

## **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**

S.No	Topics
1	Project Objective
2	Introduction
3	Overview of Dataset
4	Table Schema
5	Resource Group
6	Azure Synapse
7	Data Ingestion and Processing with Pyspark
8	Splitting the Timestamp into Date and Time
9	Extracting HTTP Method from URL Column
10	Visualizing HTTP Method Distribution
11	Extracting Country from IP Addresses
12	Outcome of Executed Job
13	Analyzing Country Occurrences in Data
14	Analyzing Status Occurrences in Data
15	Analyzing Daily Request Counts
16	Analyzing Hourly Response Occurrences by Status
17	Analyzing Web Browser Usage
18	Conclusion

## **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 Device	bytes	Days	Status	Web browser
2	0.72.41.80.5	[29/Nov/2017:06:58:50]	GET /login.php	HTTP mobile	7067	Wednesday	200	Opera
3	1.96.150.129	[29/Nov/2017:06:59:59]	POST /process.php	I tablet	5866	Wednesday	302	Safari
4	2.46.19.1.19	[29/Nov/2017:06:59:59]	GET /home.php	HTTP tablet	363	Wednesday	200	Safari
5	3.49.215.13	[29/Nov/2017:06:59:59]	GET /js/vendor/modernizr	mobile	234	Wednesday	200	Opera
6	4.8.217.38.2	[29/Nov/2017:06:59:59]	GET /bootstrap-3.3.7/laptop		786	Wednesday	200	Edge
7	5.31.101.81	[29/Nov/2017:06:59:59]	GET /profile.php?i=tablet		669	Wednesday	200	Safari
8	6.197.17.11	[29/Nov/2017:06:59:59]	GET /js/jquery.min.js	laptop	8678	Wednesday	200	Chrome
9	7.27.45.143	[29/Nov/2017:06:59:59]	GET /js/chart.min.js	tablet	11853	Wednesday	200	Mozilla Firefox
10	8.20.225.20	[29/Nov/2017:06:59:59]	GET /edit.php?name=laptop		1204	Wednesday	200	Opera
11	9.32.25.16.2	[29/Nov/2017:06:59:59]	GET /logout.php	HTTP laptop	786	Wednesday	302	Mozilla Firefox
12	10.80.22.68.1	[29/Nov/2017:06:59:59]	GET /login.php	HTTP tablet	1713	Wednesday	200	Chrome
13	11.31.152.20	[29/Nov/2017:07:00:00]	GET /login.php	HTTP mobile	1173	Wednesday	200	Safari
14	12.31.33.252	[29/Nov/2017:07:00:00]	GET /login.php	HTTP tablet	6922	Wednesday	200	Chrome
15	13.46.136.14	[29/Nov/2017:13:31:31]	GET /HTTP/1.1	laptop	11264	Wednesday	302	Mozilla Firefox
16	14.14.242.24	[29/Nov/2017:13:31:31]	GET /login.php	HTTP laptop	8083	Wednesday	200	Edge
17	15.32.101.8.1	[29/Nov/2017:13:38:00]	POST /process.php	I tablet	1204	Wednesday	302	Safari
18	16.72.232.22	[29/Nov/2017:13:38:00]	GET /home.php	HTTP mobile	1713	Wednesday	200	Opera
19	17.197.40.24	[29/Nov/2017:13:38:00]	GET /contentproblem/laptop		1173	Wednesday	200	Chrome
20	18.31.70.201	[29/Nov/2017:13:38:00]	GET /HTTP/1.1	tablet	4673	Wednesday	302	Chrome
21	19.32.106.24	[29/Nov/2017:13:38:00]	GET /login.php	HTTP laptop	46053	Wednesday	200	Chrome
22	20.14.163.19	[29/Nov/2017:13:38:00]	GET /css/bootstrap	laptop	786	Wednesday	200	Chrome
23	21.49.57.72.1	[29/Nov/2017:13:38:00]	GET /css/font-awesom	mobile	3714	Wednesday	200	Opera
24	22.80.148.19	[29/Nov/2017:13:38:00]	GET /css/normalize	tablet	13994	Wednesday	200	Edge
25	23.56.8.163.5	[29/Nov/2017:13:38:00]	GET /css/style.css	laptop	4665	Wednesday	200	Edge
26	24.32.226.20	[29/Nov/2017:13:38:00]	GET /js/vendor/mod	laptop	69067	Wednesday	200	Opera
27	25.120.231.1	[29/Nov/2017:13:38:00]	GET /css/main.css	mobile	543	Wednesday	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

The screenshot shows the Microsoft Azure Resource Groups blade. The top navigation bar includes 'Microsoft Azure', 'Upgrade', a search bar ('Search resources, services, and docs (G+)'), and user information ('tgokul24@gmail.com DEFAULT DIRECTORY (TGOKUL2...'). The main area is titled 'mathanresourcegroup' (Resource group). The left sidebar contains sections for Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Deployments, Security, Deployment stacks, Policies, Properties, Locks, Cost analysis, Cost alerts (preview), and Budgets. The 'Essentials' section is expanded, showing the 'Resources' tab selected. A filter bar at the top of the list area shows 'Type equals all' and 'Location equals all'. Below the filters, it says 'Showing 0 to 0 of 0 records.' and 'No grouping' and 'List view' dropdowns. The main content area displays a large gray cube icon and the message 'No resources match your filters' with the sub-instruction 'Try changing or clearing your filters.' Below this are 'Create resources' and 'Clear filters' buttons, and a 'Give feedback' link.

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

The screenshot shows the Microsoft Azure Storage Accounts blade. The top navigation bar includes 'Microsoft Azure', 'Upgrade', a search bar ('Search resources, services, and docs (G+)'), and user information ('tgokul24@gmail.com DEFAULT DIRECTORY (TGOKUL2...'). The main area is titled 'mathanstorage' (Storage account). The left sidebar contains sections for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, Storage Mover, Containers, and File shares. The 'Properties' tab is selected in the main content area. It displays the following details:

Setting	Value
Resource group (move)	: mathanresourcegroup
Location	: eastus
Subscription (move)	: Free_Trial
Subscription ID	: 4345dd7d-f9a0-4981-95b2-3725a3bbdb8f
Disk state	: Available
Tags (edit)	: Add tags

Below the properties, there are sections for 'Blob service' (Hierarchical namespace: Disabled, Default access tier: Hot, Blob anonymous access: Disabled) and 'Security' (Require secure transfer for REST API operations: Enabled, Storage account key access: Enabled).

## Created Container

The screenshot shows the Microsoft Azure Storage account containers page. The left sidebar includes links for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, and Storage Mover. Under Data storage, 'Containers' is selected. The main area displays a table of containers:

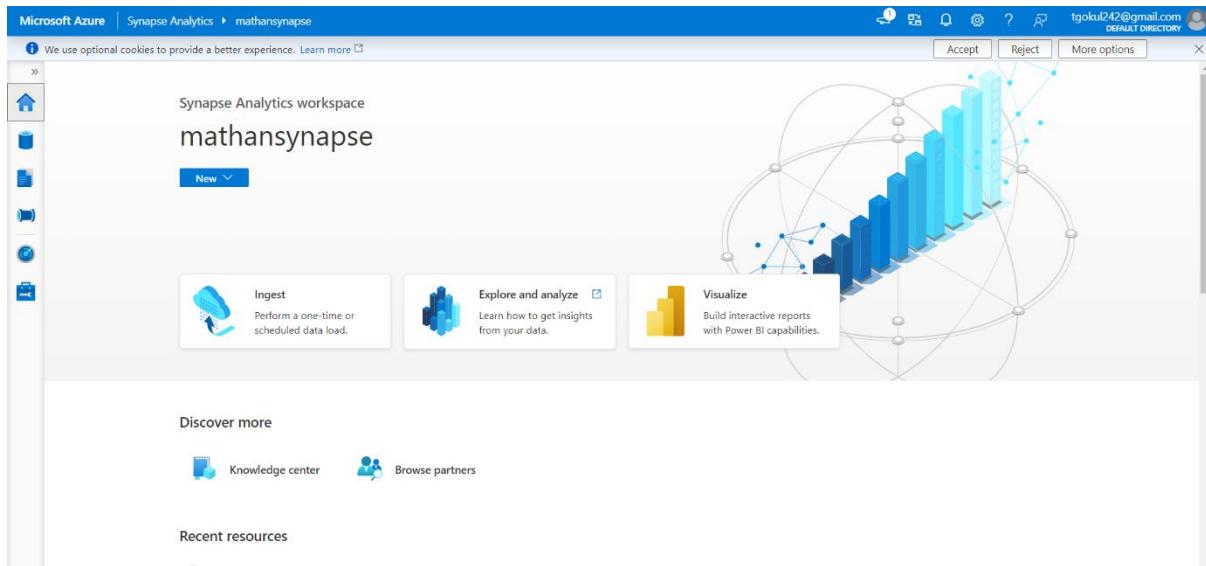
Name	Last modified	Anonymous access level	Lease state
Logs	3/8/2024, 11:56:10 PM	Private	Available
source	3/8/2024, 11:57:30 PM	Private	Available

## Created Synapse

The screenshot shows the Microsoft Azure Synapse Analytics workspace overview page for 'mathansynapse'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Microsoft Entra ID, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools, Data Explorer pools (preview)), and Security (Encryption, Networking). The main area displays the workspace's configuration details under the 'Essentials' tab:

Resource group (move)	: mathanresourcegroup
Status	: Succeeded
Location	: East US
Subscription (move)	: Free_Trial
Subscription ID	: 4345dd7d-f9a0-4981-95b2-3725a3bbdb8f
Managed virtual network	: No
Managed identity object ...	: b53965d5-7a14-4e85-92b8-dea0f3d317c7
Workspace web URL	: <a href="https://web.azuresynthesize.net?workspace=%2bsubscriptions%2b&amp;resourceId=%2bsubscriptions%2b&amp;id=%2bf1111111-1111-1111-1111-111111111111">https://web.azuresynthesize.net?workspace=%2bsubscriptions%2b&amp;resourceId=%2bsubscriptions%2b&amp;id=%2bf1111111-1111-1111-1111-111111111111</a>
Tags (edit)	: Add tags

Below the essentials, there are sections for Getting started (Open Synapse Studio, Read documentation) and Analytics pools.



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

**New Apache Spark pool**

**Basics** • Additional settings \* Tags Review + create

Create an Synapse Analytics Apache Spark pool with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize.

**Apache Spark pool details**

Name your Apache Spark pool and choose its initial settings.

Apache Spark pool name \* Apachesparkpool

Isolated compute \*  Enabled  Disabled

Node size family \* Memory Optimized

Node size \* Small (4 vCores / 32 GB)

Autoscale \*  Enabled  Disabled

Number of nodes \* 3 CO 10

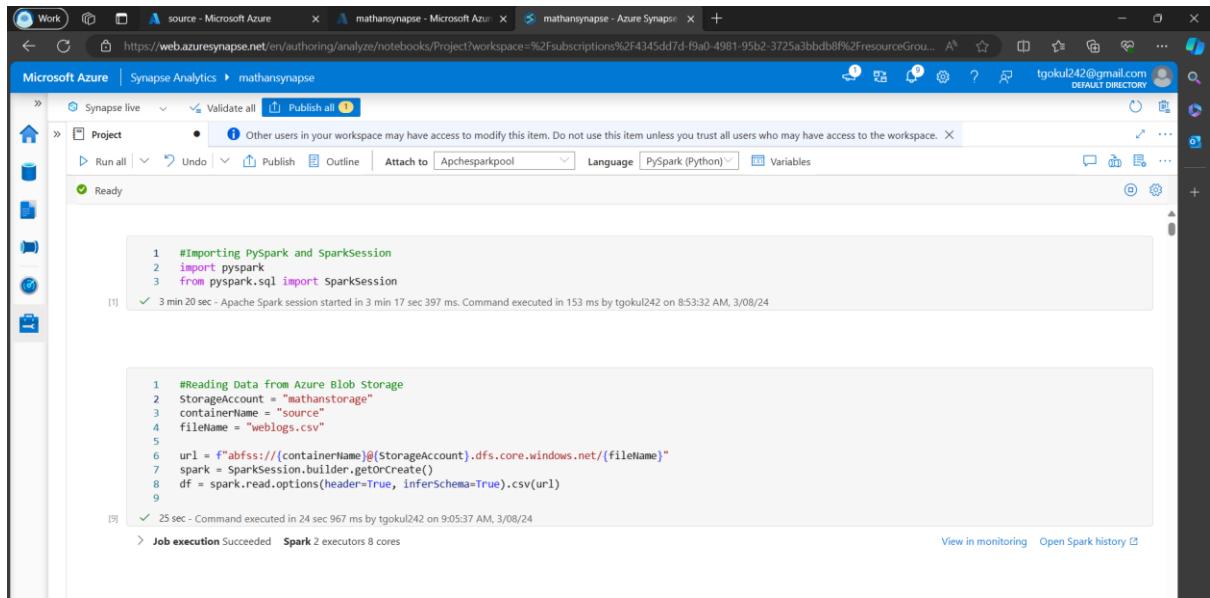
Estimated price  **137.77 to 459.23 INR** View pricing details

Dynamically allocate executors \*  Enabled  Disabled

**Review + create** **Next: Additional settings >** **Cancel**

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



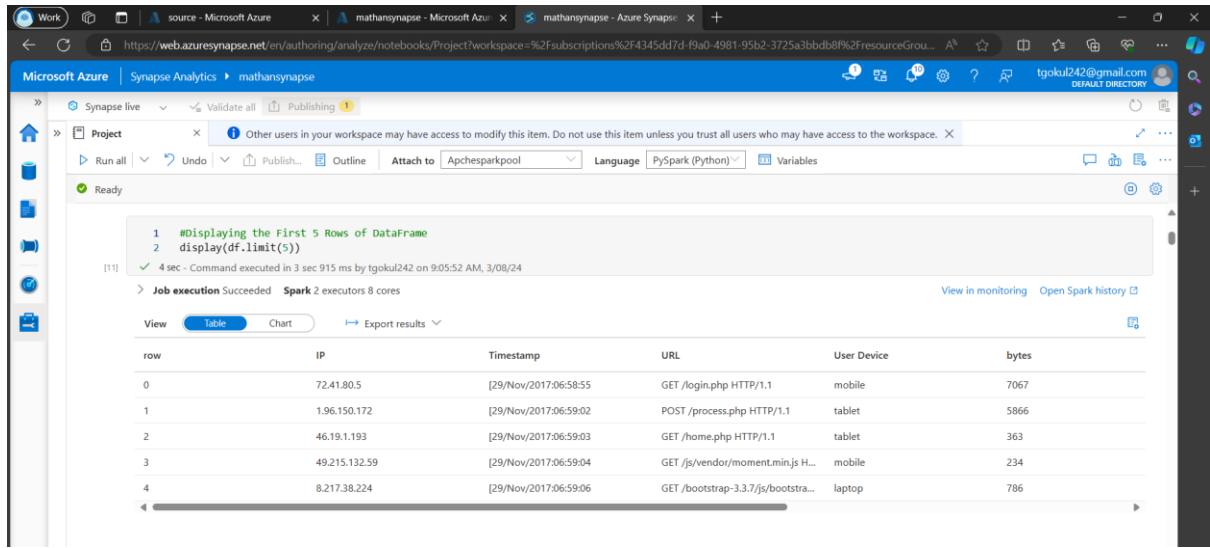
The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. A Python notebook titled 'mathansynapse' is open. The code initializes a PySpark session and reads data from Azure Blob Storage:

```
1 #Importing PySpark and SparkSession
2 import pyspark
3 from pyspark.sql import SparkSession
4
5 #Reading Data from Azure Blob Storage
6 StorageAccount = "mathanstorage"
7 containerName = "source"
8 fileName = "weblogs.csv"
9 url = f"abfss://[{containerName}]{(StorageAccount).dfs.core.windows.net/}{fileName}"
10 spark = SparkSession.builder.getOrCreate()
11 df = spark.read.options(header=True, inferSchema=True).csv(url)
```

The output shows the command was executed successfully in 24 seconds on 8 cores.

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 screenshot shows the Microsoft Azure Synapse Analytics workspace interface. A Python notebook titled 'mathansynapse' is open. The code displays the first five rows of the DataFrame `df`:

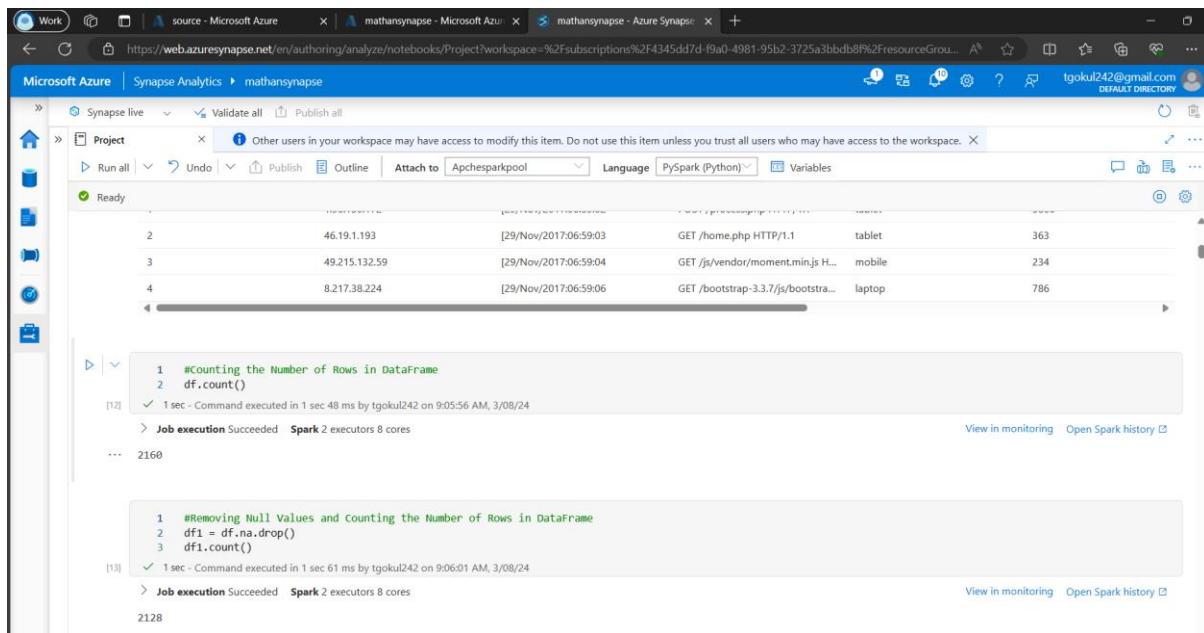
```
1 #displaying the First 5 Rows of DataFrame
2 display(df.limit(5))
```

The output shows the command was executed successfully in 4 seconds on 8 cores. The displayed data is:

row	IP	Timestamp	URL	User Device	bytes
0	72.41.80.5	[29/Nov/2017:06:58:55	GET /login.php HTTP/1.1	mobile	7067
1	1.96.150.172	[29/Nov/2017:06:59:02	POST /process.php HTTP/1.1	tablet	5866
2	46.19.1.193	[29/Nov/2017:06:59:03	GET /home.php HTTP/1.1	tablet	363
3	49.215.132.59	[29/Nov/2017:06:59:04	GET /js/vendor/moment.min.js H...	mobile	234
4	8.217.38.224	[29/Nov/2017:06:59:06	GET /bootstrap-3.3.7/js/bootstra...	laptop	786

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 screenshot shows the Microsoft Azure Synapse Analytics workspace interface. The top navigation bar includes tabs for 'Work', 'source - Microsoft Azure', 'mathansynapse - Microsoft Azure', and 'mathansynapse - Azure Synapse'. The main area has a 'Project' tab selected, showing a 'Ready' status. Below the status, there is a table with four rows of log data:

2	46.19.1.193	[29/Nov/2017:06:59:03]	GET /home.php HTTP/1.1	tablet	363
3	49.215.132.59	[29/Nov/2017:06:59:04]	GET /js/vendor/moment.min.js H...	mobile	234
4	8.217.38.224	[29/Nov/2017:06:59:06]	GET /bootstrap-3.3.7/js/bootstra...	laptop	786

Below the table, two code snippets are shown:

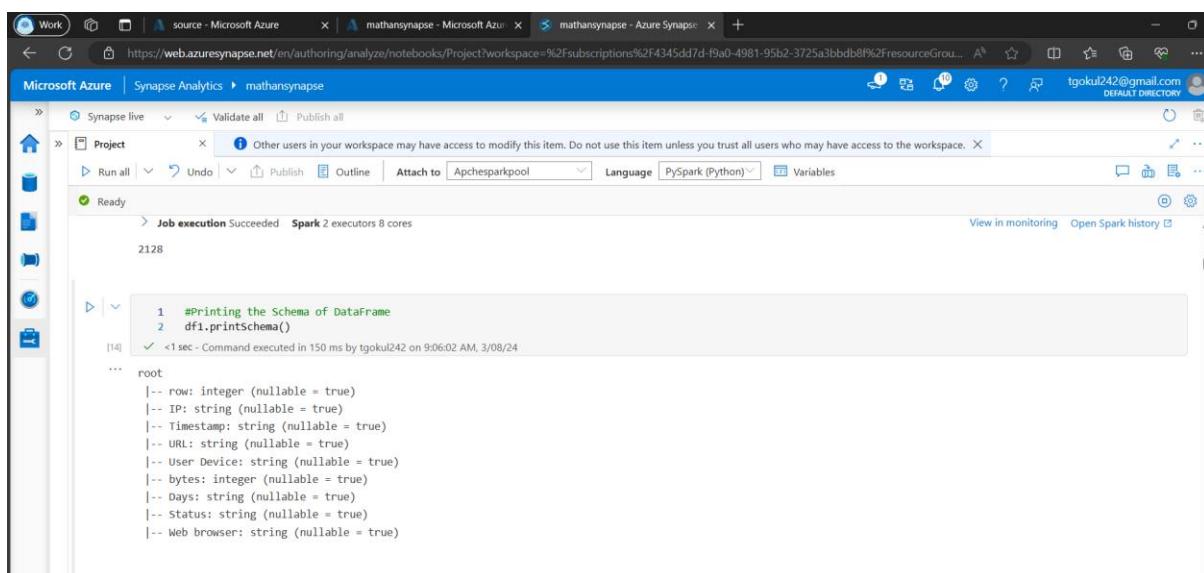
```
1 #Counting the Number of Rows in DataFrame
2 df.count()
[12] ✓ 1 sec - Command executed in 1 sec 48 ms by tgokul242 on 9:05:56 AM, 3/08/24
> Job execution Succeeded Spark 2 executors 8 cores
...
2160

1 #Removing Null Values and Counting the Number of Rows in DataFrame
2 df1 = df.na.drop()
3 df1.count()
[13] ✓ 1 sec - Command executed in 1 sec 61 ms by tgokul242 on 9:06:01 AM, 3/08/24
> Job execution Succeeded Spark 2 executors 8 cores
2128
```

Execution details for each command are provided, including the time taken and the number of executors used.

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



The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. The top navigation bar includes tabs for 'Work', 'source - Microsoft Azure', 'mathansynapse - Microsoft Azure', and 'mathansynapse - Azure Synapse'. The main area has a 'Project' tab selected, showing a 'Ready' status. Below the status, there is a message indicating a successful job execution:

> Job execution Succeeded Spark 2 executors 8 cores

Execution details for the previous command are shown:

```
2128

1 #Printing the Schema of DataFrame
2 df1.printSchema()
[14] ✓ <1 sec - Command executed in 150 ms by tgokul242 on 9:06:02 AM, 3/08/24
...
root
 |-- 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)
```

The schema of the DataFrame `df1` is printed, showing various columns and their data types.

## Identifying Duplicate IP Addresses in DataFrame



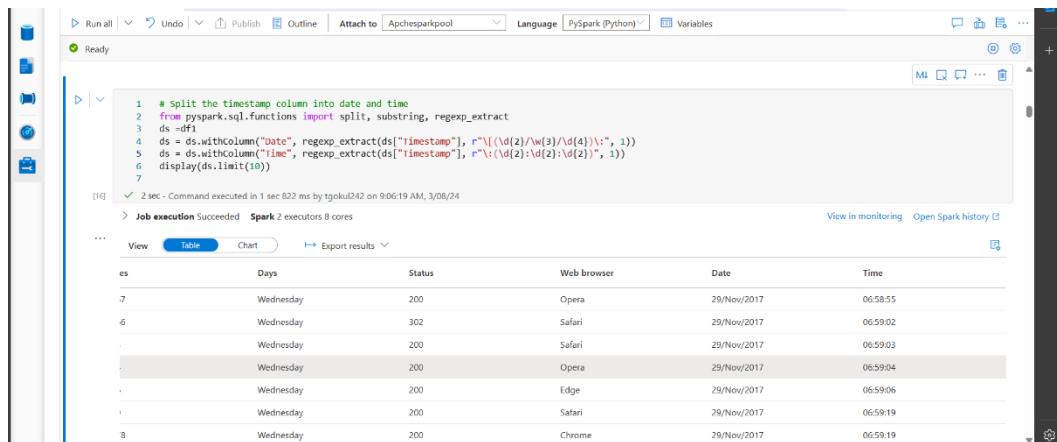
```
1 #Calculating and Printing the Number of Duplicate IP Addresses in DataFrame
2 total_rows = df1.count()
3 distinct_ip = df1.select("IP").distinct().count()
4 duplicate = total_rows - distinct_ip
5 print(duplicate)
```

[15] ✓ 4 sec - Command executed in 3 sec 743 ms by tgokul242 on 9:06:09 AM, 3/08/24  
Job execution Succeeded Spark 2 executors 8 cores

View in monitoring Open Spark history

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



```
1 # split the timestamp column into date and time
2 from pyspark.sql.functions import split, substring, regexp_extract
3 ds = df1
4 ds = ds.withColumn("Date", regexp_extract(ds["timestamp"], r"\[(\d{2})/(\w{3})/(\d{4})\:|(\d{2}):|(\d{2})\]", 1))
5 ds = ds.withColumn("Time", regexp_extract(ds["timestamp"], r"\[(\d{2})/(\w{3})/(\d{4})\:|(\d{2}):|(\d{2})\]", 2))
6 display(ds.limit(10))
```

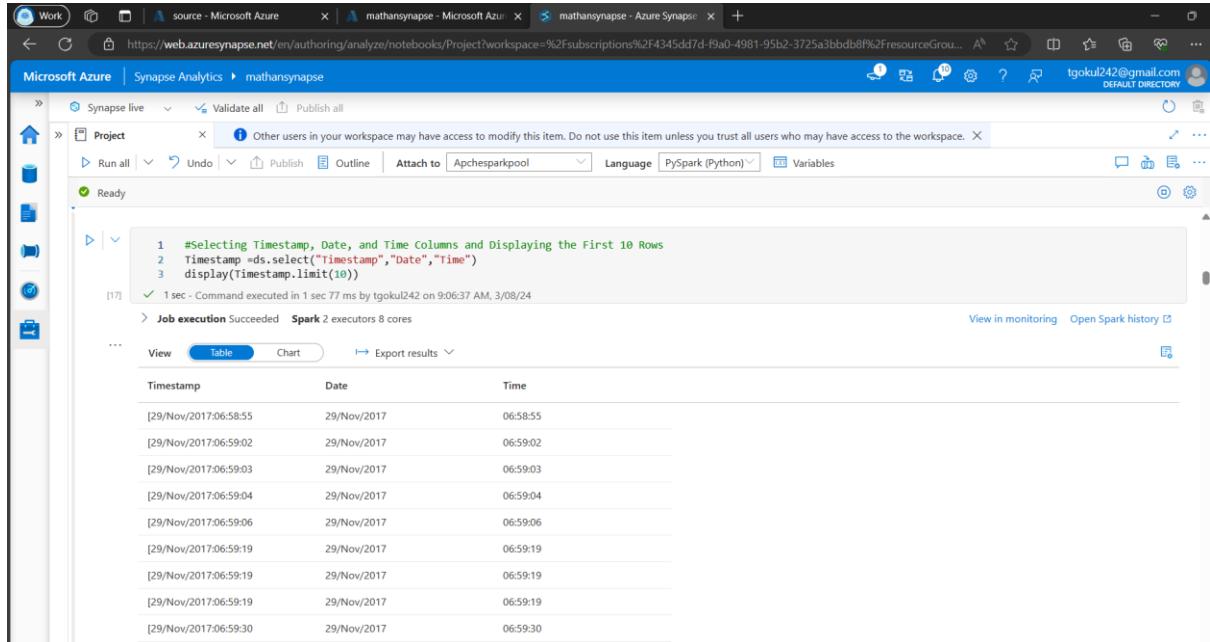
[16] ✓ 2 sec - Command executed in 1 sec 822 ms by tgokul242 on 9:06:19 AM, 3/08/24  
Job execution Succeeded Spark 2 executors 8 cores

View in monitoring Open Spark history

es	Days	Status	Web browser	Date	Time
7	Wednesday	200	Opera	29/Nov/2017	06:58:55
6	Wednesday	302	Safari	29/Nov/2017	06:59:02
5	Wednesday	200	Safari	29/Nov/2017	06:59:03
4	Wednesday	200	Opera	29/Nov/2017	06:59:04
3	Wednesday	200	Edge	29/Nov/2017	06:59:06
2	Wednesday	200	Safari	29/Nov/2017	06:59:19
1	Wednesday	200	Chrome	29/Nov/2017	06:59:19
8	Wednesday	200			

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.

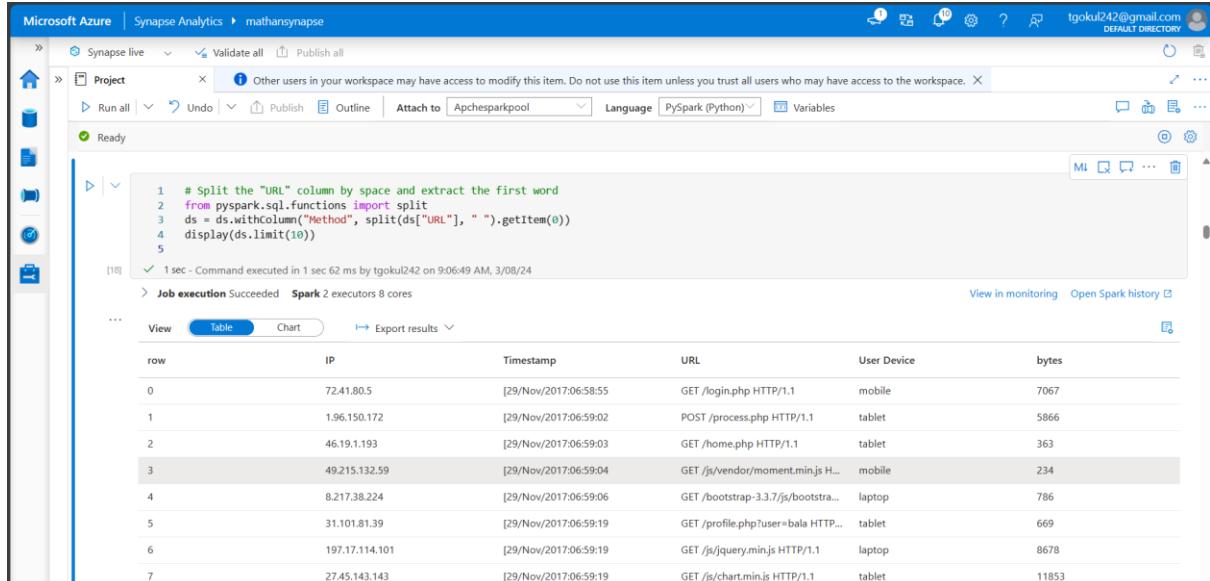
## Displaying the Timestamp



```
1 #Selecting Timestamp, Date, and Time Columns and Displaying the First 10 Rows
2 Timestamp =ds.select("Timestamp","Date","Time")
3 display(Timestamp.limit(10))
```

Timestamp	Date	Time
[29/Nov/2017:06:58:55	29/Nov/2017	06:58:55
[29/Nov/2017:06:59:02	29/Nov/2017	06:59:02
[29/Nov/2017:06:59:03	29/Nov/2017	06:59:03
[29/Nov/2017:06:59:04	29/Nov/2017	06:59:04
[29/Nov/2017:06:59:06	29/Nov/2017	06:59:06
[29/Nov/2017:06:59:19	29/Nov/2017	06:59:19
[29/Nov/2017:06:59:19	29/Nov/2017	06:59:19
[29/Nov/2017:06:59:19	29/Nov/2017	06:59:19
[29/Nov/2017:06:59:30	29/Nov/2017	06:59:30

## Extracting HTTP Method from URL Column

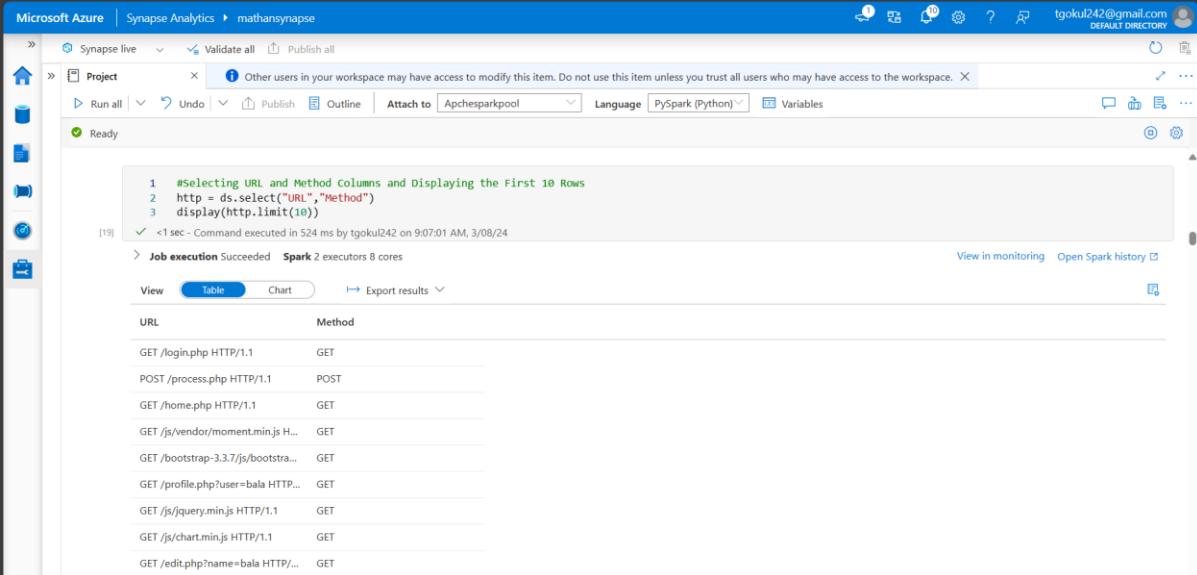


```
1 # Split the "URL" column by space and extract the first word
2 from pyspark.sql.functions import split
3 ds = ds.withColumn("Method", split(ds["URL"], " ").getItem(0))
4 display(ds.limit(10))
```

row	IP	Timestamp	URL	User Device	bytes
0	72.41.80.5	[29/Nov/2017:06:58:55	GET /login.php HTTP/1.1	mobile	7067
1	1.96.150.172	[29/Nov/2017:06:59:02	POST /process.php HTTP/1.1	tablet	5866
2	46.19.1.193	[29/Nov/2017:06:59:03	GET /home.php HTTP/1.1	tablet	363
3	49.215.132.259	[29/Nov/2017:06:59:04	GET /js/vendor/moment.min.js H...	mobile	234
4	8.217.38.224	[29/Nov/2017:06:59:06	GET /bootstrap-3.3.7/js/bootstrap...	laptop	786
5	31.101.81.39	[29/Nov/2017:06:59:19	GET /profile.php?user=bala HTTP...	tablet	669
6	197.17.114.101	[29/Nov/2017:06:59:19	GET /js/jquery.min.js HTTP/1.1	laptop	8678
7	27.45.143.143	[29/Nov/2017:06:59:19	GET /js/chart.min.js HTTP/1.1	tablet	11853

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



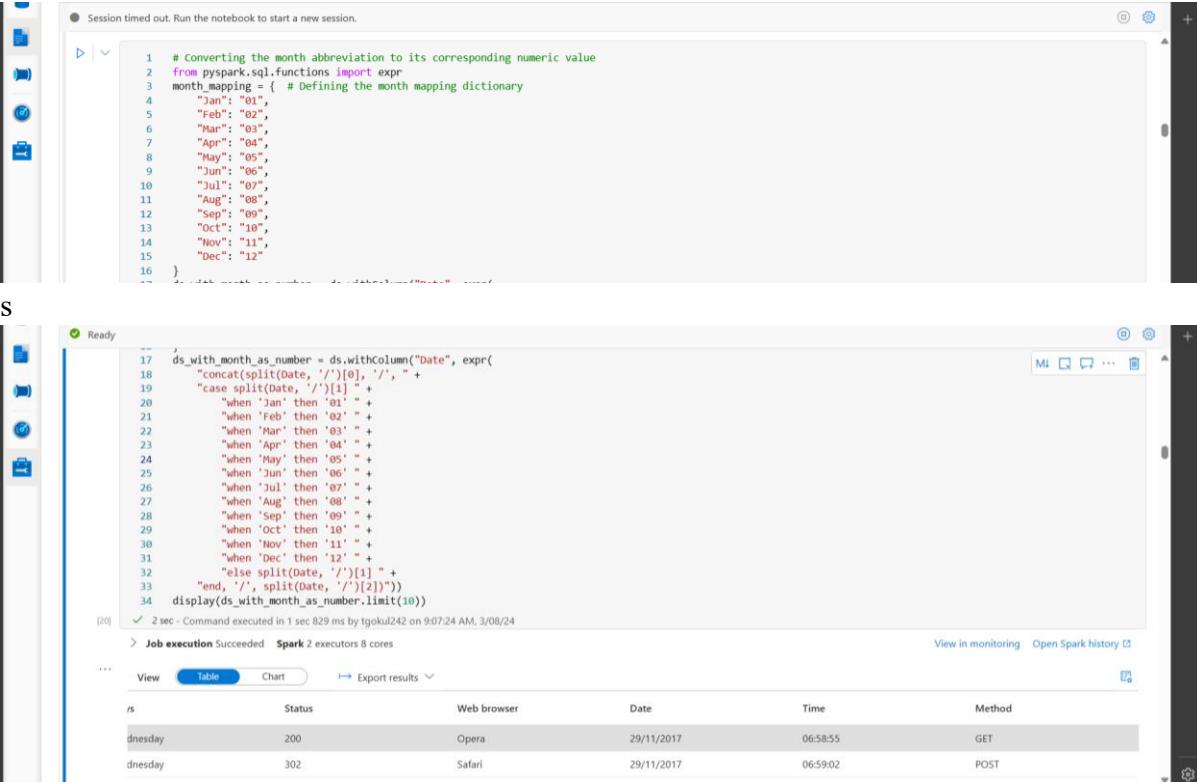
The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. A job titled "Ready" has been executed successfully with 2 cores. The code used is:

```
1 #Selecting URL and Method Columns and Displaying the First 10 Rows
2 http = ds.select("URL","Method")
3 display(http.limit(10))
```

The output table displays the following data:

URL	Method
GET /login.php HTTP/1.1	GET
POST /process.php HTTP/1.1	POST
GET /home.php HTTP/1.1	GET
GET /js/vendor/moment.min.js H...	GET
GET /bootstrap-3.3.7/js/bootstrap...	GET
GET /profile.php?user=bala HTTP/...	GET
GET /js/jquery.min.js HTTP/1.1	GET
GET /js/chart.min.js HTTP/1.1	GET
GET /edit.php?name=bala HTTP/...	GET

## Converting Month Abbreviations to Numeric Values in Date Column



The screenshot shows a PySpark notebook cell with the following code:

```
1 # Converting the month abbreviation to its corresponding numeric value
2 from pyspark.sql.functions import expr
3 month_mapping = { # Defining the month mapping dictionary
4     "Jan": "01",
5     "Feb": "02",
6     "Mar": "03",
7     "Apr": "04",
8     "May": "05",
9     "Jun": "06",
10    "Jul": "07",
11    "Aug": "08",
12    "Sep": "09",
13    "Oct": "10",
14    "Nov": "11",
15    "Dec": "12"
16 }
```

The cell output shows the conversion of month abbreviations to numeric values:

```
17 ds_with_month_as_number = ds.withColumn("Date", expr(
18     "concat(split(date, '/') [0], '/', " +
19     "case split(date, '/') [1] " +
20     "when 'Jan' then '01' " +
21     "when 'Feb' then '02' " +
22     "when 'Mar' then '03' " +
23     "when 'Apr' then '04' " +
24     "when 'May' then '05' " +
25     "when 'Jun' then '06' " +
26     "when 'Jul' then '07' " +
27     "when 'Aug' then '08' " +
28     "when 'Sep' then '09' " +
29     "when 'Oct' then '10' " +
30     "when 'Nov' then '11' " +
31     "when 'Dec' then '12' " +
32     "else split(date, '/') [1] " +
33     "end, '/', split(date, '/') [2]))")
34 display(ds_with_month_as_number.limit(10))
```

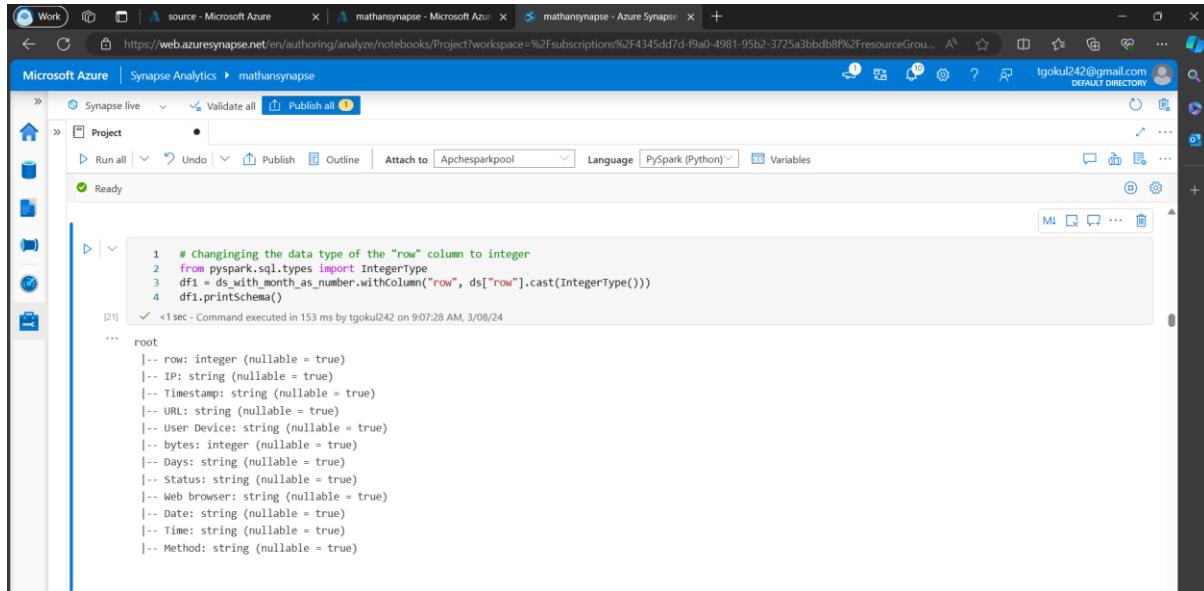
The output table shows the converted data:

rs	Status	Web browser	Date	Time	Method
dnesday	200	Opera	29/11/2017	06:58:55	GET
dnesday	302	Safari	29/11/2017	06:59:02	POST

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 mapping dictionary.

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



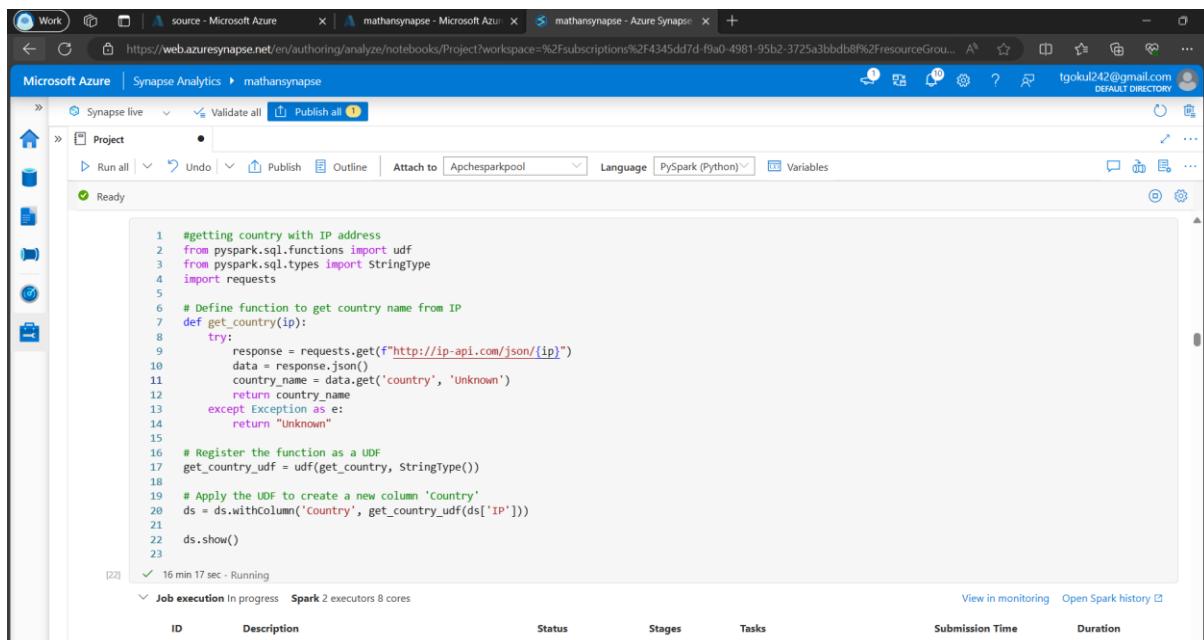
```

1 # Changing the data type of the "row" column to integer
2 from pyspark.sql.types import IntegerType
3 df1 = ds.withColumn("row", ds["row"].cast(IntegerType()))
4 df1.printSchema()

```

[21] ✓ <1 sec - Command executed in 153 ms by tgokul242 on 9:07:28 AM, 3/08/24

## Extracting Country from IP Addresses



```

1 #getting country with IP address
2 from pyspark.sql.functions import udf
3 from pyspark.sql.types import StringType
4 import requests
5
6 # Define function to get country name from IP
7 def get_country(ip):
8     try:
9         response = requests.get(f"http://ip-api.com/json/{ip}")
10        data = response.json()
11        country_name = data.get('country', 'Unknown')
12        return country_name
13    except Exception as e:
14        return "Unknown"
15
16 # Register the function as a UDF
17 get_country_udf = udf(get_country, StringType())
18
19 # Apply the UDF to create a new column 'Country'
20 ds = ds.withColumn('Country', get_country_udf(ds['IP']))
21
22 ds.show()
23

```

[22] ✓ 16 min 17 sec - Running

Job execution In progress Spark 2 executors 8 cores

ID	Description	Status	Stages	Tasks	Submission Time	Duration
----	-------------	--------	--------	-------	-----------------	----------

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

The screenshot shows the Microsoft Azure Synapse Analytics interface. In the center, there is a table titled "Job execution In progress" with 2 executors and 8 cores. It lists three jobs: Job 21, Stage 31, and Job 79, all of which have succeeded. Below the table is a large log table with columns: row, IP, Timestamp, URL, User, Device, bytes, Days, Status, Web browser, Date, Time, Method, and Country. The log contains 13 rows of data, mostly from Nov 29, 2017, showing various user agents and countries like United States, South Korea, Belgium, Taiwan, Hong Kong, United Kingdom, Tunisia, China, United States, Italy, Greece, France, and Spain.

## Analyzing Country Occurrences in Data

The screenshot shows the Microsoft Azure Synapse Analytics interface. On the left, a code editor displays a PySpark snippet:

```

1 #Counting Country Occurrences and Displaying Top 30
2 from pyspark.sql.functions import count
3 country_counts = df1.groupBy("country").agg(count("*").alias("count"))
4 display(country_counts.limit(30))

```

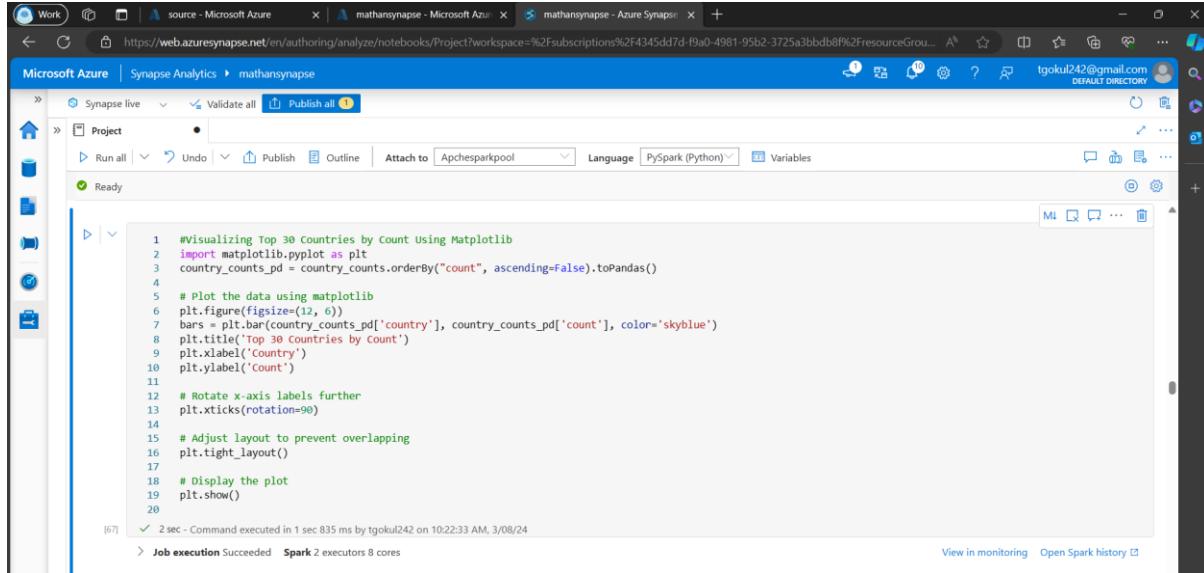
The output window shows the command was executed in 3 seconds. Below the code editor is a table titled "View" showing the top 30 countries and their counts:

country	count
Russia	57
Sweden	9
The Netherlands	34
Philippines	9
Djibouti	1
Malaysia	10
Singapore	34
Germany	72
Cambodia	1

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



```

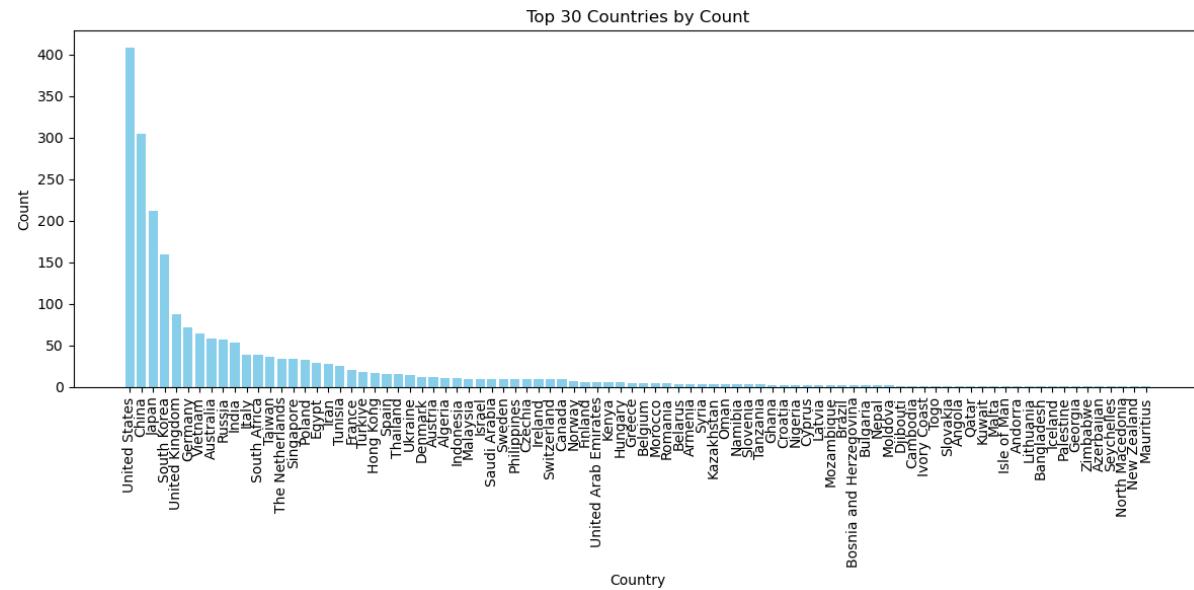
1 # Visualizing Top 30 Countries by Count Using Matplotlib
2 import matplotlib.pyplot as plt
3 country_counts_pd = country_counts.orderBy("count", ascending=False).toPandas()
4
5 # Plot the data using matplotlib
6 plt.figure(figsize=(12, 6))
7 bars = plt.bar(country_counts_pd['country'], country_counts_pd['count'], color='skyblue')
8 plt.title('Top 30 Countries by Count')
9 plt.xlabel('Country')
10 plt.ylabel('Count')
11
12 # Rotate x-axis labels further
13 plt.xticks(rotation=90)
14
15 # Adjust layout to prevent overlapping
16 plt.tight_layout()
17
18 # Display the plot
19 plt.show()
20

```

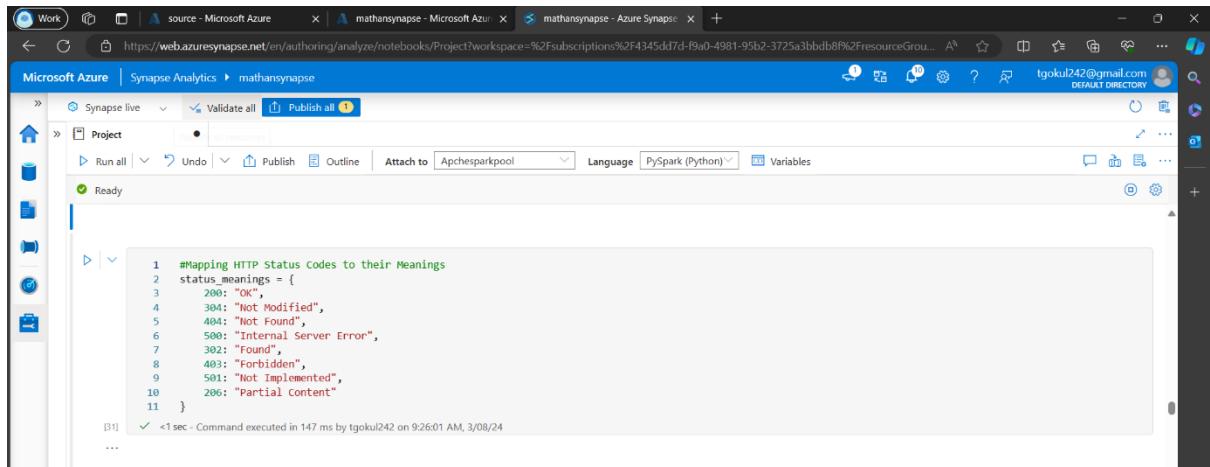
[67] 2 sec - Command executed in 1 sec 835 ms by tgokul242 on 10:22:33 AM, 3/08/24

> Job execution Succeeded Spark 2 executors 8 cores

## Visualization



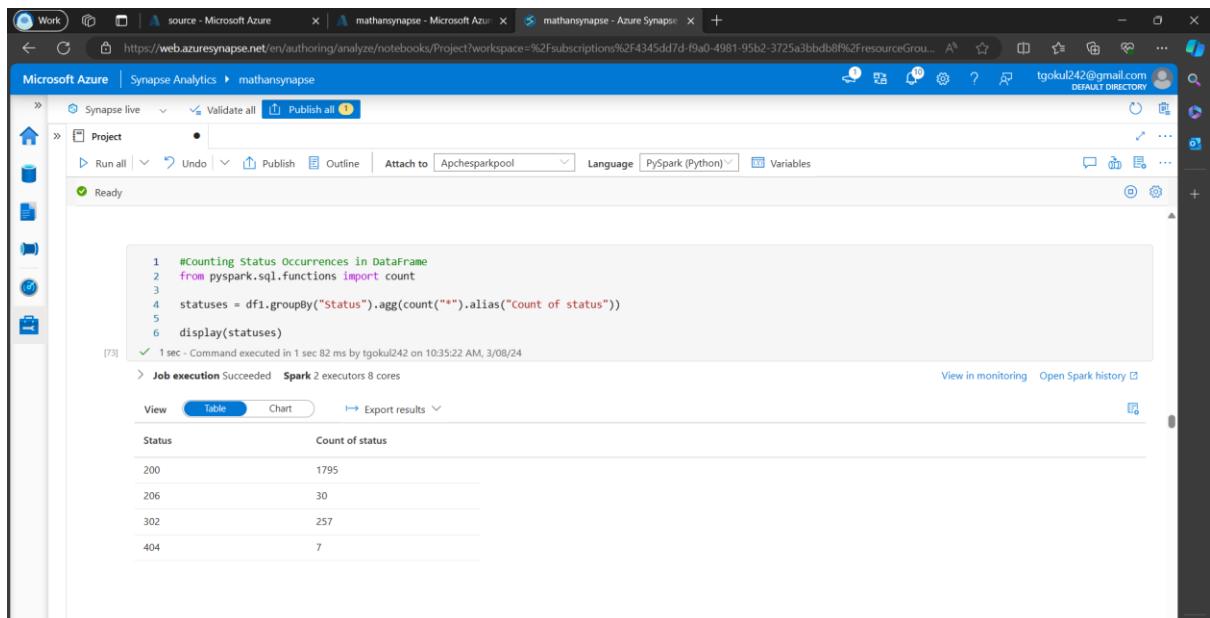
## HTTP Status Code Meanings



```
1 #Mapping HTTP Status Codes to their Meanings
2 status_meanings = {
3     200: "OK",
4     304: "Not Modified",
5     404: "Not Found",
6     500: "Internal Server Error",
7     302: "Found",
8     403: "Forbidden",
9     501: "Not Implemented",
10    206: "Partial Content"
11 }
```

[31] <1 sec - Command executed in 147 ms by tgokul242 on 9:26:01 AM, 3/08/24

## Analyzing Status Occurrences in Data



```
1 #Counting Status Occurrences in DataFrame
2 from pyspark.sql.functions import count
3
4 statuses = df1.groupBy("Status").agg(count("*").alias("Count of status"))
5
6 display(statuses)
```

[73] 1 sec - Command executed in 1 sec 82 ms by tgokul242 on 10:35:22 AM, 3/08/24

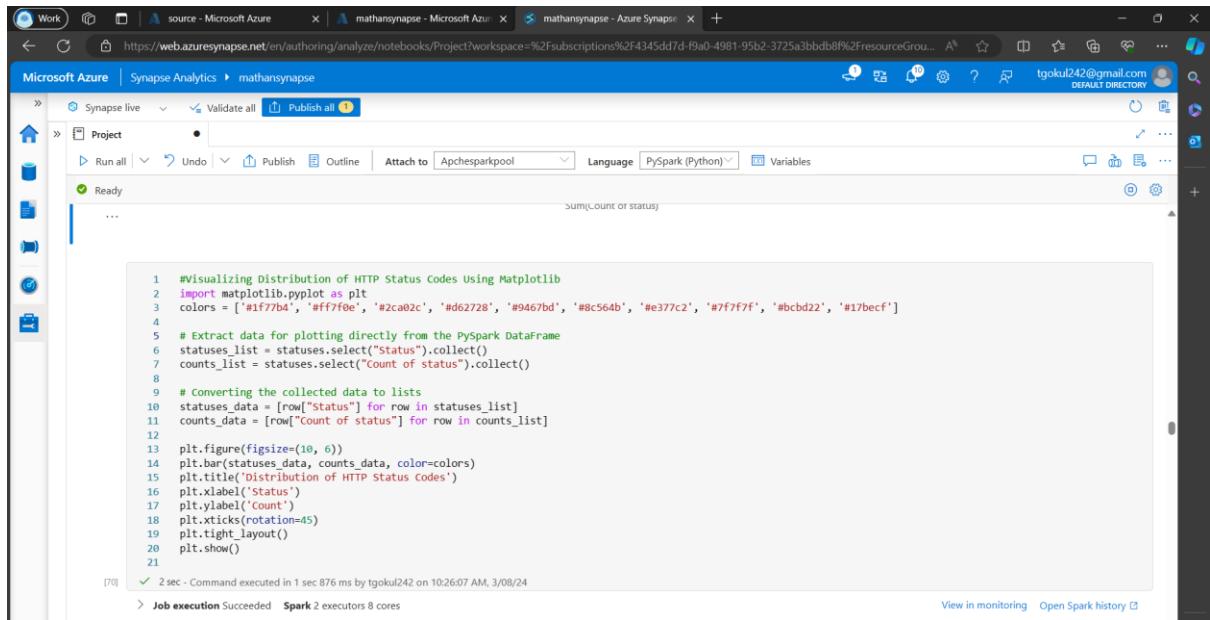
> Job execution Succeeded | Spark 2 executors 8 cores

View Table Chart Export results

Status	Count of status
200	1795
206	30
302	257
404	7

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

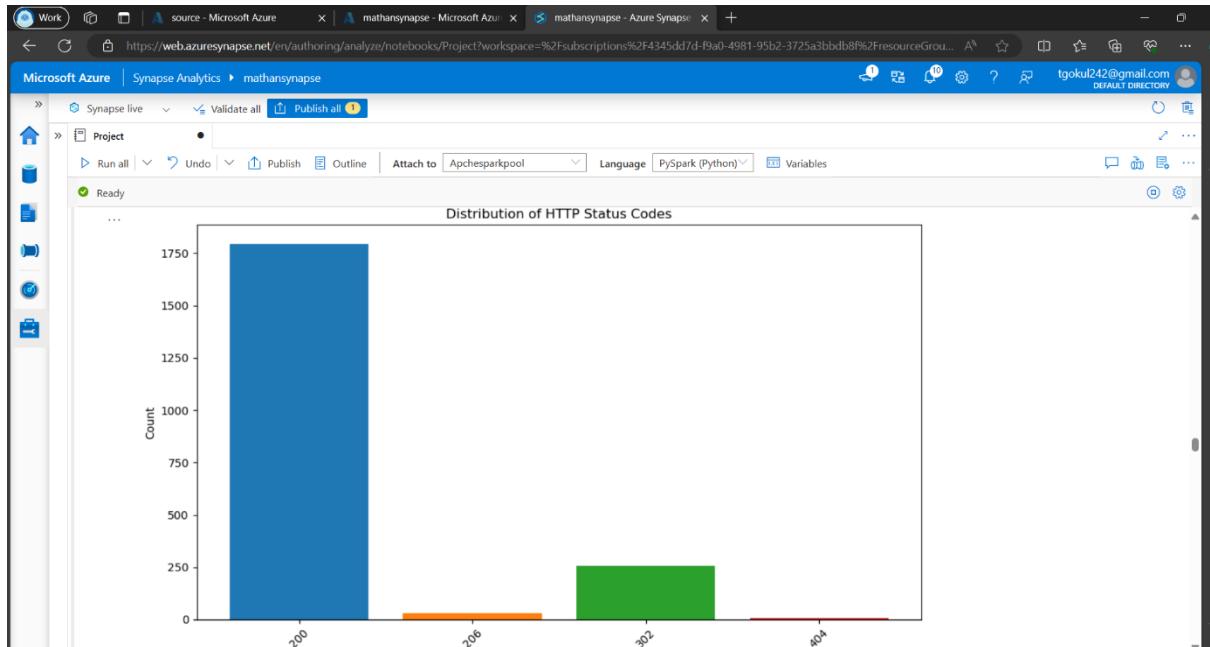


The screenshot shows a Microsoft Azure Synapse Analytics workspace. A Python notebook cell is open, displaying the following code:

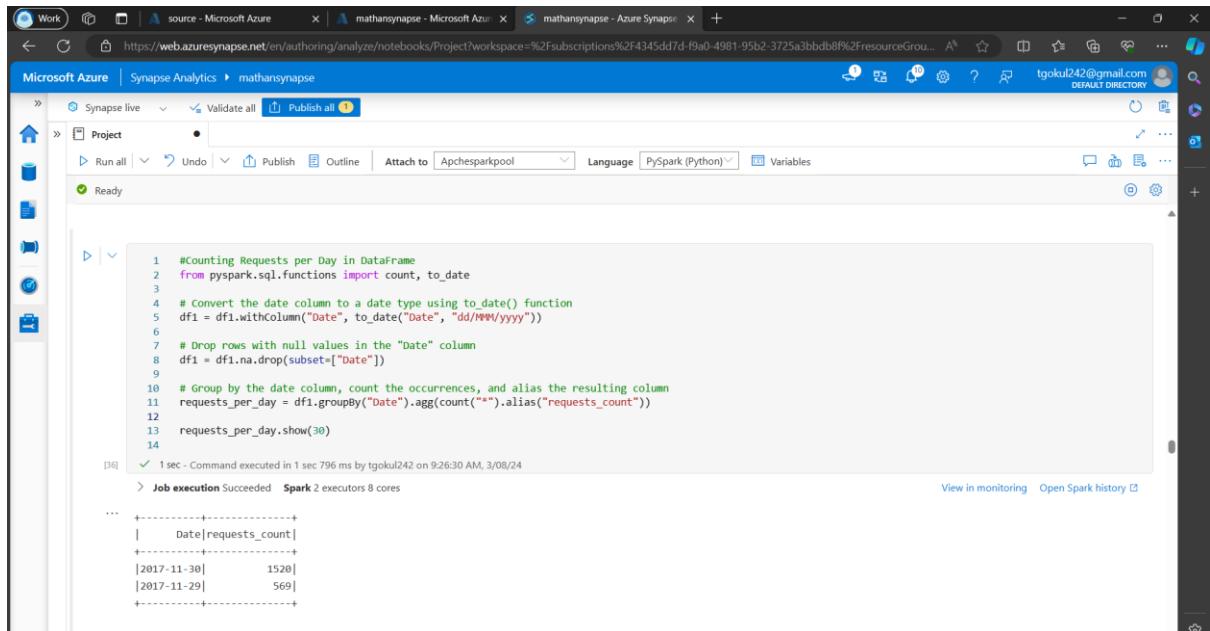
```
1 #Visualizing Distribution of HTTP Status Codes Using Matplotlib
2 import matplotlib.pyplot as plt
3 colors = ['#ff7f7f', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#1f7becf']
4
5 # Extract data for plotting directly from the PySpark DataFrame
6 statuses_list = statuses.select("Status").collect()
7 counts_list = statuses.select("Count of status").collect()
8
9 # Converting the collected data to lists
10 statuses_data = [row["status"] for row in statuses_list]
11 counts_data = [row["Count of status"] for row in counts_list]
12
13 plt.figure(figsize=(10, 6))
14 plt.bar(statuses_data, counts_data, color=colors)
15 plt.title('Distribution of HTTP Status Codes')
16 plt.xlabel('Status')
17 plt.ylabel('Count')
18 plt.xticks(rotation=45)
19 plt.tight_layout()
20 plt.show()
```

The cell has a green checkmark indicating success, with the message "2 sec - Command executed in 1 sec 876 ms by tgokul242 on 10:26:07 AM, 3/08/24". Below the cell, it says "Job execution Succeeded Spark 2 executors 8 cores".

## Visualization



## Analyzing Daily Request Counts



The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. A PySpark notebook titled "mathansynapse" is open. The code snippet calculates the number of requests per day in a DataFrame named '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.

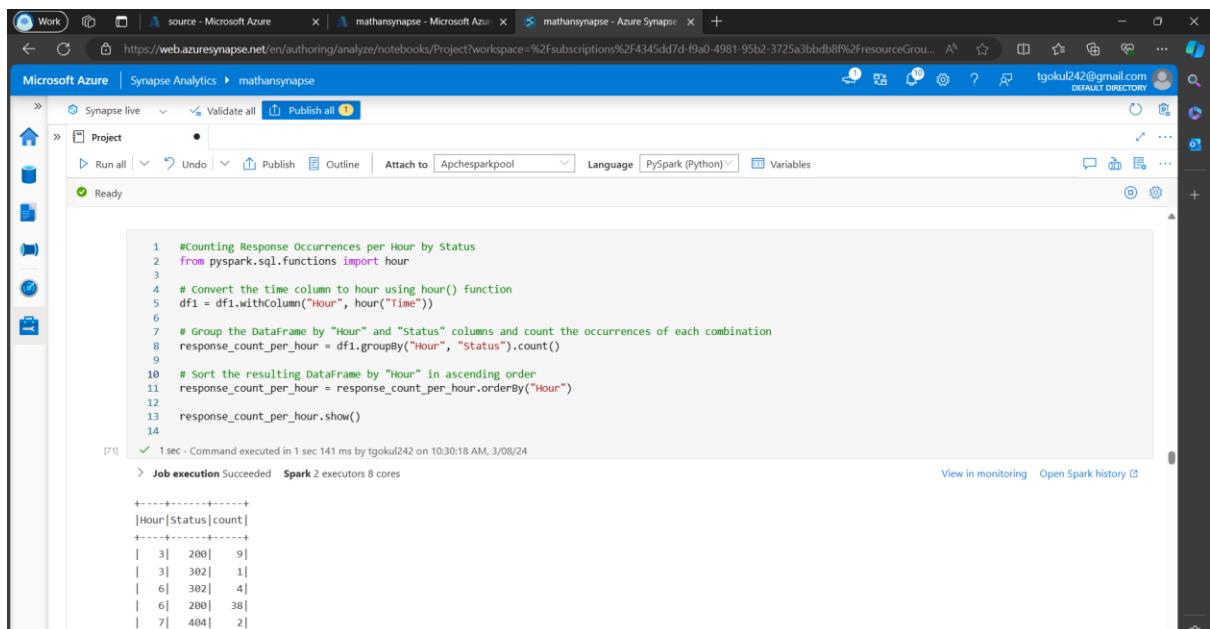
```
1 #Counting Requests per Day in DataFrame
2 from pyspark.sql.functions import count, to_date
3
4 # Convert the date column to a date type using to_date() function
5 df1 = df1.withColumn("Date", to_date("Date", "dd/MMM/yyyy"))
6
7 # Drop rows with null values in the "Date" column
8 df1 = df1.na.drop(subset=["Date"])
9
10 # Group by the date column, count the occurrences, and alias the resulting column
11 requests_per_day = df1.groupBy("Date").agg(count("*").alias("requests_count"))
12
13 requests_per_day.show(30)
14
```

[30] ✓ 1 sec - Command executed in 1 sec 796 ms by tgokul242 on 9:26:30 AM, 3/08/24  
Job execution Succeeded Spark 2 executors 8 cores

Date	requests_count
2017-11-30	1520
2017-11-29	569

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



The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. A PySpark notebook titled "mathansynapse" is open. The code snippet calculates the number of response occurrences per hour by status. It first converts the 'time' column to hour using the hour() function. Then, it groups the DataFrame by "Hour" and "Status" columns and counts the occurrences of each combination. Finally, it sorts the resulting DataFrame by "Hour" in ascending order. The output displays the hourly response counts for different status codes.

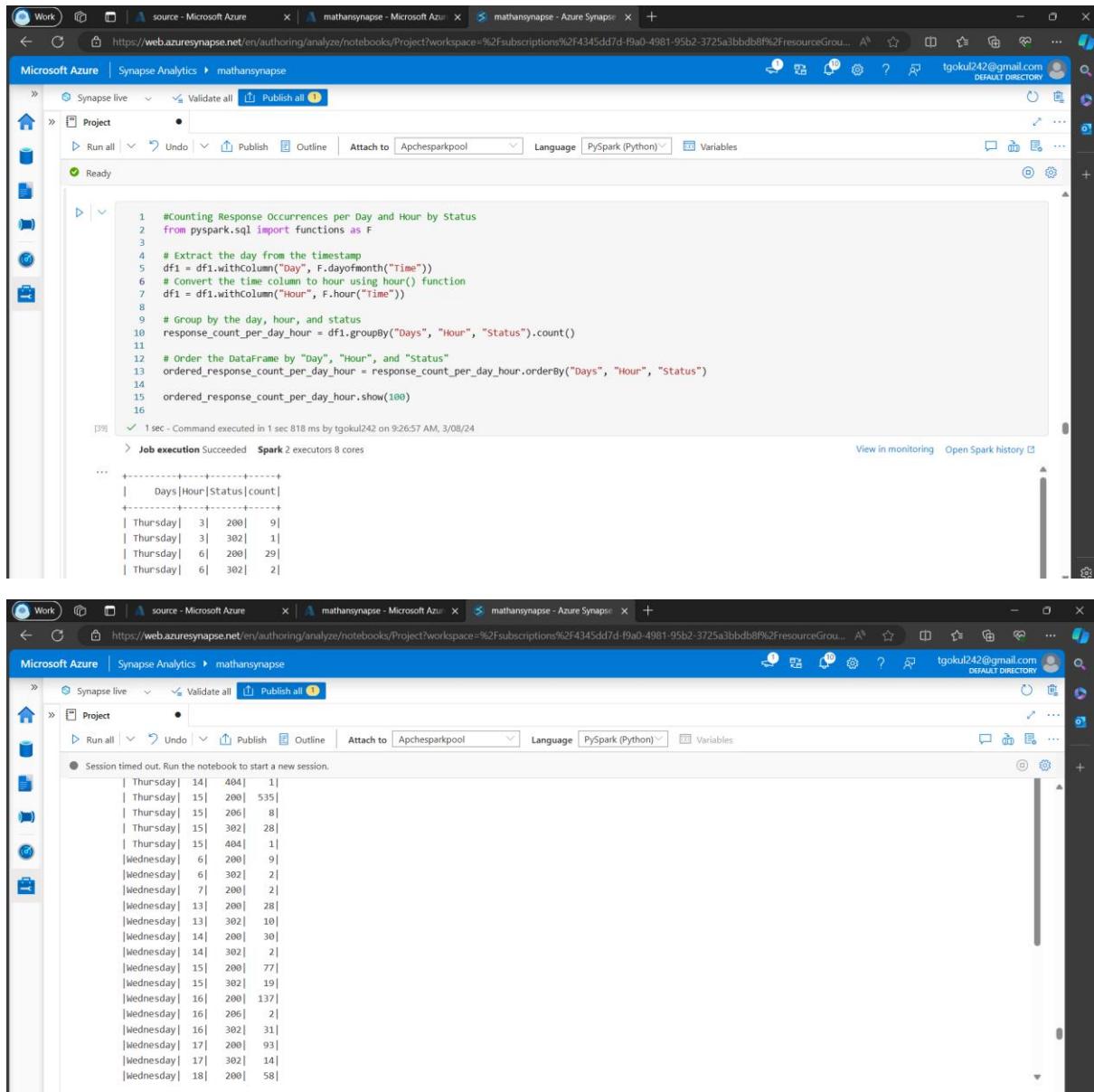
```
1 #Counting Response Occurrences per Hour by Status
2 from pyspark.sql.functions import hour
3
4 # Convert the time column to hour using hour() function
5 df1 = df1.withColumn("hour", hour("time"))
6
7 # Group the DataFrame by "Hour" and "Status" columns and count the occurrences of each combination
8 response_count_per_hour = df1.groupby("Hour", "status").count()
9
10 # Sort the resulting DataFrame by "Hour" in ascending order
11 response_count_per_hour = response_count_per_hour.orderBy("Hour")
12
13 response_count_per_hour.show()
14
```

[17] ✓ 1 sec - Command executed in 1 sec 141 ms by tgokul242 on 10:30:18 AM, 3/08/24  
Job execution Succeeded Spark 2 executors 8 cores

Hour	status	count
3	200	9
3	302	1
6	302	4
6	200	38
7	404	2

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



```

1 #Counting Response Occurrences per Day and Hour by Status
2 from pyspark.sql import functions as F
3
4 # Extract the day from the timestamp
5 df1 = df1.withColumn("Day", F.dayofmonth("Time"))
6 # Convert the time column to hour using hour() function
7 df1 = df1.withColumn("Hour", F.hour("Time"))
8
9 # Group by the day, hour, and status
10 response_count_per_day_hour = df1.groupBy("Days", "Hour", "Status").count()
11
12 # Order the DataFrame by "Days", "Hour", and "status"
13 ordered_response_count_per_day_hour = response_count_per_day_hour.orderBy("Days", "Hour", "Status")
14
15 ordered_response_count_per_day_hour.show(100)
16

```

[38] ✓ 1 sec - Command executed in 1 sec 818 ms by tgokul242 on 9:26:57 AM, 3/08/24

> Job execution Succeeded Spark 2 executors 8 cores

Days	Hour	Status	count
Thursday	3	200	9
Thursday	3	302	1
Thursday	6	200	29
Thursday	6	302	2

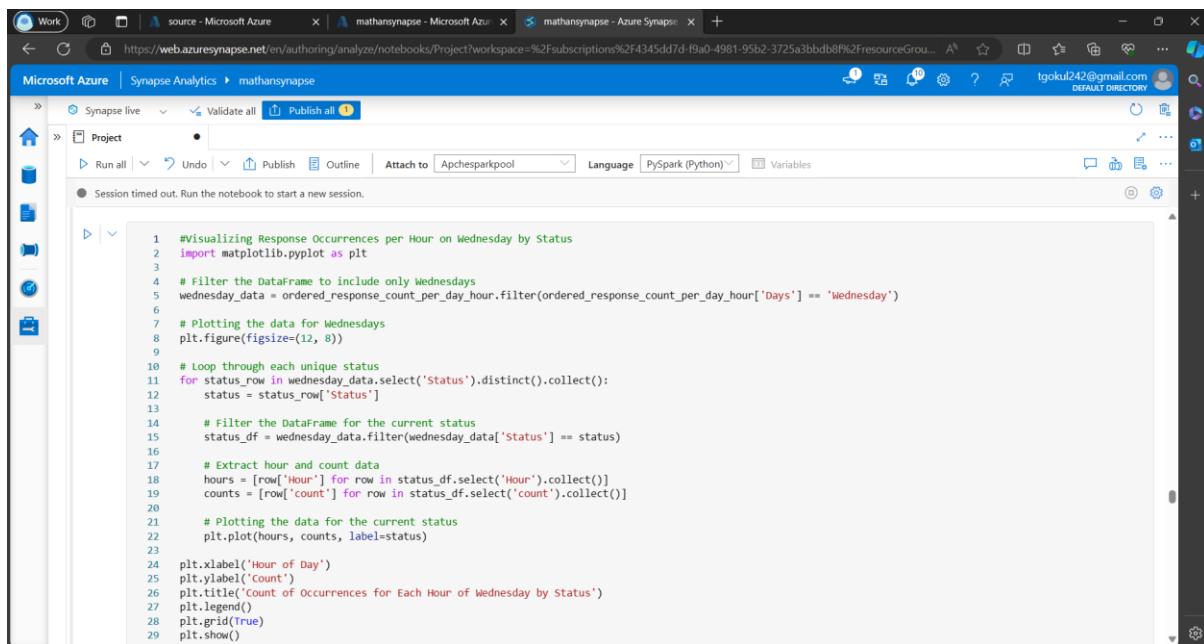
Session timed out. Run the notebook to start a new session.

Days	Hour	Status	count
Thursday	14	494	1
Thursday	15	200	535
Thursday	15	206	8
Thursday	15	302	28
Thursday	15	404	1
Wednesday	6	200	9
Wednesday	6	302	2
Wednesday	7	200	2
Wednesday	13	200	28
Wednesday	13	302	10
Wednesday	14	200	30
Wednesday	14	302	2
Wednesday	15	200	77
Wednesday	15	302	19
Wednesday	16	200	137
Wednesday	16	206	2
Wednesday	16	302	31
Wednesday	17	200	93
Wednesday	17	302	14
Wednesday	18	200	58

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

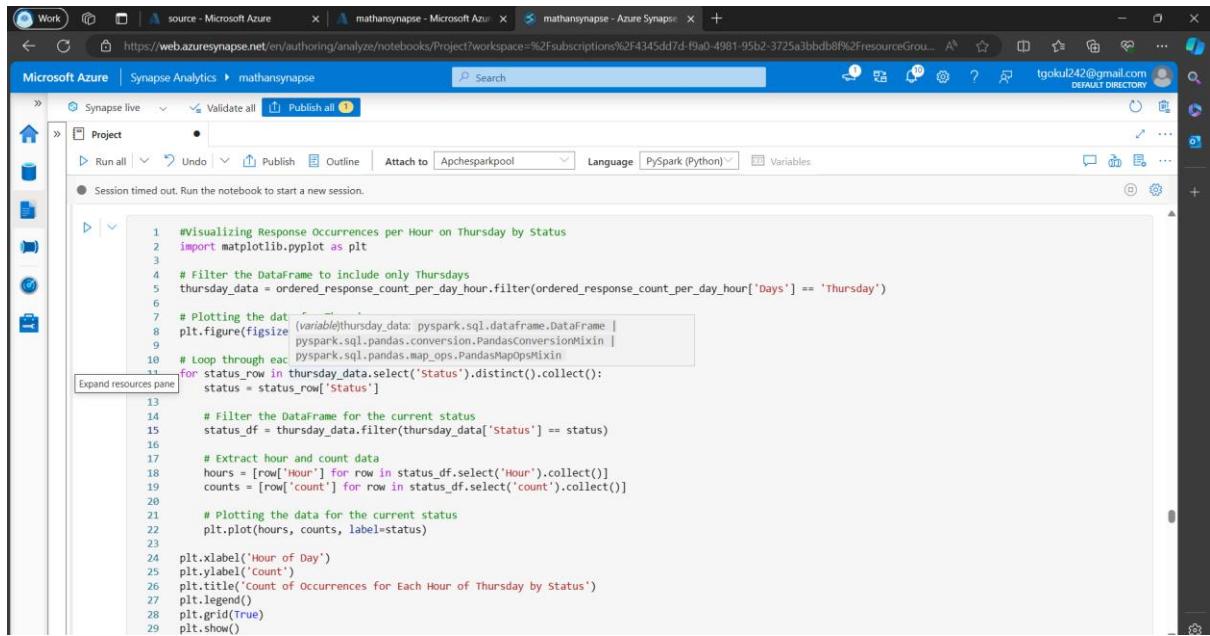


The screenshot shows a Microsoft Azure Synapse Analytics notebook interface. The top navigation bar includes tabs for 'source - Microsoft Azure', 'mathansynapse - Microsoft Azure', and 'mathansynapse - Azure Synapse'. The main area displays a Python script titled '#Visualizing Response Occurrences per Hour on Wednesday by Status'. The code uses pandas and matplotlib libraries to filter data for Wednesdays, extract hours and counts, and plot the results. The notebook interface has a sidebar with various icons and a status message at the bottom: 'Session timed out. Run the notebook to start a new session.'

```
1 #Visualizing Response Occurrences per Hour on Wednesday by Status
2 import matplotlib.pyplot as plt
3
4 # Filter the DataFrame to include only Wednesdays
5 wednesday_data = ordered_response_count_per_day_hour.filter(ordered_response_count_per_day_hour['Days'] == 'Wednesday')
6
7 # Plotting the data for Wednesdays
8 plt.figure(figsize=(12, 8))
9
10 # Loop through each unique status
11 for status_row in wednesday_data.select('Status').distinct().collect():
12     status = status_row['Status']
13
14     # Filter the DataFrame for the current status
15     status_df = wednesday_data.filter(wednesday_data['Status'] == status)
16
17     # Extract hour and count data
18     hours = [row['Hour'] for row in status_df.select('Hour').collect()]
19     counts = [row['Count'] for row in status_df.select('Count').collect()]
20
21     # Plotting the data for the current status
22     plt.plot(hours, counts, label=status)
23
24 plt.xlabel('Hour of Day')
25 plt.ylabel('Count')
26 plt.title('Count of Occurrences for Each Hour of Wednesday by Status')
27 plt.legend()
28 plt.grid(True)
29 plt.show()
```

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.

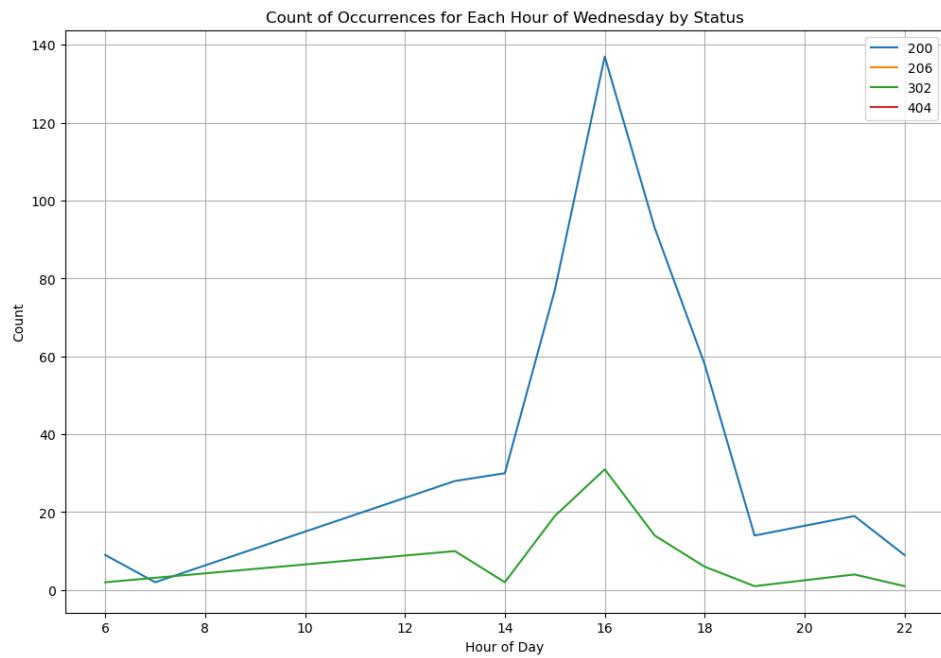
## Visualizing Hourly Response Occurrences on Thursday by Status

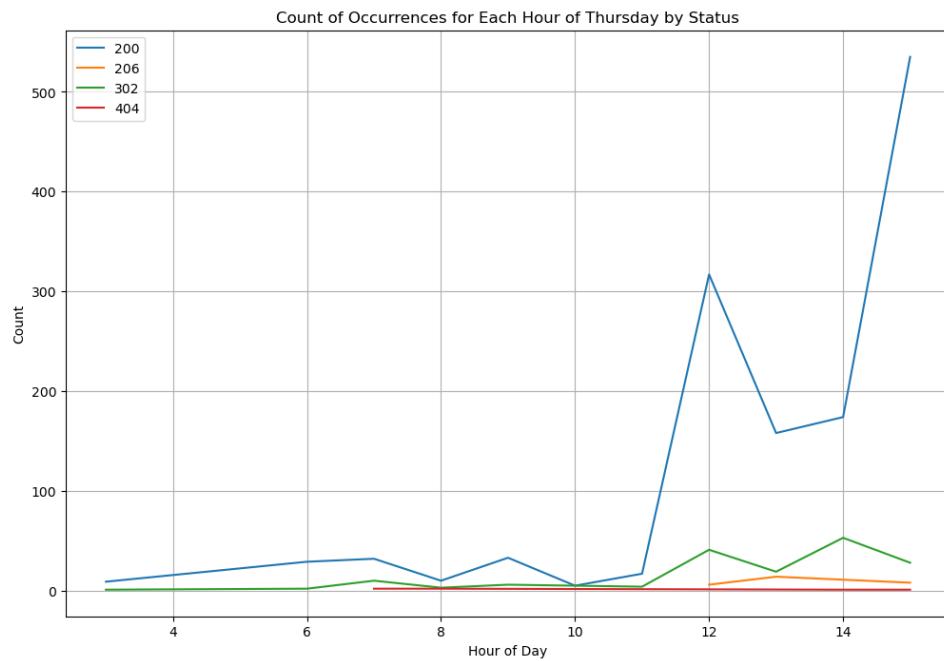


The screenshot shows a Microsoft Azure Synapse Analytics notebook interface. The code in the notebook is as follows:

```
1 #Visualizing Response Occurrences per Hour on Thursday by Status
2 import matplotlib.pyplot as plt
3
4 # Filter the DataFrame to include only Thursdays
5 thursday_data = ordered_response_count_per_day_hour.filter(ordered_response_count_per_day_hour['Days'] == 'Thursday')
6
7 # Plotting the data
8 plt.figure(figsize=(variable)thursday_data: pyspark.sql.dataframe.DataFrame |
9             pyspark.sql.pandas.conversion.PandasConversionMixin |
10             pyspark.sql.pandas.map_ops.PandasMapOpsMixin
11 # Loop through each status
12 for status_row in thursday_data.select('status').distinct().collect():
13     status = status_row['status']
14
15     # Filter the DataFrame for the current status
16     status_df = thursday_data.filter(thursday_data['status'] == status)
17
18     # Extract hour and count data
19     hours = [row['Hour'] for row in status_df.select('Hour').collect()]
20     counts = [row['count'] for row in status_df.select('count').collect()]
21
22     # Plotting the data for the current status
23     plt.plot(hours, counts, label=status)
24
25 plt.xlabel('Hour of Day')
26 plt.ylabel('Count')
27 plt.title('Count of Occurrences for Each Hour of Thursday by Status')
28 plt.legend()
29 plt.grid(True)
30 plt.show()
```

## Visualization





## Analysing HTTP Method Occurrences

```

1 #Counting Occurrences of Each HTTP Method in DataFrame
2 from pyspark.sql.functions import count
3 method_count = df1.groupby("Method").agg(count("*").alias("Method_count"))
4 method_count.show()

```

[44] 1 sec - Command executed in 1 sec 806 ms by tgokul242 on 9:28:33 AM, 3/08/24

> Job execution Succeeded Spark 2 executors 8 cores

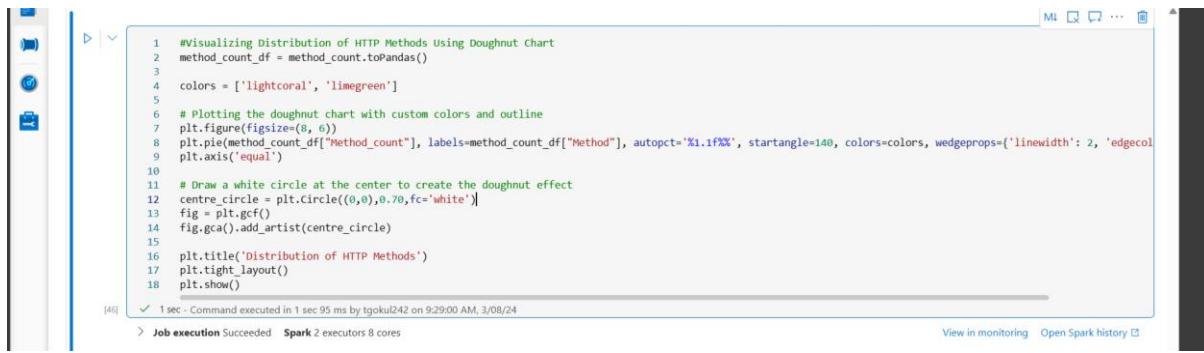
```

+-----+-----+
|Method|Method_count|
+-----+-----+
| POST |      137 |
| GET  |     1952 |
+-----+-----+

```

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.

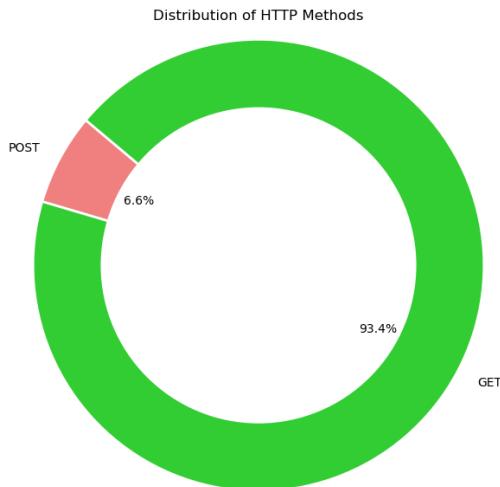
## Visualizing HTTP Method Distribution



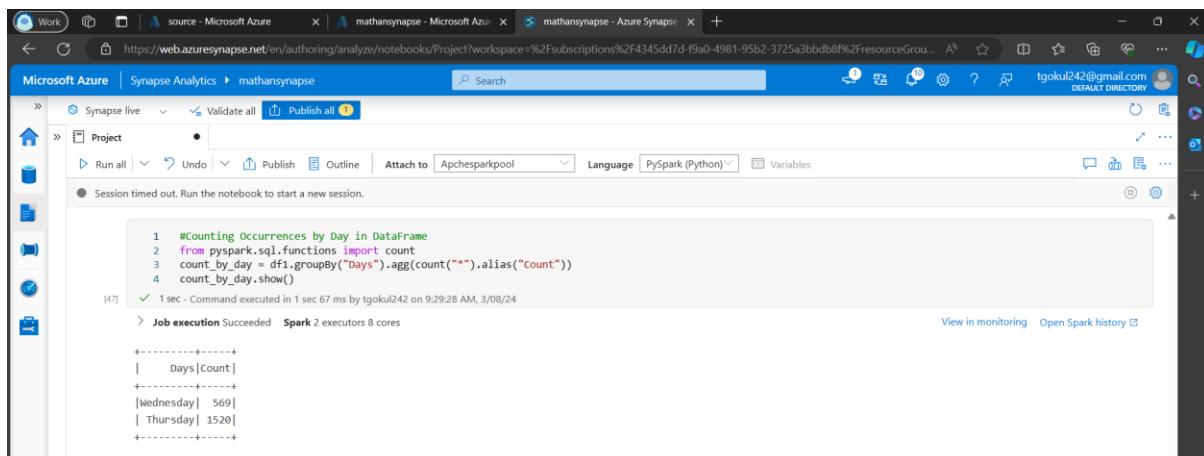
```
1 #Visualizing Distribution of HTTP Methods Using Doughnut Chart
2 method_count_df = method_count.toPandas()
3
4 colors = ['lightcoral', 'limegreen']
5
6 #Plotting the doughnut chart with custom colors and outline
7 plt.figure(figsize=(8, 6))
8 plt.pie(method_count_df["Method"], labels=method_count_df["Method"], autopct='%1.1f%%', startangle=140, colors=colors, wedgeprops={'linewidth': 2, 'edgecolor': 'black'})
9 plt.axis('equal')
10
11 # Draw a white circle at the center to create the doughnut effect
12 centre_circle = plt.Circle((0,0),0.7,fc='white')
13 fig = plt.gcf()
14 fig.gca().add_artist(centre_circle)
15
16 plt.title('Distribution of HTTP Methods')
17 plt.tight_layout()
18 plt.show()
```

[46] ✓ 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



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

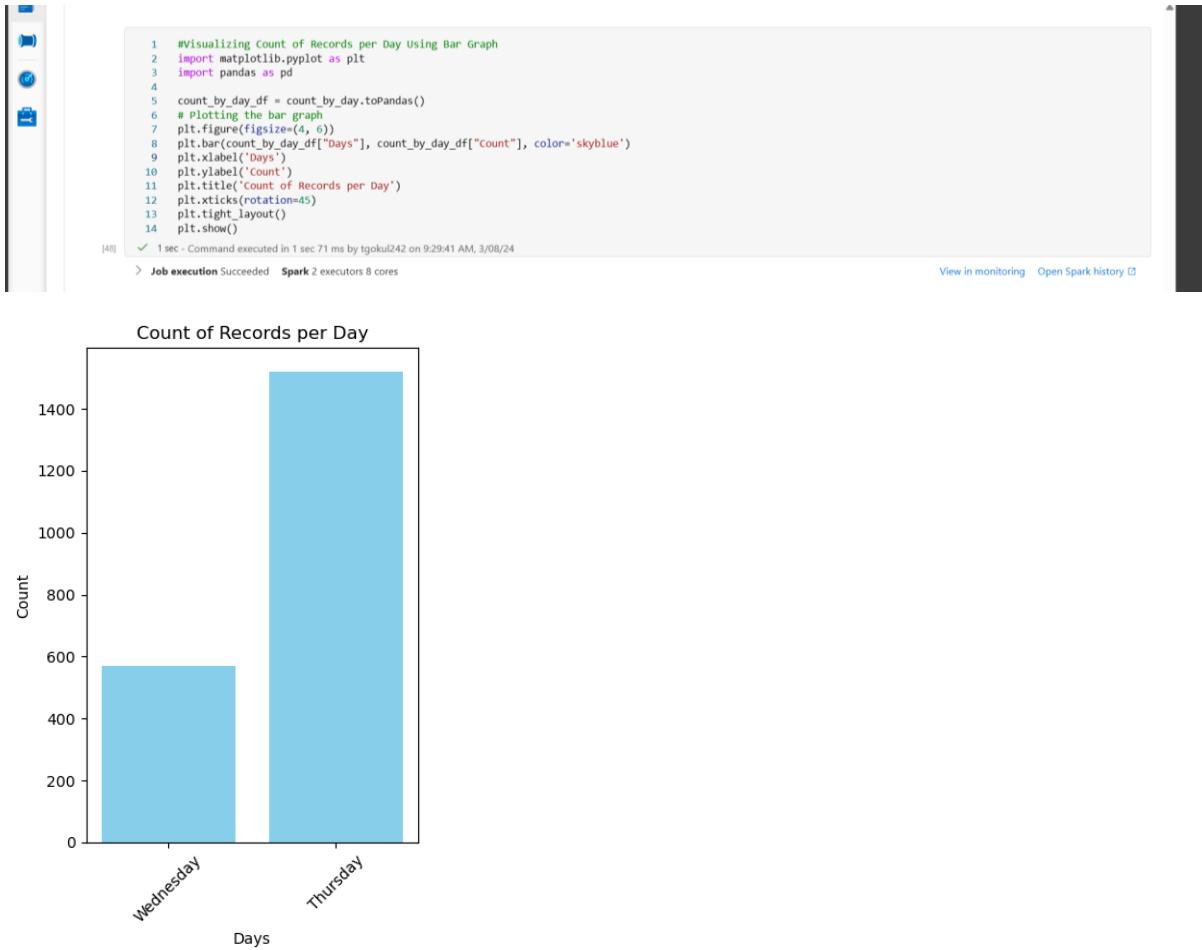
[47] ✓ 1 sec - Command executed in 1 sec 67 ms by tgokul242 on 9:29:28 AM, 3/08/24  
Job execution Succeeded Spark 2 executors 8 cores

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



## Analyzing Web Browser Usage



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.

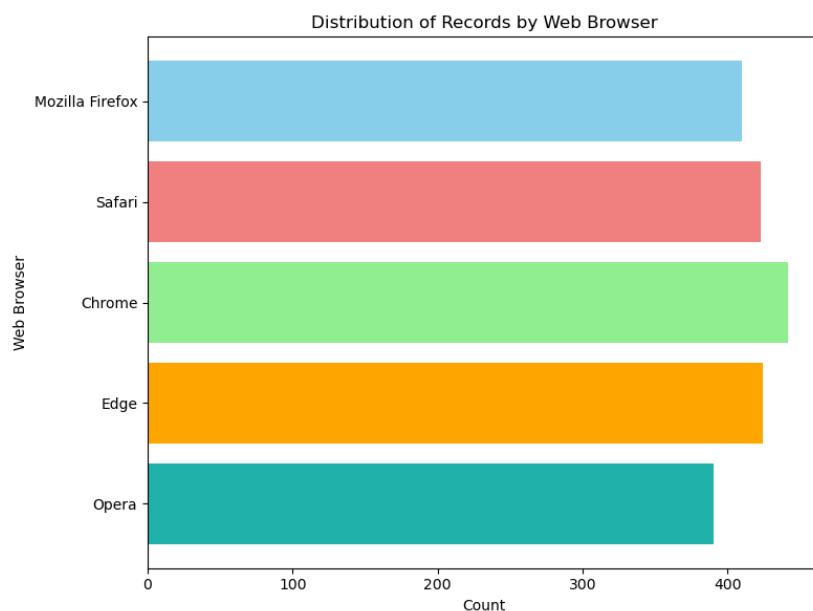
## Visualizing Web Browser Records

```
1 #visualizing Distribution of Records by Web Browser Using Horizontal Bar Plot
2 import matplotlib.pyplot as plt
3 count_by_browser_df = count_by_browser.toPandas()
4 colors = ['skyblue', 'lightcoral', 'lightgreen', 'orange', 'lightseagreen']
5
6 # Plotting the horizontal bar plot with custom colors
7 plt.figure(figsize=(8, 6))
8 plt.barh(count_by_browser_df["Web browser"], count_by_browser_df["Count"], color=colors)
9 plt.xlabel('Count')
10 plt.ylabel('Web Browser')
11 plt.title('Distribution of Records by Web Browser')
12 plt.gca().invert_yaxis() # Invert y-axis to have the highest count at the top
13 plt.tight_layout()
14 plt.show()
15
```

[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.