

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

ДИСЦИПЛИНА:
Инструменты для хранения и обработки больших данных
Лабораторная работа № 5.1

Тема:
«Развертывание и настройка кластера Nadoor»

Выполнил(а):
Вереина М.С., АДЭУ-211
Преподаватель: Босенко Т.М.

Москва
2024

Цель работы:

Ознакомление с процессом установки и настройки распределенных систем, таких как Apache (Hadoop) Hadoop. Изучить основные операции и функциональные возможности системы, что позволит понять принципы работы с данными и распределенными вычислениями.

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.

Вариант 3. Установка Apache Hadoop и выполнение задачи на сортировку данных.

Данные: Исторические данные по акциям Лукойла (LKOH) с сайта Московской биржи (moex.com).

Операции: Фильтрация данных за последние 5 лет, расчет минимальной цены закрытия, группировка по годам.


Ход работы

1. Запуск Hadoop и загрузка файла

Шаг 1. Запуск Hadoop.

1) Запуск надстройки над основной файловой системой.

2) Запуск службы YARN, необходимой для управления и контроля за затрачиваемыми ресурсами на выполнение задач, связанных с большими данными.



```
hadoop@devopsvm: ~  
devops@devopsvm:~$ sudo su hadoop  
[sudo] password for devops:  
hadoop@devopsvm: /home/devops$ cd  
hadoop@devopsvm:~$ start-dfs.sh  
Starting namenodes on [localhost]  
Starting datanodes  
Starting secondary namenodes [devopsvm]  
2024-11-07 21:48:19,265 WARN util.NativeCodeLoader: Unable to load native-hadoop library for y  
our platform... using builtin-java classes where applicable  
hadoop@devopsvm:~$ start-yarn.sh  
Starting resourcemanager  
Starting nodemanagers  
hadoop@devopsvm:~$
```

Рисунок 1 – Подключение к Hadoop и запуск Yarn

Шаг 2. Проверка работы Hadoop.

Воспользуемся `jps` — это утилита командной строки, которая отображает информацию о процессах виртуальной машины Java (JVM), запущенных в системе.

```
hadoop@devopsvm:~$ jps
6368 ResourceManager
6048 SecondaryNameNode
5844 DataNode
6501 NodeManager
6889 Jps
5612 NameNode
```

Рисунок 2 – Выполнение команды jps

После выполнения команды можно увидеть следующие процессы: NameNode, DataNode, SecondaryNameNode, ResourceManager, NodeManager.

В стандартной конфигурации Hadoop HDFS предоставляет веб-интерфейс, доступный через веб-браузер на порту 9870. Этот интерфейс позволяет просматривать состояние и структуру HDFS, а также выполнять некоторые операции.

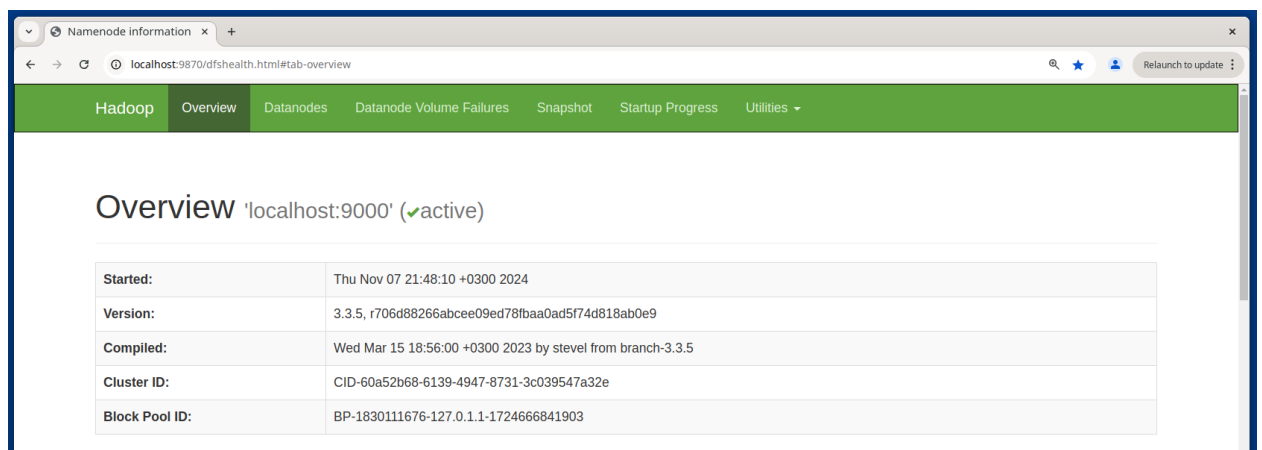


Рисунок 3 – Веб-интерфейс Hadoop в веб-браузере

На веб-интерфейсе YARN можно увидеть информацию о запущенных приложениях, статусе узлов и других метриках кластера. Это удобный способ управления и мониторинга задач.

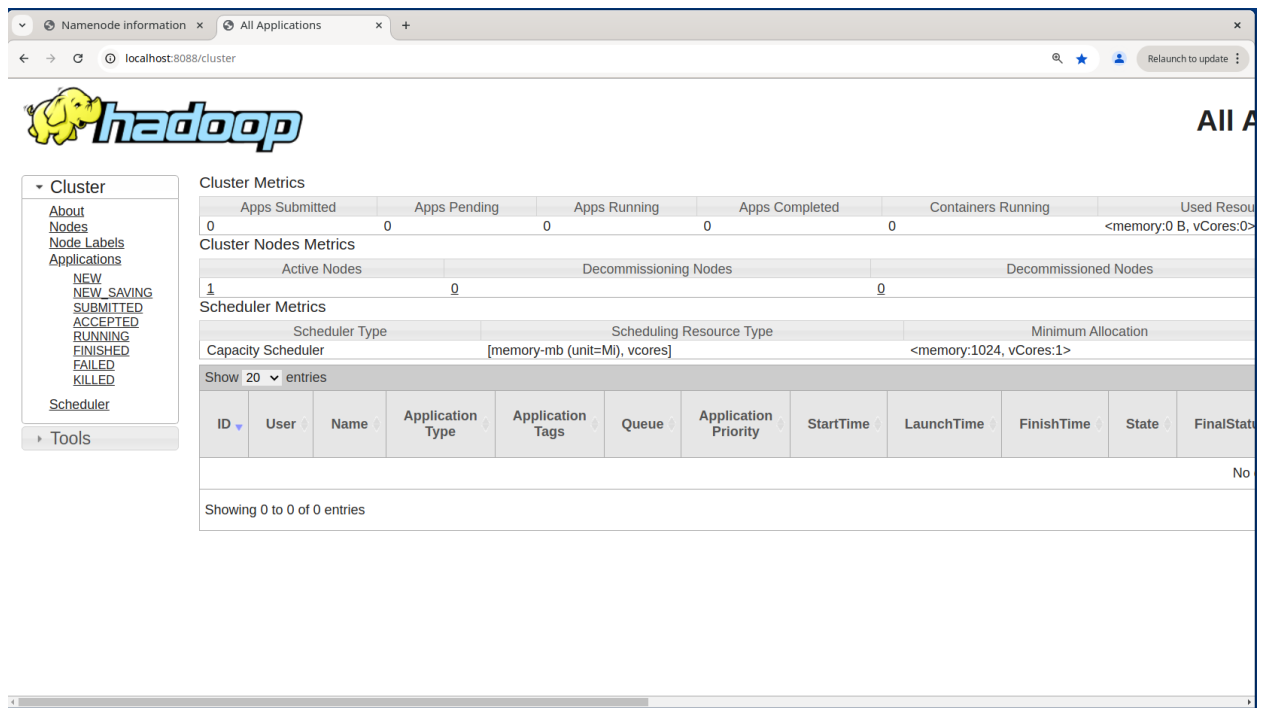


Рисунок 4 – Веб-интерфейс Yarn

Шаг 3. Работа с данными.

Необходимо создать директорию в HDFS:

```
hadoop@devopsvm:~$ hadoop fs -mkdir /vereinams
2024-11-21 01:02:56,621 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$ hadoop fs -mkdir /vereinams/hadoop
2024-11-21 01:03:42,042 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$
hadoop@devopsvm:~$ hadoop fs -mkdir /vereinams/hadoop/input
2024-11-21 01:03:58,474 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$
```

Рисунок 5 – Использование команд для создания каталога в HDFS

При успешном создании данная директория будет доступна в веб-интерфейсе Hadoop.

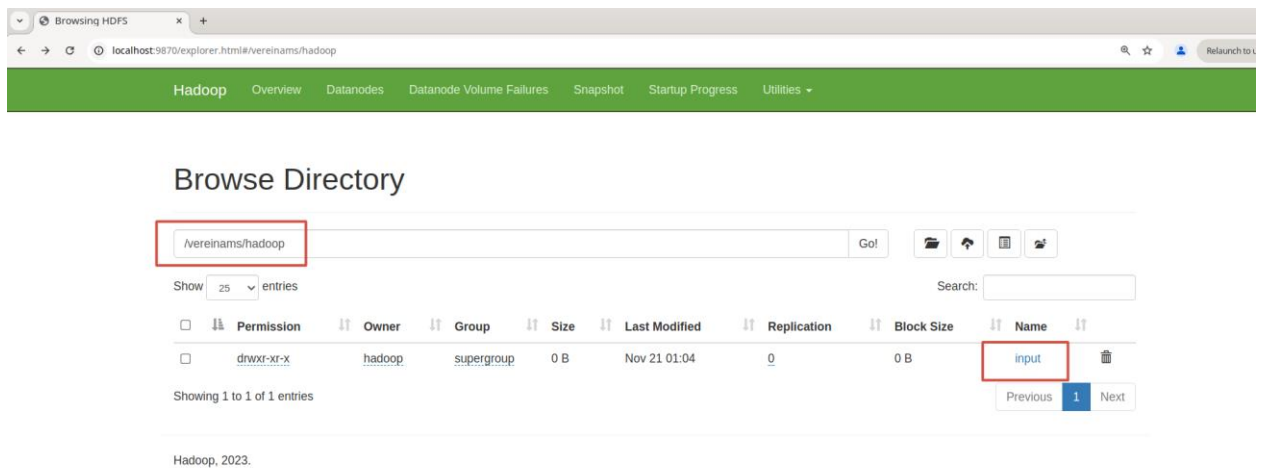


Рисунок 6 – Отображение директории в веб-интерфейсе Hadoop

Загрузка данных в файловую систему HDFS.

Для загрузки файла необходимо перейти в ранее созданную директорию и воспользоваться иконкой для загрузки.

Перед этим предоставить необходимые права доступа к созданным директориям.

```
hadoop@devopsvm:~$ hadoop fs -chmod 777 /vereinams
2024-11-21 01:11:32,195 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
hadoop@devopsvm:~$ hadoop fs -chmod 777 /vereinams/hadoop
2024-11-21 01:11:43,442 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
^[[Ahadoop@devopsvm:~$ hadoop fs -chmod 777 /vereinams/hadoop/input
2024-11-21 01:11:56,506 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
hadoop@devopsvm:~$
```

Рисунок 7 – Предоставление доступа к директориям

В рамках указанного варианта в Hadoop будут загружены 225МБ данных об изменении акций компании Лукойл.

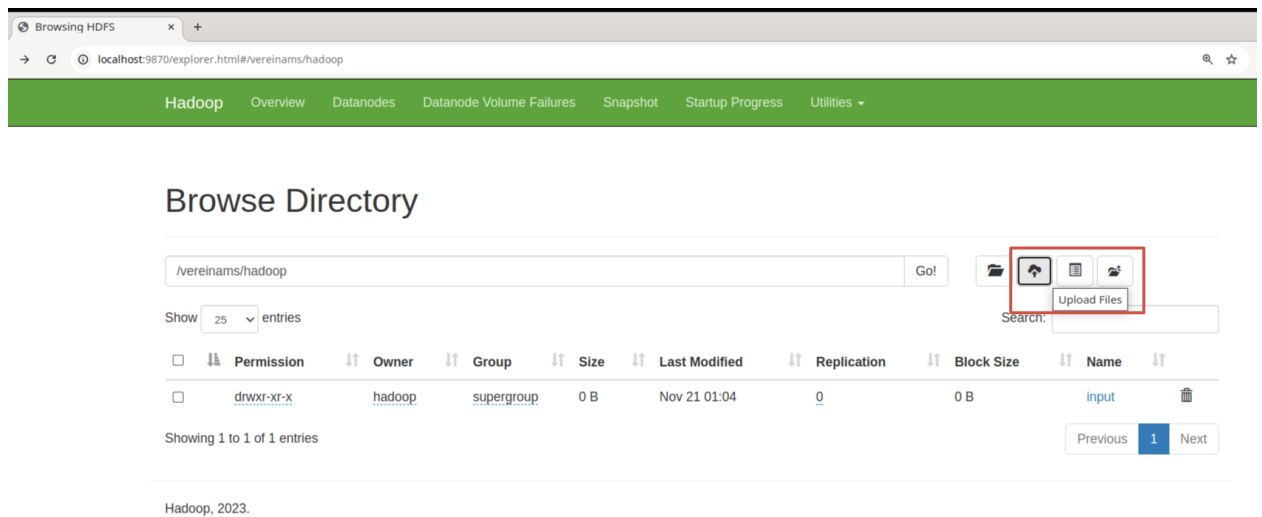


Рисунок 8 – Элемент веб-интерфейса для загрузки файла в HDFS

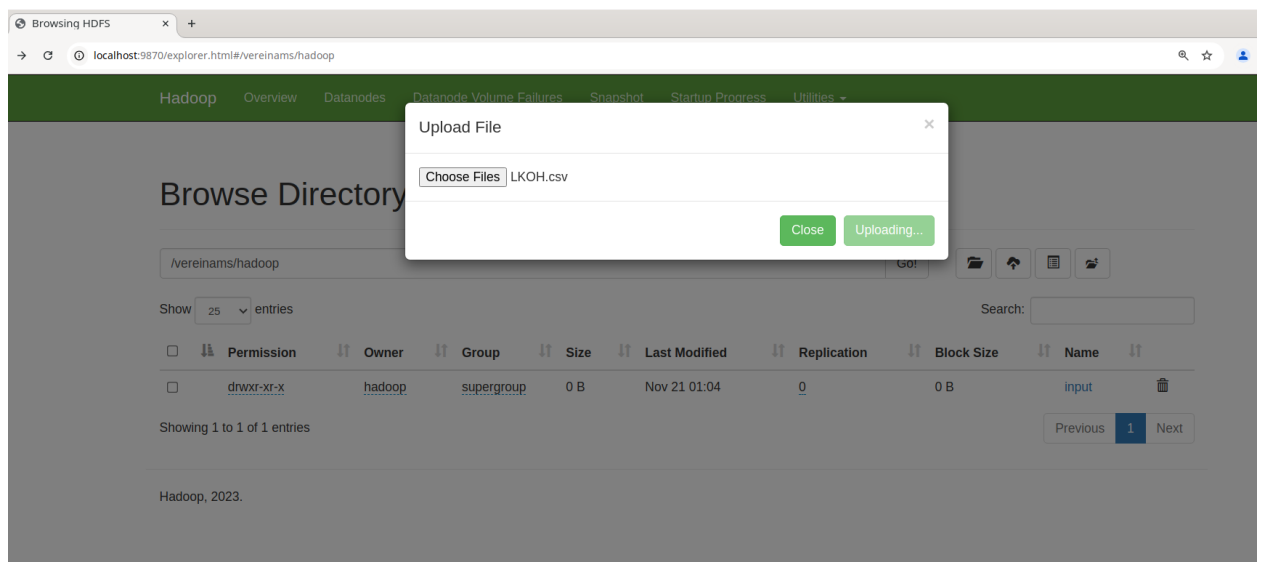


Рисунок 9 – Загрузка файла в HDFS

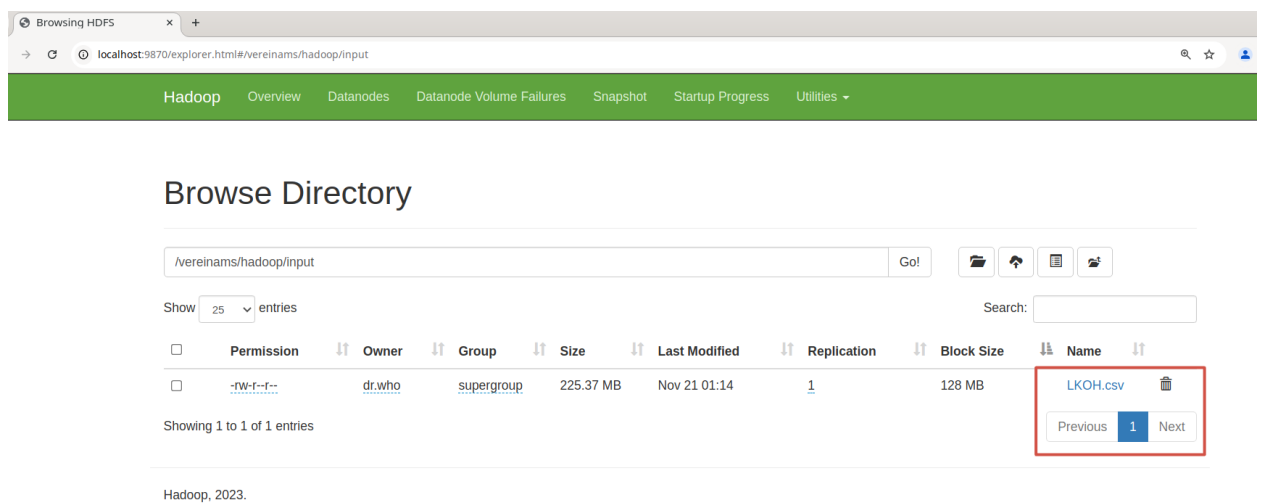


Рисунок 10 – Результат загрузки файла в HDFS

Запуск Spark

Далее необходимо запустить фреймворк Spark с помощью команды `spark-shell`, с помощью которого можно выполнять операции с загруженными данными.

```
hadoop@devopsvm:~$ spark-shell
24/11/21 01:18:21 WARN Utils: Your hostname, devopsvm resolves to a loopback address: 127.0.1.1; using 192.16
8.0.171 instead (on interface enp0s3)
24/11/21 01:18:21 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).
24/11/21 01:18:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
Spark context Web UI available at http://192.168.0.171:4040
Spark context available as 'sc' (master = local[*], app id = local-1732141115205).
Spark session available as 'spark'.
Welcome to

  ____      _
 / ___|    / \
| |  | |  / _ \
| |  | | / ___ \
| |  | || |___| \
| |  | || |___| \
| |  | || |___| \
|_|  |_| \____/

version 3.4.3

Using Scala version 2.12.17 (OpenJDK 64-Bit Server VM, Java 11.0.25)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Рисунок 11 – Запуск Spark

2. Подключение к Hadoop и загрузка данных.

Шаг 1.

Подключиться к HDFS и убедиться, что файл доступен по пути `hdfs://localhost:9000/vereinams/hadoop/input/LKOH.csv`.

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

Для этого необходимо создать `SparkSession` – это способ инициализации базовой функциональности PySpark для программного создания PySpark RDD, `DataFrame` и `Dataset`. `SparkSession` можно создать с помощью `SparkSession.builder`, который представляет собой реализацию шаблона проектирования Builder

Для того чтобы убедиться, что файл доступен по указанному пути, необходимо воспользоваться командой `spark.read.scv`, в рамках которой задать путь до файла. Если он корректен, то с файлом можно работать дальше, например, отобразить первые 5 строк.

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

[6]: from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("Lukoil Data Analysis") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://localhost:9000") \
    .config("spark.ui.port", "4050") \
    .getOrCreate()

# Установка количества разделов для shuffle операций
spark.conf.set("spark.sql.shuffle.partitions", "50")

[8]: # Чтение данных из HDFS
file_path = "hdfs://localhost:9000/vereinams/hadoop/input/LKOH.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

# Просмотр первых строк данных
df.show(5)
```

Date	Number of Transactions	Volume of Transactions (shares)	Volume of Transactions (rubles)	Max Price	Min Price	Percent Change	Number of Participants
2010-11-23	15795	5442132.515464597	5.03271638694298E9	0.0	0.0	-7.799372001012195	29
1998-10-27	860	3.0224760740093123E7	4.762930518470548E9	0.00429153647019218	0.00429153647019218	-11.710583654625012	200
2022-06-01	76820	6597478.306771004	2.749232524257618E9	0.00858307294038436	0.00858307294038436	8.368010859620327	228
2019-01-29	54886	1.718782521066418E7	7.416754297995878E9	0.01287460941057654	0.01287460941057654	-0.6765419128603583	353
2008-07-20	6265	2296277.417904647	4.177121750239943E9	0.01716614588076872	0.01716614588076872	4.645308783407199	373

only showing top 5 rows

Рисунок 12 – Создание SparkSession и отображение первых 5 строк.

3. Исследование и очистка данных.

1) Проверить структуру данных и типы столбцов (например, с помощью printSchema() для PySpark или describe() для Pandas).

```
[11]: # Проверка структуры данных и типов столбцов
df.printSchema()

root
 |-- Date: date (nullable = true)
 |-- Number of Transactions: integer (nullable = true)
 |-- Volume of Transactions (shares): double (nullable = true)
 |-- Volume of Transactions (rubles): double (nullable = true)
 |-- Max Price: double (nullable = true)
 |-- Min Price: double (nullable = true)
 |-- Percent Change: double (nullable = true)
 |-- Number of Participants: integer (nullable = true)

[18]: print(df.dtypes)

[('Date', 'date'), ('Number of Transactions', 'int'), ('Volume of Transactions (shares)', 'double'), ('Volume of Transactions (rubles)', 'double'), ('Max Price', 'double'), ('Min Price', 'double'), ('Percent Change', 'double'), ('Number of Participants', 'int')]

[ 1: ]
```

Рисунок 13 – Проверка структуры данных и типов столбцов

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

С помощью методов printSchema() и df.dtypes получилось определить структуру и типы столбцов. Все они указаны корректно:

1. **Date: date.** Столбец с датой сделок/торговых сессий. Использование типа date позволяет хранить только дату без времени.

2. **Number of Transactions: integer.** Столбец, который хранит количество транзакций. Тип integer подходит, так как количество транзакций обычно не превышает 2 миллиардов.

3. **Volume of Transactions (shares): double.** Столбец, который хранит объем транзакций в акциях. Тип double позволяет учитывать дробные значения, объем может быть нецелым.

4. **Volume of Transactions (rubles): double.** Столбец, который хранит объем транзакций в рублях. Использование типа double также корректно для хранения денежных сумм, требуется учитывать дробные значения.

5. **Max Price: double.** Столбец, который хранит максимальную цену акций за период, они могут иметь дробные значения.

6. **Min Price: double.** Столбец, который хранит минимальную цену акций за период. Аналогично предыдущему столбцу, тип double корректен.

7. **Percent Change: double.** Столбец, который хранит процентное изменение цены акций. Использование типа double оправдано, так как процентные изменения могут быть дробными.

8. **Number of Participants: integer.** Столбец, который хранит количество участников торгов. Тип integer является подходящим выбором для хранения целых чисел.

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

Для этого необходимо перевести датафрейм Spark в датафрейм Pandas, после чего проверяем наличие пропущенных нулей в каждом столбце:

```
[7]: df_pd = df.toPandas()
df_pd.head()
```

	Date	Number of Transactions	Volume of Transactions (shares)	Volume of Transactions (rubles)	Max Price	Min Price	Percent Change	Number of Participants
0	2010-11-23	15795	5.442133e+06	5.032716e+09	0.000000	0.000000	-7.799372	29
1	1998-10-27	860	3.022476e+07	4.762931e+09	0.004292	0.004292	-11.710584	200
2	2022-06-01	76820	6.597478e+06	2.749233e+09	0.008583	0.008583	8.368011	228
3	2019-01-29	54886	1.718783e+07	7.416754e+09	0.012875	0.012875	-0.676542	353
4	2008-07-20	6265	2.296277e+06	4.177122e+09	0.017166	0.017166	4.645309	373

```
[8]: # Подсчет количества пропущенных значений в каждом столбце
missing_values = df_pd.isnull().sum()
print("Количество пропущенных значений в каждом столбце:")
print(missing_values)

Количество пропущенных значений в каждом столбце:
Date                0
Number of Transactions  0
Volume of Transactions (shares)  0
Volume of Transactions (rubles)  0
Max Price           0
Min Price           0
Percent Change      0
Number of Participants  0
dtype: int64
```

Рисунок 14 – Поиск пропущенных значений

3) Анализ данных.

- Провести базовый статистический анализ данных.
- Вычислить средние значения, медианы, минимумы и максимумы для экономических параметров.

Пропущенных значений в данном случае не оказалось, поэтому можно использовать метод описательной статистики `describe()`, чтобы сделать дальнейшие выводы по корректности данных.

```
[9]: df_pd.describe()
```

	Number of Transactions	Volume of Transactions (shares)	Volume of Transactions (rubles)	Max Price	Min Price	Percent Change	Number of Participants
count	2.097152e+06	2.097152e+06	2.097152e+06	2.097152e+06	2.097152e+06	2.097152e+06	2.097152e+06
mean	5.001402e+04	1.944060e+07	4.999457e+09	4.500000e+03	4.500000e+03	-5.274294e-03	2.505453e+02
std	2.886152e+04	1.122849e+07	2.886264e+09	2.598078e+03	2.598078e+03	1.443652e+01	1.443636e+02
min	0.000000e+00	9.467930e+01	1.386966e+04	0.000000e+00	0.000000e+00	-2.499998e+01	1.000000e+00
25%	2.502500e+04	9.725092e+06	2.499802e+09	2.250000e+03	2.250000e+03	-1.250609e+01	1.250000e+02
50%	5.001350e+04	1.944261e+07	4.998798e+09	4.500000e+03	4.500000e+03	-4.421235e-03	2.510000e+02
75%	7.500200e+04	2.915813e+07	7.497931e+09	6.750000e+03	6.750000e+03	1.250248e+01	3.760000e+02
max	1.000000e+05	3.888998e+07	9.999999e+09	9.000000e+03	9.000000e+03	2.499996e+01	5.000000e+02

Рисунок 15 – Результат выполнения команды `describe()`

Приведенные значения указывают на корректность данных, как с точки зрения качества данных, так и по их наполненности. На основании этого можно сделать следующие выводы:

Количество транзакций (Number of Transactions):

- **Среднее значение:** 50 000 шт

Среднее количество транзакций указывает на активность торговли акциями Лукойла. Наличие нулевых значений лишь указывает на дни с отсутствием торговых операций.

Объем транзакций в акциях (Volume of Transactions (shares)):

- **Среднее значение:** 19 400 000 акций

Объем торгов показывает высокую активность на рынке.

Объем транзакций в рублях (Volume of Transactions (rubles)):

Высокий средний объем транзакций в рублях говорит о значительных финансовых операциях с акциями Лукойла.

Максимальная и минимальная цена акций (Max Price и Min Price):

Эти значения показывают диапазон цен за рассматриваемый период и могут быть полезны для анализа волатильности акций, в среднем составляют ~ 4500 рублей.

Процент изменения цены акций (Percent Change):

- **Среднее значение:** -5.27%

Отрицательное среднее значение указывает на общее снижение цен за рассматриваемый период.

Количество участников торгов (Number of Participants):

- **Среднее значение:** $2.51 \times 10^{22} \times 10^2$ (или около 251 участника)

Данные показывают высокую активность торговли акциями Лукойла с большим объемом транзакций как в количестве акций, так и в денежном выражении. Наблюдается высокая волатильность цен с отрицательным средним изменением цены за рассматриваемый период. Несмотря на наличие дней без торговых операций, общая тенденция указывает на активность и интерес к акциям компании.

4. Основное задание:

Установка Apache Hadoop и выполнение задачи на сортировку данных.

Данные: Исторические данные по акциям Лукойла (ЛКОН) с сайта Московской биржи (moex.com).

Операции: Фильтрация данных за последние 5 лет, расчет минимальной цены закрытия, группировка по годам.

Исходя из того, что при просмотре первых 5 значений датафрейма, данные находились вразброс, необходимо применить к ним сортировку для корректного отображения, в данном случае по дате, так как данные представляют собой исторические значения акций с помощью метода `sort_values()`.

```
[13]: # Сортировка данных по дате
df_sort = df_pd.sort_values(by = 'Date')
df_sort
```

	Date	Number of Transactions	Volume of Transactions (shares)	Volume of Transactions (rubles)	Max Price	Min Price	Percent Change	Number of Participants
1850446	1997-01-01	60035	1.566573e+06	9.286410e+08	7941.256495	7941.256495	10.208875	272
1060787	1997-01-01	24599	3.175778e+07	5.776579e+09	4552.406098	4552.406098	9.931988	306
1287007	1997-01-01	66566	1.996447e+07	2.479943e+09	5523.237478	5523.237478	-9.011868	280
289684	1997-01-01	44826	1.768331e+07	7.390749e+09	1243.189451	1243.189451	-24.147486	311
202010	1997-01-01	42882	8.373430e+06	7.269332e+09	866.933282	866.933282	6.506255	48
...
705820	2024-11-01	27559	1.564816e+07	4.782930e+09	3029.052271	3029.052271	-4.269346	368
810020	2024-11-01	89805	2.769503e+07	1.366661e+08	3476.230372	3476.230372	2.008147	268
1006936	2024-11-01	73009	1.048973e+07	9.958435e+09	4321.302567	4321.302567	-22.481559	304
1947235	2024-11-01	36732	9.767120e+06	9.285064e+09	8356.630019	8356.630019	3.275401	171
717799	2024-11-01	80920	2.031081e+07	1.293338e+09	3080.460587	3080.460587	7.923122	157

2097152 rows x 8 columns

Рисунок 16 – применение сортировки к данным

Далее работаем с фильтрацией, необходимо отфильтровать данные за последние 5 лет. Для этого используется метод `DateOffset`. Data offset в Python используется для смещения данных с целью сравнения изменений во времени или создания производных

признаков. Это применяется в финансовом анализе, прогнозировании временных рядов и при подготовке данных для алгоритмов машинного обучения.

```
[14]: import pandas as pd
      from pandas.tseries.offsets import DateOffset

[18]: df_sort['Date'] = pd.to_datetime(df_sort['Date'])

      #Определение последней даты в столбце 'Date'
      last_date = df_sort['Date'].max()

      # Фильтрация данных за последние 5 лет
      five_years_ago = last_date - DateOffset(years=5)
      df_filter = df_sort[df_sort['Date'] >= five_years_ago]

      # Вывод отфильтрованных данных
      df_filter
```

	Date	Number of Transactions	Volume of Transactions (shares)	Volume of Transactions (rubles)	Max Price	Min Price	Percent Change	Number of Participants
1826033	2019-11-01	92031	1.741402e+07	9.549088e+09	7836.487215	7836.487215	-0.084452	332
1822104	2019-11-01	47867	3.483438e+07	3.481910e+09	7819.625768	7819.625768	10.126904	260
295526	2019-11-01	74152	2.429457e+07	9.834557e+09	1268.260607	1268.260607	-12.700783	313
1840517	2019-11-01	4130	1.051023e+07	7.644111e+09	7898.645830	7898.645830	-8.713156	248
26319	2019-11-01	82283	2.143909e+07	5.088589e+09	112.948948	112.948948	8.430726	102
...
705820	2024-11-01	27559	1.564816e+07	4.782930e+09	3029.052271	3029.052271	-4.269346	368
810020	2024-11-01	89805	2.769503e+07	1.366661e+08	3476.230372	3476.230372	2.008147	268
1006936	2024-11-01	73009	1.048973e+07	9.958435e+09	4321.302567	4321.302567	-22.481559	304
1947235	2024-11-01	36732	9.767120e+06	9.285064e+09	8356.630019	8356.630019	3.275401	171
717799	2024-11-01	80920	2.031081e+07	1.293338e+09	3080.460587	3080.460587	7.923122	157

377336 rows x 8 columns

Рисунок 17 – Применение фильтрации по датам

После применения фильтрации данные начинают отображаться с 01.11.2019 – что является датой ровно 5 лет назад.

Далее для того, чтобы вывести минимальную цену закрытия и дату, в которую оно произошло, необходимо воспользоваться встроенной функцией `min()` к минимальным ценам закрытия. Далее по индексу минимальной цены необходимо вывести соответствующую ему дату с помощью функции `loc()`.

```
[22]: # Расчет минимальной цены закрытия и соответствующей даты

      min_price = df_filter['Min Price'].min() # Минимальная цена закрытия
      min_date = df_filter.loc[df_filter['Min Price'].idxmin(), 'Date'] # Дата минимальной цены

      print(f"Минимальная цена закрытия за последние 5 лет: {min_price}")
      print(f"Дата минимальной цены закрытия: {min_date.date()}")

      Минимальная цена закрытия за последние 5 лет: 0.00858307294038436
      Дата минимальной цены закрытия: 2022-06-01
```

[]:

[]:

Рисунок 18 – Определение минимальной цены закрытия

Далее для того, чтобы сгруппировать исторические данные акций по годам, необходимо в первую очередь создать столбец Года, который будет заполняться по данным из столбца Даты. Далее необходимо для каждого используемого в группировке столбца

определить соответствующее агрегированное значение. Для денежных данных, а также для процентных колебаний необходимо использовать вычисление среднего mean, для количественных показателей, таких как объем и количество транзакций, а также их количество необходимо использовать вычисление суммы sum.

Сама группировка осуществляется посредством использования метода groupby().

```
[25]: # Добавление столбца с годом
df_sort['Year'] = df_sort['Date'].dt.year

# Группировка данных по годам и вычисление агрегированных значений
df_group = df_sort.groupby('Year').agg({
    'Max Price': 'mean',
    'Min Price': 'mean',
    'Percent Change': 'mean',
    'Volume of Transactions (shares)': 'sum',
    'Volume of Transactions (rubles)': 'sum',
    'Number of Transactions': 'sum',
    'Number of Participants': 'sum'
}).reset_index()

# Вывод сгруппированных данных
df_group
```

	Year	Max Price	Min Price	Percent Change	Volume of Transactions (shares)	Volume of Transactions (rubles)	Number of Transactions	Number of Participants
0	1997	4502.422642	4502.422642	0.017063	1.458816e+12	3.754760e+14	3756348946	18858422
1	1998	4501.919973	4501.919973	-0.053180	1.466237e+12	3.770386e+14	3763351158	18886710
2	1999	4497.162497	4497.162497	-0.027472	1.467540e+12	3.764453e+14	3783518642	18869270
3	2000	4498.297501	4498.297501	-0.041945	1.480071e+12	3.815847e+14	3808919405	19039400
4	2001	4499.268370	4499.268370	-0.043357	1.455909e+12	3.747673e+14	3758559389	18778448
5	2002	4495.581329	4495.581329	0.015471	1.459859e+12	3.766974e+14	3758018242	18878263
6	2003	4501.019552	4501.019552	0.009664	1.461490e+12	3.747500e+14	3751676308	18798307
7	2004	4496.273077	4496.273077	-0.040936	1.477159e+12	3.791449e+14	3795060954	19034042
8	2005	4490.947241	4490.947241	0.009804	1.459750e+12	3.740299e+14	3748322309	18845229
9	2006	4493.647351	4493.647351	0.028786	1.463214e+12	3.764619e+14	3756467918	18832796
10	2007	4488.011367	4488.011367	-0.048371	1.457283e+12	3.745353e+14	3752596332	18802852

Рисунок 19 – Результат применения группировки к данным

Также по данным, полученным из Nadoor, можно строить графики, однако воспроизводиться они будут очень медленно.

```
# Создание фигуры для графиков
plt.figure(figsize=(14, 10))

# График количества транзакций
plt.subplot(3, 2, 1)
sns.lineplot(data=df_sort, x='Date', y='Number of Transactions', color='blue')
plt.title('Количество транзакций по времени')
plt.xlabel('Дата')
plt.ylabel('Количество транзакций')

# Настройка общего оформления графиков
plt.tight_layout()
plt.show()
```



Рисунок 20 – Визуализация данных

Также важно уметь сохранять измененные данные обратно в Hadoop, это можно сделать с помощью `write.mode()`.

The image shows two parts of a Hadoop environment. The top part is the 'Browse Directory' web interface, displaying the path `/vereinams/hadoop/input`. It shows a table of files with columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. Two files are listed: `LKOH.csv` (225.37 MB, Nov 21 01:14) and `LKOH_filter.csv` (0 B, Nov 22 01:37). The bottom part is a terminal window showing a Jupyter Notebook cell with the following code:

```
[20]: # Преобразование DataFrame Pandas в DataFrame Spark
spark_df = spark.createDataFrame(df_filter)

# Запись DataFrame в HDFS
spark_df.write.mode('overwrite').csv('hdfs://localhost:9000/vereinams/hadoop/input/LKOH_filter.csv')

# Завершение работы сессии Spark
spark.stop()
```

Below the code, a warning message is displayed: `24/11/22 01:37:49 WARN TaskSetManager: Stage 4 contains a task of very large size (4622 KiB). The maximum recommended task size is 1000 KiB.`

Рисунок 21 – Сохранение измененных данных в Hadoop

Выводы: в ходе выполнения данной лабораторной работы был изучен процесс установки и настройки распределенных систем, таких как Apache Hadoop. Изучены основные операции и функциональные возможности системы, что позволило понять принципы работы с данными и распределенными вычислениями.

Hadoop вместе со Spark и Yarn позволяет работать с большими данными гораздо эффективнее за счет следующих параметров:

1. **Скорость обработки:**

Spark значительно быстрее Hadoop MapReduce благодаря обработке данных в памяти, что делает его идеальным для интерактивных и итеративных задач.

2. Универсальность:

Spark поддерживает различные типы обработки данных: пакетную, потоковую, SQL-запросы и машинное обучение, в то время как Hadoop в основном ориентирован на пакетную обработку через MapReduce.

3. Интеграция:

Spark может работать с данными, хранящимися в HDFS, что позволяет использовать Hadoop как систему хранения. Это обеспечивает масштабируемость и долговременное хранение данных.

4. Управление ресурсами:

YARN (Yet Another Resource Negotiator) позволяет эффективно распределять ресурсы между Hadoop и Spark, что обеспечивает оптимальное использование вычислительных мощностей.

5. Обработка больших данных:

Использование Spark вместе с Hadoop позволяет обрабатывать большие объемы данных более эффективно, что особенно полезно для аналитических приложений и задач, требующих высокой скорости обработки.

6. Отказоустойчивость:

Оба инструмента обеспечивают отказоустойчивость: Hadoop записывает данные на диск после каждой операции, а Spark использует концепцию RDD (Resilient Distributed Dataset) для восстановления данных в случае сбоя.

В ходе проделанной работы на экспериментах с выполнением операций с DataFrame Spark и DataFrame Pandas первый показал лучшие результаты. При работе со вторым датафреймом виртуальная машина не справлялась, в связи с чем приходилось несколько раз перезагружать ее, теряя текущий результат. Spark же работает более стабильно и эффективно.