Департамент образования и науки города Москвы

Государственное автономное образовательное учреждение высшего образования города Москвы

«Московский городской педагогический университет»

Институт цифрового образования

Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

Проектный практикум по разработке ETL-решений

**Лабораторная работа 3.1**

**Интеграция данных из нескольких источников. Обработка и согласование данных из разных источников**

Выполнила: Сергеева А. И., группа: АДЭУ-211

Преподаватель: Босенко Т.М.

Москва

2025

**Цель работы:** получить практические навыки интеграции, обработки и согласования данных из различных источников с использованием Python и его библиотек.

**Задачи:**

- Изучить методы чтения данных из разных источников.

- Освоить техники обработки и очистки данных.

- Научиться согласовывать данные из разных источников.

- Реализовать сохранение обработанных данных.

**Программное обеспечение:**

- Python 3.12+

- Библиотеки:

- pandas,

- numpy,

- sqlalchemy,

- psycopg2,

- mysql-connector-python,

- openpyxl.

**Ход работы:**

**Вариант 12.** Интеграция данных об оборудовании:

- PostgreSQL база данных инвентаризации.

- Excel файл с графиком обслуживания.

- CSV файл с историей ремонтов.

Задача: создать систему учета технического обслуживания.

Для начала необходимо было подготовить тестовые данные, которые были сгенерированы с помощью Python. Для экспорта данных в PostgreSQL необходимо для начала создать нужную таблицу и базу, чтобы потом заполнить ее значениями, создание базы и таблицы показано на рисунках 1-2. Для генерации данных необходима библиотека Faker. Ее часто используют для наполнения тестовыми данными. Таблица по инвентаризации содержит сведения о наименовании оборудования, о типе оборудования, дате покупки сроке гарантии.

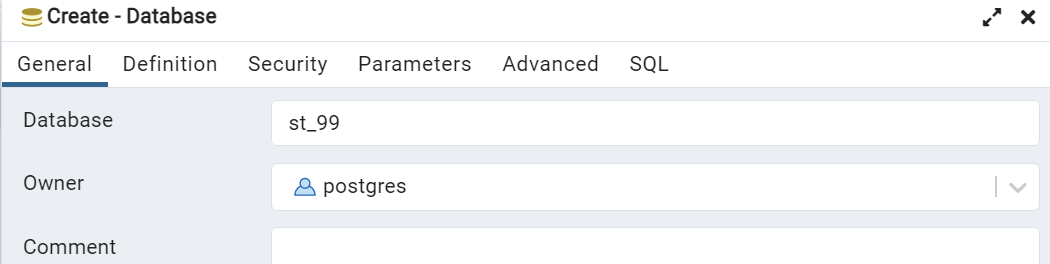


Рисунок 1 – Создание базы данных

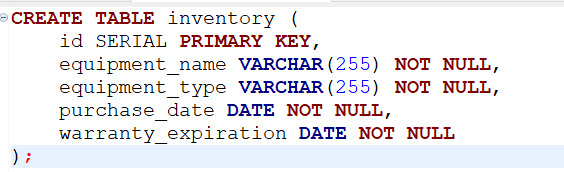


Рисунок 2 – Создание в PostgreSQL таблицы по инвентаризации

Была выполнена установка необходимых библиотек на рисунке 3.

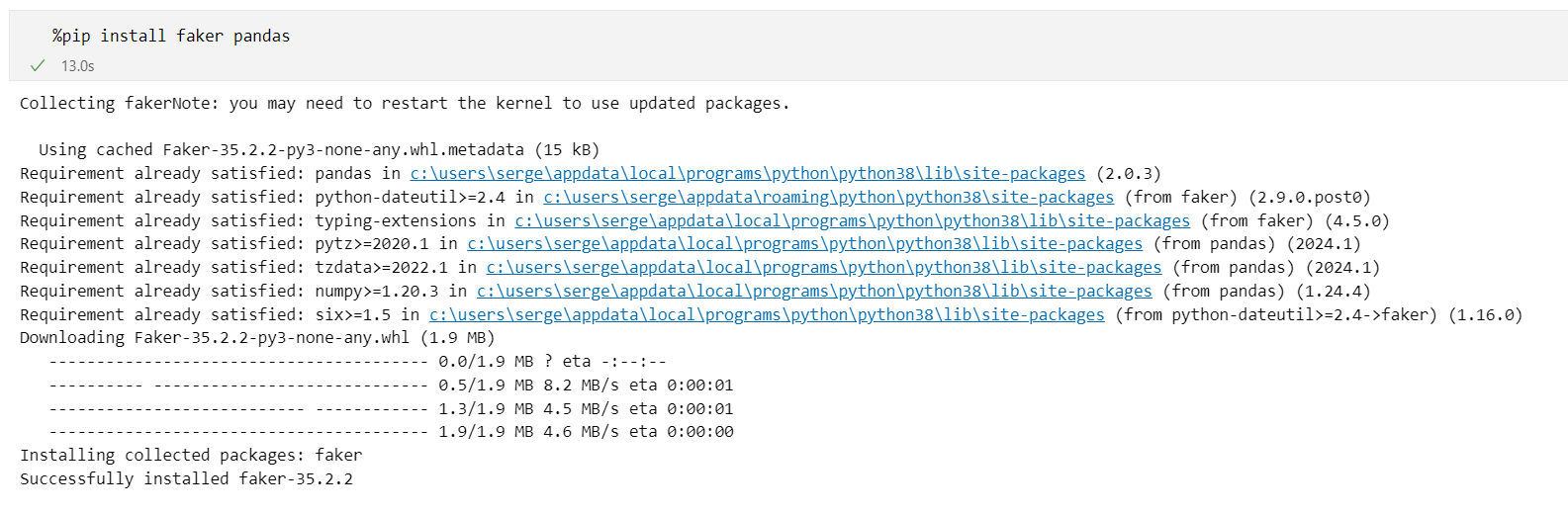


Рисунок 3 – Установка библиотек

Импорт нужных библиотек для работы с базой данных PostgreSQL, для генерации данных, для работы с типом даты и настройки подключения к базе на рисунке 4.

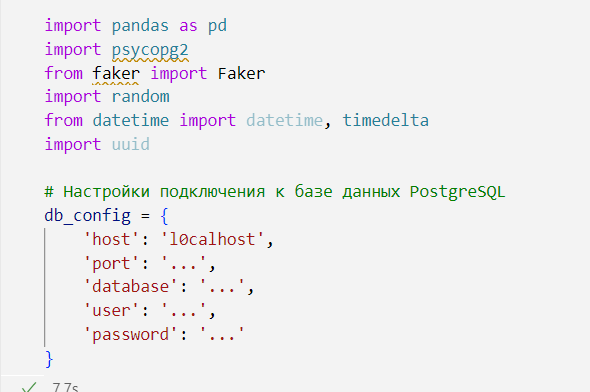


Рисунок 4 – Импорт библиотек и подключение к базе данных PostgreSQL

Для генерации сбыли созданы пустые списки для данных по инвентаризации, графику обслуживания и по ремонтам. Наименование генерируется подбором слова случайным образом, тип продукта выбирается рандомно между сервером, пк, принтером и сканером, дата покупки берется случайно между 2023 и 2025, срок гарантии исчисляется от 1 года до 3 лет в днях. Для генерации данных по обслуживанию берется период от месяца до 3, выбирается тип обслуживания между регулярным, чисткой систем охлаждения и обновлением ПО, генерируется случайное предложение на 8 слов в заметках. При генерации данных по ремонтам аналогично: тип ремонта выбирается между заменой детали, системной ошибкой, стоимость ремонта от 500 до 20.000 рублей с 2 знаками после запятой. Данная генерация показана на рисунке 5.



Рисунок 5 – Генерация данных

Далее необходимо было сгенерированные данные об обслуживании загрузить в PostgreSQL на рисунке 6.



Рисунок 6 – Загрузка сгенерированных данных в PostgreSQL

Через Dbeaver была выполнена проверка по загруженным данным на рисунке 7.

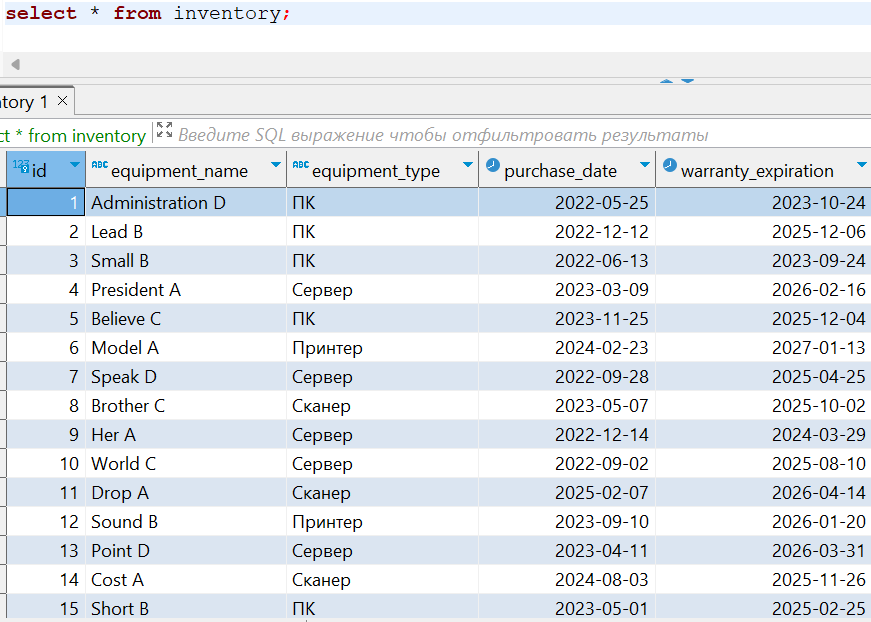


Рисунок 7 – Загруженные данные в PostgreSQL

Далее данные о ремонте загружаются в CSV файл, что показано на рисунке 8.

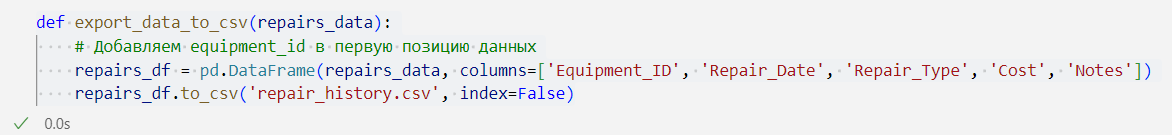


Рисунок 8 – Загрузка сгенерированных данных в CSV файл

Загрузка в Excel-файл показана на рисунке 9.

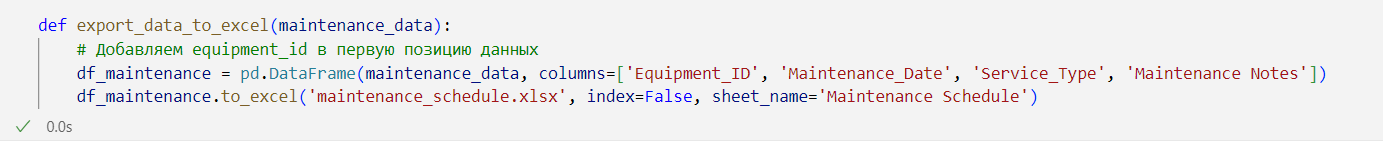


Рисунок 9 – Генерация данных в Excel-файл

Были выполнены ранее созданные методы и сгенерировано данных на 500 строк на рисунке 10.

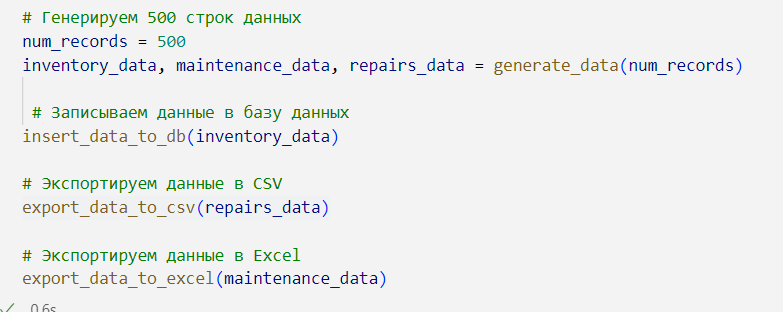


Рисунок 10 – Генерация данных и выполнение методов

Теперь данные можно прочитать и снова работать с ними, как показано на рисунке 11.



Рисунок 11 – Чтение 3 таблиц

Изучение типов и их оценка на рисунках 12-13. Необходимо было для даты ремонта, обслуживания и сроку службы задать тип данных дата, остальные можно не трогать.



Рисунок 12 – Изменение типа данных для таблицы по обслуживанию оборудования

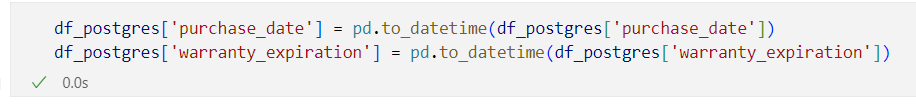


Рисунок 13 – Изменение типа данных для таблицы по инвентарю

Далее проведена описательная статистика на рисунках 14-16. Данные с датой покупки от 2023 до 2025 года, срок обслуживания от 2025 до 2028 года. Срок обслуживания от 2023 до 2025 года. Средняя стоимость ремонта 9873 рубля.

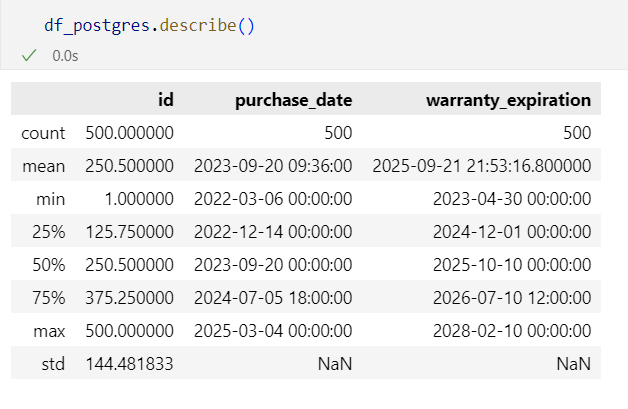


Рисунок 14 – Описательная статистика для таблицы по инвентаризации

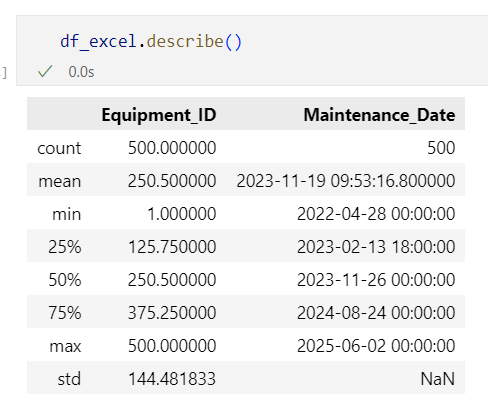


Рисунок 15 – Описательная статистика для таблицы по обслуживанию

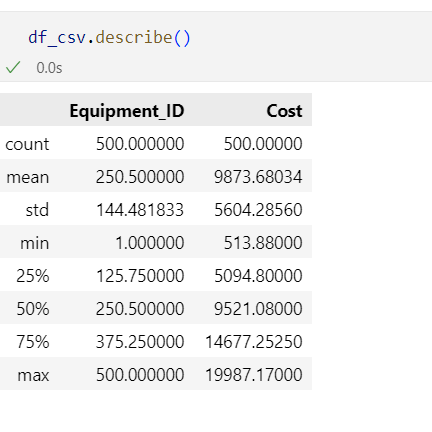


Рисунок 16 – Описательная статистика по таблице с ремонтами

Далее проводится удаление дублей на рисунке 17.



Рисунок 17 – Удаление дублей

После вышеперечисленных шагов проводилась проверка на поиск нулевых значений на рисунке 18. Нигде нет нулевых значений.

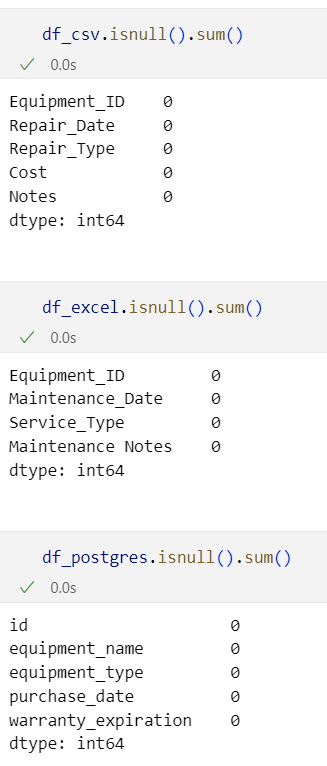


Рисунок 18 – Проверка таблиц на нулевые значения

Далее выполнено объединение очищенных данных, как показано на рисунке 19.

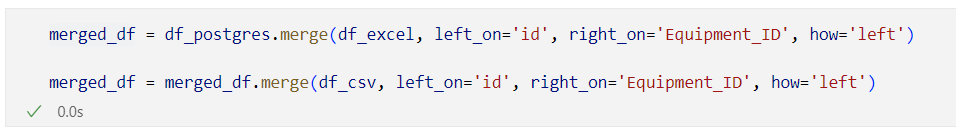


Рисунок 19 – Объединение таблиц

Была выполнена настройка фильтра по типу оборудования, создавался выпадающий список с возможностью выбора только одного значения на рисунке 20.

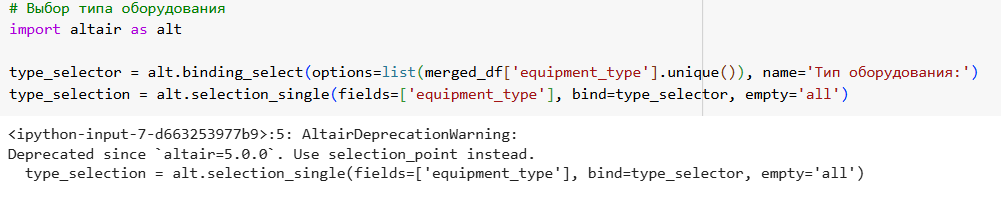


Рисунок 20 – Настройка фильтрации для будущих графиков

Подсчет количества ремонтов по типам ремонта и оборудования на рисунке 21.



Рисунок 21 - Подсчет количества ремонтов по типам ремонта и оборудования

Гистограмма частоты ремонтов показана на рисунках 22-23. Чтобы не было сложностей при работе с Altair выполнен переход от локального VS code к Google Colaboratory. Среди ПК одинаково заменяют детали и разбираются ссистемными ошибками. После обслуживания серевера чаще прибегают к замене детали, для принтеров часты системные ошибки, для сканера также.

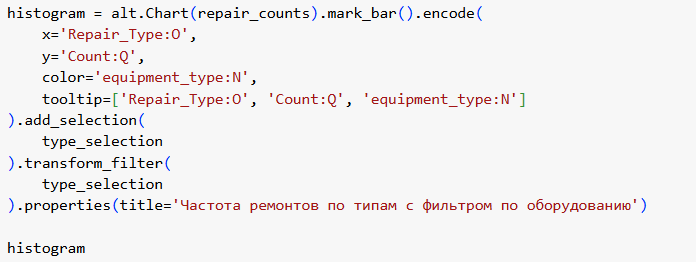


Рисунок 22 – Построение графика по ремонтам

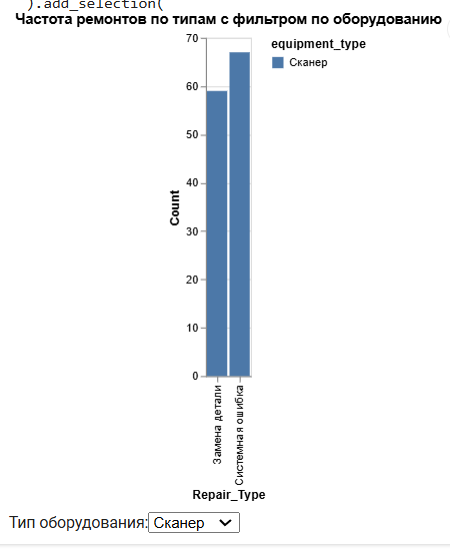


Рисунок 23 – График частоты ремонтов

Расчет суммы затрат по типам оборудования на рисунке 24.

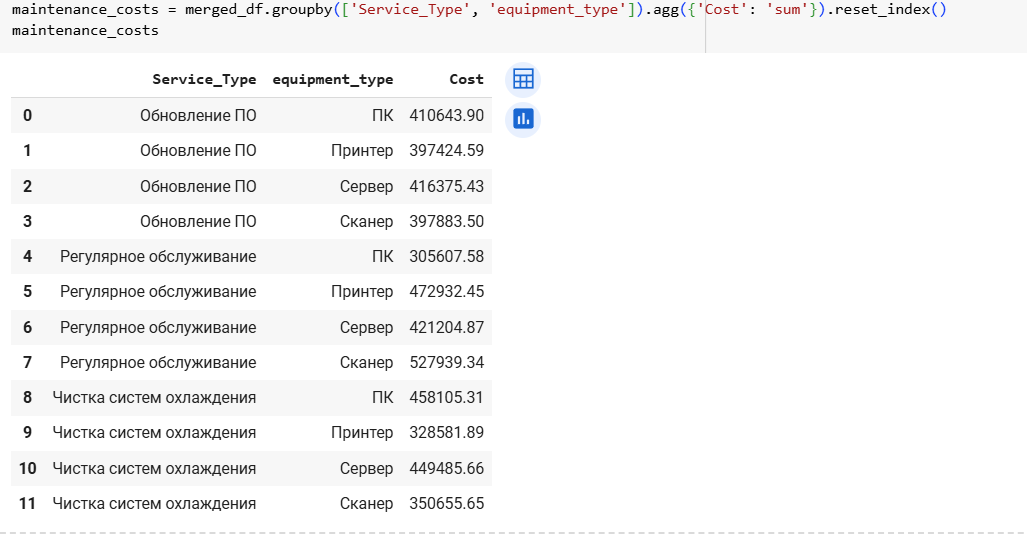


Рисунок 24 - Расчет суммы затрат по типам оборудования

Столбчатая диаграмма с затратами на обслуживание по типам оборудования на рисунке 25. Лидируют случаи регулярного обслуживания. Сервера чаще нуждаются в обновлении, для сканеров нужнее регулярное обслуживание, а для систем охлаждения чаще ПК.

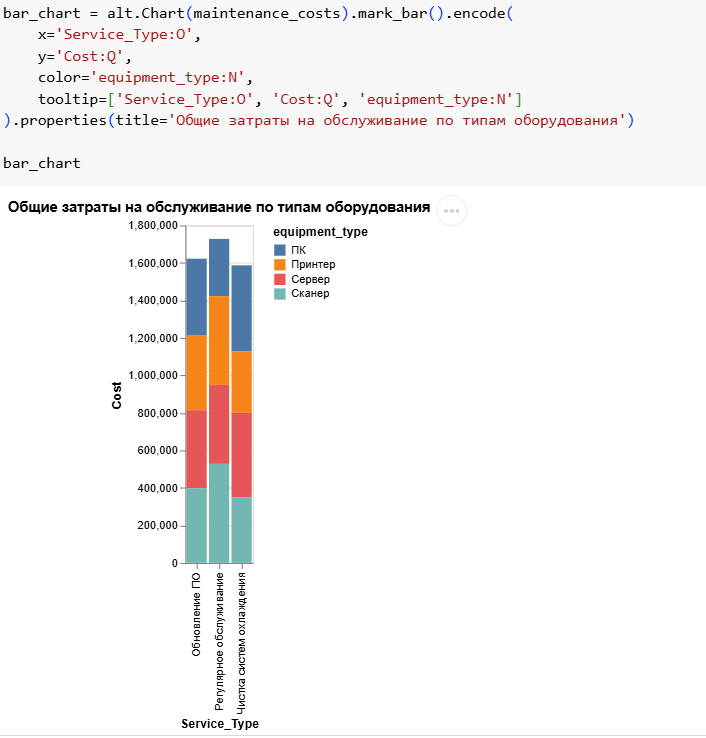


Рисунок 25 - Столбчатая диаграмма с затратами на обслуживание по типам оборудования

Линейный график с трендом затрат по времени построен на рисунке 26.

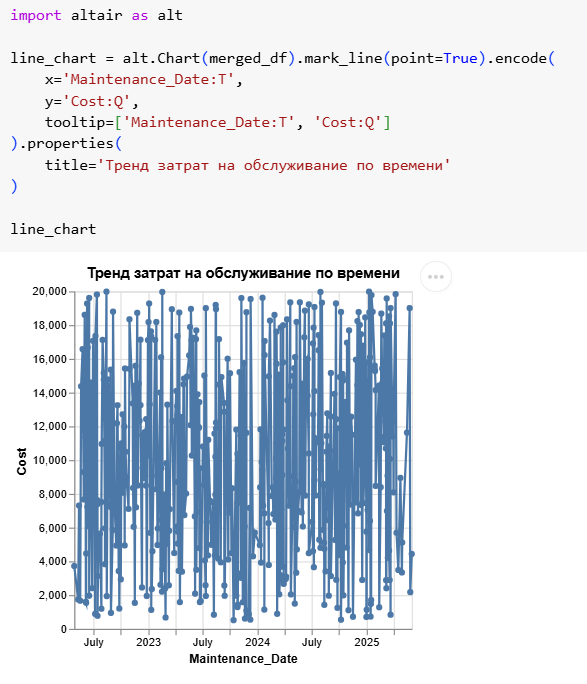


Рисунок 26 - Линейный график с трендом затрат по времени

Далее был выполнен расчет стоимости по дате и типу оборудования на рисунках 27-28, чтобы построить тепловую карту. У серверов ярче выражена зависимость от срока обслуживания.

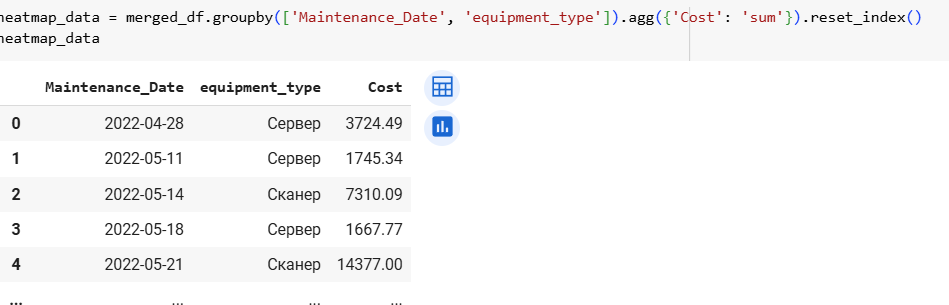


Рисунок 27 - Расчет стоимости по дате и типу оборудования

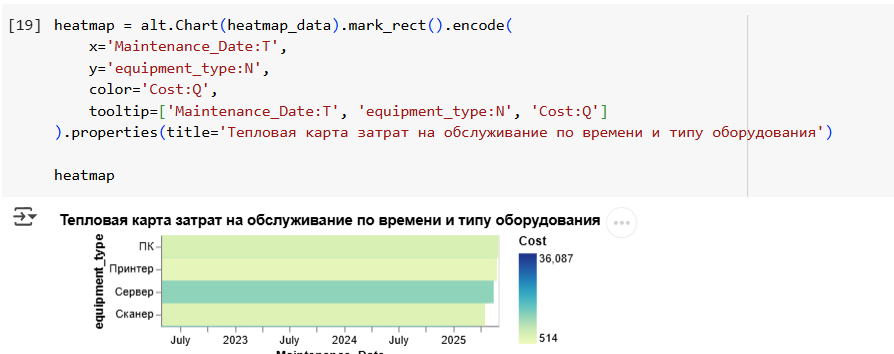


Рисунок 28 - Тепловая карта для визуализации затрат на обслуживание по времени и типу оборудования

Конечным пунктом является выгрузка объединённой таблицы в MySQL. Для этого настраивается подключение и указывается таблица, в которую будут загружены данные. Данные операции над данным и их проверка показаны рисунках 29-.

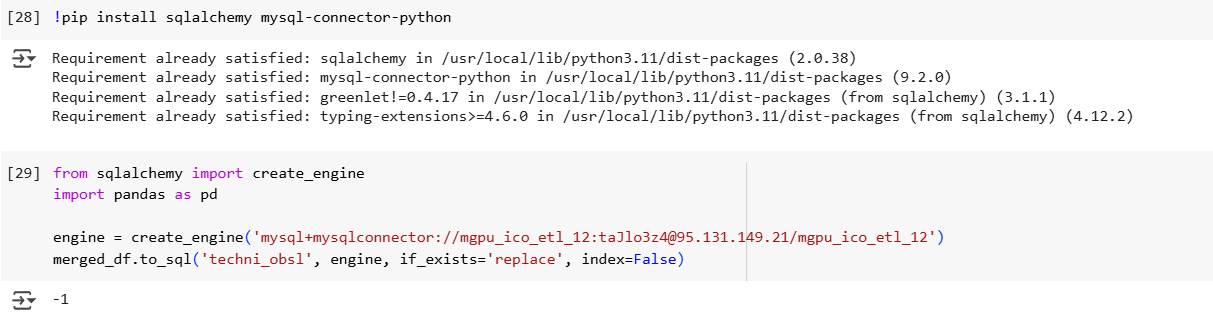


Рисунок 29 – Выгрузка данных в MySQL

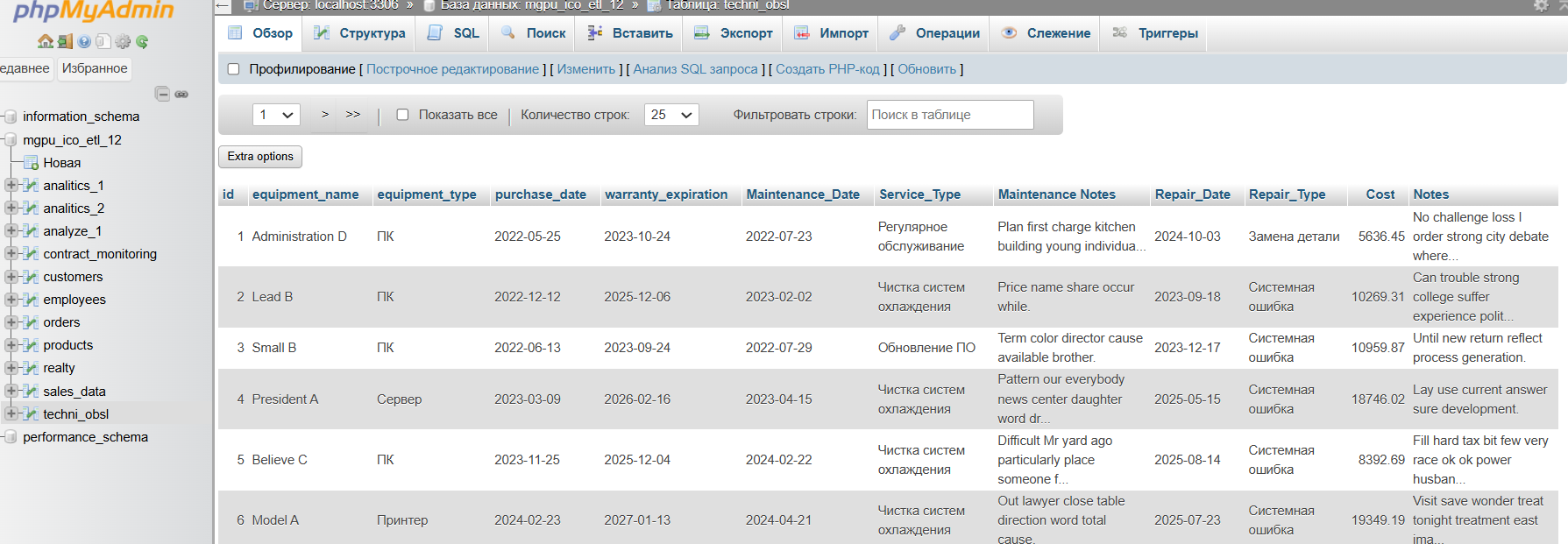


Рисунок 30 – Выгруженные данные в MySQL

При построении архитектуры учитывались слои источников данных (выступает PostgreSQL с данными по инвентаризации, CSV file с данными по ремонтам, Excel file с данными по обслуживанию), в слое сбора данных данные из базы обрабатывались в СУБД Dbeaver и в PgAdmin, а в дальнейшем, как и другие данные, обрабатывались с помощью python, а затем с помощью sqlalchemy выгружались в слой хранения данных, где они выгружались в MySQL. В рамках слоя обработки данных использовались такие инструменты, как Google colab для решения вопросов установки библиотек и версионности и VS code, позволяющий работать с локальной базой данных. В рамках слоя аналитики применялись ключевые библиотеки в анализе. С архитектурой решения можно ознакомиться на рисунке 31.

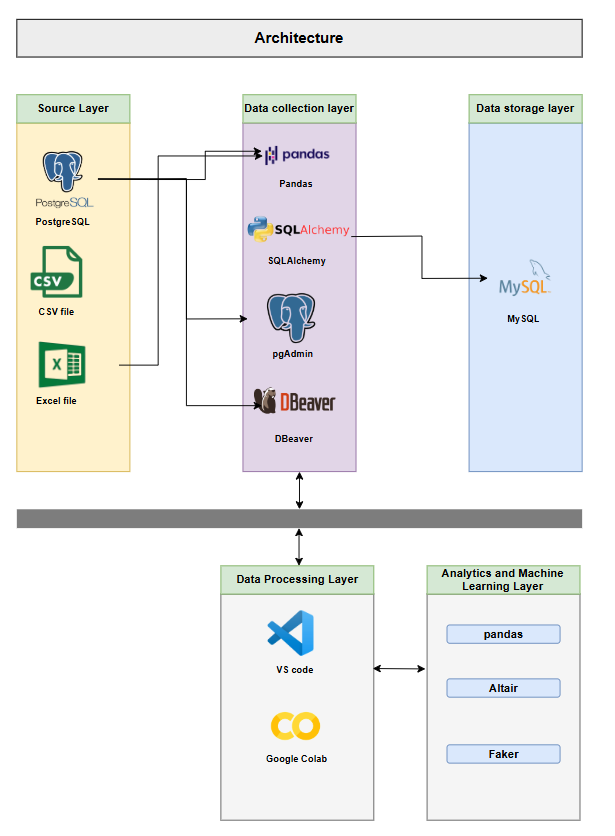


Рисунок 31 – Архитектура системы учета технического обслуживания

**Общий вывод**:

В ходе лабораторной работы были опробованы 3 разных способа подключения к данным, рассмотрены особенности для каждого метода. А также протестирована генерация данных с помощью одной из наиболее популярных библиотек для этого (Faker). Данные были обработаны и очищены, проанализированы и построены графики в Altair с возможностью фильтрации. Единственной сложностью в работе являлось то, что пришлось работать с локальной и нелокальной базами. Поставленные цель и задачи были выполнены.

**Ответы на контрольные вопросы:**

1. Методы чтения данных, предоставляемые Pandas.

* Чтение из CSV файлов: df = pd.read\_csv('file.csv')
* Чтение из Excel файлов: df = pd.read\_excel('file.xlsx', sheet\_name='Sheet1')
* Чтение из баз данных: from sqlalchemy import create\_engine

engine = create\_engine('postgresql://user:password@host/dbname')

df = pd.read\_sql('SELECT \* FROM table\_name', con=engine)

* Чтение из JSON файлов: df = pd.read\_json('file.json')
* Чтение из HTML страниц: df\_list = pd.read\_html('http://example.com')

2. Обработка пропущенных значений.

* Удаление строк с пропущенными значениями: df.dropna(inplace=True)
* Удаление столбцов с пропущенными значениями: df.dropna(axis=1, inplace=True)
* Заполнение пропущенных значений значением: df.fillna(value=0, inplace=True)
* Замена пропущенных значений: df['column\_name'].replace(to\_replace=np.nan, value=0, inplace=True)

3. Типы объединения данных.

* Inner Join: Возвращает только те строки, которые присутствуют в обоих DataFrame.
* Outer Join: Возвращает все строки из обоих DataFrame, заполняет отсутствующие значения NaN.
* Left Join: Возвращает все строки из первого (левого) DataFrame и соответствующие строки из второго (правого). Если в правом DataFrame нет соответствующих строк, то создаются NaN значения.
* Right Join: Возвращает все строки из второго (правого) DataFrame и соответствующие строки из первого. Аналогично left join, но с перевернутыми DataFrame.

4. Проверка качества объединения данных.

* Проверка пропущенных значений после объединения: print(result.isnull().sum())
* Сравнение размеров DataFrame до и после объединения: print(df1.shape, df2.shape, result.shape)
* Анализ уникальных значений в объединённых колонках: print(result['key\_column'].unique())
* Проверка на дубликаты после объединения: print(result.duplicated().sum())

5. Методы дедупликации данных.

* Удаление полных дубликатов: df.drop\_duplicates(inplace=True)
* Удаление дубликатов по определённым столбцам: df.drop\_duplicates(subset=['column1', 'column2'], inplace=True)
* Сохранение первого или последнего вхождения дубликатов: df.drop\_duplicates(keep='first', inplace=True)

df.drop\_duplicates(keep='last', inplace=True)

* Поиск дубликатов: duplicates = df[df.duplicated()]
* print(duplicates)