Анализ и сравнение различных способов обработки и хранение больших данных: Pandas, Dask, Seaborn

Обзор проекта: 2
Основные Аспекты 2
Pandas. 3
Dask. 5
Seaborn 9
Заключение: 12
Dask 12
Seaborn 12
Краткое сравнение: 13

Обзор проекта:

Предлагается в теме использовать 3 библиотеки Python. Для дальнейшего сравнения этих фреймворков. В данном проекте будет использоваться база данных пассажиров Титаника.

Данная база данных состоит из таких колонок как:

PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked

Структура проекта:

https://github.com/AtLaSFaNt0m/Diplom/blob/main/file_structure.txt

Основные Аспекты

Из-за сравнения я сделал 3 папки под характерные библиотеки. Pandas, Dask, Seaborn.

И отдельно вывел папку с базой данных пассажиров.

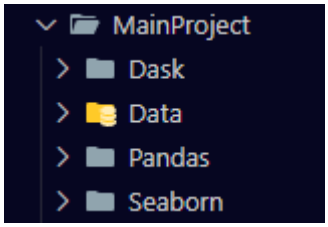


Рисунок 1. коренные директории программ

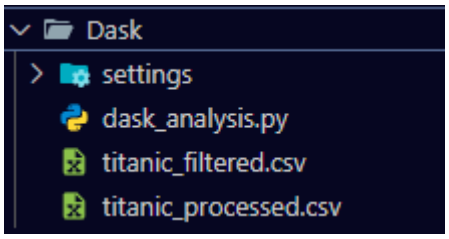


Рисунок 2. файловая структура фреймворка Dask

Рисунок 3. файловая структура базы данных

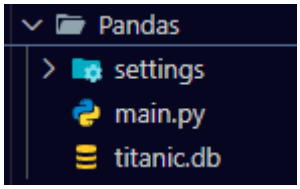


Рисунок 4. файловая структура фреймворка Pandas

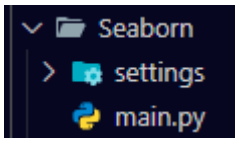


Рисунок 5. файловая структура фрейморка Seaborn

Pandas.

— это популярная библиотека для анализа данных в Python, которая предоставляет мощные структуры данных и инструменты для работы с таблицами и временными рядами.

Основные структуры:

1. **DataFrame** — двухмерная таблица, аналогичная таблицам в базах данных или Excel, где строки и столбцы имеют метки.
2. **Series** — одномерная структура данных, которая работает как столбец в таблице.

Ключевые функции:

- Чтение данных из различных источников (CSV, Excel, SQL и т.д.).
- Фильтрация, сортировка и группировка данных.
- Обработка пропущенных значений.
- Изменение структуры данных (например, поворот таблицы, слияние, объединение).

Pandas идеально подходит для работы с данными небольшого и среднего объема, где можно легко выполнять сложные манипуляции и анализ.

```python
import time
import pandas as pd
from settings.data_operations import DataProcessor, DataBaseEditor
from settings.settings import DataSorter, DataEditor
from settings.redacted import TableEditor

def main():
    # Пути к файлам и папкам
    csv_path = 'A:/Abobapy/DiplomProject/MainProject/Data/titanic/titanic.csv'
    db_path = 'A:/Abobapy/DiplomProject/MainProject/Pandas/titanic.db'
    settings_folder = 'A:/Abobapy/DiplomProject/MainProject/Pandas/settings'

    # Инициализация DataProcessor
    print("Инициализация DataProcessor...")
    start_time = time.time()
    processor = DataProcessor(csv_path, db_path, settings_folder)
    end_time = time.time()
    print(f"Инициализация DataProcessor заняла {end_time - start_time:.2f} секунд.")

    # Сортировка данных по возрасту с использованием DataSorter
    print("Сортировка данных по возрасту...")
    sorter = DataSorter(processor.df, settings_folder)
    start_time = time.time()
    sorted_file = sorter.sort_by_column('Age')
    end_time = time.time()
    print(f"Сортировка данных заняла {end_time - start_time:.2f} секунд. Сохранено в: {sorted_file}")

    # Добавление возрастной группы с использованием DataEditor
    print("Добавление возрастных групп...")
    editor = DataEditor(processor.df, settings_folder)
    start_time = time.time()
    edited_file = editor.add_age_group()
    end_time = time.time()
    print(f"Добавление возрастных групп заняло {end_time - start_time:.2f} секунд. Сохранено в: {edited_file}")

    # Построение детализированного графика по возрасту
    print("Построение детализированного графика по возрасту...")
    start_time = time.time()
    plot_file = processor.plot_age_distribution_detailed()
    end_time = time.time()
    print(f"Построение графика заняло {end_time - start_time:.2f} секунд. График сохранен в: {plot_file}")

    # Инициализация TableEditor для редактирования столбцов
    print("Инициализация TableEditor...")
    start_time = time.time()
    editor = TableEditor(csv_path, settings_folder)
    end_time = time.time()
    print(f"Инициализация TableEditor заняла {end_time - start_time:.2f} секунд.")

    # Пример удаления столбца
    print("Удаление столбца 'Cabin'...")
    start_time = time.time()
    try:
        updated_file = editor.delete_column('Cabin')
        end_time = time.time()
        print(f"Удаление столбца заняло {end_time - start_time:.2f} секунд. Обновленная таблица сохранена в: {updated_file}")
    except ValueError as e:
        print(e)

    # Пример переименования столбца
    print("Переименование столбца 'Survived' в 'SurvivalStatus'...")
    start_time = time.time()
    try:
        renamed_file = editor.rename_column('Survived', 'SurvivalStatus')
        end_time = time.time()
        print(f"Переименование столбца заняло {end_time - start_time:.2f} секунд. Обновленная таблица сохранена в: {renamed_file}")
    except ValueError as e:
        print(e)

    # Сохранение текущей таблицы
    print("Сохранение текущего состояния таблицы...")
    start_time = time.time()
    saved_path = editor.save_table()
    end_time = time.time()
    print(f"Сохранение текущей таблицы заняло {end_time - start_time:.2f} секунд. Файл сохранен в: {saved_path}")

if __name__ == "__main__":
    main()
```

Рисунок 6. коренной файл main.py для работы с Pandas

В коренной файл выводится информация из titanic.csv , обработанная, отсортированная и удаленная через data_operations.py I redacted.py I settings.py Выводящий все данные в файл .db формата и выводящий окно таблицы: Кол-во пассажиров на возраст.

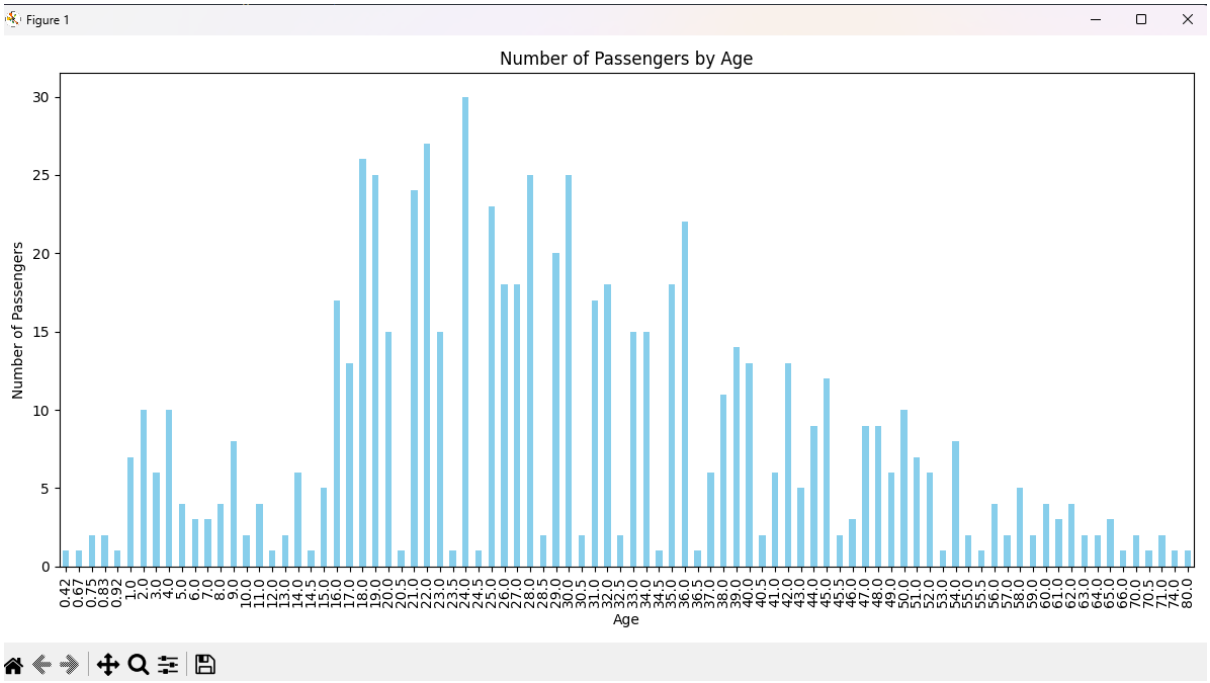


рисунок 7. таблица сделанная в основном через Pandas

```
Инициализация DataProcessor...
Инициализация DataProcessor заняла 0.01 секунд.
Сортировка данных по возрасту...
Сортировка данных заняла 0.01 секунд. Сохранено в: A:/Abobapy/DiplomProject/MainProject/Pandas/settings\sorted_by_Age.csv
Добавление возрастных групп...
Добавление возрастных групп заняло 0.01 секунд. Сохранено в: A:/Abobapy/DiplomProject/MainProject/Pandas/settings\edited_with_agegroup.csv
Построение детализированного графика по возрасту...
```

Рисунок 8. Затраченное время обработки информации

Dask.

— это библиотека для параллельных вычислений в Python, которая расширяет возможности работы с данными, предоставляемые Pandas, NumPy и другими инструментами, за счет масштабирования на кластеры и распределенные системы.

Ключевые особенности Dask:

1. **Dask DataFrame** — аналог Pandas DataFrame, но поддерживает работу с большими наборами данных, которые не помещаются в память. Dask разбивает данные на небольшие фрагменты (части) и выполняет операции параллельно.
2. **Dask Array** — аналог NumPy Array для работы с большими многомерными массивами.
3. **Dask Bag** — структура для работы с неструктурированными или слабо структурированными данными, похожими на списки Python.

4. **Dask Delayed** — позволяет декорировать функции для отложенных вычислений, а затем управлять этими вычислениями параллельно.

Преимущества Dask:

- **Масштабируемость** — Dask работает как на локальном компьютере, так и на кластере, обеспечивая высокую производительность и обработку огромных объемов данных.
- **Интеграция** — легко интегрируется с Pandas и NumPy, что упрощает перенос существующего кода на Dask.
- **Гибкость** — поддерживает как большие массивы данных, так и сложные вычислительные графы.

Dask подходит для обработки данных, которые не умещаются в память, и для случаев, когда необходима параллельная обработка.

```python
from settings.settings import create_processor, DataFilter, DataAggregator, MissingValueHandler

def main():
    # Создание экземпляра обработчика данных
    processor = create_processor()

    # Пример использования функций
    processor.remove_column('Name')
    processor.rename_column('Sex', 'Gender')
    processor.sort_by_column('Age')

    # Фильтрация данных
    filter_processor = DataFilter(processor.ddf)
    filtered_data = filter_processor.filter_by_condition('Gender', 'female')

    # Сохранение отфильтрованных данных в новый файл
    output_file_filtered = r"A:\Abobapy\DiplomProject\MainProject\Dask\titanic_filtered.csv"
    filter_processor.ddf = filtered_data  # Обновляем Dask DataFrame с отфильтрованными данными
    processor.save_to_csv(output_file_filtered)

    # Аггрегация данных
    aggregator = DataAggregator(processor.ddf)
    group_data = aggregator.group_by('Gender')
    print("Группировка по полу:")
    print(group_data)

    # Обработка пропущенных значений
    missing_value_handler = MissingValueHandler(processor.ddf)
    missing_value_handler.fill_missing_values('Age', 30)  # Заполняем пропущенные значения в 'Age' значением 30
    missing_value_handler.drop_missing_values()  # Удаляем строки с пропущенными значениями

    # Сохранение обработанных данных в новый файл
    output_file_path = r"A:\Abobapy\DiplomProject\MainProject\Dask\titanic_processed.csv"
    processor.save_to_csv(output_file_path)

    # Получение сводной информации
    summary = processor.get_summary()
    print("Сводная информация о данных:")
    print(summary)

if __name__ == "__main__":
    main()
```

Рисунок 9. Содержимого коренного файла для работы с Dask

Почти моментально Dask выполнил работу на Редактирование, Удаление, Сортировку, и вывел всё в отдельные файлы, с измененными базами данных.

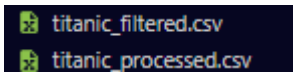
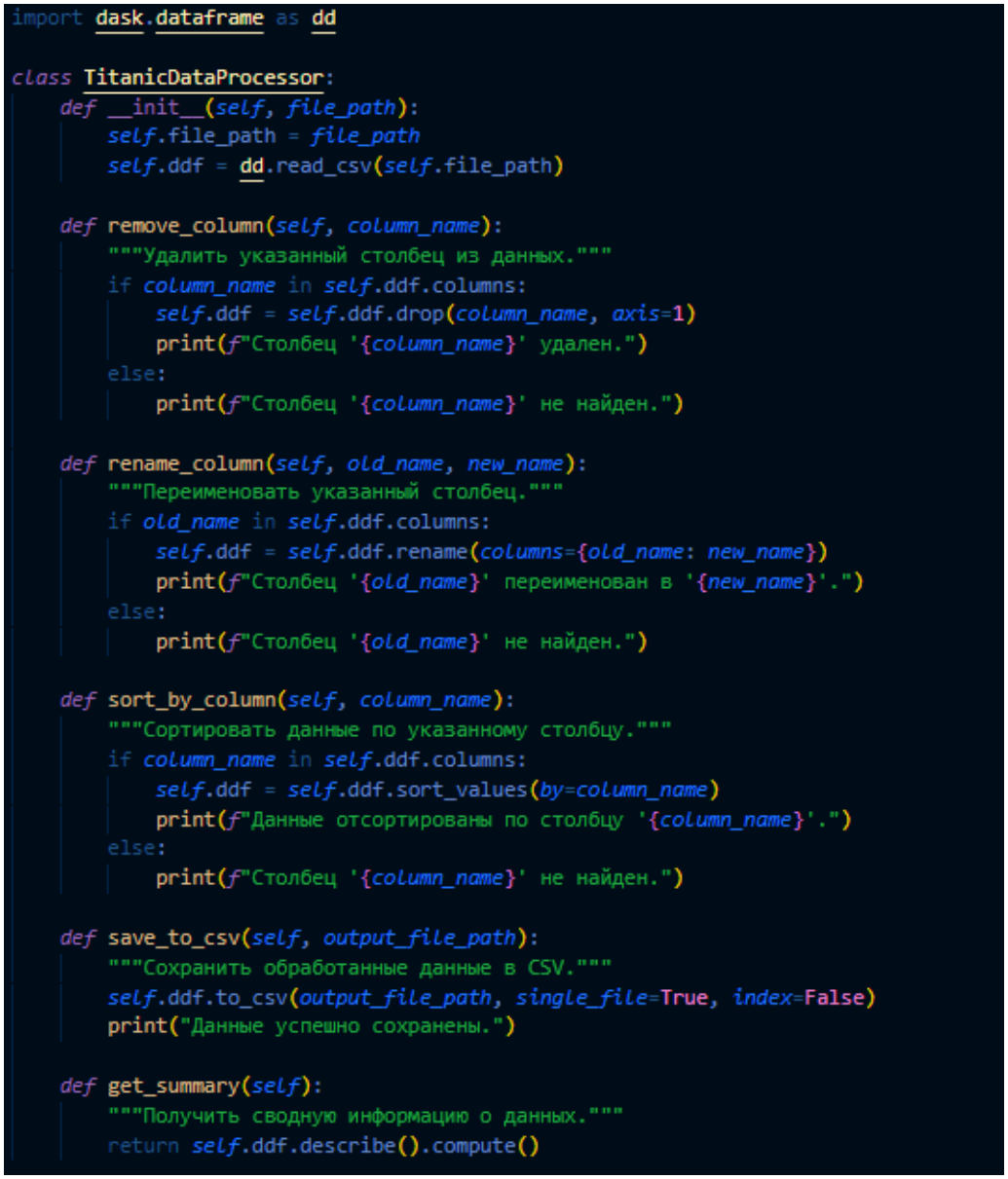


Рисунок 10. Измененные базы данных находящиеся в коренной папке Dask

```
Столбец 'Name' удален.
Столбец 'Sex' переименован в 'Gender'.
Данные отсортированы по столбцу 'Age'.
Отфильтровано по условию 'Gender == female'.
Данные успешно сохранены.
Данные сгруппированы по столбцу 'Gender'.
Группировка по полу:
Gender
female    314
male      577
dtype: int64
Пропущенные значения в 'Age' заполнены значением '30'.
Удалены строки с пропущенными значениями. Осталось строк: 202 из 891.
Данные успешно сохранены.
Сводная информация о данных:
       PassengerId    Survived      Pclass         Age       SibSp       Parch        Fare
count   891.000000  891.000000  891.000000  891.000000  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642   29.758889    0.523008    0.381594   32.204208
std     257.353842    0.486592    0.836071   13.002570    1.102743    0.806057   49.693429
min       1.000000    0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%     223.500000    0.000000    2.000000   22.000000    0.000000    0.000000    7.910400
50%     446.000000    0.000000    3.000000   30.000000    0.000000    0.000000   14.454200
75%     668.500000    1.000000    3.000000   35.000000    1.000000    0.000000   31.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000    6.000000  512.329200
```

Рисунок 11. Часть вывода в консоль со всеми махинациями.



```python
import dask.dataframe as dd

class TitanicDataProcessor:
    def __init__(self, file_path):
        self.file_path = file_path
        self.ddf = dd.read_csv(self.file_path)

    def remove_column(self, column_name):
        """Удалить указанный столбец из данных."""
        if column_name in self.ddf.columns:
            self.ddf = self.ddf.drop(column_name, axis=1)
            print(f"Столбец '{column_name}' удален.")
        else:
            print(f"Столбец '{column_name}' не найден.")

    def rename_column(self, old_name, new_name):
        """Переименовать указанный столбец."""
        if old_name in self.ddf.columns:
            self.ddf = self.ddf.rename(columns={old_name: new_name})
            print(f"Столбец '{old_name}' переименован в '{new_name}'.")
        else:
            print(f"Столбец '{old_name}' не найден.")

    def sort_by_column(self, column_name):
        """Сортировать данные по указанному столбцу."""
        if column_name in self.ddf.columns:
            self.ddf = self.ddf.sort_values(by=column_name)
            print(f"Данные отсортированы по столбцу '{column_name}'.")
        else:
            print(f"Столбец '{column_name}' не найден.")

    def save_to_csv(self, output_file_path):
        """Сохранить обработанные данные в CSV."""
        self.ddf.to_csv(output_file_path, single_file=True, index=False)
        print("Данные успешно сохранены.")

    def get_summary(self):
        """Получить сводную информацию о данных."""
        return self.ddf.describe().compute()
```

Рисунок 12. Часть кода написанного в Dask\settings\settings.py

Seaborn.

— это мощная библиотека для визуализации данных в Python, построенная на базе Matplotlib. Она делает процесс создания графиков более простым и элегантным, предоставляя высокоуровневые интерфейсы для создания красивых статистических графиков.

Основные особенности Seaborn:

1. **Статистические графики**: Seaborn предоставляет удобные инструменты для построения графиков, таких как диаграммы рассеяния (scatter plots), гистограммы, плотности распределения и тепловые карты (heatmaps). Кроме того, Seaborn может визуализировать связи между переменными через парные графики (pairplots) и линейные модели (regplot).
2. **Встроенные данные**: Seaborn поддерживает работу с DataFrame, что делает её отличным инструментом для работы с табличными данными, особенно в связке с библиотекой Pandas.
3. **Простая настройка**: Seaborn имеет привлекательный стиль по умолчанию, но также поддерживает тонкую настройку, позволяя изменять цвета, шрифты и другие элементы графиков.
4. **Интеграция с Matplotlib**: Seaborn легко интегрируется с Matplotlib, что позволяет использовать его возможности для точной настройки графиков.

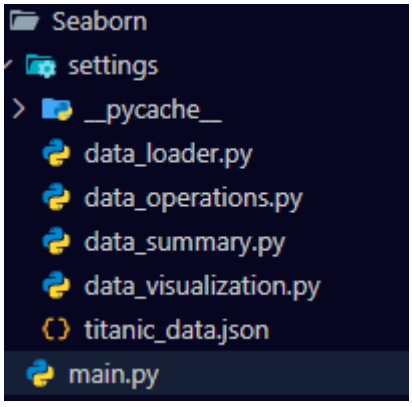


Рисунок 13. Файловая структура проекта на Seaborn

```python
from settings.data_loader import DataLoader
from settings.data_summary import DataSummary
from settings.data_visualization import DataVisualization
from settings.data_operations import DataOperations
import time

def main():
    start_time = time.time()

    # Загрузка данных из оригинального CSV
    csv_path = r'A:\Abobapy\DiplomProject\MainProject\Data\titanic\titanic.csv'
    json_path = r'A:\Abobapy\DiplomProject\MainProject\Seaborn\settings\titanic_data.json'

    loader = DataLoader(csv_path, json_path)
    data = loader.load_csv()  # Теперь это DataFrame

    # Сохраняем данные в JSON формате в папке Seaborn
    loader.save_json(data)

    # Анализ данных
    summary = DataSummary(data)
    print(f"Total passengers: {summary.total_passengers()}")
    print(f"Passengers by class: {summary.count_by_class()}")

    # Визуализация с использованием Seaborn
    visualizer = DataVisualization(data)

    # Генерация гистограммы по классам
    visualizer.generate_bar_chart()

    # Генерация двумерного графика по возрасту и количеству пассажиров
    visualizer.generate_age_passenger_chart()

    # Операции с данными
    operations = DataOperations(data)

    # Добавление новой записи (DataFrame уже использует строки вместо списков)
    new_record = {
        'Pclass': '3', 'Survived': '1', 'Name': 'New Passenger', 'Age': 29
    }
    operations.add_record(new_record)

    # Редактирование существующей записи
    operations.edit_record(0, 'Age', 30)

    # Удаление записи
    operations.delete_record(1)

    # Сортировка данных
    sorted_data = operations.sort_by_column('Age')
    print(f"Sorted data by Age: {sorted_data}")

    # Сохранение изменённых данных в JSON в папку Seaborn
    loader.save_json(data)

    # Остановка таймера и вывод времени выполнения
    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Execution time: {execution_time:.2f} seconds")

if __name__ == "__main__":
    main()
```

Рисунок 14. Коренной файл с работой Seaborn

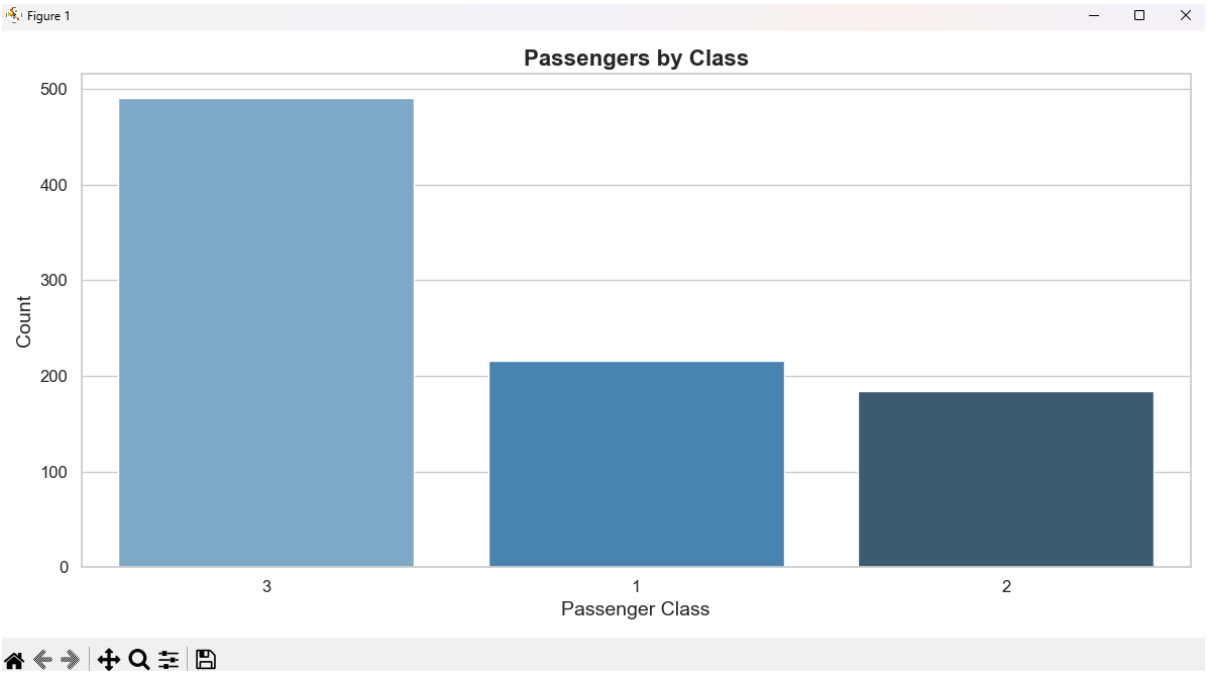


Рисунок 15. 1 часть выполнения кода выводящую в окно с кол-во пассажиров и классами билета.

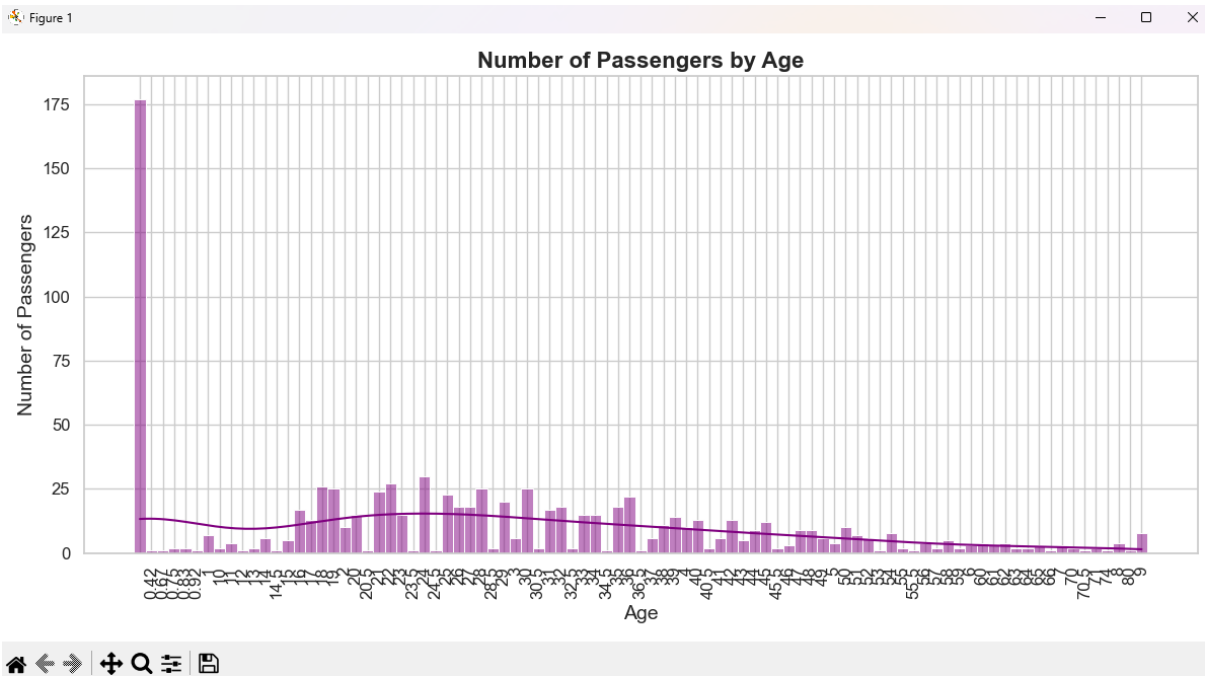


Рисунок 16. 2 часть кода где выводится информация в таблицу кол-во пассажиров по возрасту.

Заключение:

Pandas

Описание: Pandas — это библиотека Python для работы с табличными данными. Она предоставляет мощные структуры данных, такие как DataFrame, и функции для анализа и манипуляции данными.

- **Плюсы**:
  - Простота использования и интуитивный API
  - Широкий набор функций для обработки данных
  - Хорошо подходит для небольших и средних наборов данных
- **Минусы**:
  - Медленно работает с большими данными (неэффективно использует память и CPU)

Dask

Описание: Dask — это библиотека для масштабируемой обработки данных, которая работает с большими наборами данных и распределенными системами. Dask использует "ленивую" оценку и может параллелизировать задачи.

- **Плюсы**:
  - Масштабируемость на кластеры и большие наборы данных
  - Совместимость с Pandas и Numpy
  - Поддержка распределенных вычислений
- **Минусы**:
  - Требует настройки для больших проектов
  - Может быть сложнее для начинающих пользователей

Seaborn

Описание: Seaborn — библиотека для визуализации данных, построенная на основе Matplotlib. Она упрощает создание красивых и статистически осмысленных графиков.

- **Плюсы**:
  - Легкость создания сложных графиков
  - Глубокая интеграция с Pandas

    - Визуально привлекательные графики
- **Минусы**:
    - Не подходит для обработки данных (только визуализация)
    - Ограниченные возможности настройки по сравнению с Matplotlib

Краткое сравнение:

- **Pandas**: Отлично подходит для обработки данных, но не справляется с большими объемами.
- **Dask**: Хорош для больших данных и распределенных вычислений, но требует больше настроек.
- **Seaborn**: Сильна в визуализации, но не обрабатывает данные.

Выбор зависит от задачи: Pandas для небольших наборов данных, Dask для масштабируемости, Seaborn для визуализации.