



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

---

---

**Институт информационных технологий (ИИТ)**  
**Кафедра прикладной математики (ПМ)**

**КУРСОВАЯ РАБОТА**

по дисциплине

«Методы анализа данных»

**Тема курсовой работы:** «Реализация жизненного цикла анализа данных  
влияния загрязнения воздуха на уровень смертности»

Студент группы ИМБО-02-22      Ким Кирилл Сергеевич

  
(подпись)

Руководитель  
курсовой работы

Старший преподаватель Морошкин  
Н.А.

  
(подпись)

Работа представлена к защите      «\_\_» \_\_\_\_\_ 2025 г.

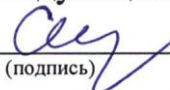
Допущен к защите      «\_\_» \_\_\_\_\_ 2025 г.

Москва 2025 г.



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
РТУ МИРЭА

Институт информационных технологий (ИИТ)  
Кафедра прикладной математики (ПМ)

Утверждаю  
Заведующий кафедрой ПМ  
  
(подпись) Смоленцева Т.Е.  
«22» февраля 2025 г.

**ЗАДАНИЕ**  
на выполнение курсовой работы  
по дисциплине «Методы анализа данных»

Студент Ким Кирилл Сергеевич

Группа ИМБО-02-22

**Тема** «Реализация жизненного цикла анализа данных влияния загрязнения воздуха на уровень смертности»

**Исходные данные:** выбранный студентом набор данных для обработки и анализа

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

Характеристика исследуемой области и анализируемых данных (описание области анализируемых данных, актуальные проблемы, в контексте решения задач в курсовой работе, описание источников данных, его особенностей и аналогов, причина выбора конкретного источника для получения анализируемых данных, структура набора данных, анализ метаданных)

Структура конвейера данных и составных процессов (архитектура конвейера процесса получения и предобработки данных из открытых источников, используемые методы и инструменты сбора и обработки данных, проведение предобработки данных, обработка грязных данных)

Анализ данных (проведение качественного анализа данных с использованием изученных методов, моделей и инструментов анализа, обработка полученных результатов, и визуализаций, реализация прогнозных оценок, оценка выдвинутых гипотез и т.д.)

**Срок представления к защите курсовой работы:**

до «23» мая 2025 г.

**Задание на курсовую работу выдал**

  
Подпись руководителя

Морошкин Н.А.  
(ФИО руководителя)

«22» февраля 2025 г.

**Задание на курсовую работу получил**

  
Подпись обучающегося

Ким К.С.  
(ФИО обучающегося)

«22» февраля 2025 г.

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ .....	5
1.1 Описание исследуемой области .....	5
1.2 Информация об источнике и о наборе данных .....	5
1.3 Описание процесса сбора данных .....	8
2 ПРАКТИЧЕСКАЯ ЧАСТЬ .....	12
2.1 Предобработка данных .....	12
2.2 Анализ данных .....	23
2.3 Обработка результатов анализа .....	26
ЗАКЛЮЧЕНИЕ .....	30
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ .....	31
ПРИЛОЖЕНИЯ .....	32
Приложение А .....	33
Приложение Б .....	34
Приложение В .....	36

# 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.

# **1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

## **1.1 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 (NO<sub>2</sub>), диоксид серы (SO<sub>2</sub>), озон (O<sub>3</sub>);
- 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.

## **1.2 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 тыс. населения. Файл представлен в формате csv [1.10].

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.
2. Entity — строковый;
  - Название страны;
  - Примеры: Afghanistan, Albania, Algeria, и т.д.
3. Year — целочисленный;
  - Год данных;
  - Диапазон: 1990-2019.
4. 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
5. 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
6. Total Deaths for Household Air Pollution from Solid Fuels — целочисленный;
- Количество смертей от загрязнения воздуха внутри помещений (из-за твердого топлива);
  - Описательная статистика:
    - Среднее: 16483.94
    - Минимум: 0.0
    - Максимум: 1329829.0
    - Стандартное отклонение: 88591.77
7. 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
8. 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.

### **1.3 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

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

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.

## 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

В данной курсовой работе будет создан конвейер для сбора, предобработки и анализа данных о влиянии загрязнения воздуха на уровень заболеваний. Он будет создан на основе схемы на Рисунок 1.1 — Схема конвейера.

### 2.1 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 тыс. человек (death-rate-from-air-pollution-per-100000.csv). Результат представлен на Рисунке 2.1.

```
1 import pandas as pd
2 # Загрузка данных
3 df1 = pd.read_csv('/content/absolute-number-of-deaths-from-ambient-particulate-air-pollution.csv') # абсолютное число смертей от загрязнения атмосферного воздуха твердыми частицами
4 df2 = pd.read_csv('/content/death-rate-from-air-pollution-per-100000.csv') # показатель смертности от загрязнения воздуха на 100 000 человек
5 df3 = pd.read_csv('/content/share-deaths-indoor-pollution.csv') # доля случаев смерти от загрязнения в помещениях
```

Рисунок 2.1 — Загрузка данных

Для удобства работы столбцы переименовываются в более читаемые форматы. Результат представлен на Рисунках 2.2-2.3.

```
(2) 1 # Переименование столбцов
2 trdf1 = df1.rename(columns={
3     'Deaths - Cause: All causes - Risk: Outdoor air pollution - OWID - Sex: Both - Age: All Ages (Number)': 'T
4     'Deaths - Cause: All causes - Risk: Household air pollution from solid fuels - Sex: Both - Age: All Ages (
5     'Deaths - Cause: All causes - Risk: Air pollution - Sex: Both - Age: All Ages (Number)': 'Total Deaths for
6 })

(3) 1 # Выбор нужных столбцов
2 apoh = trdf1[['Entity', 'Year', 'Total Deaths for Air Pollution', 'Total Deaths for Outdoor Air Pollution', 'T
3 apoh.head()
```

	Entity	Year	Total Deaths for Air Pollution	Total Deaths for Outdoor Air Pollution	Total Deaths for Household Air Pollution from Solid Fuels
0	Afghanistan	1990	37231	3169	34372
1	Afghanistan	1991	38315	3222	35392
2	Afghanistan	1992	41172	3395	38065
3	Afghanistan	1993	44488	3623	41154
4	Afghanistan	1994	46634	3788	43153

Далее: [Посмотреть рекомендованные графики](#) [New interactive sheet](#)

```
(4) 1 # Переименование столбцов в других датафреймах
2 drapph = df2.rename(columns={'Deaths - Cause: All causes - Risk: Air pollution - Sex: Both - Age: Age-standard
3 drapph.head()
```

Рисунок 2.2 — Переименование столбцов

```

1 # Переименование столбцов в нужный формат
2 drapph = df2.rename(columns={'Deaths - Cause: All causes - Risk: Air pollution - Sex: Both - Age: Age-standardized (Rate)': 'Death Rate from Air Pollution Per 100000'})
3 drapph.head()

Entity Code Year Death Rate from Air Pollution Per 100000
0 Afghanistan AFIQ 1990 402.18
1 Afghanistan AFIQ 1991 380.09
2 Afghanistan AFIQ 1992 383.20
3 Afghanistan AFIQ 1993 387.70
4 Afghanistan AFIQ 1994 394.02

1 dhapsp = df3.rename(columns={'Deaths - Cause: All causes - Risk: Household air pollution from solid fuels - Sex: Both - Age: Age-standardized (Percent)': 'Deaths for Household Air Pollution from Solid Fuels (Percent)'})
2 dhapsp.head()

Entity Code Year Deaths for Household Air Pollution from Solid Fuels (Percent)
0 Afghanistan AFIQ 1990 19.62
1 Afghanistan AFIQ 1991 19.34
2 Afghanistan AFIQ 1992 19.51
3 Afghanistan AFIQ 1993 19.68
4 Afghanistan AFIQ 1994 19.43

```

**Рисунок 2.3 — Переименование столбцов**

Два датафрейма объединяются по полям Entity (страна/регион) и Year (год). Результат представлен на Рисунке 2.4.

```

1 # Объединение данных
2 merap = apoh.merge(drapph, on=['Entity', 'Year'], how='left').drop('Code', axis=1)
3 merap.head()

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
0 Afghanistan 1990 37231 3169 34372 402.18
1 Afghanistan 1991 38315 3222 35390 380.09
2 Afghanistan 1992 41172 3395 38905 383.20
3 Afghanistan 1993 44488 3673 41154 387.70
4 Afghanistan 1994 49534 3788 43153 394.02

1 merap1 = merap.merge(dhapsp, on=['Entity', 'Year'], how='left').drop('Code', axis=1)
2 merap1.head()

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 Deaths for Household Air Pollution from Solid Fuels (Percent)
0 Afghanistan 1990 37231 3169 34372 402.18 19.62
1 Afghanistan 1991 38315 3222 35390 380.09 19.34
2 Afghanistan 1992 41172 3395 38905 383.20 19.51
3 Afghanistan 1993 44488 3673 41154 387.70 19.68
4 Afghanistan 1994 49534 3788 43153 394.02 19.43

```

**Рисунок 2.4 — Объединение данных**

Удаляются агрегированные данные по регионам и миру, чтобы оставить только страны. Код представлен в Приложение А.

Добавление столбца id, для уникального идентификатора записи. Код представлен в Приложение А.

Проверка на наличие пропусков. Результат представлен на Рисунке 2.5.

```

1 # Проверка на наличие пропусков и дубликатов
2 merap2.isnull().sum(), merap2.isna().sum(), merap2.duplicated().sum()

(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
Deaths for Household Air Pollution from Solid Fuels (Percent)
dtype: int64,
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
Deaths for Household Air Pollution from Solid Fuels (Percent)
dtype: int64,
np.int64(0))

```

**Рисунок 2.5 — Проверка на наличие пропусков и дубликатов**

Дальше датафрейм сохраняем в csv файл. Код представлен в Приложение А.

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. Результат представлен на Рисунке 2.6.

```
[student@localhost ~]$ hdfs dfs -mkdir /user/student/kurs
[student@localhost ~]$ hdfs dfs -ls /user/student/kurs
[student@localhost ~]$
```

Рисунок 2.6 — Создание пустой директории в HDFS

Импортируем файл из загрузок в только что созданную пустую директорию kurs. Проверим наличие файла в директории. Результат представлен на Рисунке 2.7.

```
[student@localhost ~]$ hdfs dfs -put /home/student/air.csv /user/student/kurs
[student@localhost ~]$ hdfs dfs -ls /user/student/kurs
Found 1 items
-rw-r--r--  1 student student    245938 2025-04-11 20:56 /user/student/kurs/air.csv
```

Рисунок 2.7 — 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 2.8.

```
MariaDB [labs]> CREATE TABLE air (
->   id INT,
->   Entity VARCHAR(255),
->   Year YEAR,
->   Total_Deaths_for_Air_Pollution INT,
->   Total_Deaths_for_Outdoor_Air_Pollution INT,
->   Total_Deaths_for_Household_Air_Pollution_from_Solid_Fuels INT,
->   Death_Rate_from_Air_Pollution_Per_100000 FLOAT,
->   Deaths_for_Household_Air_Pollution_from_Solid_Fuels_Percent FLOAT,
->   PRIMARY KEY(id)
-> );
Query OK, 0 rows affected (0.01 sec)

MariaDB [labs]> desc air;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | NO | PRI | 0 | |
| Entity | varchar(255) | YES | | NULL | |
| Year  | year(4) | YES | | NULL | |
| Total_Deaths_for_Air_Pollution | int(11) | YES | | NULL | |
| Total_Deaths_for_Outdoor_Air_Pollution | int(11) | YES | | NULL | |
| Total_Deaths_for_Household_Air_Pollution_from_Solid_Fuels | int(11) | YES | | NULL | |
| Death_Rate_from_Air_Pollution_Per_100000 | float | YES | | NULL | |
| Deaths_for_Household_Air_Pollution_from_Solid_Fuels_Percent | float | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
8 rows in set (0.01 sec)

MariaDB [labs]>
```

Рисунок 2.8 — 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 2.9.

```
[student@localhost ~]$ sqoop export \  
> --connect jdbc:mysql://localhost/labs \  
> --username student \  
> --password student \  
> --table air \  
> --export-dir kurs \  
> --input-fields-terminated-by ','  
Warning: /usr/local/sqoop/sqoop-1.4.7/./hcatalog does not exist! HCatalog jobs will fail.  
Please set $HCAT_HOME to the root of your HCatalog installation.  
Warning: /usr/local/sqoop/sqoop-1.4.7/./accumulo does not exist! Accumulo imports will fail.  
Please set $ACCUMULO_HOME to the root of your Accumulo installation.  
Warning: /usr/local/sqoop/sqoop-1.4.7/./zookeeper does not exist! Accumulo imports will fail.  
Please set $ZOOKEEPER_HOME to the root of your Zookeeper installation.  
[student@localhost ~]$
```

**Рисунок 2.9 — Импорт данных в таблицу MariaDB**

Об успешном импорте данных можно увидеть такой вывод на Рисунке 2.10.

```
2025-05-04 15:25:42,910 INFO mapreduce.ExportJobBase: Transferred 288.0371 KB in 32.703 seconds (8.8077 KB/sec)  
2025-05-04 15:25:42,922 INFO mapreduce.ExportJobBase: Exported 6240 records.  
[student@localhost ~]$
```

**Рисунок 2.10 — Консольные выводы импорта**

Проверим, что данные экспортировались успешно. Результат select запроса представлен на Рисунке 2.11.

MariaDB [labs]> select * from air limit 10;							
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	Deaths for Household Air Pollution from Solid Fuels Percent
1	Afghanistan	1990	37231	3169	34372	462.18	19.62
2	Afghanistan	1991	38315	3222	35392	390.09	19.34
3	Afghanistan	1992	41172	3395	38905	383.2	19.51
4	Afghanistan	1993	44608	3623	41154	387.7	19.68
5	Afghanistan	1994	46634	3788	43153	394.62	19.43
6	Afghanistan	1995	47566	3869	44024	394.26	19.6
7	Afghanistan	1996	48017	3943	45805	395.64	19.79
8	Afghanistan	1997	49703	4024	46817	398.58	19.7
9	Afghanistan	1998	49746	4040	46655	403.16	19.63
10	Afghanistan	1999	49349	4042	45801	403.81	19.87

10 rows in set (0.00 sec)

**Рисунок 2.11 — Результат вывода запроса в базе данных**

Далее необходимо из MariaDB экспортировать данные в HIVE с помощью Sqoop. Процесс загрузки данных представлены на Рисунке 2.12.

```
[student@localhost ~]$ sqoop import \  
> --connect jdbc:mysql://localhost/labs \  
> --username student \  
> --password student \  
> --table air \  
> --hive-import \  
> --hive-table hive_air \  
> --create-hive-table \  
> --hive-overwrite \  
> --fields-terminated-by ',' \  
> --delete-target-dir  
Warning: /usr/local/sqoop/sqoop-1.4.7/./hcatalog does not exist! HCatalog jobs will fail.  
Please set $HCAT_HOME to the root of your HCatalog installation.  
Warning: /usr/local/sqoop/sqoop-1.4.7/./accumulo does not exist! Accumulo imports will fail.  
Please set $ACCUMULO_HOME to the root of your Accumulo installation.  
Warning: /usr/local/sqoop/sqoop-1.4.7/./zookeeper does not exist! Accumulo imports will fail.  
Please set $ZOOKEEPER_HOME to the root of your Zookeeper installation.  
[student@localhost ~]$
```

**Рисунок 2.12 — Запись данных из MariaDB в Hive**

Для проверки успешности выполненной команды выведем все таблицы, содержащиеся в HIVE. Результаты представлены на Рисунке 2.13.

```
0: jdbc:hive2://> show tables;
OK
+-----+
| tab_name |
+-----+
| authors_parquet2 |
| cars |
| hive_air |
+-----+
3 rows selected (0.104 seconds)
0: jdbc:hive2://>
```

Рисунок 2.13 — Вывод всех имеющихся таблиц в Hive

Выведем содержание этой таблицы. Результаты представлены на Рисунке 2.14.

```
6208 | Zambia | 2017 | 11950 | 2834 | 9163
6209 | Zambia | 2018 | 11848 | 2911 | 8976
6210 | Zambia | 2019 | 11826 | 3048 | 8822
6211 | Zimbabwe | 1990 | 8931 | 1290 | 7653
6212 | Zimbabwe | 1991 | 8742 | 1322 | 7434
6213 | Zimbabwe | 1992 | 8757 | 1384 | 7387
6214 | Zimbabwe | 1993 | 8630 | 1425 | 7219
6215 | Zimbabwe | 1994 | 8658 | 1492 | 7177
6216 | Zimbabwe | 1995 | 8664 | 1564 | 7111
6217 | Zimbabwe | 1996 | 8523 | 1611 | 6920
6218 | Zimbabwe | 1997 | 8550 | 1698 | 6860
6219 | Zimbabwe | 1998 | 8805 | 1819 | 6994
6220 | Zimbabwe | 1999 | 9178 | 1949 | 7240
6221 | Zimbabwe | 2000 | 9640 | 2060 | 7589
6222 | Zimbabwe | 2001 | 9881 | 2102 | 7789
6223 | Zimbabwe | 2002 | 10266 | 2155 | 8119
6224 | Zimbabwe | 2003 | 10594 | 2188 | 8413
6225 | Zimbabwe | 2004 | 11006 | 2244 | 8774
6226 | Zimbabwe | 2005 | 11307 | 2287 | 9043
6227 | Zimbabwe | 2006 | 11774 | 2353 | 9458
6228 | Zimbabwe | 2007 | 12229 | 2372 | 9894
6229 | Zimbabwe | 2008 | 13064 | 2445 | 10653
6230 | Zimbabwe | 2009 | 13459 | 2442 | 11043
6231 | Zimbabwe | 2010 | 13605 | 2443 | 11185
6232 | Zimbabwe | 2011 | 13648 | 2501 | 11176
6233 | Zimbabwe | 2012 | 13493 | 2587 | 10947
6234 | Zimbabwe | 2013 | 13261 | 2675 | 10637
6235 | Zimbabwe | 2014 | 13228 | 2772 | 10499
6236 | Zimbabwe | 2015 | 13246 | 2835 | 10435
6237 | Zimbabwe | 2016 | 13131 | 2781 | 10365
6238 | Zimbabwe | 2017 | 12926 | 2700 | 10257
6239 | Zimbabwe | 2018 | 12745 | 2669 | 10113
6240 | Zimbabwe | 2019 | 12667 | 2680 | 10019
+-----+
6,240 rows selected (1.066 seconds)
0: jdbc:hive2://>
```

Рисунок 2.14 — Вывод таблицы в Hive

После этого из Hive данные отправляются в Spark. Запуск Spark представлена на Рисунке 2.15.

```
[student@localhost ~]$ nano ~/.bashrc
[student@localhost ~]$ source ~/.bashrc
[student@localhost ~]$ pyspark
[I 15:28:18.291 NotebookApp] Serving notebooks from local directory: /home/student
[I 15:28:18.291 NotebookApp] Jupyter Notebook 6.4.3 is running at:
[I 15:28:18.292 NotebookApp] http://localhost:3333/?token=7c5d52074f973be99285e5577265300f469
0943d371d5937
[I 15:28:18.292 NotebookApp] or http://127.0.0.1:3333/?token=7c5d52074f973be99285e5577265300
f4690943d371d5937
[I 15:28:18.292 NotebookApp] Use Control-C to stop this server and shut down all kernels (twi
ce to skip confirmation).
[C 15:28:18.354 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/student/.local/share/jupyter/runtime/nbserver-11981-open.html
Or copy and paste one of these URLs:
http://localhost:3333/?token=7c5d52074f973be99285e5577265300f4690943d371d5937
or http://127.0.0.1:3333/?token=7c5d52074f973be99285e5577265300f4690943d371d5937
```

Рисунок 2.15 — Запуск Spark

Посмотрим содержание файла bashrc, представленного на Рисунке 2.16.



```

GNU nano 2.3.1      File: /home/student/.bashrc      Modified
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

export PYTHONPATH=/usr/local/bin/python3.8

# Uncheck the following two lines to run PySpark from ipython
#export PYSARK_DRIVER_PYTHON="ipython"
#export PYSARK_DRIVER_PYTHON_OPTS=""

# Uncheck the following two lines to run PySpark from jupyter
export PYSARK_DRIVER_PYTHON="jupyter"
export PYSARK_DRIVER_PYTHON_OPTS="notebook --port 3333"

#export PYSARK_PYTHON=python3
export PYSARK_PYTHON=/usr/local/bin/python3.8

#export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:${TEZ_CONF_DIR}:${TEZ_JARS}/*:${TEZ_JARS}/lib/*

```

**Рисунок 2.16 — Содержимое файла .bashrc**

Импортируем данные из таблицы hive в Spark. Результат представлен на Рисунке 2.17.

```

In [12]: import os
os.environ['PYSARK_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()

In [13]: spark.sql("SHOW TABLES").show()

+-----+-----+-----+
|database|tableName|isTemporary|
+-----+-----+-----+
| default| hive_air|         false|
| default| products|        false|
| default|  test|         false|
+-----+-----+-----+

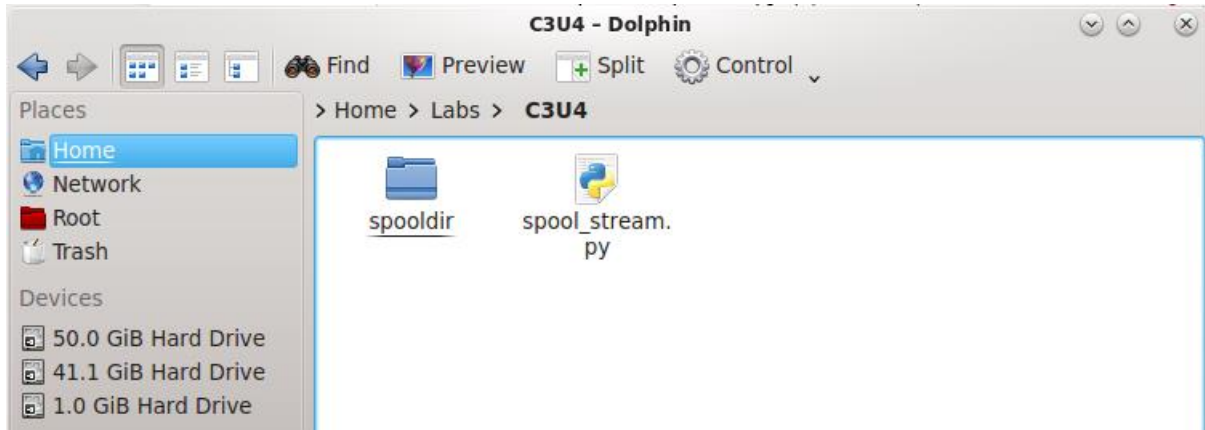
In [14]: df = spark.sql("SELECT * FROM hive_air")
df.show()

+-----+-----+-----+-----+-----+-----+-----+
| id|      entity|year|total deaths for air pollution|total deaths for outdoor air pollution|total deaths for house hold air pollution from solid fuels|death_rate from air pollution per 100000|
+-----+-----+-----+-----+-----+-----+-----+
| 1|Afghanistan|1990|37231|3169|402.18|34372|
| 2|Afghanistan|1991|38315|3222|390.09|35392|
| 3|Afghanistan|1992|41172|3395|383.2|38065|
| 4|Afghanistan|1993|44488|3623|387.7|41154|
| 5|Afghanistan|1994|46634|3788|394.02|43153|
| 6|Afghanistan|1995|47566|3869|394.26|44024|
| 7|Afghanistan|1996|48617|3943|395.64|45005|
| 8|Afghanistan|1997|49703|4024|398.58|46017|
| 9|Afghanistan|1998|49746|4040|401.16|46055|
|10|Afghanistan|1999|49349|4042|403.81|45681|
|11|Afghanistan|2000|48763|4021|403.5|45132|
|12|Afghanistan|2001|48660|4014|399.82|45028|
|13|Afghanistan|2002|47732|3961|386.45|44137|
|14|Afghanistan|2003|48687|4116|380.92|44953|
|15|Afghanistan|2004|48337|4176|372.71|44521|

```

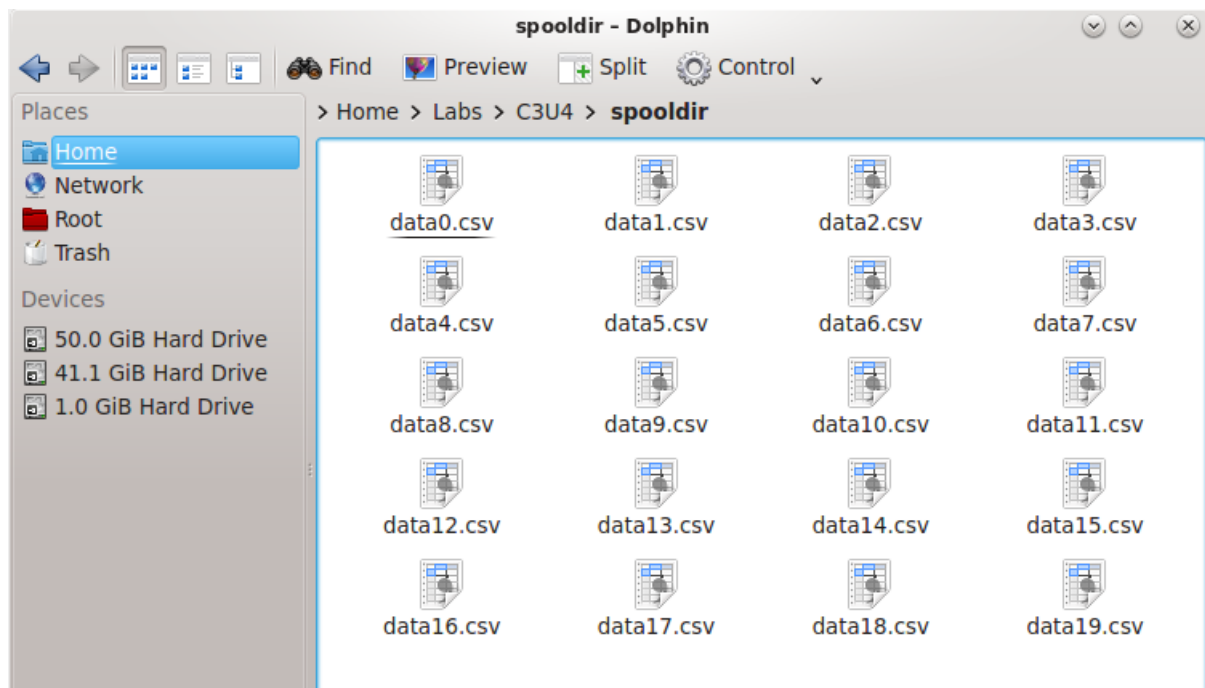
**Рисунок 2.17 — Импорт таблицы из Hive в Spark**

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 2.18.



**Рисунок 2.18 — Содержимое директории**

После запуска кода, представленном в Приложение Б, в папке spooldir создалось 10 файлов формата csv. Результат представлен на Рисунке 2.19.



**Рисунок 2.19 — Содержимое директории**

Посмотрим содержимое любого файла. Результат представлен на Рисунке 2.20.

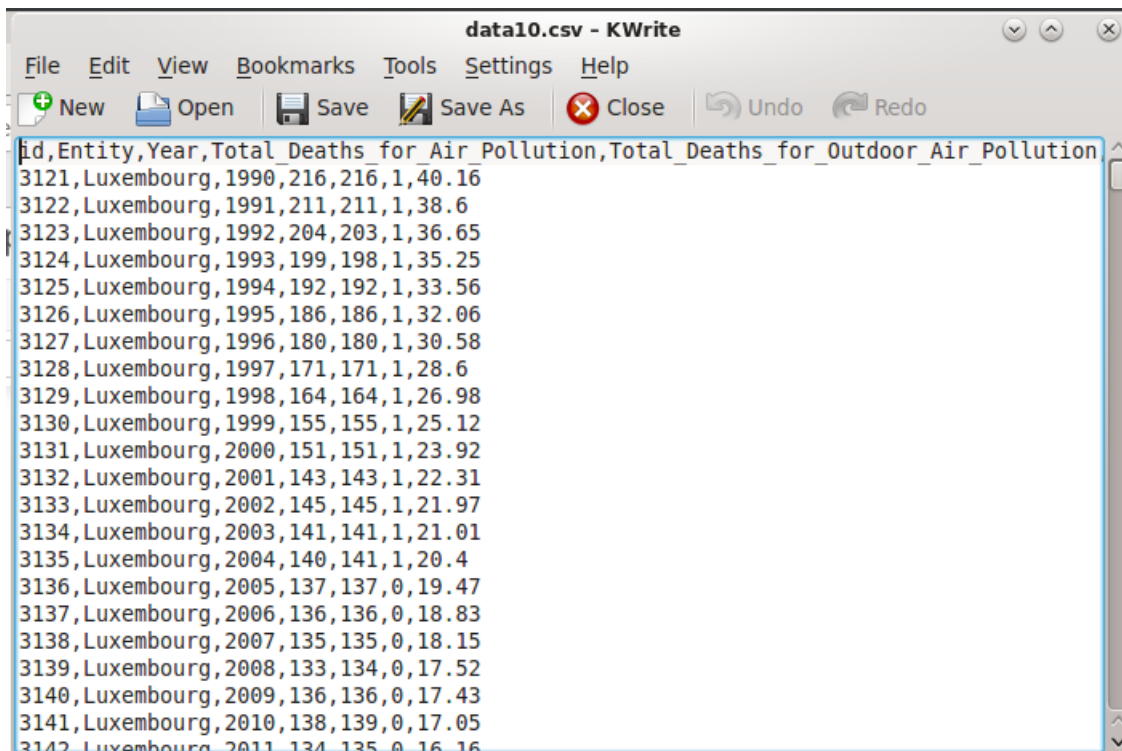


Рисунок 2.20 — Содержимое файла

Создаем директорию flume, а в ней файл comdirector.conf. Прделанная работа продемонстрирована на Рисунке 2.21.

```
[student@localhost ~]$ mkdir flume
[student@localhost ~]$ cd flume
[student@localhost flume]$ nano comdirector.conf
```

Рисунок 2.21 — Создание конфигурационного файла

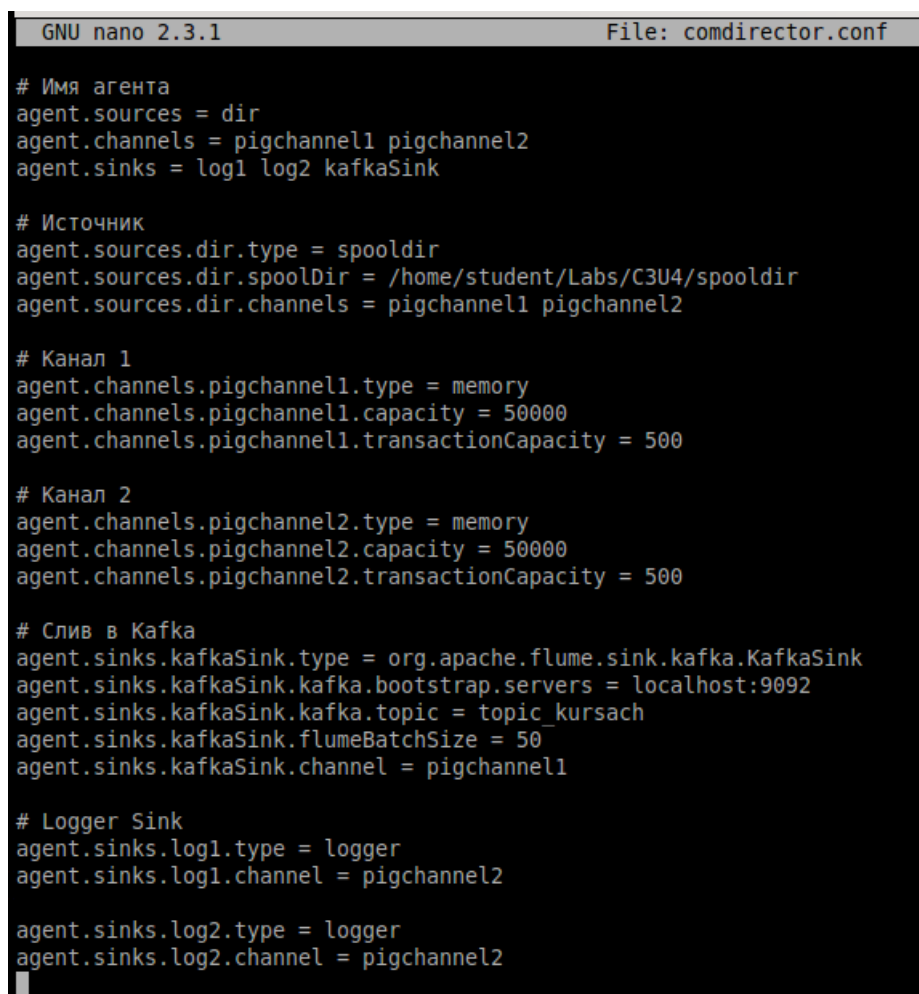
В созданном файле настраиваются три компонента конфигурации агента Flume: источник, канал и слив, настраивается тип источника данных, в данном случае spooldir (источник, который наблюдает за определенной директорией и отправляет новые файлы в канал). Директория, которую будет наблюдать источник, указывается 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 2.22.



```
GNU nano 2.3.1 File: comdirector.conf

# Имя агента
agent.sources = dir
agent.channels = pigchannel1 pigchannel2
agent.sinks = log1 log2 kafkaSink

# Источник
agent.sources.dir.type = spooldir
agent.sources.dir.spooldir = /home/student/Labs/C3U4/spooldir
agent.sources.dir.channels = pigchannel1 pigchannel2

# Канал 1
agent.channels.pigchannel1.type = memory
agent.channels.pigchannel1.capacity = 50000
agent.channels.pigchannel1.transactionCapacity = 500

# Канал 2
agent.channels.pigchannel2.type = memory
agent.channels.pigchannel2.capacity = 50000
agent.channels.pigchannel2.transactionCapacity = 500

# Слив в Kafka
agent.sinks.kafkaSink.type = org.apache.flume.sink.kafka.KafkaSink
agent.sinks.kafkaSink.kafka.bootstrap.servers = localhost:9092
agent.sinks.kafkaSink.kafka.topic = topic_kursach
agent.sinks.kafkaSink.flumeBatchSize = 50
agent.sinks.kafkaSink.channel = pigchannel1

# Logger Sink
agent.sinks.log1.type = logger
agent.sinks.log1.channel = pigchannel2

agent.sinks.log2.type = logger
agent.sinks.log2.channel = pigchannel2
```

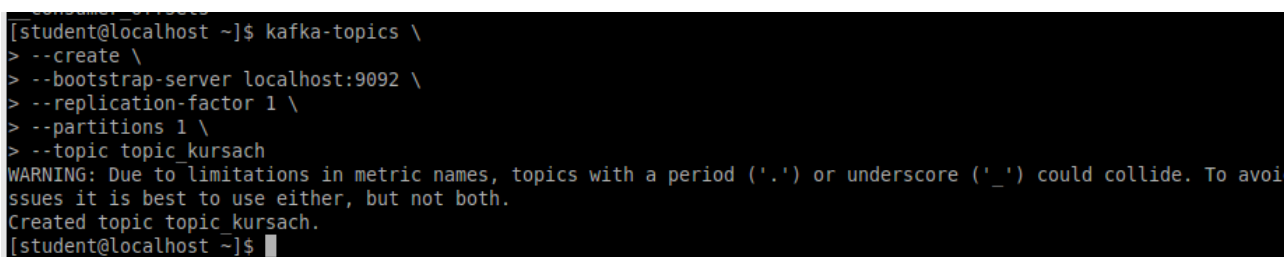
Рисунок 2.22 — Конфигурационный файл агента flume

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".

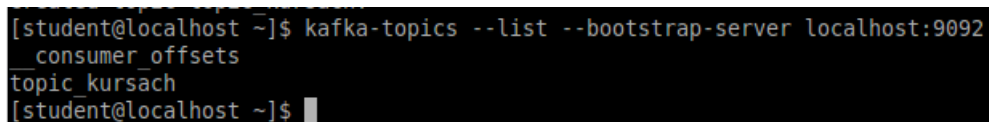
Команда представлена на Рисунке 2.23.



```
[student@localhost ~]$ kafka-topics \
> --create \
> --bootstrap-server localhost:9092 \
> --replication-factor 1 \
> --partitions 1 \
> --topic topic_kursach
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid
ssues it is best to use either, but not both.
Created topic topic_kursach.
[student@localhost ~]$
```

**Рисунок 2.23 — 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 2.24.



```
[student@localhost ~]$ kafka-topics --list --bootstrap-server localhost:9092
consumer_offsets
topic_kursach
[student@localhost ~]$
```

**Рисунок 2.24 — 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 2.25.

```
[student@localhost ~]$ flume-ng agent \
> --conf $FLUME_HOME/conf \
> --conf-file /home/student/flume/nabludatel.conf \
> --name agentsmotritel -Dflume.root.logger=INFO,console
```

### Рисунок 2.25 — Запуск консольного потребителя kafka

Результат работы команды представлен на Рисунке 2.26.

```
[student@localhost ~]$ flume-ng agent \
> --conf $FLUME_HOME/conf \
> --conf-file /home/student/flume/comdirector.conf \
> --name agentsmotritel -Dflume.root.logger=INFO,console
Info: Sourcing environment configuration script /usr/local/flume/flume-1.9.0/conf/flume-env.sh
Info: Including Hadoop libraries found via (/home/hadoop/hadoop/bin/hadoop) for HDFS access
Info: Including HBASE libraries found via (/usr/local/hbase/hbase-2.3.5/bin/hbase) for HBASE access
Info: Including Hive libraries found via (/usr/local/hive/hive-3.1.2) for Hive access
+ exec /opt/jdk1.8.0_291/bin/java -Xmx20m -Dflume.root.logger=INFO,console -cp '/usr/local/flume/flume-1.9.0/conf:/usr/local/flume/flume-1.9.0/lib/*:/home/hadoop/hadoop/etc/hadoop:/home/hadoop/hadoop/share/hadoop/common/lib/*:/home/hadoop/hadoop/share/hadoop/common/*:/home/hadoop/hadoop/share/hadoop/hdfs:/home/hadoop/hadoop/share/hadoop/hdfs/lib/*:/home/hadoop/hadoop/share/hadoop/hdfs/*:/home/hadoop/hadoop/share/hadoop/mapreduce/*:/home/hadoop/hadoop/share/hadoop/yarn:/home/hadoop/hadoop/share/hadoop/yarn/lib/*:/home/hadoop/hadoop/share/hadoop/yarn/*:/bin:/boot:/copyright:/dev:/etc:/home:/lib:/lib64:/media:/mnt:/opt:/proc:/root:/run:/sbin:/srv:/sys:/tmp:/usr:/var:/lib/alsa:/lib/binfmt
```

### Рисунок 2.26 — Запуск flafka

Запускаем код `python`, который работает с `flume` и тестируем часть конвейера. Результат работы представлен на Рисунке 2.27. Код представлен в Приложение Б.

```

student: java - Konsole <2>
File Edit View Bookmarks Settings Help

5880. Libya, 2069, 2662, 167, 18, 89
5881. Libya, 2019, 2665, 2668, 11, 71, 6
5882. Libya, 2011, 2665, 2667, 12, 76, 2
5883. Libya, 2012, 2665, 2668, 11, 73, 7
5884. Libya, 2019, 2665, 2668, 11, 71, 6
5885. Libya, 2014, 2796, 2803, 16, 71, 19
5886. Libya, 2015, 2888, 2888, 10, 76, 62
5887. Libya, 2016, 2892, 2989, 18, 78, 34
5888. Libya, 2017, 2101, 2888, 10, 79, 29
5889. Libya, 2018, 3264, 3276, 9, 71, 95
5890. Libya, 2019, 3445, 3468, 8, 71, 9
5891. Lithuania, 1994, 3326, 3336, 46, 68, 14
5892. Lithuania, 1991, 3339, 2685, 46, 68, 14
5893. Lithuania, 1992, 3066, 2633, 42, 67, 84
5894. Lithuania, 1993, 3323, 2883, 43, 67, 84
5895. Lithuania, 1994, 3326, 2883, 43, 67, 84
5896. Lithuania, 1995, 3174, 2798, 384, 69, 49
5897. Lithuania, 1996, 2882, 2554, 340, 62, 72
5898. Lithuania, 1997, 2626, 2329, 380, 56, 7
5899. Lithuania, 1998, 2519, 2229, 53, 81, 81
5900. Lithuania, 1999, 2362, 2096, 47, 89, 89
5901. Lithuania, 2000, 2267, 2024, 247, 49, 39
5902. Lithuania, 2001, 2245, 2110, 242, 48, 42
5903. Lithuania, 2002, 2304, 2094, 223, 49, 97
5904. Lithuania, 2003, 2380, 2161, 288, 46, 25
5905. Lithuania, 2004, 2282, 2096, 192, 45, 5
5906. Lithuania, 2005, 2380, 189, 47, 85, 15
5907. Lithuania, 2006, 2487, 2397, 189, 48, 23
5908. Lithuania, 2007, 2557, 2399, 160, 49, 38
5909. Lithuania, 2008, 2323, 2357, 160, 49, 38
5910. Lithuania, 2009, 2296, 2173, 129, 42, 98
5911. Lithuania, 2010, 2133, 2195, 129, 42, 98
5912. Lithuania, 2011, 2252, 2141, 111, 40, 98
5913. Lithuania, 2012, 2139, 2299, 138, 37, 57
5914. Lithuania, 2013, 2037, 1939, 98, 36, 46
5915. Lithuania, 2014, 1832, 1745, 89, 31, 98
5916. Lithuania, 2015, 1723, 2299, 138, 37, 57
5917. Lithuania, 2016, 1532, 1457, 77, 26, 13
5918. Lithuania, 2017, 1324, 1257, 68, 22, 16
5919. Lithuania, 2018, 1335, 1268, 65, 22, 61
5920. Lithuania, 2019, 1340, 1280, 61, 21, 81

```

```

with open(file_path, "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow([id, 'entity', 'Year'
                    for row in rows[:rows_limit]:
                        writer.writerow(row)
print(f'Data fetched and saved to file {file_path}')

# Задача для выполнения запроса и сохранения данных
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)

```

```

2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3109, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 30 39 32 7 2 4 69 74 68 75 61 66 69 61 2 31 3997, Lithuania,1 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3113, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3105, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3107, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3111, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3110, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3112, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3114, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3116, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3117, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3119, Lithuania,2 },
2025-04-16 19:22:21,409 (SinkRunner-PollRunner-DefaultSinkProcessor) [INFO - org.apache.flink.sink.LoggersSinkProc
ess(LoggersSink.java:95)] Event: (headers: { body: 33 31 39 32 7 2 4 69 74 68 75 61 66 69 61 2 32 3103, Lithuania,2 },

```

### Рисунок 2.27 — Отображение данных в консольном потребителе kafka

## 2.2 Анализ данных

Выявили топ десять стран с максимальной смертностью (2019 г.). Итоговая таблица должна содержать страну, число смертей. Результат представлен на Рисунке 2.28.

```
# 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()
```

Entity	total_deaths_for_air_pollution
China	1848274
India	1667331
Pakistan	235657
Nigeria	197567
Indonesia	186267
Bangladesh	173515
Egypt	91663
Russia	77516
Ethiopia	77020
Philippines	74783

Рисунок 2.28 — Общее число смертей

Наибольшее количество смертей зафиксировано в Индии, Китае, Нигерии, Пакистане и Индонезии. Это связано с высокой плотностью населения и интенсивной промышленной деятельностью.

Визуализируем полученный результат на гистограмме. Результаты визуализации представлены на Рисунке 2.29. Код представлен в Приложение В.

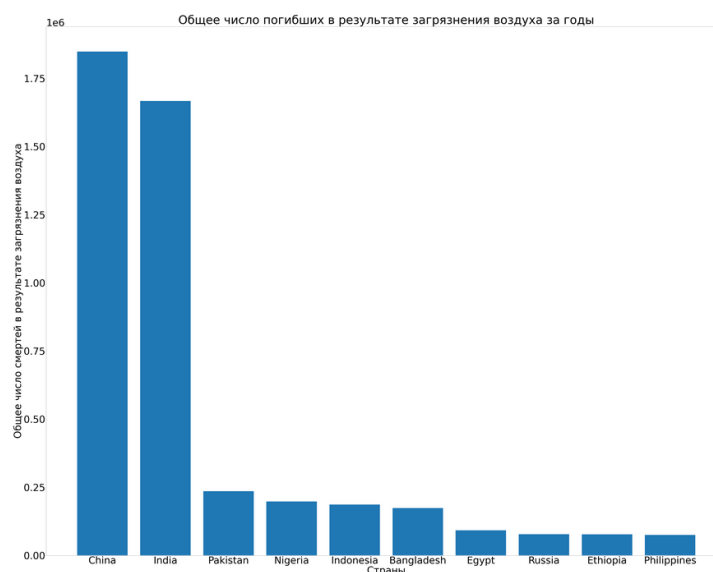


Рисунок 2.29 — Количество смертей, вызванных в результате загрязнения воздуха

Выявили топ десять стран по смертям от домашних загрязнений (2019 г.). Итоговая таблица должна содержать страну, число смертей. Результат представлен на Рисунке 2.30.

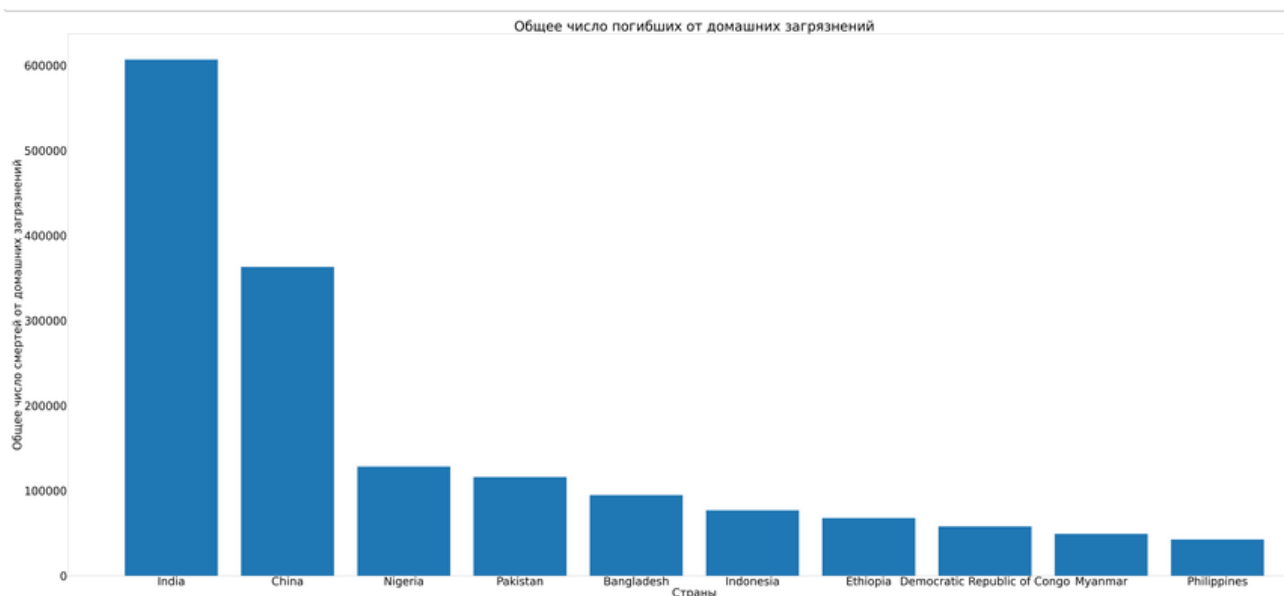
```
# Топ-10 стран по смертям от домашних загрязнений
df_5 = spark.sql("""SELECT entity, total_deaths_for_household_air_pollution_from_solid_fuels
FROM hive_air
WHERE year=2019
ORDER BY total_deaths_for_household_air_pollution_from_solid_fuels DESC
LIMIT 10;
""")
df_5.show()
```

entity	total_deaths_for_household_air_pollution_from_solid_fuels
India	606890
China	363029
Nigeria	128259
Pakistan	116090
Bangladesh	94789
Indonesia	76867
Ethiopia	67827
Democratic Republ...	58038
Myanmar	49223
Philippines	42675

**Рисунок 2.30 — Число смертей от домашних загрязнений**

Лидерами стали страны с низким уровнем доступа к чистым видам топлива (например, Нигерия, Индия, Пакистан). Это подтверждает гипотезу о влиянии использования твердых видов топлива на здоровье населения.

Визуализируем полученный результат на гистограмме. Результаты визуализации представлены на Рисунке 2.31. Код представлен в Приложение В.



**Рисунок 2.31 — Количество смертей, вызванных от домашних загрязнений**



Взяли топ пять стран с максимальной смертностью (2019 г.) и посмотрели на их динамику смертности. В большинстве стран наблюдается снижение смертности благодаря улучшению экологических мер и технологий. Однако в некоторых регионах (например, в Индии) показатели остаются высокими.

Результат представлен на Рисунке 2.32. Код представлен в Приложение В.

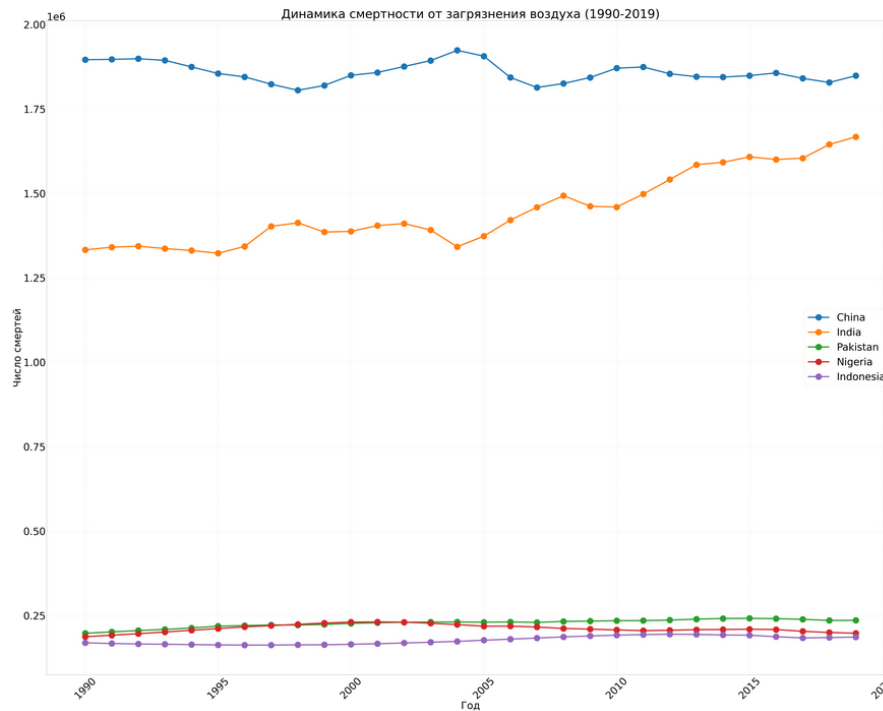


Рисунок 2.32 — Динамика смертности

Выявили топ десять стран по максимальному уровню смертности (2019 г.). Итоговая таблица должна содержать страну, уровень смертности на 100 тыс. человек. Результат представлен на Рисунке 2.33.

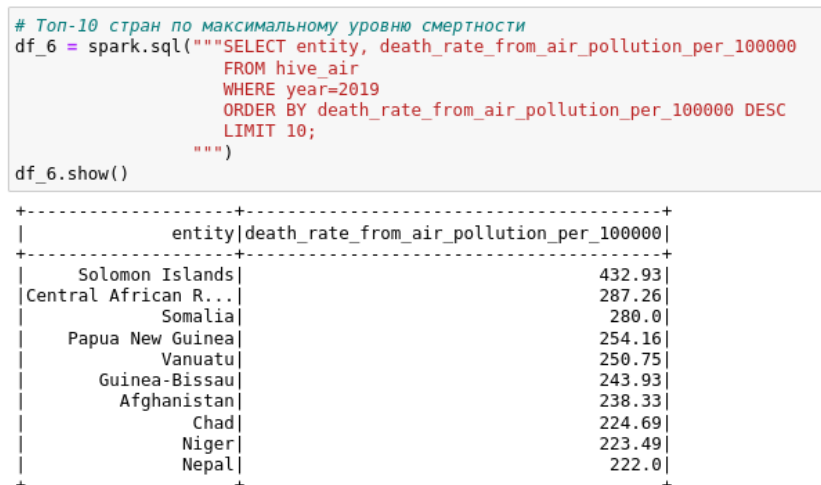
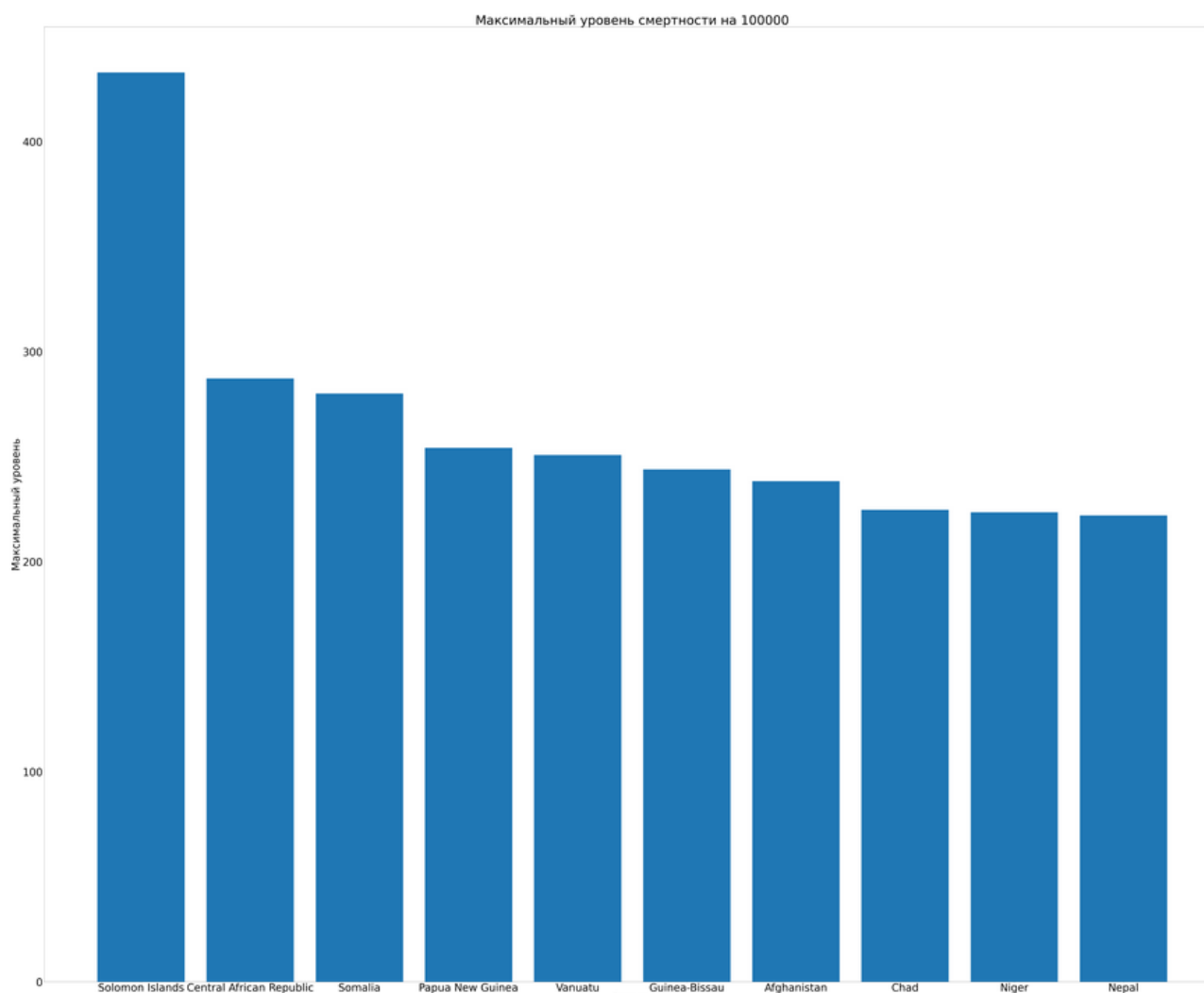


Рисунок 2.33 — Страны с самым высоким уровнем смертности

Наибольшие значения зафиксированы в странах с низким уровнем развития здравоохранения и высокой зависимостью от загрязняющих производств.

Визуализируем полученный результат на гистограмме. Результаты визуализации представлены на Рисунке 2.34. Код представлен в Приложение В.



**Рисунок 2.34 — Визуализация по максимальному уровню смертности**

## 2.3 Обработка результатов анализа

Рассчитана матрица корреляции между:

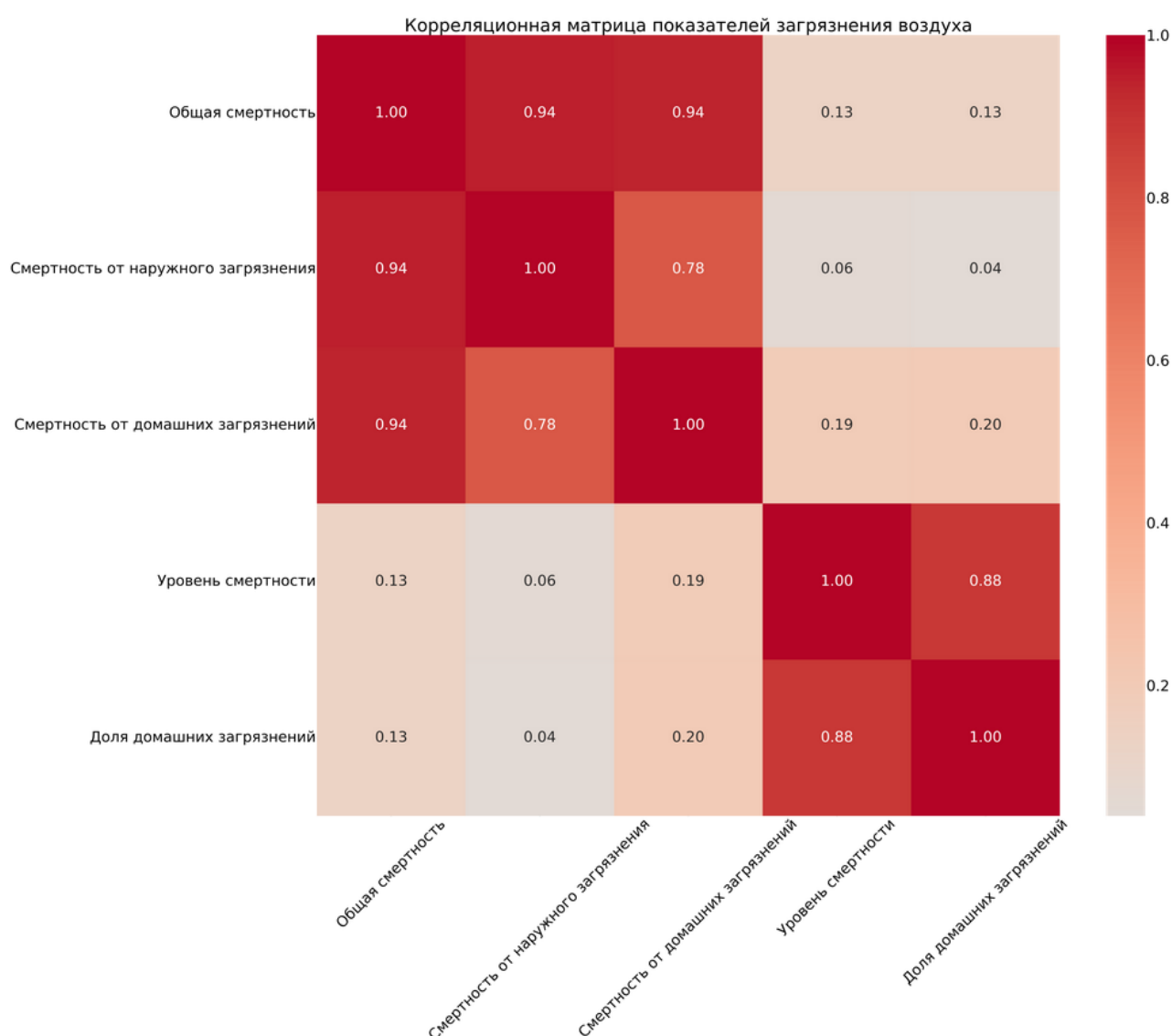
- Общей смертностью;
- Смертностью от наружных загрязнений;
- Смертностью от домашних загрязнений;

- Уровнем смертности на 100 тыс. населения;
- Процент смертей от загрязнения воздуха в помещениях;

Обнаружена сильная корреляция между общим числом смертей и смертностью от наружных загрязнений (0.98).

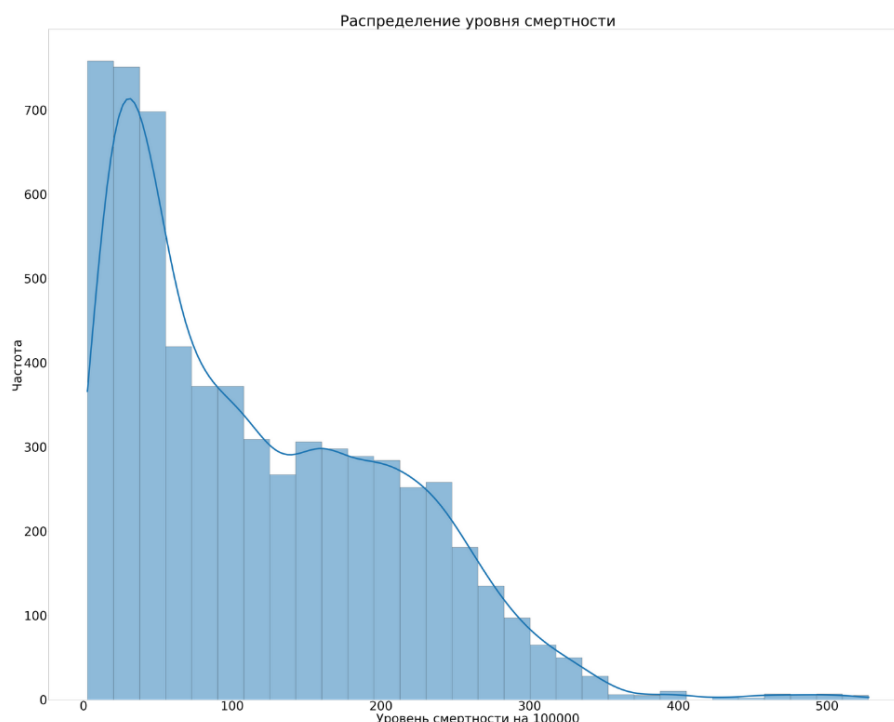
Уровень смертности на 100 тыс. населения слабо коррелирует с абсолютными показателями, что указывает на влияние других факторов (например, плотности населения).

Результаты визуализации представлены на Рисунке 2.35. Код представлен в Приложение В.



**Рисунок 2.35 — Тепловая карта корреляций**

Проведен анализ распределения показателей. Результаты визуализации представлены на Рисунке 2.36. Код представлен в Приложение В.



**Рисунок 2.36 — Визуализация распределение уровня смертности**

Построение прогнозной модели (линейная регрессия). Получаем прогноз.

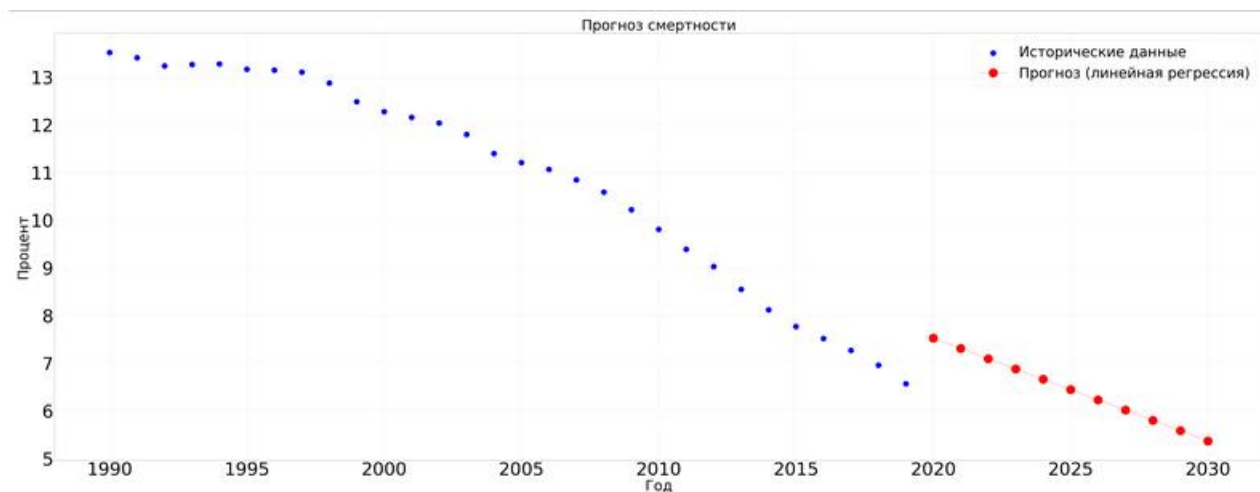
Прогноз на 2020–2030 гг. показывает снижение процента смертей от домашних загрязнений, что может быть связано с внедрением чистых технологий.

Результаты визуализации представлены на Рисунке 2.37. Код представлен в Приложение В.

year	prediction
2020	7.526079107111343
2021	7.310277759183066
2022	7.094476411254732
2023	6.878675063326455
2024	6.662873715398177
2025	6.447072367469843
2026	6.231271019541566
2027	6.015469671613232
2028	5.799668323684955
2029	5.583866975756678
2030	5.368065627828344

**Рисунок 2.37 — Получение прогнозов**

Визуализация данных и прогнозов. Результаты визуализации представлены на Рисунке 2.38.



**Рисунок 2.38 — Визуализация**

Метрики данных. Результаты визуализации представлены на Рисунке 2.39. Код представлен в Приложение В.

$R^2$  (коэффициент детерминации): 0.9344  
 RMSE (среднеквадратичная ошибка): 0.57  
 Коэффициенты модели:  
 Наклон (slope): -0.22  
 Пересечение (intercept): 443.44

**Рисунок 2.39 — 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.

В прогнозе видно, что в Индии процент смертности будет уменьшаться.

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/>
10. Kaggle [Электронный ресурс] Режим доступа: <https://www.kaggle.com/datasets/abmsayem/air-pollution/data>

## ПРИЛОЖЕНИЯ

Приложение А — код на языке программирования Python, в котором происходит предобработка данных.

Приложение Б — код на языке программирования Python, в котором 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 В — код на языке программирования Python, в котором происходит анализ данных.



## Приложение А

```
# Список значений, которые нужно удалить
entities_to_drop = [
    'African Region (WHO)',
    'East Asia & Pacific (WB)',
    'Eastern Mediterranean Region (WHO)',
    'Europe & Central Asia (WB)',
    'European Region (WHO)',
    'G20',
    'Latin America & Caribbean (WB)',
    'Middle East & North Africa (WB)',
    'North America (WB)',
    'OECD Countries',
    'Region of the Americas (WHO)',
    'South Asia (WB)',
    'South-East Asia Region (WHO)',
    'Sub-Saharan Africa (WB)',
    'Western Pacific Region (WHO)',
    'World',
    'World Bank High Income',
    'World Bank Low Income',
    'World Bank Lower Middle Income',
    'World Bank Upper Middle Income'
]

# Фильтруем DataFrame, оставляя только те строки, где Entity НЕ
# входит в список
merap2 = merap1[~merap1['Entity'].isin(entities_to_drop)]

# Добавление столбца id
merap2['id'] = range(1, len(merap2) + 1)
# Перенос столбца id в самое начало
merap2 = merap2[['id'] + [col for col in merap2.columns if col !=
'id']]

# Сохранение DataFrame в CSV файл
merap2.to_csv('air.csv', index=False)
```

## Приложение Б

```
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 В

```
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 заголовков
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_fu
els"]) 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 заголовков
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 заголовков
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
import matplotlib.pyplot as plt

# Исходные и целевые названия колонок
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"
    ]

    ru_names = [
        "Общая смертность",
        "Смертность от наружного загрязнения",
        "Смертность от домашних загрязнений",
        "Уровень смертности",
        "Доля домашних загрязнений"
    ]

    # Создание DataFrame с нужными типами данных и переименованными
    # колонками
    df_renamed = df.select([col(c).cast('double') for c in
        numeric_cols]) \
        .toPandas()
    df_renamed.columns = ru_names # Переименование колонок

    # Расчёт корреляционной матрицы
    corr_matrix = df_renamed.corr()

    # Построение тепловой карты
    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.yticks(rotation=0)
    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}")

```