СОДЕРЖАНИЕ

HYPERLINK \l "_Toc136647187"

ВВЕДЕНИЕ

Анализ данных — это одно из самых востребованных направлений на сегодняшний день. Специалисты в этой области занимаются обработкой, анализом, очисткой, преобразованием и моделированием больших объемов данных, чтобы выявить полезную информацию, обосновать выводы и поддержать принятие решений. Одной из областей работы в этой сфере является создание конвейера – потокового обработчика для передачи данных в различные инструменты для очистки, хранения, анализа и визуализации результатов.

Специалисты по анализу данных занимаются изучением самых различных явлений: политических, социологических, медицинских, транзакционных и множества других. Извечный вопрос о состоянии рынка заработных плат в различных сферах порождает множество вопросов, для ответов на которые специалисты в области анализа данных собирают огромное количество информации для анализа.

Таким образом, тема данной курсовой работы является актуальной на этот момент и позволяет обозреть рынок заработных плат специалистов в области

Цель данной курсовой работы — проанализировать имеющуюся выборку, ответить на вопросы и интерпретировать результаты.

Из поставленной цели вытекают следующие задачи:

- составить конвейер передачи данных;
- определить ключевые вопросы для проведения анализа;
- очистить и обработать данные;
- анализировать и интерпретировать данные;
- визуализировать полученные результаты.

1 СОЗДАНИЕ КОНВЕЙЕРА ДЛЯ ПЕРЕДАЧИ ДАННЫХ

1.1 Структура и описание данных

Набор данных предоставляет информацию о зарплатах и характеристиках занятости инженеров по данным за 2020-2024 года. Он включает сведения о зарплате, должности, уровне опыта, типе занятости, месте проживания сотрудника, доле удаленной работы, местоположении компании и размере компании.

Файл представлен в формате csv. В наборе данных содержится 11 признаков и более 16000 записей с данными о заработных платах [1.1].

Данные содержат следующие атрибуты:

год, в котором были собраны данные (2024);

- experience_level: уровень опыта сотрудника, категоризированный как SE (Senior Engineer), MI (Mid-Level Engineer), EL (Entry-Level Engineer), EX (Executive Engineer);

тип занятости, такой как полная занятость (FT), частичная занятость (PT), контракт (CT) или фриланс (FL);

должность или роль сотрудника;

заработная плата сотрудника в местной валюте;

валюта, в которой обозначена заработная плата;

заработная плата, преобразованная в доллары США;

- employee_residence: страна проживания сотрудника;
- remote_ratio: коэффициент, указывающий на степень дистанционности работы (0 - без дистанционной работы, 100 - полностью удаленная работа);

местоположение компании, где сотрудник работает;

размер компании категоризированный по количеству сотрудников (S - маленькая, M - средняя, L - крупная).

1.2 Стек используемых технологий

В современном анализе данных доступно множество различных инструментов. В данном исследовании мы сосредоточимся на средствах, специально предназначенных для обработки и хранения обширных объемов информации.

Весь проект будет основан на открытой программной платформе, предназначенной для распределенного хранения и обработки больших объемов данных на кластерах вычислительных узлов – Hadoop. Экосистема Hadoop позволяет использовать большое количество инструментов Apache. Для загрузки данных и их последующей передачи, используется визуальный интерфейс NiFi. Для хранения данных будут использоваться инструменты MariaDB и Apache форк MySQL, обладающая улучшенной производительностью, большим набором функций и лучшей поддержкой хранимых процедур и триггеров, что делает её более гибким и мощным инструментом для управления данными. [1.2].

Особенности MariaDB, которые отличают ее от MySQL:

олее высокая производительность, новые возможности по управлению базами данных и намного меньшее количество ошибок в коде;

спользует более производительный оптимизатор запросов и более безопасные индексы для алгоритмов хранения информации;

асширенные возможности безопасности, включая дополнительные алгоритмы шифрования и механизмы аутентификации;

оддерживает большое количество функциональных команд, которые не поддерживаются в MySQL;

ктивное сообщество разработчиков и открытое управление проектом.

Apache Hive - это инфраструктура для обработки и анализа больших данных в распределенной среде, которая работает поверх Apache Hadoop [1.3]. Hive позволяет пользователям выполнять запросы и анализировать данные, хранящиеся в различных хранилищах, таких как Hadoop Distributed File System, используя SQL-подобный язык запросов HiveQL. Hive автоматически преобразует запросы HiveQL в MapReduce задачи, которые выполняются на кластере Hadoop, обеспечивая масштабируемость и возможность обработки больших объемов данных. Использовать Hive можно через клиент командной строки Beeline, а так же с помощью Hue (Hadoop User Expirience) – веб-интерфейс предназначенный для упрощения работы в экосистеме Hadoop.

Так как хранение и анализ данных будут производится в распределенной файловой системе HDFS, данные инструменты будут наиболее подходящими для хранения и передачи информации. Для автоматизации процессов обработки и передачи данных используется визуальный интерфейс NiFi [1.5].

Для передачи данных из HDFS в MariaDB используется инструмент Apache — приложение с интерфейсом командной строки для передачи данных между реляционными базами данных и Hadoop [1.5].

Для моделирования потоковой передачи данных также будет применяться Apache Kafka — это гибридное решение, объединяющее черты распределенной базы данных и брокера сообщений с возможностью горизонтального масштабирования. Kafka собирает данные от приложений, хранит их в своем распределенном хранилище, группируя по темам (topics), и доставляет компонентам приложения по подписке. [1.6]. При этом сообщения хранятся на различных узлах-брокерах, что обеспечивает высокую доступность и отказоустойчивость. Данные в Kafka будут поступать из Hive через Flume — инструмент, позволяющий управлять потоками данных и передавать их на некоторый пункт назначения [1.7].

Перед проведением анализа и реализацией конвейера, была разработана его схема, показанная на Рисунке 1.1.

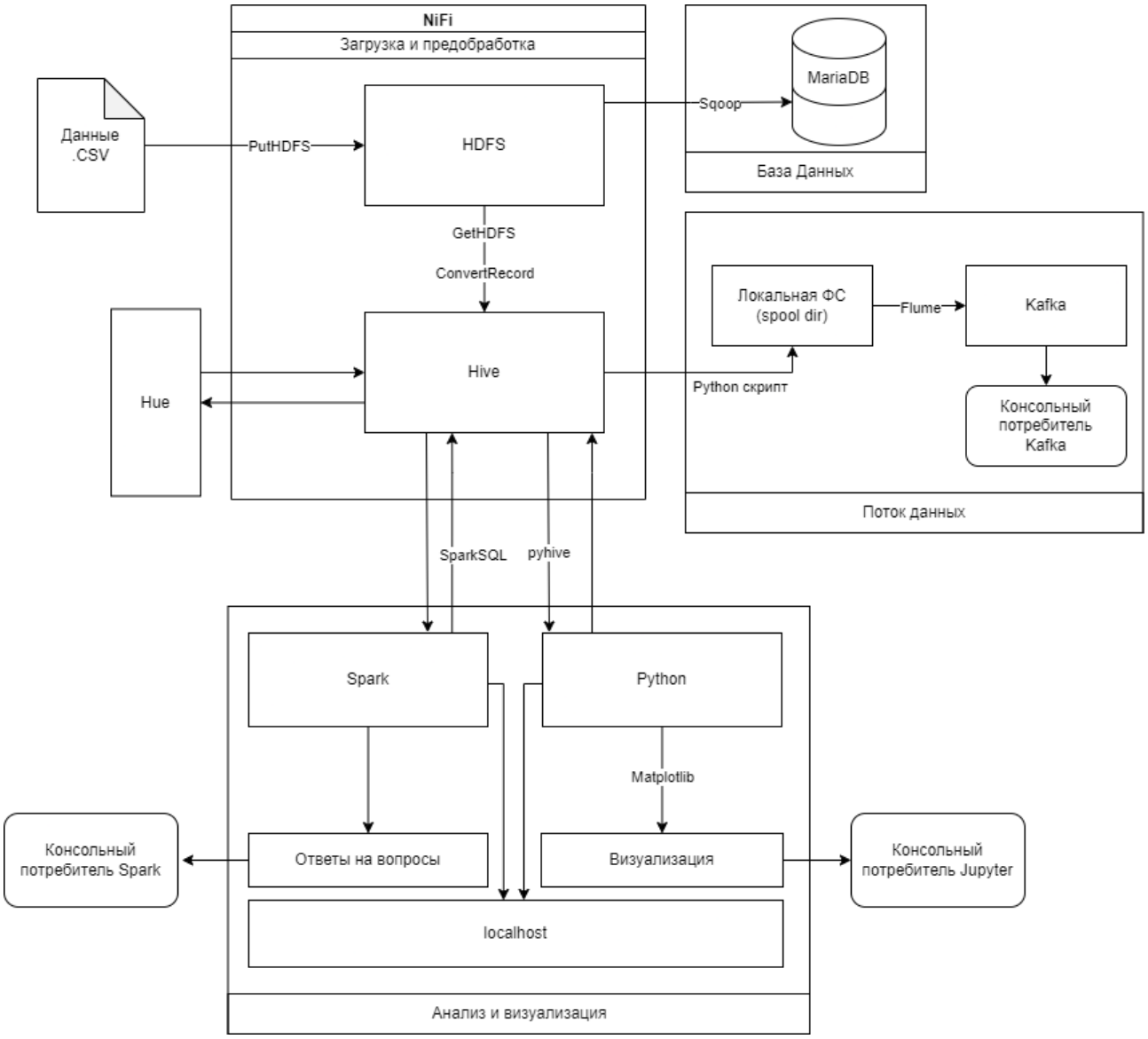

Рисунок 1.1 — Схема конвейера данных

Для загрузки данных из csv файла, воспользуемся интерфейсом Nifi (Рисунок 1.2).

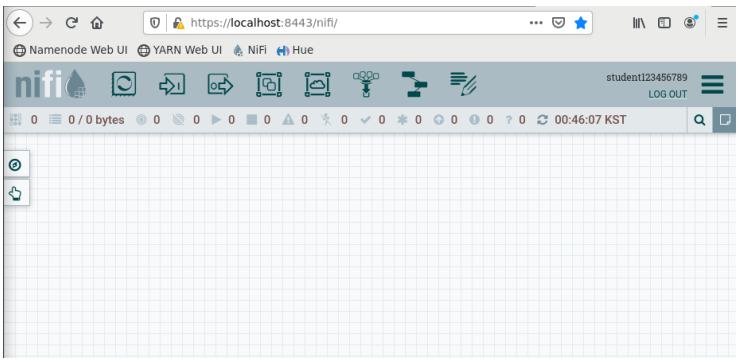

Рисунок 1.2 — Интерфейс NiFi

Создаем два процессора GetFile – получающий файл из локальной файловой системы и PutHDFS – помещающий файл в HDFS (Рисунок 1.3). Настраиваем конфигурацию процессоров, показанную на Рисунке 1.4.

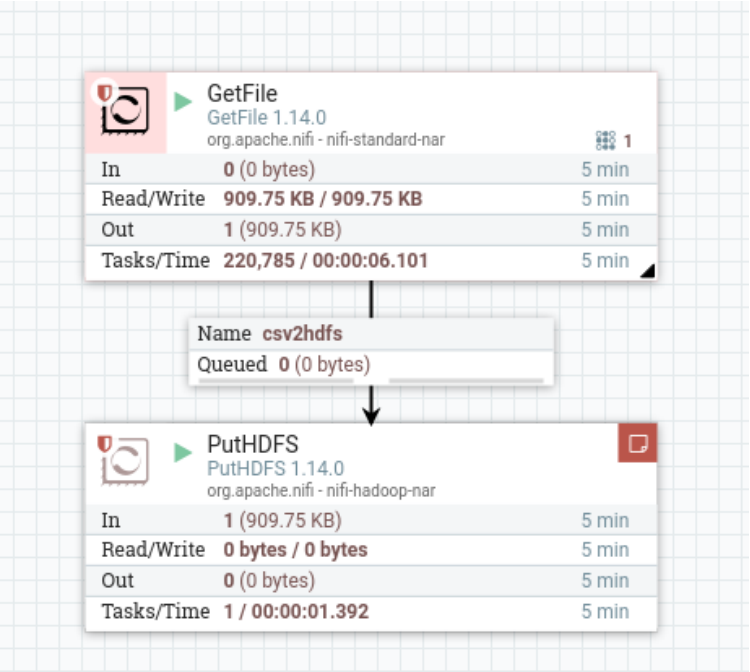

Рисунок 1.3 —Процессоры GetFile и PutHDFS в NiFi

Kerberos Relogin Period	4 hours
Additional Classpath Resources	No value set
Directory	**/user/student/data**
Conflict Resolution Strategy	**fail**
Block Size	No value set
IO Buffer Size	No value set
Replication	No value set
Permissions umask	No value set
Remote Owner	student
Remote Group	student
Compression codec	**AUTOMATIC**
Ignore Locality	false

Рисунок 1.4 — Конфигурация процессора PutHDFS

Проверяем файл в HDFS используя Hue (Рисунок 1.5). После этого, для сохранения данных, воспользуемся MariaDB.

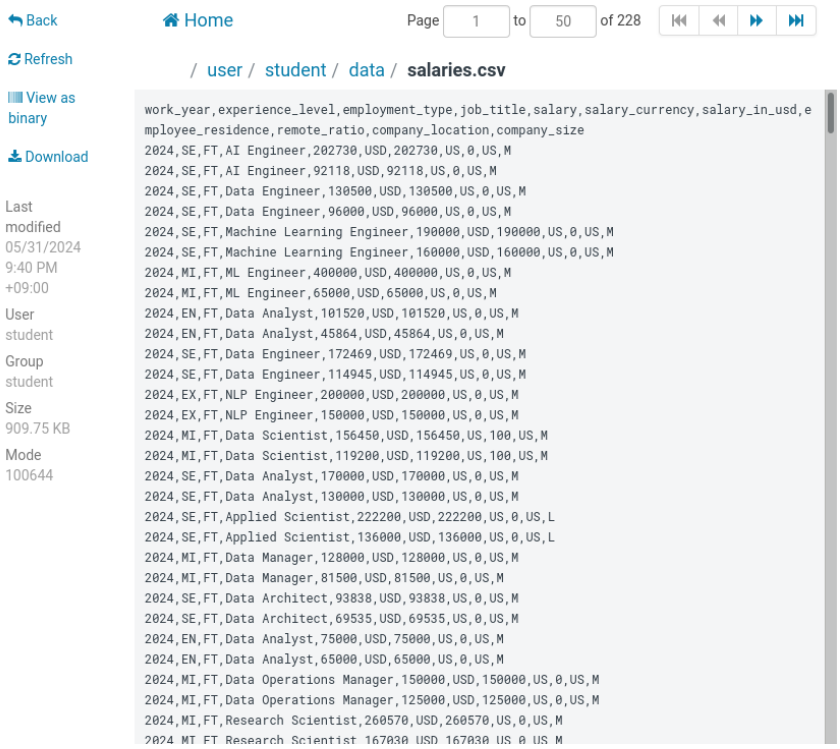

Рисунок 1.5 — Схема загруженного csv файла в HDFS

Через консоль заходим в MariaDB, создаем базу данных project и таблицу salaries, показанную на Рисунке 1.6.

```
MariaDB [project]> describe salaries;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| work_year          | int(11)     | NO   |     | NULL    |       |
| experience_level   | varchar(3)  | YES  |     | NULL    |       |
| employment_type    | varchar(3)  | YES  |     | NULL    |       |
| job_title          | varchar(50) | YES  |     | NULL    |       |
| salary             | int(11)     | YES  |     | NULL    |       |
| salary_currency    | varchar(3)  | YES  |     | NULL    |       |
| salary_in_usd      | int(11)     | YES  |     | NULL    |       |
| employee_residence | varchar(2)  | YES  |     | NULL    |       |
| remote_ratio       | int(11)     | YES  |     | NULL    |       |
| company_location   | varchar(2)  | YES  |     | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
10 rows in set (0.01 sec)
```

Рисунок 1.6 — Таблица salaries в базе данных project

Далее запускаем Zookeeper и Sqoop, с помощью запроса, подключающего таблицу MariaDB с указанием разделителей и одним mapper [1.8].

```
[student@localhost Data]$ sqoop export --connect jdbc:mysql://localhost/project --username student --password student --table salaries3 --export-dir /user/student/salaries2.csv  --fields-terminated-by ',' --lines-terminated-by '\n' --num-mappers 1
```

Рисунок 1.7 — Запрос для переноса данных с помощью Sqoop

Результат работы представлен на Рисунке 1.8: Перенесены все 16535 записей за 20.8934 секунды.

```
2024-06-01 01:40:11,908 INFO mapreduce.ExportJobBase: Transferred 909.877 KB in 20.8934 seconds (4
2024-06-01 01:40:11,916 INFO mapreduce.ExportJobBase: Exported 16535 records.
```

Рисунок 1.8 — Результат работы переноса данных

Проверяем данные в таблице MariaDB (Рисунок 1.9):

```
MariaDB [project]> select * from salaries3 limit 5;
+----------+-----------------+-----------------+--------------+--------+-----------------+--------------+---
---------------+-------------+------------------+
| work_year | expirience_level | employment_type | job_title     | salary | salary_currency | salary_in_usd | em
loyee_residence | remote_ratio | company_location |
+----------+-----------------+-----------------+--------------+--------+-----------------+--------------+---
---------------+-------------+------------------+
| work_year | experience_level | employment_type | job_title     | salary | salary_currency | salary_in_usd | em
loyee_residence | remote_ratio | company_location |
| 2024      | SE               | FT              | AI Engineer   | 202730 | USD             | 202730        | US
                | 0           | US               |
| 2024      | SE               | FT              | AI Engineer   | 92118  | USD             | 92118         | US
                | 0           | US               |
| 2024      | SE               | FT              | Data Engineer | 130500 | USD             | 130500        | US
                | 0           | US               |
| 2024      | SE               | FT              | Data Engineer | 96000  | USD             | 96000         | US
                | 0           | US               |
+----------+-----------------+-----------------+--------------+--------+-----------------+--------------+---
---------------+-------------+------------------+
```

Рисунок 1.9 — Результат работы переноса данных

После загрузки данных в MariaDB перенесем HDFS файл для дальнейшей обработки в Hive. Используем интерфейс NiFi для создание parquet файла для более удобной обработки Hive. Добавляем в новую группу процессоров и PutHDFS (Рисунок 1.10).

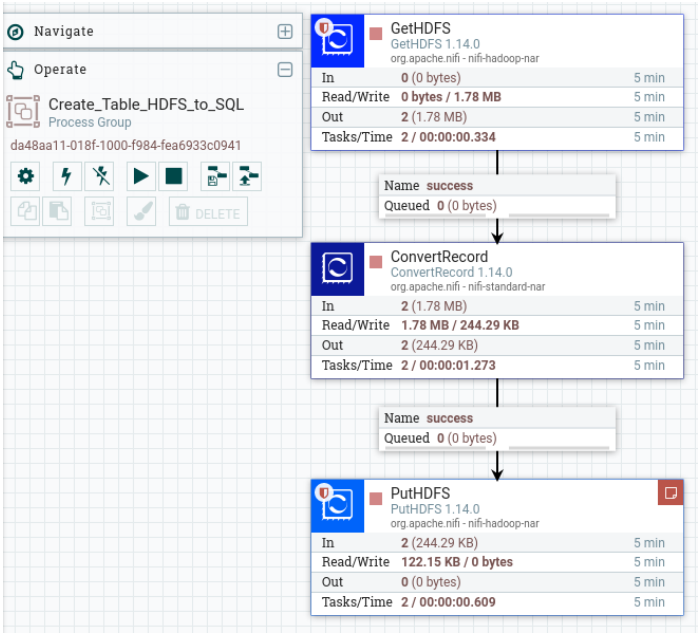

Рисунок 1.10 — Process Group в интерфейсе NiFi для обработки файла

Используя интерфейс Hue, создаем внешнюю таблицу с помощью запроса сразу проверяем её, как показано на Рисунке 1.11:

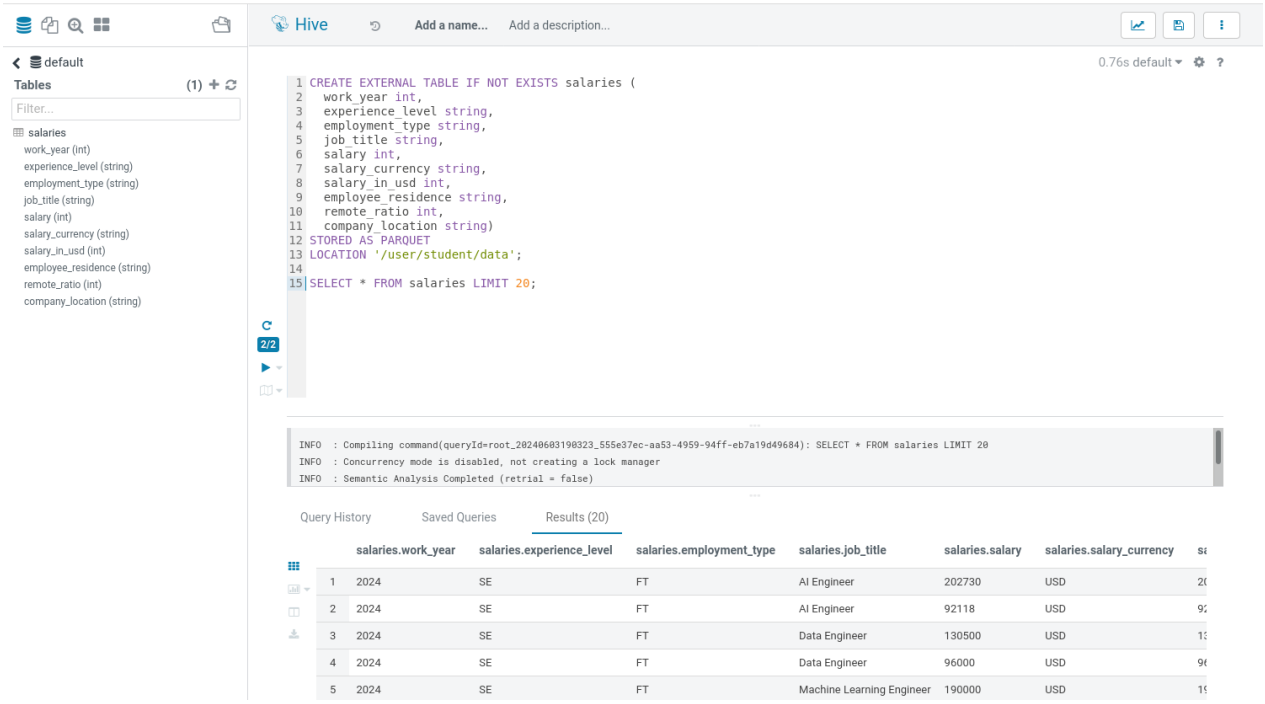

Рисунок 1.11 — Запрос создания таблицы в Hive и вывод результатов

Из данной таблицы, создаем новую – new_salaries, оставляя только те данные, которые необходимы нам для анализа. Запрос для создания и записи в таблицу показан на Рисунке 1.12:

```
CREATE EXTERNAL TABLE IF NOT EXISTS new_salaries (
  employment_type STRING,
  work_year int,
  salary int,
  job_title STRING,
  remote_ratio int,
  experience_level STRING )
LOCATION '/user/student/data';

INSERT OVERWRITE TABLE new_salaries SELECT
  employment_type,
  work_year,
  salary,
  job_title,
  remote_ratio,
  experience_level
FROM salaries;

```

Рисунок 1.12 — Запрос создания новой таблицы new_salaries

Теперь создадим поток данных, используя Apache Kafka и Flume. С помощью библиотеки pyhive загружаем данные из таблицы new_salaries в в Spark. Указываем host, port и username, создаем запрос, перемешиваем результаты и загружаем по 5% данных в локальную файловую систему (папка раз в 10 секунд. Скрипт для этого процесса продемонстрирован на Рисунке 1.13.

```
In [*]: from pyhive import hive
from random import shuffle
from time import sleep
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("DataStreaming").getOrCreate()

conn = hive.Connection(host='localhost', port=10000, username='student')
cursor = conn.cursor()
cursor.execute('SELECT * FROM new_salaries')

records = cursor.fetchall()
conn.close()


while True:
    shuffle(records)
    newData = records[:int(len(records)*0.05)]
    newDF = spark.createDataFrame(newData)
    newDF.write.option("header", "false").mode("append").csv("file:/home/student/spool")
    sleep(10)
```

Рисунок 1.13 — Скрипт для создания потока на Spark использующий pyhive

Файлы начали поступать в папку spool в локальной файловой системе, что можно заметить на Рисунке 1.14:

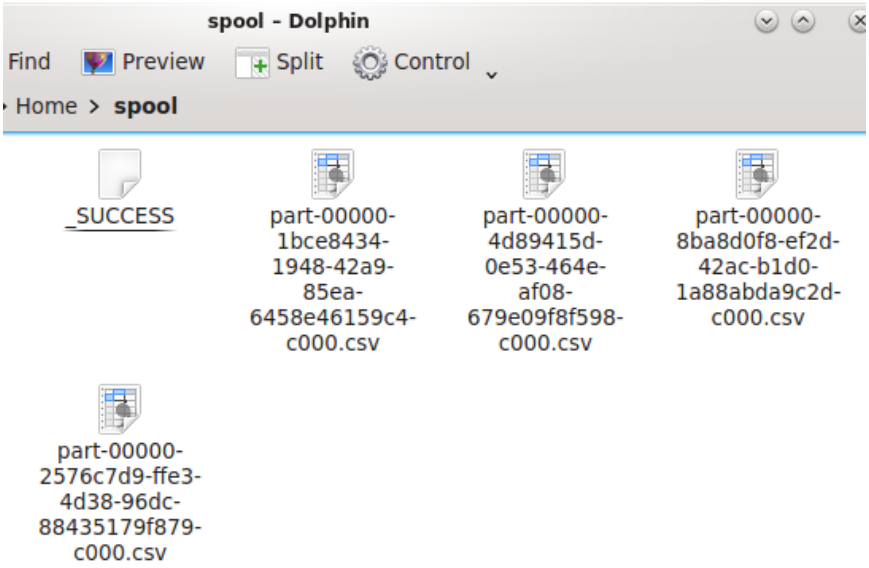

Рисунок 1.14 — Файлы, созданные скриптом в папке spool

Проверяем работу Kafka с помощью systemctl status kafka и создаем топик для получения сообщений на подписке (Рисунок 1.15).

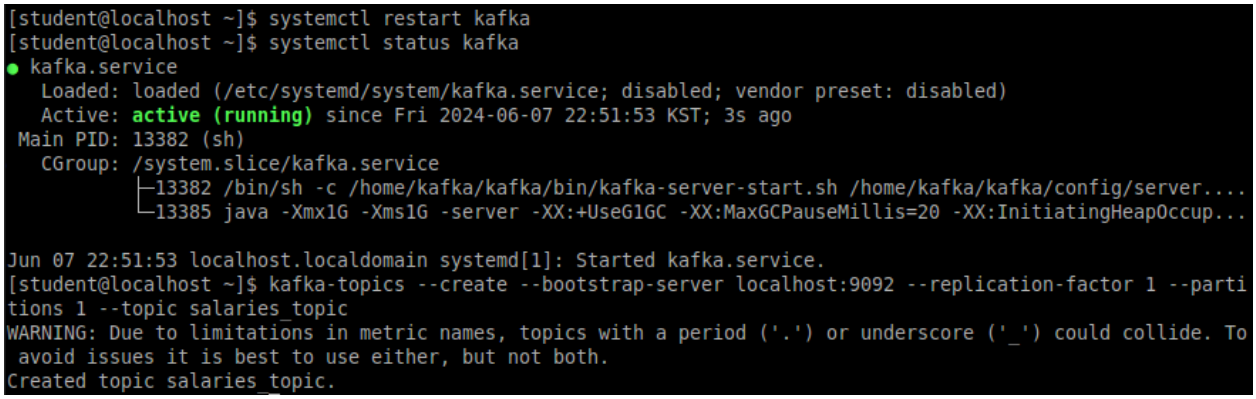

Рисунок 1.15 — Запуск Kafka и создание топика salaries_topic

Создаем flume файл конфигурации для соединения Kafka и локальной файловой системы – задаем потоки, каналы и другие параметры (Листинг 1.1).

Листинг 1.1 — Flume файл конфигурации

```
agent.sources = so1
agent.channels = ch1
agent.sinks = s1 s2
agent.sources.so1.type = spooldir
agent.sources.so1.spoolDir = /home/student/spool
agent.channels.ch1.type = memory
agent.channels.ch1.capacity = 10000
agent.channels.ch1.transactionCapacity = 100

agent.sinks.s1.type = org.apache.flume.sink.kafka.KafkaSink
agent.sinks.s1.kafka.topic = salaries_topic
agent.sinks.s1.kafka.bootstrap.servers = localhost:9092
agent.sinks.s1.kafka.flumeBatchSize = 5

agent.sinks.s2.type = logger
agent.sources.so1.channels = ch1
agent.sources.s1.channels = ch1
agent.sinks.s1.channel = ch1
```

Запускаем агента flume с параметром –Dflume.root.logger для вывода информации в консоль (Листинг 1.2).

Листинг 1.2 — Команда запуска агента Flume

```
flume-ng agent --conf /home/hadoop --conf-file /home/hadoop/flume.conf --name agent
-Dflume.root.logger=INFO,console
```

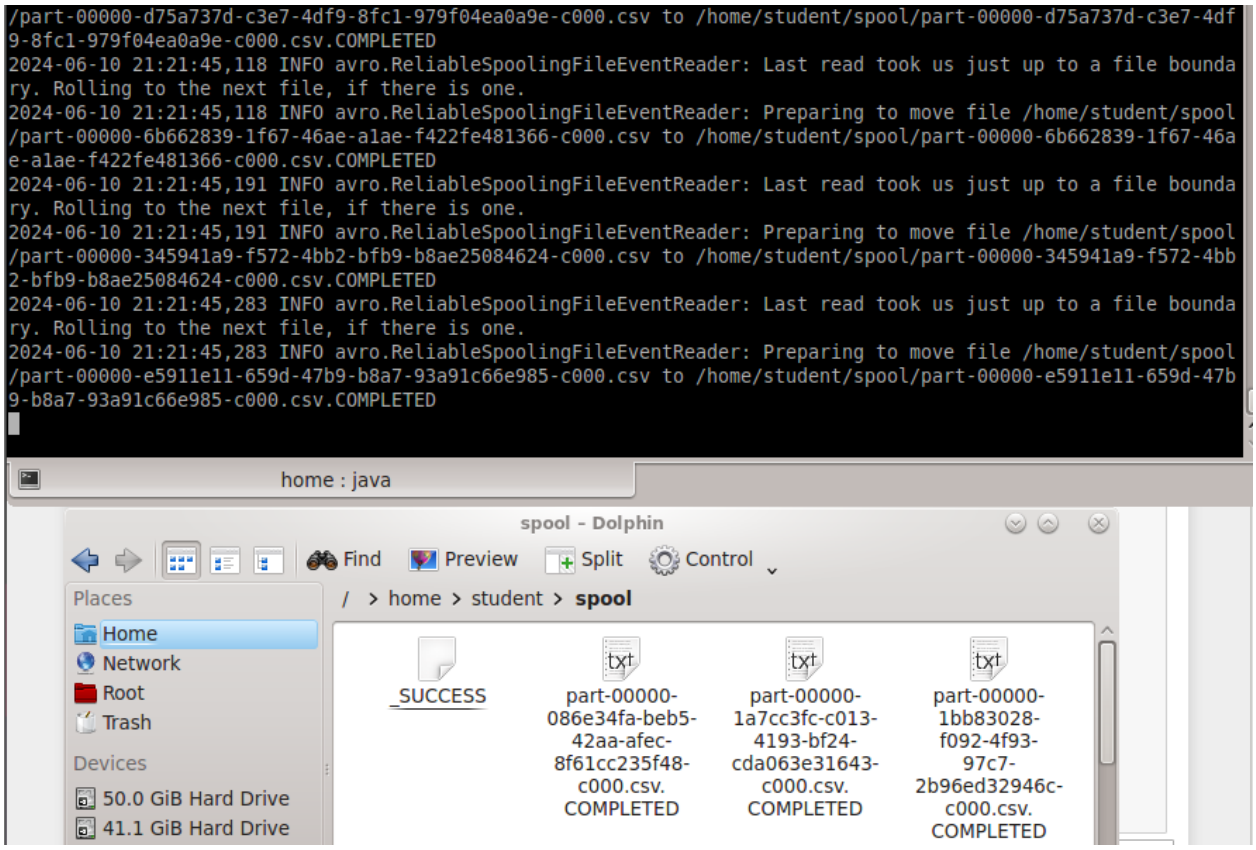

Рисунок 1.16 — Работа агента по передаче данных

Как видно из данного Рисунка 1.16, файлы успешно передаются в Kafka, о чем свидетельствуют логи Flume в консоли и названия файлов с припиской Для просмотра файлов, подписываемся на топик Kafka salaries_topic (Листинг и смотрим на поступающие данные (Рисунок 1.17).

Листинг 1.3 — Команда запуска агента Flume

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic salaries_topic
--from-beginning
```

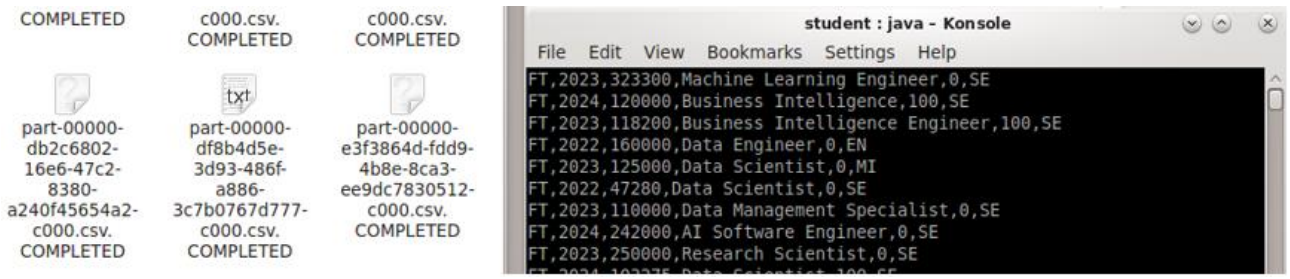

Рисунок 1.17 — Данные, поступающие в консоль Kafka по топику salaries_topic

Таким образом, была создана база данных, содержащая таблицу с данными из исходного файла и поток, создающий файлы с 5% данных, поступающий консольному потребителю Kafka. Далее, предобработанные данные из таблицы будут анализированы для интерпретации результатов и визуализации данных Рисунок 1.18).

```
FT2024202730AI Engineer0SE
FT202492118AI Engineer0SE
FT2024130500Data Engineer0SE
FT202496000Data Engineer0SE
FT2024190000Machine Learning Engineer0SE
FT2024160000Machine Learning Engineer0SE
FT2024400000ML Engineer0MI
FT202465000ML Engineer0MI
FT2024101520Data Analyst0EN
FT202445864Data Analyst0EN
FT2024172469Data Engineer0SE
FT2024114945Data Engineer0SE
```

Рисунок 1.18 — Таблица предобработанных данных new_salaries

2 АНАЛИЗ ПОЛУЧЕННЫХ ДАННЫХ

Все таблицы с анализом, сохраняются обратно в Hive. Для анализа первого вопроса загружаем данные в PySpark из таблицы Hive, используя интерфейс Jupyter и создаем DataFrame. Расшифруем сокращения (наример FT = Full time) и проведем подсчет каждого типа занятости для понимания трендов в рынке труда специалистов Data Engineer (Рисунок 2.1).

```
In [20]: from pyspark.sql import SparkSession
         from pyspark.sql import functions as F

         spark = SparkSession.builder \
                 .appName("ImportTable") \
                 .getOrCreate()

         df = spark.read.table("new_salaries")

         employment_type_new = df.withColumn (
             'employment_type',
             F.when(F.col('employment_type') == 'FT', 'Full time')
              .when(F.col('employment_type') == 'PT', 'Part time')
              .when(F.col('employment_type') == 'CT', 'Contract')
              .when(F.col('employment_type') == 'FL', 'Freelance')
              .otherwise(F.col('employment_type'))
         )

         employment_type_count = employment_type_new.groupBy("employment_type").count()

         sorted_emp_type_count = employment_type_count.orderBy(F.desc('count'))
         sorted_emp_type_count.show()

         [Stage 6:================================================>      (177 + 1) / 200]

         +---------------+-----+
         |employment_type|count|
         +---------------+-----+
         |      Full time|16454|
         |      Part time|   38|
         |       Contract|   28|
         |      Freelance|   14|
         +---------------+-----+
```

Рисунок 2.1 — Подсчет количества типов занятости среди специалистов

Результат работы скрипта:

- Полная занятость – 16454;
- Частичная занятость – 38;
- Контракт – 28;
- Фриланс – 14;

Распределение типов занятости среди специалистов Data Engineer не сбалансировано – подавляющее большинство позиций приходится на полную занятость (Full time), что значительно превышает количество других типов занятости: остальные занятости занимают менее 1% от общего количества. Для

наглядности, визуализируем эти данные круговой диаграммой (Рисунок 2.2).

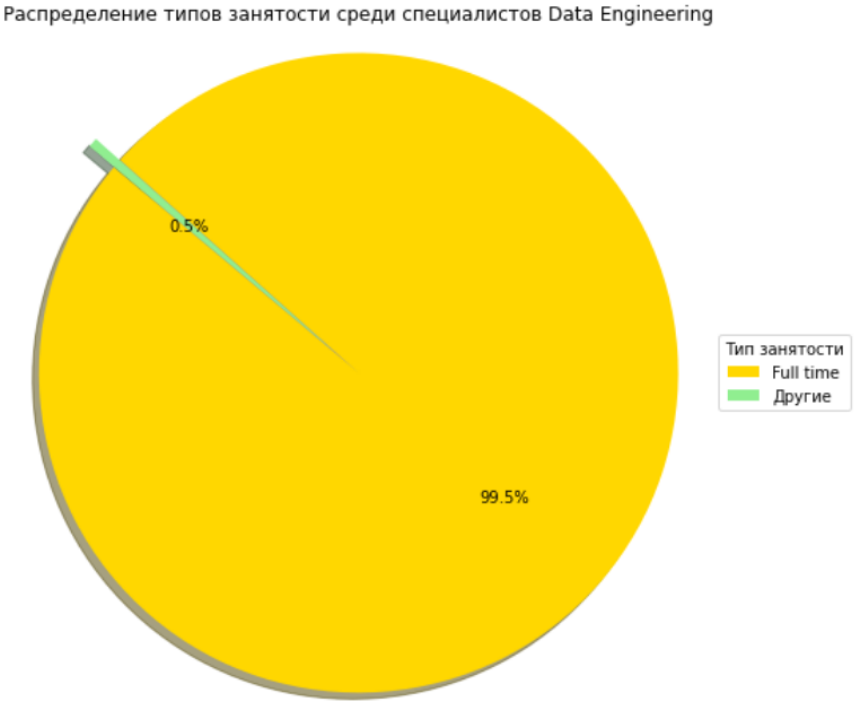

Рисунок 2.2 — Круговая диаграмма распределения типов занятости

Из полученных данных, мы можем сделать некоторые выводы:

реобладание полной занятости: большинство компаний предпочитают нанимать сотрудников на полную занятость. Это может быть связано с необходимостью постоянного присутствия сотрудников для выполнения долгосрочных проектов и задач, требующих стабильности и предсказуемости.

ебольшое количество других видов занятости: низкое количество позиций для частичной занятости, контрактов и фриланса может указывать на ограниченный спрос на гибкие формы занятости в сфере инженерии данных. Возможно, такие позиции чаще встречаются в других отраслях или компаниях с другими бизнес-моделями.

реимущества для сотрудников: полная занятость часто предоставляет более стабильные условия, такие как социальные льготы, медицинское страхование и возможности для карьерного роста, что может привлекать больше специалистов.

Анализ этих данных подчеркивает сильное предпочтение полной занятости в

сфере инженерии данных в 2024 году и маленькую гибкость в занятости.

Теперь проанализируем данные средних заработных плат по годам. Этот анализ помогает понять, как изменяются заработные платы в зависимости от экономических условий, спроса на специалистов и других факторов в области.

Используем salary_in_usd для стандартизации и более доступного понимания. Группируем зарплаты по годам и расчитываем среднее, скрипт показан на Рисунке 2.3.

```
In [10]: from pyspark.sql import functions as F

average_salary_by_year = df.groupBy("work_year") \
    .agg(F.format_number(F.avg("salary_in_usd"), 2).alias("average_salary"))

sorted_avg_salary = average_salary_by_year.orderBy("work_year")

sorted_avg_salary.show()
```

```
[Stage 7:=============================================>          (167 + 1) / 200]
+---------+--------------+
|work_year|average_salary|
+---------+--------------+
|     2020|    102,250.87|
|     2021|     99,922.07|
|     2022|    134,349.50|
|     2023|    153,732.66|
|     2024|    150,564.10|
+---------+--------------+
```

Рисунок 2.3 — Подсчет средней заработной платы по годам

Визуализируем полученные результаты для просмотра изменения средней заработной платы в области по годам, графики показаны на Рисунках 2.4 и 2.5.

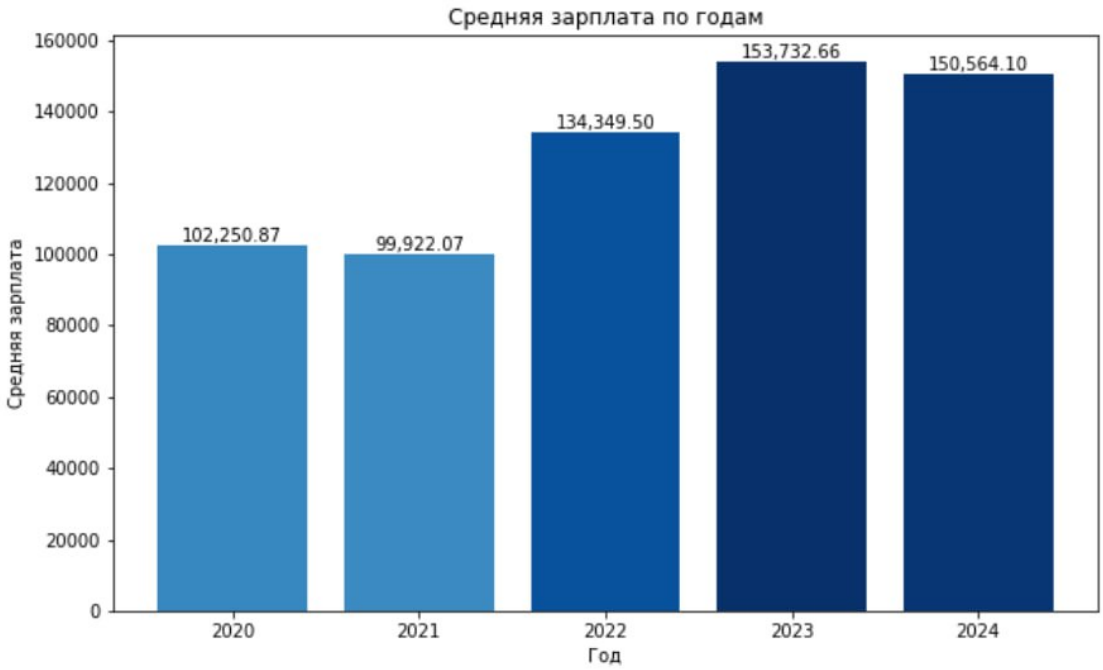

Рисунок 2.4 — Столбчатая диаграмма средней заработной платы по годам

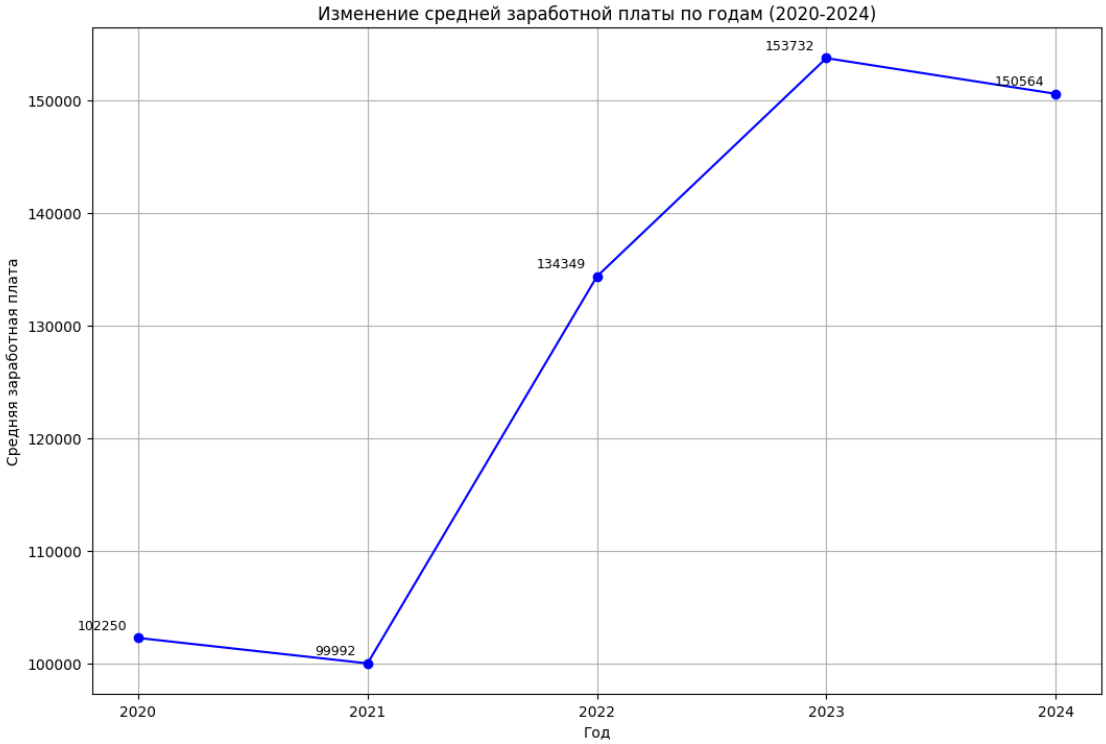

Рисунок 2.5 — График изменения средней зарплаты по годам в области

По полученным данным можно сделать некоторые выводы:

бщий рост зарплат: анализируемый период наблюдается значительное увеличение средних зарплат в области. В 2020 году средняя зарплата составляла 102,250, а к 2024 году она выросла до 150,564. Это указывает на растущий спрос на специалистов в данной области и, как следствие, увеличение их зарплаты.

нижение зарплаты в 2021: скорее всего связано с экономическими последствиями пандемии COVID-19

ущественное увеличение в 2022: В 2022 году средняя зарплата увеличилась до 134,349, что является значительным ростом по сравнению с 2021 годом. Это может свидетельствовать о восстановлении экономики после пандемии и возобновлении активного найма специалистов.

родолжение роста: В 2023 году рост продолжился, и средняя зарплата достигла 153,732. Это может быть связано с увеличением инвестиций в технологии и данных, а также повышением требований к квалификации и навыкам Data Engineers.

табилизация рынка: в 2024 году средняя зарплата снизилась до $150,564. Хотя это снижение незначительное по сравнению с предыдущим годом, оно может указывать на стабилизацию рынка или на увеличение числа специалистов в области, что снизило конкуренцию за таланты и немного выровняло зарплаты.

Исходя из этих данных, можно предположить, что рост квалификации и заработных плат будет продолжать расти, показывая положительный тренд. В целом, профессия остается востребованной и высокооплачиваемой, с большим потенциалом на развитие и масштабирование.

Далее проанализируем наиболее часто встречающиеся должности в Data Подсчет количества каждой должности является важным шагом в анализе данных, так как он позволяет получить представление о распределении ролей внутри компании и помочь понять тенденции на рынке.

Используя скрипт, представленный на рисунке 2.6, получаем количественное значение каждой профессии внутри отрасли.

```
In [13]: from pyspark.sql import functions as F
         from pyspark.sql.functions import col, lower, trim

         df = spark.read.table("new_salaries")
         df_cleaned = df.withColumn("job_title_cleaned", lower(trim(col("job_title"))))

         job_title_count = df_cleaned.groupBy("job_title_cleaned").count()

         job_title_count_sorted = job_title_count.orderBy(col("count").desc())
         job_title_count_sorted.show()

         job_title_count_sorted.write.mode("overwrite").saveAsTable("job_title_count_3")
```

```
+--------------------+-----+
|   job_title_cleaned|count|
+--------------------+-----+
|       data engineer| 3464|
|      data scientist| 3314|
|        data analyst| 2440|
|machine learning ...| 1705|
|  research scientist|  531|
|   applied scientist|  435|
|      data architect|  435|
|  analytics engineer|  431|
|   research engineer|  306|
|        data science|  271|
|business intellig...|  248|
|        data manager|  212|
|         ml engineer|  200|
|business intellig...|  191|
|machine learning ...|  138|
|    research analyst|  123|
|data science manager|  122|
|         ai engineer|  120|
|business intellig...|   98|
|        bi developer|   90|
+--------------------+-----+
only showing top 20 rows
```

Рисунок 2.6 — Подсчет количества каждой должности

Визуализируем эти данные используя столбчатую диаграмму, возьмем только 10 самых популярных должностей (Рисунок 2.7).

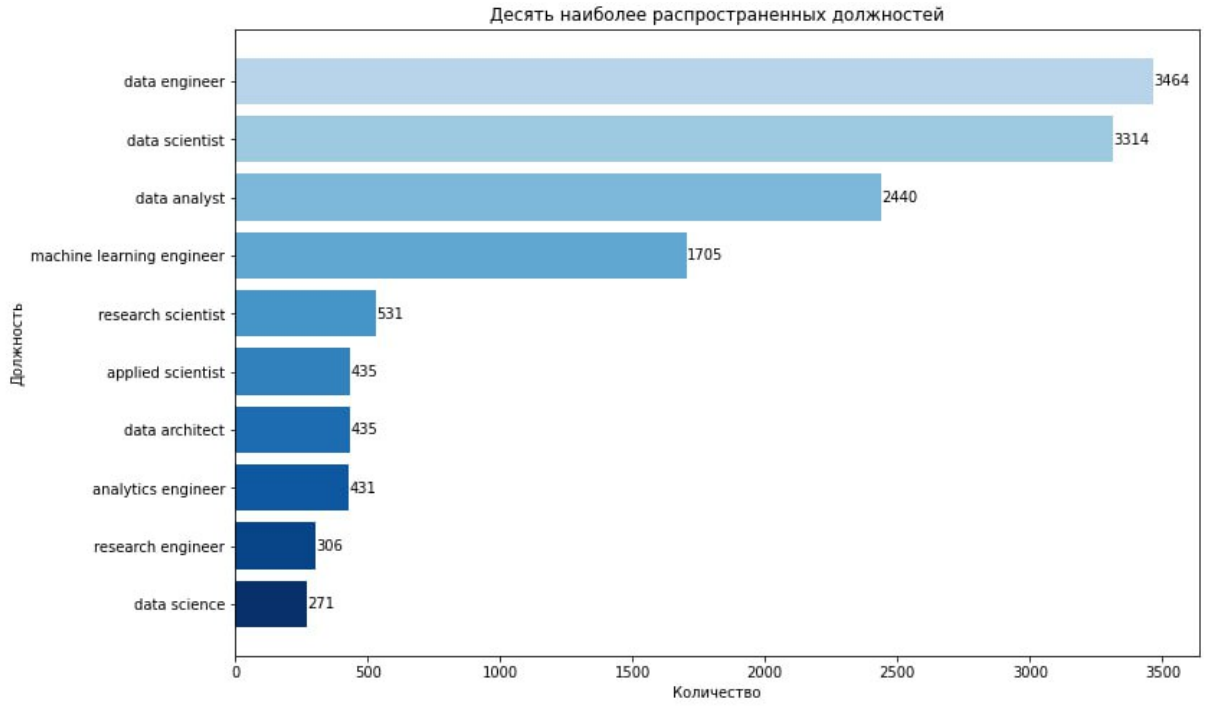

Рисунок 2.6 — Подсчет количества каждой должности

Рассмотрим верхние значения, которые с большим отрывом уходят от остальных:

- Двумя лидирующими позициями оказались Data Engineer и Data Scientist, что указывает на высокий спрос на эти должности, поскольку они занимают ключевые роли в обработке, анализе и интерпретации данных;
- Третьей стала Data Analyst из-за контекста работы – пересечение бизнесс-процессов и данных для анализа;
- С развитием технологий машинного обучения, четвертую позицию занимает Machine Learning Engineer;
- Пятое и шестое место по распределению имеют научные роли: Research S

Данные показывают следующие тенденции – рост спроса на внедрение исоздание моделей и алгоритмов машинного обучения, высокое значение научных и исследовательских ролей, большое количество вакансий на позициях, связанных с профессиями работающими с данными и разнообразие в области.

n

t

i

Анализ взаимосвязи между возможностью удаленной работы и уровнем заработной платы помогает компаниям и специалистам лучше понять текущие рыночные тенденции. Это позволяет выявить, как предпочтения и возможности удаленной работы влияют на вознаграждение сотрудников, учет предпочтений сотрудников, позволяет оценить конкурентоспособные предложения и оптимизировать стратегии найма.

Для анализа взаимосвязи между двумя переменными используется корреляция Пирсона, определяющая степень линейной зависимости между ними. Переменными для анализа будем считать salary_in_usd и remote_ratio. Корреляция Пирсона, значение от -1 до 1, рассчитывается по формуле (1):

$$\rho_{XY} = \frac{Cov(X,Y)}{\sigma_X \cdot \sigma_X}$$

где) – ковариация, σ_X – стандартное отклонение.

Ковариация рассчитывается по формуле (2):

$$Cov(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})$$

где n – количество и

Стандартное отклонение рассчитывается по формуле (3):

$$\sigma_X = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})} \quad (3)$$

На Рисунке 2.7 представлен скрипт, рассчитывающий среднюю зарплату и стандартное отклонение для каждого вида удаленной работы (0 – отсутствие удаленной работы, 50 – гибридный график, 100 – полностью удаленная).

In [39]:
```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import lit, round

spark = SparkSession.builder \
    .appName("RemoteWorkSalaryAnalysis") \
    .enableHiveSupport() \
    .getOrCreate() \


df = spark.read.table("new_salaries")


average_salary = df.groupBy("remote_ratio").agg(
    F.avg("salary_in_usd").alias("average_salary"),
    F.stddev("salary_in_usd").alias("standart_deviation")
)

average_salary = average_salary.withColumn("average_salary", round("average_salary", 2))
average_salary = average_salary.withColumn("standart_deviation", round("standart_deviation", 2))

correlation = (df.stat.corr("remote_ratio", "salary_in_usd"))

average_salary.show()
print(f"Correlation remote_ratio and salary_in_usd: {correlation}")

average_salary.write.mode("overwrite").saveAsTable("remote_to_salary_correlation_4")
```

```
+------------+--------------+------------------+
|remote_ratio|average_salary|standart_deviation|
+------------+--------------+------------------+
|         100|     145486.69|          62901.55|
|          50|      82984.47|          61779.02|
|           0|     153132.61|          70255.45|
+------------+--------------+------------------+

Correlation remote_ratio and salary_in_usd: -0.057288865300753646
```

Рисунок 2.7 — Расчет корреляции заработной платы и удаленной работы

Корреляция между удаленной работой и уровнем заработной платы составляет -0.057. Это указывает на очень слабую отрицательную связь между этими двумя переменными.

Самую высокую среднюю зарплату имеют специалисты, работающие полностью в офисе (0% удаленной работы) — 153,132. Специалисты, работающие полностью удаленно (100% удаленной работы), имеют среднюю зарплату 145,486, что немного ниже по сравнению с офисными сотрудниками. Наиболее низкую среднюю зарплату получают специалисты, работающие частично удаленно (50% удаленной работы) — 82,984.

Стандартное отклонение высоко во всех категориях, что указывает на значительное разброс зарплат в зависимости от конкретных ролей, компаний и регионов (Рисунок 2.8). Очень слабая отрицательная корреляция (-0.057) между удаленной работой и уровнем заработной платы указывает на то, что возможность удаленной работы не является значимым фактором, определяющим зарплату специалистов в области. Другие факторы, такие как опыт, роль, регион, и компания, играют большую роль.

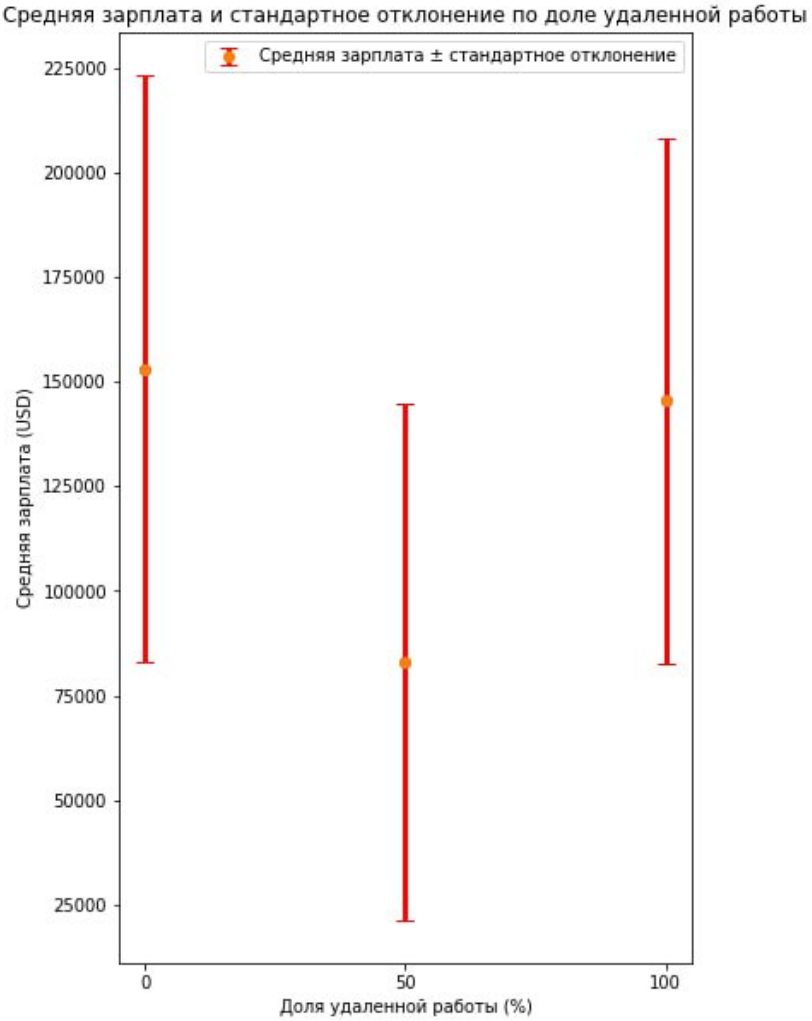

Рисунок 2.8 — Средняя зарплата и стандартное отклонение по доле удаленной работы

Возможность удаленной работы имеет некоторую связь с уровнем заработной платы среди специалистов по Data Engineering, но эта связь довольно слабая. Полностью офисные роли предлагают самые высокие средние зарплаты, за ними следуют полностью удаленные роли, и наименьшие зарплаты получают специалисты с частичной удаленной работой.

Какие уровни опыта являются наиболее востребованными среди специалистов Data Engineering? Для ответа на этот вопрос написан скрипт, представленный на Рисунке 2.8.

```
In [41]: from pyspark.sql import SparkSession
         from pyspark.sql.functions import col, when


         spark = SparkSession.builder \
             .appName("ExperienceLevelDistribution") \
             .enableHiveSupport() \
             .getOrCreate()

         df = spark.read.table("new_salaries")

         df = df.withColumn("experience_level",
                            when(col("experience_level") == "EX", "Executive") \
                            .when(col("experience_level") == "SE", "Senior") \
                            .when(col("experience_level") == "MI", "Mid") \
                            .when(col("experience_level") == "EN", "Entry") \
                            .otherwise(col("experience_level")))

         experience_level_distribution = df.groupBy("experience_level").count()
         experience_level_distribution.show()
         |
         experience_level_distribution.write.mode("overwrite").saveAsTable("experience_level_distribution_5")
```

```
+----------------+-----+
|experience_level|count|
+----------------+-----+
|          Senior|10670|
|       Executive|  501|
|             Mid| 4038|
|           Entry| 1325|
+----------------+-----+
```

Рисунок 2.8 — Распределение специалистов на рынке по уровням опыта

Визуализациия полученных данных представлена на Рисунке 2.9:

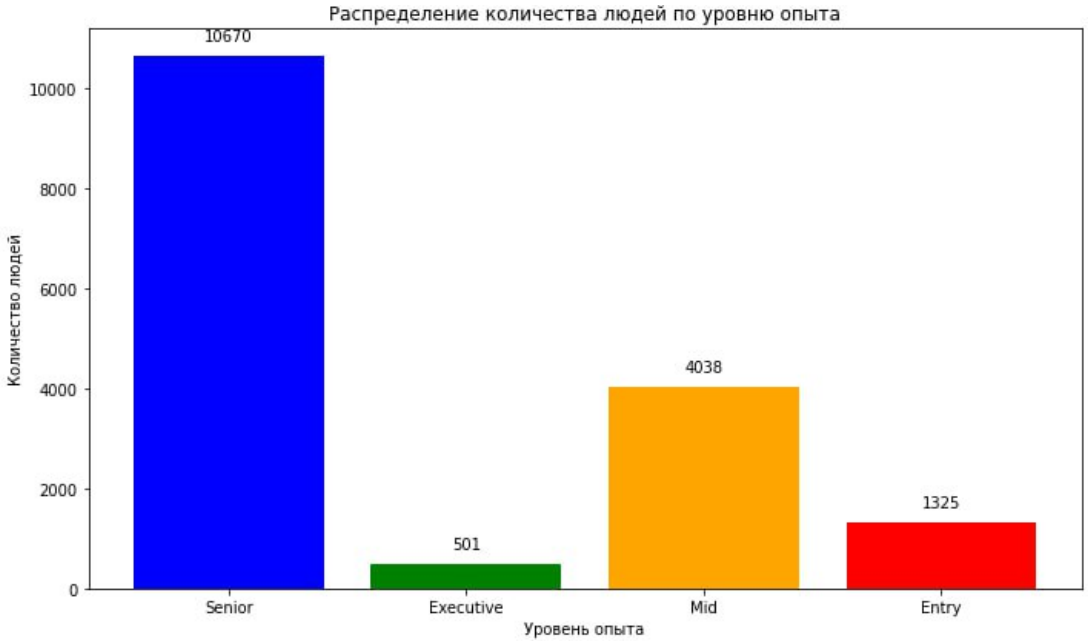

Рисунок 2.8 — Столбчатая диаграмма распределения специалистов по уровням опыта

Вакансии на позицию Senior преобладают значительно больше остальных. Из полученных значений, делаем выводы:

ольшинство вакансий предназначено для специалистов с уровнем опыта Senior или Middle. Это говорит о высоком спросе на опытных и квалифицированных профессионалов в области Data Engineering.

прос на начальный уровень (Entry) и руководителей (Executive)

значительно ниже. Это может быть связано с тем, что компании предпочитают нанимать специалистов с уже имеющимся опытом или продвигать руководителей из числа существующих сотрудников.

олодым специалистам следует сосредоточиться на получении опыта и знаний, чтобы быстрее перейти на более востребованные уровни. Опытным специалистам можно рекомендовать рассматривать возможности карьерного роста и перехода на руководящие позиции.

Анализ распределения вакансий по уровням опыта среди специалистов Data Engineering показывает значительный спрос на опытных специалистов уровня Senior и Mid, что подчеркивает необходимость в глубоком профессиональном опыте и знаниях.

ЗАКЛЮЧЕНИЕ

Статистический анализ данных является неотъемлемой частью современного мира. С его помощью специалисты изучают различные социально-экономические, политические и иные явления. Одним из наиболее важных вопросов для анализа является рынок труда.

Специалисты по анализу данных ежедневно изучают и прогнозируют такие явления как уровень опыта специалистов в облостях, количество вакансий, распределение рабочих мест и множество других.

Одним из наиболее острых вопросов в сфере анализа рынка труда Data Engineer является не только востребованность определенных компетенций, но и их соответствие потребностям рынка. Специалисты по анализу данных систематически изучают требования работодателей к специалистам в данной области, предсказывают тенденции изменения спроса на определенные навыки и помогают формировать образовательные программы и карьерные планы, соответствующие потребностям рынка труда.

Таким образом можно убедиться в том, что тема данной курсовой работы является актуальной не только для нашей страны, но и для всего мирового сообщества.

Цель данной курсовой работы — проанализировать имеющуюся выборку и определить тенденции и связи в области рынка труда Data Engineer — достигнута.

В ходе работы были выполнены следующие задачи:

- составлен конвейер передачи данных;
- определены ключевые вопросы для проведения анализа;
- очищены и обработаны данные;
- анализированы и интерпретированы данные;
- визуализированы полученные результаты.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. СОЗДАНИЕ КОНВЕЙЕРА ДЛЯ ПЕРЕДАЧИ ДАННЫХ

MariaDB Учебное пособие: изучение синтаксиса и команд с примерами

урс лекций Samsung Innovation Campus [Электронный ресурс]. —

2. АНАЛИЗ ПОЛУЧЕННЫХ ДАННЫХ

р. PySpark. Анализ больших данных, когда Pandas не достаточно [Электронный ресурс]. — Режим доступа:

. Python to Hive Connection: A Comprehensive Guide to Analyzing [Электронный ресурс]. — Режим доступа:

разбираемся в технологии в теории и на практике [Электронный ресурс]. — Режим доступа: https://selectel.ru/blog/apache-kafka-2/ — Дата доступа: 07.06.2024

ресурс]. — Режим доступа: https://anyi-guo.medium.com/correlation-

a

t

a

p

бр. Рынок труда и перспективы карьеры в Data Science в 2024 году [Электронный ресурс] —

o

t

l арк Лутц. Изучаем Python. — Том 1, 5-у изд.: Диалектика, 2019. — 721

i c

b .

M

a —

t

p э

l к

o з

t .

l

i —

b

d

o

c

u

m

e

n

t

a

t

i

o

ПРИЛОЖЕНИЯ

Приложение А — Код для аналитического расчета типов занятости, средней заработной плате по годам, распределении вакансий, корреляции между удаленной работой и заработной платой, распределении уровней опыта.

Приложение Б — Код для создания графиков анализированных данных.

Приложение А

Код для аналитического расчета типов занятости, средней заработной плате по годам, распределении вакансий, корреляции между удаленной работой и заработной платой, распределении уровней опыта.

Листинг А.1 — код для анализа представленного набора данных

```
# Самые распространенные типы занятости

from pyspark.sql import SparkSession
from pyspark.sql import functions as F
spark = SparkSession.builder \
        .appName("ImportTable") \
        .getOrCreate()
df = spark.read.table("new_salaries")
employment_type_new = df.withColumn (
    'employment_type',
    F.when(F.col('employment_type') == 'FT', 'Full time')
     .when(F.col('employment_type') == 'PT', 'Part time')
     .when(F.col('employment_type') == 'CT', 'Contract')
     .when(F.col('employment_type') == 'FL', 'Freelance')
     .otherwise(F.col('employment_type'))
)
employment_type_count =
employment_type_new.groupBy("employment_type").count()
sorted_emp_type_count = employment_type_count.orderBy(F.desc('count'))
sorted_emp_type_count.show()
sorted_emp_type_count.write.mode("overwrite").saveAsTable("emp_type_count_1")

# Средняя зп по годам

from pyspark.sql import functions as F
average_salary_by_year = df.groupBy("work_year") \
    .agg(F.format_number(F.avg("salary_in_usd"), 2).alias("average_salary"))
sorted_avg_salary = average_salary_by_year.orderBy("work_year")
sorted_avg_salary.show()
orted_avg_salary.write.mode("overwrite").saveAsTable("avg_salary_year_2")
```

```
# Наиболее часто встречаемые должности

from pyspark.sql import functions as F
from pyspark.sql.functions import col, lower, trim
df = spark.read.table("new_salaries")
df_cleaned = df.withColumn("job_title_cleaned", lower(trim(col("job_title"))))
job_title_count = df_cleaned.groupBy("job_title_cleaned").count()
job_title_count_sorted = job_title_count.orderBy(col("count").desc())
job_title_count_sorted.show()
job_title_count_sorted.write.mode("overwrite").saveAsTable("job_title_count_3")

# Корреляция удаленной работы с заработной платой

from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import lit, round
spark = SparkSession.builder \
    .appName("RemoteWorkSalaryAnalysis") \
    .enableHiveSupport() \
    .getOrCreate() \
df = spark.read.table("new_salaries")
average_salary = df.groupBy("remote_ratio").agg(
    F.avg("salary_in_usd").alias("average_salary"),
    F.stddev("salary_in_usd").alias("standart_deviation")
)
average_salary = average_salary.withColumn("average_salary",
round("average_salary", 2))
average_salary = average_salary.withColumn("standart_deviation",
round("standart_deviation", 2))
correlation = (df.stat.corr("remote_ratio", "salary_in_usd"))
average_salary.show()
print(f"Correlation remote_ratio and salary_in_usd: {correlation}")
average_salary.write.mode("overwrite").saveAsTable("remote_to_salary_correlation
_4")

# Распределение уровня опыта

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when
spark = SparkSession.builder \
    .appName("ExperienceLevelDistribution") \
    .enableHiveSupport() \
    .getOrCreate()

df = spark.read.table("new_salaries")
df = df.withColumn("experience_level",
```

```
            when(col("experience_level") == "EX", "Executive") \
            .when(col("experience_level") == "SE", "Senior") \
            .when(col("experience_level") == "MI", "Mid") \
            .when(col("experience_level") == "EN", "Entry") \
            .otherwise(col("experience_level")))
experience_level_distribution = df.groupBy("experience_level").count()
experience_level_distribution.show()
experience_level_distribution.write.mode("overwrite").saveAsTable("experience_lev
el_distribution_5")
```

Приложение Б

Код для создания графиков анализированных данных.

Листинг Б.2 — проверка выдвинутых гипотез.

```
from pyhive import hive
import pandas as pd
import matplotlib.pyplot as plt
conn = hive.Connection(host="localhost", port=10000, username="student")
query = "SELECT * FROM emp_type_count_1"
df = pd.read_sql(query, conn)
conn.close()
df_sorted = df.sort_values('emp_type_count_1.count', ascending=True)
small_categories = df_sorted.head(3)
small_categories_sum = small_categories['emp_type_count_1.count'].sum()
df_updated = df_sorted.iloc[3:]
df_updated = df_updated.append({'emp_type_count_1.employment_type': 'Другие',
'emp_type_count_1.count': small_categories_sum}, ignore_index=True)
colors = ['gold', 'lightgreen', 'lightcoral', 'lightskyblue', 'violet', 'orange', 'grey']
explode = [0.1 if i == df_updated['emp_type_count_1.count'].idxmax() else 0 for i in
range(len(df_updated))]
plt.figure(figsize=(8, 8))
plt.pie(df_updated['emp_type_count_1.count'], explode=explode,
colors=colors[:len(df_updated)], startangle=140,
        autopct='%1.1f%%', shadow=True)
plt.legend(df_updated['emp_type_count_1.employment_type'], title="Тип
занятости", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))
plt.title('Распределение типов занятости среди специалистов Data Engineering')
plt.axis('equal')
```

```
from pyhive import hive
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
conn = hive.Connection(host="localhost", port=10000, username="student")
query = "SELECT * FROM avg_salary_year_2"
df = pd.read_sql(query, conn)
conn.close()
df['avg_salary_year_2.average_salary'] =
df['avg_salary_year_2.average_salary'].replace(',', '', regex=True).astype(float)
max_salary = df['avg_salary_year_2.average_salary'].max()
```

```
colors = plt.cm.Blues(df['avg_salary_year_2.average_salary'] / max_salary)
plt.figure(figsize=(10, 6))
plt.bar(df['avg_salary_year_2.work_year'], df['avg_salary_year_2.average_salary'],
color=colors)
plt.title('Средняя зарплата по годам')
plt.xlabel('Год')
plt.ylabel('Средняя зарплата')
for i, salary in enumerate(df['avg_salary_year_2.average_salary']):
    plt.text(df['avg_salary_year_2.work_year'][i], salary, f'{salary:,.2f}', ha='center',
va='bottom')
plt.show()
plt.savefig('average_salary_by_year.png')


from pyhive import hive
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
conn = hive.Connection(host="localhost", port=10000, username="student")
query = "SELECT * FROM job_title_count_3"
df = pd.read_sql(query, conn)
conn.close()
df_sorted = df.sort_values('job_title_count_3.count', ascending=False)
top_n = 10
df_top = df_sorted.head(top_n)
colors = plt.cm.Blues(np.linspace(0.3, 1, top_n))
plt.figure(figsize=(12, 8))
bars = plt.barh(df_top['job_title_count_3.job_title_cleaned'],
                df_top['job_title_count_3.count'], color=colors)
plt.title('Десять наиболее распространенных должностей'.format(top_n))
plt.xlabel('Количество')
plt.ylabel('Должность')
for bar, value in zip(bars, df_top['job_title_count_3.count']):
    plt.text(value, bar.get_y() + bar.get_height()/2, str(value), va='center')
plt.gca().invert_yaxis()
plt.show()
plt.savefig('job_title_top10.png')


from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import lit, round
spark = SparkSession.builder \
    .appName("RemoteWorkSalaryAnalysis") \
    .enableHiveSupport() \
```

```
    .getOrCreate() \
df = spark.read.table("new_salaries")
average_salary = df.groupBy("remote_ratio").agg(
    F.avg("salary_in_usd").alias("average_salary"),
    F.stddev("salary_in_usd").alias("standart_deviation")
)
average_salary = average_salary.withColumn("average_salary",
round("average_salary", 2))
average_salary = average_salary.withColumn("standart_deviation",
round("standart_deviation", 2))
correlation = (df.stat.corr("remote_ratio", "salary_in_usd"))
average_salary.show()
print(f"Correlation remote_ratio and salary_in_usd: {correlation}")
average_salary.write.mode("overwrite").saveAsTable("remote_to_salary_correlation
_4")
plt.figure(figsize=(6, 10))
plt.errorbar(pandas_filtered_df['remote_ratio'], pandas_filtered_df['average_salary'],
            yerr=pandas_filtered_df['standart_deviation'], fmt='o', ecolor='red',
elinewidth=3, capsize=5)
plt.errorbar(pandas_filtered_df['remote_ratio'], pandas_filtered_df['average_salary'],
            yerr=pandas_filtered_df['standart_deviation'],
            fmt='o', ecolor='red', elinewidth=3, capsize=5, label='Средняя зарплата ±
стандартное отклонение')
plt.legend(loc='upper right')
plt.title('Средняя зарплата и стандартное отклонение по доле удаленной работы')
plt.xlabel('Доля удаленной работы (%)')
plt.ylabel('Средняя зарплата (USD)')
plt.xticks([0, 50, 100])
plt.grid(False)
plt.show()


from pyhive import hive
import pandas as pd
import matplotlib.pyplot as plt
conn = hive.Connection(host="localhost", port=10000, username="student")
query = "SELECT experience_level, count FROM experience_level_distribution_5"
df = pd.read_sql(query, conn)
conn.close()
print(df.columns)
df['experience_level'] = df['experience_level'].astype(str)
df['count'] = df['count'].astype(int)
plt.figure(figsize=(10, 6))
plt.bar(df['experience_level'], df['count'], color=['blue', 'green', 'orange', 'red'])
```

```
plt.title('Распределение количества людей по уровню опыта')
plt.xlabel('Уровень опыта')
plt.ylabel('Количество людей')
for i, (exp_level, count) in enumerate(zip(df['experience_level'], df['count'])):
    plt.text(i, count + 300, str(count), ha='center')
plt.tight_layout()
plt.show()
plt.savefig('experience_level_distribution.png')
```