Part 1 - Setting up Dependencies

In [1]:
```
import re
import pandas as pd

from pyspark.context import SparkContext
from pyspark.sql.context import SQLContext
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import sum as spark_sum
from pyspark.sql.functions import col, date_format, count, desc, max, regexp_extract
from pyspark.sql.types import DateType

sc = SparkContext()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/06/28 23:46:32 WARN NativeCodeLoader: Unable to load native-hadoop library for your platf
orm... using builtin-java classes where applicable
```

In [2]:
```
sc.stop()
```

In [2]:
```
sqlContext = SQLContext(sc)
spark = SparkSession(sc)
```

```
/opt/anaconda3/envs/spark/lib/python3.10/site-packages/pyspark/sql/context.py:112: FutureWar
ning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
  warnings.warn(
```

In [3]:
```
spark
```

Out[3]:

SparkSession - in-memory

SparkContext

[Spark UI (http://theo-air.attlocal.net:4040)](http://theo-air.attlocal.net:4040)

Version

`v3.4.1`

Master

`local[*]`

AppName

`pyspark-shell`

In [4]:
```
sqlContext
```

Out[4]:
```
<pyspark.sql.context.SQLContext at 0x7fe9007063b0>
```

Part 2 - Loading and Viewing the Log Dataset

Given that our data is stored in the following mentioned path, let's load it into a DataFrame. We'll do this in steps. First, we'll use `sqlContext.read.text()` or `spark.read.text()` to read the text file. This will produce a DataFrame with a single string column called `value`.

Taking a look at the metadata of our dataframe

In [19]:
```
logs_df = spark.read.csv('zemin2/kafka_test/processed/part-00000-992b77fc-636a-4c5b-b001-4b4f7
logs_df.printSchema()
```

```
root
 |-- host: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- method: string (nullable = true)
 |-- endpoint: string (nullable = true)
 |-- protocol: string (nullable = true)
 |-- status: integer (nullable = true)
 |-- content_size: integer (nullable = true)
```

Let's take a look at our dataset dimensions

In [14]:
```
print((logs_df.count(), len(logs_df.columns)))
```

```
(722, 7)
```

In [15]:
```
logs_df.show(10)
```

```
+-------------+-------------------+------+--------------------+--------+------+------------+
|         host|          timestamp|method|            endpoint|protocol|status|content_size|
+-------------+-------------------+------+--------------------+--------+------+------------+
|144.59.58.223|2005-09-18 00:36:14|DELETE|/Archives/edgar/d...|HTTP/1.0|   304|       19012|
|144.59.58.223|2005-09-18 00:36:14|DELETE|/Archives/edgar/d...|HTTP/1.0|   304|       22336|
|144.59.58.223|2005-09-18 00:36:14|DELETE|/Archives/edgar/d...|HTTP/1.0|   304|        4174|
| 82.81.29.236|2005-04-12 09:53:12|  POST|/Archives/edgar/d...|HTTP/1.0|   303|       58484|
| 82.81.29.236|2005-04-12 09:53:12|  POST|/Archives/edgar/d...|HTTP/1.0|   303|       21831|
| 82.81.29.236|2005-04-12 09:53:12|  POST|/Archives/edgar/d...|HTTP/1.0|   303|       26174|
| 82.81.29.236|2005-04-12 09:53:12|  POST|/Archives/edgar/d...|HTTP/1.0|   303|       18933|
| 82.81.29.236|2005-04-12 09:53:12|  POST|/Archives/edgar/d...|HTTP/1.0|   303|       18549|
| 82.81.29.236|2005-04-12 09:53:12|  POST|/Archives/edgar/d...|HTTP/1.0|   303|        8838|
| 82.81.29.236|2005-04-12 09:53:12|  POST|/Archives/edgar/d...|HTTP/1.0|   303|       13316|
+-------------+-------------------+------+--------------------+--------+------+------------+
only showing top 10 rows
```

Part 4 - Data Analysis on our Web Logs

Now that we have a DataFrame containing the parsed log file as a data frame, we can perform some interesting exploratory data analysis (EDA)

Content Size Statistics

Let's compute some statistics about the sizes of content being returned by the web server. In particular, we'd like to know what are the average, minimum, and maximum content sizes.

We can compute the statistics by calling `.describe()` on the `content_size` column of `logs_df`. The `.describe()` function returns the count, mean, stddev, min, and max of a given column.

In [16]:
```
content_size_summary_df = logs_df.describe(['content_size'])
content_size_summary_df.toPandas()
```

Out[16]:

	summary	content_size
0	count	722
1	mean	29212.96537396122
2	stddev	16965.937297998917
3	min	10
4	max	59905

Alternatively, we can use SQL to directly calculate these statistics. You can explore many useful functions within the `pyspark.sql.functions` module in the [documentation (https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.functions)](https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.functions).

After we apply the `.agg()` function, we call `toPandas()` to extract and convert the result into a `pandas` dataframe which has better formatting on Jupyter notebooks

In [20]:
```
(logs_df.agg(F.min(logs_df['content_size']).alias('min_content_size'),
             F.max(logs_df['content_size']).alias('max_content_size'),
             F.mean(logs_df['content_size']).alias('mean_content_size'),
             F.stddev(logs_df['content_size']).alias('std_content_size'),
             F.count(logs_df['content_size']).alias('count_content_size'))
        .toPandas())
```

Out[20]:

	min_content_size	max_content_size	mean_content_size	std_content_size	count_content_size
0	10	59905	29212.965374	16965.937298	722

HTTP Status Code Analysis

Next, let's look at the status code values that appear in the log. We want to know which status code values appear in the data and how many times.

We again start with `logs_df`, then group by the `status` column, apply the `.count()` aggregation function, and sort by the `status` column.

In [21]:
```
status_freq_df = (logs_df
                     .groupBy('status')
                     .count()
                     .sort('status')
                     .cache())
```

In [22]:
```
print('Total distinct HTTP Status Codes:', status_freq_df.count())
```

```
[Stage 27:>                                                         (0 + 2) / 2]

Total distinct HTTP Status Codes: 2
```

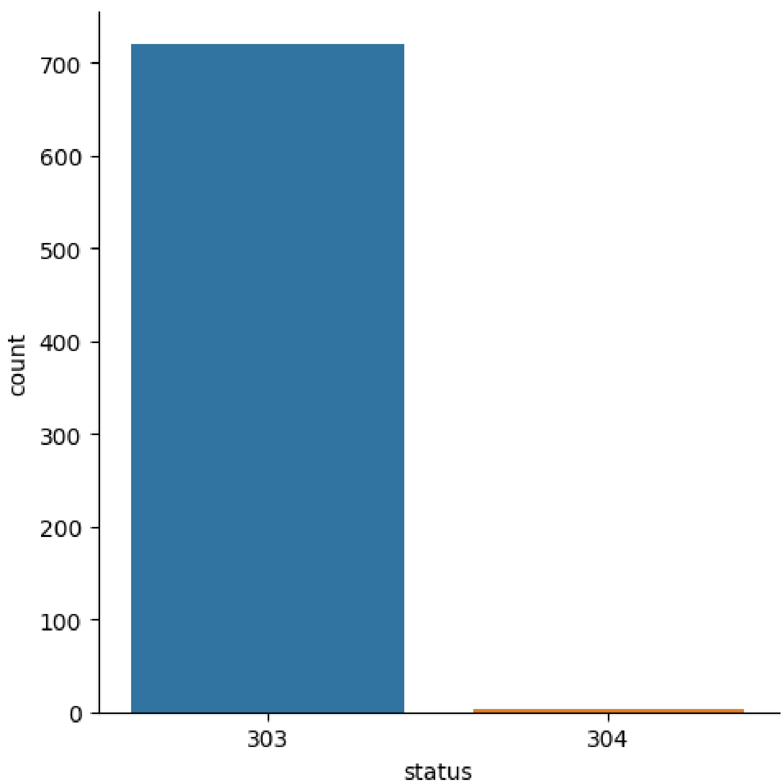
In [23]:
```
status_freq_pd_df = (status_freq_df
                         .toPandas()
                         .sort_values(by=['count'],
                                      ascending=False))
status_freq_pd_df
```

Out[23]:

	status	count
0	303	719
1	304	3

In [24]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline

sns.catplot(x='status', y='count', data=status_freq_pd_df,
            kind='bar', order=status_freq_pd_df['status'])
```

Out[24]: <seaborn.axisgrid.FacetGrid at 0x7fe901ee7a90>

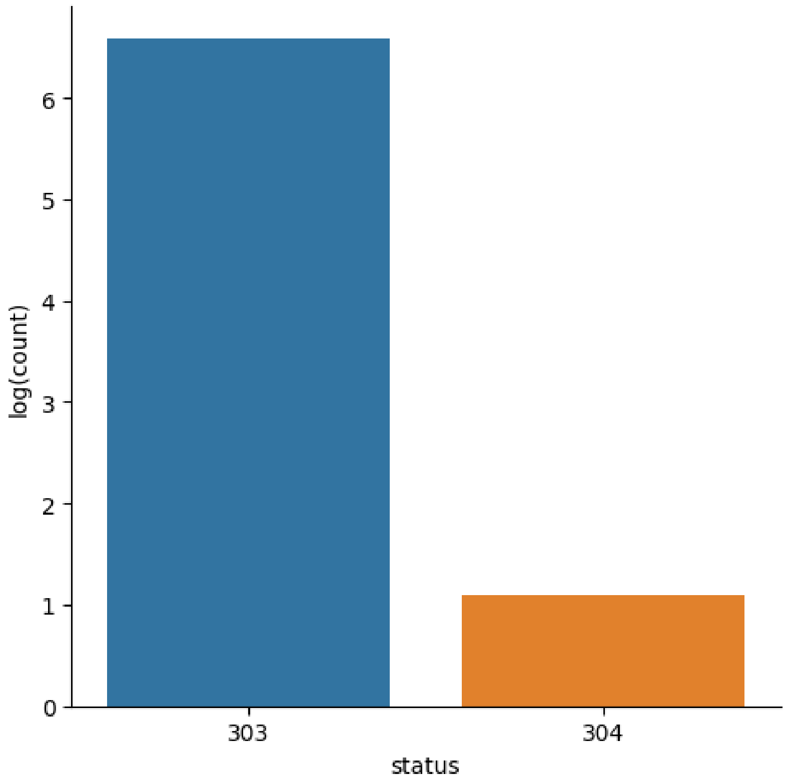


In [25]:
```
log_freq_df = status_freq_df.withColumn('log(count)', F.log(status_freq_df['count']))
log_freq_df.show()
```

```
+------+-----+------------------+
|status|count|        log(count)|
+------+-----+------------------+
|   303|  719| 6.577861357721047|
|   304|    3|1.0986122886681096|
+------+-----+------------------+
```

In [26]:
```
log_freq_pd_df = (log_freq_df
                    .toPandas()
                    .sort_values(by=['log(count)'],
                                 ascending=False))
sns.catplot(x='status', y='log(count)', data=log_freq_pd_df,
            kind='bar', order=status_freq_pd_df['status'])
```

Out[26]:
```
<seaborn.axisgrid.FacetGrid at 0x7fe902003cd0>
```



Analyzing Frequent Hosts

Let's look at hosts that have accessed the server frequently. We will try to get the count of total accesses by each `host` and then sort by the counts and display only the top ten most frequent hosts.

In [27]:
```
host_sum_df =(logs_df
              .groupBy('host')
              .count()
              .sort('count', ascending=False).limit(10))

host_sum_df.show(truncate=False)
```

```
+-------------+-----+
|host         |count|
+-------------+-----+
|82.81.29.236 |719  |
|144.59.58.223|3    |
+-------------+-----+
```

In [45]:
```
host_sum_pd_df = host_sum_df.toPandas()
host_sum_pd_df
```

Out[45]:

	host	count
0	82.81.29.236	719
1	144.59.58.223	3

Looks like we have some empty strings as one of the top host names! This teaches us a valuable lesson to not just check for nulls but also potentially empty strings when data wrangling.

Display the Top 20 Frequent EndPoints

Now, let's visualize the number of hits to endpoints (URIs) in the log. To perform this task, we start with our `logs_df` and group by the `endpoint` column, aggregate by count, and sort in descending order like the previous question.

In [46]:
```
paths_df = (logs_df
            .groupBy('endpoint')
            .count()
            .sort('count', ascending=False).limit(20))
```

In [47]:
```
paths_pd_df = paths_df.toPandas()
paths_pd_df
```

Out[47]:

	endpoint	count
0	/Archives/edgar/data/0001457049/00015745961800...	719
1	/Archives/edgar/data/0001203957/00011931251022...	3

Top Ten Error Endpoints

What are the top ten endpoints requested which did not have return code 200 (HTTP Status OK)?

We create a sorted list containing the endpoints and the number of times that they were accessed with a non-200 return code and show the top ten.

In [48]:
```
not200_df = (logs_df
               .filter(logs_df['status'] != 200))

error_endpoints_freq_df = (not200_df
                               .groupBy('endpoint')
                               .count()
                               .sort('count', ascending=False)
                               .limit(10)
                          )
```

In [49]:
```
error_endpoints_freq_df.show(truncate=False)
```

```
+--------------------------------------------------------------------------------+-----+
|endpoint                                                                        |count|
+--------------------------------------------------------------------------------+-----+
|/Archives/edgar/data/0001457049/000157459618000048/0001574596-18-000048-index.htm|719  |
|/Archives/edgar/data/0001203957/000119312510223947/0001193125-10-223947-index.htm|3    |
+--------------------------------------------------------------------------------+-----+
```

Total number of Unique Hosts

What were the total number of unique hosts who visited the NASA website in these two months? We can find this out with a few transformations.

In [50]:
```
unique_host_count = (logs_df
                     .select('host')
                     .distinct()
                     .count())
unique_host_count
```

Out[50]:
```
2
```

Number of Unique Daily Hosts

For an advanced example, let's look at a way to determine the number of unique hosts in the entire log on a day-by-day basis. This computation will give us counts of the number of unique daily hosts.

We'd like a DataFrame sorted by increasing day of the month which includes the day of the month and the associated number of unique hosts for that day.

Think about the steps that you need to perform to count the number of different hosts that make requests *each* day. *Since the log only covers a single month, you can ignore the month.* You may want to use the `dayofmonth` function (https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.functions.dayofmonth) in the `pyspark.sql.functions` module (which we have already imported as **`F`**.

`host_day_df`

A DataFrame with two columns

column	explanation
`host`	the host name
`day`	the day of the month

There will be one row in this DataFrame for each row in `logs_df`. Essentially, we are just transforming each row of `logs_df`. For example, for this row in `logs_df`:

```
unicomp6.unicomp.net - - [01/Aug/1995:00:35:41 -0400] "GET /shuttle/missions/sts-73/ne
ws HTTP/1.0" 302 -
```

your `host_day_df` should have:

```
unicomp6.unicomp.net 1
```

In [51]:
```
logs_df.show(1, truncate=False)
```

```
+-------------+-------------------+------+-----------------------------------------------------
-------------------------------+--------+------+------------+
|host         |timestamp          |method|endpoint
|protocol|status|content_size|
+-------------+-------------------+------+-----------------------------------------------------
-------------------------------+--------+------+------------+
|144.59.58.223|2005-09-18 00:36:14|DELETE|/Archives/edgar/data/0001203957/00011931251022394
7/0001193125-10-223947-index.htm|HTTP/1.0|304   |19012       |
+-------------+-------------------+------+-----------------------------------------------------
-------------------------------+--------+------+------------+
only showing top 1 row
```

In [52]:
```
host_day_df = logs_df.select(logs_df.host, F.dayofmonth('timestamp').alias('day'))
host_day_df.show(5, truncate=False)
```

```
+-------------+---+
|host         |day|
+-------------+---+
|144.59.58.223|18 |
|144.59.58.223|18 |
|144.59.58.223|18 |
|82.81.29.236 |12 |
|82.81.29.236 |12 |
+-------------+---+
only showing top 5 rows
```

`host_day_distinct_df`

This DataFrame has the same columns as `host_day_df` , but with duplicate (`day` , `host`) rows removed.

In [53]:
```
host_day_distinct_df = (host_day_df
                          .dropDuplicates())
host_day_distinct_df.show(5, truncate=False)
```

```
+-------------+---+
|host         |day|
+-------------+---+
|144.59.58.223|18 |
|82.81.29.236 |12 |
+-------------+---+
```

`daily_unique_hosts_df`

A DataFrame with two columns:

column	explanation
`day`	the day of the month
`count`	the number of unique requesting hosts for that day

Average Number of Daily Requests per Host

In the previous example, we looked at a way to determine the number of unique hosts in the entire log on a day-by-day basis. Let's now try and find the average number of requests being made per Host to the NASA website per day based on our logs.

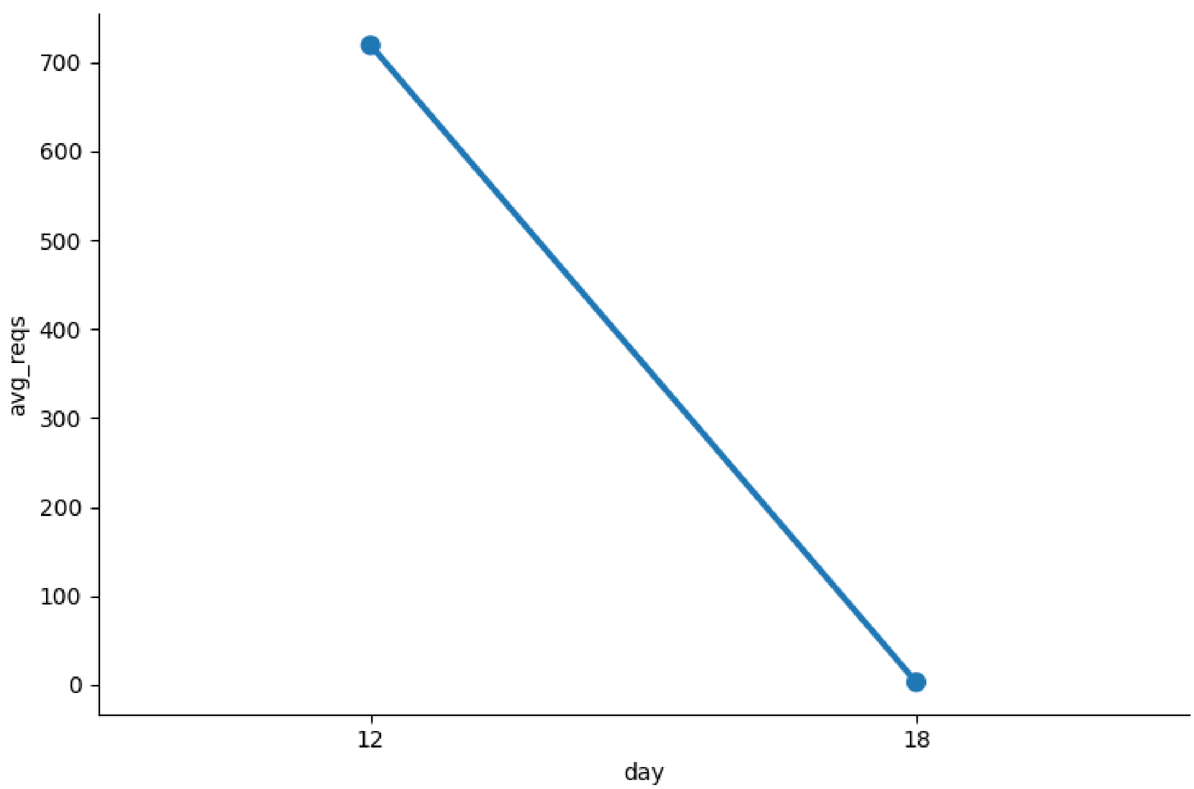
We'd like a DataFrame sorted by increasing day of the month which includes the day of the month and the associated number of average requests made for that day per Host.

In [55]:
```
daily_hosts_df = (host_day_distinct_df
                     .groupBy('day')
                     .count()
                     .select(col("day"),
                                  col("count").alias("total_hosts")))

total_daily_reqests_df = (logs_df
                              .select(F.dayofmonth("timestamp")
                                          .alias("day"))
                              .groupBy("day")
                              .count()
                              .select(col("day"),
                                  col("count").alias("total_reqs")))

avg_daily_reqests_per_host_df = total_daily_reqests_df.join(daily_hosts_df, 'day')
avg_daily_reqests_per_host_df = (avg_daily_reqests_per_host_df
                                   .withColumn('avg_reqs', col('total_reqs') / col('total_hos
                                   .sort("day"))
avg_daily_reqests_per_host_df = avg_daily_reqests_per_host_df.toPandas()
avg_daily_reqests_per_host_df
```

Out[55]:

	day	total_reqs	total_hosts	avg_reqs
0	12	719	1	719.0
1	18	3	1	3.0

In [56]:
```
c = sns.catplot(x='day', y='avg_reqs',
                data=avg_daily_reqests_per_host_df,
                kind='point', height=5, aspect=1.5)
```



Counting 404 Response Codes

Create a DataFrame containing only log records with a 404 status code (Not Found).

We make sure to `cache()` the `not_found_df` dataframe as we will use it in the rest of the examples here.

How many 404 records are in the log?

In [57]:
```
not_found_df = logs_df.filter(logs_df["status"] == 404).cache()
print(('Total 404 responses: {}').format(not_found_df.count()))
```

```
Total 404 responses: 0

23/06/28 23:53:24 WARN CacheManager: Asked to cache already cached data.
```

Listing the Top Twenty 404 Response Code Endpoints

Using the DataFrame containing only log records with a 404 response code that we cached earlier, we will now print out a list of the top twenty endpoints that generate the most 404 errors.

Remember, top endpoints should be in sorted order

In [58]:
```
endpoints_404_count_df = (not_found_df
                          .groupBy("endpoint")
                          .count()
                          .sort("count", ascending=False)
                          .limit(20))

endpoints_404_count_df.show(truncate=False)
```

```
+--------+-----+
|endpoint|count|
+--------+-----+
+--------+-----+
```

Listing the Top Twenty 404 Response Code Hosts

Using the DataFrame containing only log records with a 404 response code that we cached earlier, we will now print out a list of the top twenty hosts that generate the most 404 errors.

Remember, top hosts should be in sorted order

In [59]:
```
hosts_404_count_df = (not_found_df
                      .groupBy("host")
                      .count()
                      .sort("count", ascending=False)
                      .limit(20))

hosts_404_count_df.show(truncate=False)
```

```
+----+-----+
|host|count|
+----+-----+
+----+-----+
```

Visualizing 404 Errors per Day

Let's explore our 404 records temporally (by time) now. Similar to the example showing the number of unique daily hosts, we will break down the 404 requests by day and get the daily counts sorted by day in `errors_by_date_sorted_df` .

In [60]:
```
errors_by_date_sorted_df = (not_found_df
                                .groupBy(F.dayofmonth('timestamp').alias('day'))
                                .count()
                                .sort("day"))

errors_by_date_sorted_pd_df = errors_by_date_sorted_df.toPandas()
errors_by_date_sorted_pd_df
```

Out[60]:

day	count

Top Three Days for 404 Errors

What are the top three days of the month having the most 404 errors, we can leverage our previously created **`errors_by_date_sorted_df`** for this.

In [ ]:
```
(errors_by_date_sorted_df
    .sort("count", ascending=False)
    .show(3))
```

Visualizing Hourly 404 Errors

Using the DataFrame `not_found_df` we cached earlier, we will now group and sort by hour of the day in increasing order, to create a DataFrame containing the total number of 404 responses for HTTP requests for each hour of the day (midnight starts at 0)

In [ ]:
```
hourly_avg_errors_sorted_df = (not_found_df
                                   .groupBy(F.hour('timestamp')
                                             .alias('hour'))
                                   .count()
                                   .sort('hour'))
hourly_avg_errors_sorted_pd_df = hourly_avg_errors_sorted_df.toPandas()
```

In [ ]:
```
c = sns.catplot(x='hour', y='count',
                data=hourly_avg_errors_sorted_pd_df,
                kind='bar', height=5, aspect=1.5)
```

Section A 3.1

In [61]:
```
# Convert the timestamp column to DateType and extract the day of the week
fre_end = logs_df.withColumn("DOY", date_format(logs_df["timestamp"].cast(DateType()), "EEEE"))
```

In [62]:
```
fre_end = fre_end[['DOY','endpoint']]
```

In [63]:
```
fre_end.show(1, truncate=False)
```

```
+------+--------------------------------------------------------------------------+
|DOY   |endpoint                                                                  |
+------+--------------------------------------------------------------------------+
|Sunday|/Archives/edgar/data/0001203957/000119312510223947/0001193125-10-223947-index.htm|
+------+--------------------------------------------------------------------------+
only showing top 1 row
```

In [64]:
```
# Group by day of the week and endpoint, and count occurrences
fre_end = fre_end.groupBy("DOY", "endpoint").agg(count("*").alias("count"))
```

In [65]:
```
fre_end.show(7, truncate=False)
```

```
+-------+--------------------------------------------------------------------------------+-
----+
|DOY    |endpoint                                                                        |c
ount|
+-------+--------------------------------------------------------------------------------+-
----+
|Tuesday|/Archives/edgar/data/0001457049/000157459618000048/0001574596-18-000048-index.htm|7
19  |
|Sunday |/Archives/edgar/data/0001203957/000119312510223947/0001193125-10-223947-index.htm|3
|
+-------+--------------------------------------------------------------------------------+-
----+
```

In [66]:
```
highest_counts = fre_end.groupBy("DOY").agg(max("count").alias("highest_count"))
```

In [67]:
```
highest = highest_counts.select(highest_counts["DOY"].alias("dayofweek"), highest_counts["highe
```

In [68]:
```
highest.show(10,truncate=False)
```

```
+---------+-------------+
|dayofweek|highest_count|
+---------+-------------+
|Tuesday  |719          |
|Sunday   |3            |
+---------+-------------+
```

In [69]:
```
result = fre_end.join(highest, (fre_end["DOY"] == highest["dayofweek"]) & (fre_end["count"] ==

# Select the desired columns for the final result
result = result.select("DOY", "endpoint", "highest_count")

# Show the result
result.show()
```

```
+-------+--------------------+-------------+
|    DOY|            endpoint|highest_count|
+-------+--------------------+-------------+
|Tuesday|/Archives/edgar/d...|          719|
| Sunday|/Archives/edgar/d...|            3|
+-------+--------------------+-------------+
```

Section A 3.2

In [70]:
```
df_404 = logs_df.withColumn("date", date_format(logs_df["timestamp"].cast(DateType()), "EEEE")
```

In [71]:
```
df_404 = df_404.filter(logs_df['status']==404)
```

In [72]:
```
result = df_404.groupBy("date").count().orderBy("date")
```

In [73]:
```
result.show()
```

```
+----+-----+
|date|count|
+----+-----+
+----+-----+
```