



## **МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

---

**Институт кибербезопасности и цифровых технологий (ИКБ)**

---

**КБ-2 «Информационно-аналитические системы кибербезопасности»**

---

**ОТЧЕТ О ВЫПОЛНЕНИИ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ №6**  
**В РАМКАХ ДИСЦИПЛИНЫ «ТЕХНОЛОГИИ**  
**ХРАНЕНИЯ В СИСТЕМАХ КИБЕРБЕЗОПАСНОСТИ»**

Выполнил:

Студент 3-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

Москва 2024

Создайте каталог для нового проекта (например, spark) и сформируйте файл docker-compose.yml для развертывания контейнера с Jupyter Notebook и PySpark (<https://quay.io/repository/jupyter/pyspark-notebook>).

```
version: '3.8'
```

```
services:
```

```
  pyspark:
```

```
    image: quay.io/jupyter/pyspark-notebook:latest
```

```
    container_name: pyspark_zvs
```

```
    volumes:
```

```
      - ./notebooks:/home/jovyan/work
```

```
    ports:
```

```
      - 8805:8888
```

```
      - 4005:4040
```

```
      - 4105:4041
```

```
    networks:
```

```
      - pyspark-net
```

```
    environment:
```

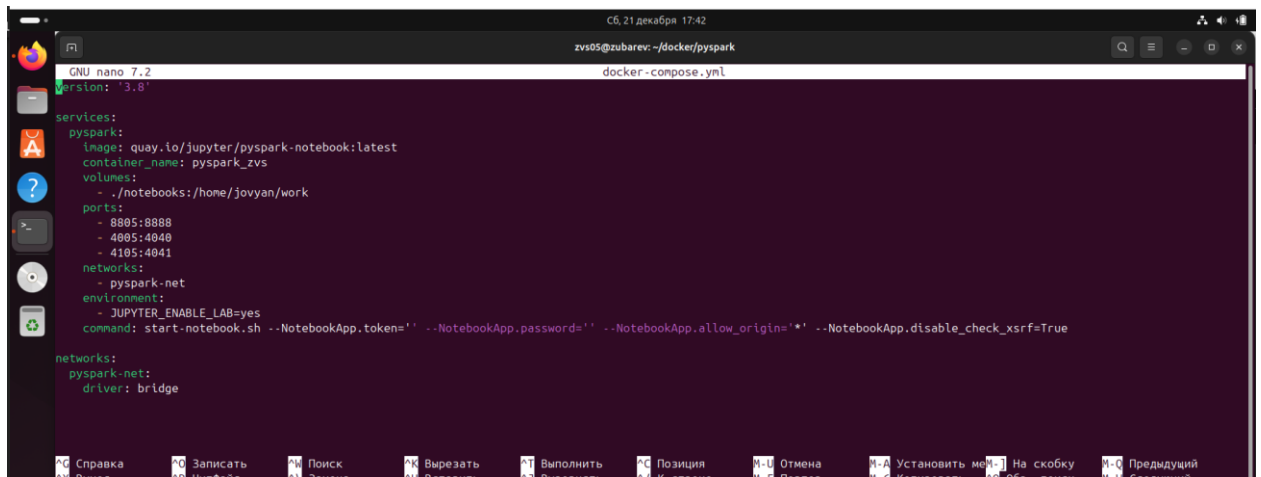
```
      - JUPYTER_ENABLE_LAB=yes
```

```
    command: start-notebook.sh --NotebookApp.token="" NotebookApp.password=""  
--NotebookApp.allow_origin='*' --NotebookApp.disable_check_xsrf=True
```

```
networks:
```

```
  pyspark-net:
```

```
    driver: bridge
```

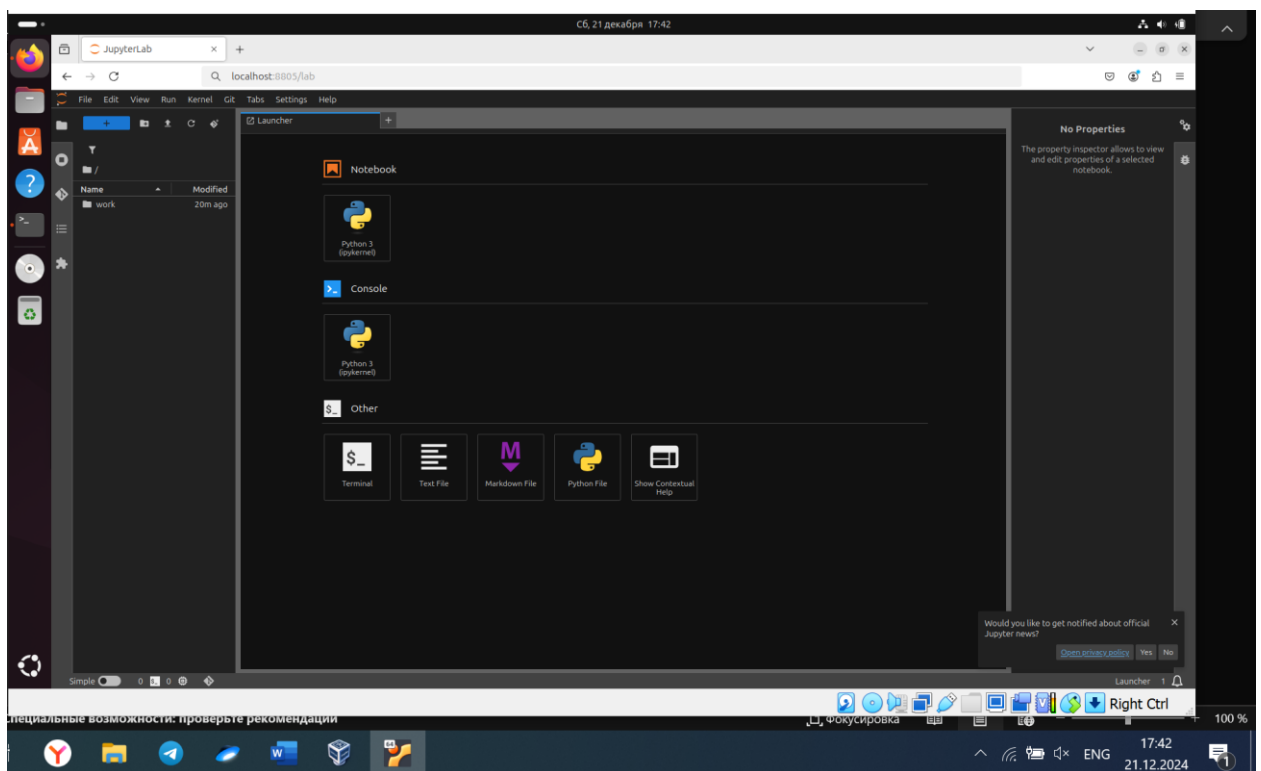


```
GNU nano 7.2
version: '3.8'

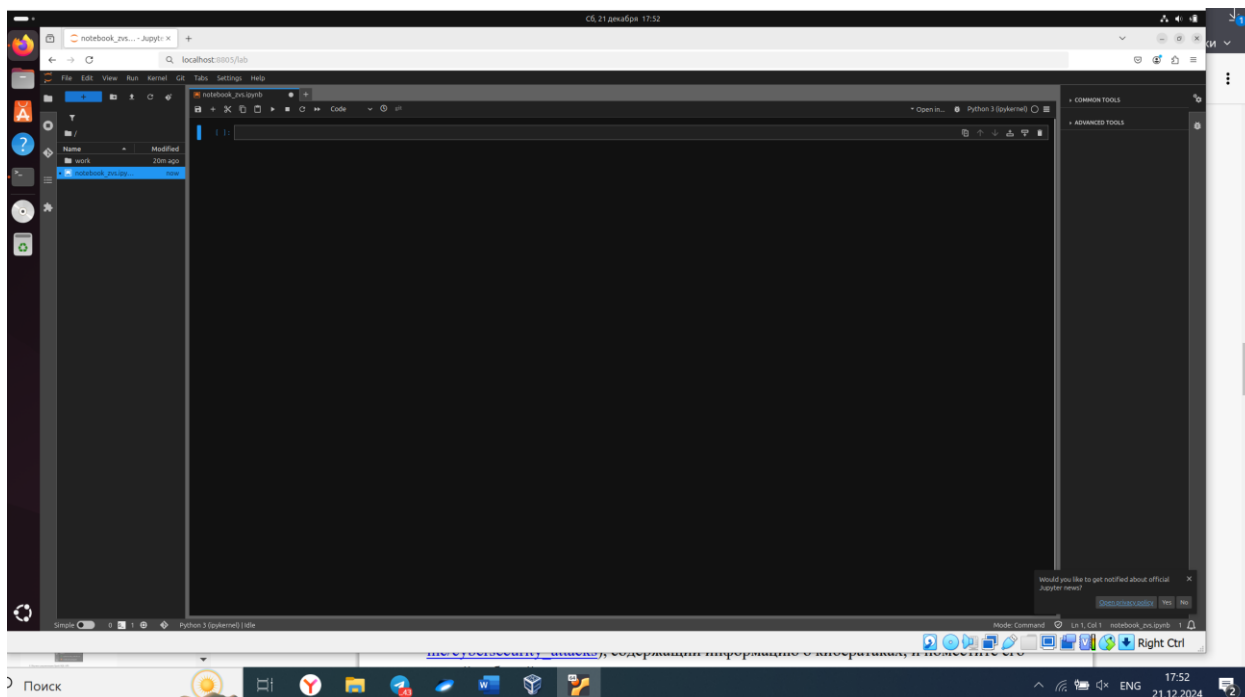
services:
  pyspark:
    image: quay.io/jupyter/pyspark-notebook:latest
    container_name: pyspark_zvs
    volumes:
      - /notebooks:/home/jovyan/work
    ports:
      - 8885:8888
      - 4005:4040
      - 4195:4041
    networks:
      - pyspark-net
    environment:
      - JUPYTER_ENABLE_LAB=yes
    command: start-notebook.sh --NotebookApp.token='' --NotebookApp.password='' --NotebookApp.allow_origin='*' --NotebookApp.disable_check_xsrf=True

networks:
  pyspark-net:
    driver: bridge
```

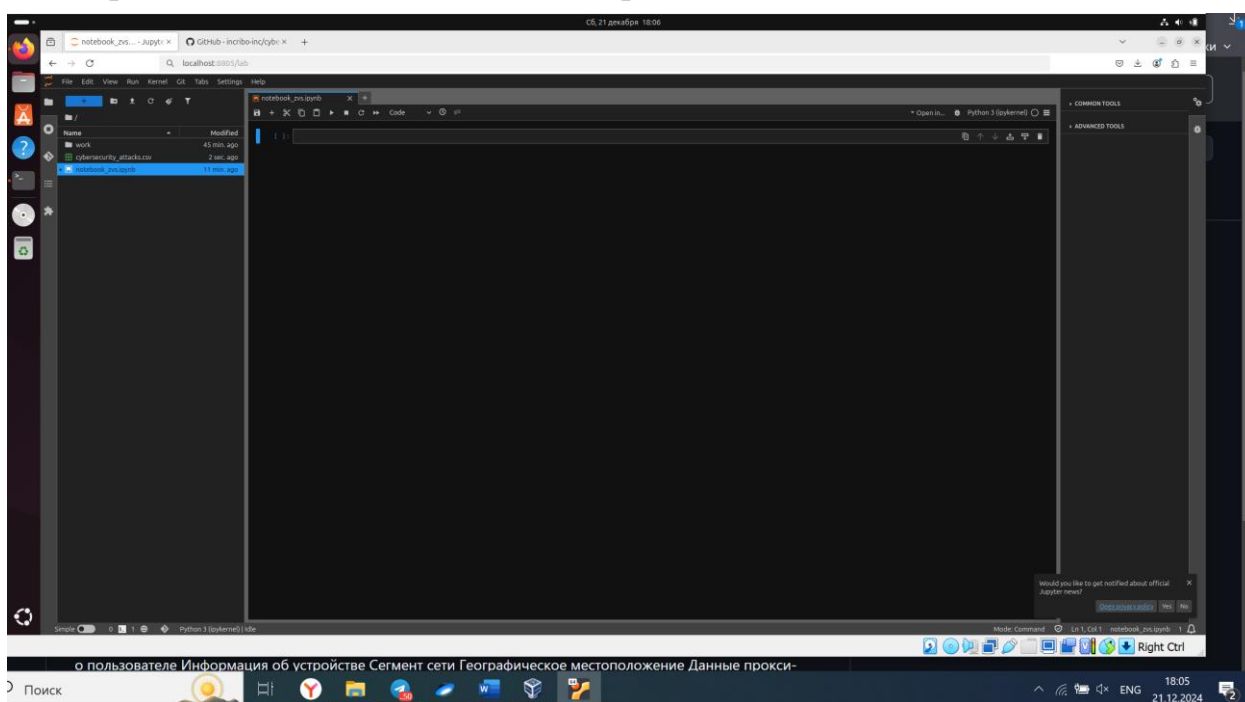
Разверните контейнер с помощью Docker Compose



Создайте блокнот (notebook). Название блокнота должно заканчиваться на символ подчеркивания и инициалы ФИО.



Загрузите CSV-файл `cybersecurity_attacks.csv` ([https://github.com/incrimboinc/cybersecurity\\_attacks](https://github.com/incrimboinc/cybersecurity_attacks)), содержащий информацию о кибератаках, и поместите его в свой рабочий каталог.



Импортируйте модули библиотеки PySpark, содержащие функции и типы данных для работы с данными в Spark DataFrame (`pyspark.sql.functions`, `pyspark.sql.types`). Создайте сессию Spark. Название приложения должно заканчиваться на символ подчеркивания и инициалы ФИО.

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as f
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,
DoubleType, DateType
```

# Создание сессии Spark

```
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()
```

# Загрузка данных из CSV файла

```
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True,
multiLine=True, inferSchema=True)
```

# Печать схемы данных для проверки

```
df_zvs.printSchema()
```

# Печать первых 5 строк данных для проверки

```
df_zvs.show()
```

```
# Печать схемы данных для проверки
df_zvs.printSchema()

# Печать первых 5 строк данных для проверки
df_zvs.show()
```

```
root
 |-- Timestamp: timestamp (nullable = true)
 |-- Source IP Address: string (nullable = true)
 |-- Destination IP Address: string (nullable = true)
 |-- Source Port: integer (nullable = true)
 |-- Destination Port: integer (nullable = true)
 |-- Protocol: string (nullable = true)
 |-- Packet Length: integer (nullable = true)
 |-- Packet Type: string (nullable = true)
 |-- Traffic Type: string (nullable = true)
 |-- Payload Data: string (nullable = true)
 |-- Malware Indicators: string (nullable = true)
 |-- Anomaly Score: double (nullable = true)
 |-- Alerts/Warnings: string (nullable = true)
 |-- Attack Type: string (nullable = true)
 |-- Attack Signature: string (nullable = true)
 |-- Action Taken: string (nullable = true)
 |-- Severity Level: string (nullable = true)
 |-- User Information: string (nullable = true)
 |-- Device Information: string (nullable = true)
 |-- Network Segment: string (nullable = true)
 |-- Geo Location Data: string (nullable = true)
 |-- Proxy Information: string (nullable = true)
 |-- Firewall Logs: string (nullable = true)
 |-- IDS/IPS Alerts: string (nullable = true)
 |-- Log Source: string (nullable = true)
```

Timestamp	Source IP Address	Destination IP Address	Source Port	Destination Port	Protocol	Packet Length	Packet Type	Traffic Type	Payload Data	Malware Indicators	Anomaly Score	Alerts/Warnings	Attack Type	Attack Sign
2022-05-30 06:33:58	102.216.15.121	84.9.164.252	31225	17618	ICMP	503	Data	HTTP[DoS status edis as...]	ICD Detected		28.67	NLLI	Malware	Known Patt
2022-06-26 07:08:30	78.199.217.198	66.181.137.154	17245	48106	SMTP	1174	Data	HTTP[Aperion quos modi...]	ICD Detected		51.51	NLLI	Malware	Known Patt
2022-11-13 08:21:25	63.79.218.48	188.119.82.171	14811	5360	HTTP	366	Control	Log Data	ICD Detected		87.42	Alert Triggered	DDoS	Known Patt
2022-07-02 08:38:46	163.47.186.181	188.224.182.253	28818	32134	HTTP	381	Data	Log Data	Alert Data	Firewall	15.79	Alert Triggered	Malware	Known Patt
2022-07-16 13:31:07	71.166.185.761	188.243.124.258	6311	2646	TCP	342	Data	HTTP[Status main head...]	ICD Detected		8.52	Alert Triggered	DDoS	Known Patt
2022-07-18 13:14:21	108.102.5.568	147.186.165.121	17408	52602	UDP	1423	Data	HTTP[Status main head...]	ICD Detected		5.76	NLLI	Malware	Known Patt
2022-08-18 13:14:21	67.253.282.299	77.66.161.51	16561	17418	UDP	1423	Data	HTTP[Status main head...]	ICD Detected		31.55	NLLI	DDoS	Known Patt
2022-02-12 07:13:17	11.48.04.245	178.157.14.118	34889	28596	SMTP	1022	Data	OS[Linux Libera opti...]	ICD Detected		54.05	Alert Triggered	Intrusion	Known Patt
2022-02-12 07:13:17	11.48.04.245	178.157.14.118	34889	28596	SMTP	1022	Data	OS[Linux Libera opti...]	ICD Detected		54.05	Alert Triggered	Intrusion	Known Patt

Сделать выборку необходимых полей можно с помощью функции select.

```

from pyspark.sql import SparkSession
import pyspark.sql.functions as f
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,
DoubleType, DateType

```

# Создание сессии Spark

```
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()
```

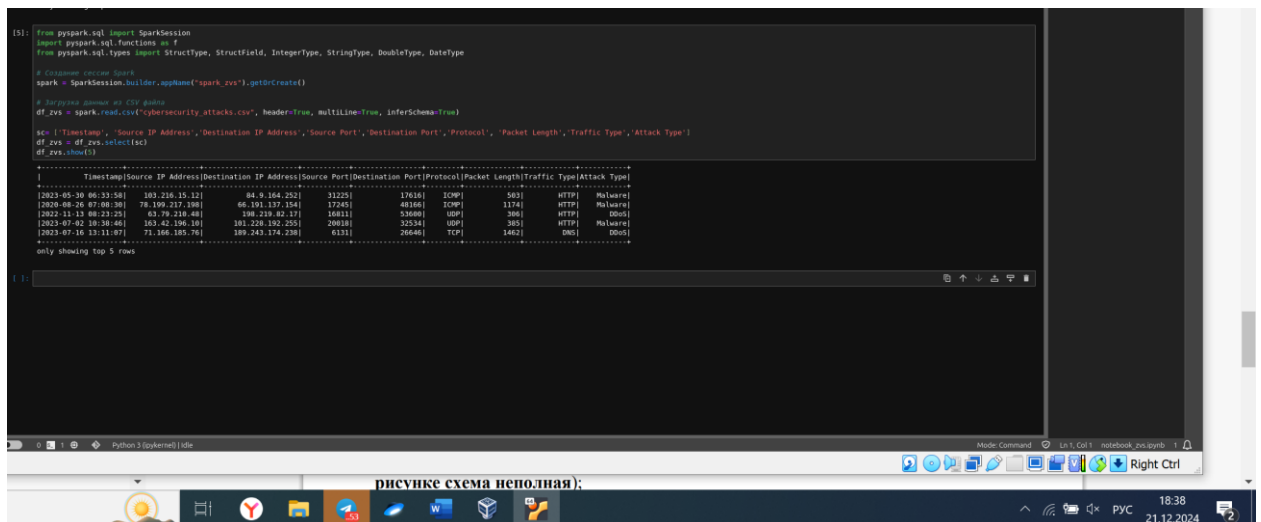
# Загрузка данных из CSV файла

```
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True,
multiLine=True, inferSchema=True)
```

```

sc= ['Timestamp', 'Source IP Address', 'Destination IP Address', 'Source
Port', 'Destination Port', 'Protocol', 'Packet Length', 'Traffic Type', 'Attack Type']
df_zvs = df_zvs.select(sc)
df_zvs.show(5)

```



```

[5]: from pyspark.sql import SparkSession
import pyspark.sql.functions as f
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType, DateType

# Создание сессии Spark
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()

# Загрузка данных из CSV файла
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True, multiLine=True, inferSchema=True)

sc= ['Timestamp', 'Source IP Address', 'Destination IP Address', 'Source Port', 'Destination Port', 'Protocol', 'Packet Length', 'Traffic Type', 'Attack Type']
df_zvs = df_zvs.select(sc)
df_zvs.show(5)

```

Timestamp	Source IP Address	Destination IP Address	Source Port	Destination Port	Protocol	Packet Length	Traffic Type	Attack Type
2023-05-30 06:33:58	103.216.15.121	84.9.184.252	31225	17638	ICMP	583	HTTP	Malware
2020-06-28 07:00:30	78.199.217.198	66.101.137.154	17245	48186	ICMP	1174	HTTP	Malware
2022-11-15 08:12:25	63.79.218.40	198.218.82.17	16811	53680	UDP	360	HTTP	OSint
2023-07-02 18:38:46	103.42.196.10	181.228.192.255	20038	32534	UDP	385	HTTP	Malware
2023-07-16 13:11:07	71.146.185.70	188.243.174.288	6131	26646	TCP	1462	DNS	OSint

only showing top 5 rows

Приведите поля к соответствующим типам данных, предварительно создав схему данных и использовав ее при загрузке

```

from pyspark.sql import SparkSession
import pyspark.sql.functions as f

```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,
DoubleType, DateType
```

```
# Создание сессии Spark
```

```
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()
```

```
# Определение схемы для загрузки данных
```

```
schema = StructType([
    StructField("Timestamp", DateType(), True),
    StructField("Source IP Address", StringType(), True),
    StructField("Destination IP Address", StringType(), True),
    StructField("Source Port", IntegerType(), True),
    StructField("Destination Port", IntegerType(), True),
    StructField("Protocol", StringType(), True),
    StructField("Packet Length", IntegerType(), True),
    StructField("Packet Type", StringType(), True),
    StructField("Traffic Type", StringType(), True),
    StructField("Payload Data", StringType(), True),
    StructField("Malware Indicators", StringType(), True),
    StructField("Anomaly Scores", DoubleType(), True),
    StructField("Alerts/Warnings", StringType(), True),
    StructField("Attack Type", StringType(), True),
    StructField("Attack Signature", StringType(), True),
    StructField("Action Taken", StringType(), True),
    StructField("Severity Level", StringType(), True),
    StructField("User Information", StringType(), True),
    StructField("Device Information", StringType(), True),
    StructField("Network Segment", StringType(), True),
    StructField("Geo-location Data", StringType(), True),
    StructField("Proxy Information", StringType(), True),
    StructField("Firewall Logs", StringType(), True),
    StructField("IDS/IPS Alerts", StringType(), True),
    StructField("Log Source", StringType(), True)
])
```

```
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True,
multiLine=True, inferSchema=True)
```

```
sc= ['Timestamp', 'Source IP Address','Destination IP Address','Source
Port','Destination Port','Protocol', 'Packet Length','Traffic Type','Attack Type']
df_zvs = df_zvs.select(sc)
df_zvs.show(5)
```

The screenshot shows a Jupyter Notebook interface with a file explorer on the left, a code editor in the center, and a console output at the bottom. The code in the notebook defines a schema with 15 fields: Timestamp, Source IP Address, Destination IP Address, Source Port, Destination Port, Protocol, Packet Length, Traffic Type, Payload Data, Malware Indicators, Anomaly Score, Attack Signature, Severity Level, Device Information, Network Segment, and Log Source. It then reads a CSV file named 'cybersecurity\_attacks.csv' using Spark's read.csv method with header=True, multiLine=True, and inferSchema=True. The code selects specific columns into a variable 'sc' and displays the first 5 rows of the resulting DataFrame.

```
schema = StructType([
    StructField('Timestamp', DataType(), True),
    StructField('Source IP Address', StringType(), True),
    StructField('Destination IP Address', StringType(), True),
    StructField('Source Port', IntegerType(), True),
    StructField('Destination Port', IntegerType(), True),
    StructField('Protocol', StringType(), True),
    StructField('Packet Length', IntegerType(), True),
    StructField('Traffic Type', StringType(), True),
    StructField('Payload Data', StringType(), True),
    StructField('Malware Indicators', StringType(), True),
    StructField('Anomaly Score', DoubleType(), True),
    StructField('Attack Signature', StringType(), True),
    StructField('Severity Level', StringType(), True),
    StructField('Device Information', StringType(), True),
    StructField('Network Segment', StringType(), True),
    StructField('Log Source', StringType(), True)
])

df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True, multiLine=True, inferSchema=True)

sc = ['Timestamp', 'Source IP Address','Destination IP Address','Source Port','Destination Port','Protocol', 'Packet Length','Traffic Type','Attack Type']
df_zvs = df_zvs.select(sc)
df_zvs.show(5)
```

Timestamp	Source IP Address	Destination IP Address	Source Port	Destination Port	Protocol	Packet Length	Traffic Type	Attack Type
2023-08-28 06:33:48	260.218.15.121	86.9.164.252	33225	17816	ICMP	593	HTTP	Malware
2020-08-28 07:08:38	78.199.217.198	66.191.137.154	17245	48166	ICMP	1174	HTTP	Malware
2022-11-13 08:23:25	63.79.218.48	198.219.42.171	16811	53686	UDP	386	HTTP	DDoS
2023-07-02 19:18:46	165.42.196.181	101.729.182.259	20018	32534	UDP	393	HTTP	Malware
2023-07-16 13:31:07	71.166.185.78	189.243.174.238	6131	28648	TCP	1462	DNS	DDoS

Выведите 10 записей, в которых порт источника (Source Port) делится без остатка на ваш номер по списку, умноженный на 10.

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as f
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,
DoubleType, DateType
```

```
# Создание сессии Spark
```



```
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()
```

# Загрузка данных из CSV файла

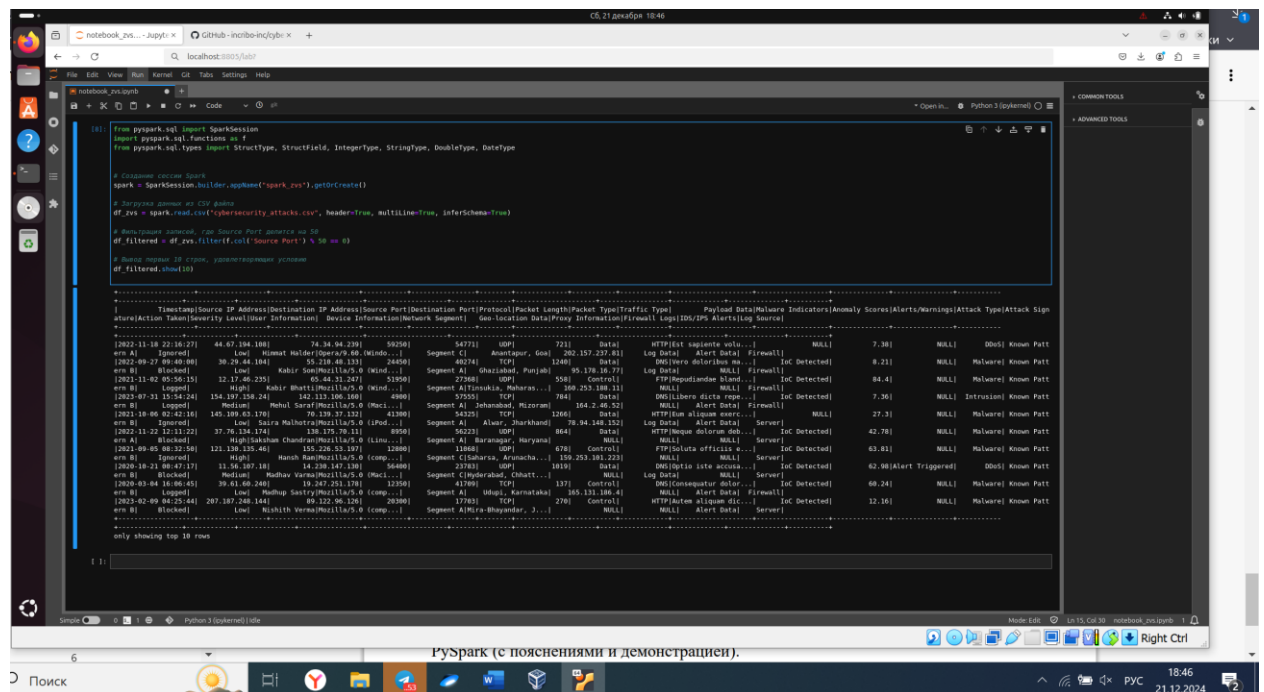
```
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True,  
multiLine=True, inferSchema=True)
```

# Фильтрация записей, где Source Port делится на 50

```
df_filtered = df_zvs.filter(f.col('Source Port') % 50 == 0)
```

# Вывод первых 10 строк, удовлетворяющих условию

```
df_filtered.show(10)
```



```
from pyspark.sql import SparkSession  
import pyspark.sql.functions as f  
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType, DateType  
  
# Создаем сессию Spark  
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()  
  
# Загружаем данные из CSV файла  
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True, multiLine=True, inferSchema=True)  
  
# Фильтруем данные, где Source Port делится на 50  
df_filtered = df_zvs.filter(f.col('Source Port') % 50 == 0)  
  
# Выводим первые 10 строк, удовлетворяющих условию  
df_filtered.show(10)
```

Timestamp	Source	Destination	IP Address	Source Port	Destination Port	Protocol	Packet Length	Packet Type	Traffic Type	Payload Data	Malware Indicators	Anomaly Scores	Alerts/Warnings	Attack Type	Attack Sign	
2022-11-18 22:16:27	44.67.194.198	74.34.94.239	56250	54771	UDP	721	Data	HTTP[Ext] sapientia volu...	NALL	7.38	NALL	Objs	Known Patt			
erm BI Ignored	Low	Kumant Noida	India	202.157.237.81	Log Data	Alert Data	Firewall			8.23	NALL	Malware	Known Patt			
2022-09-27 09:48:08	38.29.44.184	55.218.48.131	26450	40274	TCP	1240	Data	DNS[Response] 192.168.1.1	ICD Detected	84.41	NALL	Malware	Known Patt			
erm BI Blocked	Low	Kabir Saha	India	95.196.18.771	27880	UDP	558	Control	FTP[Response] 192.168.1.1	ICD Detected	7.36	NALL	Intrusion	Known Patt		
2022-11-02 05:56:15	12.17.46.235	65.44.31.247	51500	57525	TCP	744	Data	DNS[Request] 192.168.1.1	ICD Detected	27.31	NALL	Malware	Known Patt			
erm BI Ignored	High	Kumar Noida	India	159.253.181.11	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		
2022-07-31 15:54:24	154.197.258.24	142.132.106.168	4890	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt			
erm BI Ignored	Low	Saira Malhotra	India	78.94.148.152	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		
2022-10-06 02:42:10	145.199.63.138	78.94.148.152	41300	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt			
erm BI Ignored	Low	Saira Malhotra	India	78.94.148.152	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		
2022-11-02 05:56:15	12.17.46.235	65.44.31.247	51500	57525	TCP	744	Data	DNS[Request] 192.168.1.1	ICD Detected	27.31	NALL	Malware	Known Patt			
erm BI Ignored	High	Kumar Noida	India	159.253.181.11	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		
2022-09-05 08:32:24	122.128.129.44	159.253.181.11	13000	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt			
erm BI Ignored	High	Kumar Noida	India	159.253.181.11	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		
2022-10-06 02:42:10	145.199.63.138	78.94.148.152	41300	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt			
erm BI Ignored	Low	Saira Malhotra	India	78.94.148.152	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		
2022-08-04 16:06:43	39.61.66.240	19.247.251.191	13300	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt			
erm BI Ignored	Low	Madhav Varma	India	159.253.181.11	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		
2022-02-09 04:25:44	207.187.248.144	89.122.96.126	28300	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt			
erm BI Ignored	Low	Madhav Varma	India	159.253.181.11	54235	TCP	1240	Data	HTTP[Request] 192.168.1.1	ICD Detected	42.78	NALL	Malware	Known Patt		

Выведите результат группировки по типу трафика (Traffic Type) с подсчетом количества записей.

```
from pyspark.sql import SparkSession
```

```
import pyspark.sql.functions as f
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,  
DoubleType, DateType
```

# Создание сессии Spark

```
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()
```

```
# Загрузка данных из CSV файла
```

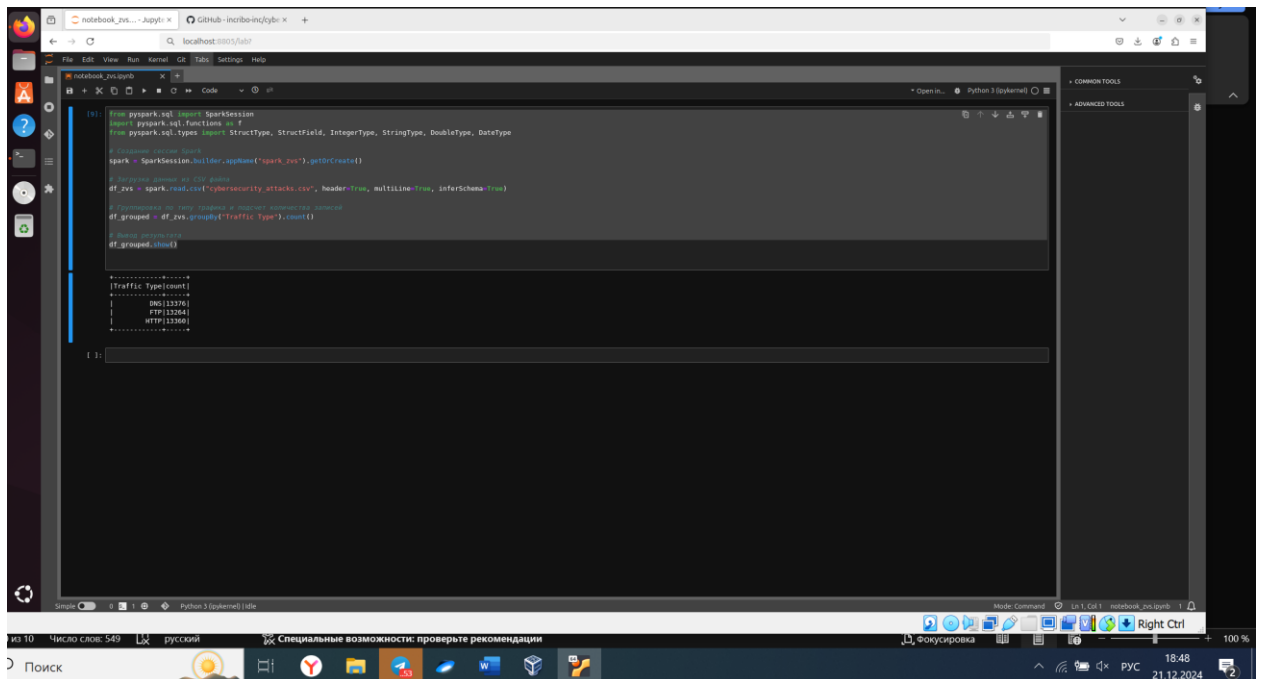
```
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True,  
multiLine=True, inferSchema=True)
```

```
# Группировка по типу трафика и подсчет количества записей
```

```
df_grouped = df_zvs.groupBy("Traffic Type").count()
```

```
# Вывод результата
```

```
df_grouped.show()
```



Выведите результат группировки по протоколу (Protocol) с подсчетом среднего размера пакета

```
from pyspark.sql import SparkSession
```

```
import pyspark.sql.functions as f
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,  
DoubleType, DateType
```

```
# Создание сессии Spark
```

```
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()
```

# Загрузка данных из CSV файла

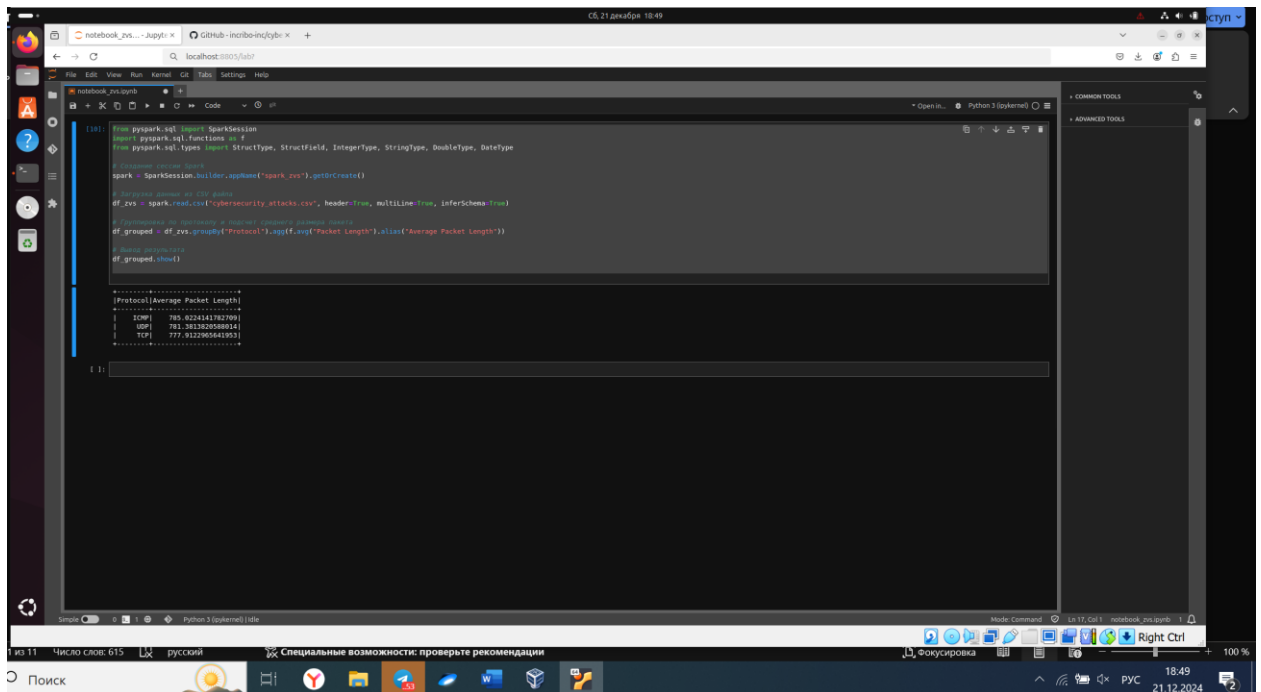
```
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True,  
multiLine=True, inferSchema=True)
```

# Группировка по протоколу и подсчет среднего размера пакета

```
df_grouped = df_zvs.groupBy("Protocol").agg(f.avg("Packet  
Length").alias("Average Packet Length"))
```

# Вывод результата

```
df_grouped.show()
```



Выведите 10 IP-адресов получателя, отсортированных в порядке убывания.

```
from pyspark.sql import SparkSession
```

```
import pyspark.sql.functions as f
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,  
DoubleType, DateType
```

# Создание сессии Spark

```
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()
```

# Загрузка данных из CSV файла

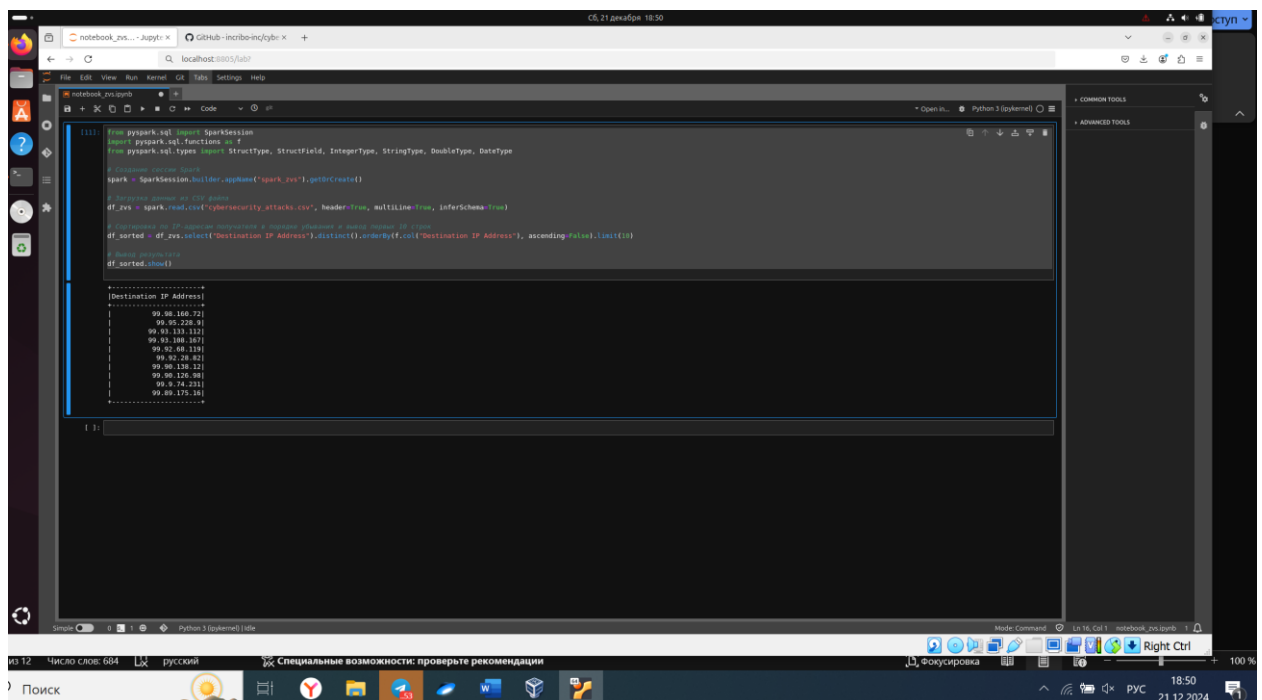
```
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True,  
multiLine=True, inferSchema=True)
```

# Сортировка по IP-адресам получателя в порядке убывания и вывод первых 10 строк

```
df_sorted = df_zvs.select("Destination IP  
Address").distinct().orderBy(f.col("Destination IP Address"),  
ascending=False).limit(10)
```

# Вывод результата

```
df_sorted.show()
```



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
from pyspark.sql import SparkSession  
import pyspark.sql.functions as f  
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType, DateType  
  
spark = SparkSession.builder.appName("spark_zvs").getOrCreate()  
  
# Загрузка данных из CSV файла  
df_zvs = spark.read.csv("cybersecurity_attacks.csv", header=True, multiLine=True, inferSchema=True)  
  
# Сортировка по IP-адресам получателя в порядке убывания и вывод первых 10 строк  
df_sorted = df_zvs.select("Destination IP Address").distinct().orderBy(f.col("Destination IP Address"),  
ascending=False).limit(10)  
  
# Вывод результата  
df_sorted.show()
```

The output of the code is displayed below the code cell, showing the first 10 distinct destination IP addresses in descending order:

```
+-----+  
|Destination IP Address|  
+-----+  
|99.98.160.74|  
|99.95.238.47|  
|99.93.133.112|  
|99.93.188.187|  
|99.92.68.139|  
|99.92.28.42|  
|99.90.138.121|  
|99.90.126.88|  
|99.9.74.281|  
|99.89.175.161|  
+-----+
```

Сформируйте свой тестовый набор данных (не менее 7 столбцов и не менее 10000 записей).

# Установка библиотеки faker (если она еще не установлена)

```
!pip install faker
```

# Импортируем необходимые библиотеки

```

import pandas as pd
import random
from faker import Faker

# Создание экземпляра Faker для генерации случайных данных
fake = Faker()

# Количество строк данных
num_rows = 10000

# Список для хранения сгенерированных данных
data = []

# Генерация данных
for _ in range(num_rows):
    timestamp = fake.date_time_this_year() # Генерация случайной временной
метки
    source_ip = fake.ipv4() # Генерация случайного IP-адреса источника
    destination_ip = fake.ipv4() # Генерация случайного IP-адреса получателя
    source_port = random.randint(1024, 65535) # Генерация случайного порта
источника
    destination_port = random.randint(1024, 65535) # Генерация случайного
порта получателя
    protocol = random.choice(['TCP', 'UDP']) # Выбор случайного протокола
    packet_length = random.randint(40, 1500) # Генерация случайной длины
пакета
    traffic_type = random.choice(['Normal', 'Attack']) # Тип трафика
(нормальный или атака)
    attack_type = random.choice(['DDoS', 'SQL Injection', 'XSS', 'None']) if
traffic_type == 'Attack' else 'None'

# Добавляем сгенерированные данные в список
data.append([timestamp, source_ip, destination_ip, source_port,
destination_port, protocol, packet_length, traffic_type, attack_type])

```

# Создаем DataFrame из сгенерированных данных

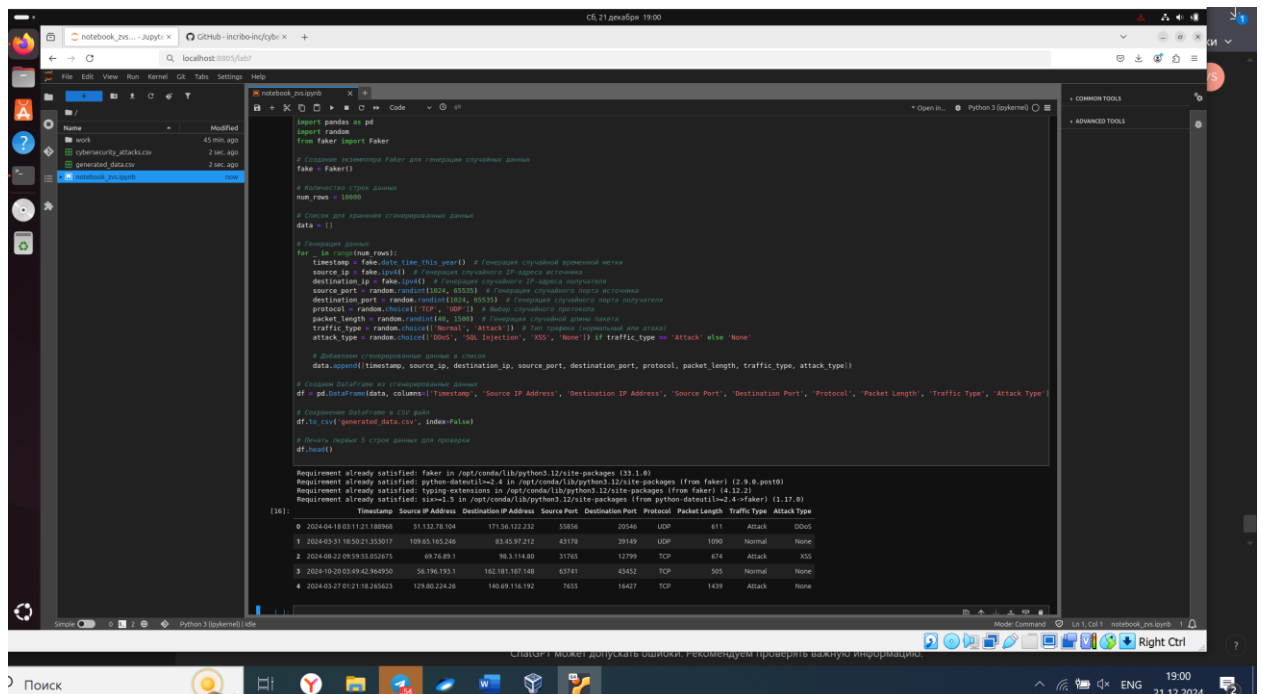
```
df = pd.DataFrame(data, columns=['Timestamp', 'Source IP Address', 'Destination IP Address', 'Source Port', 'Destination Port', 'Protocol', 'Packet Length', 'Traffic Type', 'Attack Type'])
```

# Сохранение DataFrame в CSV файл

```
df.to_csv('generated_data.csv', index=False)
```

# Печать первых 5 строк данных для проверки

```
df.head()
```



```
import pandas as pd
import random
from faker import Faker

# Создаем генератора Faker для генерации случайных данных
fake = Faker()

# Создаем список данных
num_rows = 10000

# Список для хранения сгенерированных данных
data = []

# Генерируем данные
for _ in range(num_rows):
    timestamp = fake.date_time_this_year() # Генерируем случайную временную метку
    source_ip = fake.ipv4() # Генерируем случайный IP-адрес источника
    destination_ip = fake.ipv4() # Генерируем случайный IP-адрес назначения
    source_port = random.randint(1024, 65535) # Генерируем случайный порт источника
    destination_port = random.randint(1024, 65535) # Генерируем случайный порт назначения
    protocol = random.choice(['TCP', 'UDP']) # Выбираем случайный протокол
    packet_length = random.randint(10, 1500) # Генерируем случайную длину пакета
    traffic_type = random.choice(['Normal', 'Attack']) # Выбираем тип трафика (нормальный или атака)
    attack_type = random.choice(['DoS', 'SQL Injection', 'XSS', None]) if traffic_type == 'Attack' else None

# Добавляем сгенерированные данные в список
data.append([timestamp, source_ip, destination_ip, source_port, destination_port, protocol, packet_length, traffic_type, attack_type])

# Создаем DataFrame из сгенерированных данных
df = pd.DataFrame(data, columns=['Timestamp', 'Source IP Address', 'Destination IP Address', 'Source Port', 'Destination Port', 'Protocol', 'Packet Length', 'Traffic Type', 'Attack Type'])

# Сохраняем DataFrame в CSV файл
df.to_csv('generated_data.csv', index=False)

# Выводим первые 5 строк данных для проверки
df.head()
```

	Timestamp	Source IP Address	Destination IP Address	Source Port	Destination Port	Protocol	Packet Length	Traffic Type	Attack Type
0	2024-04-18 03:11:21.089066	191.102.79.104	171.56.102.232	55550	20546	UDP	811	Attack	DoS
1	2024-05-21 18:30:21.353017	109.65.165.246	83.45.97.212	43178	39149	UDP	1090	Normal	None
2	2024-05-22 09:59:51.012475	89.76.89.1	95.3.114.80	31765	12799	TCP	674	Attack	XSS
3	2024-10-20 03:49:42.964950	56.196.193.1	162.161.187.148	63741	43452	TCP	505	Normal	None
4	2024-03-27 01:21:18.265423	129.80.224.26	148.69.116.192	7655	14627	TCP	1439	Attack	None

Проведите не менее 5 манипуляций с новым набором данных с помощью PySpark (с пояснениями и демонстрацией).

Фильтрация данных (атака или нормальная)

```
from pyspark.sql import SparkSession
import pandas as pd
```

# Создание Spark сессии

```
spark = SparkSession.builder.appName("PySpark Data  
Manipulation").getOrCreate()
```

```
# Загрузка данных из CSV в DataFrame
```

```
df = pd.read_csv('generated_data.csv')
```

```
# Конвертация pandas DataFrame в PySpark DataFrame
```

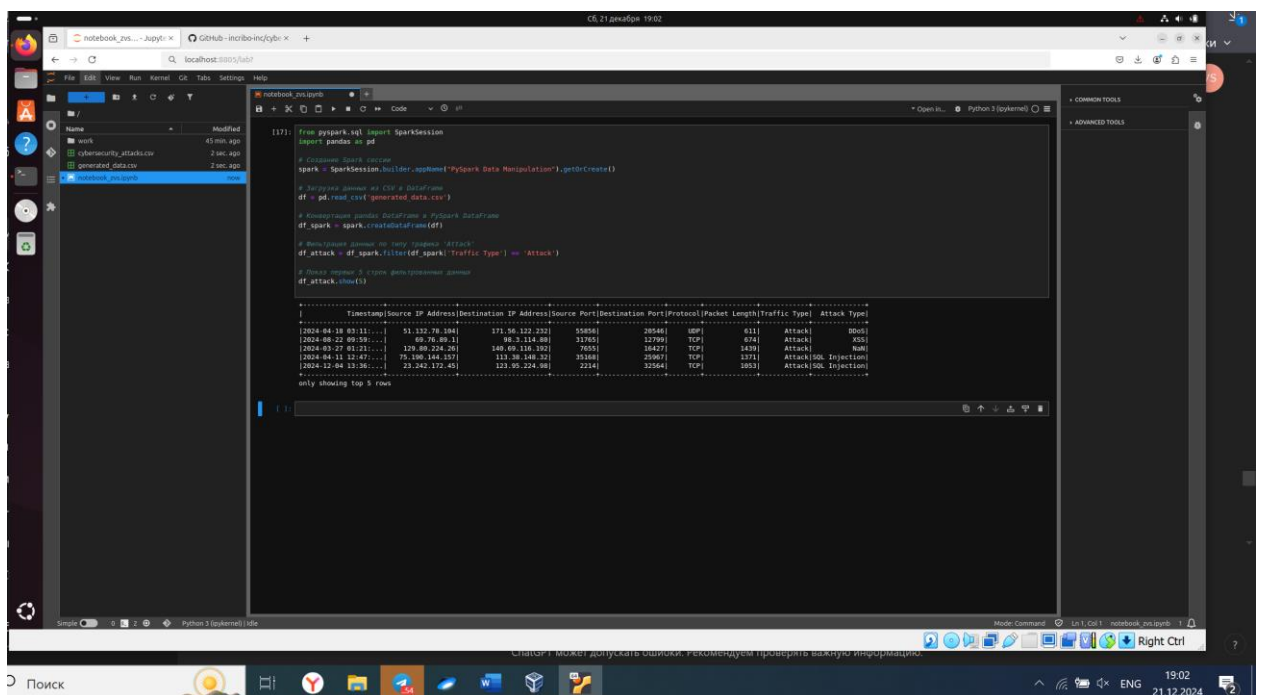
```
df_spark = spark.createDataFrame(df)
```

```
# Фильтрация данных по типу трафика 'Attack'
```

```
df_attack = df_spark.filter(df_spark['Traffic Type'] == 'Attack')
```

```
# Показ первых 5 строк фильтрованных данных
```

```
df_attack.show(5)
```

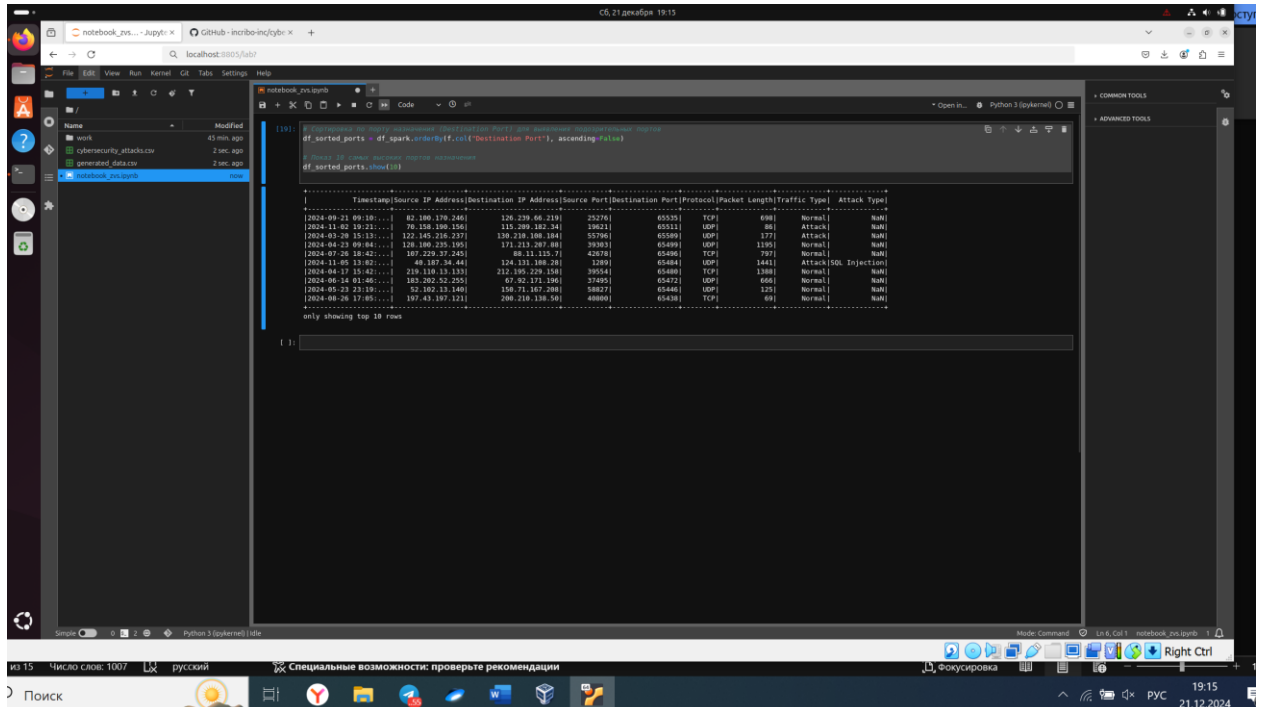


Выявление аномальной активности по портам

```
# Сортировка по порту назначения (Destination Port) для выявления  
подозрительных портов
```

```
df_sorted_ports = df_spark.orderBy(f.col("Destination Port"), ascending=False)
```

# Показ 10 самых высоких портов назначения  
`df_sorted_ports.show(10)`



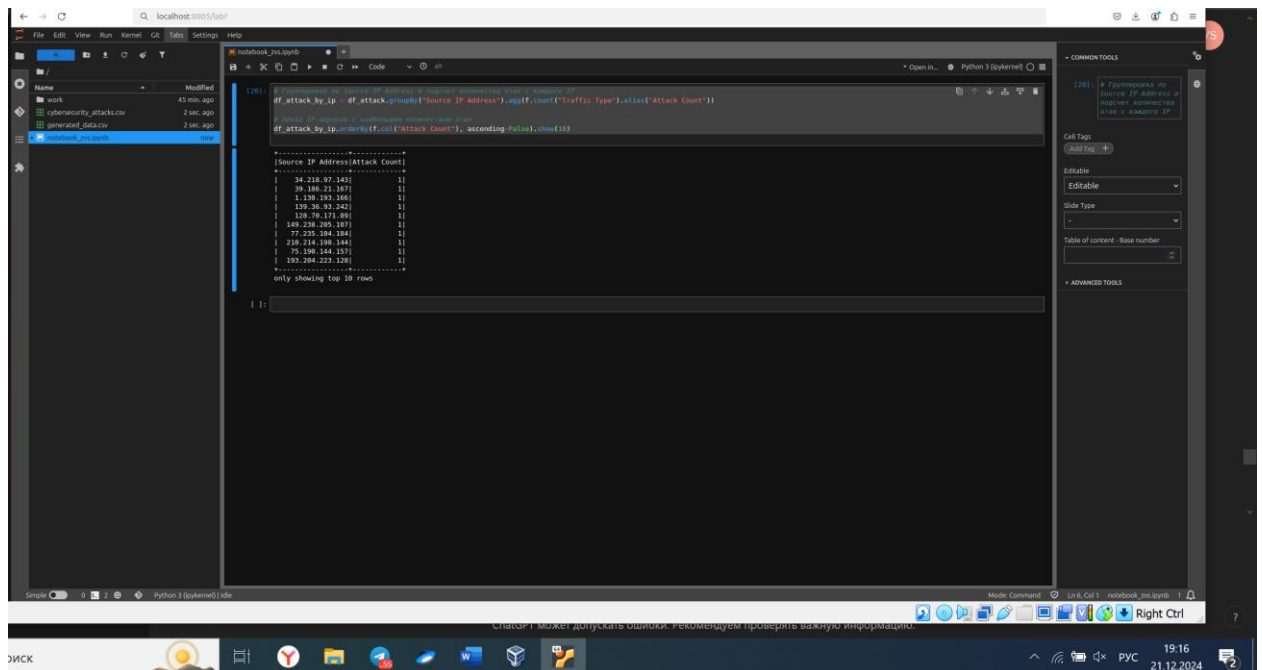
Группировка по IP-адресам для поиска атак по источнику

# Группировка по Source IP Address и подсчет количества атак с каждого IP  
`df_attack_by_ip = df_attack.groupby("Source IP Address").agg(f.count("Traffic Type").alias("Attack Count"))`

# Показ IP-адресов с наибольшим количеством атак

`df_attack_by_ip.orderBy(f.col("Attack Count"), ascending=False).show(10)`





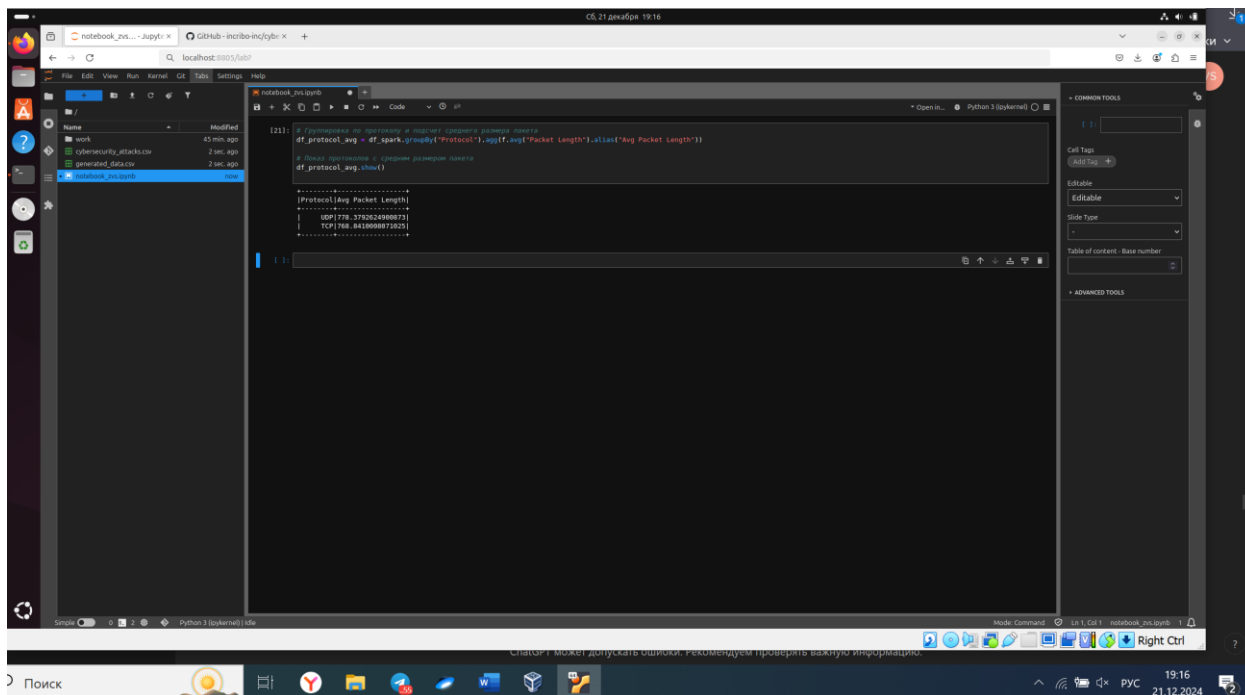
Анализ средних размеров пакетов по протоколу

# Группировка по протоколу и подсчет среднего размера пакета

```
df_protocol_avg = df_spark.groupBy("Protocol").agg(f.avg("Packet Length").alias("Avg Packet Length"))
```

# Показ протоколов с средним размером пакета

```
df_protocol_avg.show()
```



## Выявление аномального трафика по временным меткам

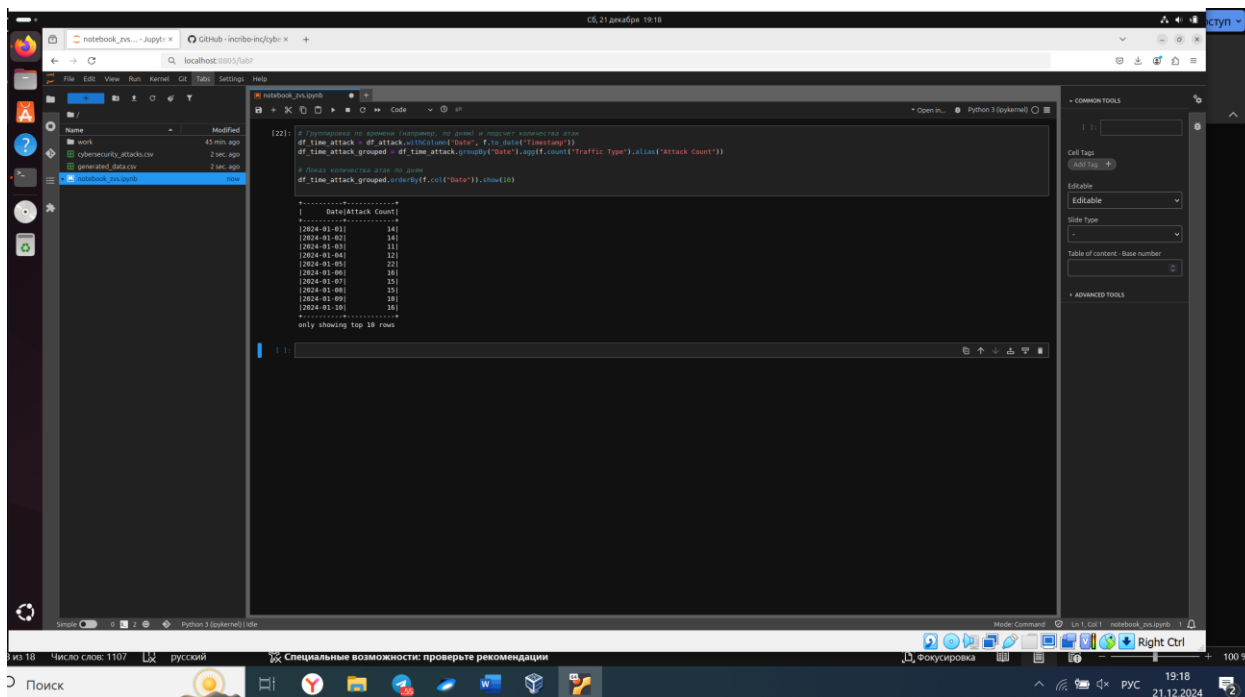
# Группировка по времени (например, по дням) и подсчет количества атак

```
df_time_attack = df_attack.withColumn("Date", f.to_date("Timestamp"))
```

```
df_time_attack_grouped = df_time_attack.groupBy("Date").agg(f.count("Traffic Type").alias("Attack Count"))
```

# Показ количества атак по дням

```
df_time_attack_grouped.orderBy(f.col("Date")).show(10)
```



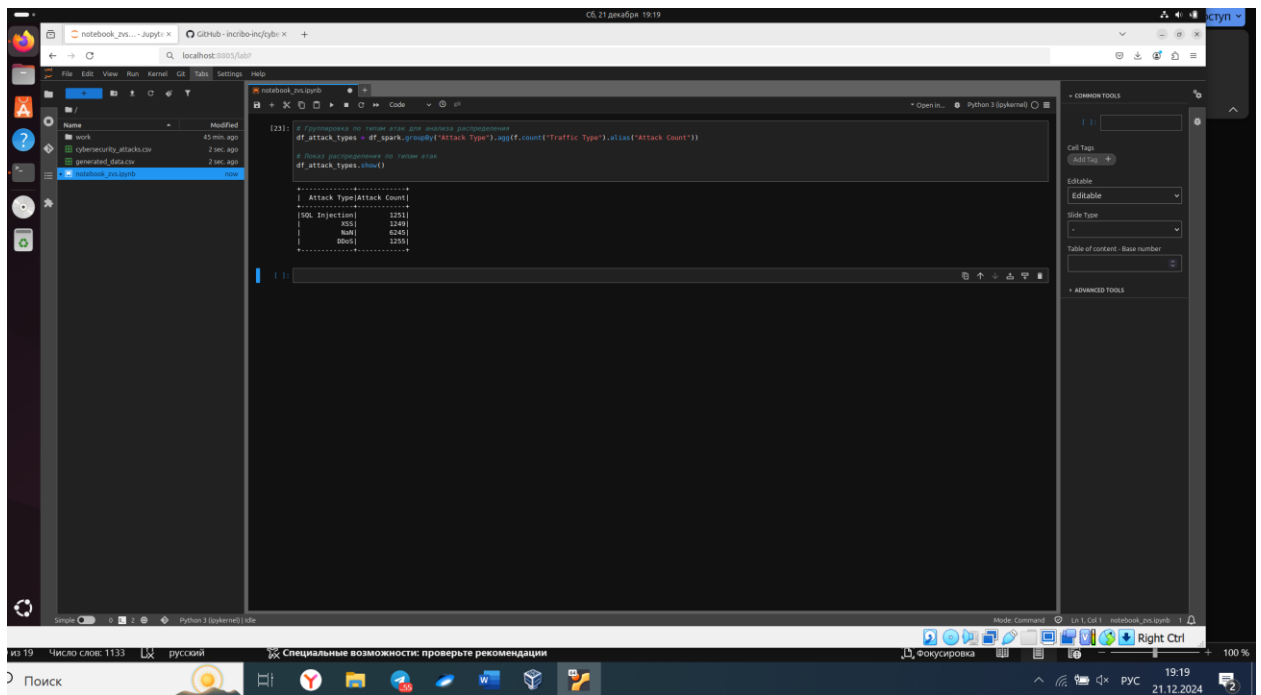
## Признаки атаки по типам атак

# Группировка по типам атак для анализа распределения

```
df_attack_types = df_spark.groupBy("Attack Type").agg(f.count("Traffic  
Type").alias("Attack Count"))
```

# Показ распределения по типам атак

```
df_attack_types.show()
```



Обогащение данных: расчет объемов трафика

# Добавление нового столбца 'Traffic Volume' (объем трафика)

```
df_with_volume = df_spark.withColumn('Traffic Volume', df_spark['Packet Length'] * df_spark['Destination Port'])
```

# Показ 5 строк с новыми данными

```
df_with_volume.show(5)
```

