

Департамент образования города Москвы

Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»

Институт цифрового образования
Департамент информатики, управления и технологий

Лабораторная работа 5-1
по дисциплине «Инструменты для хранения и обработки больших
данных»

Тема: «Развертывание и настройка кластера Hadoop»

Направление подготовки 38.03.05 – бизнес-информатика
Профиль подготовки «Аналитика данных и эффективное управление»
(очная форма обучения)

Выполнила:
Студентка группы АДЭУ-211
Белик Мария Константиновна

Преподаватель:
Босенко Т.М.

Москва
2024

ВВЕДЕНИЕ

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

Необходимые ПО:

- Ubuntu 24.04 LTS (22.04, 20.04) или новее;
- Java 8 ил Java11 или новее;
- Apache Spark 3.4.3;
- Python 3.12;
- pip (менеджер пакетов Python).

Вариант заданий 1:

Установка Apache Hadoop на одном узле и выполнение простой задачи на подсчет строк в файле.

Данные: Исторические данные по акциям Сбербанка (SBER) с сайта Московской биржи (moex.com)

Операции: Фильтрация данных за 2020 год, расчет средней цены закрытия, группировка по месяцам.

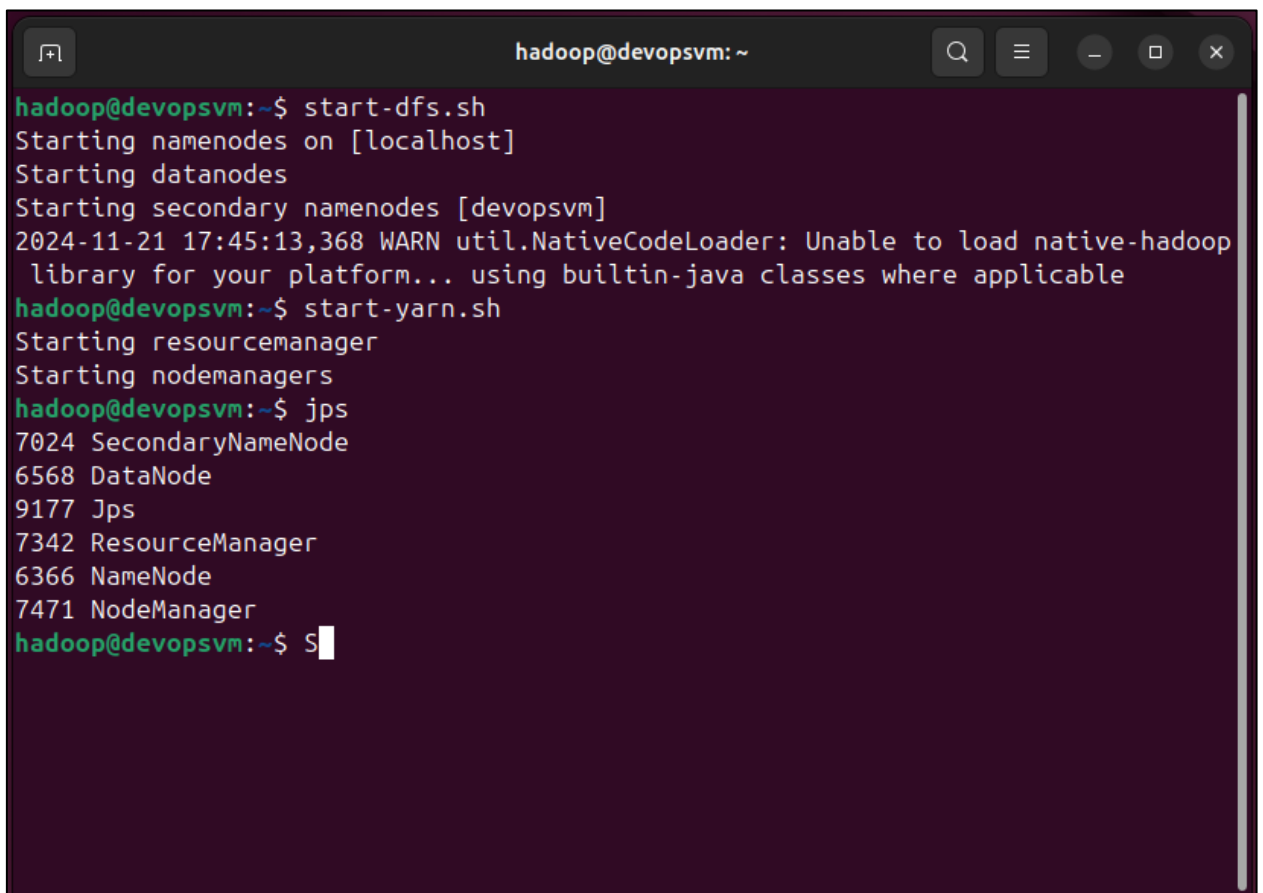
ХОД РАБОТЫ

Шаг 1. Подготовка к работе

В первую очередь, необходимо, в данном случае, запустить виртуальную машину, на которой установлена требуемая операционная система (Ubuntu 24.04 LTS). Далее посредством работы в терминале необходимо запустить Hadoop (Рисунок 1):

```
Start-dfs.sh
Start-dfs.sh
```

Для проверки успешности запуска Hadoop можно воспользоваться командой `jps`. Вы должны увидеть следующие процессы: `NameNode`, `DataNode`, `SecondaryNameNode`, `ResourceManager`, `NodeManager` (Рисунок 1).

A screenshot of a terminal window titled 'hadoop@devopsvm: ~'. The terminal shows the execution of 'start-dfs.sh' and 'start-yarn.sh' commands, followed by a 'jps' command. The output of 'jps' lists five processes: SecondaryNameNode (PID 7024), DataNode (PID 6568), Jps (PID 9177), ResourceManager (PID 7342), and NameNode (PID 6366). The terminal also shows a warning message about the native-hadoop library. The prompt 'hadoop@devopsvm:~\$ S' is visible at the bottom.

```
hadoop@devopsvm:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [devopsvm]
2024-11-21 17:45:13,368 WARN util.NativeCodeLoader: Unable to load native-hadoop
  library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@devopsvm:~$ jps
7024 SecondaryNameNode
6568 DataNode
9177 Jps
7342 ResourceManager
6366 NameNode
7471 NodeManager
hadoop@devopsvm:~$ S
```

Рисунок 1 - Запуск Hadoop и его проверка

Также для проверки успешности запуска возможен переход к веб-интерфейсу HDFS (Hadoop Distributed File System), доступный через веб-браузер на порту 9870 (Рисунок 2). Данный интерфейс позволяет просматривать состояние и структуру файловой системы, а также выполнять операции.

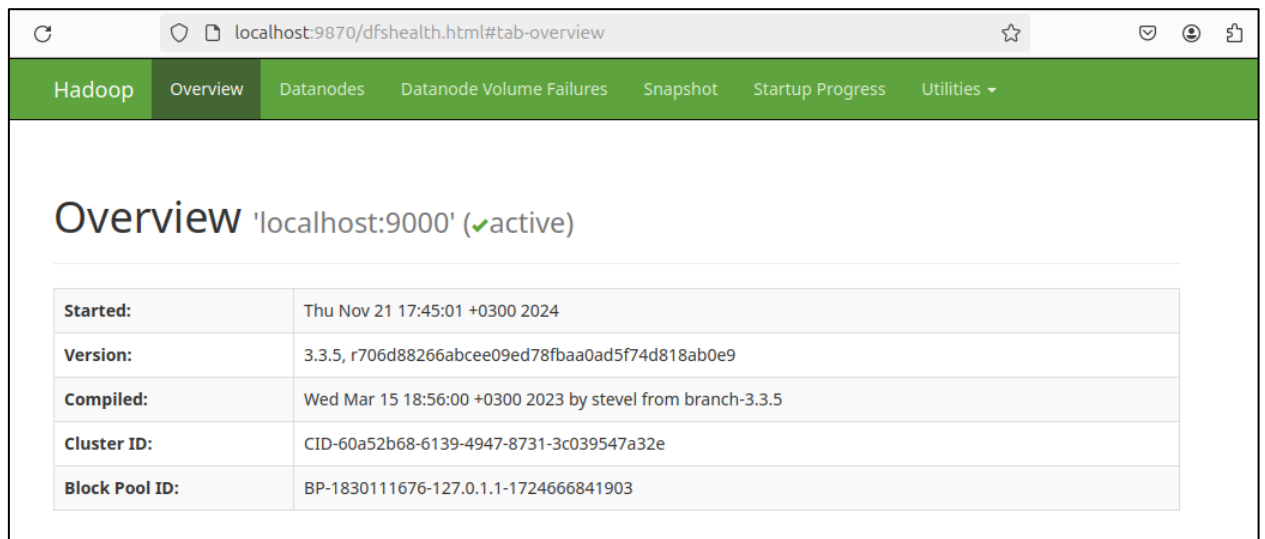


Рисунок 2 – Веб-интерфейс HDFS

Далее требуется создать директорию в HDFS, используя следующие команды в терминале (Рисунок 3):

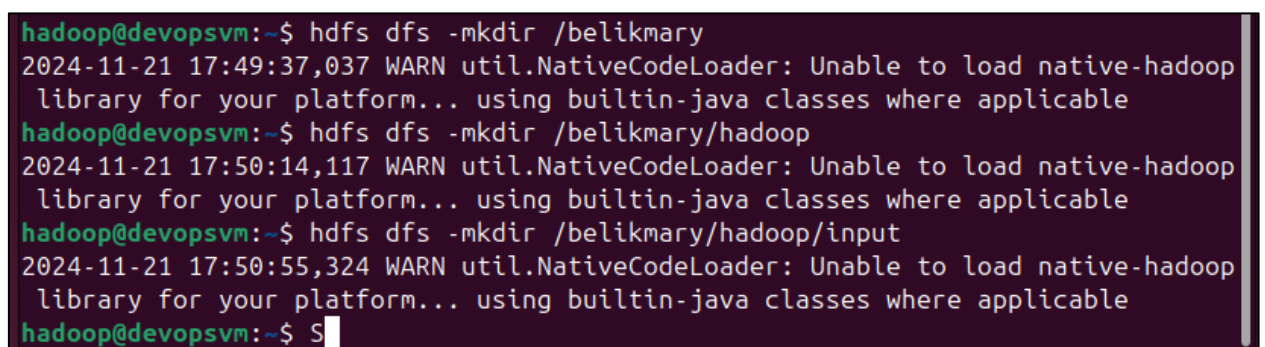


Рисунок 3 – Создание директории

Успешно созданная директория отобразится и в веб-интерфейсе в разделе «Browse Directory» (Рисунок 4).

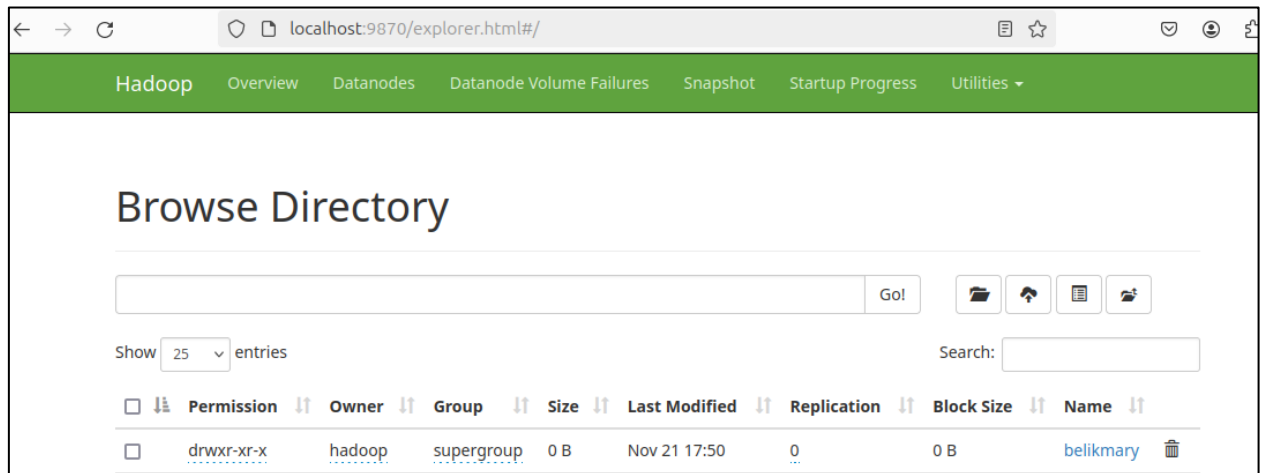


Рисунок 4 – Отображение созданной директории в веб-интерфейсе

Шаг 2. Подготовка данных

В соответствии с **1 вариантом** заданий для дальнейшей работы необходимы исторические данные по акциям Сбербанка (SBER) с сайта Московской биржи (moex.com). Однако для получения данных с данного ресурса требуется оплата.

Поэтому был найден аналогичный сайт InvestFunds (<https://investfunds.ru/stocks/Sberbank/>), в котором предоставляется бесплатный доступ к выгрузке требуемых данных (Рисунок 5).

Однако при попытке выгрузить информацию об акциях Сбербанка, выставив максимально длительный период с 2004 по 2024 года, полученный файл состоял всего из 4344 строк (276 КБ), чего оказалось недостаточно в рамках данной лабораторной работы.

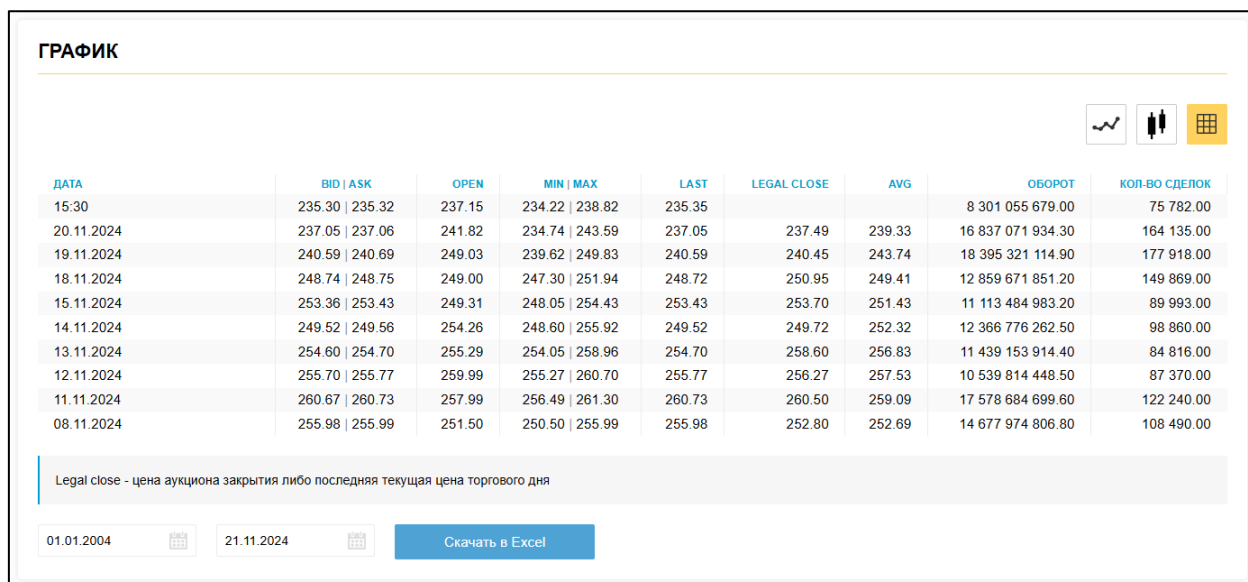


Рисунок 5 – Сайт InvestFunds

В связи с этим, было принято решение воспользоваться Google Colab и сгенерировать данные об акциях SBER в период с 2018 по 2023 года, описывающие цену открытия, закрытия, наивысшую и наименьшую цену, количество и объем продаж, а также дату начала и закрытия торгов (Рисунок 6). Итоговый файл составил 411 МБ.

```
[1] 1 import pandas as pd
    2 import numpy as np

1 start_date = '2018-01-01'
2 end_date = '2023-12-31'
3
4 date_range = pd.date_range(start=start_date, end=end_date, freq='min')
5
6 num_rows = len(date_range)
7
8 np.random.seed(42) #
9 open_prices = np.random.uniform(150, 280, size=num_rows).astype(np.float64)
10 close_prices = np.random.uniform(150, 280, size=num_rows).astype(np.float64)
11 high_prices = np.maximum(open_prices, close_prices) + np.random.uniform(0, 100, size=num_rows).astype(np.float64)
12 low_prices = np.minimum(open_prices, close_prices) - np.random.uniform(0, 100, size=num_rows).astype(np.float64)
13 values = np.random.uniform(1e5, 5e5, size=num_rows).astype(np.float64)
14 volumes = np.random.randint(100, 1000, size=num_rows).astype(np.int64)
15
16 df = pd.DataFrame({
17     'open': open_prices,
18     'close': close_prices,
19     'high': high_prices,
20     'low': low_prices,
21     'value': values,
22     'volume': volumes,
23     'begin': date_range,
24     'end': date_range + pd.Timedelta(minutes=1)
25 })
26
27 print(df.info())
28 print(df.head())

<Class 'pandas.core.frame.DataFrame'>
RangeIndex: 3153601 entries, 0 to 3153600
Data columns (total 8 columns):
#   Column  Dtype
---  ---
0   open    float64
1   close   float64
2   high    float64
3   low     float64
4   value   float64
5   volume  int64
6   begin   datetime64[ns]
7   end     datetime64[ns]
dtypes: datetime64[ns](2), float64(5), int64(1)
memory usage: 192.5 MB
None
      open      close      high      low      value  volume \
0  198.690215  267.527867  365.042998  102.215855  340625.537070    587
1  273.592860  180.404139  303.182795  104.470776  339683.454576    656
2  245.159212  174.587328  284.074778  108.165732  328793.784956    110
3  227.825603  216.535050  254.333812  205.638796  349724.290097    491
4  170.282423  274.649198  368.405402   93.761861  289300.049498    656

      begin      end
0 2018-01-01 00:00:00 2018-01-01 00:01:00
1 2018-01-01 00:01:00 2018-01-01 00:02:00
2 2018-01-01 00:02:00 2018-01-01 00:03:00
3 2018-01-01 00:03:00 2018-01-01 00:04:00
4 2018-01-01 00:04:00 2018-01-01 00:05:00

[5] 1 df.to_csv('SBER_fake_data.csv', index=False)
```

Рисунок 6 - Генерация данных

Далее также требовалось загрузить данные в HDFS, используя GitHub. Но и здесь возникли сложности, связанные с ограничением размера загружаемого файла в 25 МБ (Рисунок 7). Соответственно попытка загрузить файл в 411 МБ закончилась ошибкой и неудачей.

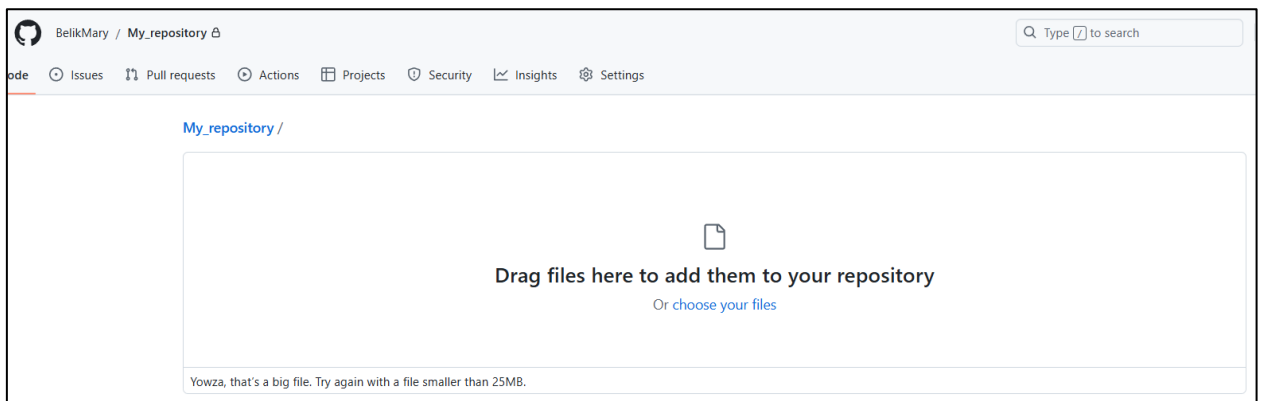


Рисунок 7 – Ограничение на загрузку файлов в GitHub

Из-за данного обстоятельства требовалось воспользоваться доступными функциями веб-интерфейса HDFS. Первым делом, необходимо предоставить другим пользователям доступ на изменения данных (Рисунок 8):

```
hadoop@devopsvm:~$ hdfs dfs -chmod 777 /belikmary/hadoop/input
2024-11-21 19:47:52,582 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$
```

Рисунок 8 - Предоставление прав

Далее перейти в веб-интерфейс, открытый в браузере, выбрать (Рисунок 9) и загрузить (Рисунок 10) требуемый файл со сгенерированными данными об акциях.

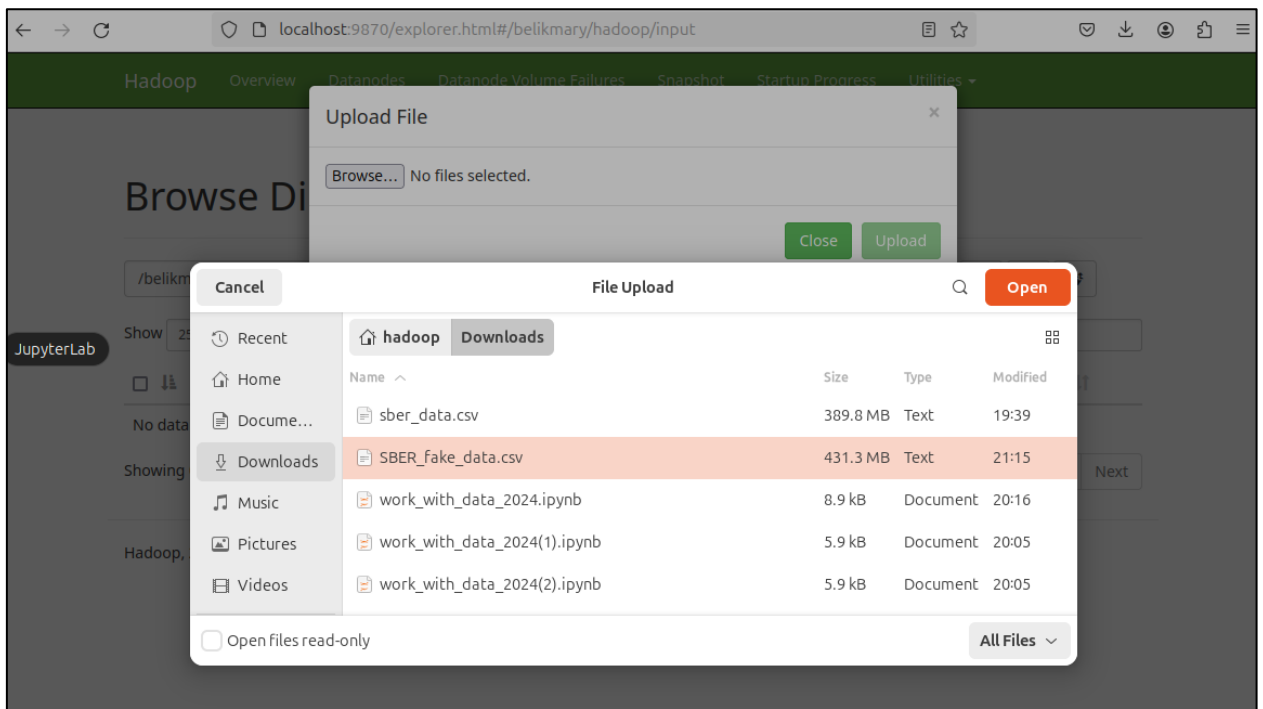


Рисунок 9 – Загрузка файла в веб-интерфейсе

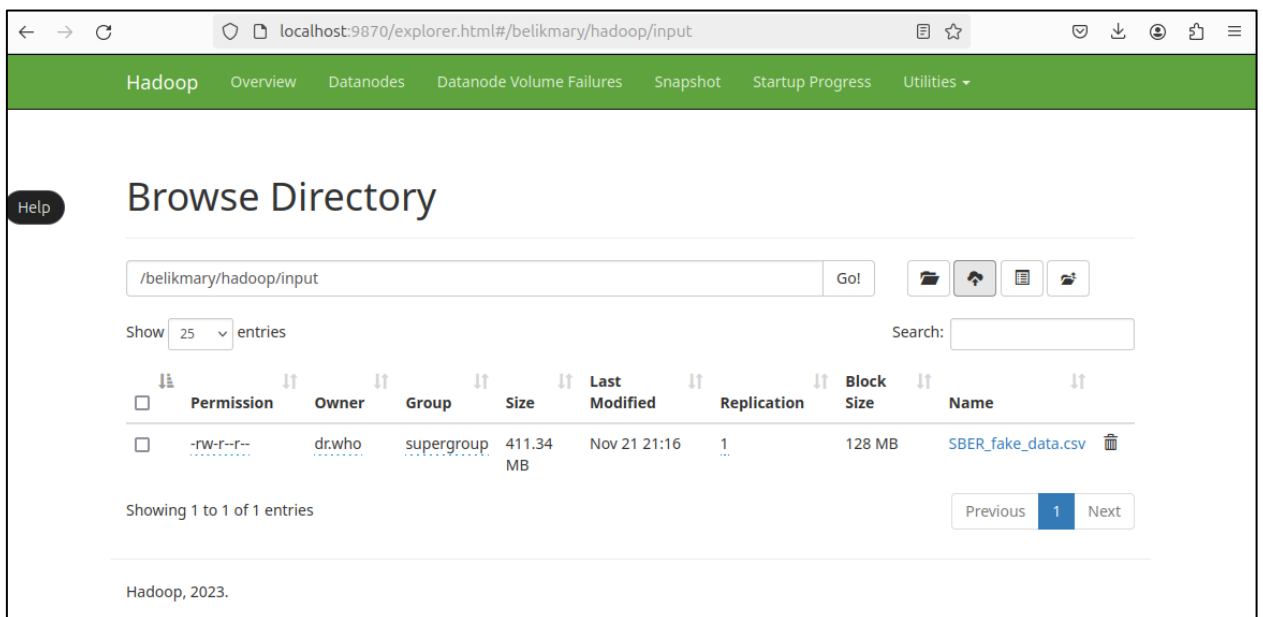


Рисунок 10 – Успешно загруженный файл

Шаг 3. Работа с данными

Дальнейшая работа осуществляется с использованием Jupiter Notebook. Требуется установить `pyspark` и необходимые библиотеки; подключиться к `spark` (Рисунок 11).

```
[1]: !pip install pyspark

Collecting pyspark
  Downloading pyspark-3.5.3.tar.gz (317.3 MB)
    317.3/317.3 MB 5.9 MB/s eta 0:00:0000:0100:01
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl.metadata (1.5 kB)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.3-py2.py3-none-any.whl size=317840629 sha256=fdb8d51184a47b0c7b5e5256e0720fa617296d23100b6c3c6c4a4f81189eeb5e
  Stored in directory: /home/hadoop/snap/jupyterlab-desktop/common/.cache/pip/wheels/07/a0/a3/d24c94bf043ab5c7e38c30491199a2a11fef8d2584e6df7fb7
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.7 pyspark-3.5.3

[2]: import pandas as pd
import matplotlib.pyplot as plt

[4]: from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("SBER 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")
```

Рисунок 11 – Загрузка библиотек и создание сессии

Далее получаем данные из загруженного ранее файла об акциях Сбербанка (Рисунок 12), а также посмотрим их краткий анализ (Рисунок 13)

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

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

	open	close	high	low	value	volume	begin	end
196	69021545015713	267.52786715801955	365.0429983406257	102.21585458949362	340625.5370703045	587	2018-01-01 00:00:00	2018-01-01 00:01:00
273	5928598332891	180.40413878605113	303.1827948065338	104.47077617661047	339683.45457577845	656	2018-01-01 00:01:00	2018-01-01 00:02:00
245	15921243548266	174.587328170176	284.0747784357767	108.16573196152315	328793.784956109	110	2018-01-01 00:02:00	2018-01-01 00:03:00
227	82560294561478	216.5350501051318	254.3338116338352	205.6387956219706	349724.2900965039	491	2018-01-01 00:03:00	2018-01-01 00:04:00
170	28242325751674	274.64919778546664	368.40540205108823	93.76186108446883	289300.04949814733	656	2018-01-01 00:04:00	2018-01-01 00:05:00

only showing top 5 rows

Рисунок 12 – Чтение данных из HDFS

```
[6]: df.describe().show()
```

24/11/21 21:22:16 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.

[Stage 3:=====> (3 + 1) / 4]

summary	open	close	high	low	value	volume
count	3153601	3153601	3153601	3153601	3153601	3153601
mean	215.00376557677103	214.98321178520473	286.6766596451561	143.3175312120938	299995.1629123011	549.433103934201
stddev	37.51930323434735	37.533887006576165	42.07546125365506	42.07156613097191	115445.65211834662	259.7992026840227
min	150.0000490694893	150.00004382571143	151.80387829445976	50.02148614167946	100000.11401646382	100
max	279.9999196913932	279.99998914648467	379.9608527181889	279.2252346608398	499999.9465766154	999

Рисунок 13 – Описание данных

Из краткого описания данных видно, что доступны 3 153 601 строки; что минимальная цена продажи акции за весь период составила 50,02 руб., а максимальная – 378 руб.

Далее, исходя из варианта задания, требовалось отфильтровать значения и вывести только данные за 2020 год (Рисунок 14). Из описания данных видно, что за 2020 год представлены 527 039 строк; средняя цена открытия совпадает со средней ценой закрытия торгов (Рисунок 15).

[10]: one_year=df.filter((df["begin"] >= "2020-01-01")& (df["end"] < "2021-01-01")) one_year.show()

[Stage 7:=====> (2 + 1) / 3]

	open	close	high	low	value	volume	begin	end
176.8347656232267	225.05647303333376	250.3892327667467	107.86926177285845	192952.72831604286	740	2020-01-01 00:00:00	2020-01-01 00:01:00	
258.0938856435462	227.91886648420424	274.13703566059235	128.63865224177619	200863.5753365056	283	2020-01-01 00:01:00	2020-01-01 00:02:00	
217.80831005123787	191.2320776395142	264.5158246061803	154.91625314143118	410816.05020737107	983	2020-01-01 00:02:00	2020-01-01 00:03:00	
179.9644272939741	205.45387895428678	231.32298142521333	147.89718950405603	381203.4364083062	820	2020-01-01 00:03:00	2020-01-01 00:04:00	
254.8256581882964	196.65725652207038	299.5338867250053	106.5774112152369	149103.90539624484	638	2020-01-01 00:04:00	2020-01-01 00:05:00	
246.40450524272328	160.98957117624337	334.46044069481857	109.77544420675686	316460.65637699224	534	2020-01-01 00:05:00	2020-01-01 00:06:00	
223.4453039293548	165.49483737282213	263.2153565054543	67.3948186710175	295243.7052691876	148	2020-01-01 00:06:00	2020-01-01 00:07:00	
191.71664958296847	173.4402122641657	211.88254757159802	172.71708791386547	126209.52229892914	134	2020-01-01 00:07:00	2020-01-01 00:08:00	
159.67145989124455	224.82395992098213	317.3787300613362	80.99105312038732	218499.55405680457	415	2020-01-01 00:08:00	2020-01-01 00:09:00	
231.9617229262083	162.876026545512652	328.44147332809507	88.87939857095265	384296.0500326166	653	2020-01-01 00:09:00	2020-01-01 00:10:00	
262.08894093281026	271.18640402429776	296.86617333884607	206.14039490190336	268594.98214067426	482	2020-01-01 00:10:00	2020-01-01 00:11:00	
160.14180312312692	178.50796294356124	238.5294940996078	90.50577601903161	267118.907782696	774	2020-01-01 00:11:00	2020-01-01 00:12:00	
198.37507171198513	182.44740673754478	248.748066695592	84.9981386117786	172436.93906482318	524	2020-01-01 00:12:00	2020-01-01 00:13:00	
161.23164309330866	173.84092671407626	248.37229109140608	109.31689659516547	259338.79313536835	357	2020-01-01 00:13:00	2020-01-01 00:14:00	
165.69266762010233	247.0877094543603	282.89816908992725	78.39020055544393	142825.2969565002	605	2020-01-01 00:14:00	2020-01-01 00:15:00	
232.61141076034912	211.8892853498798	279.0897803514548	173.59147933355888	138311.8607702084	860	2020-01-01 00:15:00	2020-01-01 00:16:00	
208.41174079964966	239.5611544607213	267.3572993042562	169.3577805831152	410910.0053193103	882	2020-01-01 00:16:00	2020-01-01 00:17:00	
260.8365600747275	242.20866230297883	362.29102874960364	201.623512805782	480909.1566143971	412	2020-01-01 00:17:00	2020-01-01 00:18:00	
250.77518494977176	259.1238152140469	352.14345584649817	241.87800454110955	334880.04757681664	604	2020-01-01 00:18:00	2020-01-01 00:19:00	
241.77927085814	246.0741485974557	336.3695158527417	237.02686064902747	307289.40734952834	965	2020-01-01 00:19:00	2020-01-01 00:20:00	

only showing top 20 rows

Рисунок 14 - Фильтрация по 2020 году

```
[11]: one_year.describe().show()
```

[Stage 8:=====> (3 + 1) / 4]

summary	open	close	high	low	value	volume
count	527039	527039	527039	527039	527039	527039
mean	214.93355493235728	214.9852425004683	286.62117288153064	143.2331524657553	300000.8471293059	549.0596122867568
stddev	37.486073910401046	37.5029822819481	42.102856756167164	42.02383158717188	115440.88795428448	259.73458070839445
min	150.0000490694893	150.00005030567135	153.10260016497065	50.14687278588248	100000.11401646382	100
max	279.9999196913932	279.9995601587561	379.7502105794392	277.70317672858	499999.8908036633	999

Рисунок 15 – Описание данных за 2020 год

Также требовалось найти среднюю цену закрытия торгов, которая в 2020 году составила 215 руб. (Рисунок 16).

```
[14]: from pyspark.sql.functions import avg

avg_close_price = one_year.agg(avg("close").alias("average_close")).collect()[0]["average_close"]
print(avg_close_price)

[Stage 25:=====] (3 + 1) / 4]
214.9852425004683
```

Рисунок 16 – Поиск средней цены закрытия

Далее по заданию необходимо сгруппировать значения по месяцам. На Рисунок 17 представлено среднее количество продаж в разрезе месяцев за весь период с 2018 по 2023 гг.

```
[24]: from pyspark.sql.functions import month, avg, round

group_month = df.groupBy(month("begin").alias("months")).agg(round(avg("value"),2).alias("avg_value")).orderBy("avg_value")
group_month.show()

[Stage 34:=====] (3 + 1) / 4]
+-----+-----+
|months|avg_value|
+-----+-----+
| 1|299493.23|
|10|299848.49|
| 3|299869.54|
| 8| 299886.6|
|12|300031.02|
| 5|300055.59|
| 9|300070.53|
| 6|300073.94|
| 4|300136.77|
|11|300137.92|
| 7|300164.17|
| 2|300207.93|
+-----+-----+
```

Рисунок 17 – Среднее количество продаж по месяцам

На Рисунок 18 представлены средняя цена акции на начало и на конец торгов в разрезе месяцев за весь период с 2018 по 2023 гг. и соответствующая визуализация результата (Рисунок 19).

```
[26]: from pyspark.sql.functions import month, avg, round

group_month = df.groupBy(month("begin").alias("months")) \
    .agg(
        round(avg("open"),2).alias("avg_open"),
        round(avg("close"),2).alias("avg_close")
    ) \
    .orderBy("months")

group_month.show()
```

[Stage 37:=====] (3 + 1) / 4]

months	avg_open	avg_close
1	214.98	215.16
2	214.94	214.95
3	214.88	214.95
4	215.1	214.93
5	215.01	214.86
6	215.02	215.07
7	215.03	214.87
8	215.01	215.1
9	215.07	214.94
10	214.94	215.1
11	215.09	214.94
12	214.96	214.92

Рисунок 18 – Средняя цена закрытия и открытия по месяцам

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

group_month = group_month.toPandas()

plt.figure(figsize=(12,8))
plt.plot(group_month['months'],group_month['avg_open'], label='Avg open price per months')
plt.plot(group_month['months'],group_month['avg_close'], label='Avg close price per months')
plt.xlabel('Month')
plt.ylabel('Avg price')
plt.legend()
plt.grid()
plt.show()
```

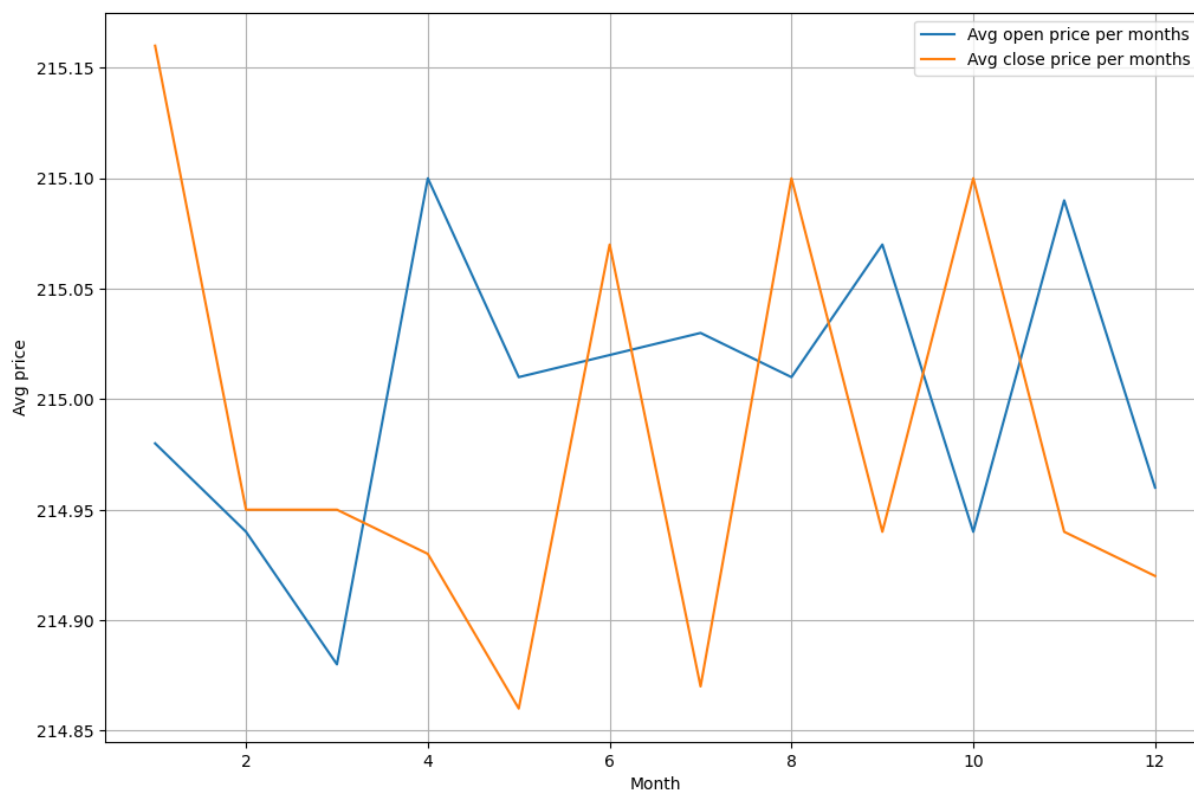


Рисунок 19 – График средних цен закрытия и открытия по месяцам

На Рисунок 20 представлены средняя цена акции на начало и на конец торгов в разрезе месяцев за отдельно взятый 2020 год и соответствующая визуализация результата (Рисунок 21, Рисунок 22).

```
[31]: from pyspark.sql.functions import month, avg, round

group_month_2020 = one_year.groupBy(month("begin").alias("months")) \
    .agg(
        round(avg("open"),2).alias("avg_open"),
        round(avg("close"),2).alias("avg_close")
    ) \
    .orderBy("months")

group_month_2020.show()
```

[Stage 51:=====> (3 + 1) / 4]

months	avg_open	avg_close
1	215.05	215.07
2	214.91	214.63
3	214.96	214.97
4	214.67	215.07
5	215.0	214.99
6	214.9	215.23
7	215.11	214.96
8	214.81	215.1
9	214.86	214.77
10	214.8	215.25
11	215.17	214.87
12	214.96	214.89

Рисунок 20 – Средние цены закрытия и открытия по месяцам за 2020 год

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

group_month_2020 = group_month_2020.toPandas()

plt.figure(figsize=(12,8))
plt.plot(group_month_2020['months'],group_month_2020['avg_open'], label='Avg open price per months')
plt.plot(group_month_2020['months'],group_month_2020['avg_close'], label='Avg close price per months')
plt.title('2020 results')
plt.xlabel('Month')
plt.ylabel('Avg price')
plt.legend()
plt.grid()
plt.show()
```

Рисунок 21 – Код для потсроения графика

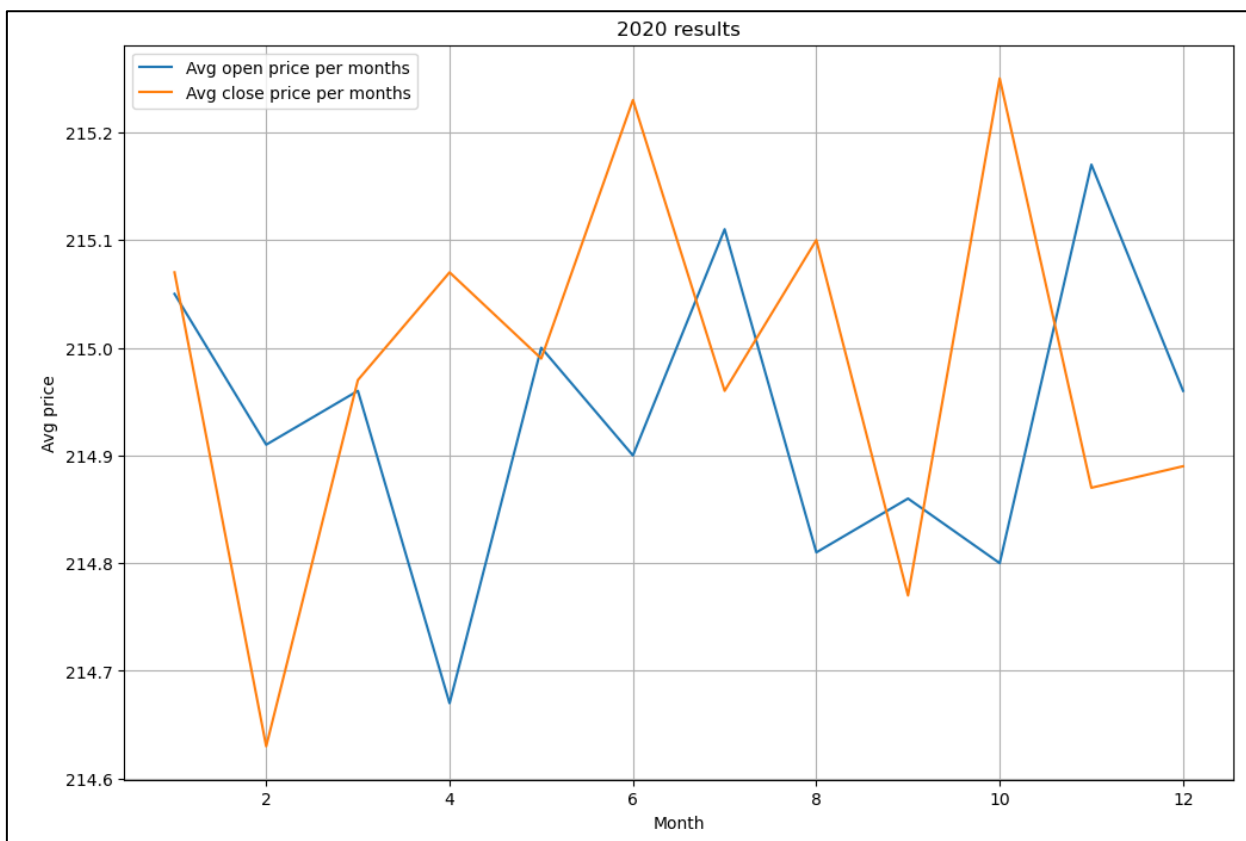


Рисунок 22 – График средних цен закрытия и открытия по месяцам за 2020 год

Исходя из данных визуализаций можно сделать вывод, что тестовые данные подходят для отработки навыков работы в HDFS, однако сделать какие-либо логичные и разумные выводы по ним довольно сложно.

Шаг 4. Загрузка полученных датасетов в HDFS

Полученный датасет отфильтрованных данных по 2020 году загружаем в HDFS (Рисунок 23). В веб-интерфейсе проверяем успешность загрузки данных (Рисунок 24).

```
[34]: file_path_hdfss = "hdfs://localhost:9000/belikmary/hadoop/input/SBER_fake_data_2020.csv"

one_year.write.csv(file_path_hdfss, header=True, mode="overwrite")
```

Рисунок 23 – Загрузка датасета в HDFS

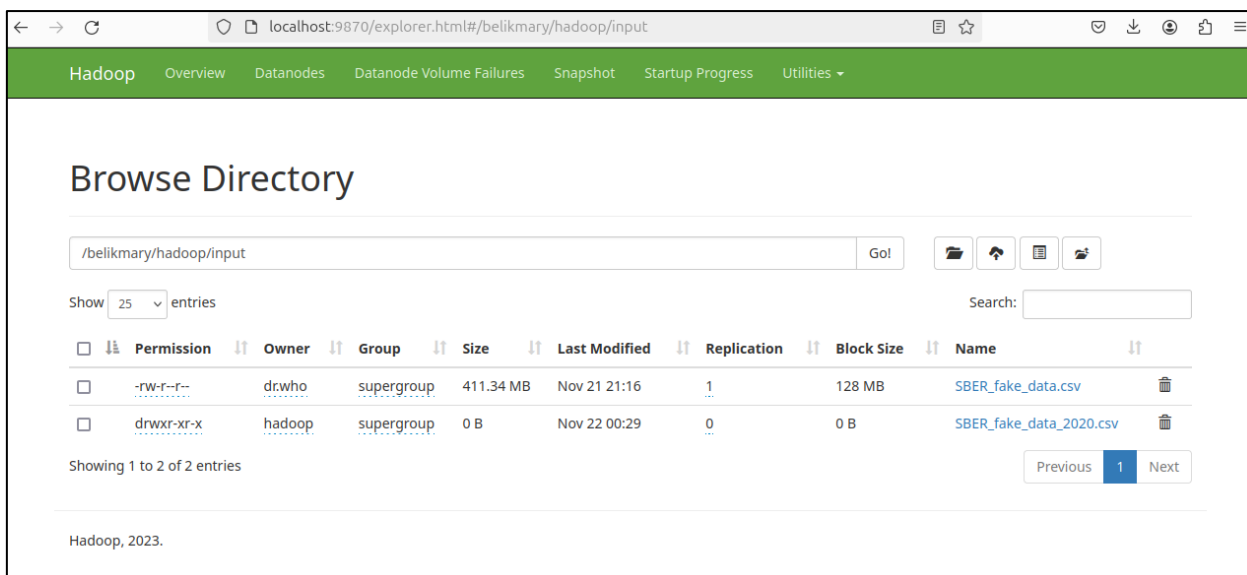


Рисунок 24 – Отображение успешно загруженного датасета в веб-интерфейсе

ВЫВОДЫ

В рамках выполнения данной лабораторной работы была достигнута основная цель – ознакомление с работой в Hadoop, были изучены основные операции и функциональные возможности системы, а также были реализованы все поставленные ранее задачи:

- удалось успешно запустить и подключиться к HDFS;
- были сгенерированы и загружены исторические данные об акциях Сбербанка;
- был произведен анализ подготовленных данных;
- созданный датасет был загружен в HDFS.