



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

«Системы управления данными»

**Тема курсовой работы:** «Проектирование системы и анализ зарплат  
дата-инженеров»

Студент группы ИМБО-01-21

Тихомиров Никита Сергеевич

(подпись)

Руководитель  
курсовой работы

ст. преп. Буданцев А.В.

(подпись)

Работа представлена к защите

«\_\_»\_\_\_\_\_2023 г.

Допущен к защите

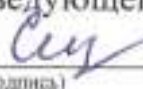
«\_\_»\_\_\_\_\_2023 г.

Москва 2023 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  
и.о. заведующего кафедрой ПМ  
 Смоленцева Т.Е.  
(подпись) « » 2023 г.

**ЗАДАНИЕ**  
**на выполнение курсовой работы**  
**по дисциплине «Системы управления данными»**

Студент Тихомиров Никита Сергеевич

Группа ИМБО-01-21

**Тема «Проектирование системы и анализ зарплат дата-инженеров»**

**Исходные данные:** выбранный датасет по зарплатам дата-инженеров

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

Возможно ли построить сложную модель, с высокой вероятностью предсказывающую зарплату дата-инженеров?

От каких характеристик зависит зарплата дата-инженеров?

Как построить конвейер обработки больших данных и как он будет работать?

**Срок представления к защите курсовой работы:**

до « » 2024 г.

**Задание на курсовую работу выдал**

  
Подпись руководителя

Буданцев А.В.  
(ФИО руководителя)

« » 2024 г.

**Задание на курсовую работу получил**

  
Подпись обучающегося

Тихомиров Н.С.  
(ФИО обучающегося)

« » 2024 г.

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ .....	5
1.1 Информация о наборе данных .....	5
1.2 Архитектура конвейера для получения и предобработки данных .....	5
2 ПРАКТИЧЕСКАЯ ЧАСТЬ.....	16
2.1 Анализ и визуализация данных .....	16
2.2 Регрессионный анализ .....	21
ЗАКЛЮЧЕНИЕ .....	27
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	28
ПРИЛОЖЕНИЯ.....	30

# ВВЕДЕНИЕ

В современном мире спрос на квалифицированных специалистов в области обработки и анализа данных стремительно растет. Дата-инженеры играют ключевую роль в создании инфраструктуры для хранения, обработки и анализа больших данных, что делает их незаменимыми в различных отраслях, включая финансы, здравоохранение, розничную торговлю и многие другие.

Одним из ключевых аспектов, определяющих привлекательность профессии дата-инженера, является уровень заработной платы. Анализ факторов, влияющих на зарплаты дата-инженеров, может помочь работодателям лучше понимать рынок труда, а потенциальным сотрудникам — оценивать свои карьерные перспективы.

Цель курсовой работы — исследовать и проанализировать уровень зарплат дата-инженеров, выявить основные тенденции и факторы, влияющие на доходы в этой сфере. В рамках работы предполагается проведение анализа больших данных о заработных платах, а также разработка моделей, предсказывающих уровень дохода на основе различных характеристик.

В ходе выполнения работы необходимо выполнить следующие задачи:

- разработать пайплайн для предобработки и маршрутизации больших данных;
- провести анализ факторов, влияющих на уровень заработной платы дата-инженеров;
- построить и проанализировать распределение зарплат в зависимости от грейда;
- провести визуализацию данных для лучшего понимания закономерностей и тенденций;
- построить и оценить регрессионную модель, используя данные о характеристиках дата-инженеров и их зарплатах.

# 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Информация о наборе данных

Данные были взяты с платформы по исследованию данных Kaggle и представляют собой данные о зарплатах дата-инженеров. [1.1]

Файл представлен в формате csv и содержит следующие атрибуты:

- work\_year — год;
- experience\_level — грейд сотрудника;
- employment\_type — тип занятости;
- job\_title — название должности;
- salary — зарплата сотрудника в местной валюте;
- salary\_currency — валюта, в которой выплачивается зарплата;
- salary\_in\_usd — зарплата в долларах США;
- employee\_residence — страна проживания сотрудника;
- remote\_ratio — коэффициент удалённости от работы;
- company\_location — страна расположения компании;
- company\_size — размер компании.

## 1.2 Архитектура конвейера для получения и предобработки данных

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

Для хранения данных используются инструменты MariaDB и Apache Hive. MariaDB — это система управления базами данных, которая является

ответвлением или улучшенной копией MySQL. [1.2]

Особенности MariaDB, которые отличают ее от MySQL:

1. Более высокая производительность, новые возможности по управлению базами данных и намного меньшее количество ошибок в коде.
2. Использует более производительный оптимизатор запросов и более безопасные индексы для алгоритмов хранения информации.
3. Система хранения информации InnoDB была заменена на XtraDB.
4. Поддерживает большое количество функциональных команд, которые не поддерживаются в 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. [1.3]

Так как хранение и анализ данных производится в распределенной файловой системе HDFS, данные инструменты будут наиболее подходящими для хранения информации.

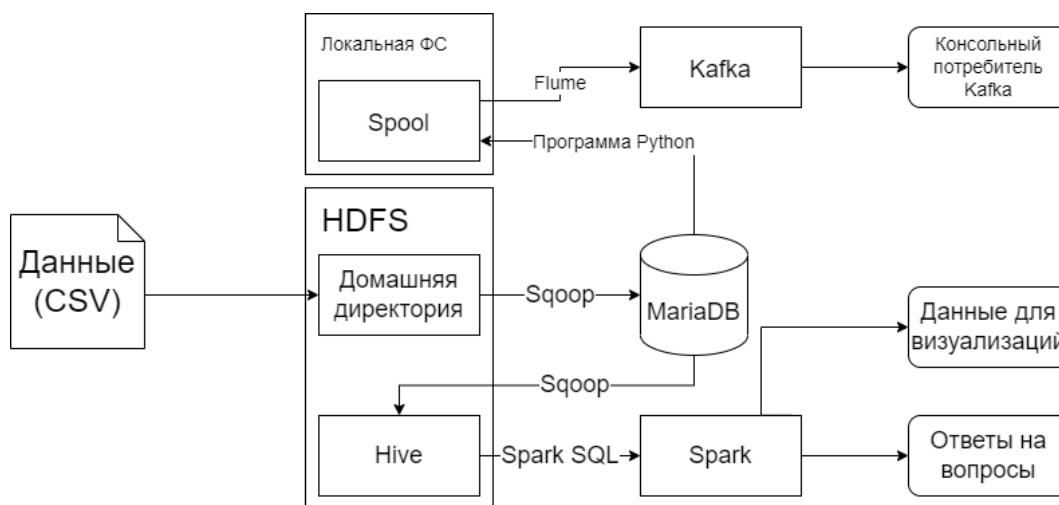
Для передачи данных из HDFS в MariaDB используется инструмент Apache Sqoop — приложение с интерфейсом командной строки для передачи данных между реляционными базами данных и Hadoop. [1.4]

Также для моделирования потоковой передачи данных используется Apache Kafka — гибрид распределенной базы данных и брокера сообщений с возможностью горизонтального масштабирования. [1.5]

Kafka собирает у приложений данные, хранит в своем распределенном хранилище, группируя по топикам, и отдает компонентам приложения по подписке. При этом сообщения хранятся на различных узлах-брокерах, что обеспечивает высокую доступность и отказоустойчивость. Данные в Kafka поступают из MariaDB через Flume — инструмент, позволяющий управлять

потоками данных и передавать их на некоторый пункт назначения. [1.6]

Перед проведением анализа и реализацией конвейера, была разработана его схема (Рисунок 1.1). [1.7]



**Рисунок 1.1 — Схема конвейера данных**

Для построения конвейера первым делом требуется запустить службы Hadoop и Yarn (Рисунок 1.2).

```
[hadoop@localhost hadoop]$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [localhost.localdomain]
Starting resourcemanager
Starting nodemanagers
[hadoop@localhost hadoop]$ jps
3314 JobHistoryServer
7123 Jps
6100 DataNode
5974 NameNode
6328 SecondaryNameNode
6604 ResourceManager
6734 NodeManager
```

**Рисунок 1.2 — Запуск служб Hadoop и Yarn**

Далее необходимо сохранить наши данные в HDFS. После запуска служб для промежуточного хранения данных и их предобработки, создадим директорию `/user/student/data` в HDFS, в которую мы перенесем файл с данными из локальной файловой системы (Рисунок 1.3).

```
[hadoop@localhost hadoop]$ hdfs dfs -mkdir /user/student/data
[hadoop@localhost hadoop]$ hdfs dfs -put /media/sf_sud/salaries_in_2024.csv /user/student/data
[hadoop@localhost hadoop]$ hdfs dfs -ls /user/student/data
Found 1 items
-rw-r--r-- 1 hadoop student 948117 2024-05-25 02:52 /user/student/data/salaries_in_2024.csv
```

**Рисунок 1.3 — Создание директории в HDFS и перенос файла данных**

Теперь перейдем в реляционную систему управления базами данных MariaDB и создадим новую базу данных «stat» (Рисунок 1.4).

```
[student@localhost ~]$ mysql -u student -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 42
Server version: 5.5.68-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database stat;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> use stat;
Database changed
```

**Рисунок 1.4 — Переход в MariaDB и создание базы данных «stat»**

Перед экспортом данных в MariaDB из HDFS требуется создать таблицу с соответствующим типом данных столбцов для правильного переноса и названиями, соответствующими исходной таблице, для удобства восприятия (Рисунок 1.5).

```
MariaDB [stat]> create table salary (work_year INT, experience_level VARCHAR(255), employment_type VARCHAR(255), job_title VARCHAR(255), salary DOUBLE, salary_currency VARCHAR(255), salary_in_usd DOUBLE, employee_residence VARCHAR(255), remote_ratio INT, company_location VARCHAR(255), company_size VARCHAR(255));
Query OK, 0 rows affected (0.14 sec)

MariaDB [stat]> desc salary;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| work_year | int(11) | YES | | NULL | |
| experience_level | varchar(255) | YES | | NULL | |
| employment_type | varchar(255) | YES | | NULL | |
| job_title | varchar(255) | YES | | NULL | |
| salary | double | YES | | NULL | |
| salary_currency | varchar(255) | YES | | NULL | |
| salary_in_usd | double | YES | | NULL | |
| employee_residence | varchar(255) | YES | | NULL | |
| remote_ratio | int(11) | YES | | NULL | |
| company_location | varchar(255) | YES | | NULL | |
| company_size | varchar(255) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
11 rows in set (0.15 sec)
```

**Рисунок 1.5 — Создание таблицы «salary» в MariaDB**

После создания таблицы в нее нужно экспортировать данные из HDFS,



выполним это при помощи Sqoop (Рисунок 1.6). Sqoop является инструментом для перемещения данных из СУБД в HDFS и обратно.

```
[student@localhost ~]$ sqoop export \
> --connect jdbc:mysql://localhost/stat \
> --table salary \
> --username student \
> --password student \
> --export-dir /user/student/data \
> --lines-terminated-by '\n' \
> --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.
2024-06-06 18:12:30,598 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
2024-06-06 18:12:30,972 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
2024-06-06 18:12:31,596 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
2024-06-06 18:12:31,606 INFO tool.CodeGenTool: Beginning code generation
2024-06-06 18:12:32,902 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'salary' AS t LIMIT 1
2024-06-06 18:12:33,108 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'salary' AS t LIMIT 1
2024-06-06 18:12:33,116 INFO orm.CompilationManager: HADOOP MAPRED HOME is /home/hadoop/hadoop
Note: /tmp/sqoop-student/compile/ecbf8e611549bdddeb5ecbe36d79d0f5/salary.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2024-06-06 18:12:54,428 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-student/compile/ecbf8e611549bdddeb5ecbe36d79d0f5/salary.jar
2024-06-06 18:12:54,513 INFO mapreduce.ExportJobBase: Beginning export of salary
2024-06-06 18:12:54,513 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2024-06-06 18:12:55,438 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
2024-06-06 18:12:59,696 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.speculative
2024-06-06 18:12:59,702 INFO Configuration.deprecation: mapred.map.tasks.speculative.execution is deprecated. Instead, use mapreduce.map.speculative
2024-06-06 18:12:59,702 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
2024-06-06 18:13:00,256 INFO client.DefaultHARMPFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-06-06 18:13:03,289 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/student/.staging/job_1717662363936_0001
2024-06-06 18:13:18,268 INFO input.FileInputFormat: Total input files to process : 1
2024-06-06 18:13:18,299 INFO input.FileInputFormat: Total input files to process : 1
2024-06-06 18:13:18,832 INFO mapreduce.JobSubmitter: number of splits:4
2024-06-06 18:13:19,091 INFO Configuration.deprecation: mapred.map.tasks.speculative.execution is deprecated. Instead, use mapreduce.map.speculative
2024-06-06 18:13:19,722 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1717662363936_0001
2024-06-06 18:13:19,723 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-06-06 18:13:24,350 INFO conf.Configuration: resource-types.xml not found
2024-06-06 18:13:24,351 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-06-06 18:13:27,672 INFO impl.YarnClientImpl: Submitted application application_1717662363936_0001
2024-06-06 18:13:27,805 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1717662363936_0001/
2024-06-06 18:13:27,806 INFO mapreduce.Job: Running job: job_1717662363936_0001
2024-06-06 18:14:06,781 INFO mapreduce.Job: Job job_1717662363936_0001 running in uber mode : false
2024-06-06 18:14:06,836 INFO mapreduce.Job: map 0% reduce 0%
```

Рисунок 1.6 — Перенос данных из HDFS в MariaDB

После успешного переноса данных выведем часть содержимого заполненной таблицы в РСУБД MariaDB при помощи команды SELECT (Рисунок 1.3).

```
MariaDB [stat]> select * from salary limit 10;
```

work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
2024	SE	FT	AI Engineer	202730	USD	202730	US	0	US	M
2024	SE	FT	AI Engineer	92118	USD	92118	US	0	US	M
2024	SE	FT	Data Engineer	130500	USD	130500	US	0	US	M
2024	SE	FT	Data Engineer	96000	USD	96000	US	0	US	M
2024	SE	FT	Machine Learning Engineer	190000	USD	190000	US	0	US	M
2024	SE	FT	Machine Learning Engineer	160000	USD	160000	US	0	US	M
2024	MI	FT	ML Engineer	400000	USD	400000	US	0	US	M
2024	MI	FT	ML Engineer	65000	USD	65000	US	0	US	M
2024	EN	FT	Data Analyst	101520	USD	101520	US	0	US	M
2024	EN	FT	Data Analyst	45864	USD	45864	US	0	US	M

10 rows in set (0.04 sec)

Рисунок 1.3 — Вывод первых 10 строк таблицы в MariaDB

Данные были экспортированы успешно. Для эффективного и корректного анализа нужно удостовериться, что в данных отсутствуют значения NULL, и если такие есть, то удалить их (Рисунок 1.4).

```

MariaDB [stat]> delete from salary where work_year = 0;
Query OK, 0 rows affected (0.05 sec)

MariaDB [stat]> delete from salary where employment_type is null;
Query OK, 0 rows affected (0.02 sec)

MariaDB [stat]> delete from salary where employment_type is null;
Query OK, 0 rows affected (0.00 sec)

MariaDB [stat]> delete from salary where job_title is null;
Query OK, 0 rows affected (0.00 sec)

MariaDB [stat]> delete from salary where salary = 0;
Query OK, 0 rows affected (0.01 sec)

MariaDB [stat]> delete from salary where salary_currency is null;
Query OK, 0 rows affected (0.01 sec)

MariaDB [stat]> delete from salary where salary_in_usd = 0;
Query OK, 0 rows affected (0.01 sec)

MariaDB [stat]> delete from salary where employee_residence is null;
Query OK, 0 rows affected (0.00 sec)

MariaDB [stat]> delete from salary where remote_ratio is null;
Query OK, 0 rows affected (0.01 sec)

MariaDB [stat]> delete from salary where company_location is null;
Query OK, 0 rows affected (0.02 sec)

MariaDB [stat]> delete from salary where company_size is null;
Query OK, 0 rows affected (0.01 sec)

```

**Рисунок 1.4 — Проверка наличия значений NULL в данных**

Следующим этапом построения конвейера является создание директории /Spool в локальной файловой системе (Рисунок 1.5).

```

[student@localhost ~]$ mkdir Spool
[student@localhost ~]$ ll
total 55880
-rw-r--r--. 1 student student 97334 Sep 29 2023 625adc81-1a01-446e-a35f-9aafa8e401d4.parquet
-rw-r--r--. 1 student student 2532562 Sep 29 2023 79811c1d-5e5c-4b40-b887-75b7185b5a81.parquet
-rw-rw-r--. 1 student student 4549 Jun 6 20:59 aka_scoop.ipynb
-rw-r--r--. 1 root root 8296049 Mar 8 2023 apache-maven-3.8.8-bin.tar.gz
-rw-rw-r--. 1 student student 20766 Sep 29 2023 authors_export.java
-rw-rw-r--. 1 student student 20598 Sep 29 2023 authors.java
-rw-rw-r--. 1 student student 42986646 Apr 18 21:29 backup.sql
-rw-rw-r--. 1 student student 20782 Sep 29 2023 codegen_authors.java
-rw-rw-r--. 1 student student 13197 Sep 29 2023 codegen_posts.java
drwxrwxr-x. 6 student student 4096 Sep 26 2021 Data
drwxr-xr-x. 2 student student 6 Jul 25 2021 Desktop
drwxr-xr-x. 2 student student 24 Jun 6 19:01 Documents
drwxr-xr-x. 2 student student 34 Apr 13 04:58 Downloads
drwxr-xr-x. 2 student student 102 Apr 20 04:08 get_authors
drwxr-xr-x. 2 student student 102 Apr 20 04:23 get_posts
drwxr-xr-x. 10 student student 193 Jan 21 1970 idea-IU-241.14494.240
drwxrwxr-x. 3 student student 19 Apr 13 05:53 IdeaProjects
drwxrwxr-x. 6 student student 54 Sep 26 2021 Labs
drwxr-xr-x. 2 student student 6 Jul 25 2021 Music
-rw-r--r--. 1 student student 842715 Sep 29 2023 part-m-00000.snappy
drwxr-xr-x. 2 student student 6 Jul 25 2021 Pictures
-rw-rw-r--. 1 student student 20029 Sep 29 2023 posts.java
drwxr-xr-x. 2 student student 6 Jul 25 2021 Public
-rw-rw-r--. 1 student student 31272 May 25 03:45 salary.java
drwxrwxr-x. 2 student student 279 Sep 26 2021 Scripts
drwxrwxr-x. 2 student student 6 Jun 6 21:07 Spool

```

**Рисунок 1.5 — Создание буферной директории в локальной ФС**

После требуется запустить PySpark и написать скрипт для передачи данных, который будет считывать по 5 % всех данных из таблицы Estate в MariaDB каждые 10 секунд. Запустим PySpark для последующей обработки данных (Рисунок 1.10).

```
[student@localhost sf_shared_folder]$ pyspark
[I 14:29:04.711 NotebookApp] Serving notebooks from local directory: /media/sf_shared_folder
[I 14:29:04.711 NotebookApp] Jupyter Notebook 6.4.3 is running at:
[I 14:29:04.711 NotebookApp] http://localhost:3333/?token=bac269edf7e95ca9d70bf4f090f7f05196bfe3b43d943baa
[I 14:29:04.711 NotebookApp] or http://127.0.0.1:3333/?token=bac269edf7e95ca9d70bf4f090f7f05196bfe3b43d943baa
[I 14:29:04.711 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 14:29:04.745 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/student/.local/share/jupyter/runtime/nbserver-29605-open.html
Or copy and paste one of these URLs:
http://localhost:3333/?token=bac269edf7e95ca9d70bf4f090f7f05196bfe3b43d943baa
or http://127.0.0.1:3333/?token=bac269edf7e95ca9d70bf4f090f7f05196bfe3b43d943baa
[I 14:29:11.398 NotebookApp] Creating new notebook in
[I 14:29:13.309 NotebookApp] Kernel started: fd46482f-0761-455c-8913-824849141e19, name: python3
2024-05-25 14:29:16,006 WARN util.Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 10.0.2.15 instead (on interface enp0s3)
2024-05-25 14:29:16,009 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
2024-05-25 14:29:16,570 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

**Рисунок 1.10 — Запуск Pyspark**

Далее создадим файл для скрипта и напомним его (Приложение А).

Данный скрипт содержит функцию `parse(args)`, которая при помощи библиотеки `rumysql` подключается к базе данных `data` по локальному серверу. После этого выполняется SQL запрос в данных, экспортирует их и закрывает соединение. Далее в функции `process(args)` происходит извлечение по 5 % данных от экспортированных ранее и запись их в файл формата `.csv`, который в свою очередь сохраняется в директорию `Spool` в локальной файловой системе.

Функция `data_to_csv(args)` будет вызывать функции извлечения и обработки, а при помощи библиотеки `schedule` это будет происходить каждые 10 секунд.

Теперь создадим топик консольного потребителя `Kafka`, а также проверим успешность его создания (Рисунок 1.1).

```
[student@localhost ~]$ kafka-topics \
> --create \
> --bootstrap-server localhost:9092 \
> --replication-factor 1 \
> --partitions 1 \
> --topic salary
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you should configure the number of parallel GC threads appropriately using -XX:ParallelGCThreads=N
Created topic salary.
[student@localhost ~]$ kafka-topics \
> --list \
> --bootstrap-server localhost:9092
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you should configure the number of parallel GC threads appropriately using -XX:ParallelGCThreads=N
salary
```

**Рисунок 1.11 — Создание топика в Kafka**

После создадим конфигурационный файл для агента `Flume`, который будет содержать источник данных в виде директории `Spool`, два канала для передачи данных, а также два слива, один из них будет передавать

информацию в консоль, а другой — в консольного потребителя (Рисунок 1.2).

```
#Names
a1.sources = src-1
a1.channels = c1 c1
a1.sinks = L1 K2

#Source
a1.sources.src-1.type = spooldir
a1.sources.src-1.spoolDir = /home/student/Spool
a1.sources.src-1.channels = c1 c2

#C1
a1.channels.c1.type = memory
a1.channels.c1.capacity = 150000
a1.channels.c1.transactionCapacity = 2000

#C2
a1.channels.c2.type = memory
a1.channels.c2.capacity = 150000
a1.channels.c2.transactionCapacity = 2000

#Sink to Kafka
a1.sinks.K2.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.K2.kafka.bootstrap.servers = localhost:9092
a1.sinks.K2.kafka.topic = salary
a1.sinks.K2.flumeBatchSize = 20
a1.sinks.K2.channel = c2

#Sink to log
a1.sinks.L1.type = logger
a1.sinks.L1.channel = c1
```

**Рисунок 1.12 — Создание конфигурационного файла Flume**

Далее, после всей настройки, требуется запустить слушателя Kafka (Рисунок 1.3).

```
[student@localhost ~]$ kafka-console-consumer --bootstrap-server localhost:9092 --topic salary --from-beginning
```

**Рисунок 1.13 — Запуск слушателя Kafka**

Также необходимо запустить агента Flume (Рисунок 1.4).

```
[student@localhost ~]$ flume-ng agent --conf $FLUME_HOME/conf --conf-file flume.conf --name a1 -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
```

**Рисунок 1.14 — Запуск агента Flume**

Запустим скрипт, написанный ранее. Он последовательно создаст в директории Spool файлы с частями данных (Рисунок 1.5).



```

Fetching data...
Data fetched and saved to file data_1717679233.3517349_1.csv
Data fetched and saved to file data_1717679243.3838525_2.csv
Data fetched and saved to file data_1717679253.3964286_3.csv
Data fetched and saved to file data_1717679263.4111536_4.csv
Data fetched and saved to file data_1717679273.424791_5.csv
Data fetched and saved to file data_1717679283.436659_6.csv
Data fetched and saved to file data_1717679293.4512072_7.csv
Data fetched and saved to file data_1717679303.4645865_8.csv
Data fetched and saved to file data_1717679313.4796329_9.csv
Data fetched and saved to file data_1717679323.4964204_10.csv
Data fetched and saved to file data_1717679333.5065124_11.csv
Data fetched and saved to file data_1717679343.521644_12.csv
Data fetched and saved to file data_1717679353.536603_13.csv
Data fetched and saved to file data_1717679363.5498934_14.csv
Data fetched and saved to file data_1717679373.5641038_15.csv
Data fetched and saved to file data_1717679383.5767334_16.csv
Data fetched and saved to file data_1717679393.5883715_17.csv
Data fetched and saved to file data_1717679403.6013865_18.csv
Data fetched and saved to file data_1717679413.6161463_19.csv
Data fetched and saved to file data_1717679423.6296597_20.csv

```

Рисунок 1.15 — Результат работы скрипта

В это время агент Flume считывает созданные файлы и логирует записи о них в консоль, а также сами данные выводятся в консольного потребителя Kafka (Рисунок 1.6, Рисунок 1.7).

```

2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 53 45 2C 46 54 2C 44 61 74 61 20 2024,SE,FT,Data
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 53 45 2C 46 54 2C 42 75 73 69 6E 2024,SE,FT,Busin
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 40 49 2C 46 54 2C 42 75 73 69 6E 2024,MI,FT,Busin
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 40 49 2C 46 54 2C 42 75 73 69 6E 2024,MI,FT,Busin
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 45 4E 2C 46 54 2C 44 61 74 61 20 2024,EN,FT,Data
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 45 4E 2C 46 54 2C 44 61 74 61 20 2024,EN,FT,Data
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 45 4E 2C 46 54 2C 44 61 74 61 20 2024,EN,FT,Data
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 53 45 2C 46 54 2C 52 65 73 65 61 2024,SE,FT,Resea
2024-06-06 23:46:34,840 (SinkRunner-PollingRunner-DefaultSinkProcessor) INFO org.apache.flume.sink.LoggerSinkProcess(LoggerSink.java:95) Event: { headers: {} body: 32 30 32 34 2C 40 49 2C 46 54 2C 44 61 74 61 20 2024,MI,FT,Data

```

Рисунок 1.16 — Информация о считывании данных в консоли

```

2024,MI,FT,Business Intelligence,78400,USD,78400,US,100,US,L
2024,SE,FT,Data Analyst,207300,USD,207300,US,0,US,M
2024,SE,FT,Data Analyst,114700,USD,114700,US,0,US,M
2024,SE,FT,Data Scientist,265000,USD,265000,US,100,US,M
2024,SE,FT,Data Scientist,220000,USD,220000,US,100,US,M
2024,SE,FT,Business Intelligence Analyst,120000,USD,120000,US,0,US,M
2024,SE,FT,Business Intelligence Analyst,100000,USD,100000,US,0,US,M
2024,SE,FT,Machine Learning Engineer,306000,USD,306000,US,0,US,M
2024,SE,FT,Machine Learning Engineer,151600,USD,151600,US,0,US,M
2024,EX,FT,Data Science,300000,USD,300000,US,100,US,M
2024,EX,FT,Data Science,220000,USD,220000,US,100,US,M
2024,SE,FT,Data Scientist,167500,USD,167500,US,0,US,M
2024,SE,FT,Data Scientist,138500,USD,138500,US,0,US,M
2024,MI,FT,Data Scientist,194000,USD,194000,US,0,US,M
2024,MI,FT,Data Scientist,125500,USD,125500,US,0,US,M
2024,SE,FT,Analytics Engineer,95000,GBP,118750,GB,0,GB,M
2024,SE,FT,Analytics Engineer,60000,GBP,75000,GB,0,GB,M
2024,SE,FT,Data Engineer,194500,USD,194500,US,0,US,M
2024,SE,FT,Data Engineer,147368,USD,147368,US,0,US,M
2024,MI,FT,Data Engineer,165000,USD,165000,US,0,US,M
2024,MI,FT,Data Engineer,117000,USD,117000,US,0,US,M
2024,SE,FT,Data Analyst,170000,USD,170000,US,0,US,M
2024,SE,FT,Data Analyst,148000,USD,148000,US,0,US,M
2024,SE,FT,Data Manager,160500,USD,160500,US,100,US,M
2024,SE,FT,Data Manager,107000,USD,107000,US,100,US,M
2024,SE,FT,Analytics Engineer,500000,USD,500000,US,0,US,M
2024,SE,FT,Analytics Engineer,80000,USD,80000,US,0,US,M
2024,MI,FT,Data Infrastructure Engineer,241000,USD,241000,US,0,US,M
2024,MI,FT,Data Infrastructure Engineer,189000,USD,189000,US,0,US,M
2024,SE,FT,Data Scientist,167500,USD,167500,US,0,US,M
2024,SE,FT,Data Scientist,138500,USD,138500,US,0,US,M
2024,SE,FT,Research Scientist,200000,USD,200000,US,100,US,M
2024,SE,FT,Research Scientist,148500,USD,148500,US,100,US,M
2024,SE,FT,Research Engineer,273000,USD,273000,US,0,US,M
2024,SE,FT,Research Engineer,166500,USD,166500,US,0,US,M
2024,SE,FT,Data Analyst,106260,USD,106260,US,0,US,M
2024,SE,FT,Data Analyst,82000,USD,82000,US,0,US,M
2024,MI,FT,Business Intelligence Analyst,115000,USD,115000,US,100,US,M
2024,MI,FT,Business Intelligence Analyst,100000,USD,100000,US,100,US,M

```

Рисунок 1.17 — Данные в консольном потребителе Kafka



Проверим наличие таблицы hive\_salary (Рисунок 1.20).

```
0: jdbc:hive2://> show tables;
OK
+-----+
| tab_name |
+-----+
| hive_salary |
+-----+
```

Рисунок 1.20 — Проверка наличия таблицы hive\_salary

Также проверим содержимое таблицы при помощи простого SQL запроса (Рисунок 1.).

```
0: jdbc:hive2://> select * from hive_salary limit 5;
OK
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| hive_salary.work_year | hive_salary.experience_level | hive_salary.employment_type | hive_salary.job_title | hive_salary.salary | hive_salary.salary_currency | hive_salary.salary_in_usd | hive_salary.employee_residence | hive_salary.employee_residence |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2024 | SE | FT | AI Engineer | 202730 | USD | 202730 | US | 0 |
| 2024 | SE | FT | AI Engineer | 92118 | USD | 92118 | US | 0 |
| 2024 | SE | FT | Data Engineer | 130500 | USD | 130500 | US | 0 |
| 2024 | SE | FT | Data Engineer | 96000 | USD | 96000 | US | 0 |
| 2024 | SE | FT | Machine Learning Engineer | 190000 | USD | 190000 | US | 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows selected (4.640 seconds)
```

Рисунок 1.6 — Вывод содержимого таблицы hive\_salary

После создания таблицы осуществляется запуск Spark для дальнейшей предобработки и анализа данных. Apache Spark — это фреймворк для обработки и анализа больших объёмов информации, входящий в инфраструктуру Hadoop. [1.8]

## 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

### 2.1 Анализ и визуализация данных

Для подключения pyspark к базе данных использована библиотека pyspark.sql и импортирован модуль SparkSession. Далее была создана spark-сессия и осуществлено подключение к базе данных Hive, а также создан dataframe из содержимого таблицы hive\_salary (Рисунок 2.1). [2.1]

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import corr
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import udf
from pyspark.sql.types import ArrayType, DoubleType
from pyspark.ml.linalg import DenseVector
from pyspark.ml.stat import Correlation
from pyspark.ml import Pipeline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

salary_df = spark.sql("SELECT * FROM hive_salary")
salary_df.show(5)
salary_df.printSchema()
```

2024-06-08 14:45:21,983 WARN conf.HiveConf: HiveConf of name hive.stats.jdbc.timeout does not exist  
2024-06-08 14:45:21,984 WARN conf.HiveConf: HiveConf of name hive.stats.retries.wait does not exist

work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_res
2024	SE	FT	AI Engineer	202730	USD	202730	
2024	SE	FT	AI Engineer	92118	USD	92118	
2024	SE	FT	Data Engineer	130500	USD	130500	
2024	SE	FT	Data Engineer	96000	USD	96000	
2024	SE	FT	Machine Learning ...	190000	USD	190000	

only showing top 5 rows

Рисунок 2.1 — Создание spark-сессии и создание dataframe из содержимого таблицы

Чтобы убедиться в том, что столбцы соответствуют заданным в MariaDB значениям, просмотрена схема данных созданного фрейма (Рисунок 2.2).



```

root
|-- work_year: integer (nullable = true)
|-- experience_level: string (nullable = true)
|-- employment_type: string (nullable = true)
|-- job_title: string (nullable = true)
|-- salary: integer (nullable = true)
|-- salary_currency: string (nullable = true)
|-- salary_in_usd: integer (nullable = true)
|-- employee_residence: string (nullable = true)
|-- remote_ratio: integer (nullable = true)
|-- company_location: string (nullable = true)
|-- company_size: string (nullable = true)

```

**Рисунок 2.2 — Просмотр схемы данных**

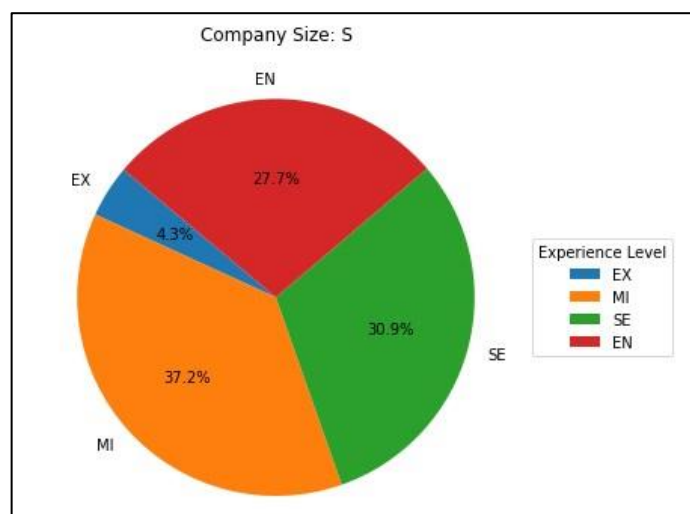
Столбец `experience_level` принимает следующие категориальные значения:

- EN — инженер начального уровня;
- MI — инженер среднего уровня;
- SE — старший инженер;
- EX — инженер-эксперт.

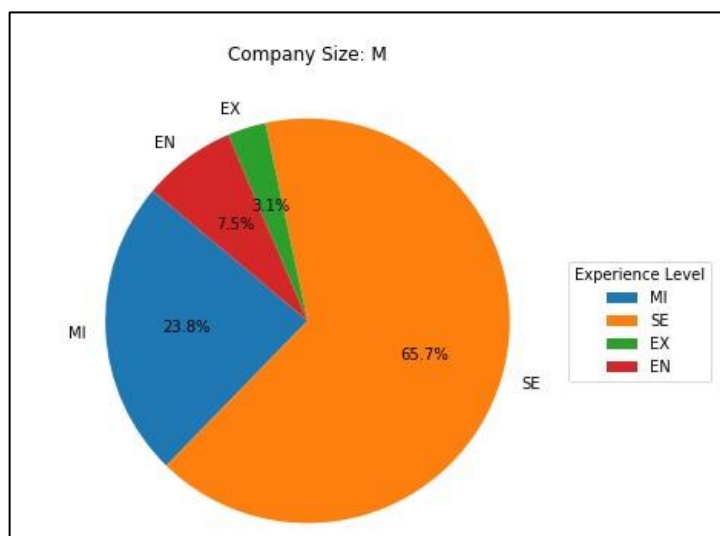
Столбец `company_size` принимает следующие категориальные значения:

- S — малая компания;
- M — средняя компания;
- L — большая компания.

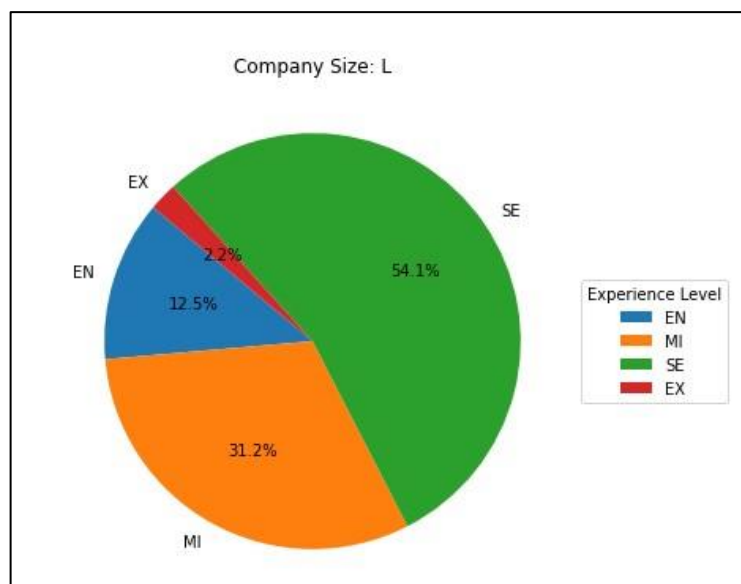
Построим круговые диаграммы, которые будут отражать долю каждого грейда в компаниях разных размеров. (Рисунок 2.3, Рисунок 2.4, Рисунок 2.5).



**Рисунок 2.3 — Доля каждого грейда в малых компаниях**



**Рисунок 2.4 — Доля каждого грейда в средних компаниях**



**Рисунок 2.5 — Доля каждого грейда в больших компаниях**

Из построенных диаграмм можно сказать, что в малых компаниях доля инженеров-экспертов не велика, а остальных — примерно одинаковое количество. В средних компаниях преобладают старшие инженеры и инженеры среднего уровня. В больших компаниях ситуация примерно такая же, как и в средних, только доля инженеров-экспертов больше, а младших инженеров — меньше.

В компаниях малых размеров нагрузка и задачи распределяются более

равномерно между сотрудниками разных грейдов. Но с ростом размера компании растёт потребность в более опытных сотрудниках.

В компаниях больших и средних размеров мало востребованы младшие инженеры. Но стоит заметить, в больших компаниях их немного больше, что может быть связано с тем, что большие компании, имея большие финансовые возможности, могут позволить себе набирать больше младших инженеров, для дальнейшего их карьерного роста внутри компании.

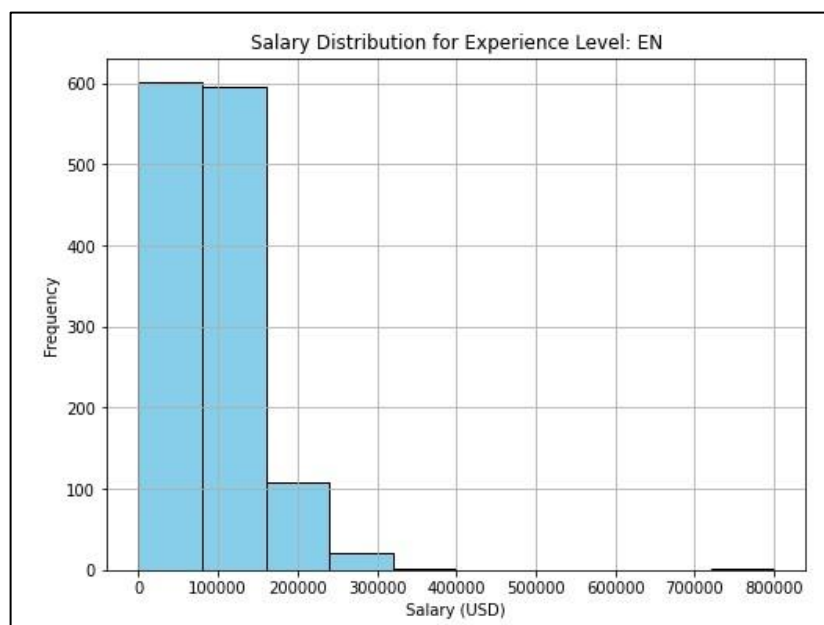
Далее, попытаемся ответить на актуальный в современном мире вопрос — зависит ли зарплата от формата работы (в офисе или удалённо). Для этого посчитаем коэффициент корреляции между столбцом `salary_in_usd` и `remote_ratio` (Рисунок 2.3).

```
Корреляция между remote_ratio и salary_in_usd: -0.057288865300753646
```

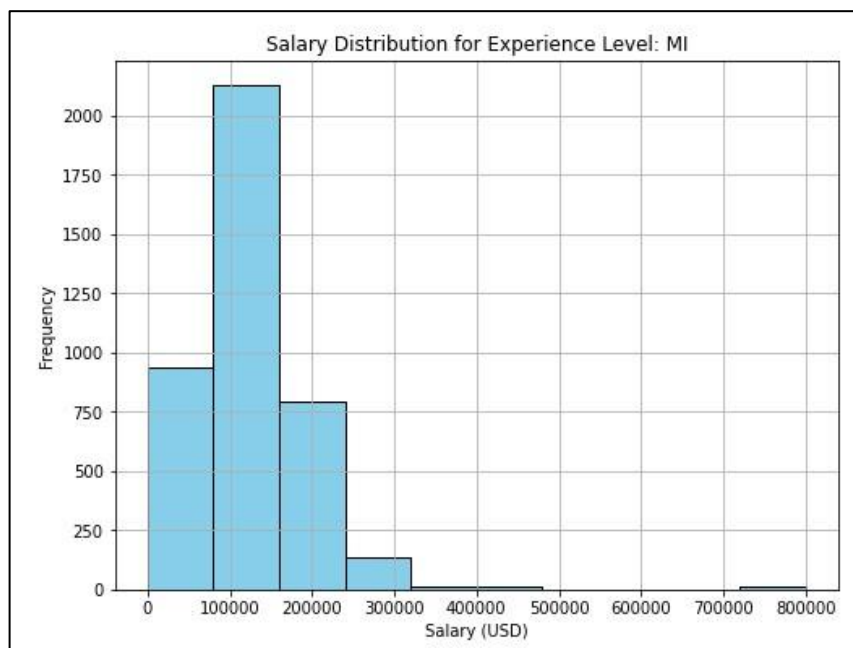
**Рисунок 2.3 — Корреляция между зарплатой и коэффициентом удалённости от работы**

Корреляция близка к 0, поэтому никакой зависимости между этими столбцами нет.

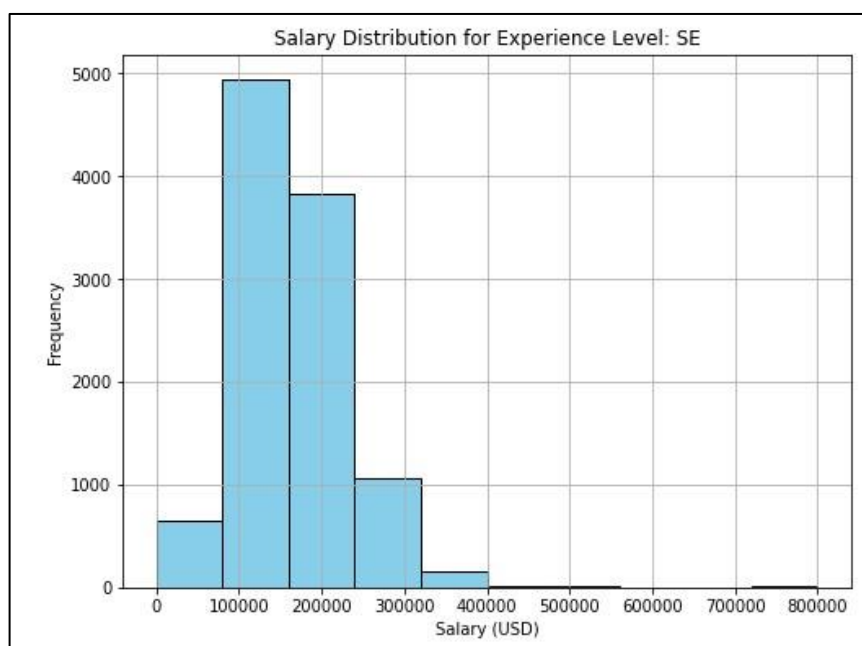
Рассчитаем распределение зарплат относительно каждого грейда (Рисунок 2.7, Рисунок 2.8, Рисунок 2.9, Рисунок 2.10).



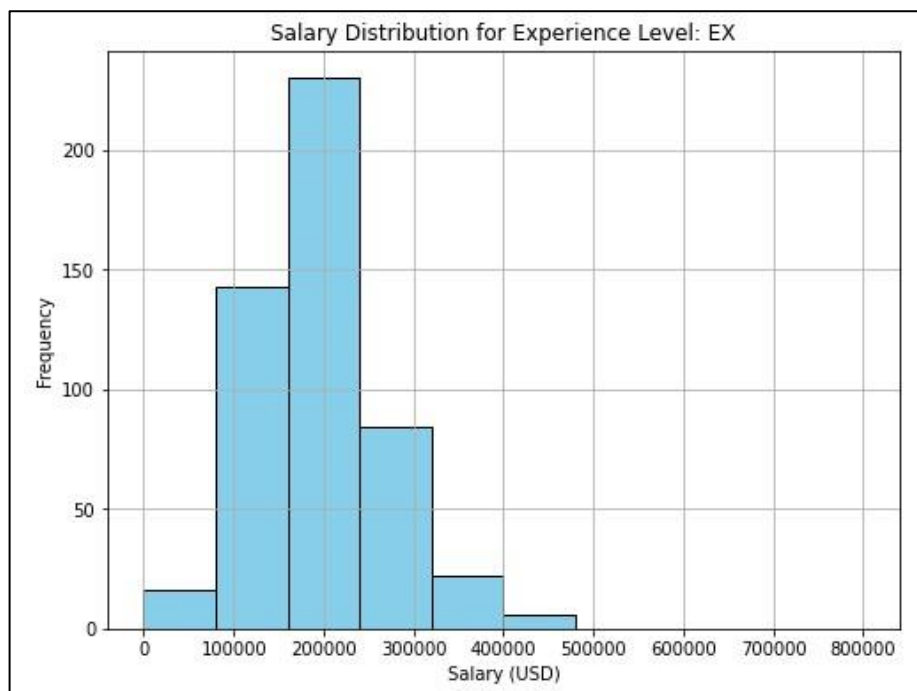
**Рисунок 2.7 — Распределение зарплат среди инженеров начального уровня**



**Рисунок 2.8 — Распределение зарплат среди инженеров среднего уровня**



**Рисунок 2.9 — Распределение зарплат среди старших инженеров**

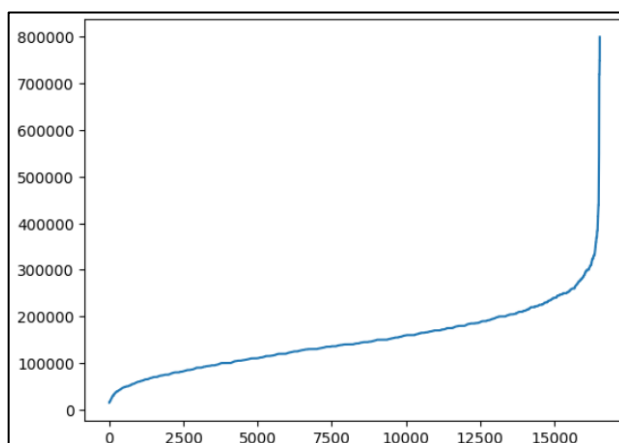


**Рисунок 2.10 — Распределение зарплат среди инженеров-экспертов**

Результаты получились вполне ожидаемыми — чем опытнее инженер, тем в среднем выше его зарплата.

## 2.2 Регрессионный анализ

Отсортируем данные в целевом столбце и посмотрим на диапазон их возможных значений. Для этого построим график с представленными значениями целевого признака (Рисунок 2.11). [2.2]



**Рисунок 2.11 — Просмотр распределения данных**

На рисунке видно, что данные имеют достаточно понятную структуру для построения модели регрессии на их основе.

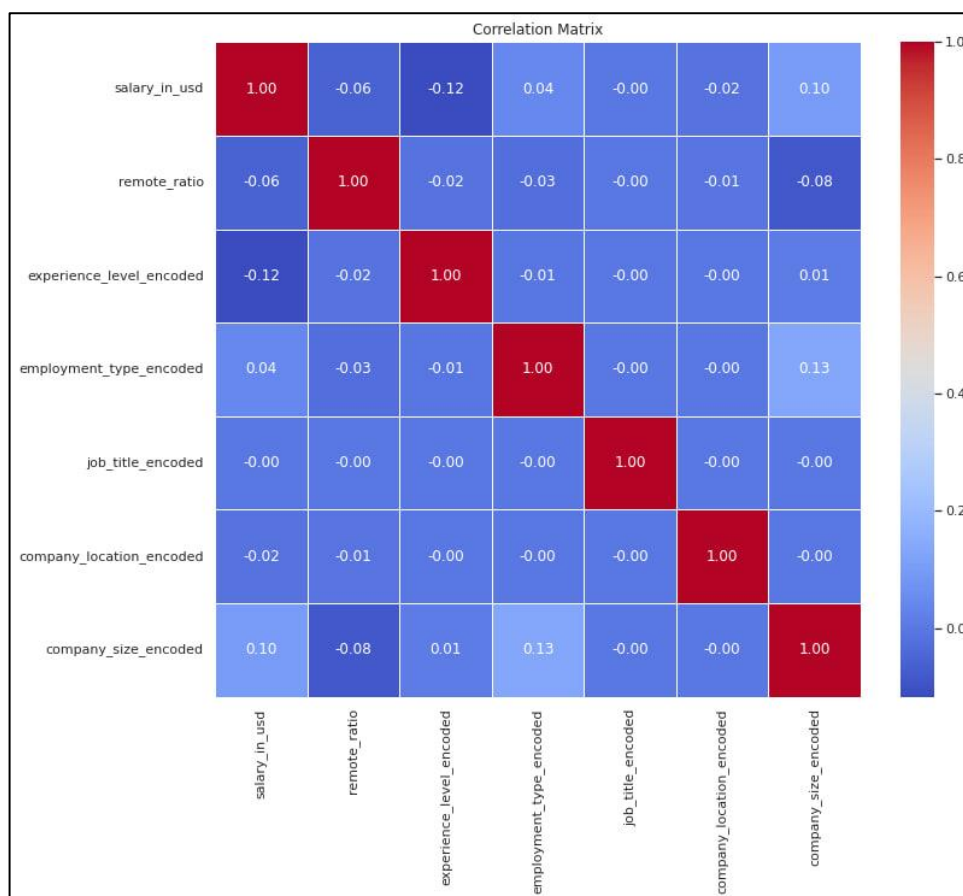
Большинство признаков в наборе данных категориальные. Чтобы использовать их для дальнейшего анализа произведём кодирование категориальных признаков при помощи one-hot кодирования (Рисунок 2.12).

salary_in_usd	remote_ratio	experience_level_encoded	employment_type_encoded	job_title_encoded	company_location_encoded	company_size_encoded
202730	0	0.3333333333333333	0.3333333333333333	0.006493506493506494	0.0131578947	
36842105	0.5					
92118	0	0.3333333333333333	0.3333333333333333	0.006493506493506494	0.0131578947	
36842105	0.5					
130500	0	0.3333333333333333	0.3333333333333333	0.006493506493506494	0.0131578947	
36842105	0.5					
96000	0	0.3333333333333333	0.3333333333333333	0.006493506493506494	0.0131578947	
36842105	0.5					
190000	0	0.3333333333333333	0.3333333333333333	0.006493506493506494	0.0131578947	
36842105	0.5					

only showing top 5 rows

**Рисунок 2.12 — Просмотр закодированных данных**

Для построения и обучения регрессионной модели машинного обучения необходимо выбрать признаки, которые будут использоваться в уравнении регрессии. Построим корреляционную матрицу, чтобы найти взаимосвязи целевой переменной с другими и проверить признаки на мультиколлинеарность в числовом коэффициенте (Рисунок 2.13). [2.3]



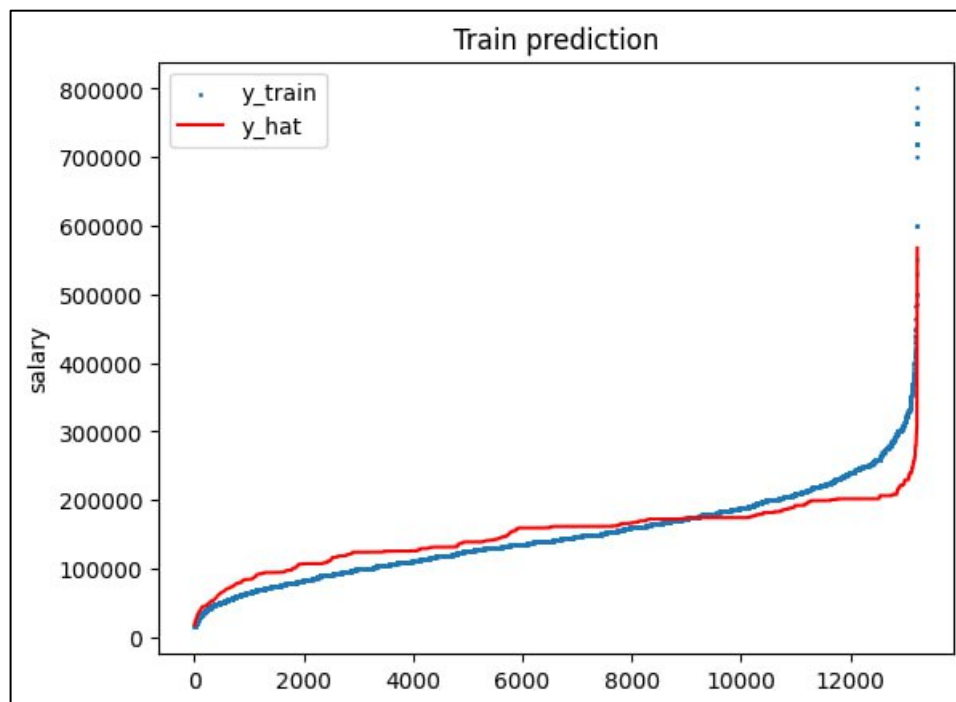
**Рисунок 2.13 — Тепловая карта корреляционной матрицы**

Между признаками отсутствует мультиколлинеарность, а целевая переменная (`salary_in_usd`) слабо связана с остальными. Это можно объяснить тем, что большинство признаков изначально были категориальными, поэтому явной связи нет.

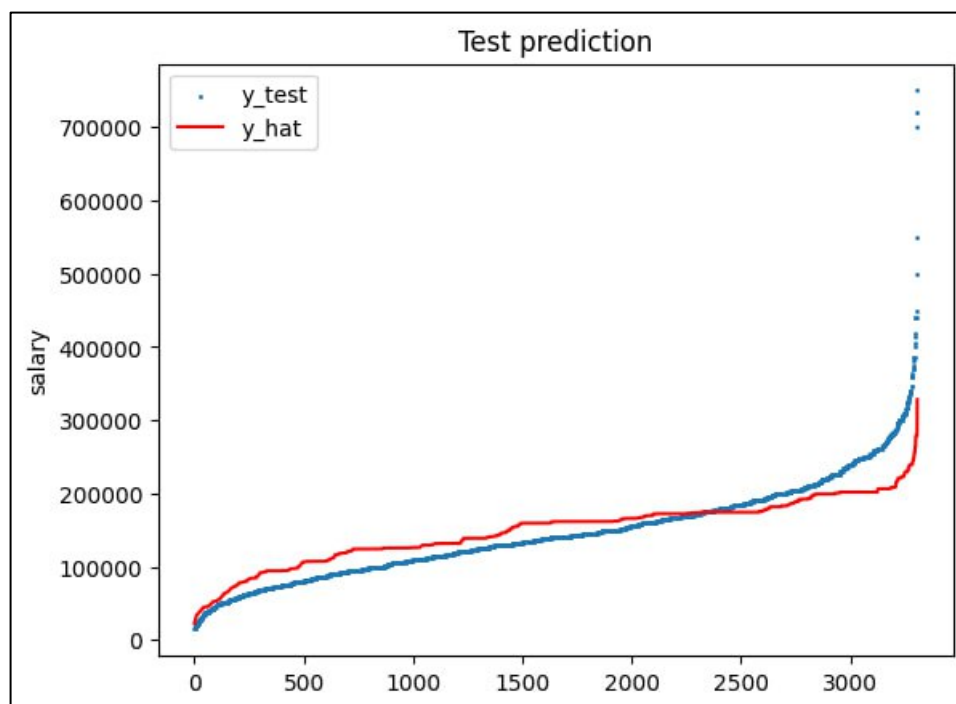
Разделим исходный датасет на тренировочную и тестовую выборку в соотношении 80 на 20. Для прогнозирования будем использовать ансамблевую модель нелинейной регрессии `RandomForestRegression`. Данная модель способна работать со сложными, нелинейными зависимостями между признаками и целевой переменной и захватывать сложные паттерны в данных, так как это ансамбль, состоящий из множества деревьев решений, использующий случайности в выборе подмножеств признаков и данных. Также `RandomForestRegressor` менее чувствителен к выбросам и шуму.

Обучим модель на тренировочной выборке и воспользуемся ей для совершения прогнозирования значений целевого признака на тренировочной

(Рисунок 2.14) и тестовой выборке (Рисунок 2.15). [2.5]



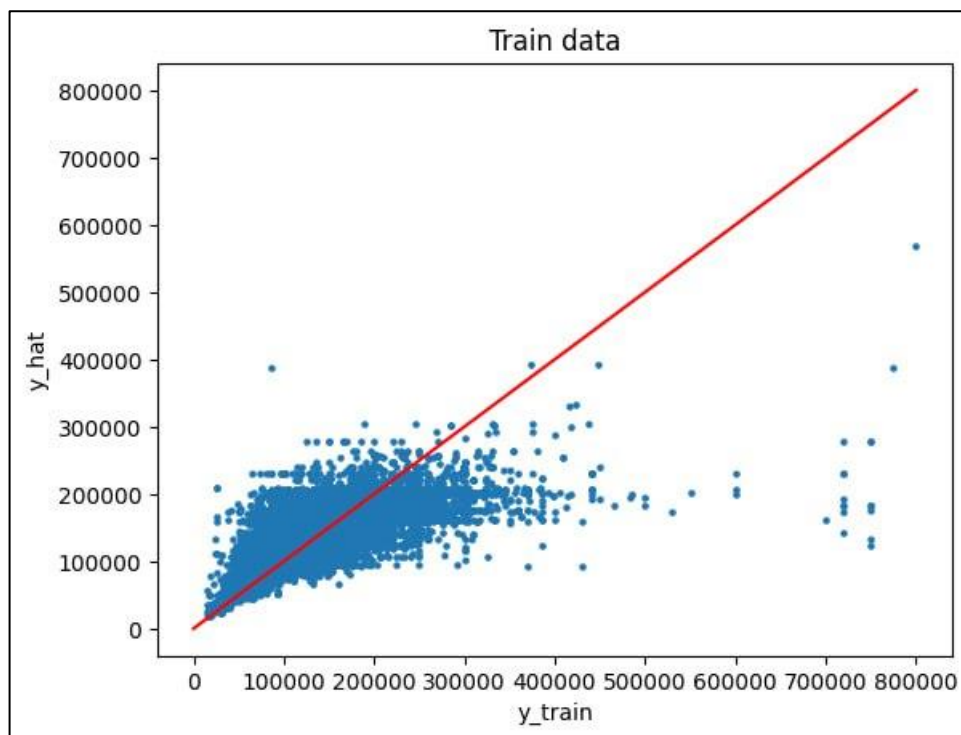
**Рисунок 2.14 — Предсказание модели на тренировочной выборке**



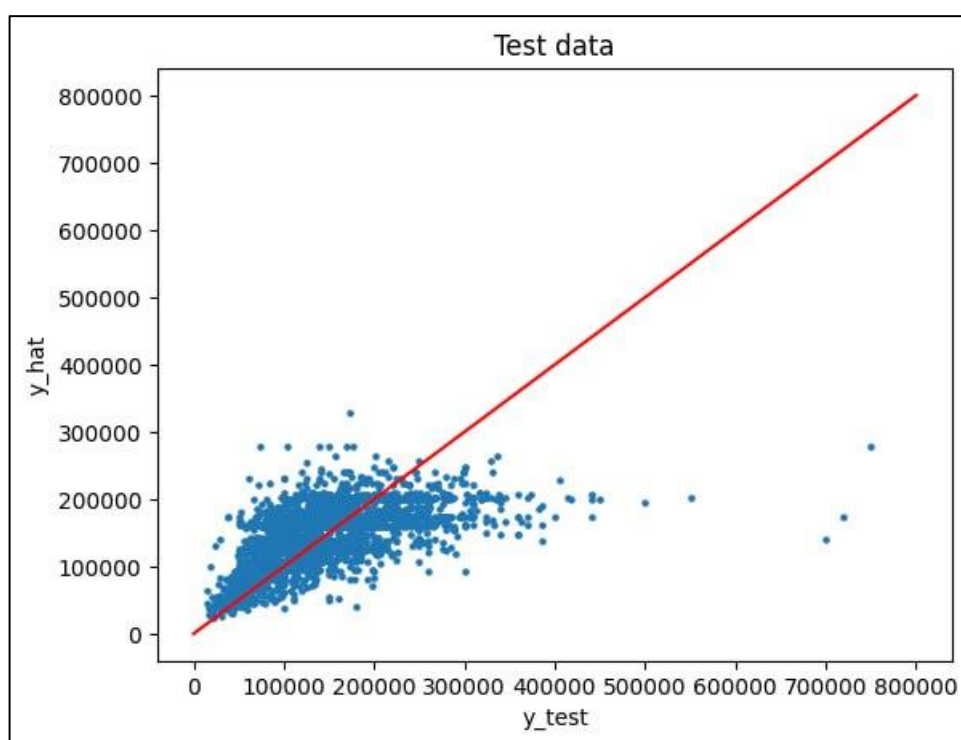
**Рисунок 2.15 — Предсказание модели на тестовой выборке**

Также построим графики для тренировочной (Рисунок 2.16) и тестовой выборки (Рисунок 2.17), расположив по оси X фактические значения, а по оси Y — предсказанные значения как результат работы модели.





**Рисунок 2.16 — Сравнение результатов на тренировочной выборке**



**Рисунок 2.17 — Сравнение результатов на тестовой выборке**

Рассчитаем коэффициент  $R^2$  для определения качества полученной модели:

- $R^2$  на тренировочной выборке — 0.4008;
- $R^2$  на тестовой выборке — 0.3435. [2.6]

Как мы видим, результат построенной модели получился довольно низким. Скорее всего это опять же связано с большим количеством категориальных признаков и их кодированием.

## ЗАКЛЮЧЕНИЕ

Анализ данных является одним из важнейших инструментов для принятия обоснованных решений в современном мире, где большое количество компаний работают с огромными объемами информации. Он помогает выявить полезную информацию и знания, скрытые за массой данных, и использовать их для оптимизации и повышения эффективности многих процессов.

В данной курсовой работе рассмотрены методы построения конвейера предобработки и маршрутизации данных, а также проведен их многофакторный анализ. По результатам этого анализа можно утверждать, что определенные параметры существенно влияют на количество производимой ветряными турбинами энергии. Также проведена сегментация данных относительно различных характеристик турбин и условий их эксплуатации. Оценены средние показатели генерации энергии, и выявлены признаки, наиболее влияющие на производительность турбин.

Цель работы — изучить методы и инструменты, необходимые для сбора, обработки и анализа данных о заработных платах дата-инженеров — достигнута.

В ходе выполнения работы выполнены следующие задачи:

- построен конвейер для предобработки и маршрутизации больших данных;
- проведен анализ факторов, влияющих на заработную плату дата-инженеров;
- построены и проанализированы распределения зарплат дата-инженеров в зависимости от грейда;
- проведена визуализация данных;
- построена регрессионная модель машинного обучения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

- 1.1. Kaggle. Russia Real Estate [Электронный ресурс]. — Режим доступа:  
<https://www.kaggle.com/datasets/chopper53/data-engineer-salary-in-2024>.
- 1.2. Codernet. Что такое MariaDB [Электронный ресурс]. — Режим доступа:  
[https://codernet.ru/articles/sql/chto\\_takoe\\_mariadb\\_gde\\_ispolzuetsya\\_eta\\_sistema\\_upravleniya/](https://codernet.ru/articles/sql/chto_takoe_mariadb_gde_ispolzuetsya_eta_sistema_upravleniya/).
- 1.3. Школа больших данных. Apache Hive [Электронный ресурс]. — Режим доступа:  
<https://bigdataschool.ru/wiki/hive>.
- 1.4. User Guide Apache Sqoop [Электронный ресурс]. — Режим доступа:  
<https://docs.cloudera.com/sqoop/1.4.7.1.6.0/user-guide/index.html>.
- 1.5. YandexCloud. Apache Kafka где применяется [Электронный ресурс]. — Режим доступа:  
<https://cloud.yandex.ru/blog/posts/2021/02/managed-kafka-overview>.
- 1.6. Habr. Flume — управление потоками. [Электронный ресурс]. — Режим доступа: <https://habr.com/ru/companies/dca/articles/280386/>.
- 1.7. Курс лекций Samsung Innovation Campus [Электронный ресурс]. — Режим доступа:  
[https://myitschool.ru/edu/mod/scorm/player.php?a=2&currentorg=rus\\_sic\\_big\\_data\\_chapter1\\_f\\_final\\_%28s\\_pravkami\\_teksta\\_pod\\_slaidami%29\\_organization&scoid=18\\_](https://myitschool.ru/edu/mod/scorm/player.php?a=2&currentorg=rus_sic_big_data_chapter1_f_final_%28s_pravkami_teksta_pod_slaidami%29_organization&scoid=18_)
- 1.8. J. W. Tukey. «Exploratory Data Analysis». Pearson, 1977. С. 32-50 — Режим доступа:  
<https://www.piter.com/product/mashinnoe-obuchenie-na-yazyke-python-s-chego-nachat-i-kak-stat-professionalom>.

## ПРАКТИЧЕСКАЯ ЧАСТЬ

- 2.1. Страчунский К. С. "Анализ больших данных. Решения на основе Apache Spark и Hadoop" (2018, издательство: Питер, 288 с.).
- 2.2. Глинский В. В., Ионин В. Г. Статистический анализ. — М.: Инфра-М, 2002. — 241 с. — (Высшее образование). — 5000 экз. — ISBN 5-16-001293-1.
- 2.3. Меридиан А. "Apache Spark: лучшие практики и примеры" (2018, издательство: БХВ-Петербург, 320 с.).
- 2.4. P. Bruce, A. Bruce. "Practical Statistics for Data Scientists: 50 Essential Concepts". O'Reilly Media, 2017 — Режим доступа: <https://www.oreilly.com/library/view/practical-statistics-for/9781491952955/>.
- 2.5. А. Кибзун, Е. Кузнецова, Н. Загоруйко. "Математическая статистика для машинного обучения и анализа данных". ДМК Пресс, 2018. С. 45-78 — Режим доступа: <https://dmkpress.com/catalog/computer/programming/mathematics/978-5-97060-496-7/>.
- 2.6. Regression Metrics in PySpark [Электронный ресурс]. — Режим доступа: <https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.lib.evaluation.RegressionMetrics.html>.
- 2.7. Норман Дрейпер, Гарри Смит. Прикладной регрессионный анализ. Множественная регрессия = Applied Regression Analysis. — 3-е изд. — М.: «Диалектика», 2007. — С. 912. — ISBN 0-471-17082-8. 2.

## ПРИЛОЖЕНИЯ

Приложение А — Программный код для скрипта записи данных по чанкам в MariaDB.

Приложение В — Программный код PySpark.

## Приложение А

```
import pymysql
import schedule
import time
import os
import csv

folder = "/home/student/Spool/"
SQL_query = "SELECT * FROM stat.salary"

def data_to_csv():
    connection = pymysql.connect(
        host = "localhost",
        port = 3306,
        user = "student",
        password = "student",
        database = "stat")

    cursor = connection.cursor()
    cursor.execute(SQL_query)
    rows = cursor.fetchall()
    cursor.close()
    connection.close()

    total_rows = len(rows)
    row_limit = round(total_rows * 0.05)
    num_files = (total_rows + row_limit - 1) // row_limit
```

```

for i in range(num_files):
    time.sleep(10)
    file_name = f"data_{time.time()}_{i + 1}.csv"
    file_path = os.path.join(folder, file_name)

    start_index = i * row_limit
    end_index = start_index + row_limit if start_index + row_limit != 0
else total_rows
    data = rows[start_index:end_index]

    with open(file_path, 'w', newline = "") as file:
        writer = csv.writer(file)
        writer.writerow(["work_year", "experience_level", \
                           "employment_type", "job_title", \
                           "salary", "salary_currency", \
                           "salary_in_usd", "employee_residence", \
                           "remote_ratio", "company_location", \
                           "company_size"])
        writer.writerows(data)

    print(f>Data fetched and saved to file {file_name}")

def job():
    print("Fetching data...")
    data_to_csv()

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

```



```
while True:  
    schedule.run_pending()  
    time.sleep(1)
```

## Приложение В

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import corr
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.sql.functions import udf, col
from pyspark.sql.types import ArrayType, DoubleType
from pyspark.ml.linalg import DenseVector
from pyspark.ml.regression import LinearRegression,
RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.stat import Correlation
from pyspark.ml import Pipeline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

salary_df = spark.sql("SELECT * FROM hive_salary")
salary_df.show(5)
salary_df.printSchema()

salary_df.na.drop()

aggregated_df=salary_df.groupBy("company_size","experience_level").count()
aggregated_df.show()
# Конвертация Spark DataFrame в Pandas DataFrame
pandas_df = aggregated_df.toPandas()

company_sizes = pandas_df['company_size'].unique()
```

```

fig, axs = plt.subplots(len(company_sizes), 1, figsize=(50, 15))

for i, size in enumerate(company_sizes):
    data = pandas_df[pandas_df['company_size'] == size]
    labels = data['experience_level']
    sizes = data['count']

    wedges, texts, autotexts = axs[i].pie(sizes, labels=labels,
autopct='%1.1f%%', startangle=140)
    axs[i].set_title(f'Company Size: {size}')

    axs[i].legend(wedges, labels, title="Experience Level", loc="center left",
bbox_to_anchor=(1, 0, 0.5, 1))

plt.tight_layout()
plt.show()

corr = salary_df.corr("remote_ratio", "salary_in_usd")
print("Корреляция между remote_ratio и salary_in_usd:", corr)

experience_levels=
salary_df.select("experience_level").distinct().rdd.flatMap(lambda x: x).collect()

max_salary = salary_df.agg({"salary_in_usd": "max"}).collect()[0][0]

for level in experience_levels:
    filtered_data = salary_df.filter(salary_df["experience_level"] == level)

    pandas_df = filtered_data.toPandas()

```

```

plt.figure(figsize=(8, 6))
plt.hist(pandas_df["salary_in_usd"], bins=10, range=(0, max_salary),
color='skyblue', edgecolor='black')
plt.title(f'Salary Distribution for Experience Level: {level}')
plt.xlabel('Salary (USD)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

eg_df = salary_df.select("experience_level", \
                        "employment_type", \
                        "job_title", \
                        "salary_in_usd", \
                        "remote_ratio", \
                        "company_location", \
                        "company_size")
reg_df.show(5)

categorical_cols = ["experience_level", "employment_type", "job_title",
"company_location", "company_size"]
numerical_cols = ["salary_in_usd", "remote_ratio"]
indexers = [StringIndexer(inputCol=col, outputCol=col + "_index") for col in
categorical_cols]
encoders = [OneHotEncoder(inputCol=indexer.getOutputCol(),
outputCol=col + "_encoded")
              for indexer, col in zip(indexers, categorical_cols)]
pipeline = Pipeline(stages=indexers + encoders)
model = pipeline.fit(reg_df)
df_transformed = model.transform(reg_df)

```

```

index_cols = [col + "_index" for col in categorical_cols]
df_transformed = df_transformed.drop(*index_cols)

def vector_to_array(v):
    return v.toArray().tolist()

vector_to_array_udf = udf(vector_to_array, ArrayType(DoubleType()))
for encoded_col in [col + "_encoded" for col in categorical_cols]:
    df_transformed = df_transformed.withColumn(encoded_col,
vector_to_array_udf(df_transformed[encoded_col]))

for encoded_col in [col + "_encoded" for col in categorical_cols]:
    size = len(df_transformed.select(encoded_col).first()[0])
    for i in range(size):
        df_transformed = df_transformed.withColumn(f"{encoded_col}_{i}",
df_transformed[encoded_col][i])

    avg_expr = sum([col(f"{encoded_col}_{i}") for i in range(size)]) / size
    df_transformed = df_transformed.withColumn(encoded_col, avg_expr)

df_transformed = df_transformed.drop(*[f"{encoded_col}_{i}" for i in
range(size)])

df_pandas = df_transformed.select(numerical_cols + [col for col in
df_transformed.columns if col not in numerical_cols]).toPandas()
corr_matrix = df_pandas.corr()

sns.set(style="white")
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f",

```

```

linewidths=.5)

plt.title("Correlation Matrix")

plt.show()


corr_df = df_transformed.select(numerical_cols + [col for col in
df_transformed.columns if col not in numerical_cols])

corr_df = corr_df.drop(*categorical_cols)

corr_df.show(5)

salary_df = corr_df.select('salary_in_usd')

salary_pd = salary_df.toPandas()


plt.figure(figsize=(10, 6))

plt.plot(sorted(salary_pd['salary_in_usd']))

plt.xlabel('Index')

plt.ylabel('Salary')

plt.title('Salary Distribution')

plt.show()


sorted_df = corr_df.orderBy("salary_in_usd")


features = [col for col in sorted_df.columns if col != 'salary_in_usd']

assembler = VectorAssembler(inputCols=features, outputCol='features')

data = assembler.transform(sorted_df).select('features', 'salary_in_usd')


train_data, test_data = data.randomSplit([0.8, 0.2], seed=12345)

rf=RandomForestRegressor(featuresCol='features', labelCol='salary_in_usd',
numTrees=100)

rf_model = rf.fit(train_data)


train_predictions = rf_model.transform(train_data)

```

```

test_predictions = rf_model.transform(test_data)

train_predictions_pd=train_predictions.select('salary_in_usd',
'prediction').toPandas().sort_values(by='salary_in_usd').reset_index(drop=True)

test_predictions_pd=test_predictions.select('salary_in_usd',
'prediction').toPandas().sort_values(by='salary_in_usd').reset_index(drop=True)

evaluator=RegressionEvaluator(labelCol='salary_in_usd',
predictionCol='prediction', metricName='r2')

r2_train = evaluator.evaluate(train_predictions)
r2_test = evaluator.evaluate(test_predictions)

print(f'R2 на тренировочной выборке: {r2_train}')
print(f'R2 на тестовой выборке: {r2_test}')

plt.figure(figsize=(12, 10))

plt.subplot(2, 2, 1)
plt.plot(train_predictions_pd.index,      train_predictions_pd['salary_in_usd'],
label='Actual')
plt.plot(train_predictions_pd.index,      train_predictions_pd['prediction'],
label='Predicted')
plt.title('Train Set Predictions')
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(test_predictions_pd.index,      test_predictions_pd['salary_in_usd'],
label='Actual')
plt.plot(test_predictions_pd.index,      test_predictions_pd['prediction'],
label='Predicted')

```

```

plt.title('Test Set Predictions')
plt.legend()

plt.subplot(2, 2, 3)
plt.scatter(train_predictions_pd['salary_in_usd'],
train_predictions_pd['prediction'])
plt.plot([train_predictions_pd['salary_in_usd'].min(),
train_predictions_pd['salary_in_usd'].max()],
[train_predictions_pd['salary_in_usd'].min(),
train_predictions_pd['salary_in_usd'].max()], color='red')
plt.title('Train Set Actual vs Predicted')
plt.xlabel('Actual Salary')
plt.ylabel('Predicted Salary')

plt.subplot(2, 2, 4)
plt.scatter(test_predictions_pd['salary_in_usd'],
test_predictions_pd['prediction'])
plt.plot([test_predictions_pd['salary_in_usd'].min(),
test_predictions_pd['salary_in_usd'].max()],
[test_predictions_pd['salary_in_usd'].min(),
test_predictions_pd['salary_in_usd'].max()], color='red')
plt.title('Test Set Actual vs Predicted')
plt.xlabel('Actual Salary')
plt.ylabel('Predicted Salary')

plt.tight_layout()
plt.show()

```