Big-Data-Analytics-in-Spark-with-Python-and-SQL

Concepts

Release 1

October 2021

Big-Data-Analytics-in-Spark-with-Python-and-SQL, Release 1

Primary Author: Panagiotis Leliopoulos

Contents

1. **Introduction** 4
2. **Coding challenge** 5
3. **Analyzing log data with Python and Apache Spark** 6
4. **Setting up dependencies** 10
5. **Loading and viewing the NASA Log Dataset** 12
6. **How to analyze log data with Python and Apache Spark** 18
7. **How to run and use the code** 26
8. **Assumptions and Conclusions** 30

1. Introduction

In this code challenge, we will see how to provide solutions and reliable answers to some analytics issues. We will see how to set up and configure our solution and what method we use for data loading and data analytics. After that, we provide our solutions and we answer the questions.

2. Coding challenge

In this chapter, we are including the description and the requirements of the Oracle GBU Cloud Services Offline Test. The pronunciation is below.

CODING CHALLENGE

Use a programming language of your choice, and keeping non-standard library dependencies to a minimum (less than 5), parse the sample HTTP sample log file available at NASA-FTP, containing the following information:

1. Top 10 requested pages and the number of requests made for each
2. Percentage of successful requests (anything in the 200s and 300s range)
3. Percentage of unsuccessful requests (anything that is not in the 200s or 300s range)
4. Top 10 unsuccessful page requests
5. The top 10 hosts making the most requests, displaying the IP address and the number of requests made.
6. Option parsing to produce only the report for one of the previous points (e.g., only the top 10 URLs, only the percentage of successful requests, and so on)
7. For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each page
8. The log file contains malformed entries; for each malformed line, display an error message and the line number.

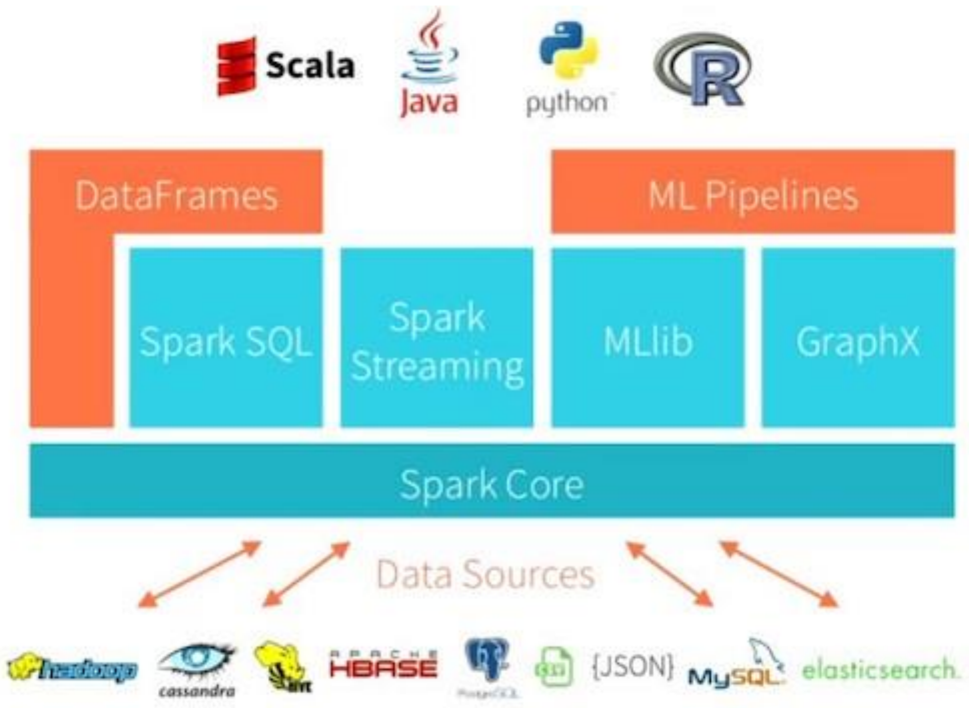
REQUIREMENTS

- The deliverable must be a repository under version control and contain the following:
- All source files.
- Documentation explaining how to use the tool, what its dependencies are, and any assumptions you made while writing it.
- Unit tests.

3. Analyzing log data with Python and Apache Spark

In this chapter, we will see how to analyze log data with python and Apache Spark.

One of the most popular and effective business use cases that use analytical data today is log data analysis. Almost every organization today has multiple systems and infrastructures that operate day by day. To keep their business running, these organizations need to know if their infrastructure is working to the best of their ability. The finding includes the analysis of system and application logs and perhaps even the application of predictive analytics to log data. The volume of log data is usually huge, depending on the type of organizational infrastructure involved and the applications running on it.



The log data processing pipeline.

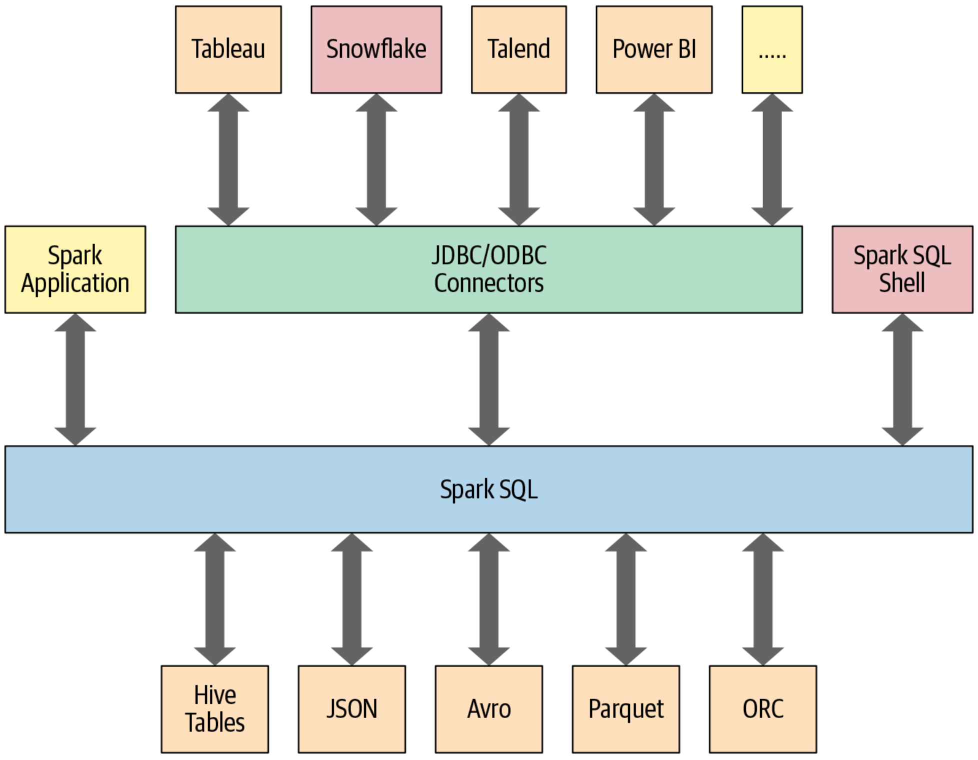
In past days users are limited to analyzing a sample of data on a single machine due to computational constraints. With big data support, better and more distributed computing, and frameworks like Apache Spark for big data processing and open-source analytics, we can perform scalable log analysis on potentially billions of calendar messages daily. The purpose of this study is to take an approach that shows how we can use Spark to perform scale log analyses on semi-structured log data. So, below we are seeing how to process data with Spark.

Spark SQL and DataFrames: Introduction to Built-in Data Sources

In this coding challenge, we use Spark and we make the data analysis with SQL queries. For this purpose, we use Spark SQL. Below we see, how Spark SQL interfaces with some of the external components.

In particular, Spark SQL:

- Provides the engine upon which the high-level Structured APIs.
- Can read and write data in a variety of structured formats (e.g., JSON, Hive tables, Parquet, Avro, ORC, CSV).
- Can query data using JDBC/ODBC connectors from external business intelligence (BI) data sources such as Tableau, Power BI, Talend, or from RDBMSs such as MySQL and PostgreSQL.
- Provides a practical interface to interact with structured data stored as tables or views in a database from a Spark application.
- Offers an interactive shell to issue SQL queries for structured data.
- Supports ANSI SQL:2003-compliant commands and HiveQL.



Spark SQL connectors and data sources

Below we begin with how we can use Spark SQL in a Spark application.

Using Spark SQL in Spark Applications

The SparkSession, introduced in Spark 2.0, provides a unified entry point for programming Spark with the Structured APIs. To issue any SQL query, we use the sql() method on
the SparkSession instance, spark, such as spark.sql("SELECT * FROM myTableName").
All spark.sql queries executed in this manner return a DataFrame on which we may perform further Spark operations.

Main objective: NASA log analytics

As we see above Apache Spark is an excellent and ideal open-source framework for wrangling, analyzing, and modeling structured and unstructured data at scale. In this code challenge, our main objective is to analyze log analytics. Server logs are a common enterprise data source and often including very useful information. Log data comes from many sources in these conditions, such as the web, client and compute servers, applications, user-generated content, and flat files. These logs can be used for monitoring servers, improving business and customer intelligence, building recommendation systems, fraud detection, and much more.

Spark allows us to store our logs into files on disk, while still providing rich APIs to perform data analysis at scale. This hands-on code challenge will show, how to use Apache Spark on real-world production logs from NASA while learning data wrangling and basic yet powerful techniques for exploratory data analysis. The full data set—containing two months' worth of all HTTP requests to the NASA Kennedy Space Center.

Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed, 167.8 MB uncompressed:
ftp://ita.ee.lbl.gov/traces/NASA_access_log_Aug95.gz

Exploring the dataset

Below we see and explore the NASA_access_log_Aug95 log file dataset. It is important before starting the processing and analysis of the dataset to be able to recognize the data according to manipulate them.

Below is an example.

127.0.0.1 – – [01/Aug/1995:00:00:01 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1839

Each part of this log entry is described below.

127.0.0.1: This is the IP address (or hostname, if available) of the client (remote host) that requested the server.

– The "hyphen" in the output indicates that the requested piece of information (user identity from remote machine) is not available.

[01/Aug/1995:00:00:01 -0400]: The time that the server finished processing the request.

The format is: [day/month/year:hour:minute:second timezone]

day = 2 digits
month = 3 letters
year = 4 digits
hour = 2 digits
minute = 2 digits
second = 2 digits
zone = (+ | -) 4 digits

“GET /images/launch-logo.gif HTTP/1.0”: This is the first line of the request string from the client. It consists of three components:

- the request method (e.g., GET, POST, etc.).
- the endpoint (a Uniform Resource Identifier).
- the client protocol version.

200: This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2), a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5).

1839: The last entry indicates the size of the object returned to the client, not including the response headers. If no content was returned to the client, this value will be “-” (or sometimes 0).

4. Setting up dependencies

In this chapter, we will see the necessary components and requirements for running the queries. The hardware configuration and operating system we use are the below:

- CPU: AMD Ryzen7 3700x 8-core processor x 16
- RAM: 32GB
- OS: Ubuntu 20.04 LTS

DOWNLOADING ANACONDA AND INSTALLING PYSPARK

Below are the necessary installations and dependencies.

The first step is to download Anaconda. After the suitable Anaconda version is downloaded, we click on it to proceed with the installation procedure which is explained step by step in the Anaconda Documentation.

When the installation is completed, the Anaconda Navigator Homepage will be opened. To use Python, we put in the terminal the below command.

```
anaconda-navigator
```

To be able to use Spark through Anaconda, the following package installation steps shall be followed.

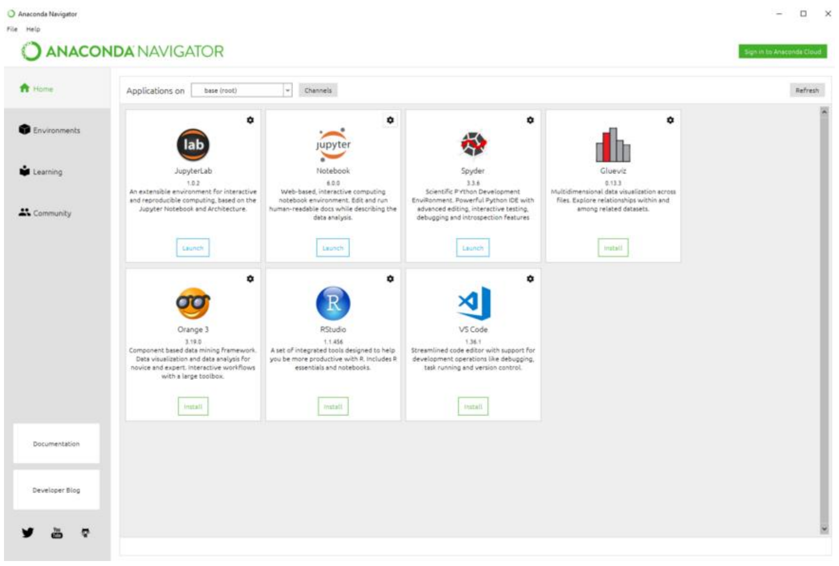
Anaconda Prompt terminal

```
conda install -c conda-forge pyspark
conda install -c conda-forge pyarrow
```

After PySpark and pyarrow packages installations are completed, we close the terminal and go back to Jupyter Notebook and import the required packages at the top of our code.

Also, we can install PySpark and pyspark SQL using PyPI in jupyter is as follows:

```
pip install pyspark
pip install pyspark[sql]
```



JupyterLab
1.0.2
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.0.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Spyder
3.3.6
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.19.0
Component based data mining Framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

VS Code
1.36.1
Streamlined code editor with support for development operations like debugging, task running and version control.

Anaconda Navigator Home Page

After that, we must make sure we have access to a Spark session and cluster. For this step, we can use our local Spark setup or a cloud-based setup.

Often pre-configured Spark setups already have the necessary environment variables or dependencies pre-loaded when we start the Jupyter Notebook server. So, we can check them using the following commands:

spark

These results show us that our cluster is running Spark 2.4.0 at the moment. We can also check if sqlContext is presently using the following code:

sqlContext

<pyspark.sql.context.SQLContext at 0x7fb1577b6400>

Now in case we do not have these variables pre-configured and get an error, we can load and configure them using the following code:

```python
# import spark liriaries
from pyspark.context import SparkContext
from pyspark.sql.context import SQLContext
from pyspark.sql.session import SparkSession
from pyspark.sql.functions import regexp_extract
from pyspark.sql.functions import col
from pyspark.sql.functions import sum as spark_sum
from pyspark.sql.functions import desc, row_number, monotonically_increasing_id
from pyspark.sql.window import Window
from pyspark.sql.functions import udf

# load up other dependencies
import re
import glob
#import optparse
```

```python
# configure spark variables
sc = SparkContext()
sqlContext = SQLContext(sc)
spark = SparkSession(sc)
```

In this case, we must mention that we keeping the non-standard library dependencies to a minimum (less than 5). So, as we can see from the above libraries, most of them are standard dependencies from the pyspark library and we use only two from other libraries. We use only three separate libraries.

5. Loading and viewing the NASA Log Dataset

In this chapter, we see how to loading and viewing the NASA Log Dataset. Below we can see step by step the process and the code we use as well as the results.

DATA LOADING

We start the coding challenge by loading the NASA log dataset. The following code downloads and inserts the dataset:

```
# Download NASA_access_log_Aug95.gz file
get_ipython().system(' wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Aug95.gz')
```

```
--2021-10-16 12:55:36--  ftp://ita.ee.lbl.gov/traces/NASA_access_log_Aug95.gz
           => ‘NASA_access_log_Aug95.gz.1’
Resolving ita.ee.lbl.gov (ita.ee.lbl.gov)... 131.243.2.164, 2620:83:8000:102::a4
Connecting to ita.ee.lbl.gov (ita.ee.lbl.gov)|131.243.2.164|:21... connected.
Logging in as anonymous ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD (1) /traces ... done.
==> SIZE NASA_access_log_Aug95.gz ... 16633316
==> PASV ... done.    ==> RETR NASA_access_log_Aug95.gz ... done.
Length: 16633316 (16M) (unauthoritative)

NASA_access_log_Aug 100%[===================>]  15,86M  3,10MB/s    in 6,7s

2021-10-16 12:55:46 (2,35 MB/s) - ‘NASA_access_log_Aug95.gz.1’ saved [16633316]
```

```
#insert data
raw_data_files = glob.glob('*.gz')
raw_data_files
```

```
['NASA_access_log_Aug95.gz']
```

Now, we will use spark.read.text() to read the text file. This code produces a Data Frame with a single string column called value:

```
#produces a DataFrame with a single string column called value:
base_df = spark.read.text(raw_data_files)
base_df.printSchema()
```

```
root
 |-- value: string (nullable = true)
```

In this case, we use Spark Data Frames. However, we can also convert a Data Frame into a Resilient Distributed Dataset (RDD) which is Spark’s original data structure:

```
#convert a DataFrame into a Resilient Distributed Dataset (RDD)—Spark’s original data structure
base_df_rdd = base_df.rdd
```

Data parsing and extraction with regular expressions

Next, we have to parse our semi-structured log data into individual columns. We will use the special built-in regexp_extract() function to do the parsing. This function matches a column against a regular expression with one or more capture groups and allows us to extract one of the matched groups. We will use one regular expression for each field we wish to extract.

In the below code we can see the total number of logs we are working on within our dataset:

```
#the total number of logs of the dataset
print((base_df.count(), len(base_df.columns)))
```

```
(1569898, 1)
```

We have a total of approximately 1.56 million log messages. Also in the below code, we can see some sample log messages.

```
#extract and have a look of some sample log messages
sample_logs = [item['value'] for item in base_df.take(15)]
sample_logs
```

```
['in24.inetnebr.com - - [01/Aug/1995:00:00:01 -0400] "GET /shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0" 200 1839',
 'uplherc.upl.com - - [01/Aug/1995:00:00:07 -0400] "GET / HTTP/1.0" 304 0',
 'uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/ksclogo-medium.gif HTTP/1.0" 304 0',
 'uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/MOSAIC-logosmall.gif HTTP/1.0" 304 0',
 'uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/USA-logosmall.gif HTTP/1.0" 304 0',
 'ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:09 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1713',
 'uplherc.upl.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/WORLD-logosmall.gif HTTP/1.0" 304 0',
 'slppp6.intermind.net - - [01/Aug/1995:00:00:10 -0400] "GET /history/skylab/skylab.html HTTP/1.0" 200 1687',
 'piweba4y.prodigy.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/launchmedium.gif HTTP/1.0" 200 11853',
 'slppp6.intermind.net - - [01/Aug/1995:00:00:11 -0400] "GET /history/skylab/skylab-small.gif HTTP/1.0" 200 9202',
 'slppp6.intermind.net - - [01/Aug/1995:00:00:12 -0400] "GET /images/ksclogosmall.gif HTTP/1.0" 200 3635',
 'ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:12 -0400] "GET /history/apollo/images/apollo-logo1.gif HTTP/1.0" 200 117
3',
 'slppp6.intermind.net - - [01/Aug/1995:00:00:13 -0400] "GET /history/apollo/images/apollo-logo.gif HTTP/1.0" 200 3047',
 'uplherc.upl.com - - [01/Aug/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0',
 '133.43.96.45 - - [01/Aug/1995:00:00:16 -0400] "GET /shuttle/missions/sts-69/mission-sts-69.html HTTP/1.0" 200 10566']
```

Extracting hostnames

In this step, with regular expressions we extract the hostname from the logs:

```
# Extracting hostnames - with regular expressions extract the hostname from the logs
host_pattern = r'(^\S+\.[\S+\.]+\S+)\s'
hosts = [re.search(host_pattern, item).group(1)
           if re.search(host_pattern, item)
           else 'no match'
           for item in sample_logs]
```

Extracting timestamps

In this step, with regular expressions we extract the timestamp fields from the logs:

```
# Extracting timestamps - with regular expressions extract the timestamp fields from the logs
ts_pattern = r'\[(\d{2}/\w{3}/\d{4}:\d{2}:\d{2}:\d{2} -\d{4})]'
timestamps = [re.search(ts_pattern, item).group(1) for item in sample_logs]
```

Extracting HTTP request method, URIs, and protocol

In this step, with regular expressions we extract the HTTP request methods, URIs, and Protocol patterns fields from the logs:

```
# Extracting HTTP request method, URIs, and protocol -
# with regular expressions extract the HTTP request methods, URIs, and Protocol patterns fields from the logs
method_uri_protocol_pattern = r'\"(\S+)\s(\S+)\s*(\S*)\"'
method_uri_protocol = [re.search(method_uri_protocol_pattern, item).groups()
               if re.search(method_uri_protocol_pattern, item)
               else 'no match'
              for item in sample_logs]
```

Extracting HTTP status codes

In this step, with regular expressions we extract the HTTP status codes from the logs:

```
# Extracting HTTP status codes - with regular expressions extract the HTTP status codes from the logs
status_pattern = r'\s(\d{3})\s'
status = [re.search(status_pattern, item).group(1) for item in sample_logs]
```

Extracting HTTP response content size

In this step, with regular expressions we extract the HTTP response content size from the logs:

```
# Extracting HTTP status codes - with regular expressions extract the HTTP status codes from the logs
status_pattern = r'\s(\d{3})\s'
status = [re.search(status_pattern, item).group(1) for item in sample_logs]
```

Putting it all together

Now we leverage all the regular expression patterns we previously built and use the regexp_extract(...) method to build our Data Frame with all of the logs attributes neatly extracted in their separate columns.

```
# Putting it all together -
# We build our DataFrame with all of the log attributes neatly extracted in their own separate columns
logs_df = base_df.select(regexp_extract('value', host_pattern, 1).alias('host'),
                         regexp_extract('value', ts_pattern, 1).alias('timestamp'),
                         regexp_extract('value', method_uri_protocol_pattern, 1).alias('method'),
                         regexp_extract('value', method_uri_protocol_pattern, 2).alias('endpoint'),
                         regexp_extract('value', method_uri_protocol_pattern, 3).alias('protocol'),
                         regexp_extract('value', status_pattern, 1).cast('integer').alias('status'),
                         regexp_extract('value', content_size_pattern, 1).cast('integer').alias('content_size'))
logs_df.show(10, truncate=True)
print((logs_df.count(), len(logs_df.columns)))
```

```
+--------------------+--------------------+------+--------------------+--------+------+------------+
|                host|           timestamp|method|            endpoint|protocol|status|content_size|
+--------------------+--------------------+------+--------------------+--------+------+------------+
|    in24.inetnebr.com|01/Aug/1995:00:00...|   GET|/shuttle/missions...|HTTP/1.0|   200|        1839|
|      uplherc.upl.com|01/Aug/1995:00:00...|   GET|                   /|HTTP/1.0|   304|           0|
|      uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/ksclogo-m...|HTTP/1.0|   304|           0|
|      uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/MOSAIC-lo...|HTTP/1.0|   304|           0|
|      uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/USA-logos...|HTTP/1.0|   304|           0|
|ix-esc-ca2-07.ix....|01/Aug/1995:00:00...|   GET|/images/launch-lo...|HTTP/1.0|   200|        1713|
|      uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/WORLD-log...|HTTP/1.0|   304|           0|
|slppp6.intermind.net|01/Aug/1995:00:00...|   GET|/history/skylab/s...|HTTP/1.0|   200|        1687|
|piweba4y.prodigy.com|01/Aug/1995:00:00...|   GET|/images/launchmed...|HTTP/1.0|   200|       11853|
|slppp6.intermind.net|01/Aug/1995:00:00...|   GET|/history/skylab/s...|HTTP/1.0|   200|        9202|
+--------------------+--------------------+------+--------------------+--------+------+------------+
only showing top 10 rows

(1569898, 7)
```

As we can see from the above schema, we have a clear vision of the columns of the dataset.

Putting index column

For the code challenge is necessary to add an extra index column as we can see below:

```
# Putting index column
df_with_seq_id = logs_df.withColumn('index', row_number().over(Window.orderBy(monotonically_increasing_id())) - 1)
df_with_seq_id.show()
```

```
+--------------------+--------------------+------+--------------------+--------+------+------------+-----+
|                host|           timestamp|method|            endpoint|protocol|status|content_size|index|
+--------------------+--------------------+------+--------------------+--------+------+------------+-----+
|   in24.inetnebr.com|01/Aug/1995:00:00...|   GET|/shuttle/missions...|HTTP/1.0|   200|        1839|    0|
|     uplherc.upl.com|01/Aug/1995:00:00...|   GET|                   /|HTTP/1.0|   304|           0|    1|
|     uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/ksclogo-m...|HTTP/1.0|   304|           0|    2|
|     uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/MOSAIC-lo...|HTTP/1.0|   304|           0|    3|
|     uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/USA-logos...|HTTP/1.0|   304|           0|    4|
|ix-esc-ca2-07.ix....|01/Aug/1995:00:00...|   GET|/images/launch-lo...|HTTP/1.0|   200|        1713|    5|
|     uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/WORLD-log...|HTTP/1.0|   304|           0|    6|
|slppp6.intermind.net|01/Aug/1995:00:00...|   GET|/history/skylab/s...|HTTP/1.0|   200|        1687|    7|
|piweba4y.prodigy.com|01/Aug/1995:00:00...|   GET|/images/launchmed...|HTTP/1.0|   200|       11853|    8|
|slppp6.intermind.net|01/Aug/1995:00:00...|   GET|/history/skylab/s...|HTTP/1.0|   200|        9202|    9|
|slppp6.intermind.net|01/Aug/1995:00:00...|   GET|/images/ksclogosm...|HTTP/1.0|   200|        3635|   10|
|ix-esc-ca2-07.ix....|01/Aug/1995:00:00...|   GET|/history/apollo/i...|HTTP/1.0|   200|        1173|   11|
|slppp6.intermind.net|01/Aug/1995:00:00...|   GET|/history/apollo/i...|HTTP/1.0|   200|        3047|   12|
|     uplherc.upl.com|01/Aug/1995:00:00...|   GET|/images/NASA-logo...|HTTP/1.0|   304|           0|   13|
|        133.43.96.45|01/Aug/1995:00:00...|   GET|/shuttle/missions...|HTTP/1.0|   200|       10566|   14|
|kgtyk4.kj.yamagat...|01/Aug/1995:00:00...|   GET|                   /|HTTP/1.0|   200|        7280|   15|
|kgtyk4.kj.yamagat...|01/Aug/1995:00:00...|   GET|/images/ksclogo-m...|HTTP/1.0|   200|        5866|   16|
|     d0ucr6.fnal.gov|01/Aug/1995:00:00...|   GET|/history/apollo/a...|HTTP/1.0|   200|        2743|   17|
|ix-esc-ca2-07.ix....|01/Aug/1995:00:00...|   GET|/shuttle/resource...|HTTP/1.0|   200|        6849|   18|
|     d0ucr6.fnal.gov|01/Aug/1995:00:00...|   GET|/history/apollo/a...|HTTP/1.0|   200|       14897|   19|
+--------------------+--------------------+------+--------------------+--------+------+------------+-----+
only showing top 20 rows
```

Finding missing values

Missing and null values are the bane of data analysis. Below we see, how well our data parsing and extraction logic worked. If our data parsing and extraction worked properly, we should not have any rows with potential null values:

```
# rows with potential null or missing values.
bad_rows_df = df_with_seq_id.filter(logs_df['host'].isNull()|
                             df_with_seq_id['timestamp'].isNull() |
                             df_with_seq_id['method'].isNull() |
                             df_with_seq_id['endpoint'].isNull() |
                             df_with_seq_id['status'].isNull() |
                             df_with_seq_id['content_size'].isNull()|
                             df_with_seq_id['protocol'].isNull())
bad_rows_df.count()
```

```
14178
```

As we can see the dataset, contains over 14K malformed entries. Below we will see how to fix them.

Finding null counts

We can typically use the following technique to find out which columns have null values.

```
# find out which columns contains malformed entries.
def count_null(col_name):
    return spark_sum(col(col_name).isNull().cast('integer')).alias(col_name)

# Build up a list of column expressions, one per column.
exprs = [count_null(col_name) for col_name in df_with_seq_id.columns]

# Run the aggregation. The *exprs converts the list of expressions into
# variable function arguments.
bad_rows_df.agg(*exprs).show()
```

```
+----+---------+------+--------+--------+------+------------+-----+
|host|timestamp|method|endpoint|protocol|status|content_size|index|
+----+---------+------+--------+--------+------+------------+-----+
|   0|        0|     0|       0|       0|     0|       14178|    0|
+----+---------+------+--------+--------+------+------------+-----+
```

As we can see we have 14178 missing values in the content_size column. The reason is the content size in the dataset has two characters “0” and “-”. So, in the case of “-” the result is null in our dataset.

Below, we will see how to handle also null values not only in the content_size column but also in other columns in case we need to make any process in the future.

HANDLING NULLS IN HTTP STATUS

Below we see how to handle null values in status columns in case we needed them.

Note: In the expression below, the tilde (**~**) means “not”.

```
# Handling nulls in HTTP status
regexp_extract('value', r'\s(\d{3})\s', 1).cast('integer').alias( 'status')
null_status_df = base_df.filter(~base_df['value'].rlike(r'\s(\d{3})\s'))
```

Pass this through our log data parsing pipeline:

```
# pass this through the log data parsing pipeline
bad_status_df = null_status_df.select(regexp_extract('value', host_pattern, 1).alias('host'),
                                      regexp_extract('value', ts_pattern, 1).alias('timestamp'),
                                      regexp_extract('value', method_uri_protocol_pattern, 1).alias('method'),
                                      regexp_extract('value', method_uri_protocol_pattern, 2).alias('endpoint'),
                                      regexp_extract('value', method_uri_protocol_pattern, 3).alias('protocol'),
                                      regexp_extract('value', status_pattern, 1).cast('integer').alias('status'),
                                      regexp_extract('value', content_size_pattern, 1).cast('integer').alias('content_size'))
```

HANDLING NULLS IN HTTP CONTENT SIZE

Below we see how to handle null values in content_size column. We first find the records with potential missing content sizes in our base Data Frame:

```
# Handling nulls in HTTP content size
logs_df = df_with_seq_id[logs_df['status'].isNotNull()]
exprs = [count_null(col_name) for col_name in logs_df.columns]
logs_df.agg(*exprs).show()
```

```
+----+---------+------+--------+--------+------+------------+-----+
|host|timestamp|method|endpoint|protocol|status|content_size|index|
+----+---------+------+--------+--------+------+------------+-----+
|   0|        0|     0|       0|       0|     0|       14178|    0|
+----+---------+------+--------+--------+------+------------+-----+
```

It is quite evident that the bad raw data records correspond to error responses, where no content was sent back and the server emitted a - for the **content_size** field. Since we don’t want to discard those rows from our analysis, let’s impute or fill them with 0.

FIX THE ROWS WITH NULL CONTENT_SIZE

The easiest solution is to replace the null values in logs_df with 0. The Spark DataFrame API provides a set of functions and fields specifically designed for working with null values, among them:

```
# find the records with potential missing content sizes in our base DataFrame
null_content_size_df = base_df.filter(~base_df['value'].rlike(r'\s\d+$'))
null_content_size_df.count()
```

```
14178
```

fillna(), which fills null values with specified non-null values. na, which returns
a DataFrameNaFunctions object with many functions for operating on null columns.

There are several ways to invoke this function. The easiest is just to replace all null columns with known values. Also, it is better to pass a Python dictionary containing (column_name, value) mappings. Now we use this function to fill all the missing values in the content_size field with 0:

```
# Fix the rows with null content_size -  replace the null values in logs_df with 0
logs_df = logs_df.na.fill({'content_size': 0})
exprs = [count_null(col_name) for col_name in logs_df.columns]
# the missing values in the content_size field with 0
logs_df.agg(*exprs).show()
```

```
+----+---------+------+--------+--------+------+------------+-----+
|host|timestamp|method|endpoint|protocol|status|content_size|index|
+----+---------+------+--------+--------+------+------------+-----+
|   0|        0|     0|       0|       0|     0|           0|    0|
+----+---------+------+--------+--------+------+------------+-----+
```

HANDLING TEMPORAL FIELDS (TIMESTAMP)

Now that we have a clean, parsed DataFrame, we have to parse the timestamp field into an actual timestamp. The Common Log Format time is somewhat non-standard. A User-Defined Function (UDF) is the most straightforward way to parse it:

```
# Handling temporal fields (timestamp) - parse the timestamp field into an actual timestamp.
month_map = {
  'Jan': 1, 'Feb': 2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6, 'Jul':7,
  'Aug':8,  'Sep': 9, 'Oct':10, 'Nov': 11, 'Dec': 12
}

def parse_clf_time(text):
    """ Convert Common Log time format into a Python datetime object
    Args:
        text (str): date and time in Apache time format [dd/mmm/yyyy:hh:mm:ss (+/-)zzzz]
    Returns:
        a string suitable for passing to CAST('timestamp')
    """
    # NOTE: We're ignoring the time zones here, might need to be handled depending on the problem you are solving
    return "{0:04d}-{1:02d}-{2:02d} {3:02d}:{4:02d}:{5:02d}".format(
      int(text[7:11]),
      month_map[text[3:6]],
      int(text[0:2]),
      int(text[12:14]),
      int(text[15:17]),
      int(text[18:20])
    )
```

We see how to use this function to parse our DataFrame's time column:

```
# parse our DataFrame's time column.
udf_parse_time = udf(parse_clf_time)

logs_df = (logs_df.select('*', udf_parse_time(logs_df['timestamp'])
                                  .cast('timestamp')
                                  .alias('time'))
                  .drop('timestamp'))
```

Also, we verify by checking our DataFrame's schema:

```
# verify by checking the DataFrame's schema.
logs_df.cache()
```

```
DataFrame[host: string, method: string, endpoint: string, protocol: string, status: int, content_size: int, index: int, time: t
imestamp]
```

Once we have prepared a clean dataset, we can finally start using it to gain useful insights about NASA servers.

6. How to analyze log data with Python and Apache Spark

In the previous chapter, we began by using Python and Apache Spark to process and wrangle our example weblogs into a format fit for analysis. We set up environment variables, dependencies, loaded the necessary libraries for working with both Data Frames and regular expressions, and of course loaded the example log data. Then we wrangled our log data into a clean, structure, and meaningful format. In this chapter, we focus on analyzing that data.

Data analysis on weblogs

Now that we have a DataFrame containing the parsed and cleaned log file as a data frame, we can perform some interesting exploratory data analysis.

Initializing Spark Session

First of all, a Spark session needs to be initialized. With the help of SparkSession, DataFrame can be created and registered as tables. Moreover, SQL tables are executed, tables can be cached, and parquet/JSON/CSV/Avro data formatted files can be read.

```
# Initializing SparkSession
sc = SparkSession.builder.appName("PysparkExample").config ("spark.sql.shuffle.partitions", "50").config("spark.driver.maxResult
Size","5g").config ("spark.sql.execution.arrow.enabled", "true").getOrCreate()
```

Running SQL Queries in Spark

Raw SQL queries can also be used by enabling the “SQL” operation on our SparkSession to run SQL queries and return the result sets as Data Frame structures. To proceed to that we must register a table first:

```
# Registering a table
logs_df.registerTempTable("data_table")
```

Challenge Code Questions

In this section, we see the challenge code questions. How to make them and the results.

1. Top 10 requested pages and the number of requests made for each

To make this question we consider the endpoint column as pages. Also, we put as fitter the “GET” in the method column because we want to receive the requested results. Finally, we order by the requests as desc according to receive a count list of the most requests.

```
print ("q1: Top 10 requested pages and the number of requests made for each",'\n')

sql = """
select endpoint as pages, count (endpoint) as requests
from data_table
where method = 'GET'
group by endpoint
order by requests desc
limit 10
"""

spark.sql(sql).show()
```

```
q1: Top 10 requested pages and the number of requests made for each

+--------------------+--------+
|               pages|requests|
+--------------------+--------+
|/images/NASA-logo...|   96963|
|/images/KSC-logos...|   75192|
|/images/MOSAIC-lo...|   67062|
|/images/USA-logos...|   66691|
|/images/WORLD-log...|   66072|
|/images/ksclogo-m...|   62405|
|           /ksc.html|   43457|
|/history/apollo/i...|   37772|
|/images/launch-lo...|   35082|
|                   /|   30097|
+--------------------+--------+
```

2. Percentage of successful requests (anything in the 200s and 300s range)

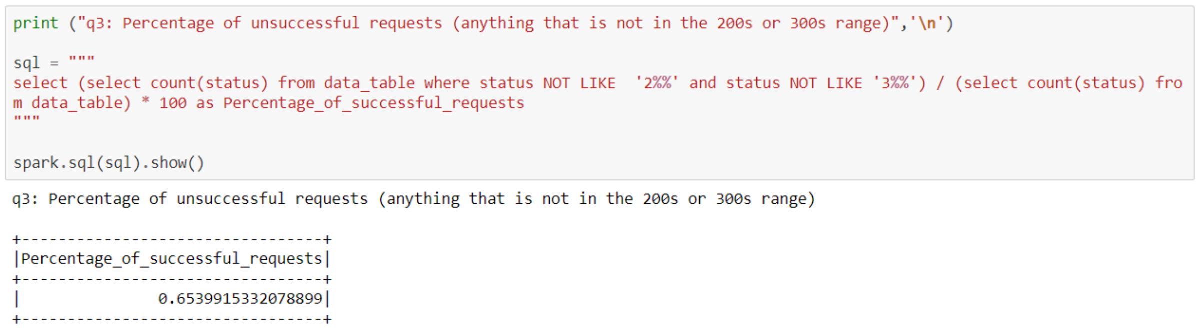
In this query, we use as a filter any status is beginning with 2 or 3. According to receive the percentage we divine with the total status count. The result is 99.34 % successful requests.

```
print ("q2: Percentage of successful requests (anything in the 200s and 300s range)",'\n')

sql = """
select (select count(status) from data_table where status like '2%%' or status like '3%%') / (select count(status) from data_tab
le) * 100 as Percentage_of_successful_requests
"""

spark.sql(sql).show()
```

```
q2: Percentage of successful requests (anything in the 200s and 300s range)

+---------------------------------+
|Percentage_of_successful_requests|
+---------------------------------+
|                99.34600846679211|
+---------------------------------+
```

3. Percentage of unsuccessful requests (anything that is not in the 200s or 300s range)

In this query, we use as a filter any status is not beginning with 2 or 3. According to receive the percentage we divine with the total status count. The result is 0.65 % successful requests.

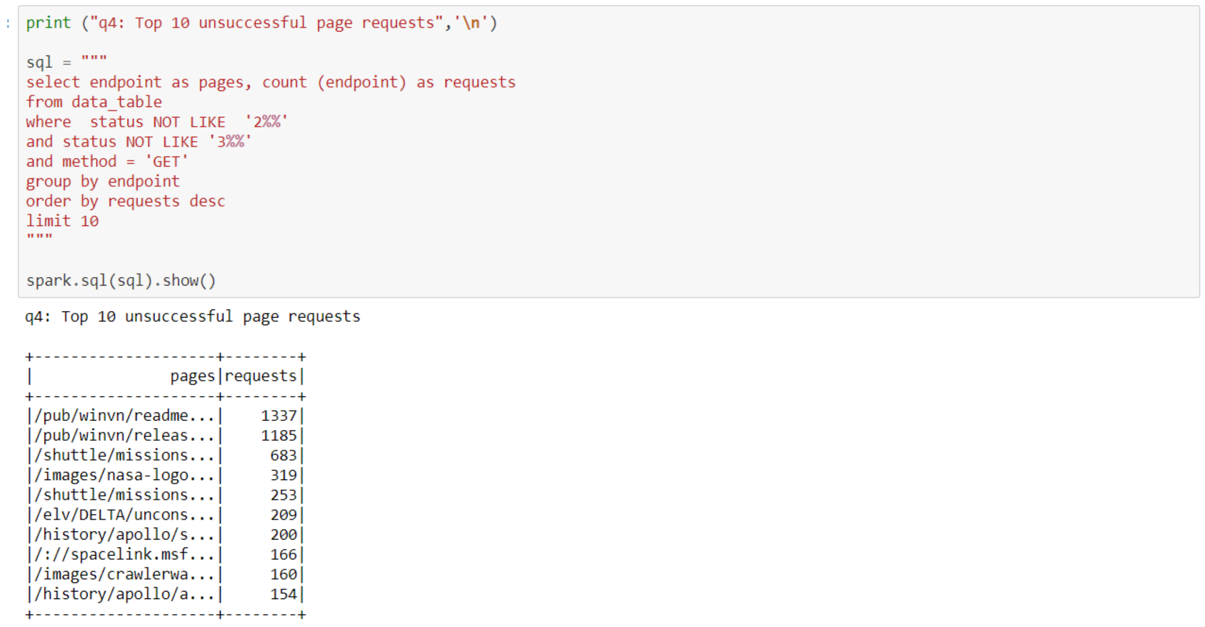


```
print ("q3: Percentage of unsuccessful requests (anything that is not in the 200s or 300s range)",'\n')

sql = """
select (select count(status) from data_table where status NOT LIKE  '2%%' and status NOT LIKE '3%%') / (select count(status) fro
m data_table) * 100 as Percentage_of_successful_requests
"""

spark.sql(sql).show()
```

```
q3: Percentage of unsuccessful requests (anything that is not in the 200s or 300s range)

+---------------------------------+
|Percentage_of_successful_requests|
+---------------------------------+
|               0.6539915332078899|
+---------------------------------+
```

4. Top 10 unsuccessful page requests

In this query, we use as a filter any status is not beginning with 2 or 3. Also, we put as fitter the "GET" in the method column because we want to receive the requested results. Finally, we order by the requests as desc according to receive a count list of the most page requests.

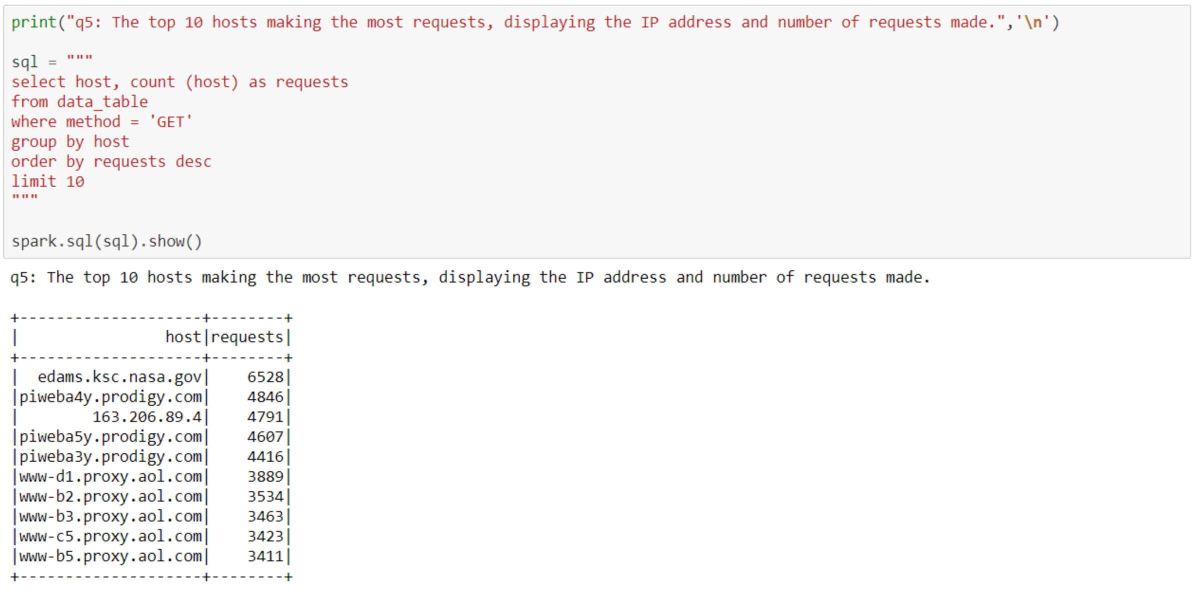


```
print ("q4: Top 10 unsuccessful page requests",'\n')

sql = """
select endpoint as pages, count (endpoint) as requests
from data_table
where  status NOT LIKE  '2%%'
and status NOT LIKE '3%%'
and method = 'GET'
group by endpoint
order by requests desc
limit 10
"""

spark.sql(sql).show()
```

```
q4: Top 10 unsuccessful page requests

+--------------------+--------+
|               pages|requests|
+--------------------+--------+
|/pub/winvn/readme...|    1337|
|/pub/winvn/releas...|    1185|
|/shuttle/missions...|     683|
|/images/nasa-logo...|     319|
|/shuttle/missions...|     253|
|/elv/DELTA/uncons...|     209|
|/history/apollo/s...|     200|
|/://spacelink.msf...|     166|
|/images/crawlerwa...|     160|
|/history/apollo/a...|     154|
+--------------------+--------+
```

5. THE TOP 10 HOSTS MAKING THE MOST REQUESTS, DISPLAYING THE IP ADDRESS AND A NUMBER OF REQUESTS MADE.

This query is similar to the first question but we select the column host.

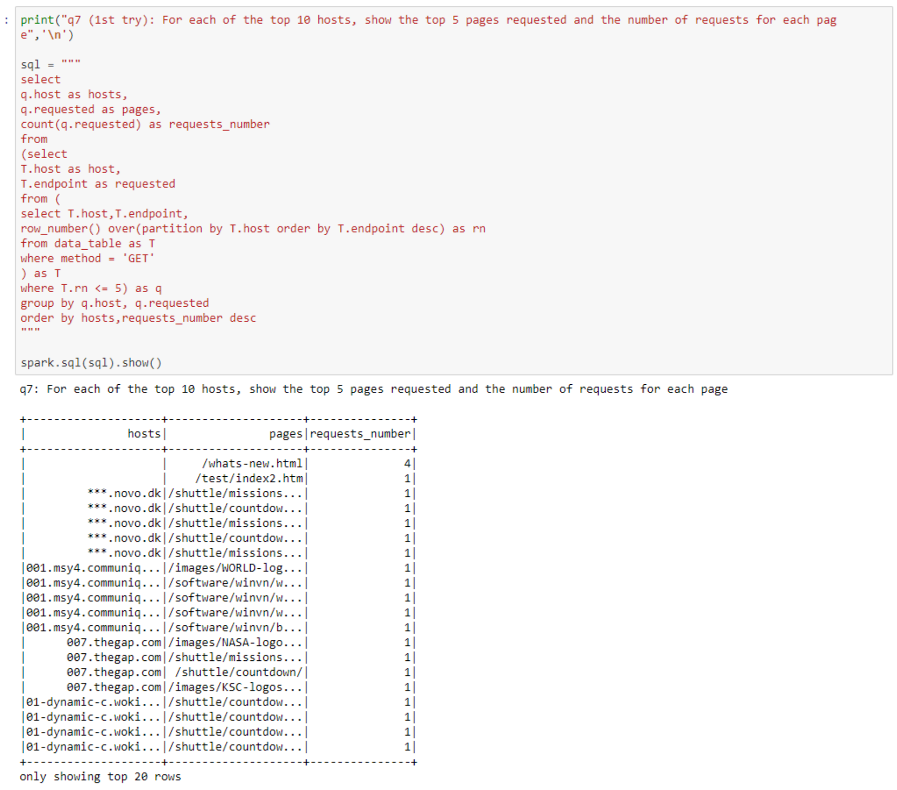


```
print("q5: The top 10 hosts making the most requests, displaying the IP address and number of requests made.",'\n')

sql = """
select host, count (host) as requests
from data_table
where method = 'GET'
group by host
order by requests desc
limit 10
"""

spark.sql(sql).show()
```

```
q5: The top 10 hosts making the most requests, displaying the IP address and number of requests made.

+--------------------+--------+
|                host|requests|
+--------------------+--------+
|   edams.ksc.nasa.gov|    6528|
|piweba4y.prodigy.com|    4846|
|        163.206.89.4|    4791|
|piweba5y.prodigy.com|    4607|
|piweba3y.prodigy.com|    4416|
|www-d1.proxy.aol.com|    3889|
|www-b2.proxy.aol.com|    3534|
|www-b3.proxy.aol.com|    3463|
|www-c5.proxy.aol.com|    3423|
|www-b5.proxy.aol.com|    3411|
+--------------------+--------+
```

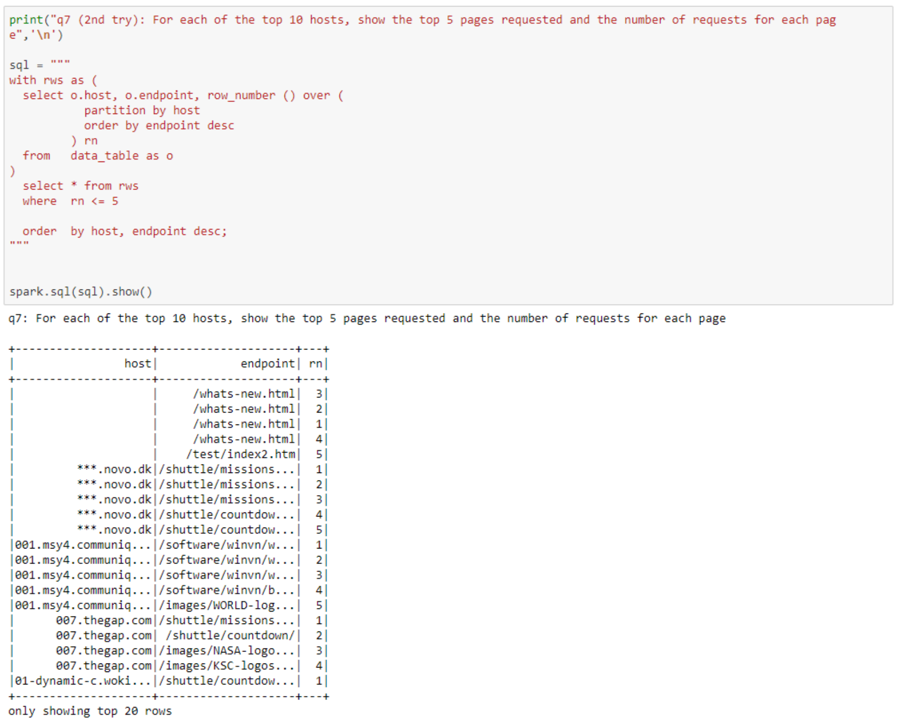
6. For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each page

In this question, we try to show for each of the top 10 hosts the top 5 pages requested. As we can see from the results, we can see the top 5 requested pages of the hosts, but not in the top 10 hosts. Making a comparison separately below, we see that the results of page requests and hosts requests are the same.



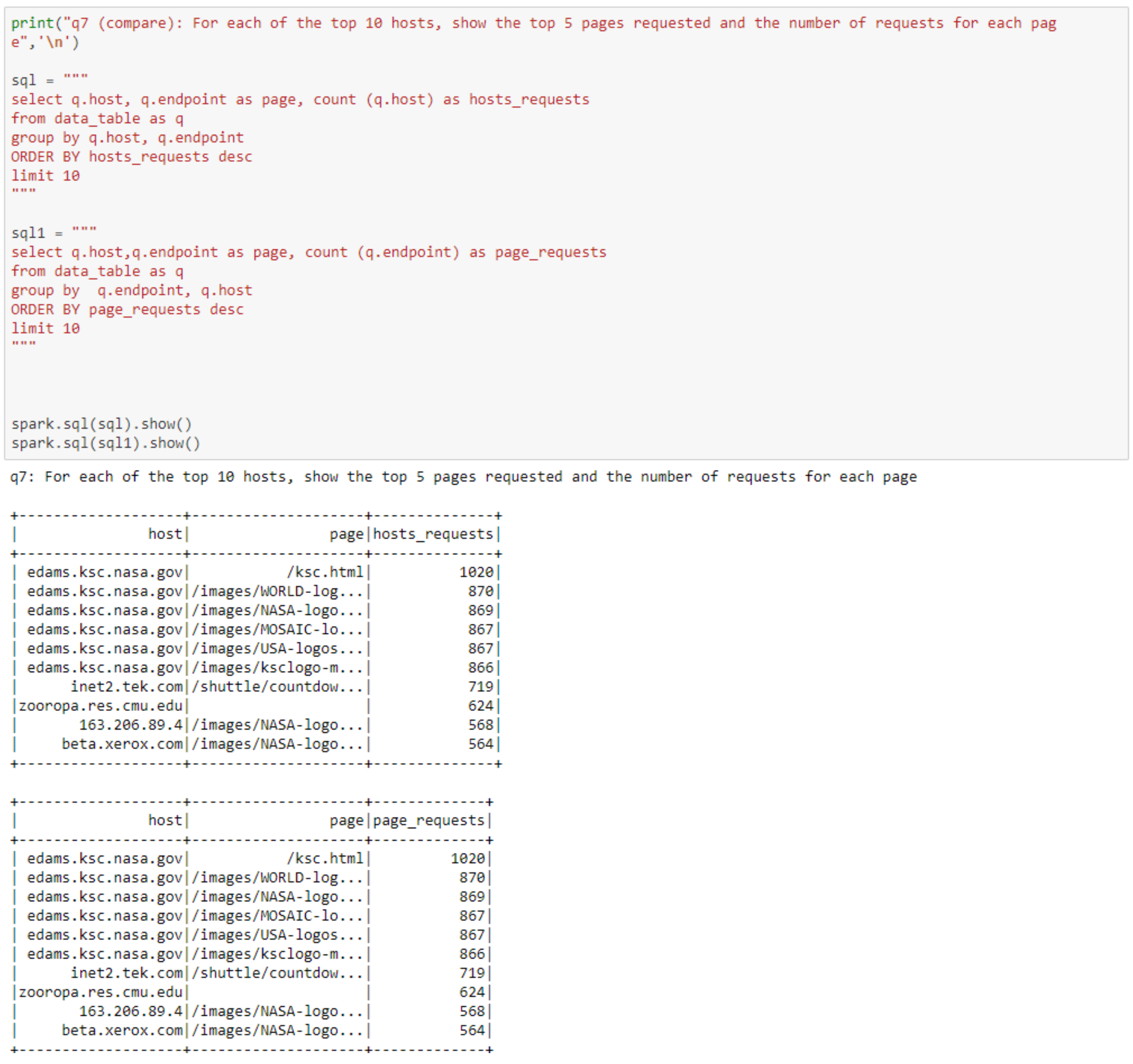
```
print("q7 (1st try): For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each pag
e",'\n')

sql = """
select
q.host as hosts,
q.requested as pages,
count(q.requested) as requests_number
from
(select
T.host as host,
T.endpoint as requested
from (
select T.host,T.endpoint,
row_number() over(partition by T.host order by T.endpoint desc) as rn
from data_table as T
where method = 'GET'
) as T
where T.rn <= 5) as q
group by q.host, q.requested
order by hosts,requests_number desc
"""

spark.sql(sql).show()
```

```
q7: For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each page

+--------------------+--------------------+---------------+
|               hosts|               pages|requests_number|
+--------------------+--------------------+---------------+
|                    |      /whats-new.html|              4|
|                    |     /test/index2.htm|              1|
|          ***.novo.dk|/shuttle/missions...|              1|
|          ***.novo.dk|/shuttle/countdow...|              1|
|          ***.novo.dk|/shuttle/missions...|              1|
|          ***.novo.dk|/shuttle/countdow...|              1|
|          ***.novo.dk|/shuttle/missions...|              1|
|001.msy4.communiq...|/images/WORLD-log...|              1|
|001.msy4.communiq...|/software/winvn/w...|              1|
|001.msy4.communiq...|/software/winvn/w...|              1|
|001.msy4.communiq...|/software/winvn/w...|              1|
|001.msy4.communiq...|/software/winvn/b...|              1|
|       007.thegap.com|/images/NASA-logo...|              1|
|       007.thegap.com|/shuttle/missions...|              1|
|       007.thegap.com| /shuttle/countdown/|              1|
|       007.thegap.com|/images/KSC-logos...|              1|
|01-dynamic-c.woki...|/shuttle/countdow...|              1|
|01-dynamic-c.woki...|/shuttle/countdow...|              1|
|01-dynamic-c.woki...|/shuttle/countdow...|              1|
|01-dynamic-c.woki...|/shuttle/countdow...|              1|
+--------------------+--------------------+---------------+
only showing top 20 rows
```



```
print("q7 (2nd try): For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each pag
e",'\n')

sql = """
with rws as (
  select o.host, o.endpoint, row_number () over (
           partition by host
           order by endpoint desc
         ) rn
  from   data_table as o
)
  select * from rws
  where  rn <= 5

  order  by host, endpoint desc;
"""

spark.sql(sql).show()
```

```
q7: For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each page

+--------------------+--------------------+---+
|                host|            endpoint| rn|
+--------------------+--------------------+---+
|                    |      /whats-new.html|  3|
|                    |      /whats-new.html|  2|
|                    |      /whats-new.html|  1|
|                    |      /whats-new.html|  4|
|                    |     /test/index2.htm|  5|
|          ***.novo.dk|/shuttle/missions...|  1|
|          ***.novo.dk|/shuttle/missions...|  2|
|          ***.novo.dk|/shuttle/missions...|  3|
|          ***.novo.dk|/shuttle/countdow...|  4|
|          ***.novo.dk|/shuttle/countdow...|  5|
|001.msy4.communiq...|/software/winvn/w...|  1|
|001.msy4.communiq...|/software/winvn/w...|  2|
|001.msy4.communiq...|/software/winvn/w...|  3|
|001.msy4.communiq...|/software/winvn/b...|  4|
|001.msy4.communiq...|/images/WORLD-log...|  5|
|       007.thegap.com|/shuttle/missions...|  1|
|       007.thegap.com| /shuttle/countdown/|  2|
|       007.thegap.com|/images/NASA-logo...|  3|
|       007.thegap.com|/images/KSC-logos...|  4|
|01-dynamic-c.woki...|/shuttle/countdow...|  1|
+--------------------+--------------------+---+
only showing top 20 rows
```



```
print("q7 (compare): For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each pag
e",'\n')

sql = """
select q.host, q.endpoint as page, count (q.host) as hosts_requests
from data_table as q
group by q.host, q.endpoint
ORDER BY hosts_requests desc
limit 10
"""

sql1 = """
select q.host,q.endpoint as page, count (q.endpoint) as page_requests
from data_table as q
group by  q.endpoint, q.host
ORDER BY page_requests desc
limit 10
"""


spark.sql(sql).show()
spark.sql(sql1).show()
```

```
q7: For each of the top 10 hosts, show the top 5 pages requested and the number of requests for each page

+------------------+--------------------+-------------+
|              host|                page|hosts_requests|
+------------------+--------------------+-------------+
| edams.ksc.nasa.gov|           /ksc.html|         1020|
| edams.ksc.nasa.gov|/images/WORLD-log...|          870|
| edams.ksc.nasa.gov|/images/NASA-logo...|          869|
| edams.ksc.nasa.gov|/images/MOSAIC-lo...|          867|
| edams.ksc.nasa.gov|/images/USA-logos...|          867|
| edams.ksc.nasa.gov|/images/ksclogo-m...|          866|
|      inet2.tek.com|/shuttle/countdow...|          719|
|zooropa.res.cmu.edu|                    |          624|
|       163.206.89.4|/images/NASA-logo...|          568|
|     beta.xerox.com|/images/NASA-logo...|          564|
+------------------+--------------------+-------------+

+------------------+--------------------+------------+
|              host|                page|page_requests|
+------------------+--------------------+------------+
| edams.ksc.nasa.gov|           /ksc.html|        1020|
| edams.ksc.nasa.gov|/images/WORLD-log...|         870|
| edams.ksc.nasa.gov|/images/NASA-logo...|         869|
| edams.ksc.nasa.gov|/images/MOSAIC-lo...|         867|
| edams.ksc.nasa.gov|/images/USA-logos...|         867|
| edams.ksc.nasa.gov|/images/ksclogo-m...|         866|
|      inet2.tek.com|/shuttle/countdow...|         719|
|zooropa.res.cmu.edu|                    |         624|
|       163.206.89.4|/images/NASA-logo...|         568|
|     beta.xerox.com|/images/NASA-logo...|         564|
+------------------+--------------------+------------+
```

7. FOR EACH MALFORMED LINE, DISPLAY AN ERROR MESSAGE AND THE LINE NUMBER.

As we have seen above during the data entry, we have many null results in the content_size column because many entries have "-" and not "0". According to answer this question we have already put an index column. Also, as we can see below, we registering a new table including the malformed entries and we built a query that can appear as an error message the line number of each error entry and the necessary fields.

```
# counting malformed entries
bad_rows_df.count()
```

```
14178
```

```
# show all the malformed entries
bad_rows_df.show(10)
```

```
+--------------------+--------------------+------+--------------------+--------+------+------------+-----+
|                host|           timestamp|method|            endpoint|protocol|status|content_size|index|
+--------------------+--------------------+------+--------------------+--------+------+------------+-----+
|          gw1.att.com|01/Aug/1995:00:03...|   GET|/shuttle/missions...|HTTP/1.0|   302|        null|  158|
|js002.cc.utsunomi...|01/Aug/1995:00:07...|   GET|/shuttle/resource...|HTTP/1.0|   404|        null|  321|
|     tia1.eskimo.com|01/Aug/1995:00:28...|   GET|/pub/winvn/releas...|HTTP/1.0|   404|        null|  779|
|itws.info.eng.nii...|01/Aug/1995:00:38...|   GET|/ksc.html/facts/a...|HTTP/1.0|   403|        null| 1066|
|grimnet23.idirect...|01/Aug/1995:00:50...|   GET|/www/software/win...|HTTP/1.0|   404|        null| 1426|
|miriworld.its.uni...|01/Aug/1995:01:04...|   GET|/history/history.htm|HTTP/1.0|   404|        null| 1730|
|        ras38.srv.net|01/Aug/1995:01:05...|   GET|/elv/DELTA/uncons...|HTTP/1.0|   404|        null| 1735|
|   cs1-06.leh.ptd.net|01/Aug/1995:01:17...|   GET|      /sts-71/launch/|        |   404|        null| 2021|
|www-b2.proxy.aol.com|01/Aug/1995:01:22...|   GET|   /shuttle/countdown|HTTP/1.0|   302|        null| 2072|
|       maui56.maui.net|01/Aug/1995:01:31...|   GET|            /shuttle|HTTP/1.0|   302|        null| 2303|
+--------------------+--------------------+------+--------------------+--------+------+------------+-----+
only showing top 10 rows
```

```
# Registering a table with malformed entries
bad_rows_df.registerTempTable("bad_rows_data_table")
```

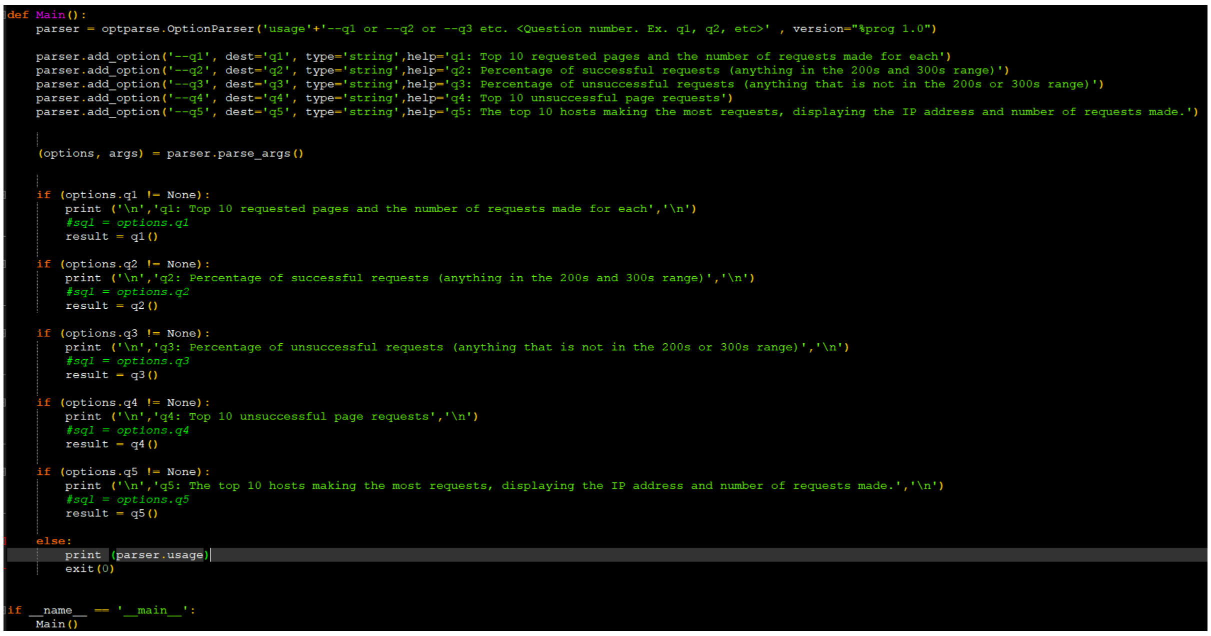
```
print("q8: The log file contains malformed entries; for each malformed line, display an error message and the line numbe
r.,'\n'")

sql = """
SELECT  case
when content_size is null
then 'field size is null'
when host is null
then 'field host is null'
when host is null
then 'field host is null'
when timestamp is null
then 'field timestamp is null'
when endpoint is null
then 'field endpoint is null'
when protocol = ' '
then 'field protocol is null'
when method is null
then 'field method is null'
else 'No errors'
end  as error_message , index, host, timestamp, endpoint, protocol, content_size as size
FROM bad_rows_data_table
"""

spark.sql(sql).show(10)
```

```
q8: The log file contains malformed entries; for each malformed line, display an error message and the line number.,'
'
+------------------+-----+--------------------+--------------------+--------------------+--------+----+
|     error_message|index|                host|           timestamp|            endpoint|protocol|size|
+------------------+-----+--------------------+--------------------+--------------------+--------+----+
|field size is null|  158|         gw1.att.com|01/Aug/1995:00:03...|/shuttle/missions...|HTTP/1.0|null|
|field size is null|  321|js002.cc.utsunomi...|01/Aug/1995:00:07...|/shuttle/resource...|HTTP/1.0|null|
|field size is null|  779|     tia1.eskimo.com|01/Aug/1995:00:28...|/pub/winvn/releas...|HTTP/1.0|null|
|field size is null| 1066|itws.info.eng.nii...|01/Aug/1995:00:38...|/ksc.html/facts/a...|HTTP/1.0|null|
|field size is null| 1426|grimnet23.idirect...|01/Aug/1995:00:50...|/www/software/win...|HTTP/1.0|null|
|field size is null| 1730|miriworld.its.uni...|01/Aug/1995:01:04...|/history/history.htm|HTTP/1.0|null|
|field size is null| 1735|       ras38.srv.net|01/Aug/1995:01:05...|/elv/DELTA/uncons...|HTTP/1.0|null|
|field size is null| 2021|  cs1-06.leh.ptd.net|01/Aug/1995:01:17...|     /sts-71/launch/|        |null|
|field size is null| 2072|www-b2.proxy.aol.com|01/Aug/1995:01:22...|  /shuttle/countdown|HTTP/1.0|null|
|field size is null| 2303|     maui56.maui.net|01/Aug/1995:01:31...|            /shuttle|HTTP/1.0|null|
+------------------+-----+--------------------+--------------------+--------------------+--------+----+
only showing top 10 rows
```

8. Option parsing to produce only the report for one of the previous points (e.g. only the top 10 URLs, only the percentage of successful requests, and so on)

In this question, we develop an optional parser in python for producing only the report for one of the previous points (e.g., only the top 10 URLs, only the percentage of successful requests, and so on)

So, the user can provide the question number and receives the results. Also, the parser gives the option to the user to get help with the description of the question. The code we use is below. More detailed usage is described in the next chapter.



```
def Main():
    parser = optparse.OptionParser('usage'+'--q1 or --q2 or --q3 etc. <Question number. Ex. q1, q2, etc>' , version="%prog 1.0")

    parser.add_option('--q1', dest='q1', type='string',help='q1: Top 10 requested pages and the number of requests made for each')
    parser.add_option('--q2', dest='q2', type='string',help='q2: Percentage of successful requests (anything in the 200s and 300s range)')
    parser.add_option('--q3', dest='q3', type='string',help='q3: Percentage of unsuccessful requests (anything that is not in the 200s or 300s range)')
    parser.add_option('--q4', dest='q4', type='string',help='q4: Top 10 unsuccessful page requests')
    parser.add_option('--q5', dest='q5', type='string',help='q5: The top 10 hosts making the most requests, displaying the IP address and number of requests made.')


    (options, args) = parser.parse_args()


    if (options.q1 != None):
        print ('\n','q1: Top 10 requested pages and the number of requests made for each','\n')
        #sql = options.q1
        result = q1()

    if (options.q2 != None):
        print ('\n','q2: Percentage of successful requests (anything in the 200s and 300s range)','\n')
        #sql = options.q2
        result = q2()

    if (options.q3 != None):
        print ('\n','q3: Percentage of unsuccessful requests (anything that is not in the 200s or 300s range)','\n')
        #sql = options.q3
        result = q3()

    if (options.q4 != None):
        print ('\n','q4: Top 10 unsuccessful page requests','\n')
        #sql = options.q4
        result = q4()

    if (options.q5 != None):
        print ('\n','q5: The top 10 hosts making the most requests, displaying the IP address and number of requests made.','\n')
        #sql = options.q5
        result = q5()

    else:
        print (parser.usage)
        exit(0)


if __name__ == '__main__':
    Main()
```

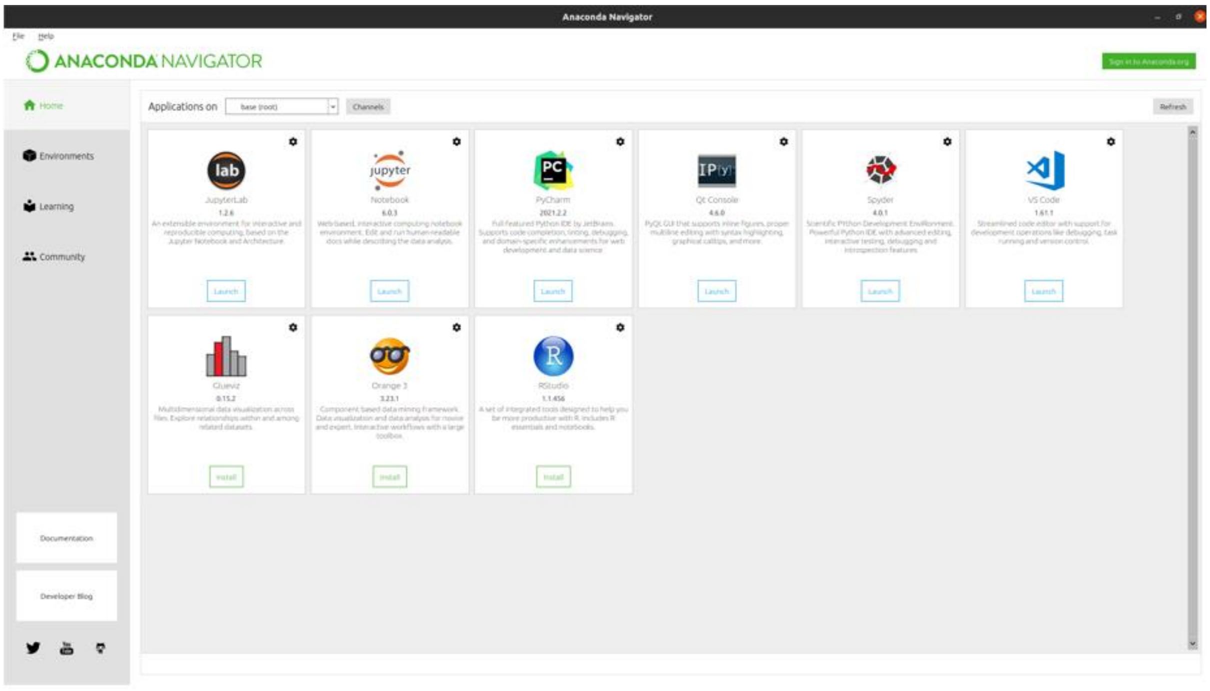
7. How to run and use the code

In this chapter, we see how to run and use the code we build.

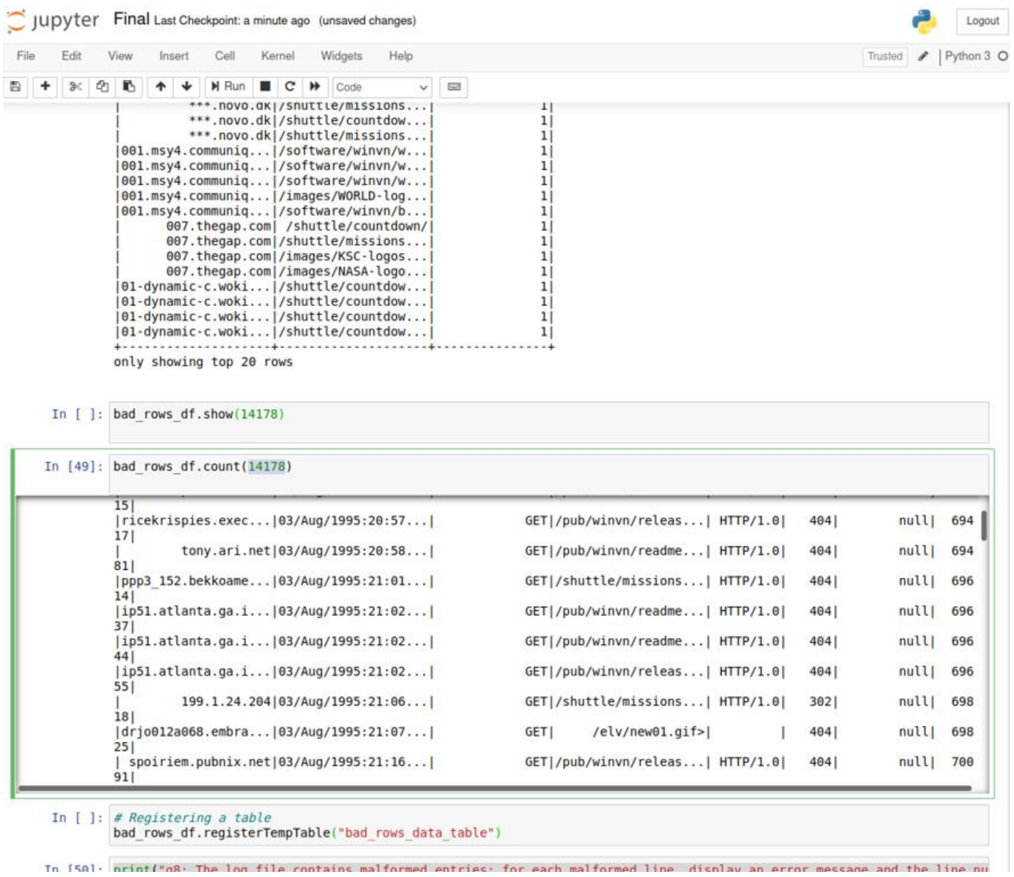
As we see in previous chapters first, we begin the anaconda with the below command in the Linux console:

anaconda-navigator

after that, we can see the below menu.



In the navigator, we choose the Jupyter and we choose the fie of the code challenge and we press the run according to start running the code block via block.



```
|         ***.novo.dk|/shuttle/missions...|             1|
|         ***.novo.dk|/shuttle/countdow...|             1|
|         ***.novo.dk|/shuttle/missions...|             1|
|001.msy4.communiq...|/software/winvn/w...|             1|
|001.msy4.communiq...|/software/winvn/w...|             1|
|001.msy4.communiq...|/software/winvn/w...|             1|
|001.msy4.communiq...|/images/WORLD-log...|             1|
|001.msy4.communiq...|/software/winvn/b...|             1|
|      007.thegap.com| /shuttle/countdown/|             1|
|      007.thegap.com|/shuttle/missions...|             1|
|      007.thegap.com|/images/KSC-logos...|             1|
|      007.thegap.com|/images/NASA-logo...|             1|
|01-dynamic-c.woki...|/shuttle/countdow...|             1|
|01-dynamic-c.woki...|/shuttle/countdow...|             1|
|01-dynamic-c.woki...|/shuttle/countdow...|             1|
|01-dynamic-c.woki...|/shuttle/countdow...|             1|
+--------------------+--------------------+--------------+
only showing top 20 rows
```

```
In [ ]: bad_rows_df.show(14178)
```

```
In [49]: bad_rows_df.count(14178)
```

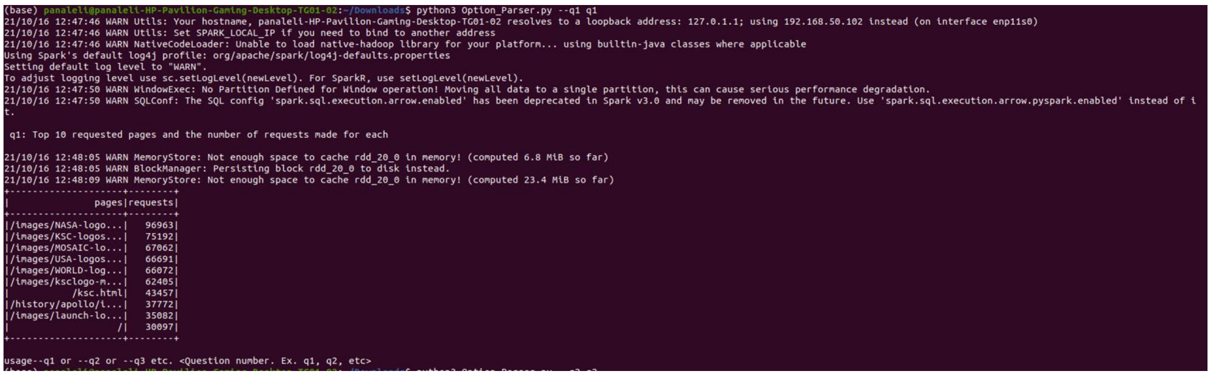
```
15|
|ricekrispies.exec...|03/Aug/1995:20:57...|         GET|/pub/winvn/releas...| HTTP/1.0|  404|      null|  694
17|
|        tony.ari.net|03/Aug/1995:20:58...|         GET|/pub/winvn/readme...| HTTP/1.0|  404|      null|  694
81|
|ppp3_152.bekkoame...|03/Aug/1995:21:01...|         GET|/shuttle/missions...| HTTP/1.0|  404|      null|  696
14|
|ip51.atlanta.ga.i...|03/Aug/1995:21:02...|         GET|/pub/winvn/readme...| HTTP/1.0|  404|      null|  696
37|
|ip51.atlanta.ga.i...|03/Aug/1995:21:02...|         GET|/pub/winvn/readme...| HTTP/1.0|  404|      null|  696
44|
|ip51.atlanta.ga.i...|03/Aug/1995:21:02...|         GET|/pub/winvn/releas...| HTTP/1.0|  404|      null|  696
55|
|        199.1.24.204|03/Aug/1995:21:06...|         GET|/shuttle/missions...| HTTP/1.0|  302|      null|  698
18|
|drjo012a068.embra...|03/Aug/1995:21:07...|         GET|     /elv/new01.gif>|         |  404|      null|  698
25|
| spoiriem.pubnix.net|03/Aug/1995:21:16...|         GET|/pub/winvn/releas...| HTTP/1.0|  404|      null|  700
91|
```

```
In [ ]: # Registering a table
        bad_rows_df.registerTempTable("bad_rows_data_table")
```

The above process can run all the queries of the code challenge. But to run question 6 which is the option parsing, we need to follow the below process.

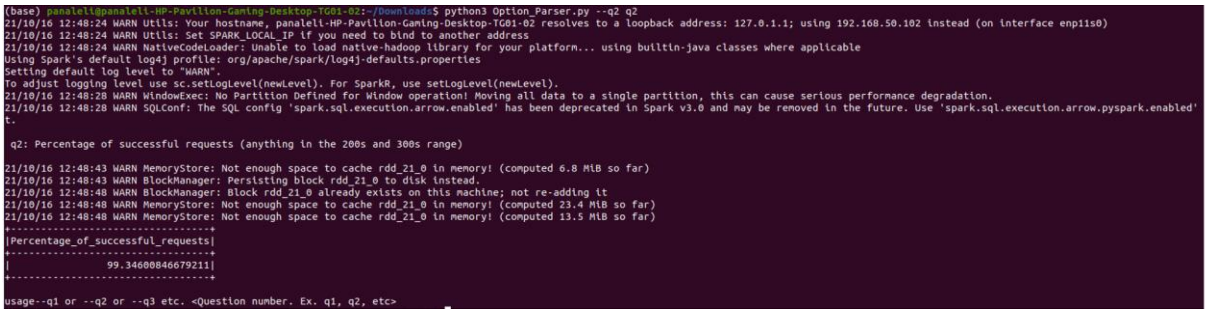
In the Linux console, we can enter the below commands. Also, in the below figures we see the results of each option.

- python3 Option_Parser.py --q1 q1: Parsing question 1



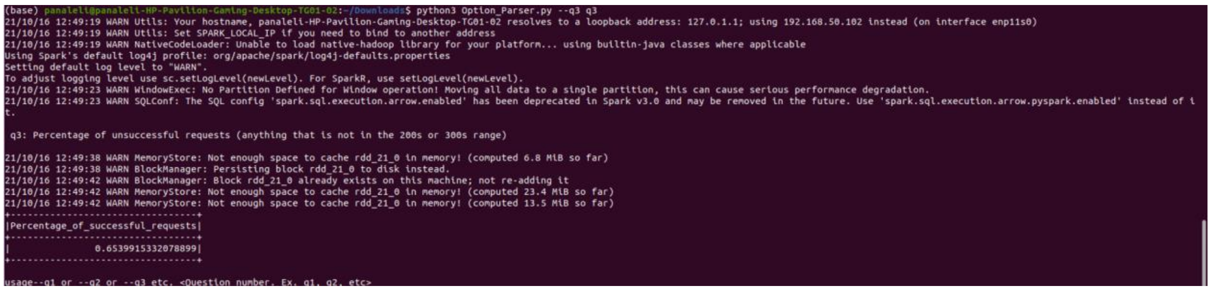
```
(base) panaleli@panaleli-HP-Pavilion-Gaming-Desktop-TG01-02:~/Downloads$ python3 Option_Parser.py --q1 q1
21/10/16 12:47:46 WARN Utils: Your hostname, panaleli-HP-Pavilion-Gaming-Desktop-TG01-02 resolves to a loopback address: 127.0.1.1; using 192.168.50.102 instead (on interface enp11s0)
21/10/16 12:47:46 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/10/16 12:47:46 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/10/16 12:47:50 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
21/10/16 12:47:50 WARN SQLConf: The SQL config 'spark.sql.execution.arrow.enabled' has been deprecated in Spark v3.0 and may be removed in the future. Use 'spark.sql.execution.arrow.pyspark.enabled' instead of i
t.

 q1: Top 10 requested pages and the number of requests made for each

21/10/16 12:48:05 WARN MemoryStore: Not enough space to cache rdd_20_0 in memory! (computed 6.8 MiB so far)
21/10/16 12:48:05 WARN BlockManager: Persisting block rdd_20_0 to disk instead.
21/10/16 12:48:09 WARN MemoryStore: Not enough space to cache rdd_20_0 in memory! (computed 23.4 MiB so far)
+--------------------+--------+
|               pages|requests|
+--------------------+--------+
|/images/NASA-logo...|   96963|
|/images/KSC-logos...|   75192|
|/images/MOSAIC-lo...|   67062|
|/images/USA-logos...|   66691|
|/images/WORLD-log...|   66072|
|/images/ksclogo-m...|   62405|
|           /ksc.html|   43457|
|/history/apollo/i...|   37772|
|/images/launch-lo...|   35082|
|                   /|   30097|
+--------------------+--------+

usage--q1 or --q2 or --q3 etc. <Question number. Ex. q1, q2, etc>
```

- python3 Option_Parser.py --q2 q2: Parsing question 2



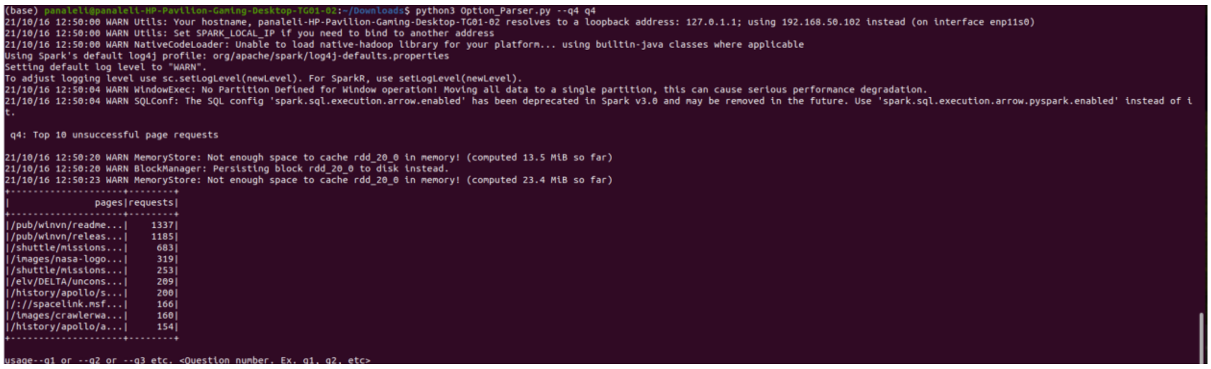
```
(base) panaleli@panaleli-HP-Pavilion-Gaming-Desktop-TG01-02:~/Downloads$ python3 Option_Parser.py --q2 q2
21/10/16 12:48:24 WARN Utils: Your hostname, panaleli-HP-Pavilion-Gaming-Desktop-TG01-02 resolves to a loopback address: 127.0.1.1; using 192.168.50.102 instead (on interface enp11s0)
21/10/16 12:48:24 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/10/16 12:48:24 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/10/16 12:48:28 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
21/10/16 12:48:28 WARN SQLConf: The SQL config 'spark.sql.execution.arrow.enabled' has been deprecated in Spark v3.0 and may be removed in the future. Use 'spark.sql.execution.arrow.pyspark.enabled'
t.

 q2: Percentage of successful requests (anything in the 200s and 300s range)

21/10/16 12:48:43 WARN MemoryStore: Not enough space to cache rdd_21_0 in memory! (computed 6.8 MiB so far)
21/10/16 12:48:43 WARN BlockManager: Persisting block rdd_21_0 to disk instead.
21/10/16 12:48:48 WARN BlockManager: Block rdd_21_0 already exists on this machine; not re-adding it
21/10/16 12:48:48 WARN MemoryStore: Not enough space to cache rdd_21_0 in memory! (computed 23.4 MiB so far)
21/10/16 12:48:48 WARN MemoryStore: Not enough space to cache rdd_21_0 in memory! (computed 13.5 MiB so far)
+--------------------------------+
|Percentage_of_successful_requests|
+--------------------------------+
|               99.34600846679211|
+--------------------------------+

usage--q1 or --q2 or --q3 etc. <Question number. Ex. q1, q2, etc>
```

- python3 Option_Parser.py --q3 q3: Parsing question 3



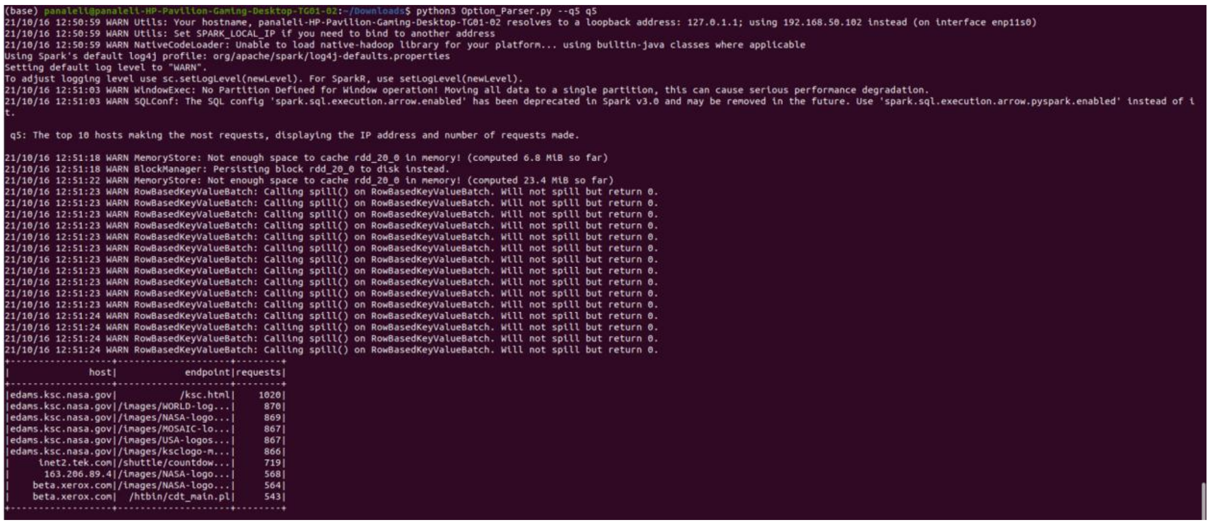
```
(base) panaleli@panaleli-HP-Pavilion-Gaming-Desktop-TG01-02:~/Downloads$ python3 Option_Parser.py --q3 q3
21/10/16 12:49:19 WARN Utils: Your hostname, panaleli-HP-Pavilion-Gaming-Desktop-TG01-02 resolves to a loopback address: 127.0.1.1; using 192.168.50.102 instead (on interface enp11s0)
21/10/16 12:49:19 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/10/16 12:49:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/10/16 12:49:23 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
21/10/16 12:49:23 WARN SQLConf: The SQL config 'spark.sql.execution.arrow.enabled' has been deprecated in Spark v3.0 and may be removed in the future. Use 'spark.sql.execution.arrow.pyspark.enabled' instead of i
t.

 q3: Percentage of unsuccessful requests (anything that is not in the 200s or 300s range)

21/10/16 12:49:38 WARN MemoryStore: Not enough space to cache rdd_21_0 in memory! (computed 6.8 MiB so far)
21/10/16 12:49:38 WARN BlockManager: Persisting block rdd_21_0 to disk instead.
21/10/16 12:49:42 WARN BlockManager: Block rdd_21_0 already exists on this machine; not re-adding it
21/10/16 12:49:42 WARN MemoryStore: Not enough space to cache rdd_21_0 in memory! (computed 23.4 MiB so far)
21/10/16 12:49:42 WARN MemoryStore: Not enough space to cache rdd_21_0 in memory! (computed 13.5 MiB so far)
+--------------------------------+
|Percentage_of_successful_requests|
+--------------------------------+
|              0.6539915332078899|
+--------------------------------+

usage--q1 or --q2 or --q3 etc. <Question number. Ex. q1, q2, etc>
```

- python3 Option_Parser.py --q4 q4: Parsing question 4



```
(base) panaleli@panaleli-HP-Pavilion-Gaming-Desktop-TG01-02:~/Downloads$ python3 Option_Parser.py --q4 q4
21/10/16 12:50:00 WARN Utils: Your hostname, panaleli-HP-Pavilion-Gaming-Desktop-TG01-02 resolves to a loopback address: 127.0.1.1; using 192.168.50.102 instead (on interface enp11s0)
21/10/16 12:50:00 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/10/16 12:50:00 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/10/16 12:50:04 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
21/10/16 12:50:04 WARN SQLConf: The SQL config 'spark.sql.execution.arrow.enabled' has been deprecated in Spark v3.0 and may be removed in the future. Use 'spark.sql.execution.arrow.pyspark.enabled' instead of i
t.

 q4: Top 10 unsuccessful page requests

21/10/16 12:50:20 WARN MemoryStore: Not enough space to cache rdd_20_0 in memory! (computed 13.5 MiB so far)
21/10/16 12:50:20 WARN BlockManager: Persisting block rdd_20_0 to disk instead.
21/10/16 12:50:23 WARN MemoryStore: Not enough space to cache rdd_20_0 in memory! (computed 23.4 MiB so far)
+--------------------+--------+
|               pages|requests|
+--------------------+--------+
|/pub/winvn/readme...|    1337|
|/pub/winvn/releas...|    1185|
|/shuttle/missions...|     683|
|/images/nasa-logo...|     319|
|/shuttle/missions...|     253|
|/elv/DELTA/uncons...|     209|
|/history/apollo/s...|     200|
|/://spacelink.msf...|     166|
|/images/crawlerwa...|     160|
|/history/apollo/a...|     154|
+--------------------+--------+

usage--q1 or --q2 or --q3 etc. <Question number. Ex. q1, q2, etc>
```

- python3 Option_Parser.py --q5 q5: Parsing question 5



```
(base) panaleli@panaleli-HP-Pavilion-Gaming-Desktop-TG01-02:~/Downloads$ python3 Option_Parser.py --q5 q5
21/10/16 12:50:59 WARN Utils: Your hostname, panaleli-HP-Pavilion-Gaming-Desktop-TG01-02 resolves to a loopback address: 127.0.1.1; using 192.168.50.102 instead (on interface enp11s0)
21/10/16 12:50:59 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/10/16 12:50:59 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/10/16 12:51:03 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
21/10/16 12:51:03 WARN SQLConf: The SQL config 'spark.sql.execution.arrow.enabled' has been deprecated in Spark v3.0 and may be removed in the future. Use 'spark.sql.execution.arrow.pyspark.enabled' instead of i
t.

 q5: The top 10 hosts making the most requests, displaying the IP address and number of requests made.

21/10/16 12:51:18 WARN MemoryStore: Not enough space to cache rdd_20_0 in memory! (computed 6.8 MiB so far)
21/10/16 12:51:18 WARN BlockManager: Persisting block rdd_20_0 to disk instead.
21/10/16 12:51:22 WARN MemoryStore: Not enough space to cache rdd_20_0 in memory! (computed 23.4 MiB so far)
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:23 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:24 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:24 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:24 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
21/10/16 12:51:24 WARN RowBasedKeyValueBatch: Calling spill() on RowBasedKeyValueBatch. Will not spill but return 0.
+-----------------+--------------------+--------+
|             host|            endpoint|requests|
+-----------------+--------------------+--------+
|edams.ksc.nasa.gov|           /ksc.html|    1020|
|edams.ksc.nasa.gov|/images/WORLD-log...|     870|
|edams.ksc.nasa.gov|/images/NASA-logo...|     869|
|edams.ksc.nasa.gov|/images/MOSAIC-lo...|     867|
|edams.ksc.nasa.gov|/images/USA-logos...|     867|
|edams.ksc.nasa.gov|/images/ksclogo-m...|     866|
|      inet2.tek.com|/shuttle/countdow...|     719|
|       163.206.89.4|/images/NASA-logo...|     568|
|     beta.xerox.com|/images/NASA-logo...|     564|
|     beta.xerox.com|  /htbin/cdt_main.pl|     543|
+-----------------+--------------------+--------+
```

For help, we can enter the below commands.

- python3 Option_Parser.py -help q2: Help for question 1
- python3 Option_Parser.py -help q2: Help for question 2
- python3 Option_Parser.py -help q2: Help for question 3
- python3 Option_Parser.py -help q2: Help for question 4
- python3 Option_Parser.py -help q2: Help for question 5

8. Assumptions and Conclusions

As we see in previous chapters, we face a code challenge where we must answer some questions. Due to that, we download the file NASA_access_log_Aug95.gz and we provide the solution analyzing the log data with Python and Apache Spark. As we see Spark is a very reliable and useful framework for analyzing a big amount of log files.

First, we make all the appropriate installations and configurations and we load the necessary libraries. After that, we provide our queries, in the Spark SQL, and we use the data frames method for our dataset. We continue building the queries in SQL and provide our answers.

The conclusion we can draw is that this is a very good and interesting experience where it helped us set up and analyze a large volume of data and provide reliable solutions for data analysis. We can use this knowledge in real flow data and be able to solve complex data analysis issues even in almost real-time for the industry and to provide to our customers the best results.

References

1. 4. Spark SQL and DataFrames: Introduction to Built-in Data Sources - Learning Spark, 2nd Edition [Book]
   Type Web Page
   URL https://www.oreilly.com/library/view/learning-spark-2nd/9781492050032/ch04.html
   Attachments

2. DataFrames Vs RDDs in Spark – Part 1 | DataScience+
   Type Blog Post
   URL https://datascienceplus.com/dataframes-vs-rdds-in-spark-part-1/

3. How to wrangle log data with Python and Apache Spark
   Type Web Page
   Author 14 May 2019 DipanjanSarkarFeed 212up
   Abstract Case study with NASA logs to show how Spark can be leveraged for analyzing data at scale.

4. Installing PySpark with JAVA 8 on ubuntu 18.04 | by Parijat Bhatt | Towards Data Science
   PySpark and SparkSQL Basics
   Type Web Page
   Author Pınar Ersoy
   Abstract How to implement Spark with Python Programming
   Date 2021-06-11T22:56:03.783Z
   URL https://towardsdatascience.com/pyspark-and-sparksql-basics-6cb4bf967e53

5. Scalable Log Analytics with Apache Spark — A Comprehensive Case-Study | by Dipanjan (DJ) Sarkar | Towards Data Science
   Type Web Page
   URL https://towardsdatascience.com/scalable-log-analytics-with-apache-spark-a-comprehensive-case-study-2be3eb3be977

6. Spark RDDs Vs DataFrames vs SparkSQL – Part 3 : Web Server Log Analysis | DataScience+
   Type Blog Post
   Short Title Spark RDDs Vs DataFrames vs SparkSQL – Part 3
   URL https://datascienceplus.com/spark-rdds-vs-dataframes-vs-sparksql-part-3-web-server-log-analysis/