`

# FINAL PROJECT

## Analyse web server logs to understand website traffic patterns and optimize server performance using Spark Data Frames in Azure Synapse

**Prepared by:**
Name: Mathan Prasath R
Emp ID: 2308187
Cohort: INT2AIA23AZ001

Mentor: Kavin Kumar
Trainer: Karthick Selvam
Coach: Lavanya Palani

`

**Table of contents**

`

**Project Objective:**

The objective of this project is to Analyse web server logs to understand website traffic patterns and optimize server performance using Spark DataFrames in Azure Synapse.

**Introduction:**

In today's digital landscape, understanding website traffic patterns is crucial for optimizing server performance and enhancing user experience. In this project, we aim to leverage the power of Spark DataFrames within Azure Synapse to analyze web server logs. By doing so, we seek to gain insights into user behavior, identify issues in server performance, and ultimately enhance the efficiency and responsiveness of our web infrastructure.

**Key Steps:**

**Data Collection:** Acquired web server logs from Kaggle and uploaded the CSV file to a storage container within the Azure resource group.

**Data Ingestion:** Loaded the log data from the storage container into Azure Synapse, creating a DataFrame for further analysis.

**Analysis with Spark DataFrames:** Leveraged the powerful capabilities of Spark DataFrames within Azure Synapse to perform in-depth analysis of the log data.

**Insights Generation:** Generated insights into website traffic patterns, including peak traffic times, error analysis and more, using the analysed data.

**Dataset:**

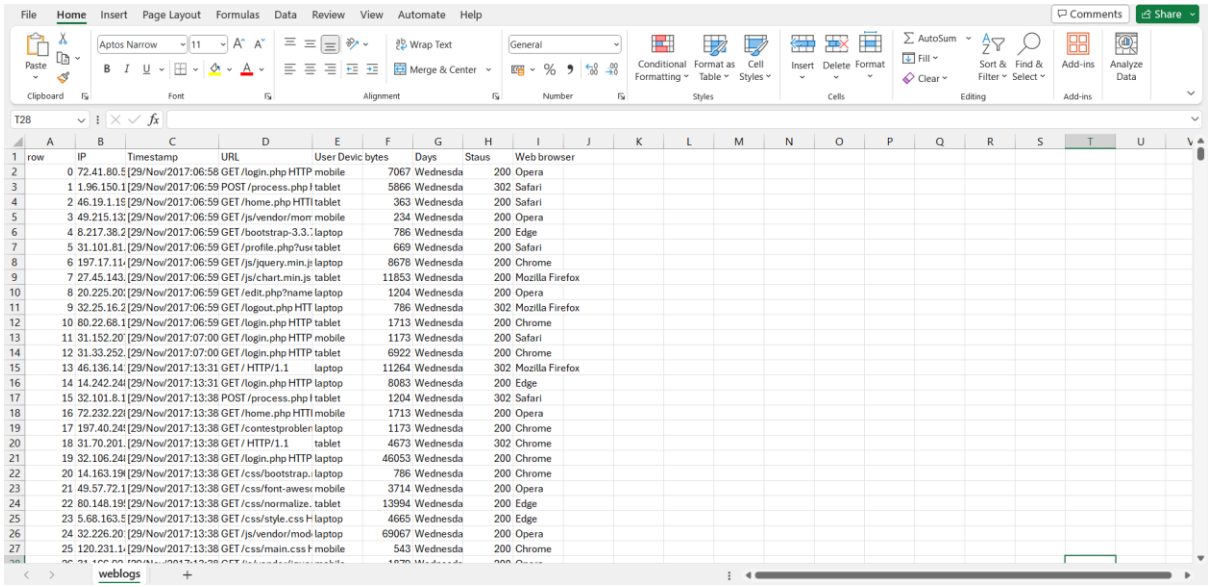
The dataset used for this project was sourced from Kaggle and comprises more than 10,500 rows with 9 columns. It encompasses various aspects of web server activity, including IP addresses, HTTP methods, error codes, user devices, and web browsers. This comprehensive dataset forms the foundation for analysing website traffic patterns and optimizing server performance.

**Dataset Used:**

Webslogs.csv This dataset contains 10600 rows and 9 columns.

`

**Web Server Log Dataset:**



| row | IP | Timestamp | URL | User Devic | bytes | Days | Staus | Web browser |
|---|---|---|---|---|---|---|---|---|
| 0 | 72.41.80.5 | [29/Nov/2017:06:58 | GET /login.php HTTP | mobile | 7067 | Wednesda | 200 | Opera |
| 1 | 1.96.150.1 | [29/Nov/2017:06:59 | POST /process.php H | tablet | 5866 | Wednesda | 302 | Safari |
| 2 | 46.19.1.15 | [29/Nov/2017:06:59 | GET /home.php HTTI | tablet | 363 | Wednesda | 200 | Safari |
| 3 | 49.215.13 | [29/Nov/2017:06:59 | GET /js/vendor/mor | mobile | 234 | Wednesda | 200 | Opera |
| 4 | 8.217.38.2 | [29/Nov/2017:06:59 | GET /bootstrap-3.3. | laptop | 786 | Wednesda | 200 | Edge |
| 5 | 31.101.81. | [29/Nov/2017:06:59 | GET /profile.php?use | tablet | 669 | Wednesda | 200 | Safari |
| 6 | 197.17.11 | [29/Nov/2017:06:59 | GET /js/jquery.min.j | laptop | 8678 | Wednesda | 200 | Chrome |
| 7 | 27.45.143. | [29/Nov/2017:06:59 | GET /js/chart.min.js | tablet | 11853 | Wednesda | 200 | Mozilla Firefox |
| 8 | 20.225.20 | [29/Nov/2017:06:59 | GET /edit.php?name | laptop | 1204 | Wednesda | 200 | Opera |
| 9 | 32.25.16.2 | [29/Nov/2017:06:59 | GET /logout.php HTT | laptop | 786 | Wednesda | 302 | Mozilla Firefox |
| 10 | 80.22.68.1 | [29/Nov/2017:06:59 | GET /login.php HTTP | tablet | 1713 | Wednesda | 200 | Chrome |
| 11 | 31.152.20 | [29/Nov/2017:07:00 | GET /login.php HTTP | mobile | 1173 | Wednesda | 200 | Safari |
| 12 | 31.33.252. | [29/Nov/2017:07:00 | GET /login.php HTTP | tablet | 6922 | Wednesda | 200 | Chrome |
| 13 | 46.136.14 | [29/Nov/2017:13:31 | GET / HTTP/1.1 | laptop | 11264 | Wednesda | 302 | Mozilla Firefox |
| 14 | 14.242.24 | [29/Nov/2017:13:31 | GET /login.php HTTP | laptop | 8083 | Wednesda | 200 | Edge |
| 15 | 32.101.8.1 | [29/Nov/2017:13:38 | POST /process.php H | tablet | 1204 | Wednesda | 302 | Safari |
| 16 | 72.232.22 | [29/Nov/2017:13:38 | GET /home.php HTTI | mobile | 1713 | Wednesda | 200 | Opera |
| 17 | 197.40.24 | [29/Nov/2017:13:38 | GET /contestproblen | laptop | 1173 | Wednesda | 200 | Chrome |
| 18 | 31.70.201. | [29/Nov/2017:13:38 | GET / HTTP/1.1 | tablet | 4673 | Wednesda | 302 | Chrome |
| 19 | 32.106.24 | [29/Nov/2017:13:38 | GET /login.php HTTP | laptop | 46053 | Wednesda | 200 | Chrome |
| 20 | 14.163.19 | [29/Nov/2017:13:38 | GET /css/bootstrap. | laptop | 786 | Wednesda | 200 | Chrome |
| 21 | 49.57.72.1 | [29/Nov/2017:13:38 | GET /css/font-awes | mobile | 3714 | Wednesda | 200 | Opera |
| 22 | 80.148.19 | [29/Nov/2017:13:38 | GET /css/normalize. | tablet | 13994 | Wednesda | 200 | Edge |
| 23 | 5.68.163.5 | [29/Nov/2017:13:38 | GET /css/style.css H | laptop | 4665 | Wednesda | 200 | Edge |
| 24 | 32.226.20 | [29/Nov/2017:13:38 | GET /js/vendor/mod | laptop | 69067 | Wednesda | 200 | Opera |
| 25 | 120.231.1 | [29/Nov/2017:13:38 | GET /css/main.css H | mobile | 543 | Wednesda | 200 | Chrome |

**Table Schema:**

- row: integer (nullable = true)
- IP: string (nullable = true)
- Timestamp: string (nullable = true)
- URL: string (nullable = true)
- User Device: string (nullable = true)
- Bytes: integer (nullable = true)
- Days: string (nullable = true)
- Status: string (nullable = true)
- Web Browser: string (nullable = true)

**About Weblog:**

The log files used for this assignment are taken from Kaggle. Each log file entry produced in CLF will look something like this: 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:

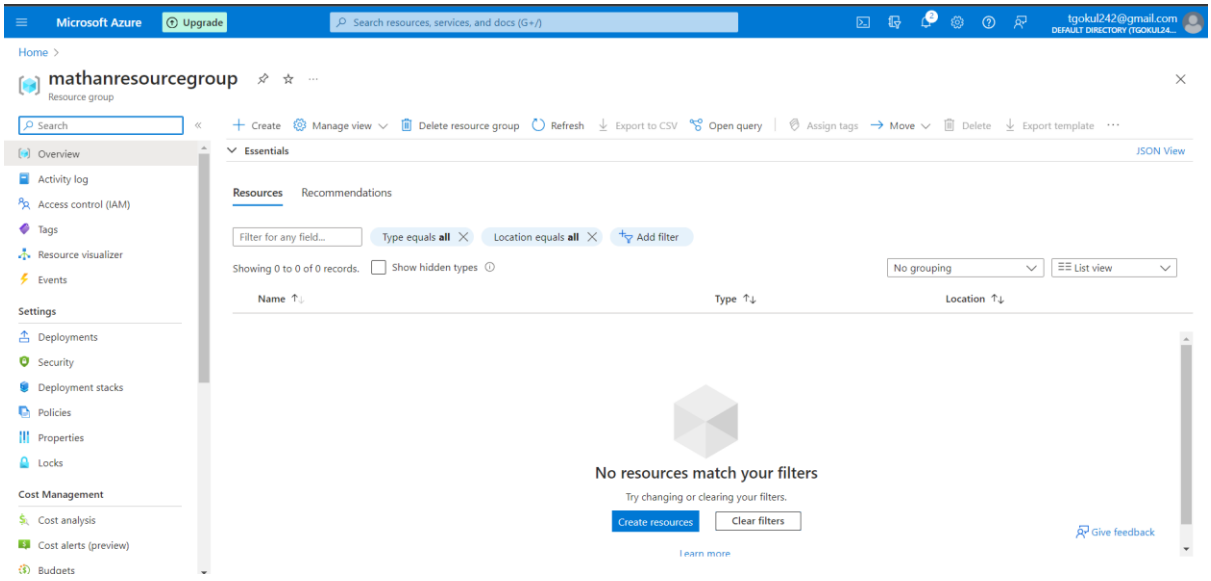
- **127.0.0.1:** This is the IP address (or host name, if available) of the client (remote host) which made the request to the server.

`

- **[01/Nov/2017:00:00:01 -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), and 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). The full list of possible status codes can be found in the HTTP specification (RFC 2616 section 10).
- **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).

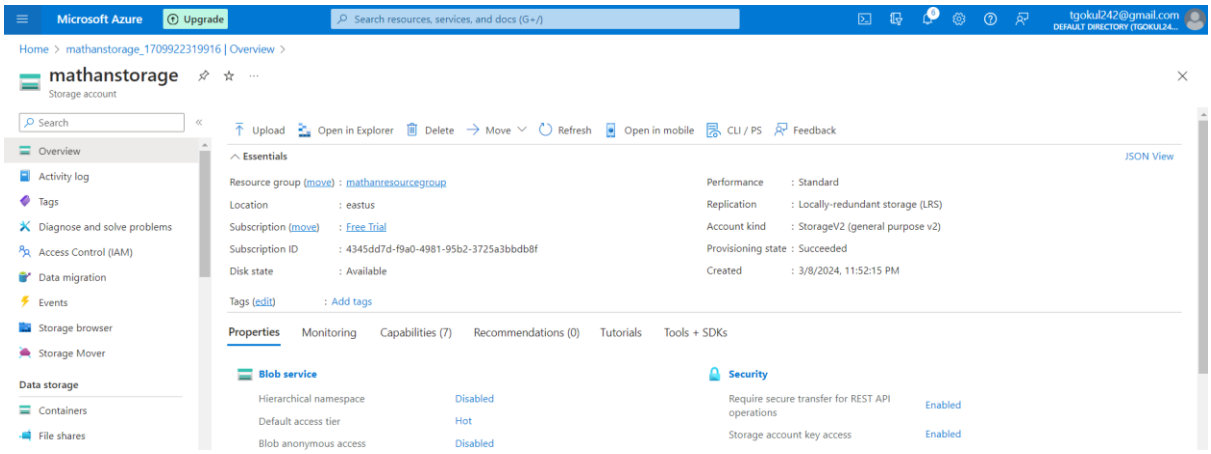
`

## Created Resource Group



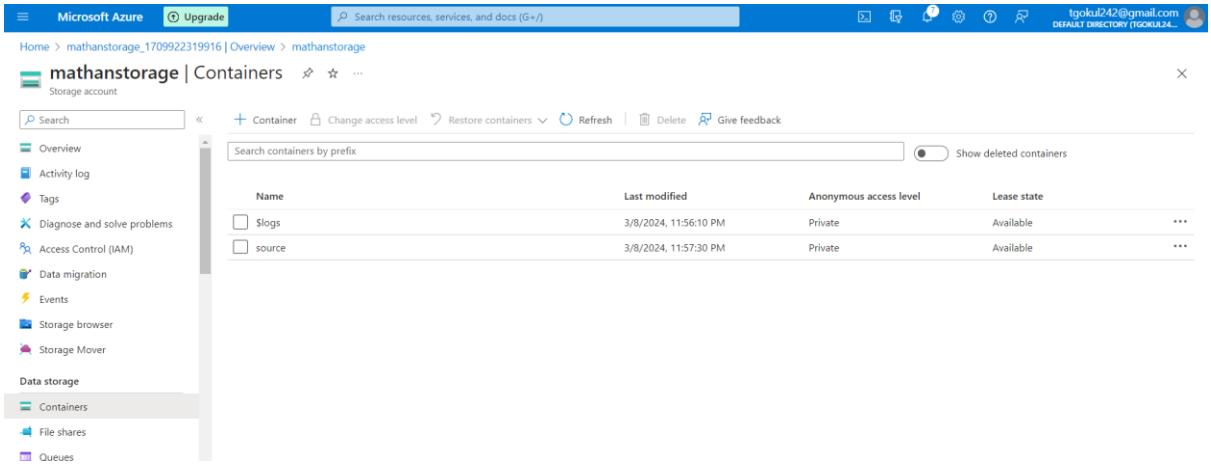

Established a resource group to keep everything organized, including storing the source files uploaded to a container within a storage account, and for housing the Synapse workspace.

## Created Storage Account

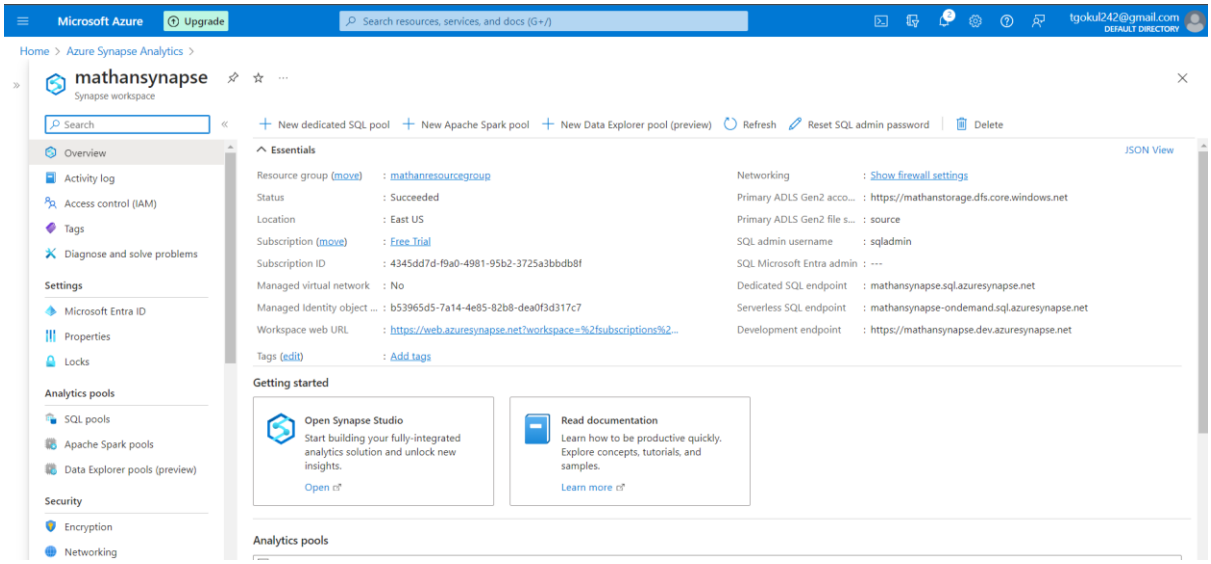


`

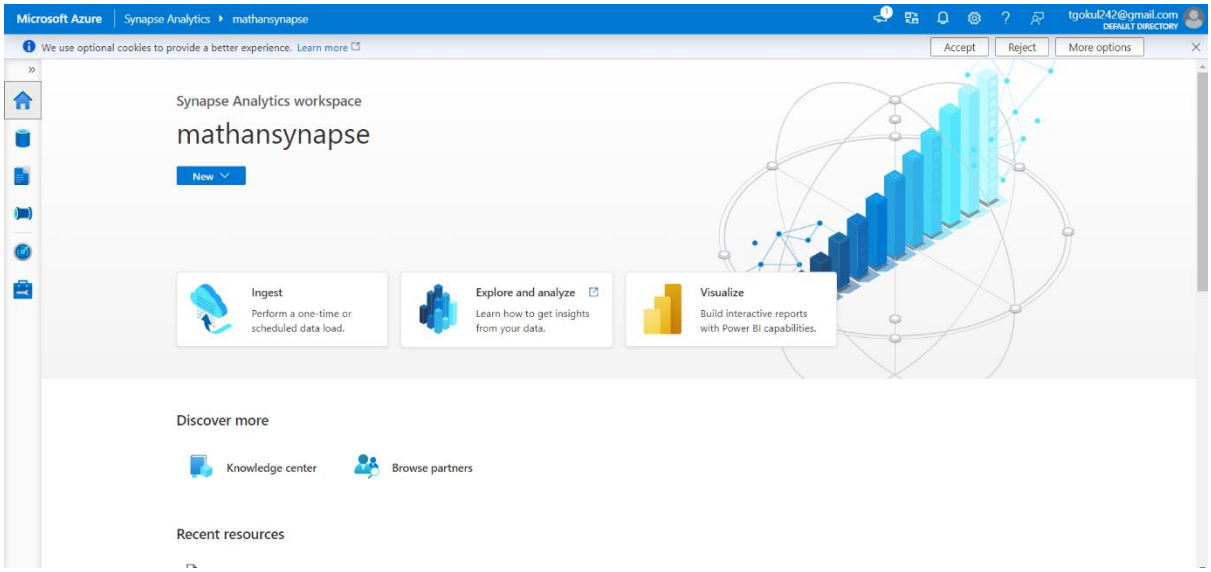
# Created Container




# Created Synapse

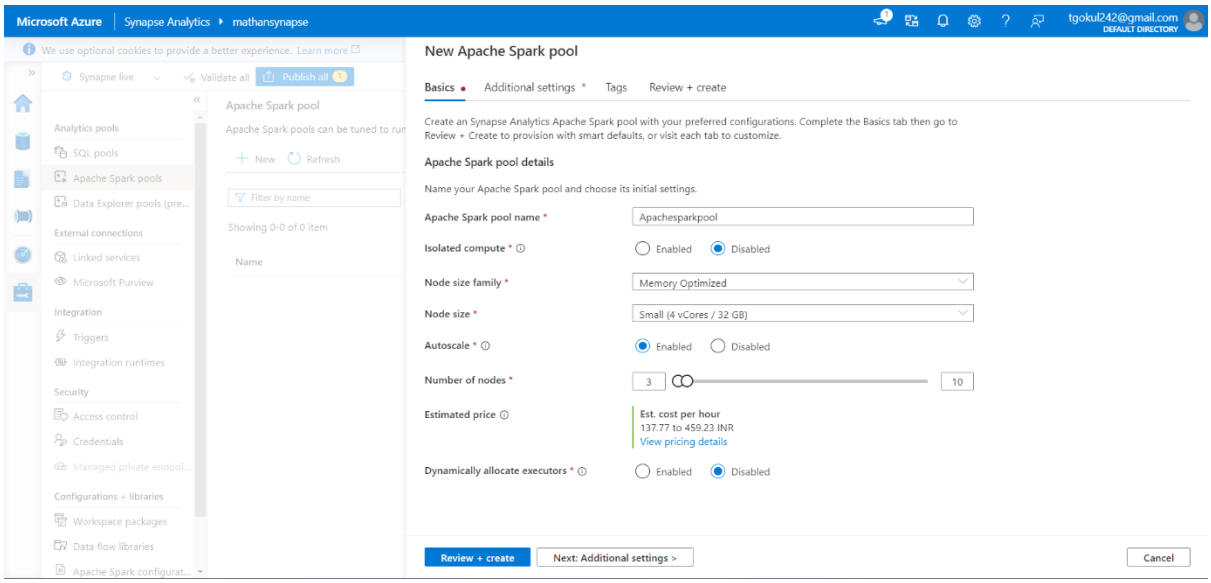


`




Created a Synapse workspace to leverage the power of PySpark for data manipulation and analysis, enabling seamless processing of web server logs. This centralized environment offers advanced analytics capabilities, empowering efficient exploration and visualization of insights derived from the data.

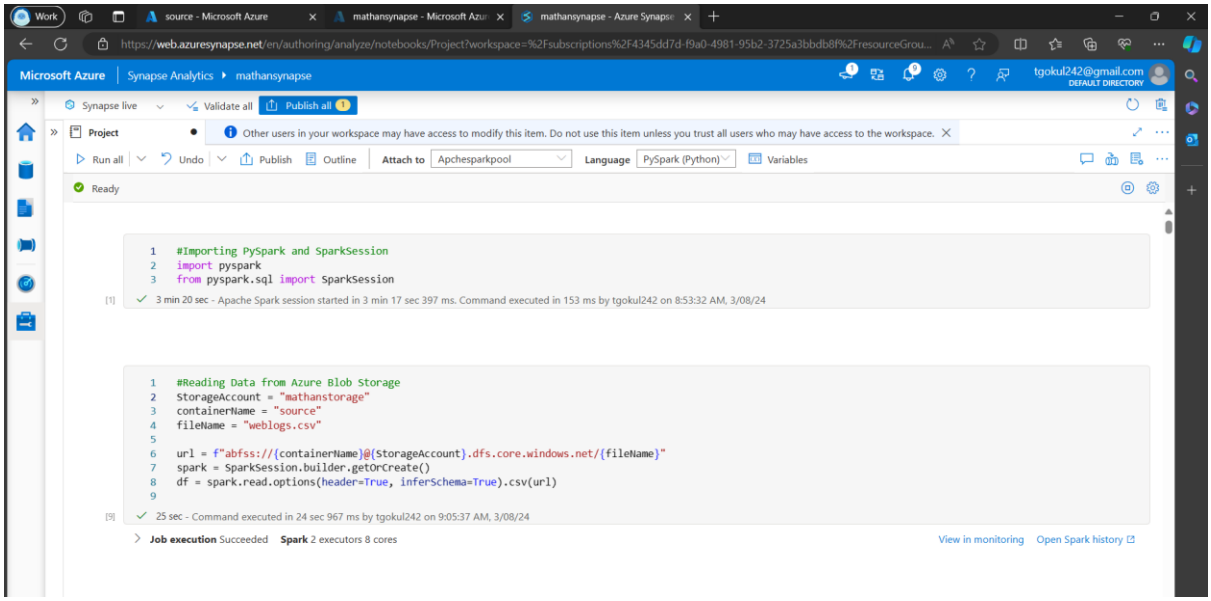
## Created Apache Spark Pool




Configured an Apache Spark pool within the Synapse workspace to seamlessly execute PySpark scripts, facilitating efficient data processing on large datasets. This setup ensures optimal resource utilization and enables swift analysis, empowering insightful decision-making from web server logs.

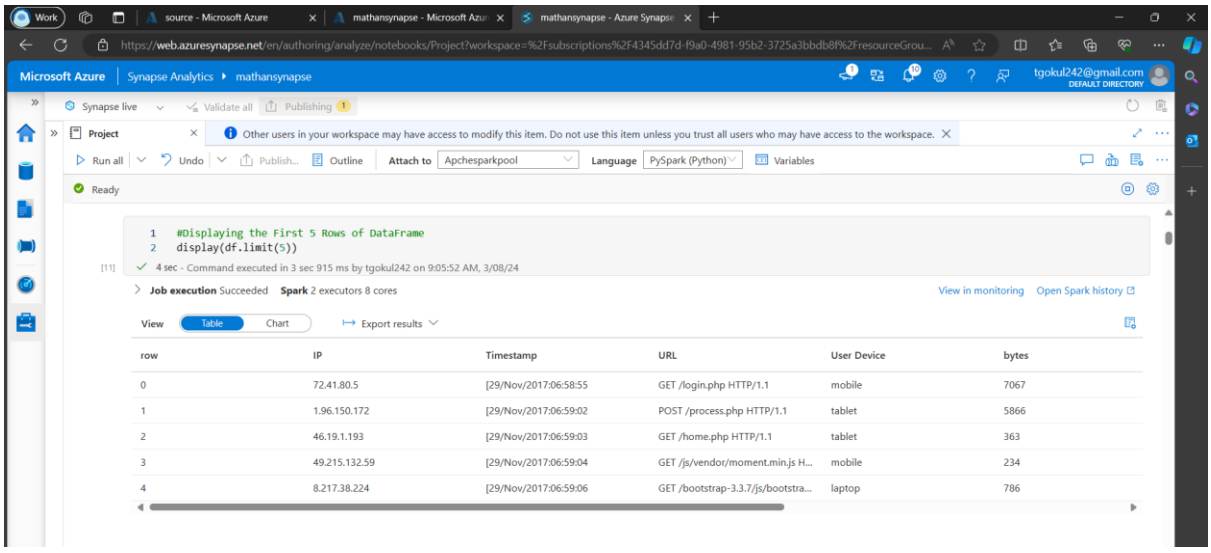
`

## Data Ingestion and Processing with PySpark and Azure Storage




The provided code initializes a SparkSession for PySpark operations and reads data from Azure Blob Storage. It specifies the Azure Storage Account, container, and file name, constructing the URL accordingly. The data is loaded into a DataFrame named df with inferred schema and header options enabled.

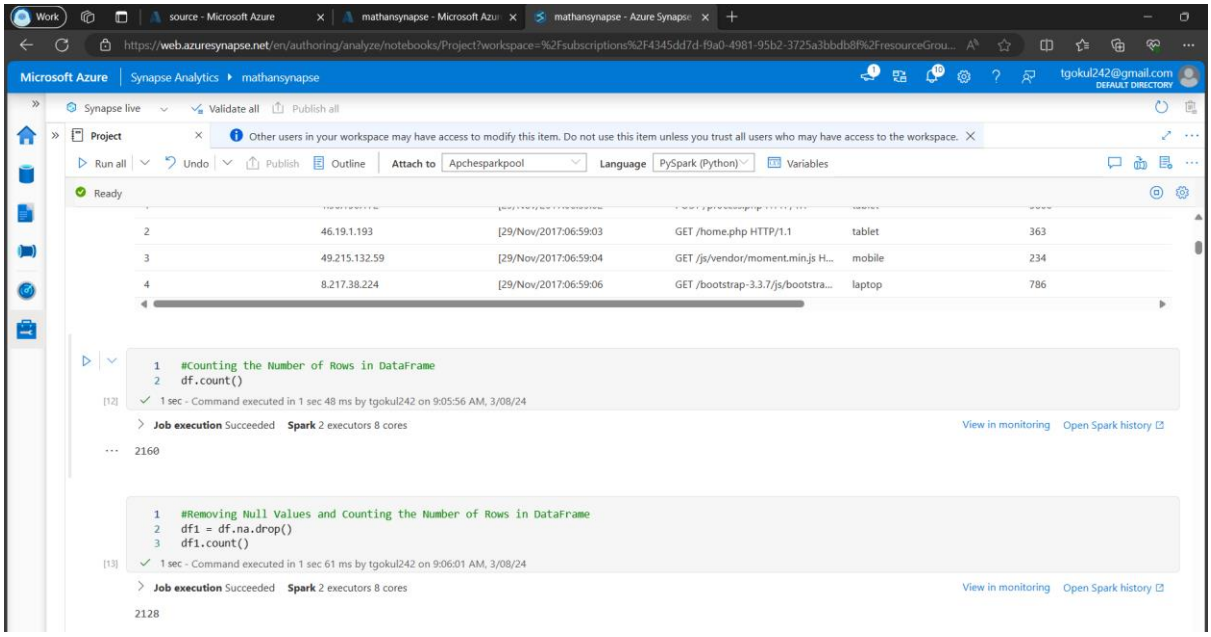
## Displaying the First 5 Rows of DataFrame




The code snippet displays the first five rows of the DataFrame named df using the display function, providing a quick preview of the dataset's structure and content. This allows for initial data exploration and validation before proceeding with further analysis.

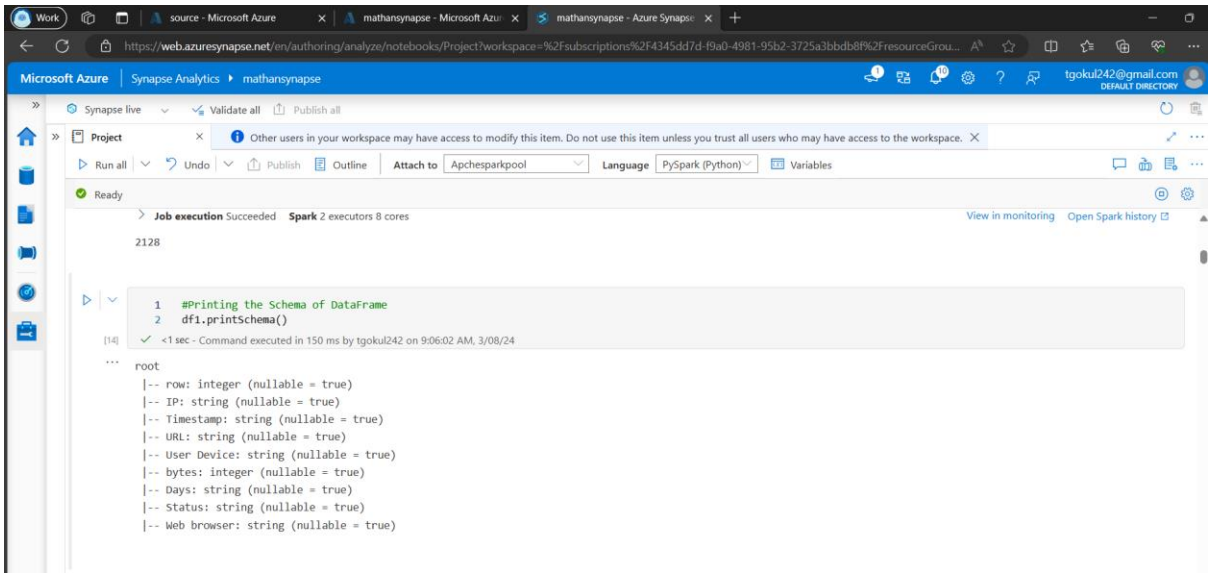
`

## Data Validation and Cleansing with Apache Spark




The above code snippet first counts the number of rows in the DataFrame df, providing a quick overview of the dataset size. It then removes any null values from the DataFrame 'df' and counts the number of rows again in the cleaned DataFrame df1, ensuring data integrity and preparing for further analysis.

## Schema Overview for DataFrame

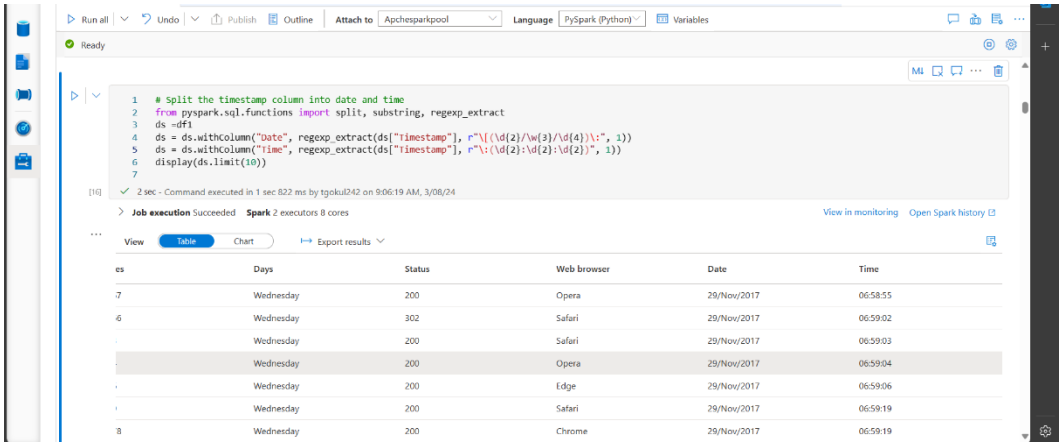


`

## Identifying Duplicate IP Addresses in DataFrame


The code snippet calculates the number of duplicate IP addresses in the DataFrame df1. It first counts the total number of rows in the DataFrame, then determines the number of distinct IP addresses. By subtracting the count of distinct IP addresses from the total number of rows, it identifies and prints the number of duplicate IP addresses in the DataFrame.

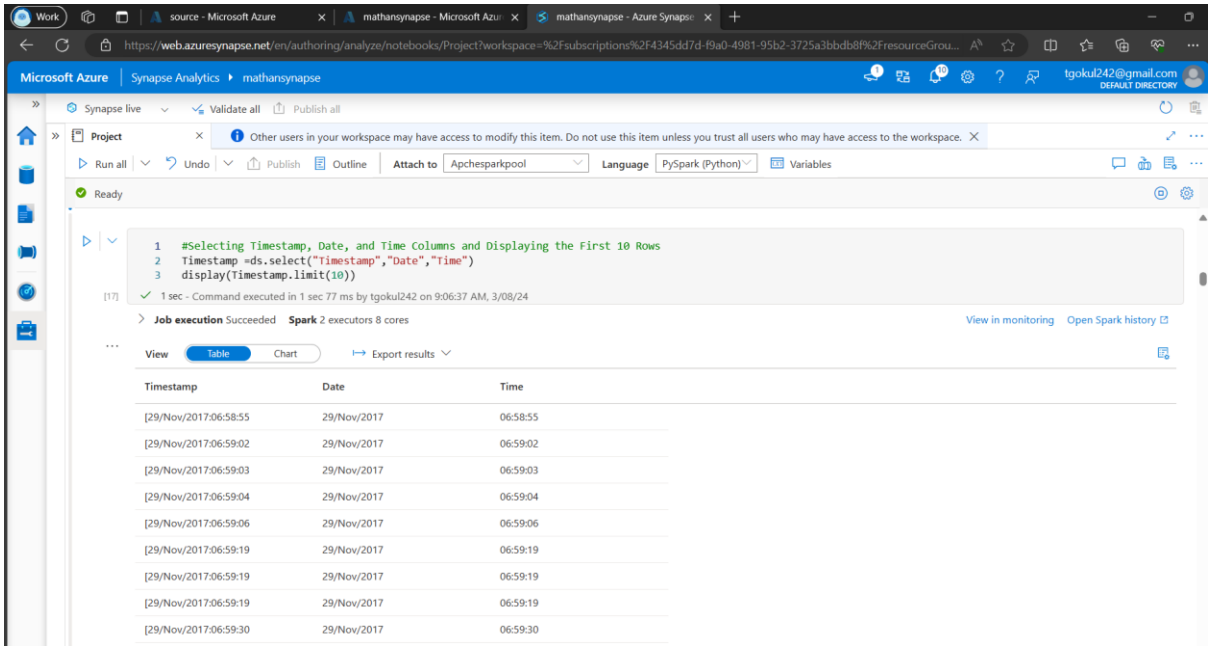
## Splitting the Timestamp into date and time



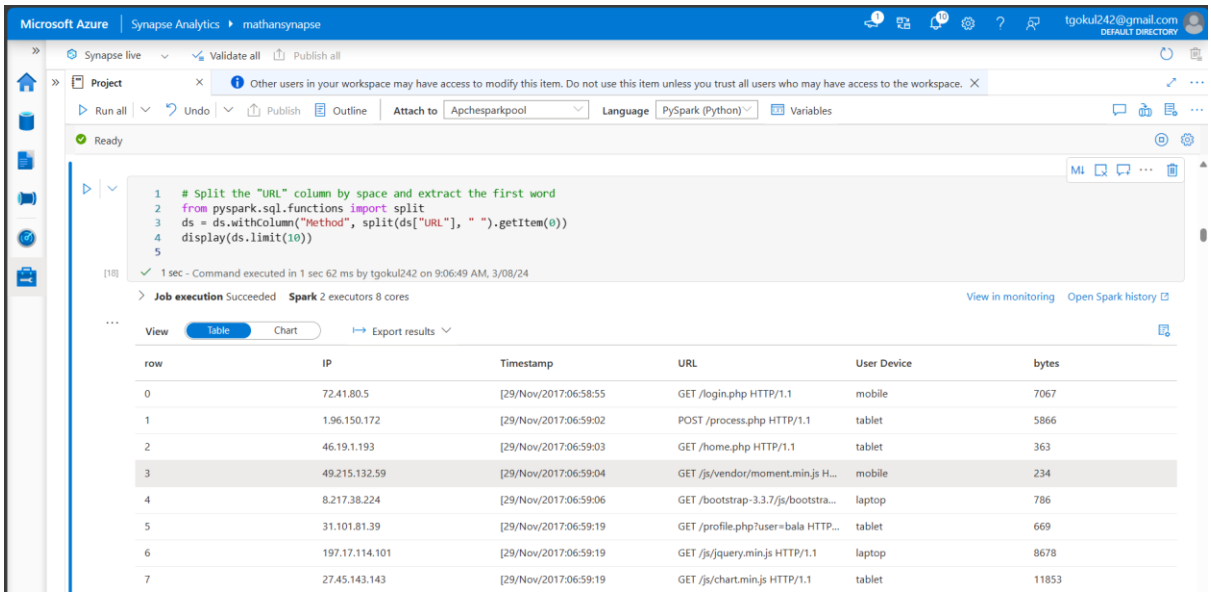

The provided code splits the timestamp column of the DataFrame 'df1' into separate date and time columns. It utilizes PySpark's 'regexp_extract' function to extract the date and time components from the timestamp using regular expressions. The resulting DataFrame 'ds' contains additional 'Date' and 'Time' columns, showcasing the date and time information extracted from the original timestamp. Finally, the first 10 rows of the transformed DataFrame are displayed, offering a glimpse into the newly structured data.

`

## Displaying the Timestamp



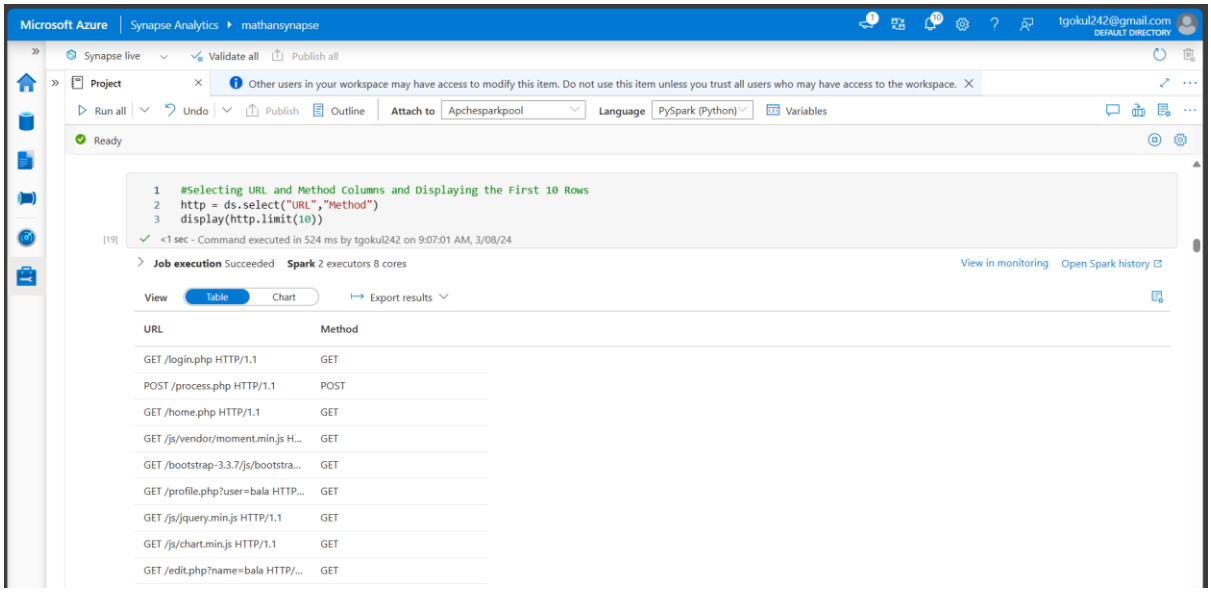

## Extracting HTTP Method from URL Column




This code splits the "URL" column by space and extracts the first word, which typically represents the HTTP method used in the request (e.g., GET, POST). By utilizing PySpark's split function, it isolates the method component from the URL string.

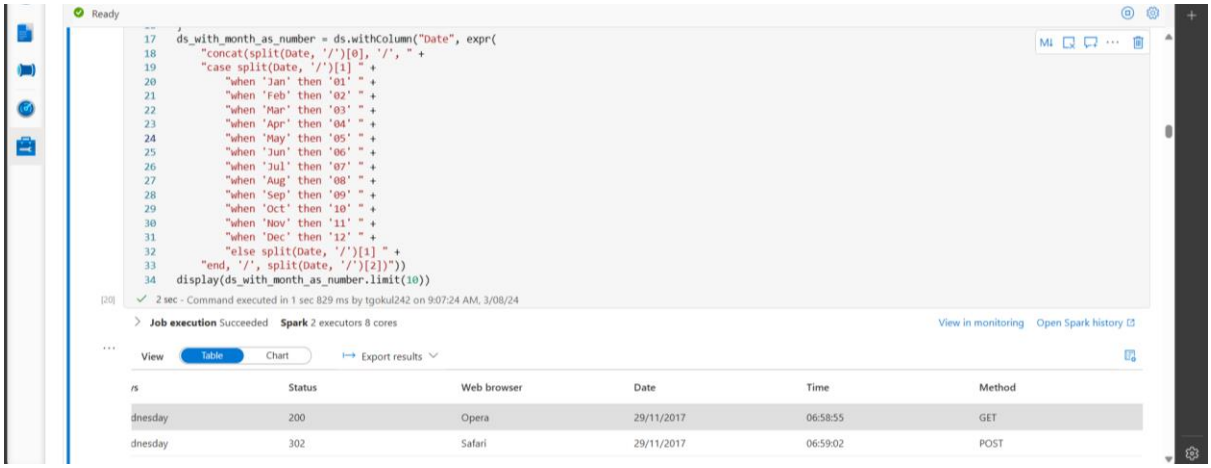
`

## Displaying URL and Method Data




## Converting Month Abbreviations to Numeric Values in Date Column


S

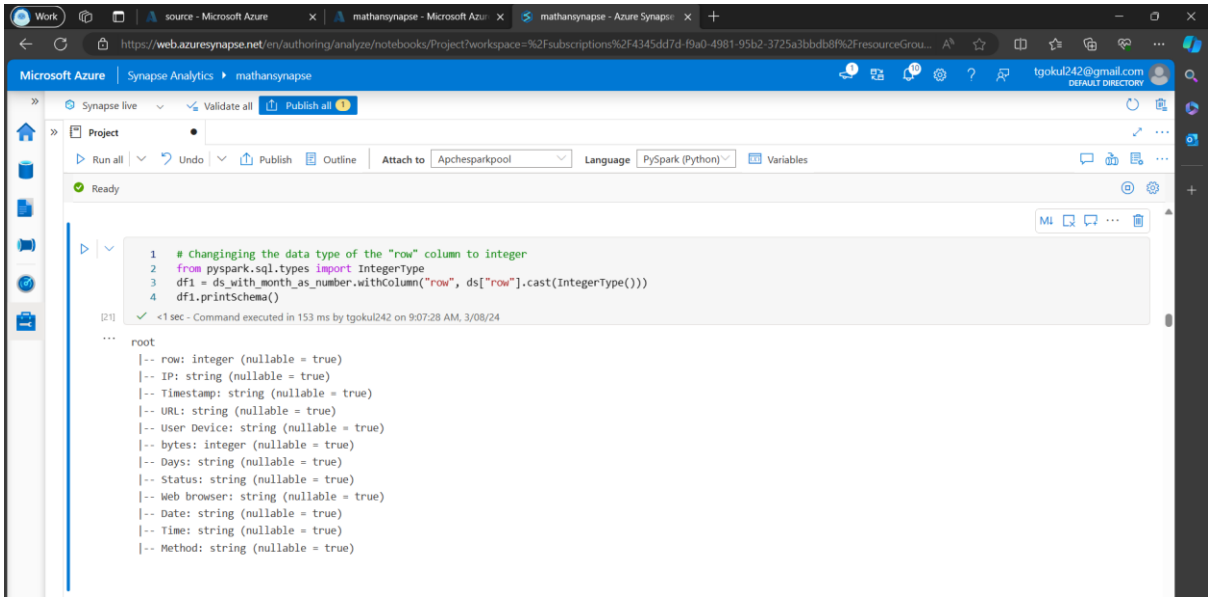



This code snippet converts the month abbreviations in the 'Date' column to their corresponding numeric values. It utilizes PySpark's 'expr' function to apply a custom transformation, mapping each abbreviation to its numeric equivalent using a predefined

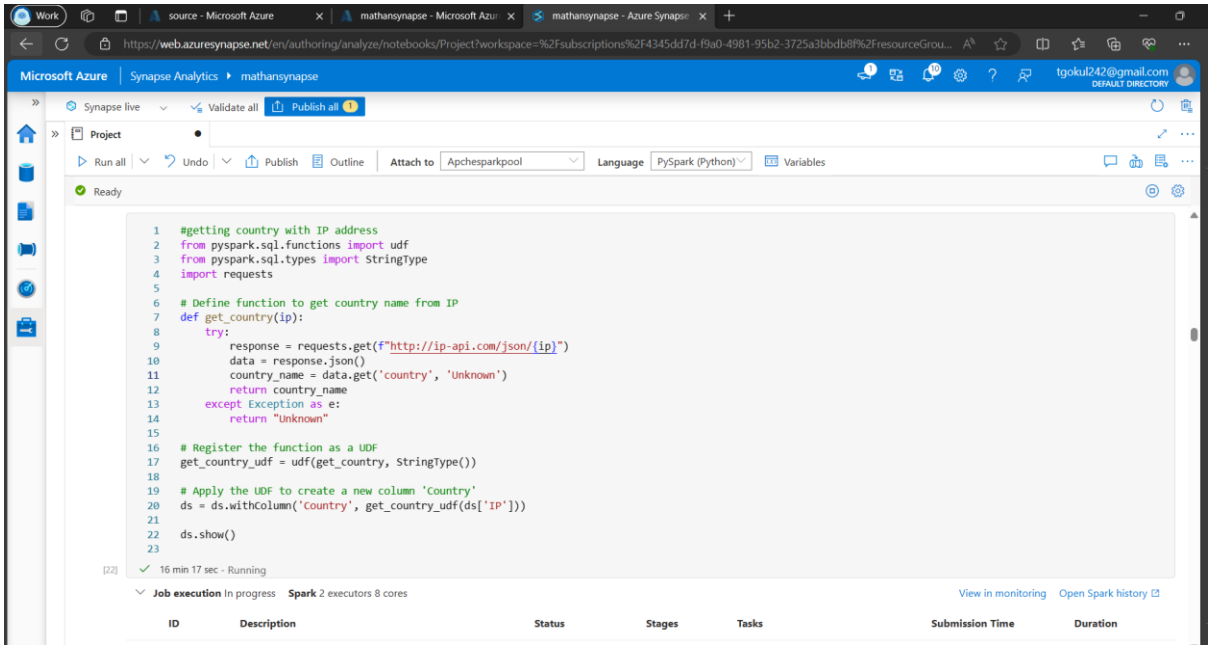
`

dictionary. The resulting DataFrame 'ds_with_month_as_number' features the 'Date' column with month values represented numerically, enhancing the dataset's clarity and suitability for analysis.

## Converting 'row' Column to Integer Type




## Extracting Country from IP Addresses

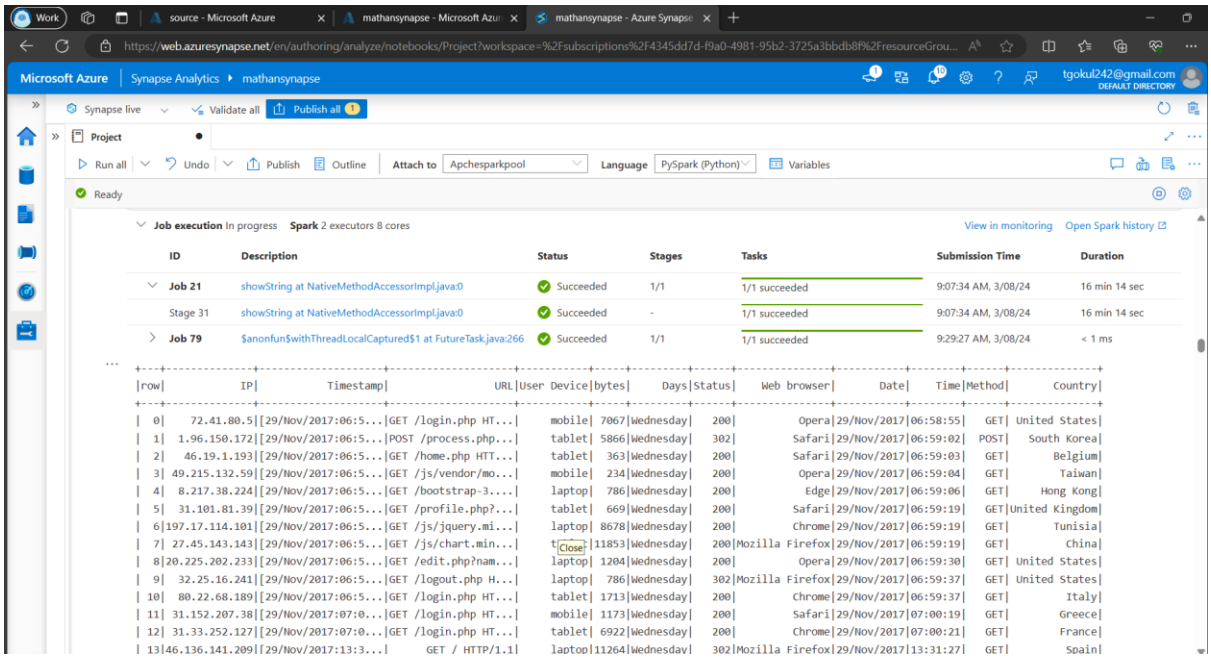



This code snippet defines a function to retrieve the country name associated with an IP address using an external API. It then registers the function as a User Defined Function (UDF)

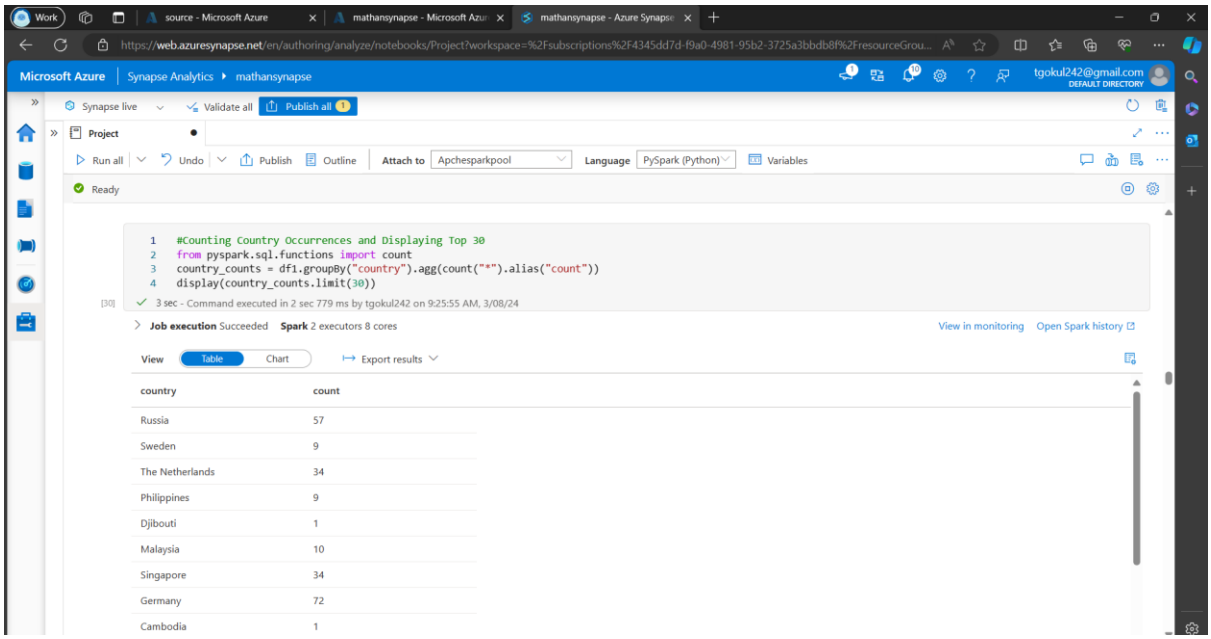
`

and applies it to create a new column 'Country' in the DataFrame 'ds'. This process enriches the dataset with country information derived from IP addresses, facilitating geographical analysis and insights.

## Outcome of Executed Job




## Analyzing Country Occurrences in Data

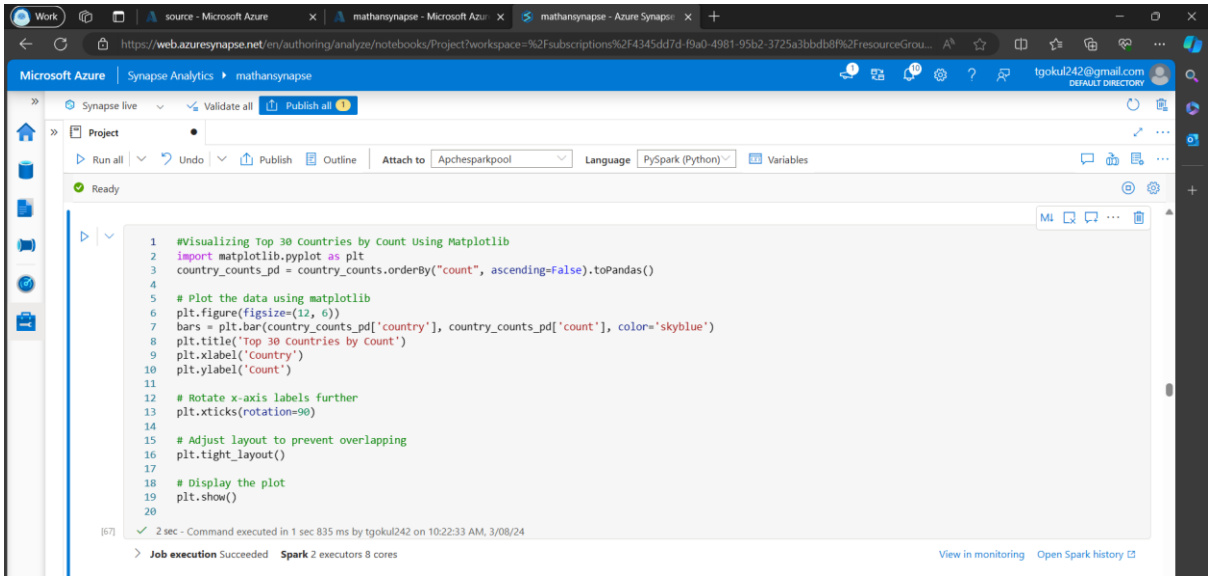



This code snippet counts the occurrences of each country in the DataFrame 'df1' and displays the top 30 countries by frequency. Utilizing PySpark's 'groupBy' and 'count' functions,

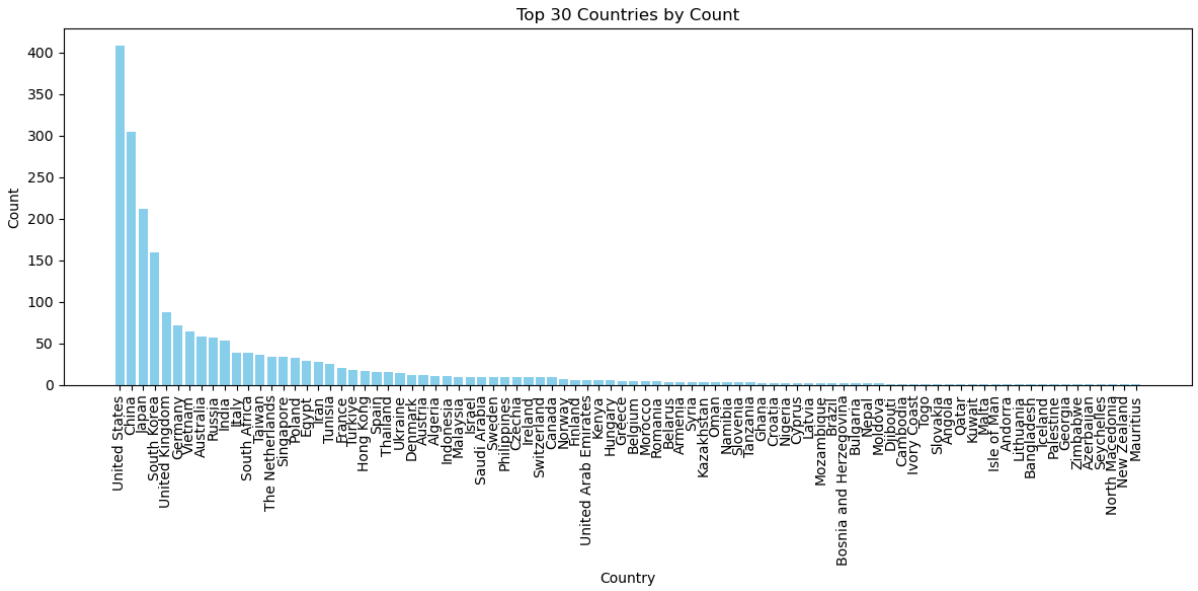
`

it aggregates country occurrences and creates a new DataFrame 'country_counts'. Displaying the top 30 countries provides insights into the distribution of website visitors' geographical locations, aiding in understanding user demographics and behavior.

## Visualizing Top 30 Countries by Count

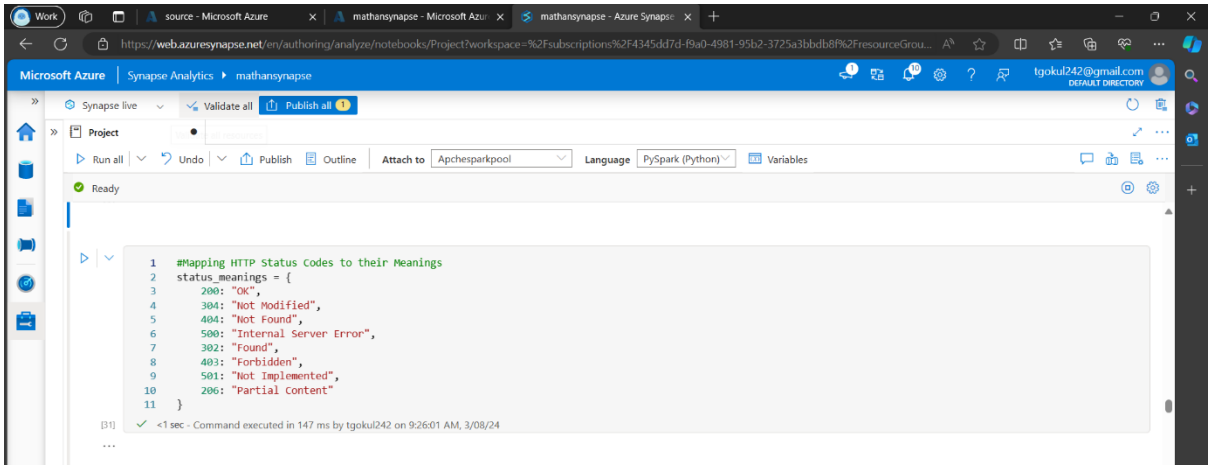



## Visualization

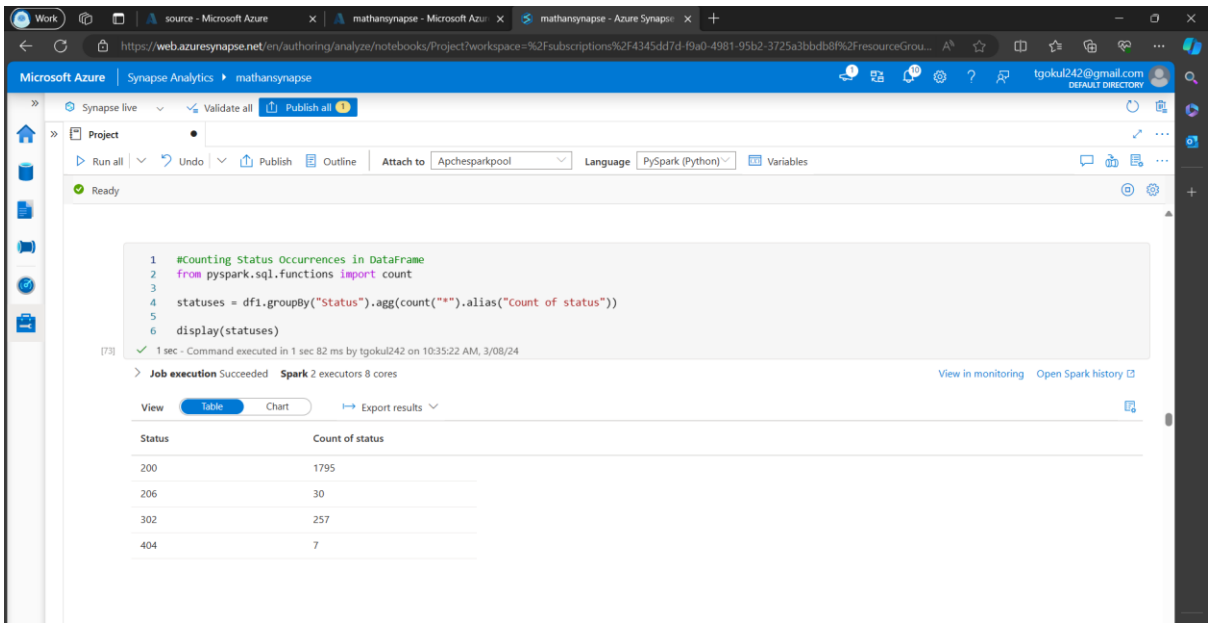


`

## HTTP Status Code Meanings



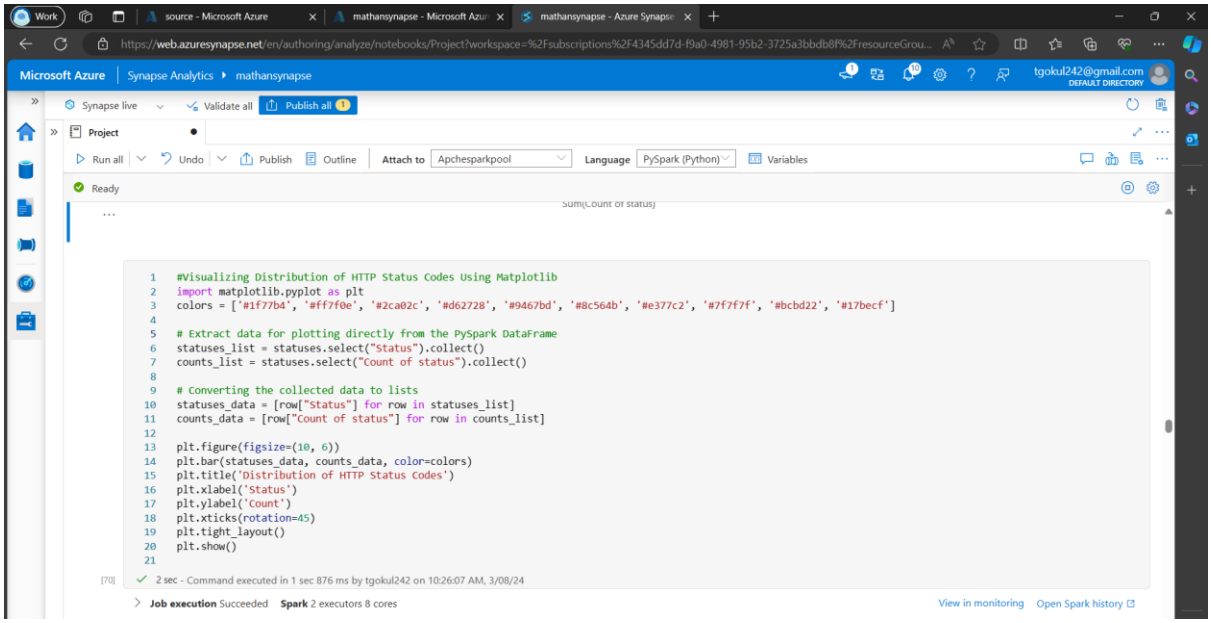

## Analyzing Status Occurrences in Data



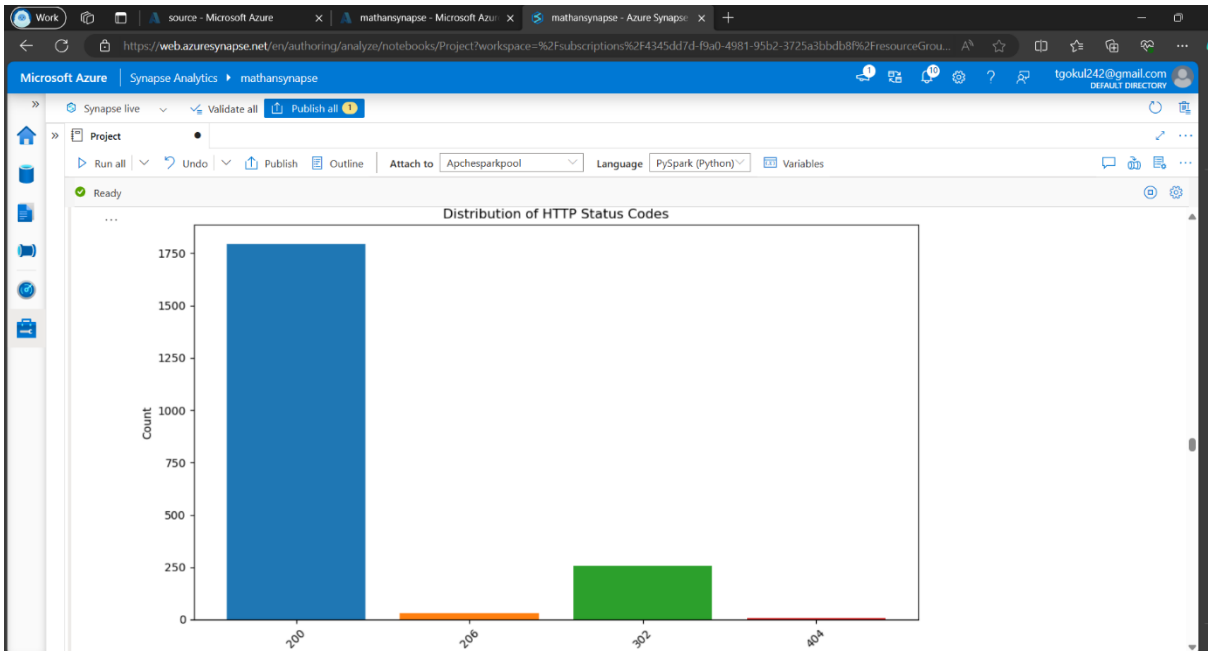

This code snippet counts the occurrences of each status code in the DataFrame 'df1'. By utilizing PySpark's 'groupBy' and 'count' functions, it aggregates status occurrences and presents them in a new DataFrame 'statuses'. Displaying these counts offers insights into the frequency of different HTTP status codes, aiding in understanding the server's responses and identifying potential issues.

`

# Visualizing HTTP Status Code Distribution

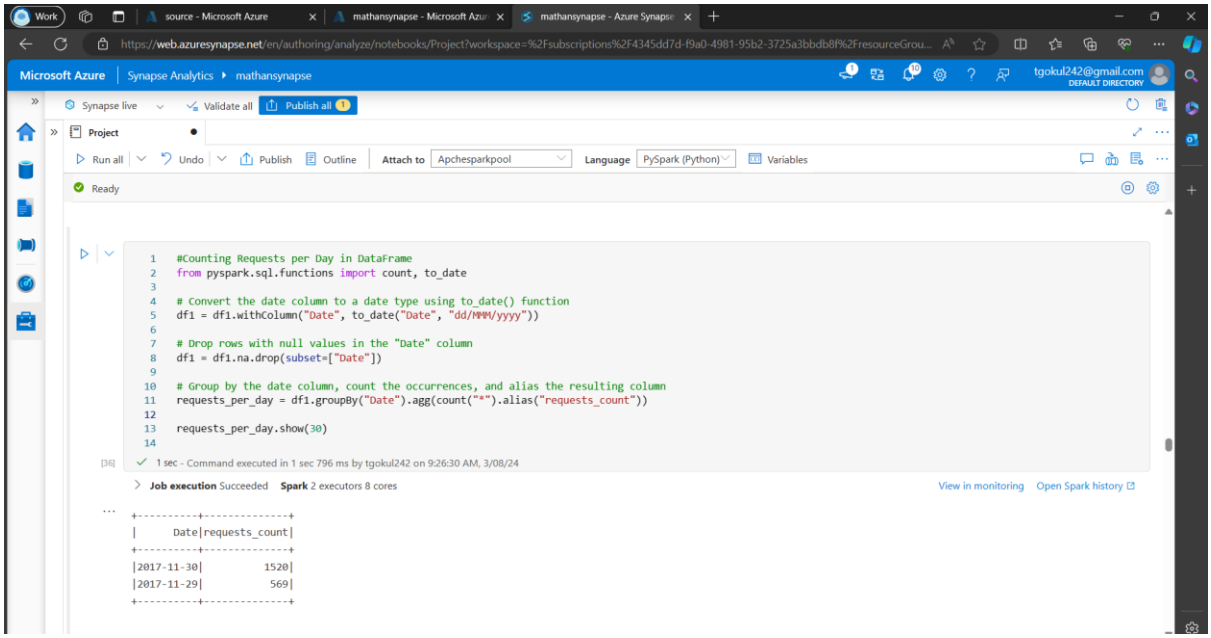



# Visualization



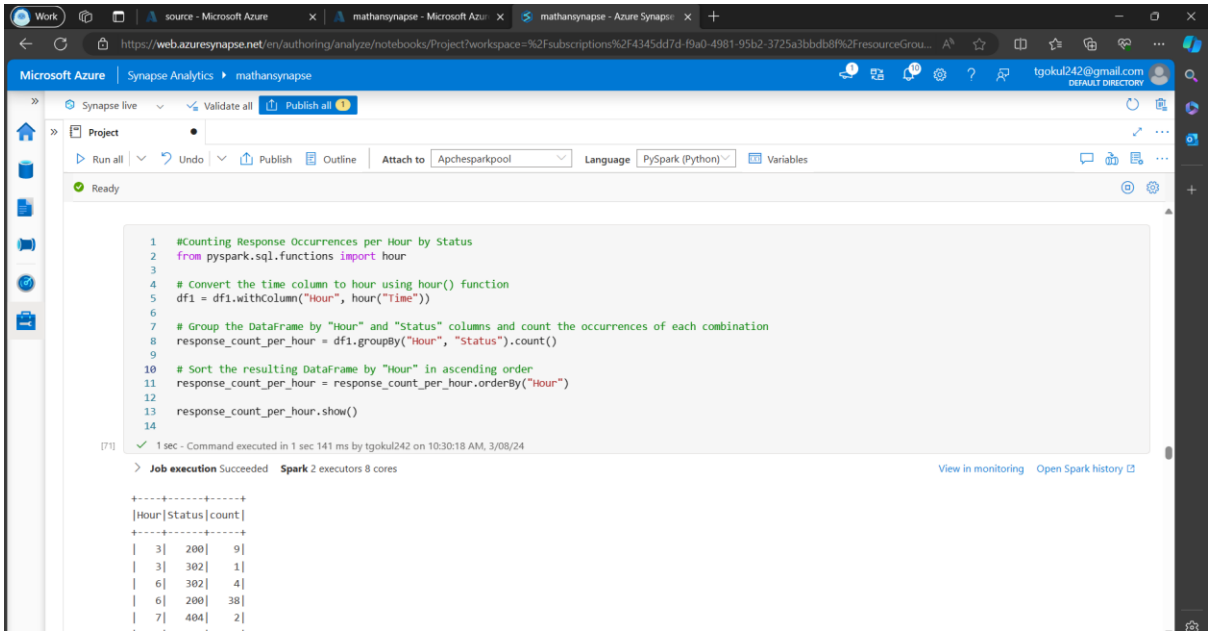
`

## Analyzing Daily Request Counts




This code snippet calculates the number of requests per day in the DataFrame 'df1'. It first converts the 'Date' column to a date type using the 'to_date' function and then drops any rows with null values in the 'Date' column. After grouping the data by date and counting the occurrences, it aliases the resulting column as 'requests_count'. The output displays the daily request counts for further analysis, aiding in understanding the traffic patterns over time.

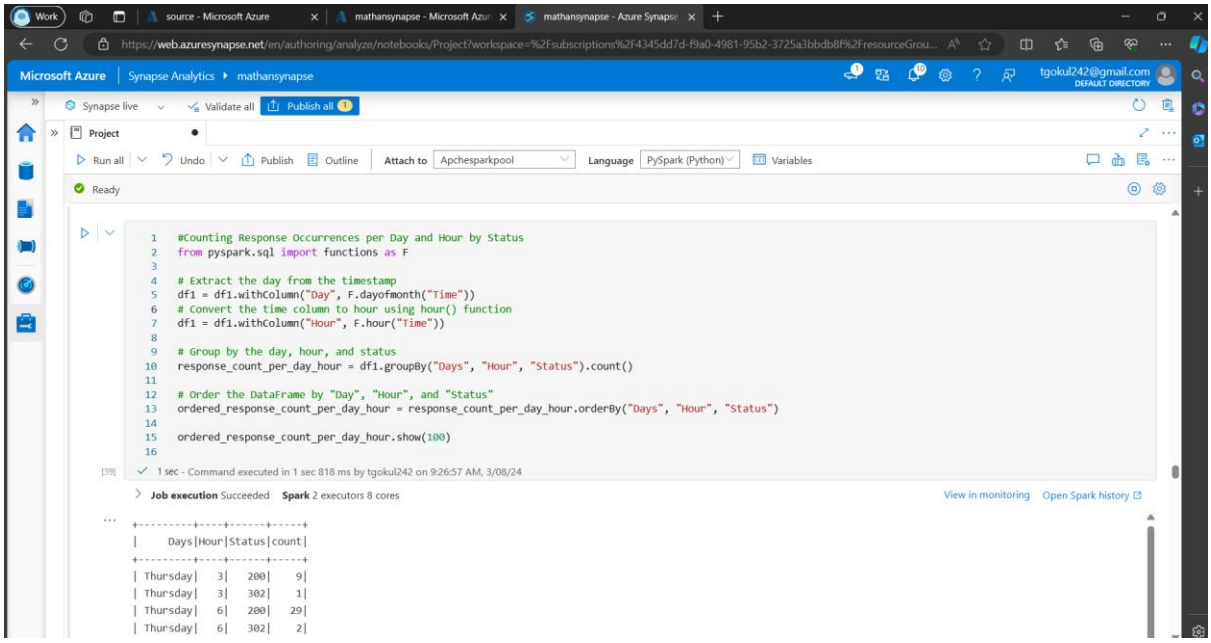
## Analyzing Hourly Response Occurrences by Status

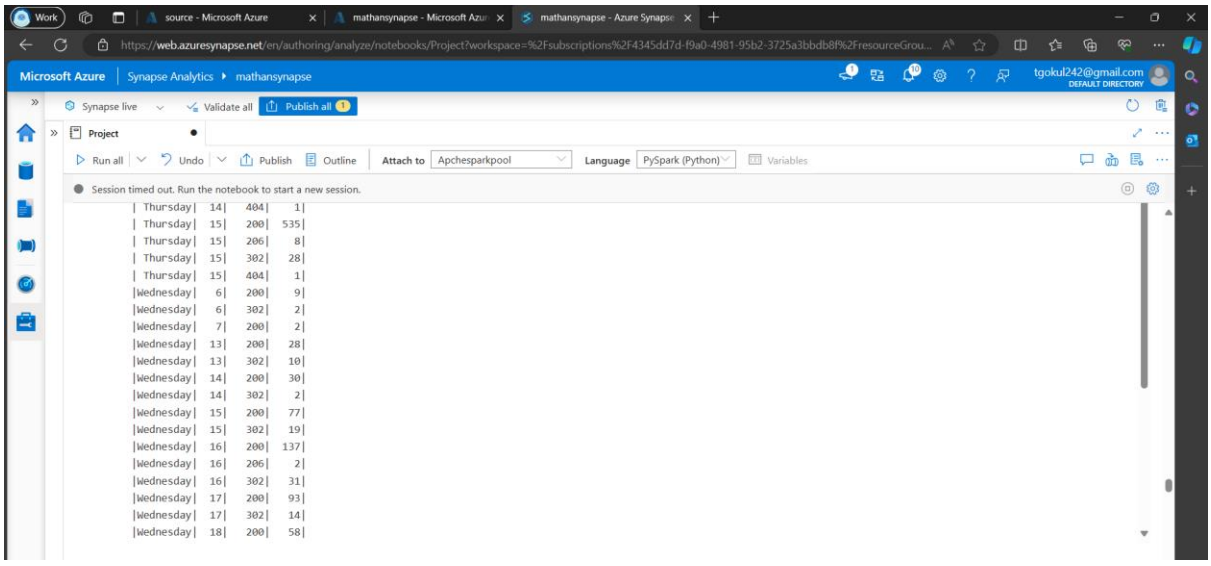


`

This code calculates the occurrences of responses per hour, categorized by status, in the DataFrame 'df1'. It first extracts the hour component from the 'Time' column using the 'hour' function and creates a new column named 'Hour'. Then, it groups the DataFrame by both 'Hour' and 'Status', counting the occurrences of each combination. The resulting DataFrame is sorted by 'Hour' in ascending order to facilitate analysis. Displaying the data provides insights into the distribution of response statuses across different hours of the day.

## Analyzing Response Occurrences per Day and Hour by Status



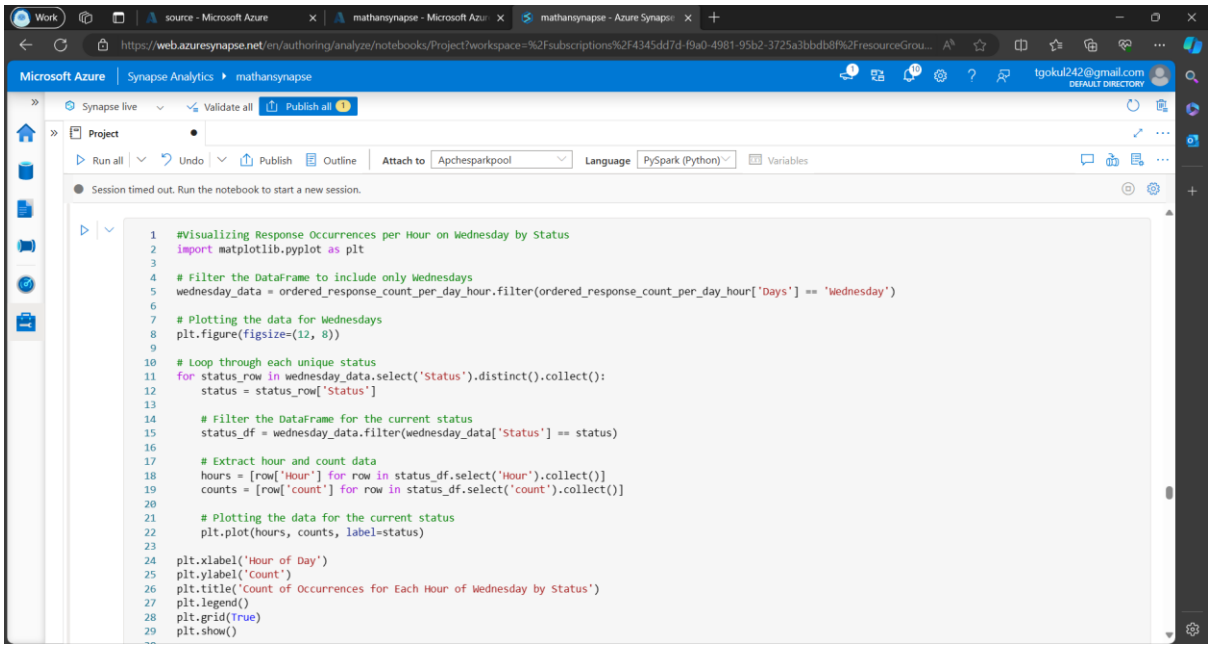




This code computes the occurrences of responses per day and hour, segmented by status, in the DataFrame 'df1'. It begins by extracting the day and hour components from the 'Time' column and creating new columns named 'Day' and 'Hour', respectively. Next, it groups the DataFrame by 'Day', 'Hour', and 'Status', calculating the count of occurrences for each

`

combination. The resulting DataFrame is ordered by 'Day', 'Hour', and 'Status', facilitating further analysis. Displaying the data offers insights into the distribution of response statuses over different days and hours.

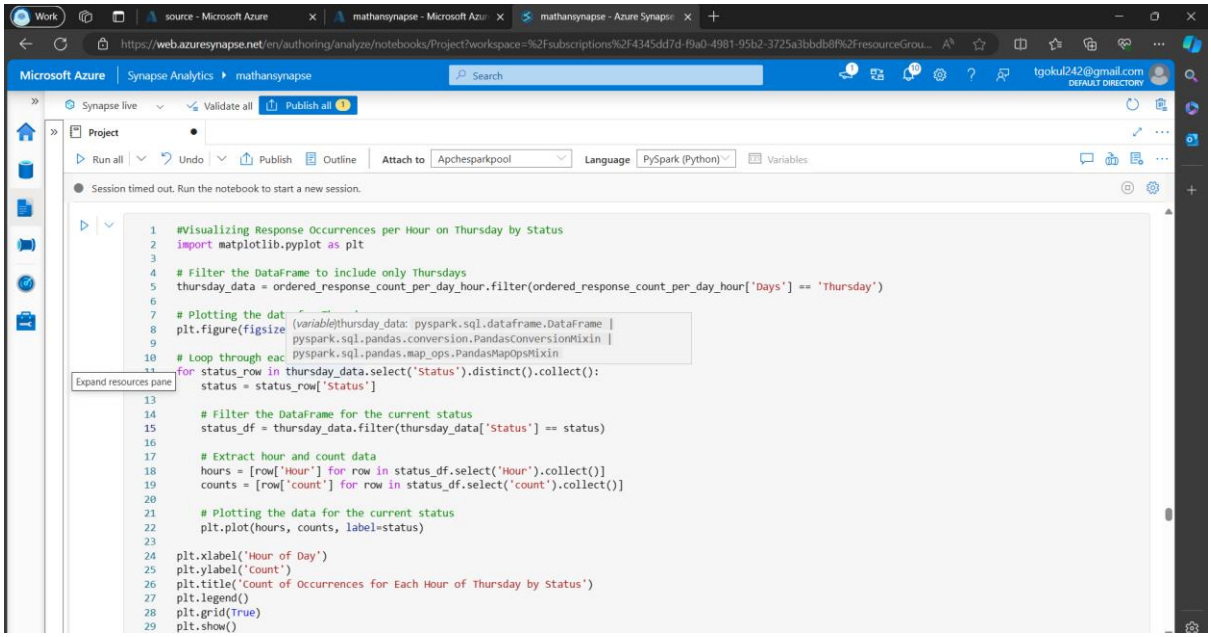
## Visualizing Hourly Response Occurrences on Wednesdays by Status



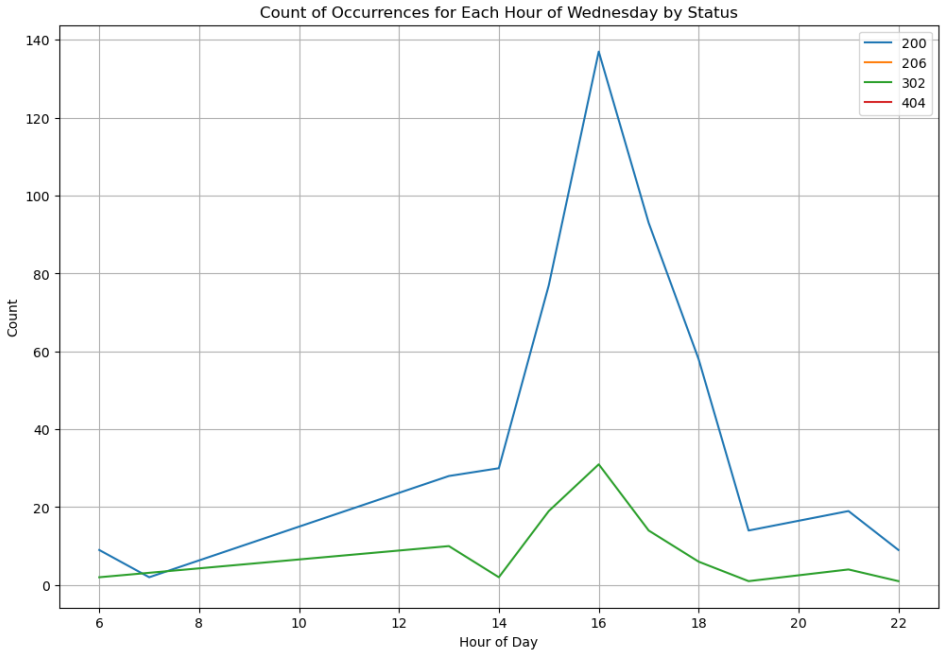

This code snippet filters the DataFrame to include only data for Wednesdays and then plots the count of response occurrences for each hour, categorized by status. It iterates through each unique status, filters the DataFrame accordingly, and extracts hour and count data. Finally, it plots the data for each status, displaying the count of occurrences for each hour of Wednesday. The resulting visualization offers insights into the distribution of response statuses over different hours of Wednesdays.

`

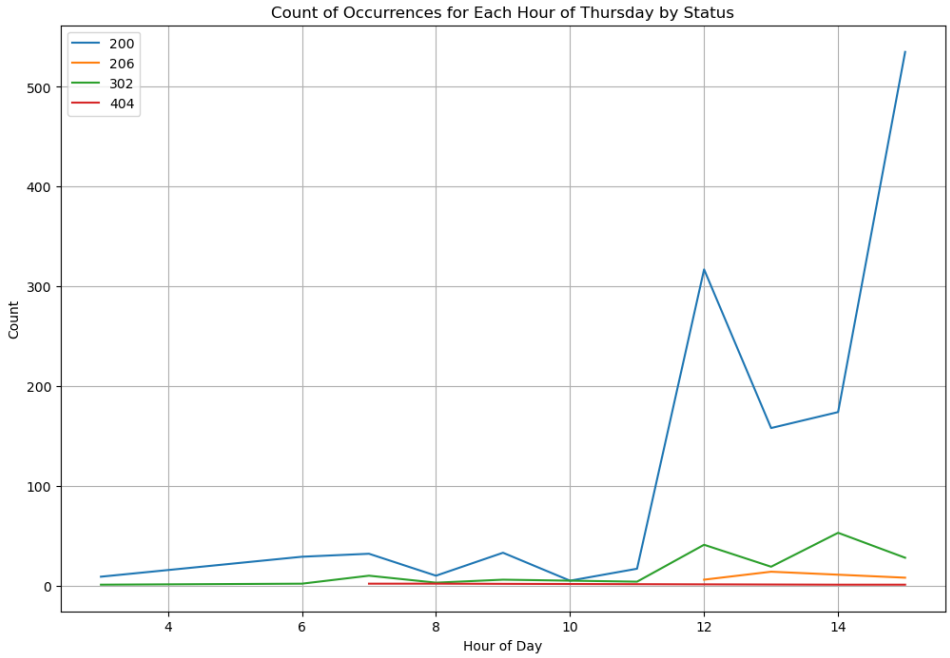
# Visualizing Hourly Response Occurrences on Thursday by Status



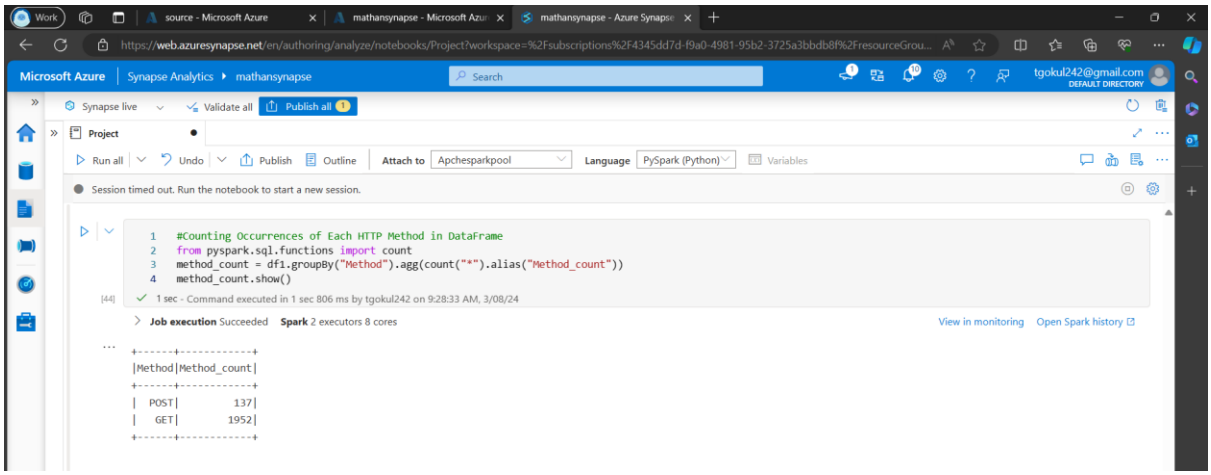

# Visualization



`




## Analysing HTTP Method Occurrences




Calculates the occurrences of each HTTP method in the DataFrame 'df1'. It uses PySpark's 'groupBy' function to group the data by the HTTP method and then applies the 'count' function to count the occurrences. The resulting DataFrame 'method_count' displays the count of occurrences for each HTTP method. This analysis provides insights into the distribution of HTTP methods used in the dataset, aiding in understanding the nature of the requests.

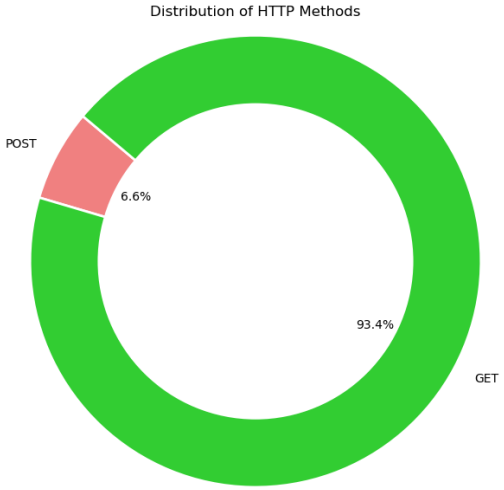
`

## Visualizing HTTP Method Distribution

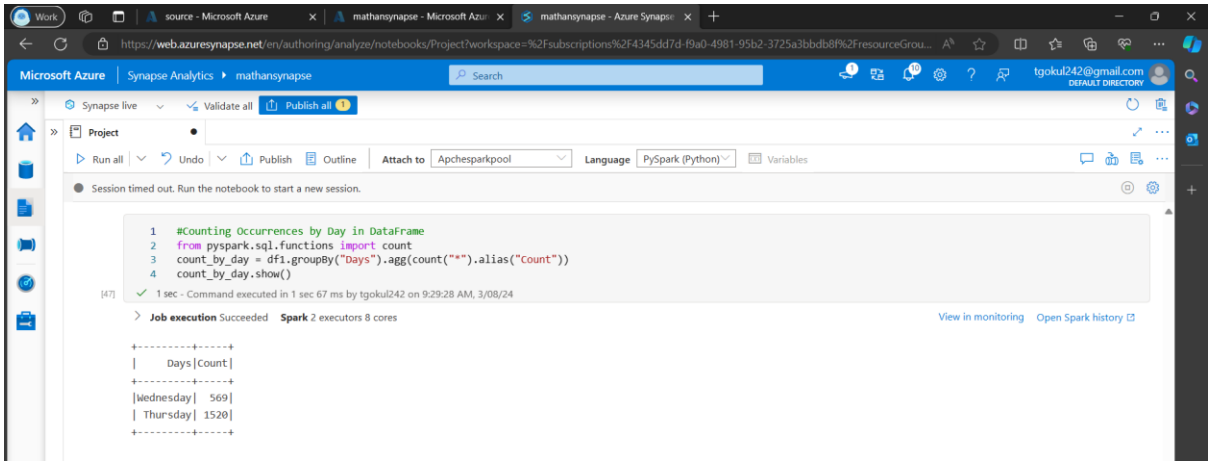
```
#Visualizing Distribution of HTTP Methods Using Doughnut Chart
method_count_df = method_count.toPandas()

colors = ['lightcoral', 'limegreen']

# Plotting the doughnut chart with custom colors and outline
plt.figure(figsize=(8, 6))
plt.pie(method_count_df["Method_count"], labels=method_count_df["Method"], autopct='%1.1f%%', startangle=140, colors=colors, wedgeprops={'linewidth': 2, 'edgecol
plt.axis('equal')

# Draw a white circle at the center to create the doughnut effect
centre_circle = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Distribution of HTTP Methods')
plt.tight_layout()
plt.show()
```

1 sec - Command executed in 1 sec 95 ms by tgokul242 on 9:29:00 AM, 3/08/24

Job execution Succeeded Spark 2 executors 8 cores View in monitoring Open Spark history




## Analyzing Daily Occurrences



```
#Counting Occurrences by Day in DataFrame
from pyspark.sql.functions import count
count_by_day = df1.groupBy("Days").agg(count("*").alias("Count"))
count_by_day.show()
```

1 sec - Command executed in 1 sec 67 ms by tgokul242 on 9:29:28 AM, 3/08/24

Job execution Succeeded Spark 2 executors 8 cores View in monitoring Open Spark history

```
+---------+-----+
|     Days|Count|
+---------+-----+
|Wednesday|  569|
| Thursday| 1520|
+---------+-----+
```

The above code calculates the occurrences of events by day in the DataFrame 'df1'. It utilizes PySpark groupBy function to group the data by the 'Days' column and then applies the 'count' function to count the number of occurrences. The resulting DataFrame 'count_by_day'

`

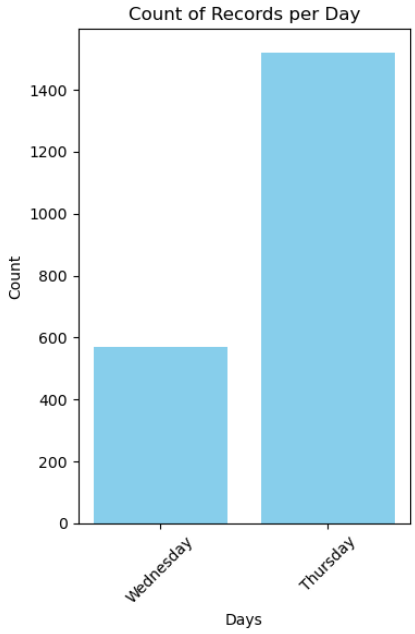
displays the count of events for each day. Analyzing daily occurrences provides insights into patterns or trends that may vary depending on the day of the week.

## Visualizing Daily Record Counts with Bar Graph

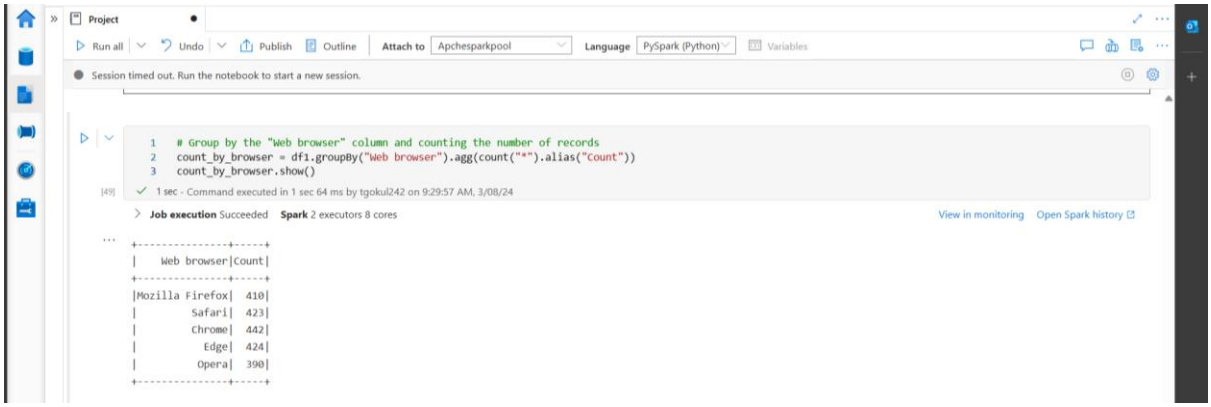
```
#Visualizing Count of Records per Day Using Bar Graph
import matplotlib.pyplot as plt
import pandas as pd

count_by_day_df = count_by_day.toPandas()
# Plotting the bar graph
plt.figure(figsize=(4, 6))
plt.bar(count_by_day_df["Days"], count_by_day_df["Count"], color='skyblue')
plt.xlabel('Days')
plt.ylabel('Count')
plt.title('Count of Records per Day')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

1 sec - Command executed in 1 sec 71 ms by tgokul242 on 9:29:41 AM, 3/08/24

Job execution Succeeded Spark 2 executors 8 cores View in monitoring Open Spark history




## Analyzing Web Browser Usage



Project

Run all Undo Publish Outline Attach to Apchesparkpool Language PySpark (Python) Variables

Session timed out. Run the notebook to start a new session.

```
# Group by the "Web browser" column and counting the number of records
count_by_browser = df1.groupBy("Web browser").agg(count("*").alias("Count"))
count_by_browser.show()
```

1 sec - Command executed in 1 sec 64 ms by tgokul242 on 9:29:57 AM, 3/08/24

Job execution Succeeded Spark 2 executors 8 cores View in monitoring Open Spark history

```
+---------------+-----+
|    Web browser|Count|
+---------------+-----+
|Mozilla Firefox|  410|
|         Safari|  423|
|         Chrome|  442|
|           Edge|  424|
|          Opera|  390|
+---------------+-----+
```

`

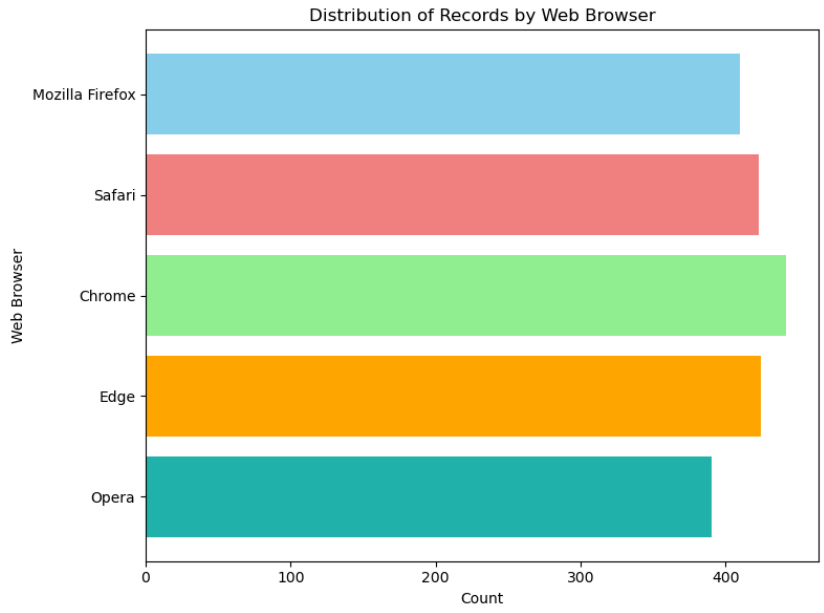
This code snippet groups the DataFrame 'df1' by the "Web browser" column and calculates the count of records for each browser. It utilizes PySpark's 'groupBy' function to group the data and then applies the 'count' function to determine the number of records per browser. The resulting DataFrame 'count_by_browser' displays the count of records for each web browser, providing insights into browser usage patterns.

## Visualizing Web Browser Records

```
#Visualizing Distribution of Records by Web Browser Using Horizontal Bar Plot
import matplotlib.pyplot as plt
count_by_browser_df = count_by_browser.toPandas()
colors = ['skyblue', 'lightcoral', 'lightgreen', 'orange', 'lightseagreen']

# Plotting the horizontal bar plot with custom colors
plt.figure(figsize=(8, 6))
plt.barh(count_by_browser_df["Web browser"], count_by_browser_df["Count"], color=colors)
plt.xlabel('Count')
plt.ylabel('Web Browser')
plt.title('Distribution of Records by Web Browser')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest count at the top
plt.tight_layout()
plt.show()
```

[51] ✓ 1 sec - Command executed in 1 sec 81 ms by tgokul242 on 9:31:39 AM, 3/08/24

Job execution Succeeded Spark 2 executors 8 cores View in monitoring Open Spark history



`

## Conclusion:

This project involved analyzing web server logs to understand website traffic patterns and optimize server performance using Spark DataFrames in Azure Synapse. Beginning with the establishment of a resource group for organizing source files and housing the Synapse workspace, the process seamlessly integrated analysis and visualization techniques. Key steps included data collection from Kaggle, ingestion into Azure Synapse, and analysis using Spark DataFrames. Insights were generated regarding website traffic patterns, peak traffic times, user device analysis, error analysis, and more. Throughout the project, various transformations, functions, and visualizations were employed to gain valuable insights and drive informed decision-making.