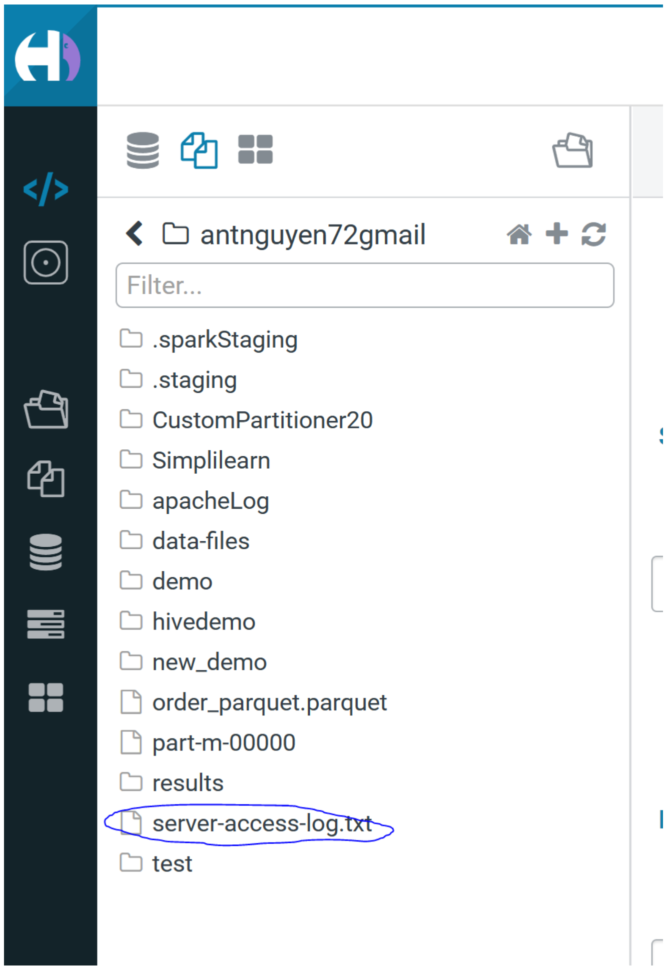
Objective:

Perform server log analysis to assist businesses in identifying and analyzing critical business errors, as well as potential customers and their domains

Steps to be performed:

Step 1: Upload the "**server-access-log**" file to the HDFS



antnguyen72gmail

Filter...

- .sparkStaging
- .staging
- CustomPartitioner20
- Simplilearn
- apacheLog
- data-files
- demo
- hivedemo
- new_demo
- order_parquet.parquet
- part-m-00000
- results
- server-access-log.txt
- test

Step 2: Execute the PySpark Commands following the below steps:

- Login to the web console
- To enter the PySpark console run the following command:

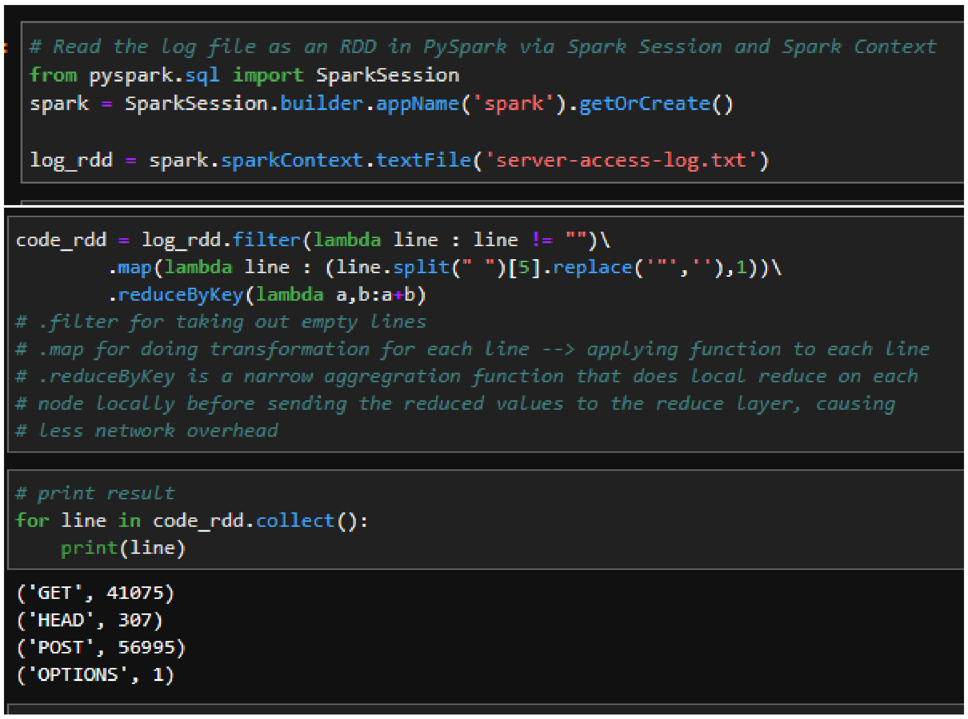
Command: pyspark3

I am using Jupyter Notebook

Step 3: Perform the below tasks on the uploaded dataset:

- Status code analysis:
  - Read the log file as an RDD in PySpark
  - Consider the sixth element as it is "**request type**" and replace the "single quote" with blank
  - Convert each word into a tuple of (word,1)
  - Apply "reduceByKey" transformation to count the values

- Display the data



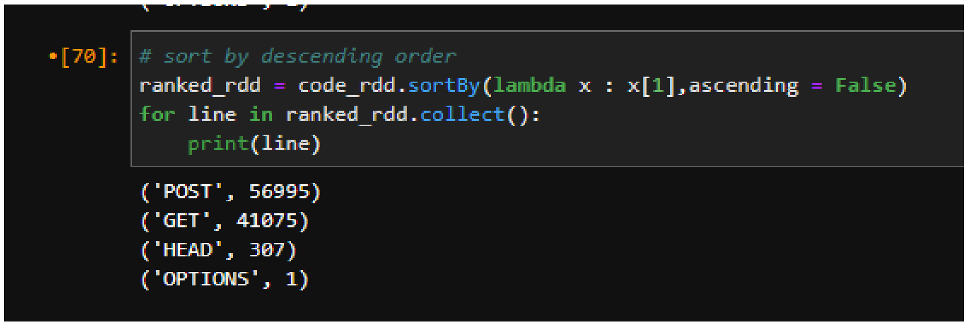
```python
# Read the log file as an RDD in PySpark via Spark Session and Spark Context
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('spark').getOrCreate()

log_rdd = spark.sparkContext.textFile('server-access-log.txt')
```

```python
code_rdd = log_rdd.filter(lambda line : line != "")\
        .map(lambda line : (line.split(" ")[5].replace('"',''),1))\
        .reduceByKey(lambda a,b:a+b)
# .filter for taking out empty lines
# .map for doing transformation for each line --> applying function to each line
# .reduceByKey is a narrow aggregration function that does local reduce on each
# node locally before sending the reduced values to the reduce layer, causing
# less network overhead
```

```python
# print result
for line in code_rdd.collect():
    print(line)
```

```
('GET', 41075)
('HEAD', 307)
('POST', 56995)
('OPTIONS', 1)
```

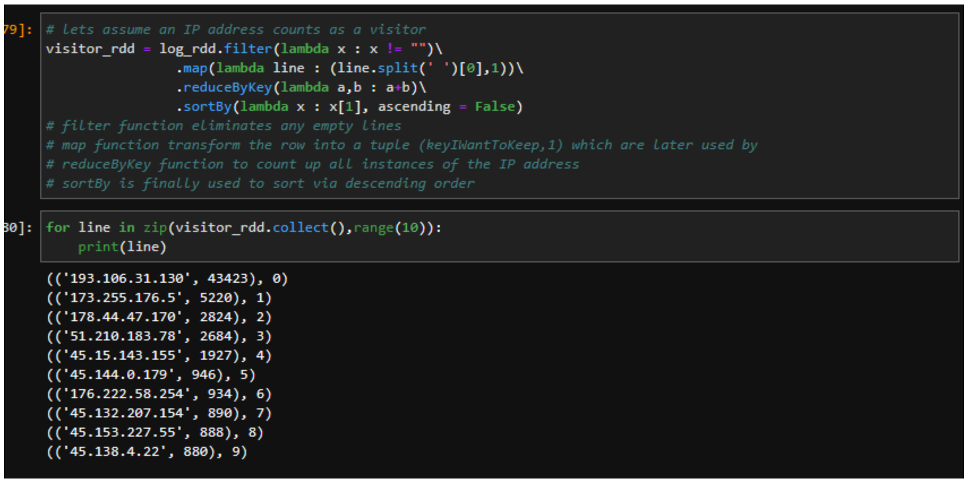
- Arrange the result in descending order and display the result



```python
[70]: # sort by descending order
ranked_rdd = code_rdd.sortBy(lambda x : x[1],ascending = False)
for line in ranked_rdd.collect():
    print(line)
```

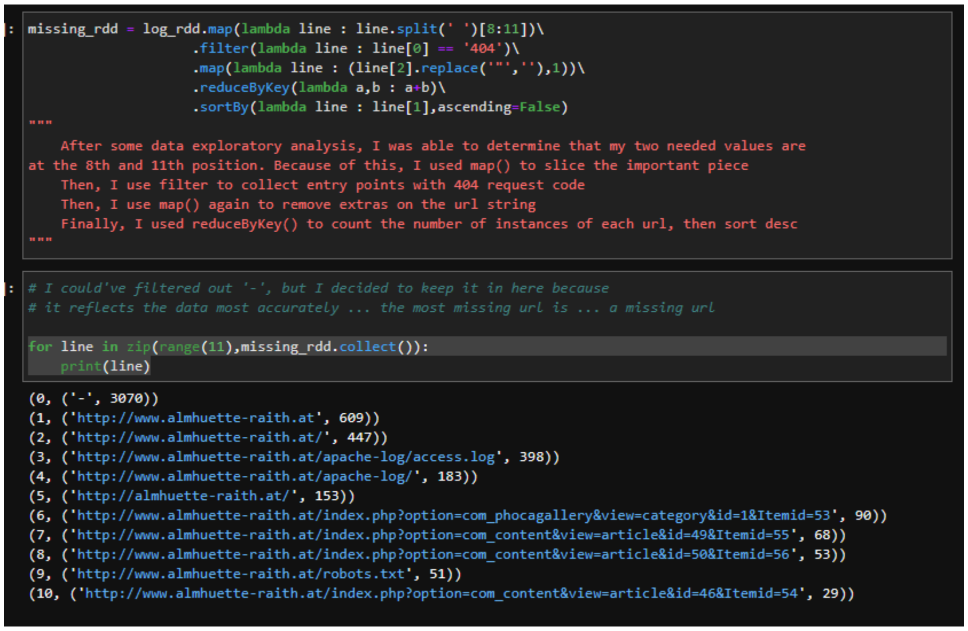
```
('POST', 56995)
('GET', 41075)
('HEAD', 307)
('OPTIONS', 1)
```

- Identify the top 10 frequent visitors of the website and show the result in the RDD



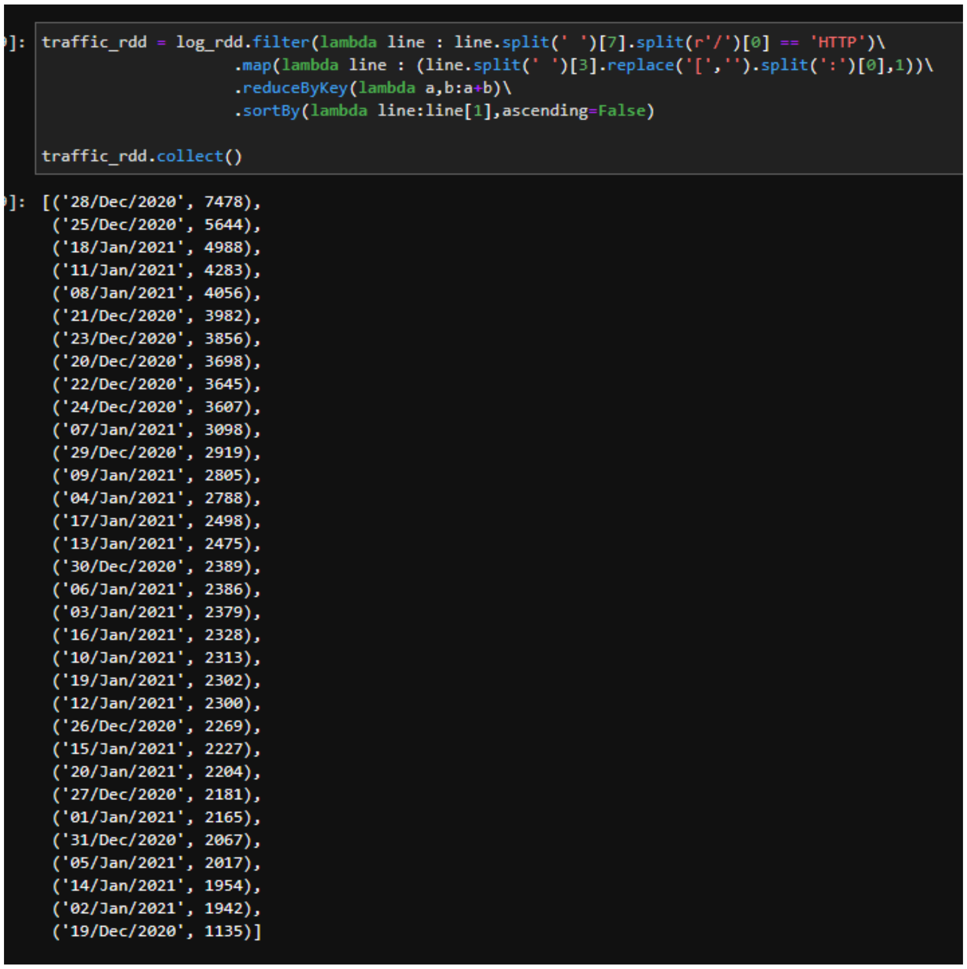
```
79]: # lets assume an IP address counts as a visitor
     visitor_rdd = log_rdd.filter(lambda x : x != "")\
                     .map(lambda line : (line.split(' ')[0],1))\
                     .reduceByKey(lambda a,b : a+b)\
                     .sortBy(lambda x : x[1], ascending = False)
     # filter function eliminates any empty lines
     # map function transform the row into a tuple (keyIWantToKeep,1) which are later used by
     # reduceByKey function to count up all instances of the IP address
     # sortBy is finally used to sort via descending order

30]: for line in zip(visitor_rdd.collect(),range(10)):
         print(line)

     (('193.106.31.130', 43423), 0)
     (('173.255.176.5', 5220), 1)
     (('178.44.47.170', 2824), 2)
     (('51.210.183.78', 2684), 3)
     (('45.15.143.155', 1927), 4)
     (('45.144.0.179', 946), 5)
     (('176.222.58.254', 934), 6)
     (('45.132.207.154', 890), 7)
     (('45.153.227.55', 888), 8)
     (('45.138.4.22', 880), 9)
```

- Identify the top 10 missing (does not exist) URLs using these steps:
  - Read the log file as an RDD in PySpark
  - Identify the URLs for which the server is returning the 404-request code and display the data



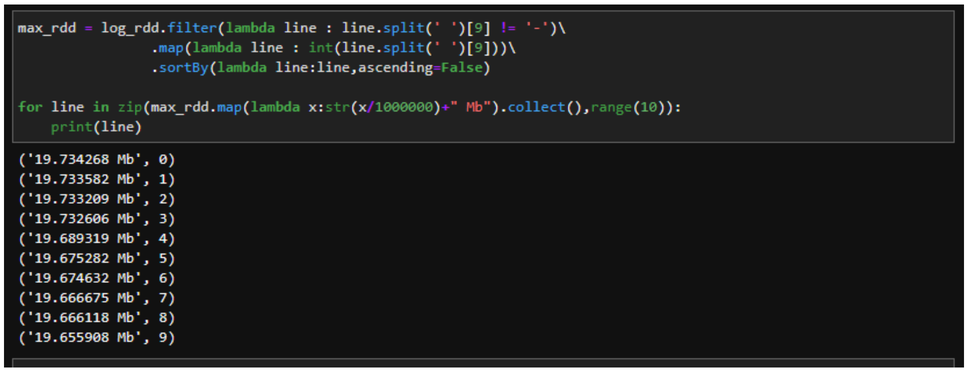
```
]: missing_rdd = log_rdd.map(lambda line : line.split(' ')[8:11])\
                       .filter(lambda line : line[0] == '404')\
                       .map(lambda line : (line[2].replace('"',''),1))\
                       .reduceByKey(lambda a,b : a+b)\
                       .sortBy(lambda line : line[1],ascending=False)
   """
       After some data exploratory analysis, I was able to determine that my two needed values are
   at the 8th and 11th position. Because of this, I used map() to slice the important piece
       Then, I use filter to collect entry points with 404 request code
       Then, I use map() again to remove extras on the url string
       Finally, I used reduceByKey() to count the number of instances of each url, then sort desc
   """

]: # I could've filtered out '-', but I decided to keep it in here because
   # it reflects the data most accurately ... the most missing url is ... a missing url

   for line in zip(range(11),missing_rdd.collect()):
       print(line)

   (0, ('-', 3070))
   (1, ('http://www.almhuette-raith.at', 609))
   (2, ('http://www.almhuette-raith.at/', 447))
   (3, ('http://www.almhuette-raith.at/apache-log/access.log', 398))
   (4, ('http://www.almhuette-raith.at/apache-log/', 183))
   (5, ('http://almhuette-raith.at/', 153))
   (6, ('http://www.almhuette-raith.at/index.php?option=com_phocagallery&view=category&id=1&Itemid=53', 90))
   (7, ('http://www.almhuette-raith.at/index.php?option=com_content&view=article&id=49&Itemid=55', 68))
   (8, ('http://www.almhuette-raith.at/index.php?option=com_content&view=article&id=50&Itemid=56', 53))
   (9, ('http://www.almhuette-raith.at/robots.txt', 51))
   (10, ('http://www.almhuette-raith.at/index.php?option=com_content&view=article&id=46&Itemid=54', 29))
```

- Identify the traffic (total number of HTTP requests received per day) using the below steps:
  - Read the log file as an RDD in PySpark
  - Fetch the DateTime string and replace "[" with blank
  - Get the date string from the DateTime
  - Identify HTTP requests using the map function
  - Display the data



```
]: traffic_rdd = log_rdd.filter(lambda line : line.split(' ')[7].split(r'/')[0] == 'HTTP')\
                        .map(lambda line : (line.split(' ')[3].replace('[','').split(':')[0],1))\
                        .reduceByKey(lambda a,b:a+b)\
                        .sortBy(lambda line:line[1],ascending=False)

   traffic_rdd.collect()

]: [('28/Dec/2020', 7478),
    ('25/Dec/2020', 5644),
    ('18/Jan/2021', 4988),
    ('11/Jan/2021', 4283),
    ('08/Jan/2021', 4056),
    ('21/Dec/2020', 3982),
    ('23/Dec/2020', 3856),
    ('20/Dec/2020', 3698),
    ('22/Dec/2020', 3645),
    ('24/Dec/2020', 3607),
    ('07/Jan/2021', 3098),
    ('29/Dec/2020', 2919),
    ('09/Jan/2021', 2805),
    ('04/Jan/2021', 2788),
    ('17/Jan/2021', 2498),
    ('13/Jan/2021', 2475),
    ('30/Dec/2020', 2389),
    ('06/Jan/2021', 2386),
    ('03/Jan/2021', 2379),
    ('16/Jan/2021', 2328),
    ('10/Jan/2021', 2313),
    ('19/Jan/2021', 2302),
    ('12/Jan/2021', 2300),
    ('26/Dec/2020', 2269),
    ('15/Jan/2021', 2227),
    ('20/Jan/2021', 2204),
    ('27/Dec/2020', 2181),
    ('01/Jan/2021', 2165),
    ('31/Dec/2020', 2067),
    ('05/Jan/2021', 2017),
    ('14/Jan/2021', 1954),
    ('02/Jan/2021', 1942),
    ('19/Dec/2020', 1135)]
```

- Identify the top 10 endpoints that transfer maximum content in megabytes and display the data



```
max_rdd = log_rdd.filter(lambda line : line.split(' ')[9] != '-')\
                .map(lambda line : int(line.split(' ')[9]))\
                .sortBy(lambda line:line,ascending=False)

for line in zip(max_rdd.map(lambda x:str(x/1000000)+" Mb").collect(),range(10)):
    print(line)

('19.734268 Mb', 0)
('19.733582 Mb', 1)
('19.733209 Mb', 2)
('19.732606 Mb', 3)
('19.689319 Mb', 4)
('19.675282 Mb', 5)
('19.674632 Mb', 6)
('19.666675 Mb', 7)
('19.666118 Mb', 8)
('19.655908 Mb', 9)
```