**СОДЕРЖАНИЕ**

[Введение 3](#_Toc197952741)

[1. Теоретическая часть 4](#_Toc197952742)

[1.1 Описание исследуемой области 4](#_Toc197952743)

[1.2 Информация об источнике и о наборе данных 4](#_Toc197952744)

[1.3 Описание процесса сбора данных 7](#_Toc197952745)

[2 Практическая часть 11](#_Toc197952746)

[2.1 Предобработка данных 11](#_Toc197952747)

[2.2 Анализ данных 25](#_Toc197952748)

[2.3 Обработка результатов анализа 32](#_Toc197952749)

[Заключение 36](#_Toc197952750)

[Список используемых источников 37](#_Toc197952751)

[Приложения 38](#_Toc197952752)

[Приложение А 39](#_Toc197952753)

[Приложение Б 41](#_Toc197952754)

# Введение

Загрязнение воздуха является глобальной проблемой, которая представляет серьезную угрозу здоровью и благополучию людей. Воздействие загрязнения воздуха связано с целым рядом проблем со здоровьем, включая сердечно-сосудистые заболевания, инсульт и рак легких. Воздействие загрязнения воздуха на здоровье населения отражается в тревожной статистике смертей, вызванных им. Согласно последним исследованиям, загрязнение воздуха ежегодно приводит к миллионам смертей во всем мире.

В данной работе рассматривается жизненный цикл анализа данных включающий сбор, хранение, обработку и визуализацию информации о загрязнении воздуха и связанных с ним заболеваниях. Используются технологии Apache Sqoop, Hive, Spark и MariaDB, что позволяет эффективно обрабатывать большие объёмы данных и получать аналитические выводы.

Цель работы — разработка конвейера данных на основе технологий Big Data для анализа взаимосвязи между загрязнением воздуха и уровнем заболеваемости. В качестве среды для развертывания инструментов используется VirtualBox.

Из поставленной цели вытекают следующие задачи:

* Собрать данные из открытых источников (Kaggle).
* Составить конвейер для сбора и передачи данных.
* Выдвинуть гипотезы и проверить их;
* Визуализировать полученные результаты для выявления взаимосвязей между загрязнением воздуха и уровнем заболеваемости.

# Теоретическая часть

## Описание исследуемой области

Анализ влияния загрязнения воздуха на уровень заболеваний представляет собой важное направление в экологической эпидемиологии и здравоохранении. Исследуемая область охватывает взаимосвязь между качеством атмосферного воздуха и показателями здоровья населения, включая:

Область исследования:

* Основные загрязнители: PM2.5, PM10, диоксид азота (NO2), диоксид серы (SO2), озон (O3);
* Заболевания: астма, сердечно-сосудистые, онкологические;
* Географический охват: глобальный анализ с акцентом на урбанизированные территории.

Актуальность исследования обусловлена:

1. Ростом урбанизации и промышленного производства;
2. Увеличением нагрузки на экосистемы;
3. Необходимостью доказательной базы для принятия управленческих решений.

В контексте курсовой работы анализ данных позволяет:

* Строить прогнозные модели;
* Оптимизировать систему экологического мониторинга.

## Информация об источнике и о наборе данных

В данной работе набор данных "Air Pollution" был взят с платформы Kaggle и содержит информацию о смертности, связанной с загрязнением воздуха, по различным странам и регионам за период с 1990 по 2019 год. Данные включают в себя показатели смертности от общего загрязнения воздуха, загрязнения атмосферного воздуха, загрязнения воздуха внутри помещений (из-за использования твердых видов топлива), а также уровень смертности на 100 000 населения. Файл представлен в формате csv.

Описательная статистика набора данных.

Общая информация:

* Количество записей (строк): 6240
* Количество атрибутов (столбцов): 8
* Период данных: 1990 – 2019 годы
* Количество уникальных стран/регионов: 208

Описание атрибутов:

1. id — целочисленный;

* Уникальный идентификатор записи;
* Диапазон значений от 1 до 6240.

1. Entity — строковый;

* Название страны;
* Примеры: Afghanistan, Albania, Algeria, и т.д.

1. Year — целочисленный;

* Год данных;
* Диапазон: 1990-2019.

1. Total Deaths for Air Pollution — целочисленный;

* Общее количество смертей от загрязнения воздуха;
* Описательная статистика:
* Среднее: 31676.41
* Минимум: 0.0
* Максимум: 1923489.0
* Стандартное отклонение: 164500.0

1. Total Deaths for Outdoor Air Pollution — целочисленный;

* Количество смертей от загрязнения атмосферного воздуха
* Описательная статистика:
* Среднее: 15839.07
* Минимум: 0.0
* Максимум: 1516904.0
* Стандартное отклонение: 91681.62

1. Total Deaths for Household Air Pollution from Solid Fuels — целочисленный;

* Количество смертей от загрязнения воздуха внутри помещений (из-за твердого топливо);
* Описательная статистика:
* Среднее: 16483.94
* Минимум: 0.0
* Максимум: 1329829.0
* Стандартное отклонение: 88591.77

1. Death Rate from Air Pollution Per 100000 — дробный.

* Коэффициент смертности на 100 тыс. человек;
* Описательная статистика:
* Среднее: 117.790615
* Минимум: 2.66
* Максимум: 527.89
* Стандартное отклонение: 91.438539

1. Deaths for Household Air Pollution from Solid Fuels (Percent) — дробный.

* Процент смертей от загрязнения воздуха в помещениях;
* Описательная статистика:
* Среднее: 5.210405
* Минимум: 0.0
* Максимум: 23.53
* Стандартное отклонение: 5.714961

Причина выбора данного источника:

* Данные структурированы и готовы к анализу.
* Включают широкий охват стран и временных периодов.
* Позволяют проводить сравнительный анализ между странами и регионами.
* Доступны на Kaggle, что упрощает их загрузку и использование.

## Описание процесса сбора данных

В современном анализе данных существует множество различных инструментов. В данной работе будем использовать инструменты, предназначенные для обработки и хранения больших массивов данных.

Для хранения данных будут использоваться инструменты MariaDB и Apache Hive. MariaDB — это система управления базами данных, которая является ответвлением или улучшенной копией MySQL.

Apache Hive — это SQL интерфейс доступа к данным для платформы Apache Hadoop. Hive позволяет выполнять запросы, агрегировать и анализировать данные используя SQL синтаксис. Для данных в файловой системе HDFS используется схема доступа на чтение, позволяющая обращаться с данными, как с обыкновенной таблицей или реляционной СУБД. Запросы HiveQL транслируются в Java-код заданий MapReduce.

Так как хранение и анализ данных будут производится в распределенной файловой системе HDFS, способная хранить очень большие файлы (размером в гигабайты или терабайты) и потоки данных, работающие на оборудовании стандартного серверного оборудования, данные инструменты будут наиболее подходящими для хранения информации.

Для передачи данных из HDFS в MariaDB используется инструмент Apache Sqoop — приложение с интерфейсом командной строки для передачи данных между реляционными базами данных и Hadoop.

Также для моделирования потоковой передачи данных будет использоваться Apache Kafka — гибрид распределённой базы данных и брокера сообщений с возможностью горизонтального масштабирования. Kafka собирает у приложений данные, хранит в своем распределённом хранилище, группируя по топикам, и отдаёт компонентам приложения по подписке. При этом сообщения хранятся на различных узлах-брокерах, что обеспечивает высокую доступность и отказоустойчивость. Данные в Kafka будут поступать из MariaDB через Flume — инструмент, позволяющий управлять потоками данных и передавать их на некоторый пункт назначения. Перед проведением анализа и реализацией конвейера, была разработана его схема, показанная на Рисунке 1.1.

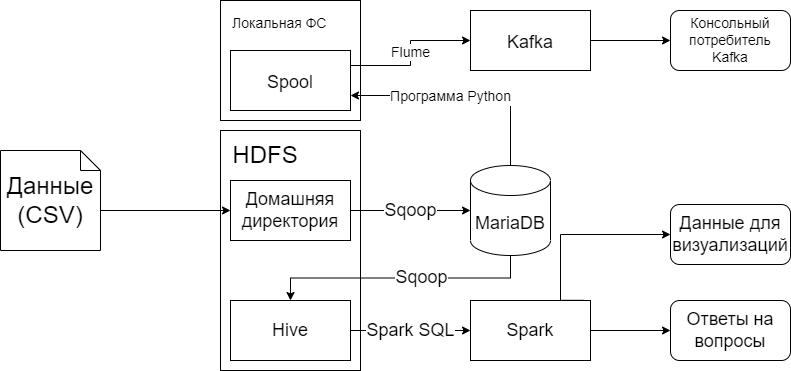


Рисунок 1.1 — Схема конвейера

Первоначально исходный набор данных в формате CSV загружается из репозитория Kaggle (<https://www.kaggle.com/datasets/abmsayem/air-pollution/data>) и сохраняется в домашней директории виртуальной машины.

Домашняя директория виртуальной машины — это директория, которая содержит файлы и настройки профиля пользователя виртуальной машины.

Домашняя директория виртуальной машины отличается от локальной директории тем, что она существует внутри виртуальной машины и содержит файлы и настройки для профиля пользователя виртуальной машины. В отличие от этого, локальная директория находится на реальной физической машине, на которой запущена виртуальная машина, и содержит файлы и настройки для профиля пользователя на этой машине. Кроме того, доступ к домашней директории виртуальной машины может быть ограничен только для пользователей этой виртуальной машины, в то время как локальная директория может быть доступна для других пользователей на реальной машине.

Затем данные необходимо загрузить в HDFS.

Перемещение файла из домашней директории в HDFS может быть полезным, если мы хотим обработать эти данные с помощью инструментов Hadoop. Файлы, хранящиеся в HDFS, доступны для обработки большим количеством узлов кластера, что позволяет распараллеливать и ускорять вычисления. Кроме того, HDFS обеспечивает резервное копирование данных и восстановление при сбое узлов, что обеспечивает надежность и отказоустойчивость системы.

Существует несколько способов загрузки данных в HDFS:

* использование команды hdfs dfs в терминале виртуальной машины;
* использование веб-интерфейса HDFS;
* команда mv для перемещения файлов в HDFS из домашней директории виртуальной машины;

С помощью Sqoop мы передаем данные в MariaDB.

Устанавливается Sqoop и настраивается соединение с базой данных MariaDB. Затем необходимо создать базу данных и таблицу в этой базе данных, куда впоследствии Sqoop перенесет данные из файла.

Sqoop создает временную таблицу в MariaDB и копирует данные из источника в эту временную таблицу. Далее происходит обработка и преобразование данных, если это необходимо, и Sqoop переносит данные из временной таблицы в целевую таблицу в MariaDB. По окончании импорта, Sqoop выводит отчет о выполненной операции, который включает информацию о количестве импортированных строк и времени, затраченном на импорт данных.

Данные в MariaDB с помощью Sqoop должны быть импортированы в таблицу Hive, так как Hive больше подходит для обработки больших объемов данных.

Импорт данных из MariaDB в Hive с помощью Sqoop на виртуальной машине происходит похожим на импорт в HDFS образом. Sqoop использует JDBC-драйвер для подключения к MariaDB и выгрузки данных, которые затем передаются в Hive. На основе данных в Hive нужно создать DataFrame в Spark.

DataFrame в Spark — это распределенный набор данных, организованный в столбцы, похожий на таблицу в реляционной базе данных. DataFrame можно рассматривать как логическую конструкцию, которая представляет данные, которые могут быть распределены на несколько узлов кластера для выполнения параллельных операций.

DataFrame обладает высокой производительностью благодаря распределенной обработке данных в памяти. Он позволяет выполнять различные операции над данными, такие как фильтрация, сортировка, группировка, агрегация, соединение таблиц и многое другое.

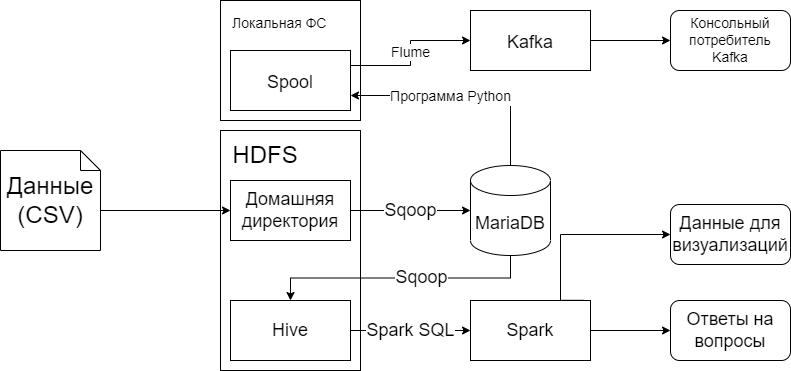
Использование DataFrame позволяет ускорить процесс обработки больших объемов данных и обеспечить эффективную работу с данными в распределенных системах, таких как Apache Spark.

Средствами Spark произвести нужные запросы для получения ответов на поставленные преподавателем вопросы.

Таким образом, создается в VirtualBox конвейер сбора, предобработки и анализа данных о влиянии загрязнения воздуха на уровень заболеваний с использованием технологий Apache и СУБД MariaDB. Проводится исследование и анализ данных, по которым делаются определенные выводы, отображенные посредством визуализации в виде графиков.

# Практическая часть

В данной курсовой работе будет создан конвейер для сбора, предобработки и анализа данных о влияния загрязнения воздуха на уровень заболеваний. Он будет создан на основе схемы на Рисунке 2.1.



**Рисунок 2.1 — Схема конвейера**

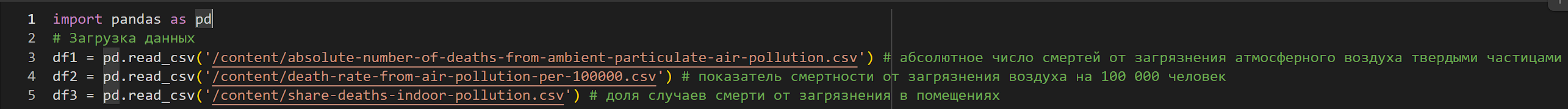
## Предобработка данных

На данном этапе проводится загрузка, очистка и преобразование данных для дальнейшего анализа.

Используются два набора данных:

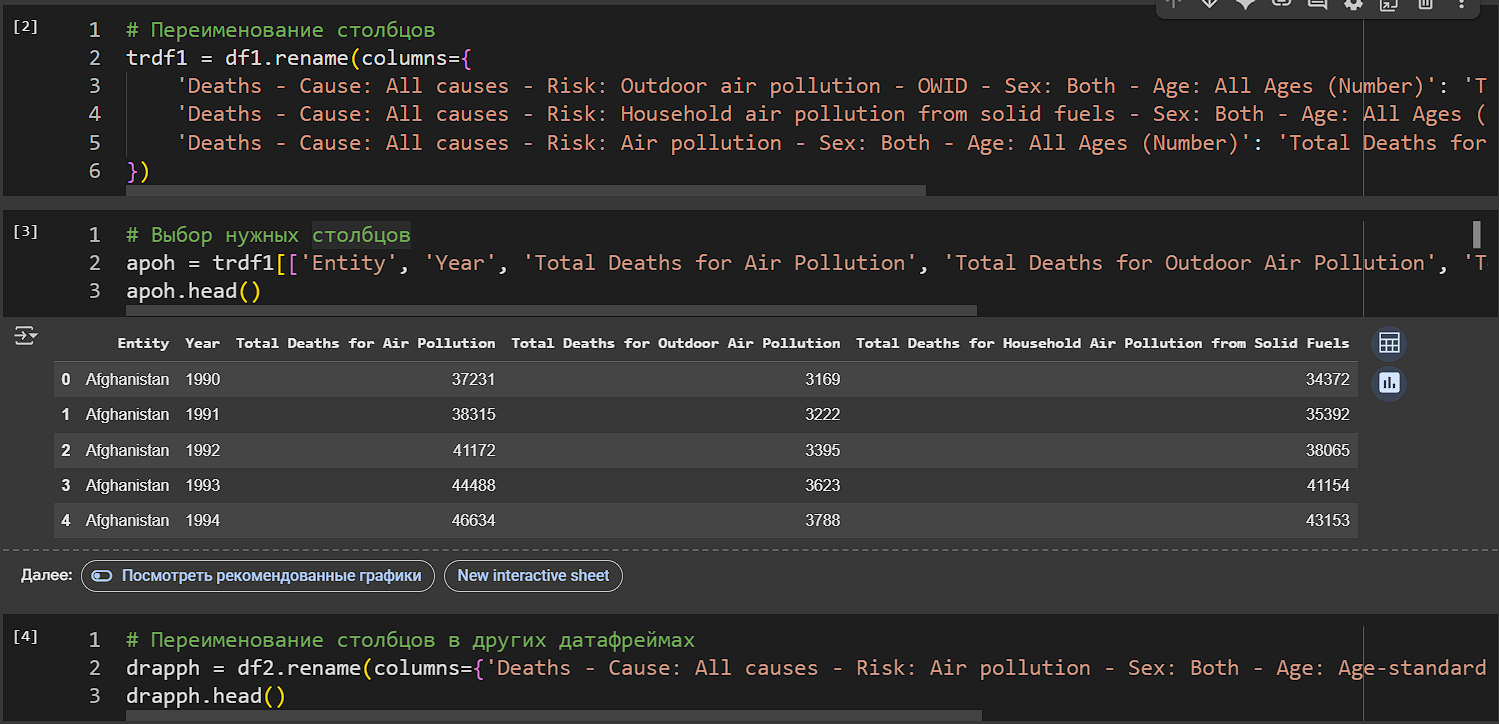
Абсолютное число смертей от загрязнения воздуха (absolute-number-of-deaths-from-ambient-particulate-air-pollution.csv).

Показатель смертности от загрязнения воздуха на 100 000 человек (death-rate-from-air-pollution-per-100000.csv).



**Рисунок 2.2 — Загрузка данных**

Для удобства работы столбцы переименовываются в более читаемые форматы.

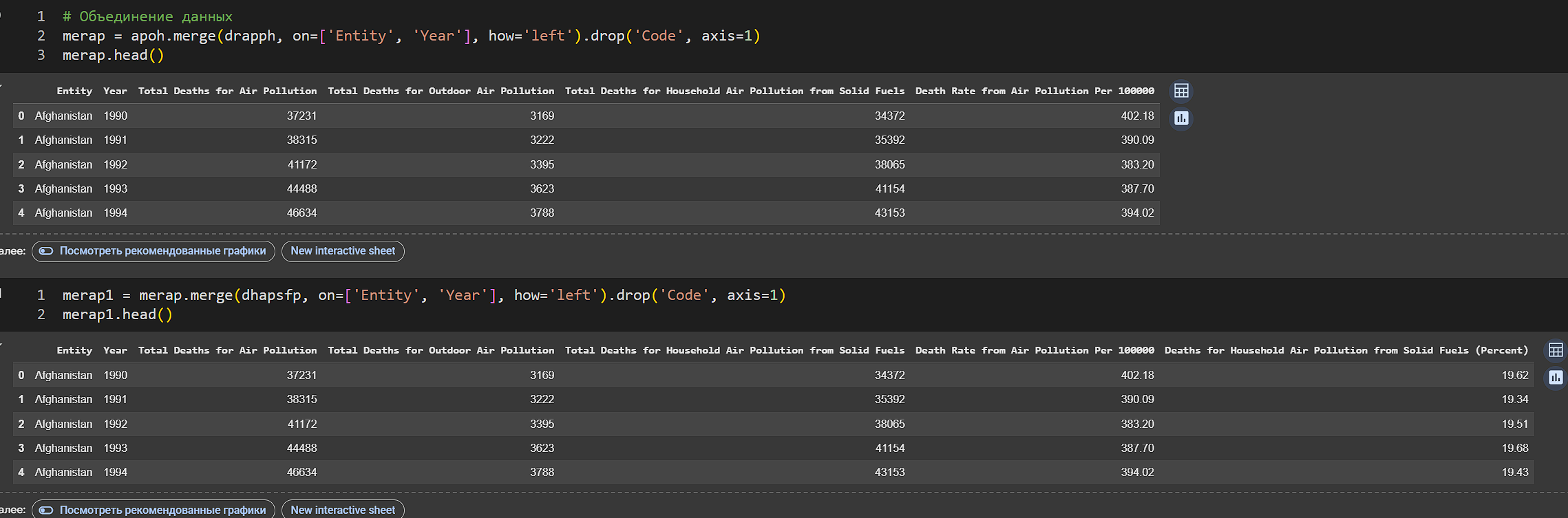


**Рисунок 2.3 — Переименование столбцов**



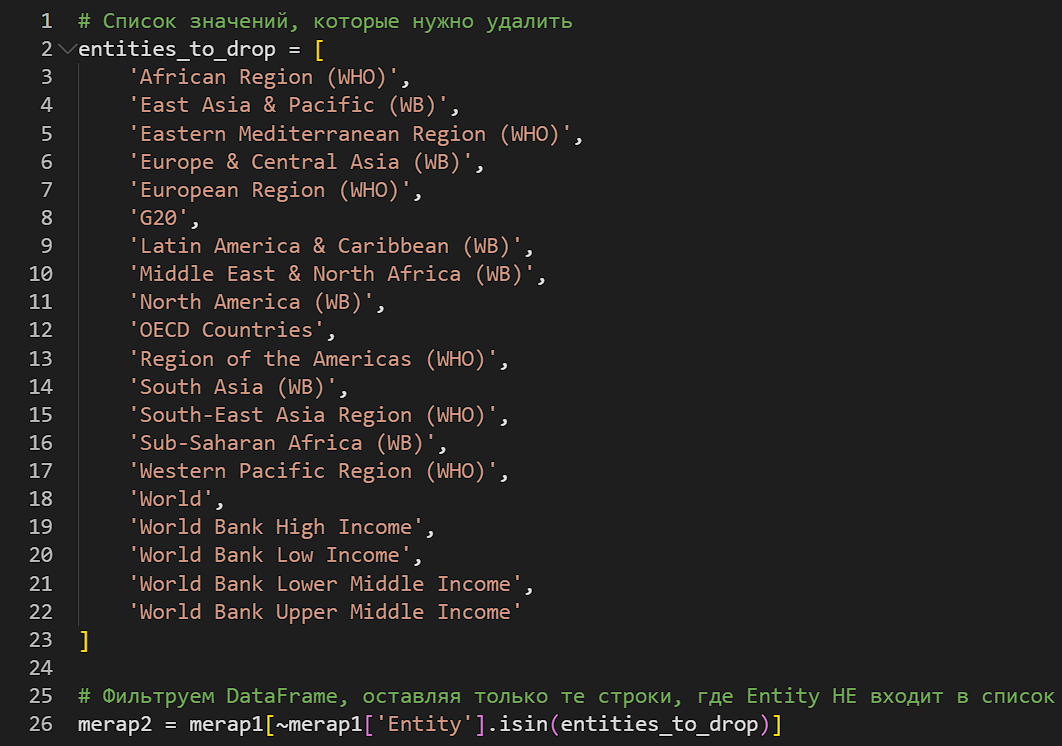
**Рисунок 2.4 — Переименование столбцов**

Два датафрейма объединяются по полям Entity (страна/регион) и Year (год).



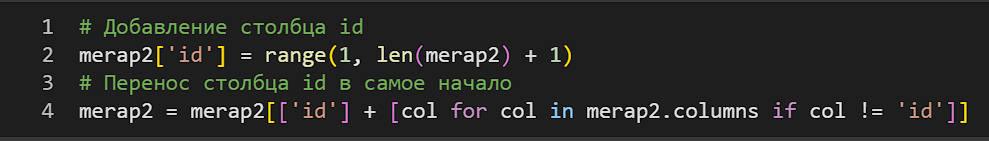
**Рисунок 2.5 — Объединение данных**

Удаляются агрегированные данные по регионам и миру, чтобы оставить только страны.



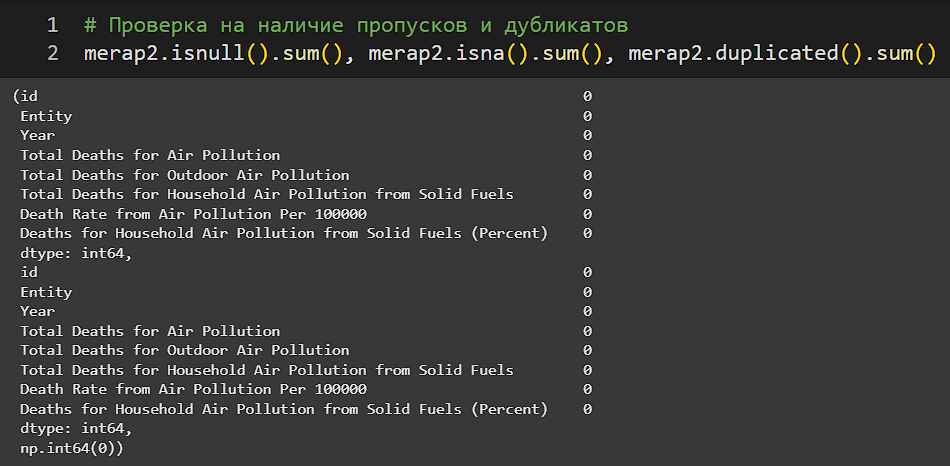
**Рисунок 2.6 — Фильтрация данных**

Добавление столбца id, для уникального идентификатора записи



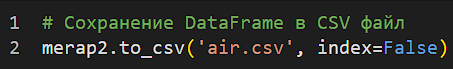
**Рисунок 2.7 — Добавление столбца**

Проверка на наличие пропусков



**Рисунок 2.8 — Проверка на наличие пропусков и дубликатов**

Дальше датафрейм сохраняем в csv файл



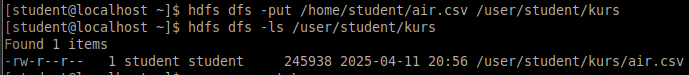
**Рисунок 2.9 — Сохранение в csv файл**

Файл загружаем в виртуальную машину и автоматически помещен в папку. Теперь мы должны загрузить этот файл в HDFS. Создаем пустую директорию в Hadoop HDFS. Проверим, что она пустая.



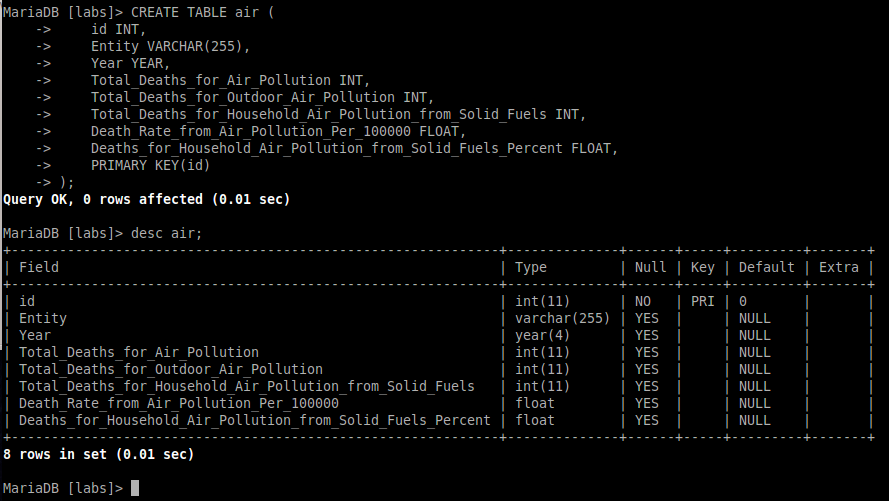
**Рисунок 2.10 — Создание пустой директории в HDFS**

Импортируем файл из загрузок в только что созданную пустую директорию kurs. Проверим наличие файла в директории.



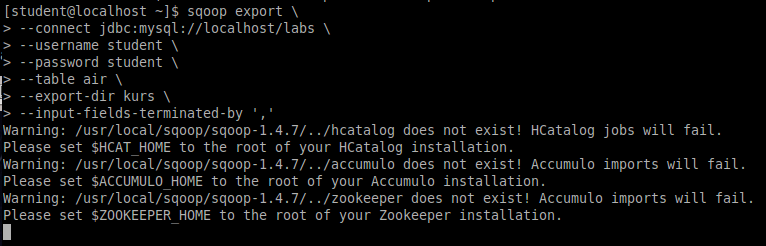
**Рисунок 2.11 — Импорт и проверка наличия файла**

Теперь необходимо поместить файл в MariaDB. Но для начала необходимо использовать в MariaDB базу данных, в которую мы будем импортировать данные и создать таблицу, чтобы данные записались в нее. Убедимся, что все было успешно проверим структуру таблицы. Результат представлен на Рисунке 2.12.



**Рисунок 2.12 — Создание таблицы и просмотр структуры таблицы в базе данных labs**

После того, как мы убедились, что таблица в базе данных успешно создана, можно загружать в нее данные из файла. Результат представлен на Рисунке 2.13.



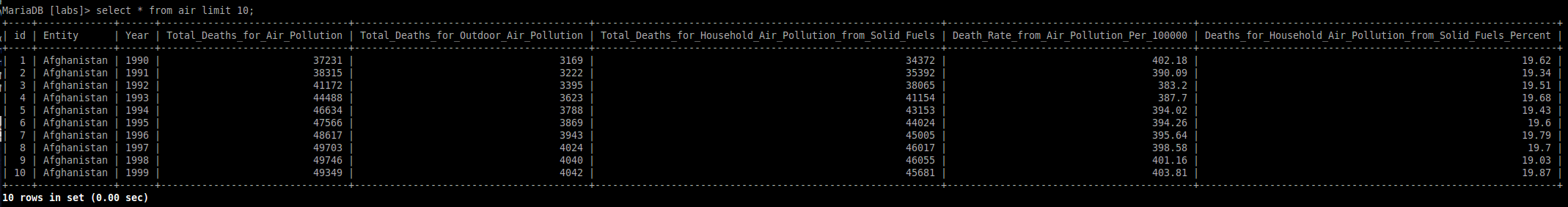
**Рисунок 2.13 — Импорт данных в таблицу MariaDB**

Об успешном импорте данных можно увидеть такой вывод на Рисунке 2.14.



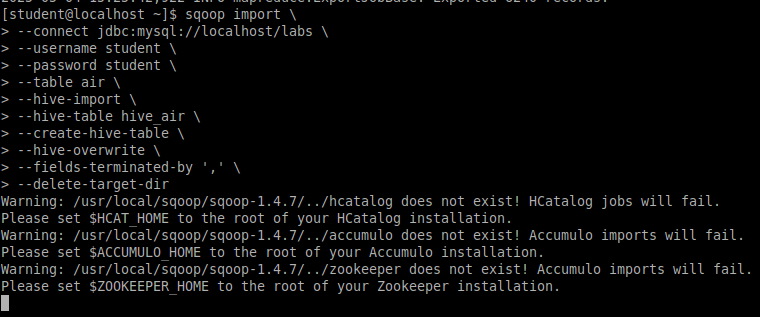
**Рисунок 2.14 — Консольные выводы импорта**

Проверим, что данные экспортировались успешно. Результат select запроса представлен на Рисунке 2.15.



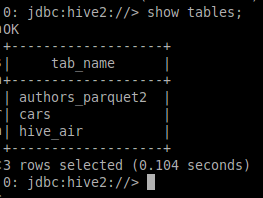
**Рисунок 2.15 — Результат вывода запроса в базе данных**

Далее необходимо из MariaDB экспортировать данные в Hive с помощью Sqoop. Процесс загрузки данных представлены на Рисунке 2.16.



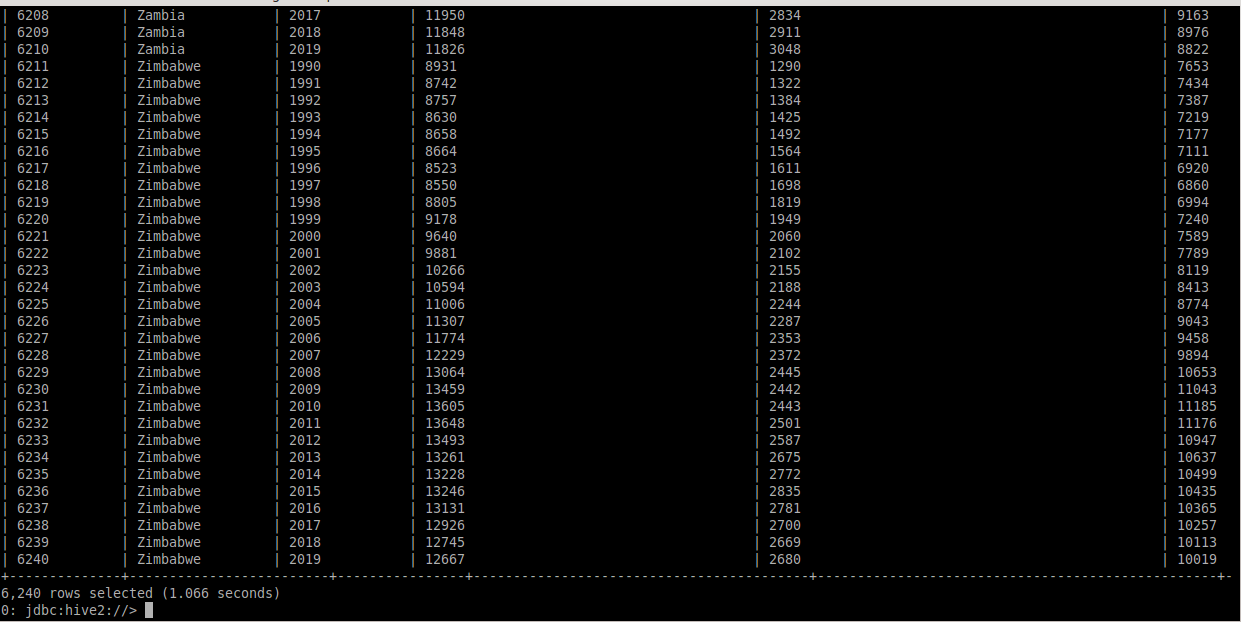
**Рисунок 2.16 — Запись данных из MariaDB в Hive**

Для проверки успешности выполненной команды выведем все таблицы, содержащиеся в Hive. Результаты представлены на Рисунке 2.17.



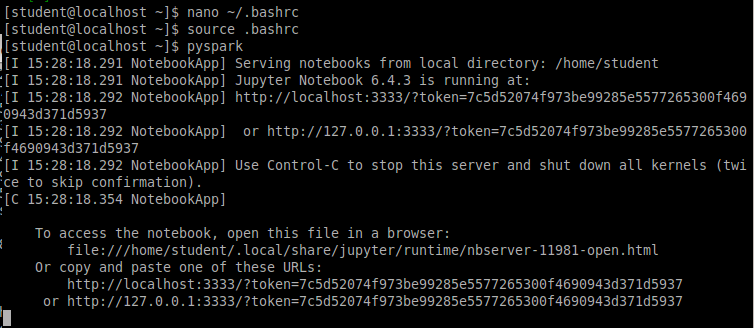
**Рисунок 2.17 — Вывод всех имеющихся таблиц в Hive**

Выведем содержание этой таблицы. Результаты представлены на Рисунке 2.18.



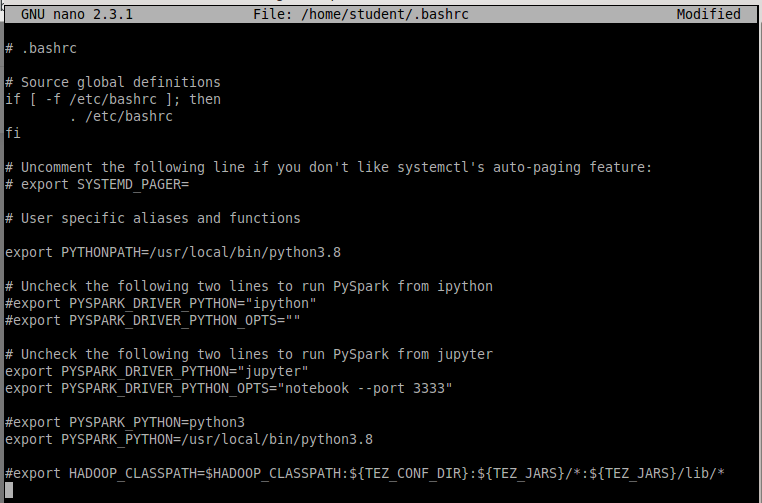
**Рисунок 2.18 — Вывод таблицы в Hive**

После этого из Hive данные отправляются в Spark. Запуск Spark представлена на Рисунке 2.19.



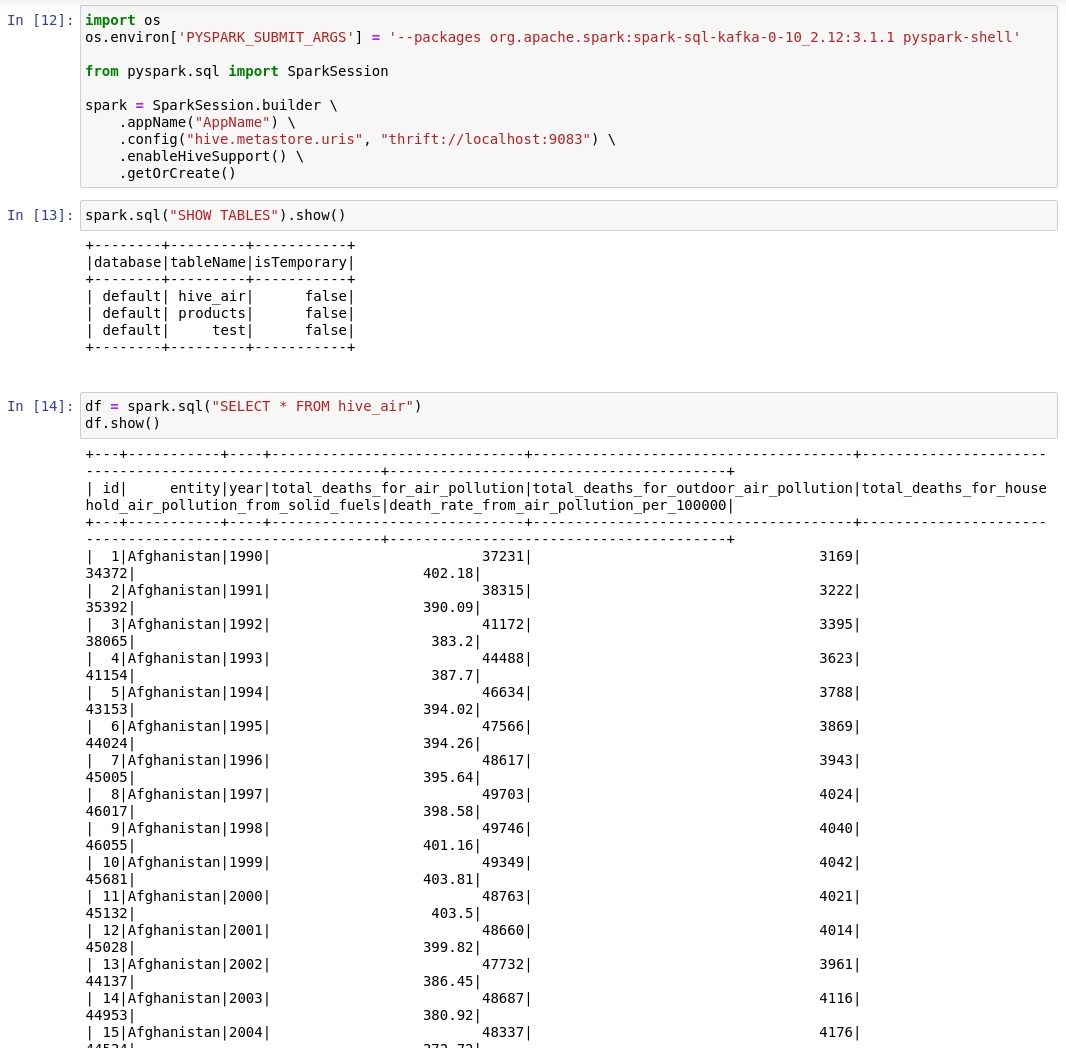
**Рисунок 2.19 — Запуск Spark**

Просмотрим содержание файла bashrc, представленного на Рисунке 2.20.



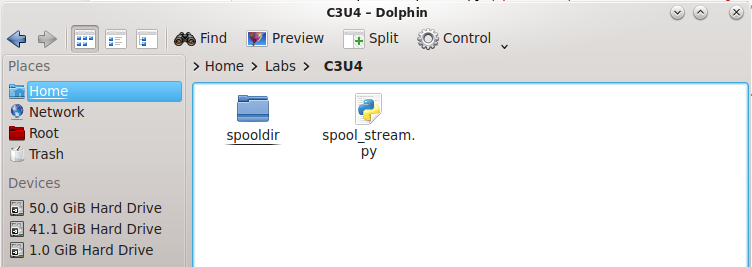
**Рисунок 2.20 — Содержимое файла .bashrc**

Импортируем данные из таблицы hive в Spark. Результат представлен на Рисунке 2.21.



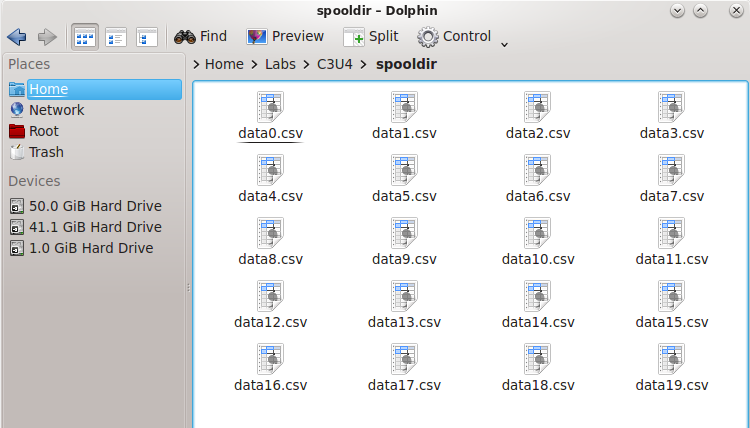
**Рисунок 2.21 — Импорт таблицы из Hive в Spark**

Написать программа на Python, в которой происходит подключение к базе данных, запрос 5-ти % всех строк в таблице с данными каждые 10 секунд и создание файла из полученных строк (в формате CSV) в папке Spooldir. Результат представлен на Рисунке 2.22.



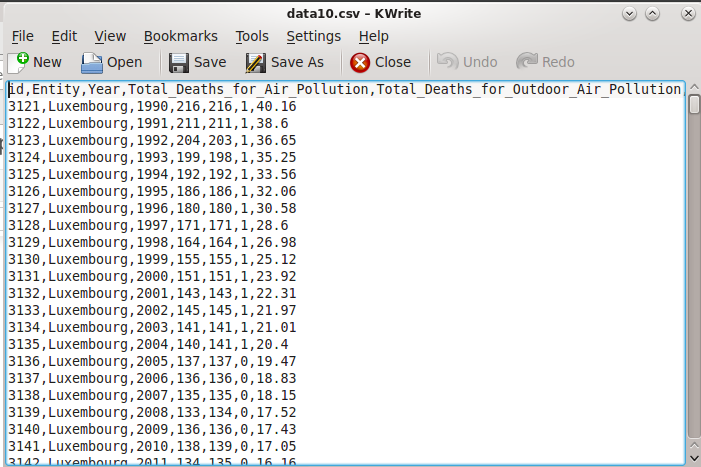
**Рисунок 2.22 — Содержимое директории**

После запуска кода, представленном в приложении А, в папке spooldir создалось 10 файлов формата csv. Результат представлен на Рисунке 2.23.



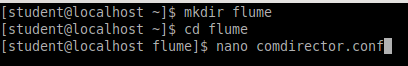
**Рисунок 2.23 — Содержимое директории**

Просмотрим содержимое любого файла. Результат представлен на Рисунке 2.24.



**Рисунок 2.24 — Содержимое файла**

Создаем директорию flume, а в ней файл comdirector.conf. Проделанная работа продемонстрирована на Рисунке 2.25.



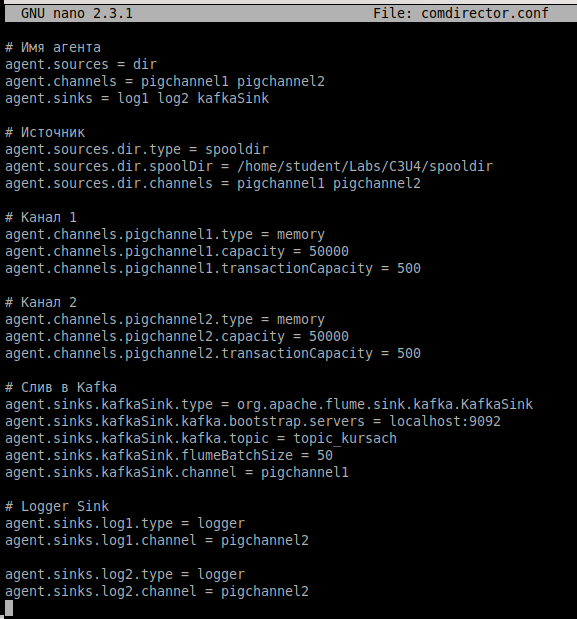
**Рисунок 2.25 — Создание конфигурационного файла**

В созданном файле настраиваются три компонента конфигурации агента Flume: источник, канал и слив, настраивается тип источника данных, в данном случае spooldir (источник, который наблюдает за определенной директорией и отправляет новые файлы в канал). Директория, которую будет наблюдать источник, указывается в параметре spooldir. Также источник подключается к двум каналам — pigchannel1 и pigchannel2.

Определяются параметры для двух каналов pigchannel1 и pigchannel2. Оба канала имеют тип memory, что означает, что сообщения будут храниться в памяти. Параметр capacity определяет максимальное количество событий, которое может храниться в канале, а параметр transactionCapacity указывает максимальное количество событий, которые могут быть обработаны в одной транзакции.

Определяются параметры для слива в Kafka. Слив типа KafkaSink позволяет отправлять события в брокер Kafka. В параметре kafka.bootstrap.servers указывается адрес и порт брокера, куда будут отправляться события. В параметре kafka.topic указывается название топика, в который будут отправляться сообщения. Параметр flumeBatchSize указывает количество событий, которые будут отправляться в одной пакетной операции.

Весь файл представлен на Рисунке 2.26.

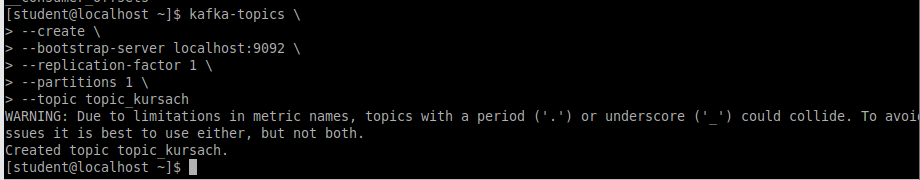


**Рисунок 2.26 — Конфигурационный файл агента flume**

Создание нового топика в Apache Kafka с названием "topic\_kursach".

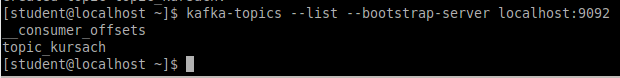
* kafka-topics — это исполняемый файл, который предоставляет команды для работы с топиками в Kafka;
* --create — это параметр, указывающий на создание нового топика;
* --bootstrap-server localhost:9092 — это параметр, указывающий на адрес и порт сервера Kafka, который нужен для создания топика;
* --replication-factor 1 — это параметр, указывающий на количество реплик, которые будут хранить данные топика. В данном случае установлено значение 1, что означает, что данные будут храниться только на одном брокере Kafka;
* --partitions 1 — это параметр, указывающий на количество партиций топика. В данном случае установлено значение 1, что означает, что все данные будут храниться в одной партиции;
* --topic topic\_kursach — это параметр, указывающий на название создаваемого топика. В данном случае топик будет называться "topic\_kursach".

Команда представлена на Рисунке 2.27.



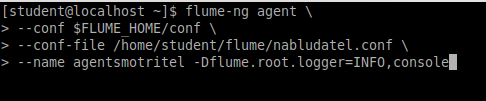
**Рисунок 2.27 — Создание топика в kafka**

Эта команда для чтения сообщений из топика topic\_kursach с самого первого сообщения. Bootstrap-server указывает адрес брокера Kafka, который будет использоваться для чтения сообщений. В данном случае используется брокер, запущенный локально на порту 9092. Команда представлена на Рисунке 2.28.



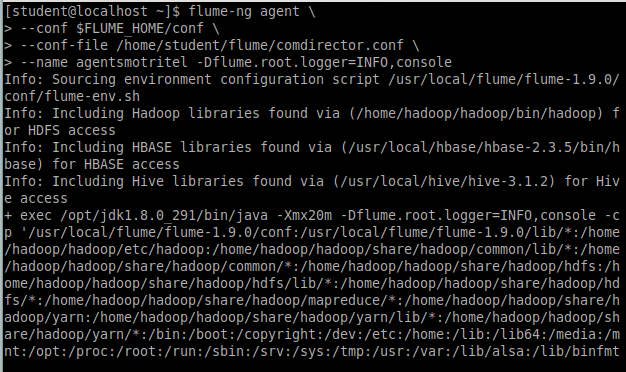
**Рисунок 2.28 — Проверка существующих топиков kafka**

Эта команда запускает Flume агента с именем agent, используя конфигурационный файл /home/student/flume/comdirector.conf. Conf указывает путь к директории с конфигурационными файлами Flume, а Dflume.root.logger=INFO,console задает уровень логирования для агента и вывод логов в консоль. Команда представлена на Рисунке 2.29.



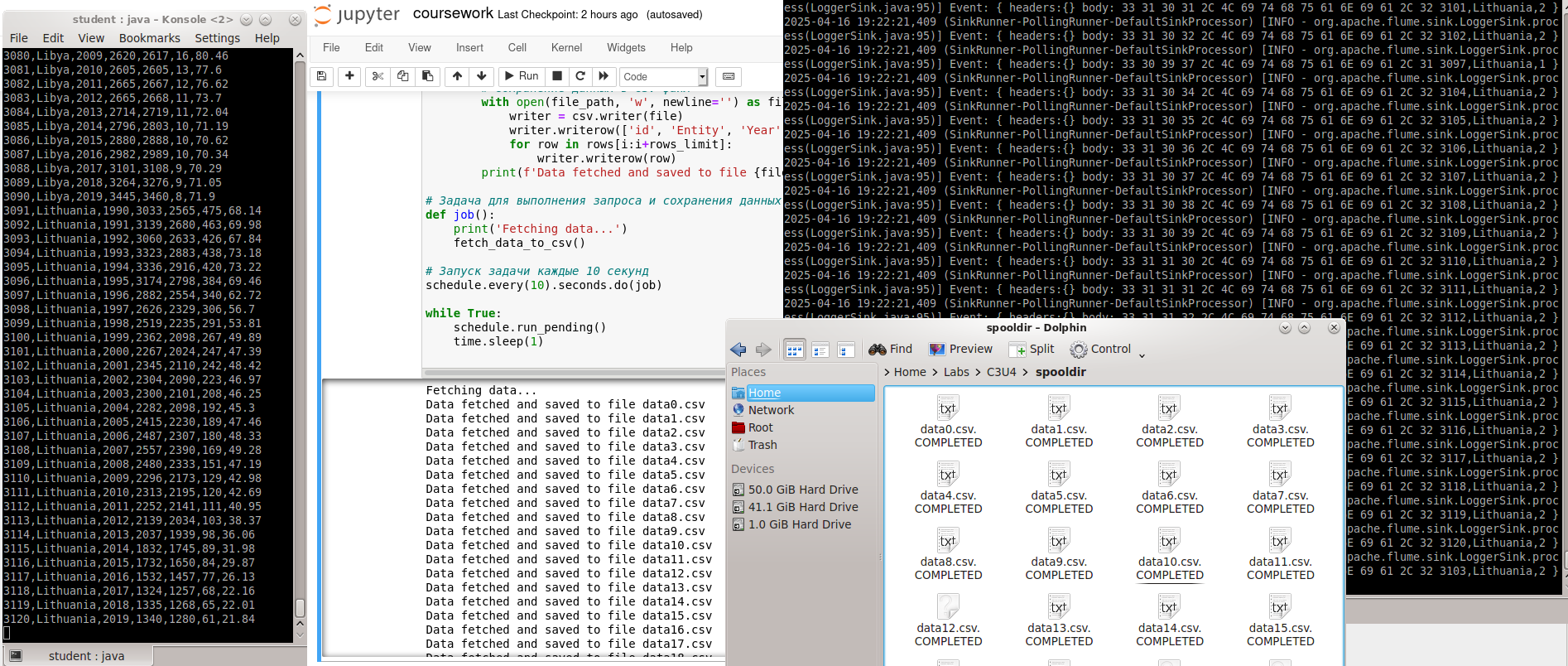
**Рисунок 2.29 — Запуск консольного потребителя kafka**

Результат работы команды представлен на Рисунке 2.30.



**Рисунок 2.30 — Запуск flafka**

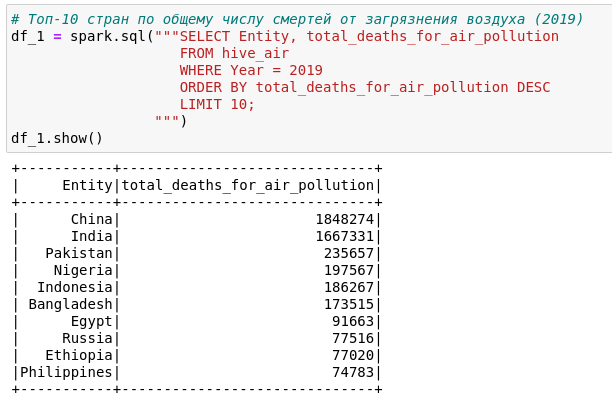
Запускаем код python, который работает с flume и тестируем часть конвейера. Результат работы представлен на Рисунке 2.31.



**Рисунок 2.31 — Отображение данных в консольном потребителе kafka**

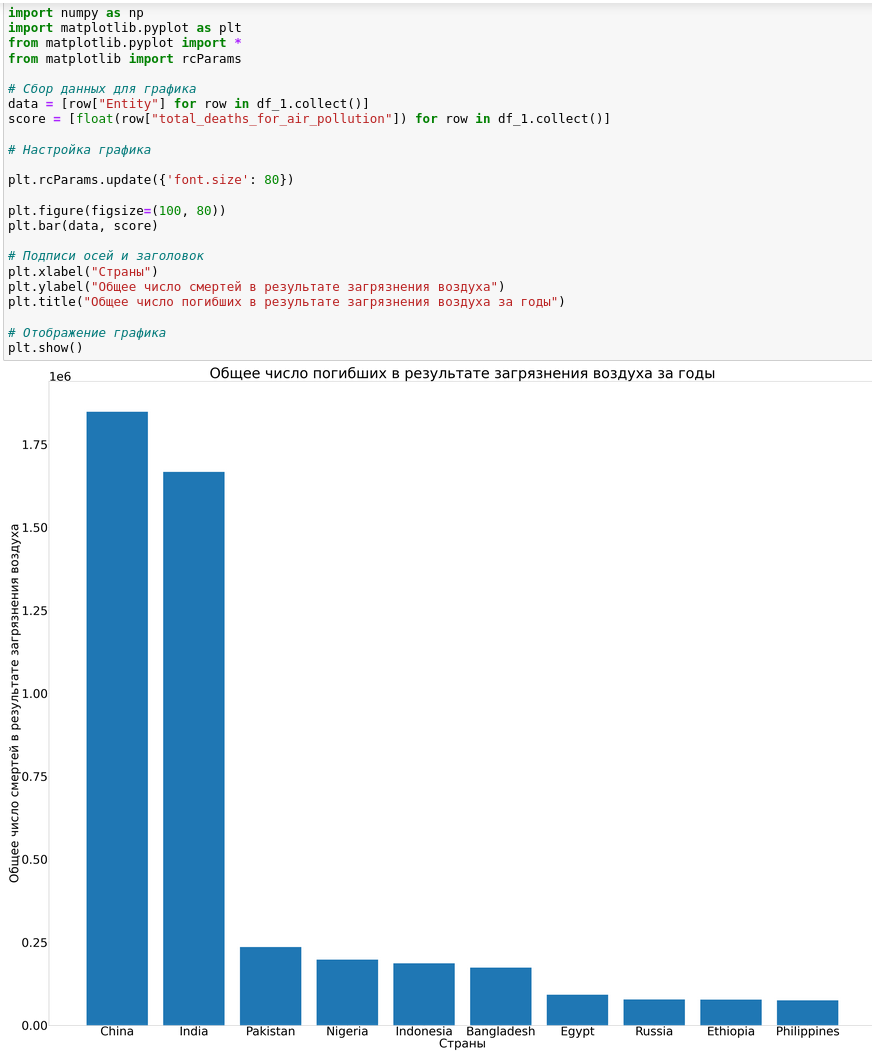
## Анализ данных

Выявили топ десять стран с максимальной смертностью (2019 г.). Итоговая таблица должна содержать страну, число смертей. Результат представлен на Рисунке 2.32.



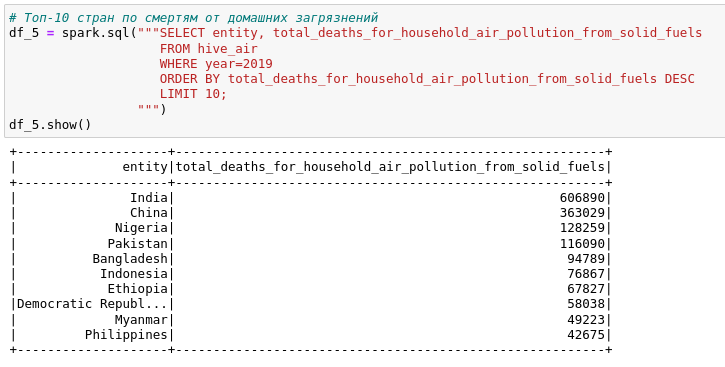
**Рисунок 2.32 — Общее число смертей**

Визуализируем полученный результат на гистограмме. Результаты визуализации представлены на Рисунке 2.33.



**Рисунок 2.33 — Количество смертей, вызванных в результате загрязнения воздуха**

Выявили топ десять стран по смертям от домашних загрязнений (2019 г.). Итоговая таблица должна содержать страну, число смертей. Результат представлен на Рисунке 2.34.



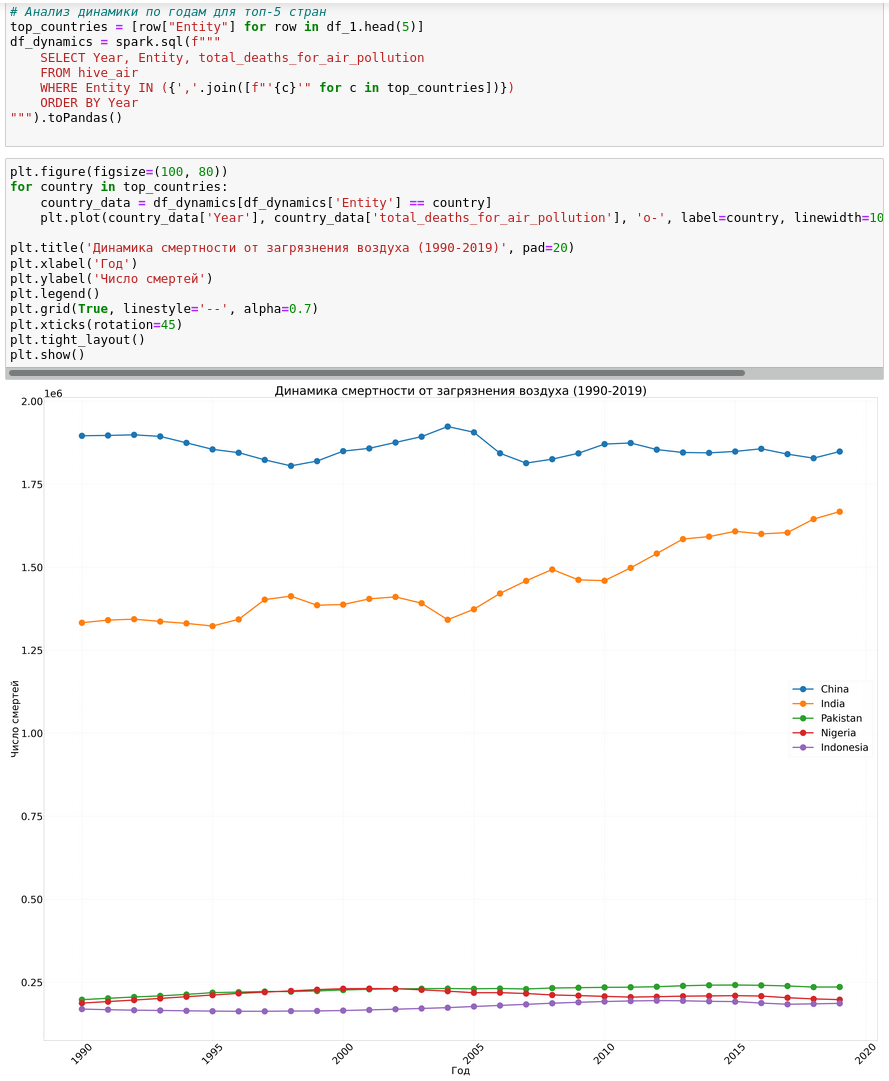
**Рисунок 2.34 — Число смертей от домашних загрязнений**

Визуализируем полученный результат на гистограмме. Результаты визуализации представлены на Рисунке 2.35.



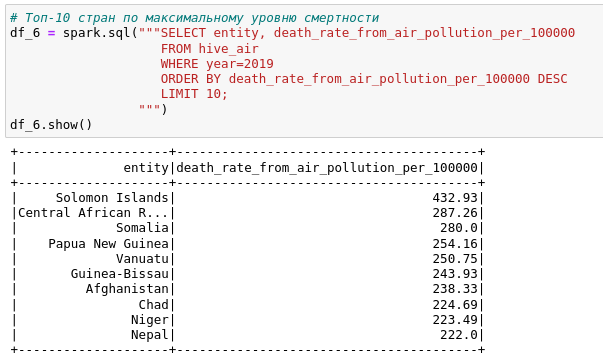
**Рисунок 2.35 — Количество смертей, вызванных от домашних загрязнений**

Взяли топ пять стран с максимальной смертностью (2019 г.) и посмотрели на их динамику смертности. Результат представлен на Рисунке 2.36.



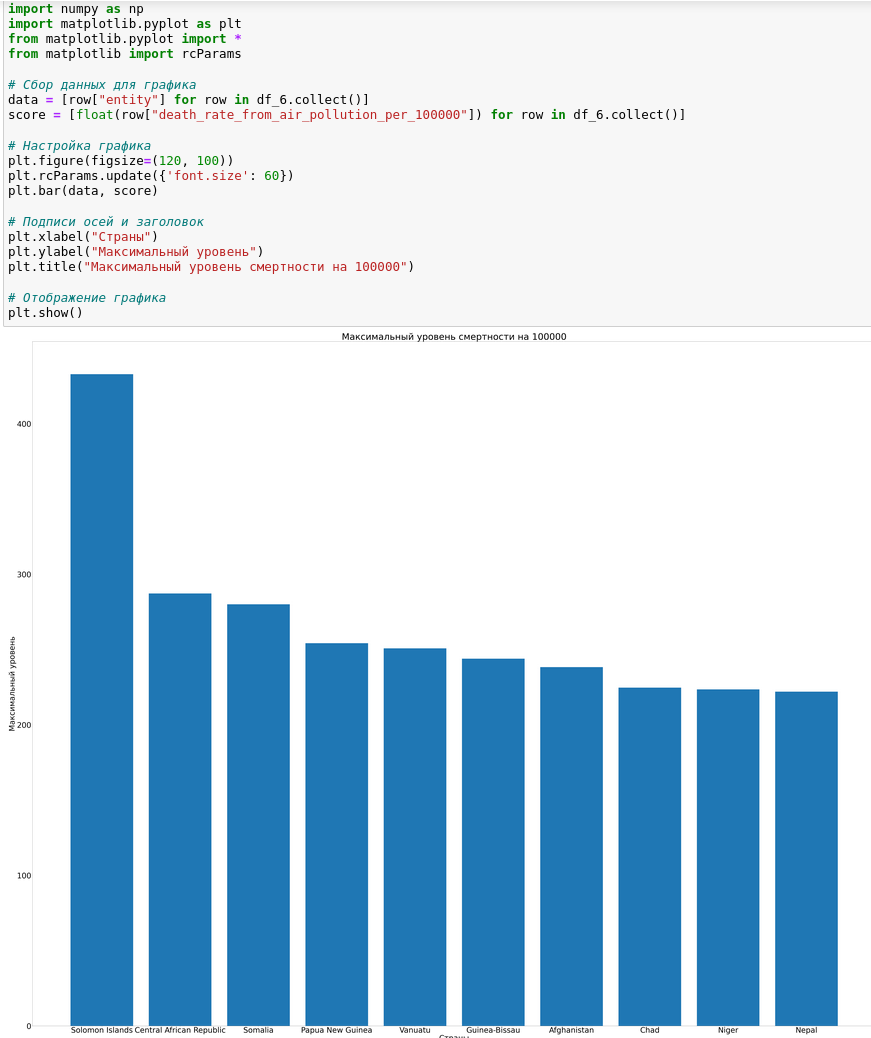
**Рисунок 2.36 — Динамика смертности**

Выявили топ десять стран по максимальному уровню смертности (2019 г.). Итоговая таблица должна содержать страну, уровень смертности на 100000 человек. Результат представлен на Рисунке 2.37.



**Рисунок 2.37 — Смены с самыми большими продажами**

Визуализируем полученный результат на гистограмме. Результаты визуализации представлены на Рисунке 2.38.



**Рисунок 2.38 — Визуализация по максимальному уровню смертности**

## Обработка результатов анализа

Рассчитана матрица корреляции между:

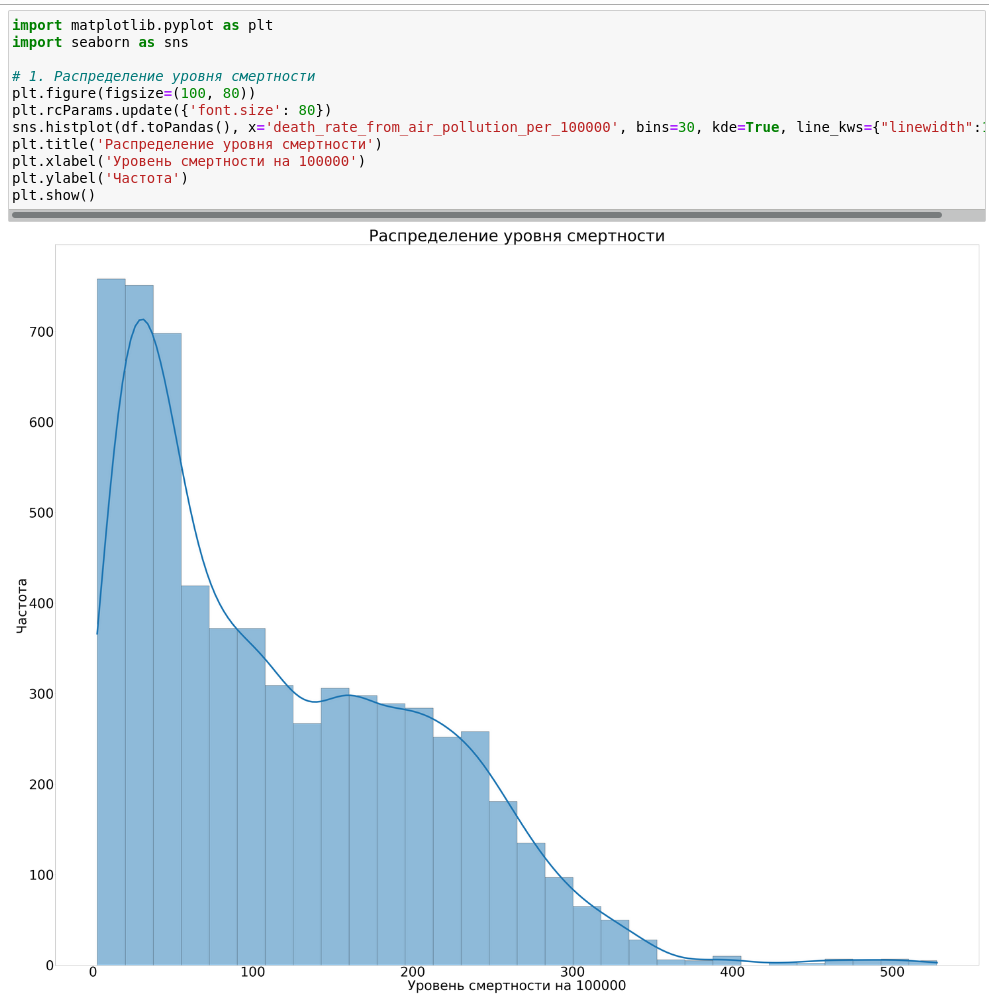
* Общей смертностью;
* Смертностью от наружных загрязнений;
* Смертностью от домашних загрязнений;
* Уровнем смертности на 100 тыс. населения;
* Процент смертей от загрязнения воздуха в помещениях;

Результаты визуализации представлены на Рисунке 2.39.



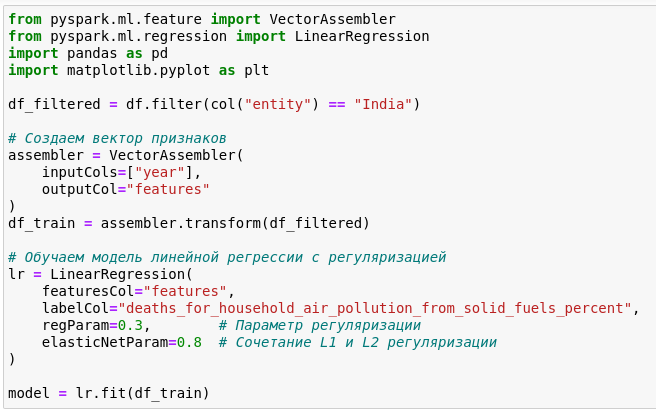
**Рисунок 2.39 — Тепловая карта корреляций**

Сделал анализ распределения показателей. Результаты визуализации представлены на Рисунке 2.40.



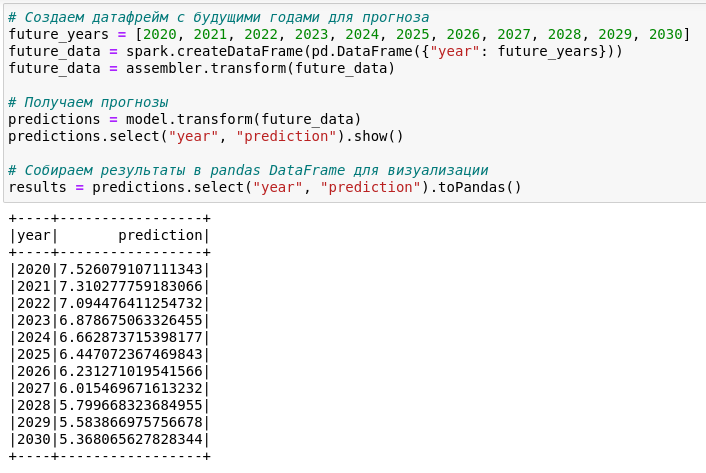
**Рисунок 2.40 — Визуализация распределение уровня смертности**

Построение прогнозной модели (линейная регрессия). Результаты визуализации представлены на Рисунке 2.41.



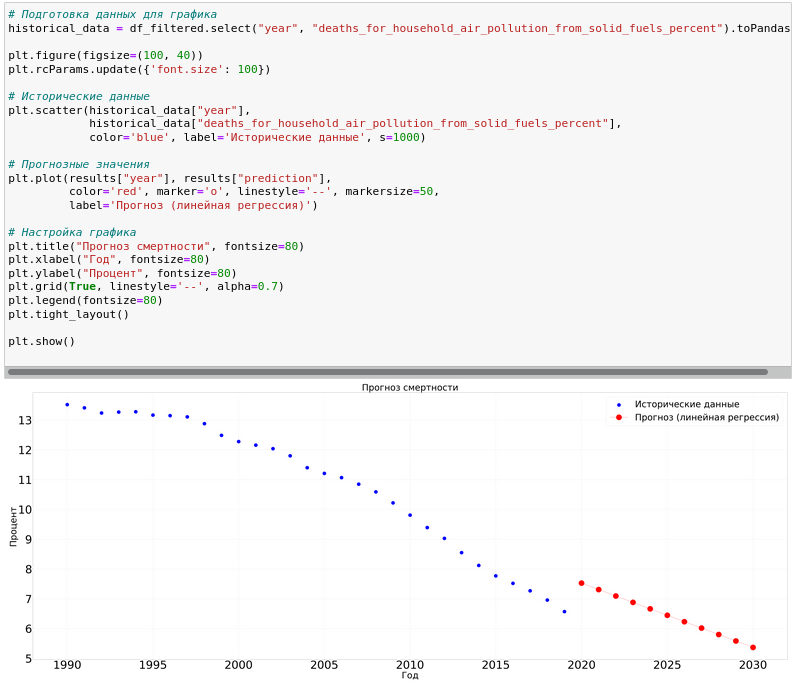
**Рисунок 2.41 — Создание линейной регрессии**

Получаем прогноз. Результаты визуализации представлены на Рисунке 2.42.



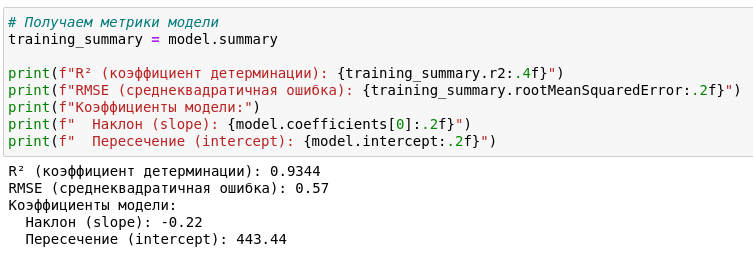
**Рисунок 2.42 — Получение прогнозов**

Визуализация данных и прогнозов. Результаты визуализации представлены на Рисунке 2.43.



**Рисунок 2.43 — Визуализация**

Метрики данных. Результаты визуализации представлены на Рисунке 2.44.



**Рисунок 2.44 — Метрики модели**

# Заключение

Таким образом, в курсовой работе разработан конвейер для предобработки, анализа и визуализации данных с помощью VirtualBox. Использованы продукты экосистемы Apache (Spark, Hadoop), которые обеспечили эффективную обработку больших массивов данных. Разработанный конвейер продемонстрировал свою масштабируемость и гибкость при работе с разнородными данными о загрязнении воздуха.

Была успешно реализована комплексная система анализа данных о загрязнении воздуха и его влиянии на здоровье населения.

Цель работы — разработка конвейера данных на основе технологий Big Data для анализа взаимосвязи между загрязнением воздуха и уровнем заболеваемости — была достигнута. В качестве среды для развертывания инструментов используется VirtualBox.

Задачи, решенные в данной курсовой работе:

* Собрать данные из открытых источников (Kaggle).
* Составить конвейер для сбора и передачи данных.
* Выдвинуть гипотезы и проверить их;
* Визуализировать полученные результаты для выявления взаимосвязей между загрязнением воздуха и уровнем заболеваемости.

# Список используемых источников

Теоретическая часть

1. Hadoop: The Definitive Guide" — Tom White (O'Reilly Media, Inc., 2015). [Электронный ресурс]
2. Hive [Электронный ресурс]. Режим доступа: https://hive.apache.org/
3. Apache Sqoop User Guide" — Apache Sqoop PMC (Apache Software Foundation, 2019) [Электронный ресурс]. Режим доступа:
4. Spark [Электронный ресурс]. Режим доступа: https://spark.apache.org/
5. Освоение MariaDB — Федерико Радззоли, Packt Publishing, 2017 [Электронный ресурс].
6. Oracle [Электронный ресурс]. Режим доступа: https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/home-dir.html
7. HDFS [Электронный ресурс] Режим доступа: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
8. DataFrame [Электронный ресурс] Режим доступа: https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html
9. Python [Электронный ресурс] Режим доступа: <https://docs.python.org/3/>

# Приложения

Приложение А —код на языке программирование Python, в которой происходит подключение к базе данных, запрос 5-ти % всех строк в таблице с данными каждые 10 секунд и создание файла из полученных строк в формате CSV в папке Spooldir.

## Приложение А

import csv

import pymysql

import schedule

import time

import os

# Параметры подключения к базе данных

DB\_HOST = 'localhost'

DB\_PORT = 3306

DB\_USER = 'student'

DB\_PASSWORD = 'student'

DB\_NAME = 'labs'

# Папка для сохранения данных

CSV\_FOLDER = '/home/student/Labs/C3U4/spooldir/'

# Запрос для получения всех строк в таблице

SQL\_QUERY = 'SELECT \* FROM air'

# Функция для выполнения запроса и сохранения данных в CSV-файл

def fetch\_data\_to\_csv():

# Подключение к базе данных

conn = pymysql.connect(host=DB\_HOST, port=DB\_PORT, user=DB\_USER, password=DB\_PASSWORD, database=DB\_NAME)

cursor = conn.cursor()

cursor.execute(SQL\_QUERY)

# Получение данных из базы данных

rows = cursor.fetchall()

# Закрытие соединения

cursor.close()

conn.close()

# Разбиение данных на части по 5%

total\_rows = len(rows)

rows\_limit = round(total\_rows \* 0.05)

for i in range(0, total\_rows, rows\_limit):

# Генерация имени файла

filename = f"data{i//rows\_limit}.csv"

file\_path = os.path.join(CSV\_FOLDER, filename)

# Сохранение данных в CSV-файл

with open(file\_path, 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['id', 'Entity', 'Year', 'Total\_Deaths\_for\_Air\_Pollution', 'Total\_Deaths\_for\_Outdoor\_Air\_Pollution', 'Total\_Deaths\_for\_Household\_Air\_Pollution\_from\_Solid\_Fuels', 'Death\_Rate\_from\_Air\_Pollution\_Per\_100000'])

for row in rows[i:i+rows\_limit]:

writer.writerow(row)

print(f'Data fetched and saved to file {filename}')

# Задача для выполнения запроса и сохранения данных в CSV-файл

def job():

print('Fetching data...')

fetch\_data\_to\_csv()

# Запуск задачи каждые 10 секунд

schedule.every(10).seconds.do(job)

while True:

schedule.run\_pending()

time.sleep(1)

## Приложение Б

import os

os.environ['PYSPARK\_SUBMIT\_ARGS'] = '--packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.1.1 pyspark-shell'

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("AppName") \

.config("hive.metastore.uris", "thrift://localhost:9083") \

.enableHiveSupport() \

.getOrCreate()

# Топ-10 стран по общему числу смертей от загрязнения воздуха (2019)

df\_1 = spark.sql("""SELECT Entity, total\_deaths\_for\_air\_pollution

                    FROM hive\_air

                    WHERE Year = 2019

                    ORDER BY total\_deaths\_for\_air\_pollution DESC

                    LIMIT 10;

                 """)

df\_1.show()

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.pyplot import \*

from matplotlib import rcParams

# Сбор данных для графика

data = [row["Entity"] for row in df\_1.collect()]

score = [*float*(row["total\_deaths\_for\_air\_pollution"]) for row in df\_1.collect()]

# Настройка графика

plt.rcParams.update({'font.size': 80})

plt.figure(*figsize*=(100, 80))

plt.bar(data, score)

# Подписи осей и заголовок

plt.xlabel("Страны")

plt.ylabel("Общее число смертей в результате загрязнения воздуха")

plt.title("Общее число погибших в результате загрязнения воздуха за годы")

# Отображение графика

plt.show()

# Топ-10 стран по смертям от домашних загрязнений

df\_5 = spark.sql("""SELECT entity, total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels

                    FROM hive\_air

                    WHERE year=2019

                    ORDER BY total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels DESC

                    LIMIT 10;

                 """)

df\_5.show()

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.pyplot import \*

from matplotlib import rcParams

# Сбор данных для графика

data = [row["entity"] for row in df\_5.collect()]

score = [*float*(row["total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels"]) for row in df\_5.collect()]

# Настройка графика

plt.rcParams.update({'font.size': 90})

plt.figure(*figsize*=(180, 80))

plt.bar(data, score)

# Подписи осей и заголовок

plt.xlabel("Страны")

plt.ylabel("Общее число смертей от домашних загрязнений")

plt.title("Общее число погибших от домашних загрязнений")

# Отображение графика

plt.show()

# Анализ динамики по годам для топ-5 стран

top\_countries = [row["Entity"] for row in df\_1.head(5)]

df\_dynamics = spark.sql(*f*"""

    SELECT Year, Entity, total\_deaths\_for\_air\_pollution

    FROM hive\_air

    WHERE Entity IN ({','.join([*f*"'{c}'" for c in top\_countries])})

    ORDER BY Year

""").toPandas()

plt.figure(*figsize*=(100, 80))

for country in top\_countries:

    country\_data = df\_dynamics[df\_dynamics['Entity'] == country]

    plt.plot(country\_data['Year'], country\_data['total\_deaths\_for\_air\_pollution'], 'o-', *label*=country, *linewidth*=10, *markersize*=50)

plt.title('Динамика смертности от загрязнения воздуха (1990-2019)', *pad*=20)

plt.xlabel('Год')

plt.ylabel('Число смертей')

plt.legend()

plt.grid(True, *linestyle*='--', *alpha*=0.7)

plt.xticks(*rotation*=45)

plt.tight\_layout()

plt.show()

# Топ-10 стран по максимальному уровню смертности

df\_6 = spark.sql("""SELECT entity, death\_rate\_from\_air\_pollution\_per\_100000

                    FROM hive\_air

                    WHERE year=2019

                    ORDER BY death\_rate\_from\_air\_pollution\_per\_100000 DESC

                    LIMIT 10;

                 """)

df\_6.show()

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.pyplot import \*

from matplotlib import rcParams

# Сбор данных для графика

data = [row["entity"] for row in df\_6.collect()]

score = [*float*(row["death\_rate\_from\_air\_pollution\_per\_100000"]) for row in df\_6.collect()]

# Настройка графика

plt.figure(*figsize*=(120, 100))

plt.rcParams.update({'font.size': 60})

plt.bar(data, score)

# Подписи осей и заголовок

plt.xlabel("Страны")

plt.ylabel("Максимальный уровень")

plt.title("Максимальный уровень смертности на 100000")

# Отображение графика

plt.show()

from pyspark.sql.functions import col

import seaborn as sns

# Корреляционный анализ

numeric\_cols = [

    "total\_deaths\_for\_air\_pollution",

    "total\_deaths\_for\_outdoor\_air\_pollution",

    "total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels",

    "death\_rate\_from\_air\_pollution\_per\_100000",

    "deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent"

]

corr\_matrix = df.select([col(c).cast('double') for c in numeric\_cols]).toPandas().corr()

plt.figure(*figsize*=(120, 100))

sns.heatmap(corr\_matrix, *annot*=True, *cmap*='coolwarm', *center*=0, *fmt*='.2f')

plt.title('Корреляционная матрица показателей загрязнения воздуха', *pad*=20)

plt.xticks(*rotation*=45)

plt.tight\_layout()

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

# 1. Распределение уровня смертности

plt.figure(*figsize*=(100, 80))

plt.rcParams.update({'font.size': 80})

sns.histplot(df.toPandas(), *x*='death\_rate\_from\_air\_pollution\_per\_100000', *bins*=30, *kde*=True, *line\_kws*={"linewidth":10})

plt.title('Распределение уровня смертности')

plt.xlabel('Уровень смертности на 100000')

plt.ylabel('Частота')

plt.show()

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.regression import LinearRegression

import pandas as pd

import matplotlib.pyplot as plt

df\_filtered = df.filter(col("entity") == "India")

# Создаем вектор признаков

assembler = VectorAssembler(

*inputCols*=["year"],

*outputCol*="features"

)

df\_train = assembler.transform(df\_filtered)

# Обучаем модель линейной регрессии с регуляризацией

lr = LinearRegression(

*featuresCol*="features",

*labelCol*="deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent",

*regParam*=0.3,        # Параметр регуляризации

*elasticNetParam*=0.8  # Сочетание L1 и L2 регуляризации

)

model = lr.fit(df\_train)

# Создаем датафрейм с будущими годами для прогноза

future\_years = [2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030]

future\_data = spark.createDataFrame(pd.DataFrame({"year": future\_years}))

future\_data = assembler.transform(future\_data)

# Получаем прогнозы

predictions = model.transform(future\_data)

predictions.select("year", "prediction").show()

# Собираем результаты в pandas DataFrame для визуализации

results = predictions.select("year", "prediction").toPandas()

# Подготовка данных для графика

historical\_data = df\_filtered.select("year", "deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent").toPandas()

plt.figure(*figsize*=(100, 40))

plt.rcParams.update({'font.size': 100})

# Исторические данные

plt.scatter(historical\_data["year"],

            historical\_data["deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent"],

*color*='blue', *label*='Исторические данные', *s*=1000)

# Прогнозные значения

plt.plot(results["year"], results["prediction"],

*color*='red', *marker*='o', *linestyle*='--', *markersize*=50,

*label*='Прогноз (линейная регрессия)')

# Настройка графика

plt.title("Прогноз смертности", *fontsize*=80)

plt.xlabel("Год", *fontsize*=80)

plt.ylabel("Процент", *fontsize*=80)

plt.grid(True, *linestyle*='--', *alpha*=0.7)

plt.legend(*fontsize*=80)

plt.tight\_layout()

plt.show()

# Получаем метрики модели

training\_summary = model.summary

print(*f*"R² (коэффициент детерминации): {training\_summary.r2*:.4f*}")

print(*f*"RMSE (среднеквадратичная ошибка): {training\_summary.rootMeanSquaredError*:.2f*}")

print(*f*"Коэффициенты модели:")

print(*f*"  Наклон (slope): {model.coefficients[0]*:.2f*}")

print(*f*"  Пересечение (intercept): {model.intercept*:.2f*}")