Assignment – 2

CWID: 50380788
Name: Mukesh Ravichandran

Source Code and Output Screenshots

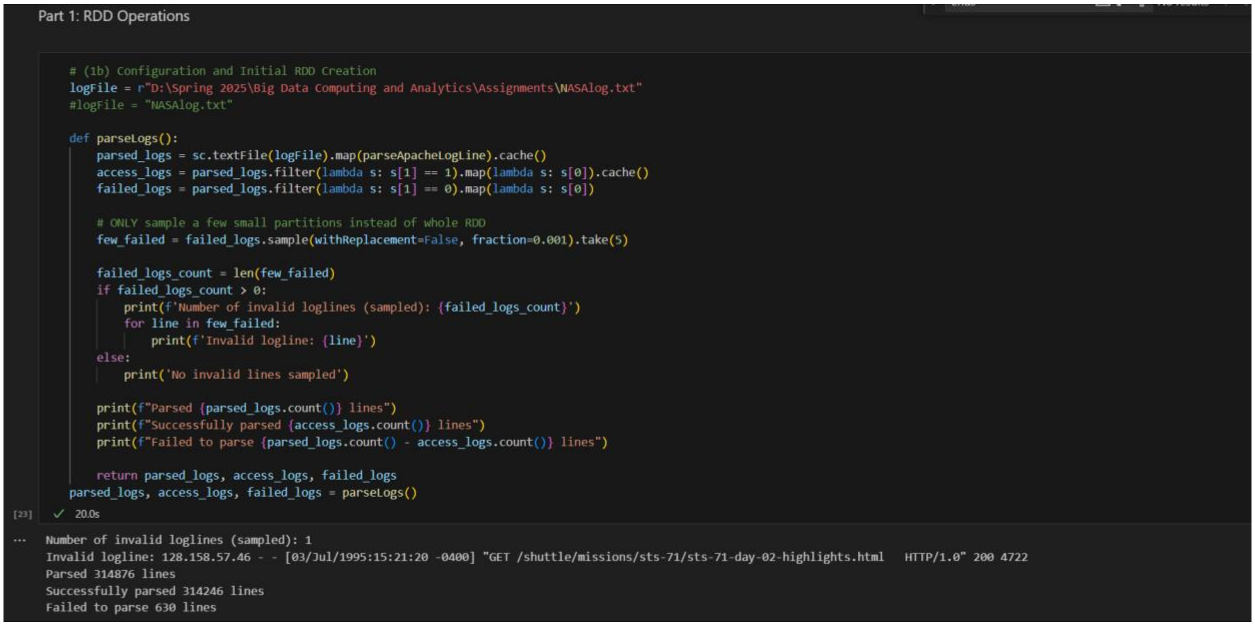
Environment Setup

```
# Set the PySpark environment variables
from pyspark import SparkContext, SparkConf
from operator import add
import re
import os
os.environ["PYSPARK_PYTHON"] = r"C:\Users\mukeshravichandran\AppData\Local\Programs\Python\Python310\python.exe"
import datetime
from pyspark.sql import Row
import matplotlib.pyplot as plt
# Initialize SparkContext
sc = SparkContext("local", "NASA RDD")
```

Log Parsing Logic

```
def parseApacheLogLine(logline):
    """ Parse a line in the Apache Common Log format
    Args:
        logline (str): a line of text in the Apache Common Log format
    Returns:
        tuple: either a dictionary containing the parts of the Apache Access Log and 1,
               or the original invalid log line and 0
    """
    match = re.search(APACHE_ACCESS_LOG_PATTERN, logline)
    if match is None:
        return (logline, 0)
    size_field = match.group(9)
    if size_field == '-':
        size = int(0)
    else:
        size = int(match.group(9))
    return (Row(
        host          = match.group(1),
        client_identd = match.group(2),
        user_id       = match.group(3),
        date_time     = parse_apache_time(match.group(4)),
        method        = match.group(5),
        endpoint      = match.group(6),
        protocol      = match.group(7),
        response_code = int(match.group(8)),
        content_size  = size
    ), 1)
 # A regular expression pattern to extract fields from the log line
APACHE_ACCESS_LOG_PATTERN = '^(\S+) (\S+) (\S+) \[([\w:/]+\s[+\-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3}) (\S+)$'
```

Configuration and RDD creation



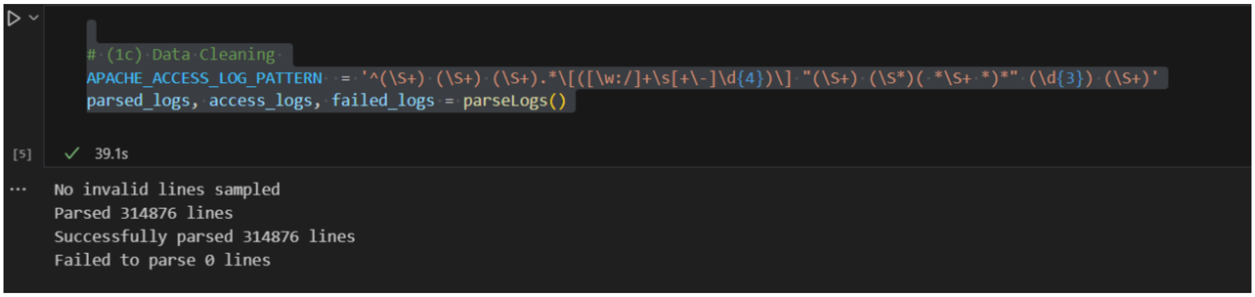
Part 1: RDD Operations

```
# (1b) Configuration and Initial RDD Creation
logFile = r"D:\Spring 2025\Big Data Computing and Analytics\Assignments\NASAlog.txt"
#logFile = "NASAlog.txt"

def parseLogs():
    parsed_logs = sc.textFile(logFile).map(parseApacheLogLine).cache()
    access_logs = parsed_logs.filter(lambda s: s[1] == 1).map(lambda s: s[0]).cache()
    failed_logs = parsed_logs.filter(lambda s: s[1] == 0).map(lambda s: s[0])

    # ONLY sample a few small partitions instead of whole RDD
    few_failed = failed_logs.sample(withReplacement=False, fraction=0.001).take(5)

    failed_logs_count = len(few_failed)
    if failed_logs_count > 0:
        print(f'Number of invalid loglines (sampled): {failed_logs_count}')
        for line in few_failed:
            print(f'Invalid logline: {line}')
    else:
        print('No invalid lines sampled')

    print(f"Parsed {parsed_logs.count()} lines")
    print(f"Successfully parsed {access_logs.count()} lines")
    print(f"Failed to parse {parsed_logs.count() - access_logs.count()} lines")

    return parsed_logs, access_logs, failed_logs
parsed_logs, access_logs, failed_logs = parseLogs()
```

```
Number of invalid loglines (sampled): 1
Invalid logline: 128.158.57.46 - - [03/Jul/1995:15:21:20 -0400] "GET /shuttle/missions/sts-71/sts-71-day-02-highlights.html   HTTP/1.0" 200 4722
Parsed 314876 lines
Successfully parsed 314246 lines
Failed to parse 630 lines
```

Initial Data Cleaning

630 records failed to parse. These were addressed by refining the regular expression to ensure consistent parsing.

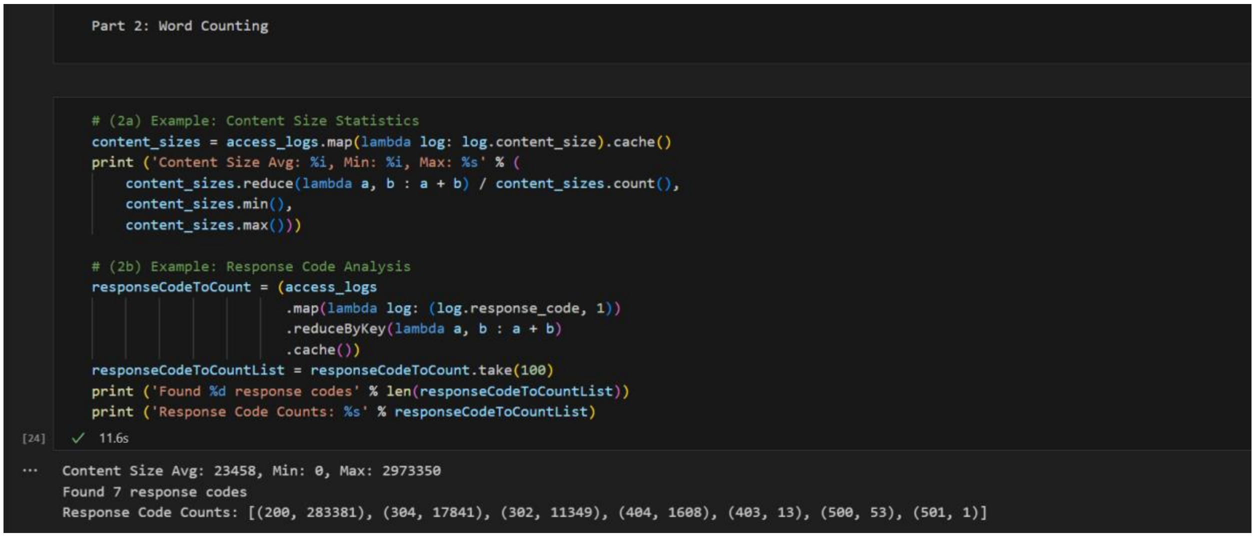


```
# (1c) Data Cleaning
APACHE_ACCESS_LOG_PATTERN  = '^(\S+) (\S+) (\S+).*\[([\w:/]+\s[+\-]\d{4})\] "(\S+) (\S*)( *\S+ *)*" (\d{3}) (\S+)'
parsed_logs, access_logs, failed_logs = parseLogs()
```

```
No invalid lines sampled
Parsed 314876 lines
Successfully parsed 314876 lines
Failed to parse 0 lines
```

Statistics and Response Code Analysis

Content size statistics and response code frequencies were computed using `reduceByKey` with lambda expressions.



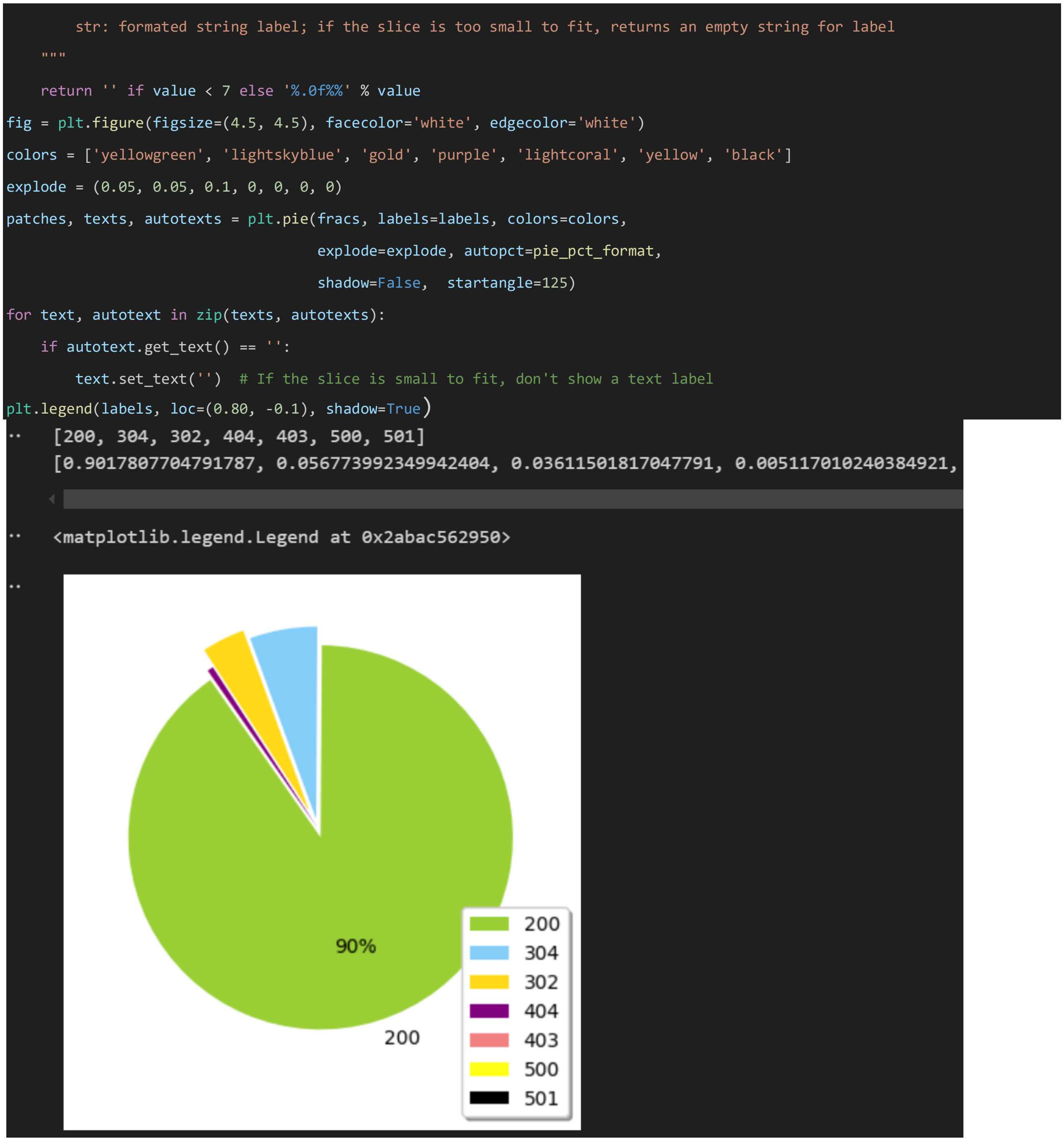
Part 2: Word Counting

```
# (2a) Example: Content Size Statistics
content_sizes = access_logs.map(lambda log: log.content_size).cache()
print ('Content Size Avg: %i, Min: %i, Max: %s' % (
    content_sizes.reduce(lambda a, b : a + b) / content_sizes.count(),
    content_sizes.min(),
    content_sizes.max()))

# (2b) Example: Response Code Analysis
responseCodeToCount = (access_logs
                       .map(lambda log: (log.response_code, 1))
                       .reduceByKey(lambda a, b : a + b)
                       .cache())
responseCodeToCountList = responseCodeToCount.take(100)
print ('Found %d response codes' % len(responseCodeToCountList))
print ('Response Code Counts: %s' % responseCodeToCountList)
```

```
Content Size Avg: 23458, Min: 0, Max: 2973350
Found 7 response codes
Response Code Counts: [(200, 283381), (304, 17841), (302, 11349), (404, 1608), (403, 13), (500, 53), (501, 1)]
```

Response Code Visualization

```
#  (2c) Example: Response Code Graphing with matplotlib
labels = responseCodeToCount.map(lambda pair: pair[0]).collect()
print(labels)
count = access_logs.count()
fracs = responseCodeToCount.map(lambda pair: float(pair[1]) / count).collect()
print(fracs)

def pie_pct_format(value):
    """ Determine the appropriate format string for the pie chart percentage label
    Args:
        value: value of the pie slice
    Returns:
```

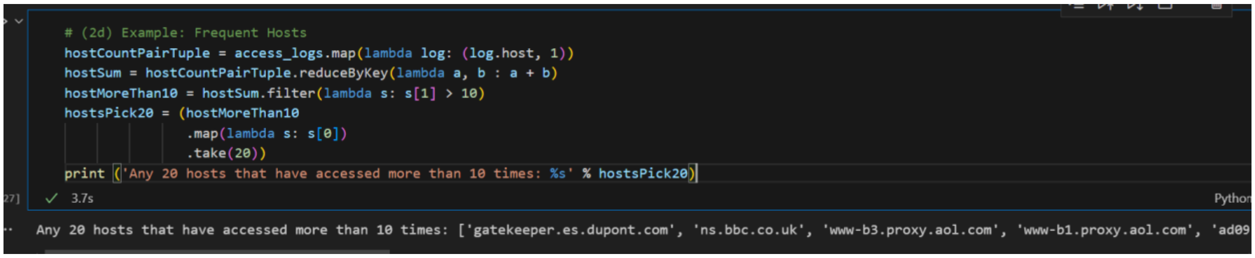


```
        str: formated string label; if the slice is too small to fit, returns an empty string for label
    """
    return '' if value < 7 else '%.0f%%' % value
fig = plt.figure(figsize=(4.5, 4.5), facecolor='white', edgecolor='white')
colors = ['yellowgreen', 'lightskyblue', 'gold', 'purple', 'lightcoral', 'yellow', 'black']
explode = (0.05, 0.05, 0.1, 0, 0, 0, 0)
patches, texts, autotexts = plt.pie(fracs, labels=labels, colors=colors,
                                    explode=explode, autopct=pie_pct_format,
                                    shadow=False,  startangle=125)
for text, autotext in zip(texts, autotexts):
    if autotext.get_text() == '':
        text.set_text('')  # If the slice is small to fit, don't show a text label
plt.legend(labels, loc=(0.80, -0.1), shadow=True)
```

```
[200, 304, 302, 404, 403, 500, 501]
[0.9017807704791787, 0.056773992349942404, 0.036115018170477791, 0.005117010240384921,
```

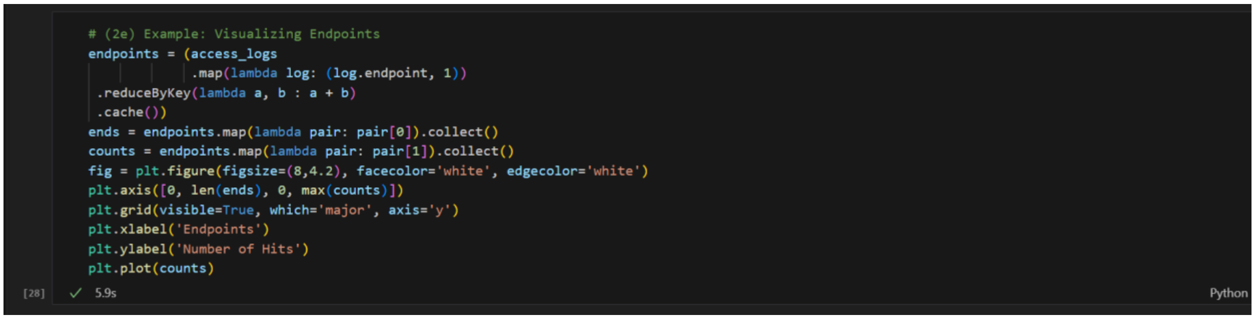
```
<matplotlib.legend.Legend at 0x2abac562950>
```

Analyzing the frequent hosts that have accessed the server

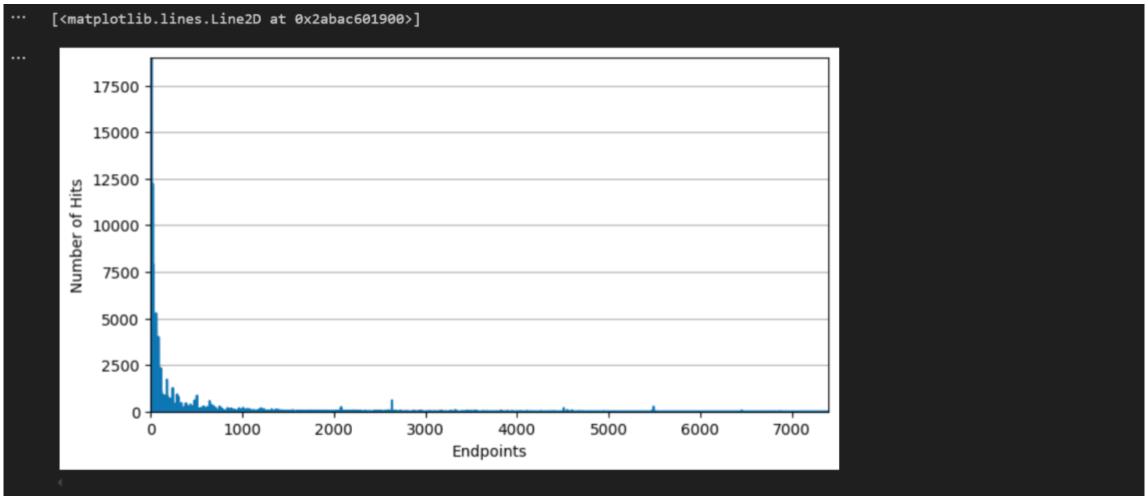


```python
# (2d) Example: Frequent Hosts
hostCountPairTuple = access_logs.map(lambda log: (log.host, 1))
hostSum = hostCountPairTuple.reduceByKey(lambda a, b : a + b)
hostMoreThan10 = hostSum.filter(lambda s: s[1] > 10)
hostsPick20 = (hostMoreThan10
               .map(lambda s: s[0])
               .take(20))
print ('Any 20 hosts that have accessed more than 10 times: %s' % hostsPick20)
```

```
Any 20 hosts that have accessed more than 10 times: ['gatekeeper.es.dupont.com', 'ns.bbc.co.uk', 'www-b3.proxy.aol.com', 'www-b1.proxy.aol.com', 'ad09
```



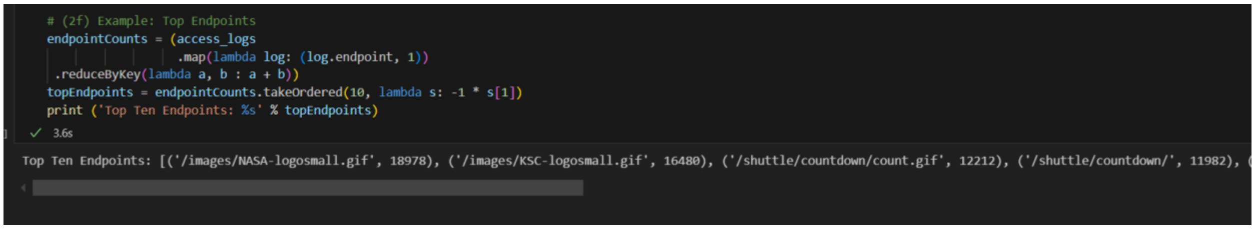
```python
# (2e) Example: Visualizing Endpoints
endpoints = (access_logs
             .map(lambda log: (log.endpoint, 1))
 .reduceByKey(lambda a, b : a + b)
 .cache())
ends = endpoints.map(lambda pair: pair[0]).collect()
counts = endpoints.map(lambda pair: pair[1]).collect()
fig = plt.figure(figsize=(8,4.2), facecolor='white', edgecolor='white')
plt.axis([0, len(ends), 0, max(counts)])
plt.grid(visible=True, which='major', axis='y')
plt.xlabel('Endpoints')
plt.ylabel('Number of Hits')
plt.plot(counts)
```



```
[<matplotlib.lines.Line2D at 0x2abac601900>]
```

Endpoint Analysis and Visualization

Utilized `reduceByKey` to rank endpoints. Top 10 were visualized using a matplotlib plot.

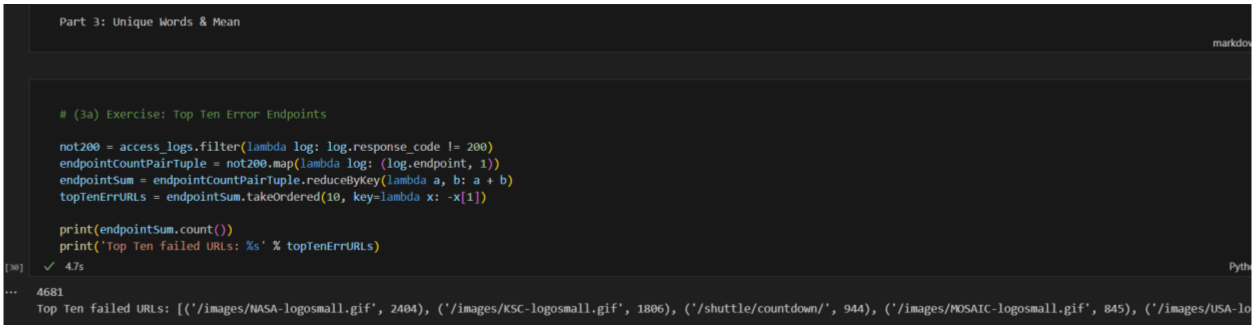


```python
# (2f) Example: Top Endpoints
endpointCounts = (access_logs
                  .map(lambda log: (log.endpoint, 1))
 .reduceByKey(lambda a, b : a + b))
topEndpoints = endpointCounts.takeOrdered(10, lambda s: -1 * s[1])
print ('Top Ten Endpoints: %s' % topEndpoints)
```

```
Top Ten Endpoints: [('/images/NASA-logosmall.gif', 18978), ('/images/KSC-logosmall.gif', 16480), ('/shuttle/countdown/count.gif', 12212), ('/shuttle/countdown/', 11982), (
```

Error Endpoint Detection

Filtered records where response code ≠ 200 and sorted them by frequency to isolate the top 10 problematic endpoints.



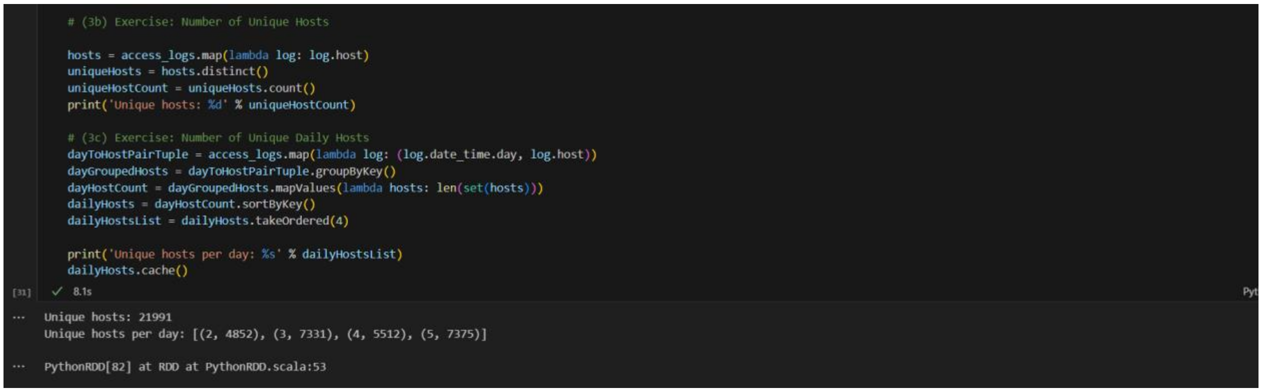
```
Part 3: Unique Words & Mean
```

```
# (3a) Exercise: Top Ten Error Endpoints

not200 = access_logs.filter(lambda log: log.response_code != 200)
endpointCountPairTuple = not200.map(lambda log: (log.endpoint, 1))
endpointSum = endpointCountPairTuple.reduceByKey(lambda a, b: a + b)
topTenErrURLs = endpointSum.takeOrdered(10, key=lambda x: -x[1])

print(endpointSum.count())
print('Top Ten failed URLs: %s' % topTenErrURLs)
```

```
4681
Top Ten failed URLs: [('/images/NASA-logosmall.gif', 2404), ('/images/KSC-logosmall.gif', 1806), ('/shuttle/countdown/', 944), ('/images/MOSAIC-logosmall.gif', 845), ('/images/USA-lo
```

Unique Host Count

Calculated the number of distinct hosts across the logs.

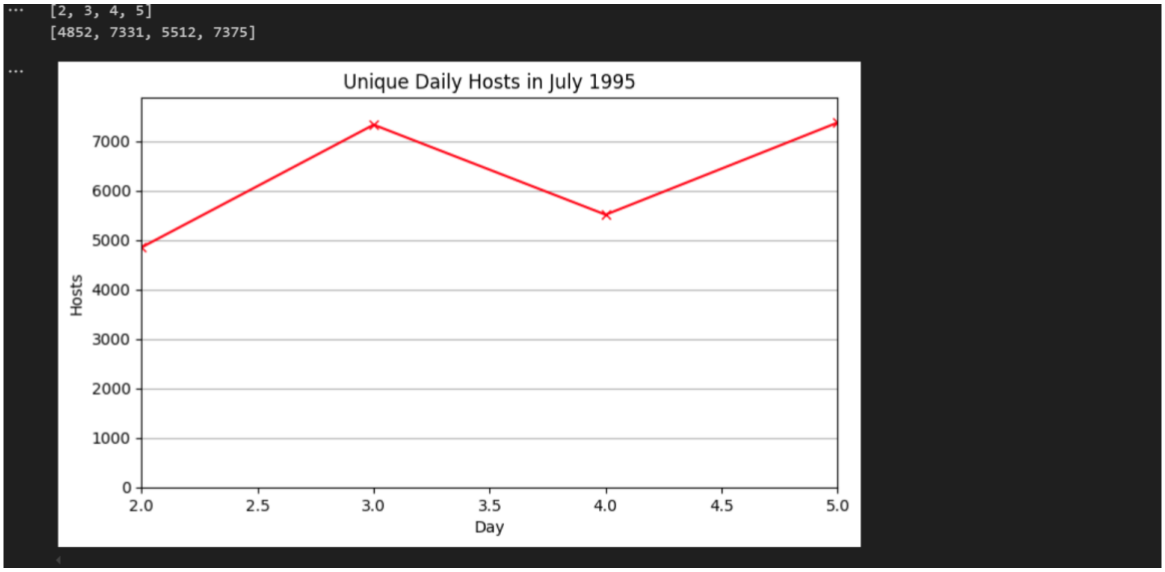


```
# (3b) Exercise: Number of Unique Hosts

hosts = access_logs.map(lambda log: log.host)
uniqueHosts = hosts.distinct()
uniqueHostCount = uniqueHosts.count()
print('Unique hosts: %d' % uniqueHostCount)

# (3c) Exercise: Number of Unique Daily Hosts
dayToHostPairTuple = access_logs.map(lambda log: (log.date_time.day, log.host))
dayGroupedHosts = dayToHostPairTuple.groupByKey()
dayHostCount = dayGroupedHosts.mapValues(lambda hosts: len(set(hosts)))
dailyHosts = dayHostCount.sortByKey()
dailyHostsList = dailyHosts.takeOrdered(4)

print('Unique hosts per day: %s' % dailyHostsList)
dailyHosts.cache()
```

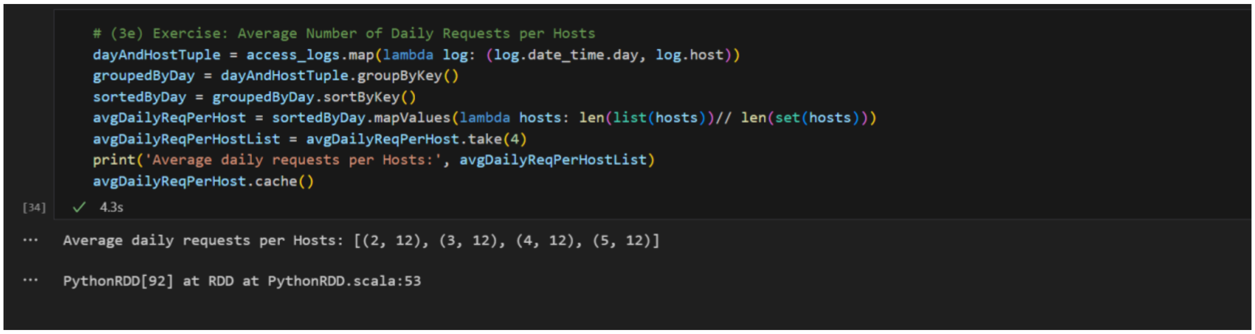
```
Unique hosts: 21991
Unique hosts per day: [(2, 4852), (3, 7331), (4, 5512), (5, 7375)]

PythonRDD[82] at RDD at PythonRDD.scala:53
```

Daily Unique Host Visualization

```
# (3d) Exercise: Visualizing the Number of Unique Daily Hosts

daysWithHosts = dailyHosts.map(lambda x: x[0]).collect()
hosts = dailyHosts.map(lambda x: x[1]).collect()
print(daysWithHosts)  # [2, 3, 4, 5, ...]
print(hosts)          # [4859, 7336, 5524, 7383, ...]

fig = plt.figure(figsize=(8, 4.5), facecolor='white', edgecolor='white')
plt.axis([min(daysWithHosts), max(daysWithHosts), 0, max(hosts) + 500])
plt.grid(visible=True, which='major', axis='y')
plt.xlabel('Day')
plt.ylabel('Hosts')
plt.plot(daysWithHosts, hosts, marker='x', color='red')
plt.title('Unique Daily Hosts in July 1995')
plt.show()
```



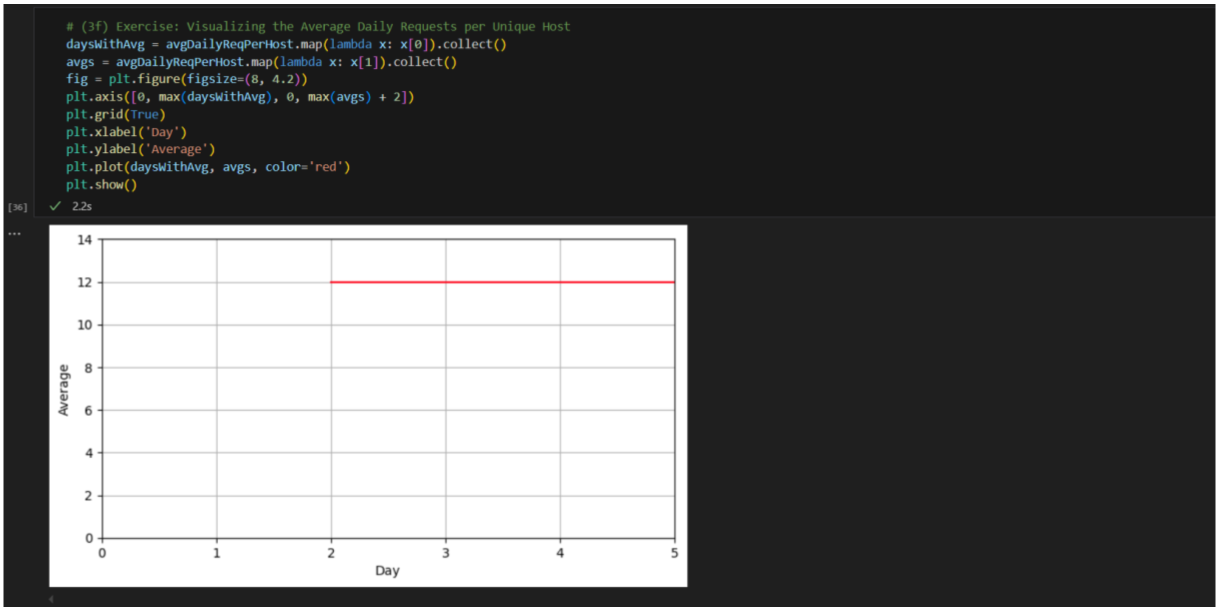
```
[2, 3, 4, 5]
[4852, 7331, 5512, 7375]
```



```
# (3e) Exercise: Average Number of Daily Requests per Hosts
dayAndHostTuple = access_logs.map(lambda log: (log.date_time.day, log.host))
groupedByDay = dayAndHostTuple.groupByKey()
sortedByDay = groupedByDay.sortByKey()
avgDailyReqPerHost = sortedByDay.mapValues(lambda hosts: len(list(hosts))// len(set(hosts)))
avgDailyReqPerHostList = avgDailyReqPerHost.take(4)
print('Average daily requests per Hosts:', avgDailyReqPerHostList)
avgDailyReqPerHost.cache()
```

[34] ✓ 4.3s

```
Average daily requests per Hosts: [(2, 12), (3, 12), (4, 12), (5, 12)]

PythonRDD[92] at RDD at PythonRDD.scala:53
```

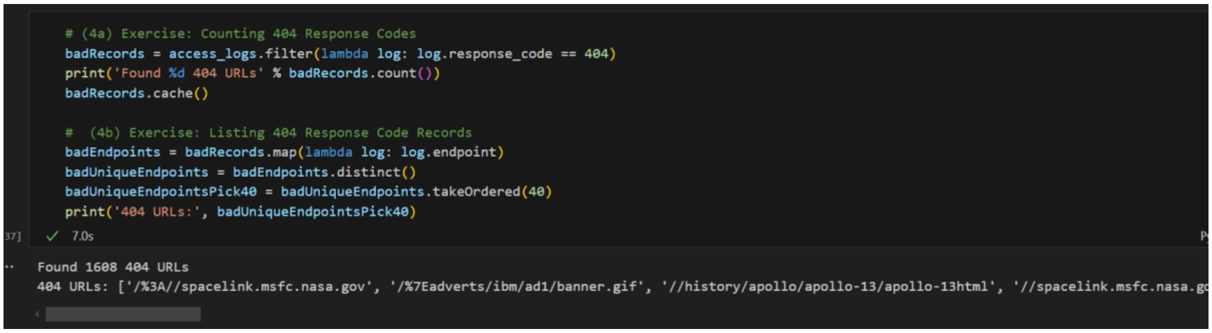


```
# (3f) Exercise: Visualizing the Average Daily Requests per Unique Host
daysWithAvg = avgDailyReqPerHost.map(lambda x: x[0]).collect()
avgs = avgDailyReqPerHost.map(lambda x: x[1]).collect()
fig = plt.figure(figsize=(8, 4.2))
plt.axis([0, max(daysWithAvg), 0, max(avgs) + 2])
plt.grid(True)
plt.xlabel('Day')
plt.ylabel('Average')
plt.plot(daysWithAvg, avgs, color='red')
plt.show()
```

[36] ✓ 2.2s

404 Response Analysis

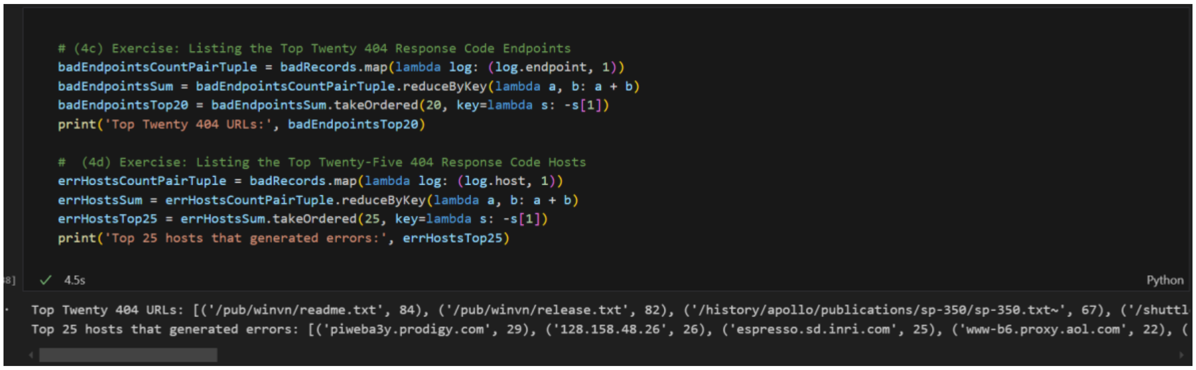
Explored all records that returned a 404 response.



```
# (4a) Exercise: Counting 404 Response Codes
badRecords = access_logs.filter(lambda log: log.response_code == 404)
print('Found %d 404 URLs' % badRecords.count())
badRecords.cache()

#  (4b) Exercise: Listing 404 Response Code Records
badEndpoints = badRecords.map(lambda log: log.endpoint)
badUniqueEndpoints = badEndpoints.distinct()
badUniqueEndpointsPick40 = badUniqueEndpoints.takeOrdered(40)
print('404 URLs:', badUniqueEndpointsPick40)
```

37] ✓ 7.0s

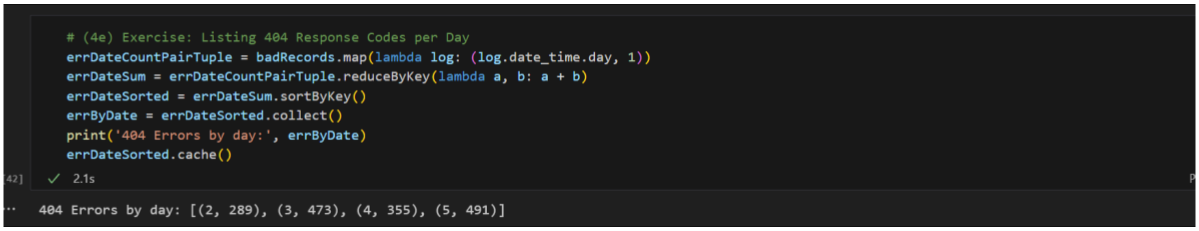
```
Found 1608 404 URLs
404 URLs: ['/%3A//spacelink.msfc.nasa.gov', '/%7Eadverts/ibm/ad1/banner.gif', '//history/apollo/apollo-13/apollo-13html', '//spacelink.msfc.nasa.go
```



```
# (4c) Exercise: Listing the Top Twenty 404 Response Code Endpoints
badEndpointsCountPairTuple = badRecords.map(lambda log: (log.endpoint, 1))
badEndpointsSum = badEndpointsCountPairTuple.reduceByKey(lambda a, b: a + b)
badEndpointsTop20 = badEndpointsSum.takeOrdered(20, key=lambda s: -s[1])
print('Top Twenty 404 URLs:', badEndpointsTop20)

#  (4d) Exercise: Listing the Top Twenty-Five 404 Response Code Hosts
errHostsCountPairTuple = badRecords.map(lambda log: (log.host, 1))
errHostsSum = errHostsCountPairTuple.reduceByKey(lambda a, b: a + b)
errHostsTop25 = errHostsSum.takeOrdered(25, key=lambda s: -s[1])
print('Top 25 hosts that generated errors:', errHostsTop25)
```

8] ✓ 4.5s Python

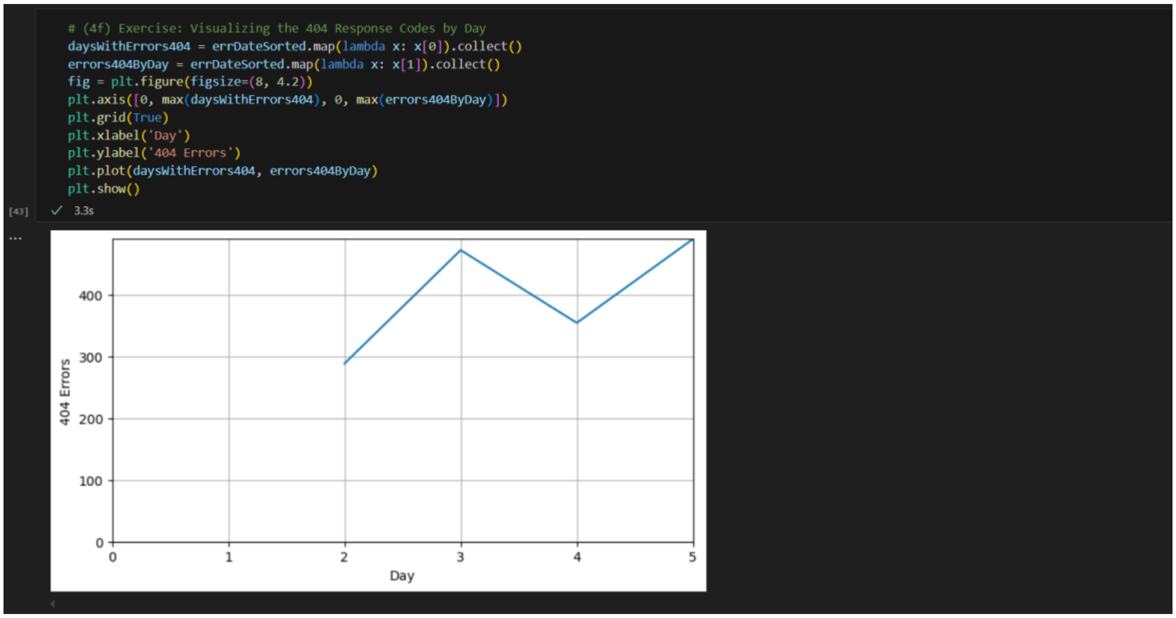
```
Top Twenty 404 URLs: [('/pub/winvn/readme.txt', 84), ('/pub/winvn/release.txt', 82), ('/history/apollo/publications/sp-350/sp-350.txt~', 67), ('/shuttl
Top 25 hosts that generated errors: [('piweba3y.prodigy.com', 29), ('128.158.48.26', 26), ('espresso.sd.inri.com', 25), ('www-b6.proxy.aol.com', 22), (
```



```
# (4e) Exercise: Listing 404 Response Codes per Day
errDateCountPairTuple = badRecords.map(lambda log: (log.date_time.day, 1))
errDateSum = errDateCountPairTuple.reduceByKey(lambda a, b: a + b)
errDateSorted = errDateSum.sortByKey()
errByDate = errDateSorted.collect()
print('404 Errors by day:', errByDate)
errDateSorted.cache()
```

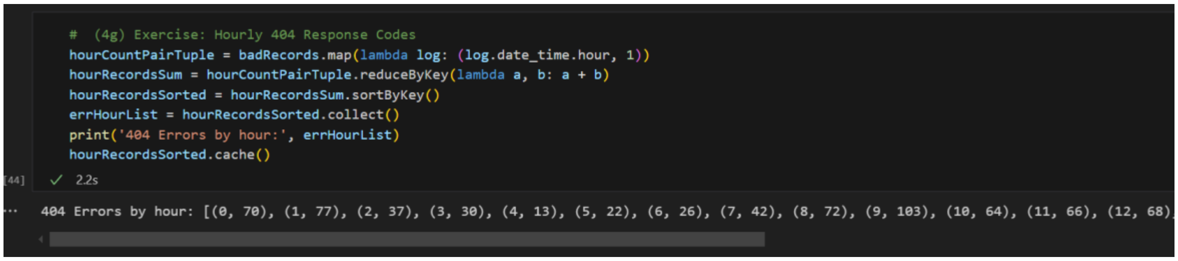
[42] ✓ 2.1s

```
404 Errors by day: [(2, 289), (3, 473), (4, 355), (5, 491)]
```



```
# (4f) Exercise: Visualizing the 404 Response Codes by Day
daysWithErrors404 = errDateSorted.map(lambda x: x[0]).collect()
errors404ByDay = errDateSorted.map(lambda x: x[1]).collect()
fig = plt.figure(figsize=(8, 4.2))
plt.axis([0, max(daysWithErrors404), 0, max(errors404ByDay)])
plt.grid(True)
plt.xlabel('Day')
plt.ylabel('404 Errors')
plt.plot(daysWithErrors404, errors404ByDay)
plt.show()
```

[43] ✓ 3.3s

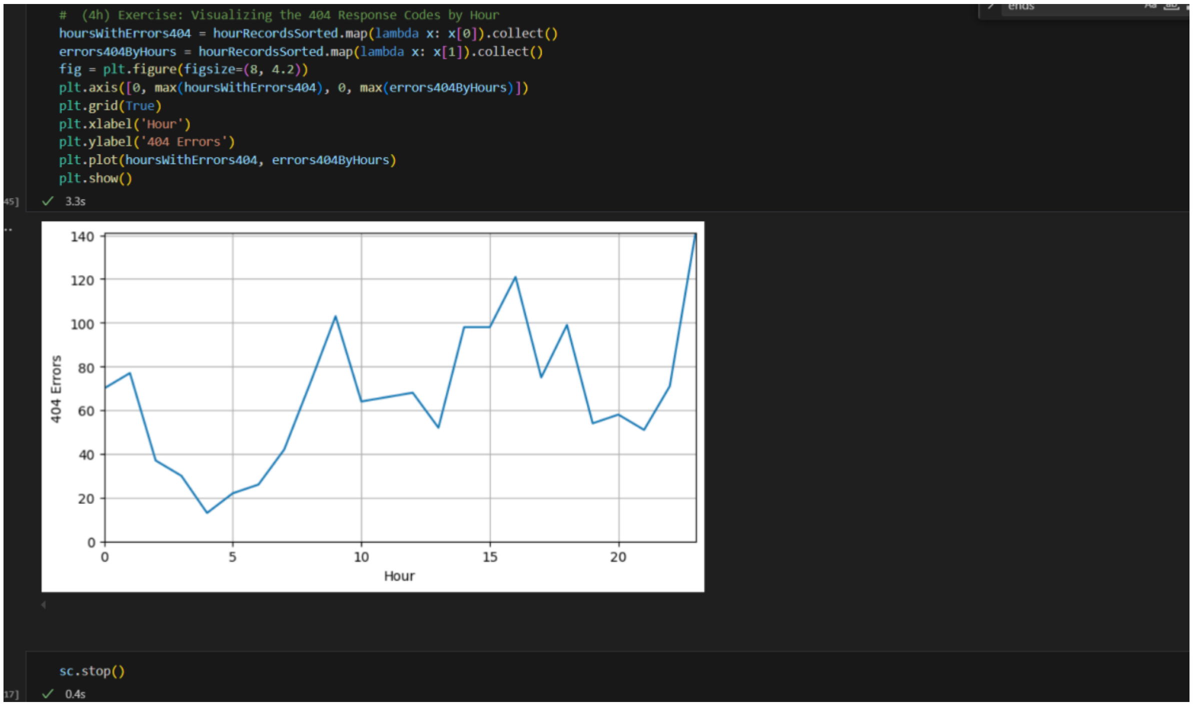


```
#  (4g) Exercise: Hourly 404 Response Codes
hourCountPairTuple = badRecords.map(lambda log: (log.date_time.hour, 1))
hourRecordsSum = hourCountPairTuple.reduceByKey(lambda a, b: a + b)
hourRecordsSorted = hourRecordsSum.sortByKey()
errHourList = hourRecordsSorted.collect()
print('404 Errors by hour:', errHourList)
hourRecordsSorted.cache()
```

```
404 Errors by hour: [(0, 70), (1, 77), (2, 37), (3, 30), (4, 13), (5, 22), (6, 26), (7, 42), (8, 72), (9, 103), (10, 64), (11, 66), (12, 68)
```

Final Steps

Visualized the findings and gracefully terminated the Spark context.



```
#  (4h) Exercise: Visualizing the 404 Response Codes by Hour
hoursWithErrors404 = hourRecordsSorted.map(lambda x: x[0]).collect()
errors404ByHours = hourRecordsSorted.map(lambda x: x[1]).collect()
fig = plt.figure(figsize=(8, 4.2))
plt.axis([0, max(hoursWithErrors404), 0, max(errors404ByHours)])
plt.grid(True)
plt.xlabel('Hour')
plt.ylabel('404 Errors')
plt.plot(hoursWithErrors404, errors404ByHours)
plt.show()
```

```
sc.stop()
```

Challenges and Resolutions

1. **PySpark Module Not Found in Jupyter**
   - **Resolution:** Installed PySpark in Jupyter, but due to further issues, switched to VS Code where the environment was already configured correctly.
2. **AttributeError for Fields like `log.ip` or `log.date_time`**
   - **Resolution:** This error occurred because the raw text was not parsed into structured Row objects. Solved by applying the `parseLogs()` function before transformations.
3. **SparkContext Initialization Fails (ConnectionRefusedError)**
   - **Resolution:** Resolved by checking for existing SparkContexts and stopping them before starting a new one.
4. **Grid Plotting Error in Matplotlib**
   - **Issue:** Using `plt.grid(b=True)` raised a warning.
   - **Fix:** Replaced with `visible=True` as `b` is deprecated.