Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики управления и технологий

Инструменты хранения и анализа больших данных

Смоляков Руслан Игоревич БД-241м

Лабораторная работа 1.1. Введение в большие данные и их хранение. Инструменты обработки больших данных (Hadoop)

Вариант 23

Направление подготовки/специальность
38.04.05 - Бизнес-информатика
Бизнес-аналитика и большие данные
(очная форма обучения)

Руководитель дисциплины:
Босенко Т.М., доцент департамента
информатики, управления и технологий,
кандидат технических наук

Москва
2025

Введение

Цель: изучить основные операции и функциональные возможности системы, что позволит понять принципы работы с данными и распределенными вычислениями.

Основная часть

Переходим на нового пользователя:

```
devops@devopsvm:~$ sudo su - hadoop
hadoop@devopsvm:~$
```

Шаг 1. Запуск Hadoop.

start-dfs.sh

start-yarn.sh

```
hadoop@devopsvm:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [devopsvm]
2025-03-27 00:02:05,618 WARN util.NativeCodeLoader: Unable to load native-hado
op library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@devopsvm:~$ S
```

Шаг 2. Проверка работы Hadoop.

jps

```
hadoop@devopsvm:~$ jps
8113 ResourceManager
7813 SecondaryNameNode
7365 NameNode
8600 Jps
8234 NodeManager
7630 DataNode
hadoop@devopsvm:~$
```

Шаг 3. Подготовка рабочего пространства

#Создание директории в HDFS
hdfs dfs -mkdir -p /user01/hadoop/input

```
hadoop@devopsvm:~$ hdfs dfs -mkdir -p /user01/hadoop/input
2025-03-27 00:04:37,919 WARN util.NativeCodeLoader: Unable to load native-hado
op library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$
```

Шаг 4. Загрузка и подготовка данных

Скачивание файла с данными

wget
https://raw.githubusercontent.com/BosenkoTM/Distributed_systems/main/practice/2024/l w_01/GDP.csv

```
hadoop@devopsvm:~$ wget https://raw.githubusercontent.com/BosenkoTM/Distribute
d_systems/main/practice/2024/lw_01/GDP.csv
--2025-03-27 00:06:53--  https://raw.githubusercontent.com/BosenkoTM/Distribut
ed_systems/main/practice/2024/lw_01/GDP.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109
.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.10
9.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 30268 (30K) [text/plain]
Saving to: ‘GDP.csv.1’

GDP.csv.1           100%[==================>]  29.56K  --.-KB/s    in 0.006s

2025-03-27 00:06:53 (4.49 MB/s) - ‘GDP.csv.1’ saved [30268/30268]

hadoop@devopsvm:~$
```

Шаг 5. Обработка данных с помощью Spark

spark-shell

```
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.3
      /_/

Using Scala version 2.12.17 (OpenJDK 64-Bit Server VM, Java 11.0.24)
Type in expressions to have them evaluated.
Type :help for more information.

scala> S
```

// Загрузка данных из HDFS с правильным URI

val data = spark.read.option("header", "true").csv("file:///home/hadoop/GDP.csv")

```
scala> val data = spark.read.option("header", "true").csv("file:///home/hadoop
/GDP.csv")
[Stage 0:>                                                          (0 + 1) /
[Stage 0:============================================================(1 + 0) /

data: org.apache.spark.sql.DataFrame = [Country: string, Year: string ... 21 m
ore fields]

scala>
```

// Проверка схемы данных

data.printSchema()

```
scala> data.printSchema()
root
 |-- Country: string (nullable = true)
 |-- Year: string (nullable = true)
 |-- GDP: string (nullable = true)
 |-- Urban_population: string (nullable = true)
 |-- Industry: string (nullable = true)
 |-- Business: string (nullable = true)
 |-- Mining: string (nullable = true)
 |-- Manufacturing: string (nullable = true)
 |-- Electricity_supply: string (nullable = true)
 |-- Water_supply: string (nullable = true)
 |-- Construction: string (nullable = true)
 |-- Retail trade: string (nullable = true)
 |-- Transportation: string (nullable = true)
 |-- Accommodation: string (nullable = true)
 |-- Information: string (nullable = true)
 |-- Financial: string (nullable = true)
 |-- Real estate : string (nullable = true)
 |-- Professional_scientific: string (nullable = true)
 |-- Administrative: string (nullable = true)
 |-- Education: string (nullable = true)
 |-- Human_health: string (nullable = true)
 |-- Arts: string (nullable = true)
 |-- Other: string (nullable = true)
```

// Вычисление среднего значения GDP

val result = data.selectExpr("avg(GDP) as avg_GDP")

```
scala> val result = data.selectExpr("avg(GDP) as avg_GDP")
result: org.apache.spark.sql.DataFrame = [avg_GDP: double]

scala>
```

// Сохранение результата в CSV файл

result.write.option("header", "true").csv("/home/hadoop/output/avg_GDP.csv")

// Выход из Spark Shell

:q

```
scala> result.write.option("header", "true").csv("/home/hadoop/output/avg_GDP.csv")

scala> :q
hadoop@devopsvm:~$
```

Шаг 6. Работа с результатами

Переходим в директорию с результатами

cd /home/hadoop/output/avg_GDP.csv

Переименование файла с результатами

mv part-00000-*.csv avg_GDP.csv

Загрузка результатов в HDFS

hdfs dfs -put /home/hadoop/output/avg_GDP.csv/avg_GDP.csv /user01/hadoop/input/

Проверка загрузки

hdfs dfs -ls /user01/hadoop/input/

```
hadoop@devopsvm:~$ cd /home/hadoop/output/avg_GDP.csv
hadoop@devopsvm:~/output/avg_GDP.csv$ mv part-00000-*.csv avg_GDP.csv
hadoop@devopsvm:~/output/avg_GDP.csv$ hdfs dfs -put /home/hadoop/output/avg_GDP.csv/avg_GDP.csv /u
ser01/hadoop/input/
2025-03-27 00:15:46,454 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
hadoop@devopsvm:~/output/avg_GDP.csv$ hdfs dfs -ls /user01/hadoop/input/
2025-03-27 00:16:09,693 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 hadoop supergroup         27 2025-03-27 00:15 /user01/hadoop/input/avg_GDP.csv
hadoop@devopsvm:~/output/avg_GDP.csv$
```

Шаг 7. Завершение работы с Hadoop

stop-yarn.sh stop-dfs.sh

Для полной остановки всех Hadoop-демонов:

stop-all.sh

Проверка остановки всех процессов:

jps

```
hadoop@devopsvm:~/output/avg_GDP.csv$ stop-yarn.sh stop-dfs.sh
Stopping nodemanagers
Stopping resourcemanager
hadoop@devopsvm:~/output/avg_GDP.csv$ stop-all.sh
WARNING: Stopping all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: Use CTRL-C to abort.
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [devopsvm]
2025-03-27 00:17:54,142 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
Stopping nodemanagers
Stopping resourcemanager
hadoop@devopsvm:~/output/avg_GDP.csv$ jps
10228 Jps
hadoop@devopsvm:~/output/avg_GDP.csv$
```

Задание для самостоятельной работы

1. Загрузите данные по акциям другой компании.

2. Выполните аналогичный анализ для новых данных.

3. Сравните результаты анализа двух компаний.

4. Напишите Spark-приложение, которое находит дни с максимальным объемом торгов для обеих компаний.

Постановка задачи

Проанализировать экономические данные, содержащиеся в вашем файле, который

находится в файловой системе Hadoop (HDFS). Задача заключается в извлечении, обработке, и

анализе данных с целью выявления закономерностей, тенденций, и создания визуализаций на

основе предоставленных данных.

Действия, которые требуется выполнить:

1. Подключение к Hadoop и загрузка данных.

- Подключиться к HDFS и убедиться, что файл доступен по пути

hdfs://localhost:9000/user01/hadoop/economic_data/ВАШ_ФАЙЛ.csv

- Использовать PySpark или Pandas для загрузки данных из HDFS в DataFrame, который можно будет использовать для анализа.

2. Исследование и очистка данных.

- Проверить структуру данных и типы столбцов (например, с помощью printSchema() для

PySpark или describe() для Pandas).

- Убедиться, что все данные корректны, и преобразовать необходимые столбцы в числовые форматы, если они изначально представлены в виде строк.

- Проверить данные на наличие пропущенных или некорректных значений, удалить или заполнить такие значения в зависимости от ситуации.

3. Анализ данных.

- Провести базовый статистический анализ данных:

- Вычислить средние значения, медианы, минимумы и максимумы для экономических параметров.

- Проанализировать и выявить тенденции.

- Построить временные ряды, чтобы понять, как изменялась их экономика с течением времени.

4. Визуализация данных.

- Построить графики (например, графики временных рядов).

- Построить диаграммы для сравнения экономических показателей.

5. Сохранение и экспорт результатов.

- Сохранить результаты анализа и визуализации в формате CSV или изображений.

- Сохранить обработанные данные (например, данные только для отдельных стран) обратно в HDFS, чтобы другие команды могли использовать их для дальнейшего анализа.

- Создать отчет, включающий ключевые выводы и визуализации, для представления результатов анализа заинтересованным сторонам.

6. Автоматизация процесса (опционально).

- Создать скрипт или Jupyter Notebook, который автоматизирует процесс загрузки, анализа и визуализации данных для упрощения дальнейших исследований и повторного использования кода.

Вариант 23

Трансформация данных:

Вычисление статистических параметров и месячная агрегация

Данные для анализа:

Данные по акциям X5 Retail Group (FIVE): Finam (https://www.finam.ru/profile/mo ex-akcii/x5-retailgroup/export/), MOEX (https://www.moex.com/ ru/issue.aspx?board=TQBR&code=FIVE)

Операции анализа:

Фильтрация данных за последние 2 года, расчет средней цены закрытия, группировка по месяцам

1. Переходим на пользователя «hadoop»

```
devops@devopsvm:~$ sudo su - hadoop
[sudo] password for devops:
hadoop@devopsvm:~$
```

Подключаемся к hadoop и проверяем:

```
hadoop@devopsvm:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [devopsvm]
2025-03-28 15:58:15,933 WARN util.NativeCodeLoader: Unable to load native-hadoop librar
y for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$ jps
3504 NameNode
3681 DataNode
4119 Jps
3900 SecondaryNameNode
hadoop@devopsvm:~$
```

2. Создаем директорию «economic_data» в HDFS

```
hadoop@devopsvm:~$ hdfs dfs -mkdir -p /user01/hadoop/economic_data
2025-03-28 16:00:48,554 WARN util.NativeCodeLoader: Unable to load native-hadoop librar
y for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$
```

3. Загружаем данные по акциям X5 Retail Group и обрабатываем с помощью spark

Требуемые данные – X5_230401_250401.csv

```
hadoop@devopsvm:~$ ls
GDP.csv  GDP.csv.1  hadoop-3.3.5.tar.gz  hdfs  output  snap  spark-3.4.3-bin-hadoop3.tgz  X5_230401_250401.csv
```

Загрузка данных из HDFS с правильным URI:

```
hadoop@devopsvm:~$ spark-shell
25/04/05 12:25:56 WARN Utils: Your hostname, devopsvm resolves to a loopback address: 127.0.1.1; using 192.168.
 instead (on interface enp0s3)
25/04/05 12:25:56 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/04/05 12:26:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
va classes where applicable
Spark context Web UI available at http://192.168.1.12:4040
Spark context available as 'sc' (master = local[*], app id = local-1743845174201).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.3
      /_/

Using Scala version 2.12.17 (OpenJDK 64-Bit Server VM, Java 11.0.26)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val data = spark.read.option("header", "true").csv("file:///home/hadoop/X5_230401_250401.csv")
data: org.apache.spark.sql.DataFrame = [<TICKER>: string, <PER>: string ... 7 more fields]

scala>
```

JupyterLab

Проверка схемы данных

data.printSchema()

```
scala> data.printSchema()
root
 |-- <TICKER>: string (nullable = true)
 |-- <PER>: string (nullable = true)
 |-- <DATE>: string (nullable = true)
 |-- <TIME>: string (nullable = true)
 |-- <OPEN>: string (nullable = true)
 |-- <HIGH>: string (nullable = true)
 |-- <LOW>: string (nullable = true)
 |-- <CLOSE>: string (nullable = true)
 |-- <VOL>: string (nullable = true)
```

Видно, что цена закрытия «CLOSE» представлена в строчном формате, требуется изменить его на формат double, для этого выполним следующие команды:

```
scala> val dataWithDoubleCLose = data.withColumn("<CLOSE>", col("`<CLOSE>`").cast("double"))
dataWithDoubleCLose: org.apache.spark.sql.DataFrame = [<TICKER>: string, <PER>: string ... 7 more fields]
```

Так как в исходной схеме изменять формат нельзя, создаем новую схему с измененным форматом

Проверяем новую схему:

```
scala> dataWithDoubleCLose.printSchema()
root
 |-- <TICKER>: string (nullable = true)
 |-- <PER>: string (nullable = true)
 |-- <DATE>: string (nullable = true)
 |-- <TIME>: string (nullable = true)
 |-- <OPEN>: string (nullable = true)
 |-- <HIGH>: string (nullable = true)
 |-- <LOW>: string (nullable = true)
 |-- <CLOSE>: double (nullable = true)
 |-- <VOL>: string (nullable = true)
```

Видим, что формат изменился на «double»

Расчет средней цены закрытия:

```
scala> val result = dataWithDoubleCLose.selectExpr("avg(`<CLOSE>`) as avg_Close")
result: org.apache.spark.sql.DataFrame = [avg_Close: double]
```

Сохранение результата в CSV файл и выход из spark:

```
scala> result.write.option("header","true").csv("/home/hadoop/economic_data/avg_Close.csv")

scala> :q
hadoop@devopsvm:~$
```

4. Работа с результатами

Переходим в директорию с результатами

cd /home/hadoop/ economic_data /avg_Close.csv

Переименование файла с результатами

mv part-00000-*.csv avg_Close.csv

Загрузка результатов в HDFS

hdfs dfs -put /home/hadoop/economic_data/avg_Close.csv/avg_Close.csv /user01/hadoop/economic_data/

Проверка загрузки

hdfs dfs -ls /user01/hadoop/economic_data/

```
hadoop@devopsvm: $ cd /home/hadoop/economic_data/avg_Close.csv
hadoop@devopsvm:~/economic_data/avg_Close.csv$ mv part-00000-*.csv avg_Close.csv
hadoop@devopsvm:~/economic_data/avg_Close.csv$ hdfs dfs -put /home/hadoop/economic_data/avg_Close.csv/avg_Close.csv /user01/hadoop
/economic_data/
2025-04-05 12:42:10,303 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java c
lasses where applicable
hadoop@devopsvm:~/economic_data/avg_Close.csv$ hdfs dfs -ls /user01/hadoop/economic_data/
2025-04-05 12:42:44,148 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java c
lasses where applicable
Found 1 items
-rw-r--r--   1 hadoop supergroup         28 2025-04-05 12:42 /user01/hadoop/economic_data/avg_Close.csv
hadoop@devopsvm:~/economic_data/avg_Close.csv$
```

Смотрим вычисленное значение:

```
hadoop@devopsvm:~/economic_data/avg_Close.csv$ cat avg_Close.csv
avg_Close
2470.705882352941
hadoop@devopsvm:~/economic_data/avg_Close.csv$
```

Выполнили основное задание по вариантам, приступаю к следующему:

1. Исследование и очистка данных.

- Проверить структуру данных и типы столбцов (например, с помощью printSchema()

- Убедиться, что все данные корректны, и преобразовать необходимые столбцы в числовые форматы, если они изначально представлены в виде строк.

```
scala> data.printSchema()
root
 |-- <TICKER>: string (nullable = true)
 |-- <PER>: string (nullable = true)
 |-- <DATE>: string (nullable = true)
 |-- <TIME>: string (nullable = true)
 |-- <OPEN>: string (nullable = true)
 |-- <HIGH>: string (nullable = true)
 |-- <LOW>: string (nullable = true)
 |-- <CLOSE>: string (nullable = true)
 |-- <VOL>: string (nullable = true)
```

В моем случае это будут:
<OPEN>, <HIGH>, <LOW>, <CLOSE>

- Эти поля представляют цены открытия, максимума, минимума и закрытия акций. (Формат «double»)

<VOL>

- Это объем торгов акцией (количество акций, проданных за определенный период). (Формат «long»

Меняем и проверяем:

```
scala> val transformedData = data.withColumn("<OPEN>", $"<OPEN>".cast("double")).withColumn("<HIGH>", $"<HIGH>".cast("dou
")).withColumn("<LOW>", $"<LOW>".cast("double")).withColumn("<CLOSE>", $"<CLOSE>".cast("double")).withColumn("<VOL>", $"<
>".cast("long"))
transformedData: org.apache.spark.sql.DataFrame = [<TICKER>: string, <PER>: string ... 7 more fields]

scala> transformedData.printSchema()
root
 |-- <TICKER>: string (nullable = true)
 |-- <PER>: string (nullable = true)
 |-- <DATE>: string (nullable = true)
 |-- <TIME>: string (nullable = true)
 |-- <OPEN>: double (nullable = true)
 |-- <HIGH>: double (nullable = true)
 |-- <LOW>: double (nullable = true)
 |-- <CLOSE>: double (nullable = true)
 |-- <VOL>: long (nullable = true)


scala>
```

Делаем те же действия, что и раньше, но в этот раз заменим уже все типы данных переменны, которые должны быть представлены в числовом формате

2. Анализ данных.

- Провести базовый статистический анализ данных:

С помощью метода «describe» вычислим средние значения, минимум и максимум. Показать результаты с помощью метода «show».

```
scala> val stats = data.describe("<OPEN>", "<HIGH>", "<LOW>", "<CLOSE>", "<VOL>")
stats: org.apache.spark.sql.DataFrame = [summary: string, <OPEN>: string ... 4 more fields]

scala> stats.show()
25/04/06 23:31:57 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be
adjusted by setting 'spark.sql.debug.maxToStringFields'.
+-------+-----------------+-----------------+-----------------+-----------------+--------------------+
|summary|           <OPEN>|           <HIGH>|            <LOW>|          <CLOSE>|               <VOL>|
+-------+-----------------+-----------------+-----------------+-----------------+--------------------+
|  count|               17|               17|               17|               17|                  17|
|   mean|2366.205882352941|2609.9411764705883|2233.0882352941176|2470.705882352941|1.1513384588235294E7|
| stddev| 657.87154565936| 715.1533245210635| 616.7592529528367|643.1677478218536|  9037943.529789735|
|    min|             1435|           1559.5|           1351.5|           1495.5|            12350817|
|    max|             3508|             3764|             3191|           3507.5|             9209947|
+-------+-----------------+-----------------+-----------------+-----------------+--------------------+
```

- Проанализировать и выявить тенденции.

Чтобы проанализировать тенденции, можно рассчитать изменение цен (**<CLOSE> - <OPEN>**) и другие показатели. Например, вычислим дневную доходность "DailyReturn" и тренд изменения объема торгов "VolumeTrend":

Дневная доходность **DailyReturn** - рассчитывается как **(CLOSE - OPEN) / OPEN.**

VolumeTrend - Определяет, увеличивается или уменьшается объем торгов по сравнению с предыдущим днем.

```
scala> val trends = data.withColumn("DailyReturn", ($"<CLOSE>" - $"<OPEN>") / $"<OPEN>").withColumn("VolumeTrend", when($"<V
OL>" > lag($"<VOL>", 1).over(Window.orderBy($"<DATE>")), "Increase").otherwise("Decrease"))
trends: org.apache.spark.sql.DataFrame = [<TICKER>: string, <PER>: string ... 9 more fields]

scala> trends.select("DailyReturn", "VolumeTrend").show()
25/04/06 23:47:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this ca
n cause serious performance degradation.
25/04/06 23:47:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this ca
n cause serious performance degradation.
25/04/06 23:47:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this ca
n cause serious performance degradation.
25/04/06 23:47:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this ca
n cause serious performance degradation.
25/04/06 23:47:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this ca
n cause serious performance degradation.
+--------------------+-----------+
|         DailyReturn|VolumeTrend|
+--------------------+-----------+
| 0.08013937282229965|   Decrease|
|-0.04349216501439...|   Decrease|
| 0.00766922307435812|   Increase|
|  0.4009917355371901|   Decrease|
| 0.09404315196998124|   Decrease|
|-0.04243463351907415|   Increase|
|  0.0165585142089953|   Decrease|
|-0.06621205455345358|   Decrease|
| 0.03839811542991755|   Increase|
| 0.07647058823529412|   Decrease|
|  0.0932327676513723|   Increase|
|  0.1827051846183279|   Decrease|
| -0.1086333227142402|   Increase|
|  0.2017103550845882|   Decrease|
|0.031080872119993816|   Decrease|
|0.050149700598802395|   Increase|
|-0.08452109464082098|   Increase|
+--------------------+-----------+
```

- Построить временные ряды, чтобы понять, как изменялась их экономика с течением времени.

Группируем данные по дате и вычисляет среднюю цену закрытия (AvgClose) и общий объем торгов (TotalVolume):

```
scala> val timeSeries = data.groupBy($"<DATE>").agg(avg($"<CLOSE>").as("AvgClose"), sum($"<VOL>").as("TotalVolume")).orderBy
($"<DATE>")
timeSeries: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [<DATE>: string, AvgClose: double ... 1 more field]

scala> timeSeries.show()
+------+--------+-----------+
|<DATE>|AvgClose|TotalVolume|
+------+--------+-----------+
|230401|  1550.0|  3327286.0|
|230501|  1495.5|  3104908.0|
|230601|  1511.0|  3340214.0|
|230701|  2119.0|1.8990956E7|
|230801|  2332.5|1.3804526E7|
|230901|  2234.0|  8818341.0|
|231001|  2271.5|  5645380.0|
|231101|  2122.5|  4162343.0|
|231201|  2204.0|  8328631.0|
|240101|  2379.0|  6869968.0|
|240201|  2609.0|  8152619.0|
|240301|  3091.0|1.2350817E7|
|240401|  2798.0|  9209947.0|
|250101|  3232.0|3.1895776E7|
|250201|  3334.0| 2.423743E7|
|250301|  3507.5|2.8243449E7|
|250401|  3211.5|  5244947.0|
+------+--------+-----------+


scala> 
```

3. Визуализация данных.

- Построить графики (например, графики временных рядов).

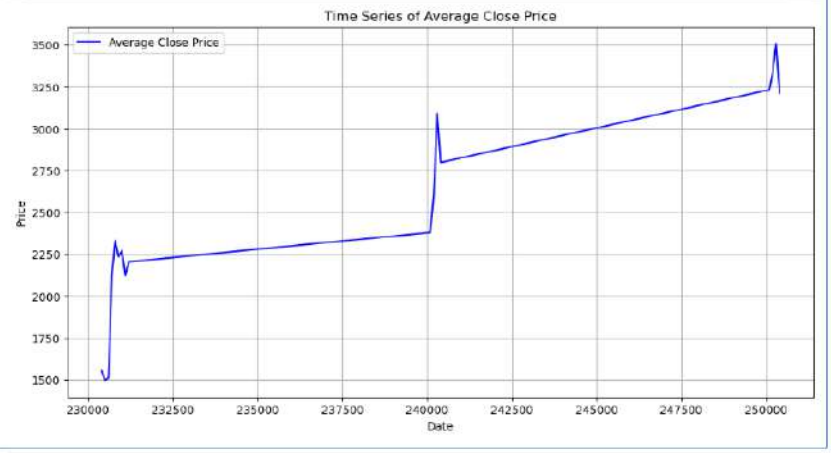
[5]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[6]:
```
# Загрузка данных из CSV
data = pd.read_csv('X5_230401_250401.csv')
```

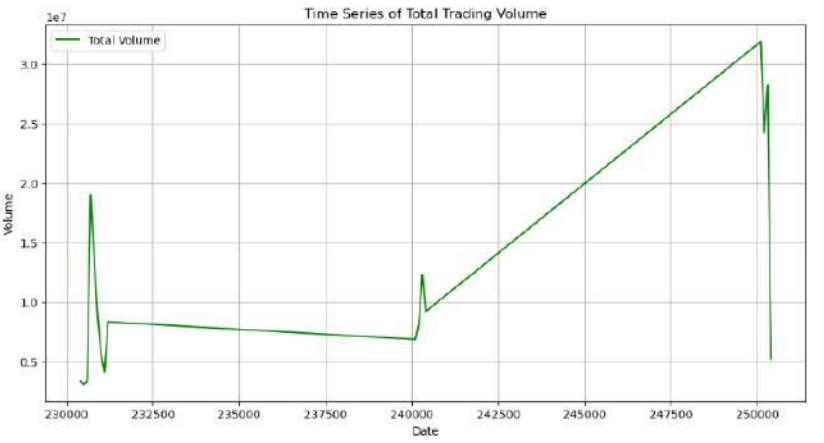
[7]:
```
# Создание временных рядов
time_series = data.groupby('<DATE>').agg({
    '<CLOSE>': 'mean',  # Средняя цена закрытия
    '<VOL>': 'sum'      # Суммарный объем торгов
}).reset_index()
```

[8]:
```
# Переименование столбцов для удобства
time_series.rename(columns={'<CLOSE>': 'AvgClose', '<VOL>': 'TotalVolume'}, inplace=True)
```

[9]:
```
# График временных рядов для средней цены закрытия
plt.figure(figsize=(12, 6))
plt.plot(time_series['<DATE>'], time_series['AvgClose'], label='Average Close Price', color='blue
plt.title('Time Series of Average Close Price')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



[10]:
```
# График временных рядов для объема торгов
plt.figure(figsize=(12, 6))
plt.plot(time_series['<DATE>'], time_series['TotalVolume'], label='Total Volume', color='green')
plt.title('Time Series of Total Trading Volume')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]: