_\_init\_\_(self, names, lastnames, patronymics):

        self.names = names

        self.lastnames = lastnames

        self.patronymics = patronymics

    def generate(self):

        return " ".join(

            [

                random.choice(self.lastnames),

                random.choice(self.names),

                random.choice(self.patronymics),

            ]

        )

class SamplesGenerator:

    def \_\_init\_\_(self, symptoms, k):

        self.symptoms = symptoms

        self.k = k

    def generate(self):

        return "|".join(random.sample(self.symptoms, k=self.k))

class DatetimeGenerator:

    def \_\_init\_\_(self):

        self.last = 0

    def generate(self):

        return (self.\_generate\_date() + self.\_generate\_time()).strftime(

            "%Y-%m-%dT%H:%M+03:00"

        )

    def \_generate\_date(self):

        if self.last == 0:

            rnd\_days\_ago = random.randrange(300)

            date = datetime.datetime.now() - datetime.timedelta(days=rnd\_days\_ago)

            if date.weekday() >= 5:

                date = date - datetime.timedelta(days=2)

            self.last = date

            return date

        else:

            date = self.last + datetime.timedelta(days=1)

            if date.weekday() >= 5:

                date = date - datetime.timedelta(days=2)

            self.last = 0

            return date

    def \_generate\_time(self):

        time = datetime.timedelta(minutes=(random.randrange(-180, 360)))

        return time

class CardGenerator:

    def \_\_init\_\_(self, banks, systems, keys):

        self.banks = banks

        self.systems = systems

        self.keys = keys

        self.used = {}

    def \_generate\_key(self, p\_dict):

        rnd = random.randrange(0, 100)

        accum = 0

        for key in p\_dict:

            accum += p\_dict[key]

            if rnd <= accum:

                return key

        return ""

    def generate(self):

        bank = self.\_generate\_key(self.banks)

        system = self.\_generate\_key(self.systems)

        key = str(self.keys[bank + "\_" + system])

        self.used[key] = (random.randrange(10\*\*10 - 1))

        card = key + "{:010d}".format(self.used[key])

        card\_split = []

        for i in range(4):

            card\_split.append(card[i \* 4 : (i + 1) \* 4])

        return " ".join(card\_split)